Proceedings of the Workshop on Change of Representation and Problem Reformulation

Michael R. Lowry
Recom Technologies, Inc.
AI Research Branch, Mail Stop 269-2
NASA Ames Research Center
Moffett Field, CA 94025, USA

NASA Ames Research Center

Artificial Intelligence Research Branch

Technical Report FIA-92-06

April, 1992

Preface

From the earliest days of AI, scientists have recognized that the representation of a task domain is a major factor in the success of symbolic processing systems. However, current AI systems are usually limited to the representations that are hardwired by their programmers. To develop truly intelligent systems, AI needs to endow its creations with the ability to select and even generate their own representations and problem formulations. This ability distinguishes human culture from animal learning: the plethora of languages, formalisms, and notations we invent to help us solve problems and enrich our lives.

The pioneering work of Saul Amarel demonstrated that, in principle, it is possible for an automated system to change its representation of a task domain; and furthermore that the payoff in problem solving performance could be exponential. Studies of expert problem solving behavior confirm the importance of the ability to choose good models and representations. Since the mid eighties, research has greatly expanded in this field of AI. Various approaches have been developed, including automatic theory revision; automatic abstraction, approximation and refinement of task domains; and transfer of knowledge from one representational formalism to another. Prototype systems have been developed in areas such as automatic planning and programming, engineering design, analytical reasoning problems and qualitative physics.

These are the proceedings for the third bi-annual workshop on change of representation and problem reformulation. The first workshop was chaired by Paul Benjamin in June of 1988 and sponsored by Philips Laboratories at Tarrytown, New York. Paul Benjamin edited a collection of revised papers from the workshop and published the book Change of Representation and Inductive Bias through Kluwer Academic in 1989. The second workshop was chaired by Jeffrey Van Baalen in March of 1990 at Menlo Park, California. Facilities were provided by Price Waterhouse Technology Center, with grant money provided by ACM SIGART. The third workshop was held at the Asilomar conference center in Pacific Grove, California in late April of 1992. Grant money was provided by AAAI and ACM SIGART.

In contrast to the first two workshops, this workshop was focused on analytic or knowledge-based approaches, as opposed to statistical or empirical approaches called 'constructive induction'. The organizing committee believes that there is a potential for combining analytic and inductive approaches at a future date. However, it became apparent at the previous two workshops that the communities pursuing these different approaches are currently interested in largely non-overlapping issues. The constructive induction community has been holding its own workshops, principally in conjunction with the machine learning conference. While this workshop is more focused on analytic approaches, the organizing committee has made an effort to include more application domains. We have greatly expanded from the origins in the machine learning community. Participants in this workshop come from the full spectrum of AI application domains including planning, qualitative physics, software engineering, knowledge representation, and machine learning.

Organizing Committee

Dr. Michael R. Lowry (Chair)
AI Research Branch
NASA Ames Research Center

Professor Jeffrey Van Baalen
Computer Science Dept.
University of Wyoming

Professor Robert C. Holte
Computer Science Department
University of Ottawa

Dr. Mark Shirley
Xerox Palo Alto Research Center

Professor Christopher H. Tong
Computer Science Dept.
Rutgers University

Workshop Participants

Saul Amarel
Laboratory for Computer Science Research
Hill Center for Mathematical Sciences
Rutgers University
New Brunswick, NJ 08903
amarel@cs.rutgers.edu

John Anderson
Dept of Comp. and Info. Science
University of Oregon
Eugene, OR 97403
john@cs.uoregon.edu

Jacky Baltes
2212 Victoria Crescent NW
Calgary, Alberta
CANADA T2M 4E3
baltes@cpsc.ucalgary.ca

Ranan Banerji
Dept of Math. and Comp. Science
St. Joseph's University
5600 City Avenue
Philadelphia, PA 19131
rbanerji@sju.edu

Tony Barrett
Dept of Comp. Science FR-35
University of Washington
Seattle, WA 98195
barrett@cs.washington.edu

Paul Benjamin
Department of Mathematics and Computer Science
St. Joseph's University
5600 City Avenue
Philadelphia, PA 19131
pbenjami@sju.edu

Kevin Benner
University of Southern California
Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292-6695
Benner@isi.edu

Giuseppe Cerbone
Giuseppe Cerbone
c/o Oregon State University
Computer Science Department
Corvallis, OR 97331
cerbone@cs.orst.edu

Thomas Ellman
Lab for Computer Science Research
Hill Center for Math. Sciences
Rutgers University
New Brunswick, NJ 08903
ellman@cs.rutgers.edu

Fausto Giunchiglia
fausto@sail.stanford.edu

Diana Gordon
Naval Research Laboratory
Code 5510
4555 Overlook Avenue, S.W.
Washington, D.C. 20375-5000
gordon@aic.nrl.navy.mil

Benjamin Grosof
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
grosof@watson.ibm.com

Robert Helm
Dept of Comp. and Info. Science
University of Oregon
Eugene, OR 97403
bhelm@cs.uoregon.edu

Robert Holte
Computer Science Department
University of Ottawa Ottawa
Ontario CANADA K1N 6N5
holte@csi.UOttawa.CA

Neil Iscoe
EDS Research, Austin Lab
1601 Rio Grande, #451
Austin, Texas 78701
iscoe@austin.eds.com

Rich Keller
AI Research Branch
NASA Ames Research Center
Mail Stop 269-2
Moffett Field, CA 94035-1000
Keller@ptolemy.arc.nasa.gov

Craig Knoblock
University of Southern California
Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292-6695
knoblock@isi.edu

Amy Lansky
AI Research Branch
NASA Ames Research Center
Mail Stop 269-2
Moffett Field, CA 94035-1000
lansky@ptolemy.arc.nasa.gov

Alon Levy
Knowledge Systems Laboratory
Department of Computer Science
Stanford University
701 Welch Road, Bldg C.
Palo Alto, California, 94304
alevy@cs.stanford.edu

Theodore Linden
2245 Tasso Street
Palo Alto, CA 94301
linden@ads.com

Zheng-Yang Liu
EDS Research, Austin Lab
1601 Rio Grande, #451
Austin, Texas 78701
liu@austin.eds.com

Michael Lowry
AI Research Branch
NASA Ames Research Center
Mail Stop 269-2
Moffett Field, CA 94035-1000
lowry@ptolemy.arc.nasa.gov

Steve Minton
AI Research Branch
NASA Ames Research Center
Mail Stop 269-2
Moffett Field, CA 94035-1000
Minton@ptolemy.arc.nasa.gov

Armand Prieditis
prieditis@cs.ucdavis.edu

Spiridon Reveliotis
785 Broadway #1,
Somerville MA 02144
spyros@nueng.coe.northeastern.edu

Josh Tenenberg
Dept of Computer Science
University of Rochester
Rochester , NY 14627-0226
josh@cs.rochester.edu

Chris Tong
Lab for Computer Science Research
Hill Center for Math. Sciences
Rutgers University
New Brunswick, NJ 08903
ctong@cs.rutgers.edu

Jeffrey Van Baalen
Computer Science Department
P.O. Box 3682
University of Wyoming
Laramie, WY 82071
jvb@moran.UWyo.Edu

Kyriakos Zavoleas
Dept of Industrial Eng. & Inform. Sys.
330 Snell Engineering Center
Northeastern University
360 Huntington Avenue
Boston, MA 02115
kyriakos@nueng.coe.northe

Contents

An Integrated Planning Representation using Macros, Abstractions, and Cases

Jacky Baltes and Bruce MacDonald
2500 University Drive NW
Calgary, Alberta T2N 1N4, Canada
{baltes,bruce}@cpsc.ucalgary.ca

Abstract

Planning will be an essential part of future autonomous robots and integrated intelligent systems. This paper focuses on learning problem solving knowledge in planning systems. The system is based on a common representation for macros, abstractions, and cases. Therefore, it is able to exploit both classical and case–based techniques. The general operators in a successful plan derivation would be assessed for their potential usefulness, and some stored. The feasibility of this approach was studied through the implementation of a learning system for abstraction. New macros are motivated by trying to improve the operator–set. One heuristic used to improve the operator–set is generating operators with more general pre–conditions than existing ones. This heuristic leads naturally to abstraction hierarchies. This investigation showed promising results on the towers of Hanoi problem. The paper concludes by describing methods for learning other problem solving knowledge. This knowledge can be represented by allowing operators at different levels of abstraction in a refinement.

Introduction

This paper advocates a common representation for operators that includes abstract plans, cases and macros [Baltes, 1991]. An important aspect of this representation is that a system should be able to learn the necessary problem solving knowledge.

The implementation of a prototype system that learns abstraction hierarchies is described. The learning system tries to improve the operator–set by extracting macros with more general pre–conditions than existing ones. This leads naturally to the generation of abstraction hierarchies. Rather than searching for new macros explicitly, the learner extracts new macros from a successful plan. It tries to find operators that result in identical states and that differ in exactly one pre–condition predicate. If such operators are found, the system deletes the differing predicate from the pre–conditions, thus forming an abstract operator. Since the planning system is intended to support case–based planning techniques, a generalized and an instantiated version of the macro is stored. We intend to use a novel dynamic filtering scheme [Baltes, 1991] to delete poor macros.

The learning system was tested on the towers of Hanoi problem and showed promising results. The remainder of the paper is organized as follows: first, the paper presents a common representation for planners, then reviews previous work on operator learning with macros. Section explains how operators are learned in our representation. Then, a description of the implementation and an example are given. The paper concludes by describing how we intend to learn other problem solving knowledge such as reactive rules or anticipation of failure.

Macro–Operators

A linear macro is a sequence of primitive operators. This sequence is usually generalized and added to the operator set as a new operator. For example, useful macros in the blocks world domain are `pickup=(goto,grasp)` and `putdown=(goto,ungrasp)`. Macros can be used in the construction of new macros, for example `move=(pickup,putdown)`.

This paper focuses on linear macros because the formation of iterative or disjunctive macros depends on good linear ones [Shell and Carbonnel, 1989]. Macros speed up the planning process because they reduce the solution length. On the other hand, the generation of macros must be carefully controlled because new operators increase the branching factor of the search space. Minton showed that simply generating all possible macros from a successful solution decreases performance [Minton, 1985].

Dynamic Filters

As mentioned above, only a small number should be generated, ideally ones that will be useful in future problem solving tasks. The basic problem of macro learning is that the system has to predict the useful-

ness of a macro based on its previous experience. The following paragraphs describe the effect of adding one macro to the operator set and derive a *usefulness* measure for such an addition. This measure can be used to dynamically delete macros.

By adding a macro m, the branching factor is increased. However, the new macro will not be applicable in all situations. Let b be the branching factor without the macro in question. Let c be the fraction of states where m is applicable. Furthermore, not all applications of m will lead to a solution, so let u_m be the *usefulness* of m, which is the overall chance of applying m to achieve a solution; the ratio of the total number of times m leads to a solution, to the total number of times any operator is applicable. If l_m is the number of primitive operators in m, then the time complexity for the new operator set is of the order:

$$(b+c)^{n/l_m u_m}$$

The branching factor is increased by the applicability of m, and the effective solution length is reduced by the chances of using m, and in proportion to m's length. If u_m is 1, this means m is used at each step of the solution, and the plan is divided in length by the number of operators in m. If planning is to be faster when a macro is added, then the following inequality must be satisfied:

$$b^n \geq (b+c)^{n/l_m u_m} \quad (1)$$

$$\log b \geq \frac{1}{l_m u_m} \log(b+c) \quad (2)$$

$$u_m \geq \frac{1}{l_m} \cdot \frac{\log(b+c)}{\log b} \quad (3)$$

The macro length predominates the generality of its pre–conditions, c, in 3. So it will be more important to allow long macros than more specific macro pre–conditions. However, the pre–conditions cannot be ignored; impractically large values are required for b to make the second fraction in 3 approach unity. Note that u_m and l_m are not independent; as the length grows the chances of the macro being used in a plan decrease. Furthermore, it is also assumed that the space searched does not change with m. Under this assumption, b and c are independent.

Equation 3 cannot be used directly for selecting new macros because u_m, b, and c cannot be effectively computed *a priori*. However, these values can be approximated statistically. After a number of trials, equation 3 can be used as a dynamic filter to remove unnecessary macros.

Iba [Iba, 1989] proposes dynamic filters in his MACLEARN system. However, the implementation seems ad hoc; the user determines when to call the dynamic filter routine, which deletes all macros that have not been used at least once in a previous problem solving episode.

The utility measure is not based on the number of pre–conditions in a macro–operator (as it is done in Minton's system), since as will be explained in the remainder of the paper, the number of pre–conditions does not increase in my system.

Macros, Abstractions, and Cases

This section suggests a common representation for macros, abstractions, and cases in a planning system. It will show the similarity and differences between these methods, and suggest that a common representation will allow a problem solver to use all three strategies simultaneously.

The proposed representation will enable a planning system to maintain important advantages of previous systems:

- The planner will learn only when there is strong motivation, in order to increase performance in the future. This point has been shown by the MACLEARN system (flatten the search space, [Iba, 1989]) and by the CHEF system (repair failed plans and anticipate problems, [Hammond, 1989]).
- Proposed macros are filtered statically as well as dynamically. A new heuristic described in equation 3 is used.
- The planner learns from a worked example (similar to MACLEARN, PiL2 [Yamada and Tsuji, 1989], CHEF) rather than using a brute force search to find new operators (which would be similar to MPS ,[Korf, 1985].
- The planner should be able to use a heuristic function or other knowledge that is available.

Korf mentioned the similarity between abstractions and macros [Korf, 1987]. Both methods try to reduce the search by generating a skeleton search space of the original problem space. Instead of searching in the original space, a solution is found in the skeleton space and this solution is then refined into a solution in the original problem space. One difference, however, is that there can be more than one abstraction level whereas macros normally generate only one skeleton space.

Cases can be viewed as long, specific macros. The main distinction between macros and cases is the way in which they are used in a planning system. Cases are fetched from memory and some plan critics are applied to change the case to the new situation. Macros are usually not altered, i.e. the sequence of elementary operators is not adapted to the new situation.

The common feature among all three items is that the most important information stored is a set of pre–conditions and a set of effects, as for elementary operators.

Abstraction hierarchies are equivalent to elementary operators that are missing some pre–condition predicates. This means that although the specific execution depends on all pre–conditions, the effects can be achieved independently of the actual value of the

deleted predicates in the pre–conditions. One abstract operator can be specialized in a number of different ways. The representation can capture this by associating a set of operators with pre–conditions and effects. This structure represents that the effects can be achieved given that the pre–conditions are true, but that the instantiation of the plan may depend on predicates not mentioned in the pre–conditions. A method similar to PiL2's perfect causality heuristic is used to generate new operators that depend on fewer pre–conditions. More than one level of abstraction can be represented by showing that elements of the refinement of an abstract operator can consist of abstract operators.

Common representation

In our common representation, shown in Figure 1, an operator is recursively represented as (a) a pre–conditions and effects pair, and (b) a set of refinements, each of which is an operator sequence. A primitive operator has no refinements, and can be executed.

A variety of well–known problem solving knowledge is supported. An operator is like a macro if (a) there is only one refinement, (b) each operator in the refinement is a primitive, and (c) pre–conditions and effects predicate arguments are instantiated in neither the operator nor the refinement (i.e. the macro has formal parameters). A case is an operator with a refinement that is a fully instantiated (long) sequences of primitives (i.e. an instantiated macro). An abstract operator at criticality level k has refinement/s whose operators are abstract ones at level $k-1$. This representation supports *relaxed* (predicates deleted from pre–conditions, ABTWEAK) as well as *reduced* (predicates deleted in pre–conditions and effects, ALPINE) models of abstraction [Knoblock, 1991].

An operator is a subgoal sequence if all refinements contain no primitive operators (e.g. means ends analysis). Anticipation of failure can be represented by an operator whose single refinement is a single operator pre–conditions, effects pair in which there is an additional effect (such as "avoid soggy broccoli"). This ensures that the planner knows about the problem, and the refinement of the failure anticipation operator will be expanded using the successful plan, which is also stored as an operator. Since our representation does not enforce a common level of abstraction for operators in the refinement, a case or macro can also be generalized by making some operators non–primitives. This allows us to store adaptations of a case such as the replacement of some steps. Reactive rules may be represented as an operator whose single, two–operator refinement is a fully instantiated, primitive first operator, followed by a non–primitive pre–conditions, effects pair. If this two–operator refinement is reversed, then the resulting operator is suitable for backward chaining from the goal (similar to RWM [Güvenir and Ernst, 1990]).

While this generality provides a common operator representation, it also presents the immediate problem of controlling the creation of operators, so that planning is not impossibly expensive. We intend to control this using the dynamic operator deletion mechanism introduced above. In addition, the learning methods that add operators to the case memory must do so only when there is strong justification, and must choose "important" parts of new plans for storage, deciding the level of abstraction, number of refinements, and so on. This is the subject of current work. The common representation enables us to treat the various kinds of planning system in a single consistent framework, to better aid analysis and comparison.

The planner may choose to "forget" the refinements of some operators, when their usefulness decreases. But the pre–conditions, effects pair is retained, and the details can be replanned if necessary.

Planning using a common Representation

This section indicates how one might use the operator representation given in this paper. The planner should combine case–based as well as classical planning techniques, to take advantage of both previous experience, and the ability to solve new problems. What is needed is a control strategy that recalls and uses previous experiences to solve similar new problems, but gracefully moves into classical planning if no similar cases can be found. The recursive structure of the representation lends itself well to a recursive control strategy. Learning is designed to support and improve the planning process, by storing new operators. The planner should restrict the branching factor of the search space by focusing on a small number of operators instead of all applicable ones.

The input to the planner are initial state, goal state, and the operator set. Additional input is a resource limit and a skeleton plan agenda, which may support resource limited and multiple task planning. The planning begins by matching and recalling stored operators that have pre–conditions and effects similar to the current state and goal. The refinement/s of these operator/s will give various types of plans to be considered for solving the current problem.

Recalling similar operators A stored, similar case may have additional pre–conditions or effects, or be missing some. Operators should be recalled when their pre–conditions and/or effects are similar to the current goal and initial state. Possible indexing schemes for recall can be based on the number of predicates in pre–conditions and effects, the predicates themselves, or combinations of predicates.

Recalling is independent of the learned operator hierarchy; the fetched operator is not necessarily at the top level of a refinement tree. For example, if there is an abstract operator to move a medium disk independently of the small disk, and one refinement of this is

KEY: [Pre, Post] indicates a non–primitive operator, < Pre, Post > a primitive one, and <[Pre, Post]> either. $<[Pre, Post]>^k$ is an operator with predicates at criticality level k or above.

General Operator:

$$[Pre, Post] \longrightarrow \begin{array}{l} <[Pre_{11}, Post_{11}]>, <[Pre_{12}, Post_{12}]> \ldots <[Pre_{1n_1}, Post_{1n_1}]> \\ <[Pre_{21}, Post_{21}]>, <[Pre_{22}, Post_{22}]> \ldots <[Pre_{2n_2}, Post_{2n_2}]> \\ \vdots \\ <[Pre_{m1}, Post_{m1}]>, <[Pre_{m2}, Post_{m2}]> \ldots <[Pre_{mn_m}, Post_{mn_m}]> \end{array}$$

Each $Pre_{i1} \Rightarrow Pre$ and each $Post_{in_i} \Rightarrow Post$.

Macro (uninstantiated arguments) or Case (instantiated arguments):

$$[Pre, Post] \longrightarrow < Pre, Post_1 >< Pre_2, Post_2 > \ldots < Pre_n, Post >$$

Abstract operator:

$$[Pre, Post]^k \longrightarrow \begin{array}{l} <[Pre_{11}, Post_{11}]>^{k-1}, <[Pre_{12}, Post_{12}]>^{k-1} \ldots <[Pre_{1n_1}, Post_{1n_1}]>^{k-1} \\ <[Pre_{21}, Post_{21}]>^{k-1}, <[Pre_{22}, Post_{22}]>^{k-1} \ldots <[Pre_{2n_2}, Post_{2n_2}]>^{k-1} \\ \vdots \\ <[Pre_{m1}, Post_{m1}]>^{k-1}, <[Pre_{m2}, Post_{m2}]>^{k-1} \ldots <[Pre_{mn_m}, Post_{mn_m}]>^{k-1} \end{array}$$

Automatic subgoaling: $[Pre, Post] \longrightarrow [Pre, Post_1][Pre_2, Post_2] \ldots [Pre_n, Post]$

Anticipation of failure: $[Pre, Post] \longrightarrow [Pre, Post_1]$

Reactive rules: $[Pre, Post] \longrightarrow < Pre, Post_1 >[Pre_2, Post]$

RWM–type operator subgoal: $[Pre, Post] \longrightarrow [Pre, Post_1]< Pre_2, Post >$

Figure 1: The representation of planning operators.

to move the medium disk when both are on the first peg, and if the current state is that both disks are on that peg, that refinement is retrieved, rather than a more abstract one.

Adapting an existing plan Adaptation of a plan to new initial states and goal states is done by analysing the differences among the initial state and the pre–conditions and among the goal state and the effects of the similar plan. There are a number of general purpose adaptations to a plan that substitute one operator, listed below. If these don't provide a complete plan, then we treat the adapted plan as a subplan and use means ends analysis to complete it.

Replace Steps: An operator should be removed and steps inserted to achieve either pre–conditions or effects.

- Remove Side effect: If a plan fails because one operator has a specific side effect try to replace this operator with one that works.
- Protect effect: A following operator destroys an effect of the solution. Try to replace this operator with one that does not change the side effect.

Substitution: Replace a variable instantiation with a different object (the operator sequence is unchanged).

To find where to replace an operator, pre–conditions of the case that are not given in the current situation are pushed forward up to the first operator depending on those pre–conditions. Then the planner finds all elements of the effects that are dependent on this operator. If the effects are also part of the current goal, the system generates a new planning problem from the state just before the operator with the non–matching pre–condition to the first operator that uses any of the effects established. For example, if the goal is to have a barbecue and one of the pre–conditions is to have a match, this pre–condition is pushed forward to the operator **make-fire**. Since fire is a prerequisite for a barbecue, this effects is pushed backward to the operator **put steaks on fire** which has **has-fire** as a pre–condition. The system then tries to "improvise" and generate a plan using the state just before the operator **make-fire** to operator **put steaks on fire**. Given that we have a lighter in the current state, this plan can be easily generated. The **light match** operator is replaced by the **use lighter** operator. "Remove side effects" and "Protect effect" are specializations of the Replace operator strategy.

If the non–matching pre–condition does not establish a current goal predicate, the system tries to apply all operators of the plan, substituting variables where necessary (e.g. beans for broccoli). For example, given a plan to make a beef and bean dish from the ingredients, and if the system returns a plan for a beef and broccoli dish, the pre–condition have broccoli does not establish any predicate in the current goal (beef and

beans dish). In this case, the system simulates the plan and uses beans instead of broccoli.

After the case has been fixed to achieve all its effects with the new initial conditions, the system tries to achieve missing goal conditions one by one, using means ends analysis. The first non–satisfied term of the goal conjunction is selected and a new planning problem is generated from the goal state of the case to the goal state of the original problem.

The planner computes the subgoals that are necessary for the achievement of any of the adaptations or classical planning rules. It then retrieves similar cases for each of the generated subgoals and tries to work on them in order of similarity. This can also be used to repair failed plans, if the failed plan is stored in memory with a new effects added so that the failure is avoided.

Learning General Operators

Many researches have investigated different methods for constructing macros [Korf, 1985; Korf, 1987; Minton, 1985; Iba, 1989; Yamada and Tsuji, 1989]. A comparison of those methods leads to the following issues:

Generalized macros Korf's MPS system [Korf, 1985] stores instantiated macros, whereas Yamada's PiL2 system [Yamada and Tsuji, 1989] and Iba's MAC–LEARN system [Iba, 1989] generalize macros, so that they are more widely applicable. Although generalization of macros seems intuitive, it also increases the search space, especially if many objects exist in the domain.

Worked examples The MPS system searches for macros to fill the table. In the worst case, this may prevent the algorithm from terminating, although a solution to the problem may exist. This can occur if MPS is trying to find an impossible macro. PiL2 and MAC–LEARN use a "worked example" to extract macros. A " worked example" is either a successful plan or part of the search space that was searched when trying to find a successful plan. This means that no extra search effort is required for the generation of macros.

Motivation The motivation behind the MPS system is to combine automatic subgoaling with macros. Macros are used to *serialize* a subgoal sequence by guaranteeing that the goal conditions of previous subgoals are satisfied after the application of the macro, although they may be temporarily destroyed during its application. The heuristic used in the MAC–LEARN system is to generate macros between peaks in the heuristic evaluation function. This means that macros are used to flatten the search space of the heuristic evaluation function so that valleys can be traversed faster. The PiL2 system uses the *perfect causality* heuristic. It extracts a macro from a successful plan if (a) the pre–conditions of an operator in the plan were not satisfied in the initial state, and (b) the pre–conditions of this operator were satisfied after the application of previous operators. The motivation is to generate macros that allow the system to apply more operators to the initial state.

Learning Abstraction Hierarchies with Macros

Although a common representation is powerful, the manual generation of useful operators requires a large amount of domain knowledge and is tedious. Ideally, the planning system should learn operators from its previous experience. Therefore, a learning system was designed to create new operators for the representation. There are two major motivations for the system to learn:

Failure The system generates a plan that failed. Here, it tries to avoid generation of invalid plans in the future. Examples are anticipation of failure in case–based planning systems or explanation based learning rules.

Success Given that the system generated a successful plan, extract information from this plan to speed up the process for similar goals in the future. The generation of macro–operators and automatic subgoaling fall into this category.

The "need to learn" is easily recognized in the failure driven approach. The system knows exactly when new information has to be added, that is exactly when a generated plan failed. The problem is to decide what information should be stored in order to avoid failure in the future. For example, should the fully instantiated problem be stored or a generalization of it.

Learning in the success driven approach is harder, because the system must decide when to integrate new knowledge as well as what knowledge to integrate. For example, the MAC–LEARN system motivates learning by trying to flatten the search space. In PiL2, a sequence of operators that are used only to generate pre–conditions of a following operator should be combined in one macro so that the operator can be applied to the initial state.

This paper proposes two new heuristics for the generation of macros. The motivation is that to improve performance, a macro learner must improve the operator set. A macro–learner only changes the operator set, it does not tell the planner when to apply new operators. For example, it does not affect the heuristic evaluation function. Previous systems such as MAC–LEARN and PiL2, however, do not take the current operator set into consideration when learning new macros. There are two ways in which an operator can be improved.

Heuristic 1 Try to create new abstract operators that contain fewer pre–conditions than existing operators, and identical effects. That means that certain conditions can more easily be generated.

Heuristic 2 Try to introduce operators that have fewer effects than existing ones. This generates operators with more specific effects, so that the planning system can affect the world more controlled.

Heuristic 1 is more interesting because it generates an abstraction hierarchy of operators. This paper describes the implementation of a macro–learner that uses only the first heuristic to find new macros.

Implementation of the Macro–Learner

The learning system described in this paper is an addition to the AbTweak planning system implemented by Yang [Yang and Tenenberg, 1990]. The macro–learner generates a successful plan using AbTweak and extracts macros from it.

Figure 2 is a pseudo code description of the algorithm used. Explanation based generalization (EBG) is a common technique for the generalization of a macro [Minton, 1985]. The problem is given a sequence of operators and variable instantiations to compute the weakest set of pre–conditions that still allow the achievement of its effects.

Post–Conditions The post–conditions of an operator are the set of facts that must be true after application of the operator. It is different from the effects of an operator because the effects only mention facts that are *changed* by the operator. However, the pre–conditions that are not affected must also be true after application of the operator. The post–conditions are equivalent to the effects plus all predicates in the pre–conditions that are unchanged.

Logically Equivalent Descriptions The major problem in the implementation of the system is that there is more than one possible logical description of the world. For example, in the towers of Hanoi problem, the states `(not ons Peg1)(not ons Peg2)` and `(ons Peg3)` are equivalent because there are only three possible pegs and each disk must always be on a peg. The macro–learner extracts macros that have the same post–conditions but can be used to generate abstract macros. This means that the algorithm must establish the logical equivalence of world states. There are two possible solutions to this problem.

The first method uses a resolution theorem prover to prove the equivalence of post–conditions. This method is the most general one. A set of axioms can be given that can be used to prove facts about the domain. Since the operator set describes all elementary actions by which the world can be affected, it is also possible to derive certain facts used in the proof. For example, since the only operator that moves the small disk establishes the fact that the small disk is on some peg, the system can derive the fact that the disk is always on some peg.

The second method uses a unique description of the world. Two states are identical if and only if they have the same description. This can be achieved by changing the representation of operators or by designing a set of domain dependent rewrite rules that change a description dynamically. For example, a rewrite rule can be used to convert `(not ons Peg1)(not ons Peg2)` to `(ons Peg3)`.

This projects focuses on the feasibility of learning abstract operators rather than the design of a practical planning system. Therefore, the standard description language of operators was changed to generate a unique description of all world states. For example, the standard definition of the operator to move the big disk in the towers of Hanoi problem is

```
move-big($X $Z)
    Pre: (onb $X) (not ons $X) (not ons $Z)
         (is-peg $X) (is-peg $Z)
   Post: (onb $Z) (not onb $X)
```

This definition was changed to allow unique descriptions of post–conditions. Therefore, the `move-big` operator was defined as follows:

```
move-big($X $Y $Z)
    Pre: (onb $X) (ons $Y)
         (is-peg $X) (is-peg $Y) (is-peg $Z)
   Post: (onb $Z) (not onb $X)
```

This means that all operators must reference all three pegs in their argument list, and that all facts are represented directly instead of indirectly.

Example

This section shows an example of the macro–learner in the towers of Hanoi domain with two disks. There are three reasons for selecting this problem.

- It was easy to find a representation of operators that resulted in a unique logical description of the world.
- The problem is well studied and comparison to other planners can be made. Also, the optimal solution for this problem is known.
- The structure of the problem is well suited to abstract operators. In fact, abstract operators can reduce its time complexity to be linear in the length of the solution.

In the initial state, both disks are on the first peg. The goal is to move both disks on the third peg. The standard operator set is changed to use a unique logical description and is represented by:

```
Macro-Learner(Plan, Operator-Set)
   Compute all possible macros in the plan.
   For each macro in the plan and each operator do
      macro-gen := EBG(macro,op)
      world := post-conditions(macro-gen)
      if world = post-conditions(op) then
         if pre(macro) and pre(op) differ in one predicate
             ab := create-abstract-operator(macro,op)
             link(ab,op)
             link(ab,macro)
         operator-set := add-operator(ab, operator-set)
   return(operator-set)
```

Figure 2: Macro–Learner Algorithm

```
move-s($X $Y $Z)
   Pre: (ispeg $X)(ispeg $Y)(ispeg $Z)
        (ons $X)
  Post: (not ons $X)(ons $Z)

move-b($X $Y $Z)
   Pre: (ispeg $X)(ispeg $Y)(ispeg $Z)
        (ons $Y)
        (onb $X)
  Post: (not onb $X)(onb $Z)
```

AbTweak is used to find a solution for this problem which is the sequence **move-s(Peg1,Peg2), move-b(Peg1,Peg3), move-s(Peg2,Peg3)**. From this solution three macro sequences can be extracted.

```
seq-1: move-s(Peg1,Peg2),move-b(Peg1,Peg3)
seq-2: move-b(Peg1,Peg3),move-s(Peg2,Peg3)
seq-3: move-s(Peg1,Peg2),move-b(Peg1,Peg3),
       move-s(Peg2,Peg3)
```

From the first sequence **seq-1**, the following macro can be generated after using EBG to compute its pre–conditions and effects. Operator **macro1** is not changed when generalizing with the original operator **move-b** because neither restricts the variable instantiations.

```
macro1($V1 $V2 $V3)
     Pre:(is-peg $V1) (is-peg $V2)
         (is-peg $V3)
         (ons $V1) (onb $V1)
    Post:(ons $V2) (onb $V3)
         (not ons $V1) (not onb $V1)
```

The post–conditions of **macro1** and **move-b** are identical (except renaming of variables) and are given by the following set of facts:

```
(is-peg $V1) (is-peg $V2)(is-peg $V3)
(ons $V2)(onb $V3)
```

The algorithm then compares the pre–conditions of **macro1** and **move-b**. The pre–conditions differ only in the predicate **ons** which is **(ons $V1)** for **macro1** and **(ons $V2)** for **move-b**. Therefore, the macro–learner constructs an abstract operator in which the **ons** predicate is deleted. This abstract operator is generalized, and it contains two refinements: **move-b** and **macro1**. However, in order to avoid unnecessary variable instantiations, the linked macro is fully instantiated. In that way, if the same problem has to be solved in the future, the variables do not need to be re–instantiated. However, the abstract operator shows the generalization that is possible. Figure 3 shows the resulting operator hierarchy.

Since only **macro1** and **move-b** have identical post–conditions, the abstract operator in figure 3 is the only new operator that is added to the operator set.

Evaluation

With the implementation of the macro–learner, we tried to establish the usefulness of our first heuristic (to improve the operator set by generating new operators with more general pre–conditions). It is interesting to note that this heuristic leads to an abstraction hierarchy for the two–disk towers of Hanoi problem that is identical to the one shown to be optimal by Knoblock [Knoblock, 1991]. If the planner uses a control strategy that supports abstractions, the time complexity grows only linearly with the length of the solution [Knoblock, 1991].

Good performance of the planning system with the new operator set was expected, because the optimal set of abstractions was generated. This was verified in a number of experiments where the solution time was reduced from 40 to 24 seconds on a Sparc 2 station. It took ten seconds to compute the abstract operators. Similar results were obtained for the problem with the initial state **(ons peg3)(onb peg1)** and the goal state **(onb peg3)(ons peg3)**.

The macro–learner was also tested on the towers of Hanoi puzzle with three disks. The results of these experiments were similar to the ones for the previous example. The macro learner created two abstract operators:

- move the medium disk ignoring the small disk.

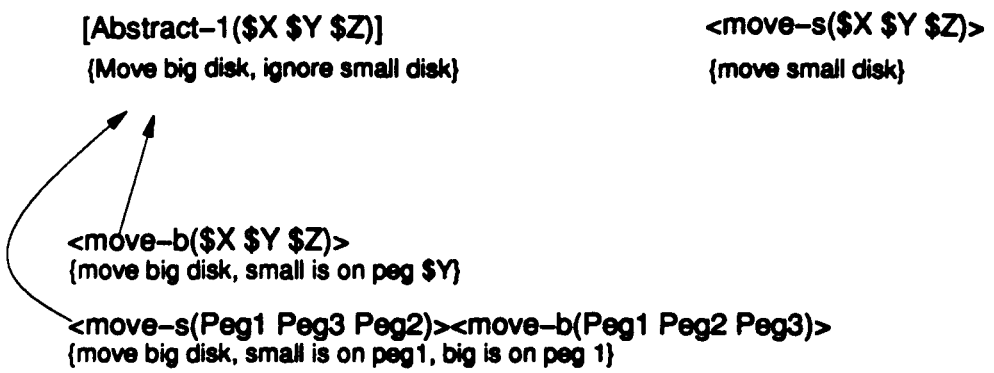


Figure 3: Learned Operator Hierarchy for the Towers of Hanoi 2

- move the large disk ignoring the medium disk.

These abstract operators are learned after only one successful plan is generated. After solving the problem a second time, the abstract operator learned to move the large disk regardless of where the medium and small disks are. These two abstract operators together with the primitive operator to move the small disk form the optimal operator set for the towers of hanoi problem with three disks. The solution time decreased from 2658 seconds to 29 seconds. The computation time for learning the abstract operators was 81 seconds. This result suggests that the learning time scales up much better than the planning time.

The most interesting result of the towers of Hanoi problem with two and three disks was that the system learned the optimal set of abstractions. This means that it not only learned the correct number of abstraction levels, but also the correct number of abstract operators for each level. Previous systems such as MPS, MAC-LEARN, and PiL2 are unable to learn these abstract operators.

Learning other Problem Solving Knowledge

This section describes methods for learning other problem solving strategies that can be represented in our common representation. One of the main advantages of a common representation is that not all operators in a refinement have the same level of abstraction. Therefore, other strategies such as reactive rules can be used. These strategies can be learned by comparing all refinements of a general operator.

After learning a new operator, the system uses additional heuristics to incorporate the new operator into the existing operator set. All new macros are part of the refinement of a more abstract operator (or the original initial state, goal state pair). The system compares the new operator to all other refinements of its abstract operator.

Raising Operators First the system tries to extract operators that occur in all refinements. The common operators are "raised" in the abstraction hierarchy, so that the planning system can focus on those operators. For example, given the abstract operator in figure 3, the operator to move the big disk is common in both refinements. In this case, the abstraction hierarchy is changed to reflect the fact that the operator `move-b` is an essential part of moving the big disk. The resulting abstract operator is similar to the RWM type operator subgoals [Güvenir and Ernst, 1990]. If the common operator occurs at the beginning of the sequence, a reactive rule is formed.

Generating Abstractions from Subsequences The system also extracts equivalent post-conditions resulting from the execution of operators in all refinements. If matching post-conditions are found, these states are extracted as new abstract operators. For example, assume that there are two operators to move the big disk, `move-b1` and `move-b2`. Furthermore, in figure 3, the system used `move-b1` in the first refinement and `move-b2` in the second refinement. In that case, there are no common operators in all refinements. Nevertheless, common to all refinements is a state where the small disk is on peg `$Y`. Therefore, the original problem is broken up into two abstract operators. The first one moves the small disk onto the medium peg, the second abstract operator moves the medium disk. The resulting operator hierarchy is identical to a subgoal sequence.

Failure When the system generates an unsuccessful plan, some of its expectations are wrong. If the user provides the system with additional information explaining why the plan failed (e.g. `(problem_x)`), the system can generate an abstract operator that relates the original problem to an elaboration of the problem, where the effects have additional conditions. (e.g. `(not problem_x)`). In the future, the planner is reminded of this problem and can avoid it.

Conclusions

The major contribution of this paper is the design of a learning system for a planner that combines macros, abstraction hierarchies, and case–based planning. The advantage of this approach is that techniques from both classical planning and case–based planning can be combined in the problem solving process.

The paper describes an analytical dynamic filtering scheme used to rule out inefficient operators. The dynamic filter is based on a formula relating the empirical *usefulness* and the length and branching factor of the operator. The common representation means that the dynamic filter can be applied to abstract operators and cases as well.

The paper also compares three previous approaches to the selection of new macros: Korf's MPS, Iba's MAC–LEARN, and Yamada's PiL2 system. From this comparison, guidelines are suggested for the selection of new operators. The goal in creating a new operator is trying to improve the current operator set. There are two ways in which an operator can be improved:

- Create an operator with more general pre–conditions. The effects of this operator can then be achieved in more states. The removal of predicates in pre–conditions leads to the generation of abstraction hierarchies.
- Create an operator with more specific effects. This removes side–effects of existing operators.

A macro learner was implemented and tested on a number of problems in the towers of Hanoi domain. As important parts of the complete planning systems are still missing, the implementation focused on comparing the learned macros to the ones learned by other systems. The results of the towers of Hanoi domain are promising.The system learned the optimal set of abstract operators for the two and three disk problems.

The paper also describes methods to learn diverse problem solving knowledge such as reactive rules, and automatic subgoaling. The next step in our research is the implementation of a complete planning system that incorporates these methods.

The paper also compares three different approaches to the selection of new macros. From this comparison guidelines are suggested for the selection of new operators. The main motivation for these heuristics is to find widely applicable operators with very specific effects.

References

Jacky Baltes. A symmetric version space algorithm for learning disjunctive string concepts. In *Proc. Fourth UNB Artificial Intelligence Symposium*, pages 55–65, Fredericton, New Brunswick, September 20–1 1991.

H. Altay Güvenir and George W. Ernst. Learning problem solving strategies using refinement and macro generation. *Artificial Intelligence*, 44(3):209–243, 1990.

Kristian J. Hammond. *Case Based Planning*. Academic Press Inc., 1989.

G. A. Iba. A heuristic approach to the discovery of macro–operators. *Machine Learning*, 3(4):285–318, 1989.

Craig A. Knoblock. *Automatically Generating Abstractions for Problem Solving*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1991. Tech. Report CMU-CS-91-120.

R. E. Korf. Macro–operators: A weak method for learning. *Artificial Intelligence*, 26(1):35–77, 1985.

R. E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33(1):65–88, 1987.

S. Minton. Selectively generalizing plans for problem solving. In *Proceedings of the Ninth International Conference on Artificial Intelligence*, pages 595–599, 1985.

P. Shell and J. Carbonnel. Towards a general framework for composing disjunctive and interative macro–operators. In N. S. Sridharan, editor, *IJCAI–89 Proceedings of the Eleventh International Joint Conference on Articifical Intelligence*, volume 1, pages 596–602, 1989.

S. Yamada and S. Tsuji. Selective learning of macro–operators with perfect causality. In N. S. Sridharan, editor, *IJCAI–89 Proceedings of the Eleventh International Joint Conference on Articifical Intelligence*, volume 1, pages 603–608, 1989.

Qiang Yang and Josh D. Tenenberg. ABTWEAK: Abstracting a nonlinear, least commitment planner. Technical report, University of Waterloo, 1990.

Search-Control Heuristics and Abstraction in Least-Commitment Planning

Anthony Barrett
Department of Computer Science and Engineering, FR-35
University of Washington
Seattle, WA 98195
barrett@cs.washington.edu

Abstract

This paper is a research summary on our work regarding the interaction of abstraction and search control heuristics. This work involves the least-commitment planner SNLP, an abstraction generation system similar to ALPINE to automaticly generate problem reformulations, and a system similar to STATIC to generate search-control heuristics. We describe an elegant way to make SNLP shift between representations automaticly.

Introduction

We are performing research on the interaction of abstraction and search control heuristics. To get this research started we have developed a lifted version of McAllister's partial order planning algorithm [6], called SNLP. Extending an incomplete plan in SNLP involves choosing what part of a plan to extend, and how to extend it. Choosing what to extend affects the structure of SNLP's search space, and choosing how to extend defines how the search space is traversed.

Previous work on the interaction of abstraction and search control rules acquired through EBL [5] used PRODIGY. PRODIGY is a total order planner that commits to planning decisions much earlier than SNLP. There are many differences between PRODIGY and SNLP. These differences cause problems as we adapt ALPINE [4] and STATIC [2] to generate abstractions and control rules for SNLP instead of PRODIGY.

This paper starts with a description of SNLP and shows how a planning decision can be delayed. The second part of the paper discusses strategies for deciding when to consider which planning decision. The third part discusses two search control heuristics for guiding a decision. The last part discusses the status of our research.

The Planner

As part of our research into step ordering commitment strategies when planning [1] we created a least commitment planner using a lifted version of McAllester's Algorithm [6]. Like many of the planners created since Sacerdoti's NOAH [7], our planner searches through a space of plans to find a plan that solves a STRIPS planning problem.

definition 1 *A* PLAN *is a triple:* $\prec S, O, B \succ$ *in which* S *denotes a set of plan steps (also known as actions),* O *denotes a set of ordering constraints that specify a partial order on* S*, and* B *denotes a set of binding constraints over the variables mentioned by the steps in* S*.*

The search starts with an *initial plan* which encodes the STRIPS planning problem to be solved. This plan consists of two steps s_0 and s_∞. The step s_0 adds the problem's initial conditions, and s_∞ requires the problem's goals (as preconditions). The algorithm that defines the search appears in figure 1. The algorithm is invoked with a plan P, a set of open goals G, and a set of causal links L. A causal link $S_i \xrightarrow{p} S_j$ is a triple denoting that a step S_i adds a proposition p to fulfill a precondition of step S_j. The set of open goals in G are the preconditions of steps S_j that do not have associated causal links $S_i \xrightarrow{p} S_j$ in L. In the initial invocation P, G, and L are the initial state, the preconditions of s_∞, and the empty set respectively. The algorithm works by solving open goals in G while protecting causal links in L from the effects of other steps.

This planner is provably complete for breadth-first and IDA* searches. Its least-commitment nature provides opportunities for reordering the consideration of different planning decisions. Step 2 can select any open goal at any time. A related algorithm provides even more flexibility in that it can perform causal link protection at any time in the planning process. Causal-link protection need not appear in step 5.

Deciding What to Refine

Our focus, for now, is on step 2. The order in which open goals are solved affects the difficulty of finding a solution. For instance, we implemented G as a stack and as a queue. For some of our experimental domains, the stack was exponentially worst than the queue. The reverse was true for other experiments.

Algorithm: *SNLP(P,G,L)*

1. **Termination:** If G is empty, report success and stop.
2. **Goal selection:** Let c be a proposition in G, and let S_{need} be the step for which c is a precondition.
3. **Operator selection:** Let S_{add} be an existing, or new, step that adds c. If no such step exists and none can be added then terminate and report failure. Let $L' = L \cup \{S_{add} \xrightarrow{c} S_{need}\}$. *Backtrack point: All existing and addable steps must be considered for completeness.*
4. **Update goal set:** Let $G' = (G - \{c\}) \cup$ the set of preconditions of the new step if one was added.
5. **Causal link protection:** For every step s_k that might affect a causal link $s_i \xrightarrow{p} s_j \in L'$:
   - Protect $s_i \xrightarrow{p} s_j$ from s_k by adding constraints between the 3 steps involved. *Backtrack point: All ways to protect $s_i \xrightarrow{p} s_j$ from s_k must be considered for completeness.*

   Let P' be the resulting plan.
6. **Recursive invocation:** *SNLP(P',G',L').*

Figure 1: The Planning Algorithm SNLP

Strategies for deciding which goal to handle next have a profound effect on the structure of the resulting search space. This is because delaying a planning decision increases the amount of information available when the decision is finally made. In the best case there would be enough information to force the decision. The optimal strategy would maximize the amount of relevant information and reduce the branching at each of the backtrack points. If the number of branches can be reduced to one at each point, then planning becomes trivial. Unfortunately, it is often the case that we cannot hold the number of branches to one, and determining the goal to select for reducing the branching factor is a nontrivial problem.

One strategy we have experimented with involves assigning a static priority value to each possibly open goal before starting the planning process. This is essentially the "abstractness" number mentioned by McAllister [6]. One way to define these priorities uses ALPINE. ALPINE was originally developed for PRODIGY, but it maps very easily into SNLP. The effect of using ALPINE ensures that a causal link $s_i \xrightarrow{p} s_j$ is never threatened when solving an open goal q when the priority of p is greater than q. This is due to ALPINE's *ordered monotonicity property*. This property has interesting effects when considering the causal structure of a plan.

The causal links in the set L can be thought of as defining a plan P's *causal structure* [10, 8, 3, 11]. A *causal support* for some proposition P is a minimal set of causal links, illustrated in figure 2 as arrows, between steps, illustrated as circles. There is a link to provide P, and every step mentioned in the causal support has all of its preconditions provided by links in the causal support.

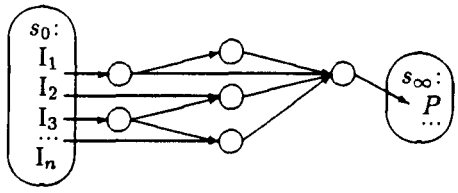


Figure 2: A causal support's structure.

Using ALPINE to assign priorities to open goals, our planner builts abstract causal supports for high-priority open goals, and then refines them as the priorities of the pending open goals in G decrease. Unfortunately, even though high-priority links never get threatened when low priority open goals are being solved, existing steps can threaten the causal links created while solving low-priority goals. Sometimes a new link cannot be protected from these threats. Such cases show that ALPINE does not give us the *downward solution property* [4]. This property states that if an abstract plan can be found, then it can always be refined into a less abstract plan [9]. ALPINE does give us the *downward failure property*. This ensures that if an abstract plan cannot be found, then a concrete plan cannot be found either.

Currently, our algorithm protects causal links as they are threatened. This is not necessary. It may not even be desirable since there are a huge number of ways to protect a causal link when propositions can have unbound variables. For example, there are 5 different sets of constraints that can be added to a plan to protect $s_i \xrightarrow{p} s_j$ (where p is `(on ?x ?y)`) from a step s_k that deletes `(on ?a ?b)`. They are:

1. $\{s_k$ before $s_i\}$
2. $\{s_k$ after $s_j\}$
3. $\{$`?x` $\neq$ `?a`, `?y` $\neq$ `?b`, and s_k between s_i and $s_j\}$
4. $\{$`?x` $\neq$ `?a`, `?y` $=$ `?b`, and s_k between s_i and $s_j\}$
5. $\{$`?x` $=$ `?a`, `?y` $\neq$ `?b`, and s_k between s_i and $s_j\}$

If causal link protection was delayed until all variables were bound, the threat might have taken care of itself as a side effect of other planning decisions. If the threat still exists when all variables are bound, then the last three branches can be avoided altogether.

Deciding How to Refine

Step 3 of the original algorithm is very simplistic in that it tries to refine an open goal in every way possible. For example, in the blocks world domain, to solve the goal `(on A B)` the original algorithm might generate the sequence of steps:

`(move A B) > (move A C) > (move A B)`

We are implementing techniques from STATIC to avoid generating these sequences. STATIC takes a domain description and generates a set of control rules for deciding how to resolve a goal taking into account the goal's purpose and the other steps currently in the plan. Like ALPINE, STATIC was originally created for PRODIGY. Unfortunately, problems occur when we try to apply it to improve SNLP. For instance, one of the problems that STATIC detects and generates rules to avoid is due to a concept called *goal stack cycles*. The example above is an instance of such a cycle for PRODIGY.

Unfortunatly, goal stacks are a artifact of the way PRODIGY plans. SNLP does not have them, but it has a related concept called a goal's *purpose*. This purpose is defined to be a list of propositions created by looking at the path of causal links that starts at the goal's associated step and ends at the step s_∞. Several examples of such paths appear in figure 2. At first glance SNLP's goal purposes seem to be identical to PRODIGY's goal stacks, but such is not the case. A PRODIGY goal has only one goal stack, but since there can be several paths from a step to s_∞, an SNLP goal can have more than one purpose. Also, as new causal links are added to a plan, the number of purposes for a goal increases.

We will use a system similar to STATIC to create a set of control rules. A goal's purpose specifies a set of control rules that apply in solving the goal. Since a goal may have more than one purpose, more than one set of control rules may apply. Each control rule that applies specifies some plan modification. STATIC/PRODIGY also matched conditions of a control rule against conditions provided by existing steps in order to further restrict branching. We cannot do this in SNLP and keep completeness. Also, as a goal's number of purposes increases, more control rules apply. This can happen after SNLP actually attempts to solve a goal. We have not figured out the best way to handle this yet.

Our planner performs a best first search over the planning search space. One of our ranking functions sums the number of causal links with the number of open goals. This gives us an A* search that prefers plans with fewer causal links. We hope to use the problem space graphs generated by STATIC to provide a better admissible ranking function.

Research Status

Currently we have implemented the planner, and a very simple version of ALPINE, which assigns a priority to a proposition's predicate name. We quickly discovered that we need a more refined priority function. One way to get this function is to observe that some goal propositions can never be added by an action. We can use these propositions to replace a small set of highly variablized action descriptions with a larger set of actions in which some of the variables are bound to constants. From this new set we can generate a set of priorities for propositions where some of the variables have been bound to constants. We have not implemented this yet.

Acknowledgements.

This research was improved by discussions with Dan Weld, Steve Hanks, Oren Etzioni, Scott Penberthy, and Craig Knoblock. It was funded in part by National Science Foundation Grant IRI-8957302, Office of Naval Research Grant 90-J-1904, and a grant from the Xerox corporation.

References

[1] A. Barrett, S. Soderland, and D. Weld. The Effect of Step-Order Representations on Planning. Technical Report 91-05-06, University of Washington, Department of Computer Science and Engineering, June 1991.

[2] Oren Etzioni. Static: A problem-space compiler for prodigy. In *the Proceedings of the Ninth National Conference on Artificial Intelligence.*, 1991.

[3] S. Kambhampati and J. Hendler. A Validation Structure Based Theory of Plan Modification and Reuse. *Artificial Intelligence*, to appear, 1992.

[4] C. Knoblock. *Automatically Generating Abstractions for Problem Solving.* PhD thesis, Carnegie Mellon University, 1991. Available as technical report CMU-CS-91-120.

[5] Craig Knoblock, Steve Minton, and Oren Etzioni. Integrating abstraction and explanation-based learning in prodigy. In *Proceedings of the Ninth National Conference on Artificial Intelligence.*, 1991.

[6] D. McAllester and D. Rosenblitt. Systematic Nonlinear Planning. In *Proceedings of AAAI-91*, pages 634–639, July 1991.

[7] E. Sacerdoti. The Nonlinear Nature of Plans. In *Proceedings of IJCAI-75*, pages 206–214, 1975.

[8] A. Tate. Generating Project Networks. In *Proceedings of IJCAI-77*, pages 888–893, 1977.

[9] J. Tenenberg. Abstraction in Planning. Ph.d. thesis, University of Rochester, Department of Computer Science, May 1988.

[10] D. Warren. WARPLAN: A system for generating plans. Memo No. 76, Univerity of Edinburgh, Department of Computational Logic, 1974.

[11] Q. Yang and J. Tenenberg. ABTWEAK: Abstracting a Nonlinear, Least-Commitment Planner. In *Proceedings of AAAI-90*, pages 204–209, August 1990.

TOWARDS AN EFFECTIVE THEORY OF REFORMULATION
Part 1: Semantics

D. Paul Benjamin
Department of Mathematics and Computer Science
St. Joseph's University
5600 City Avenue
Philadelphia, PA 19131-1395
pbenjami@sju.edu

ABSTRACT

This paper describes an investigation into the structure of representations of sets of actions, utilizing semigroup theory. The goals of this project are twofold: to shed light on the relationship between tasks and representations, leading to a classification of tasks according to the representations they admit; and to develop techniques for automatically transforming representations so as to improve problem-solving performance. A method is demonstrated for automatically generating serial algorithms for representations whose actions form a finite group. This method is then extended to representations whose actions form a finite inverse semigroup.

Introduction

This paper describes an algebraic approach to building systems that can automatically change their representations. Representation change, also called *reformulation*, has long been recognized as an essential component of intelligent systems (Amarel, 1968) (Simon, 1969), but the automation of representation change has proved elusive. The understanding of representations and their properties lags far behind the understanding of search methods and their properties. This difference is reflected in the structure of AI programs: most contain a large number of search methods acting on a single representation. This was true for GPS, and remains true today, e.g., SOAR, Prodigy, and automated theorem provers, which typically possess a multitude of variants of resolution acting on a representation in normal form.

This paper attempts to begin to rectify this situation, with a formal investigation of the properties of representations, and algorithms for representation change. This paper does not examine representation changes that are heuristic or inductive (these have been investigated by a large number of researchers in machine learning), but rather *deductive reformulations* that preserve logical soundness and completeness: no solvable problems are rendered unsolvable, nor are unsolvable problems rendered solvable.

Deductive reformulations are much less well understood than heuristic or inductive transformations. In this type of reformulation, representations are not changed to alter their logical properties, but are changed to improve their *computational* properties, especially their search and input characteristics. As we will see, these computational properties can be well characterized algebraically.

Representations

It is well understood that representation selection sets the stage for both problem solving and learning, and that the choice of representation can greatly affect the cost of both. The examples in the next section will illustrate that the proper choice of representation is data-dependent, so how can a system know the best concept language for the data before seeing the data?

This leads to a problem: a system must choose a representation before it can know what representations would be good.

In AI practice, this problem is resolved by the humans who develop the system. They have prior knowledge of the classes of tasks that the system will face and the demands that will be placed on it, and they engineer a representation whose properties will aid the system in meeting these demands.

This has led to the current situation in AI: research is concentrated almost exclusively on development of planning and learning algorithms, and these algorithms are cast as search in problem spaces and concept spaces, respectively. The search through the space of representations is performed by skilled humans. Although research on planning and learning algorithms is certainly important, the neglect of research on reformulation has led to three major limitations of AI research.

First, a wide variety of truly autonomous systems cannot be constructed as long as skilled humans are required to engineer the representations for the systems. It will only remain possible to build expert systems that can function in small, static domains, in which the representational demands on the system do not change over time. This limitation has special significance for the application of AI techniques to robotics.

Second, the dependence of planning and learning algorithms on the properties of the representation is unstated in AI papers, thereby raising questions about the validity of the conclusions drawn about the properties of the algorithms, as it is unclear how to separate the properties of the algorithms from those of the representations. This leads to the unsettling possibility that researchers may have (subconsciously) engineered representations that cause planning or learning algorithms to perform well. If true, research results would be irreproducible (as other researchers might engineer different representations), and the underpinnings of AI as a science would we weakened.

Third, this leads to very narrow conceptions of problem solving activities. For example, research in planning has focused on algorithms that construct a set of behaviors for the agent to exhibit for a particular task. These behaviors may be organized so that different behaviors are executed dependent on runtime conditions in the environment, but the limitation is that planning has been conceived as the process of constructing this set of behaviors. This conception has been challenged by recent work (Agre & Chapman, etc.) which argues that in complex domains the number of behaviors necessary for a successful plan is too large to construct before execution. Instead, a system plans by *designing a program* that will generate at execution time a behavior to attain the goal. In this new conception, planning becomes a design process, consisting of repeated cycles of design and performance testing. The design steps consist of both representation design and algorithm design, and the performance testing is the actual execution. In this way, the planner is constantly redesigning the program (if necessary) *during* execution. (Note that the bees of Agre & Chapman or the robot insects of Brooks are programs that are designed by humans, so the planning was done by the humans.) The classical conception of planning as constructing a set of behaviors is a special case, when the program simply consists of the actions to perform in various situations.

Similarly, research in learning has primarily focused on algorithms for constructing a new hypothesis from an existing hypothesis, given a set of new examples. But, beginning with the work of Mitchell and Utgoff, machine learning researchers began to realize the importance of representation design. It is by now widely recognized in the machine learning community that the design of the hypothesis language is crucial in efficient learning: the language must be restricted to permit the system to successfully identify a hypothesis without having to see all the possible cases, but choosing a sublanguage that doesn't contain any good hypotheses will lead to failure no matter what learning algorithm is used. Recently, conferences and workshops have been held on this topic, and books have begun to appear. In this respect,

the machine learning community is ahead of the planning community. Researchers in planning should note that Amarel's seminal paper dealt with reformulation in problem solving, not in learning.

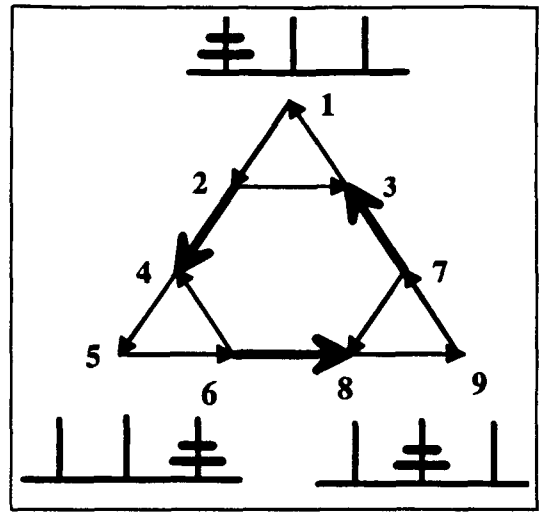
As a result of these three limitations, we are led to the conclusions that representation change is a necessary capability of any autonomous intelligent system, and that AI needs a fuller understanding of representations and their properties. In the next section, we consider the types of properties that characterize good and bad representations, to understand the goals of reformulation.

Computational Properties of Representations

Representations vary as to the amount of search they require, the input they require, their memory usage, etc. Similarly, agents vary as to their memory, sensory, and motor capabilities. Tasks have constraints on usage of various resources, e.g., time. We have argued that only agents with the capability to change representations can select a representation whose characteristics are appropriate for the particular task at hand. For example, the agent may not need to find a globally optimal solution, but only one that meets certain criteria; in this case, the agent may be able to simplify the task description, and find an acceptable solution more quickly.

In order to investigate the relevant properties of representations, we must first choose the appropriate tools. Virtually all of the knowledge representation community uses the tools of logic to investigate the properties of representations. Certainly, the soundness, completeness, and complexity of a representation are important properties; however, in this paper we are concerned with the *computational* properties of a representation, rather than whether it models the task environment accurately in all cases. The computational properties of a representation are independent of soundness, completeness, or complexity. To see this, consider the following two representations of the two-disk Towers of Hanoi:

Representation TOH1: Let us number the nine states of the 2-disk Towers of Hanoi:



Let the two possible actions be denoted by "x" and "y". "x" moves the small disk right one peg (wrapping around from peg 3 to peg 1), and "y" moves the large disk one peg to the left (wrapping around from peg 1 to peg 3). In the figure, "x" is shown by narrow, counterclockwise arrows, and "y" is shown by thick, counterclockwise arrows.

Representation TOH2: Let the states be numbered in the same way, and let the six possible actions be:

X1 = move the top disk from peg 1 to peg 2

X2 = move the top disk from peg 2 to peg 3

X3 = move the top disk from peg 3 to peg 1

Y1 = move the top disk from peg 1 to peg 3

Y2 = move the top disk from peg 2 to peg 1

Y3 = move the top disk from peg 3 to peg 2

These two representations are both sound and complete. Furthermore, they have exactly the same complexity, as they have the same number of states and possible actions in each state. The only difference is in the labeling of the actions.

Yet, these two representations have very different computational properties: the first representation *decomposes:* it has a subgoal

reduction, whereas the second has none. This decomposition permits an agent to solve each subgoal independently, and then compose the solutions to form a solution to the task. In this case, the set of actions decomposes into actions for moving the larger disk and actions for moving the smaller disk, permitting the system to first bring one disk to its goal position and then bring the other disk to its goal position without disturbing the position of the first disk. As we will see, it is possible to solve either disk first and then the other.

Certainly it is possible for a system using the second representation to bring one disk to its goal position and then bring the other disk to its goal position; obviously, it must do so to solve the task. However, *there is no structure in this second representation of actions that can be used to find this decomposition*, i.e., the actions do not admit a subgoal reduction.

Thus, we see that a good representation facilitates problem solving by structuring the knowledge in a way that helps the agent to identify relevant actions - the actions for the first subgoal. We also see that we cannot characterize this structure by considering soundness, completeness, or complexity. This approach is consonant with the ideas of Doyle & Patil (1991), who argue that "logical soundness, completeness, and worst-case complexity are inadequate measures" for evaluating representations. We are therefore led to consider an alternative formal method of chararacterizing the structure of sets of actions. One of the primary purposes of this paper is to show that the tools of algebra are well suited to this purpose.

In particular, the method used in this work is to apply the theory of semigroups to the analysis of representations of actions, to yield both an intuitive understanding of representations and algorithms for reformulation. The theory of semigroups is important in the study of algebraic linguistics (Chomsky, 1957), (Lallement, 1979), so it is not surprising that it can prove useful in the study of the languages used to represent tasks.

This paper describes only the semantics of representation change, i.e., it examines the structure of sets of actions. The various symbolic encodings of each such structure in terms of state description functions is agent-dependent and deserving of a separate treatment, and so will be examined in a subsequent paper.

A Prototypical Example of Reformulation

To get a more intuitive feel for the issues involved in reformulation, let us first consider a familiar example. When we are posed the problem of finding the volume of a cylinder in an arbitrary position, the first thing we do is change the coordinates of the problem so that an axis passes lengthwise through the middle of the cylinder (the coordinates are moved, not the cylinder).

We do this because otherwise the calculations are very expensive. For example, we could compute derivatives at two places on the edge of one of the circular ends, find perpendicular lines (with slopes that are negative reciprocals of the tangent lines), find the intersection point of these lines (the center of the circle), and use the distance formula to find the radius of the circle. We could then compute the area of the circle, and apply the distance formula again to yield the length of the side of the cylinder. A final multiplication gives the volume. This is a very expensive procedure involving 3-dimensional calculations. Another computationally expensive possibility is performing an integration to find the volume.

Changing the basis gives a nice representation of the cylinder. Now, all we need to do is read the x-value when y and z are both zero to get the radius, and read the z-value when x and y are both zero to get the length. Just two multiplications are required (squaring the radius and multiplying the areas by the length). No 3-dimensional computations are used. The 3-dimensional problem has been decomposed into two 2-dimensional subproblems: finding the area of the circle and extending this area through the length of the cylinder. Note especially the reduction in the perceptual and memory abilities required of the problem solver: it need only be able to read values at which the surface intersects

coordinate planes, which are single numbers, and need only manipulate two numbers at a time. This contrasts with the original representation, which requires the problem solver to read triples of numbers, and to be able to simultaneously store several numbers at a time, e.g., the equations of the two lines that intersect at the center of the circle. Low memory and perceptual (input) cost are key computational properties of a good representation, and hence are important goals for reformulation.

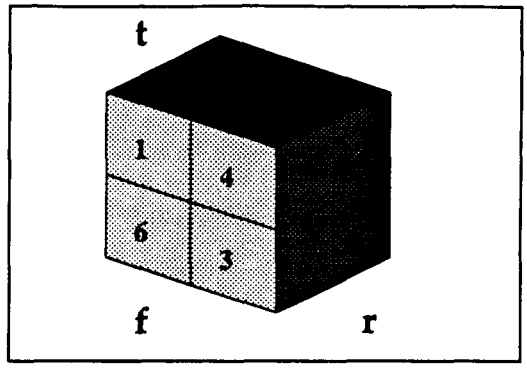
The subproblems are obtained by projecting the cylinder onto the x-axis and z-axis, respectively. In the new coordinates, good subproblems are obtained by projection. In the original coordinates, this is not the case; projecting onto any coordinate axis or coordinate plane yields a subproblem that is not cheaper to solve than the original problem.

As long as the coordinate change process is not too expensive, this will result in a net savings, especially if many computations are performed on the cylinder. Good subproblems are characterized in this case by their dimensionality: the lower the dimensionality, the better the subproblem. The goal of general reformulation is to find a representation that facilitates problem solving by permitting projection to more tractable subproblems, i.e., by permitting creation of good abstractions.

The 2x2x2 Rubik's Cube

It is remarkable that we can use this approach to reformulation on tasks that appear very different. Let us examine the 2x2x2 Rubik's Cube. The techniques we will use here scale up; we are using this small Cube to save space in the paper. Let the 8 cubicles (the fixed positions) in the 2x2x2 Cube be numbered as in the figure (8 is the number of the hidden cubicle).

Number the cubies (the movable, colored cubes) similarly, and let the goal be to get each cubie in the cubicle with the same number. For brevity of presentation, we will consider only 180° twists of the cube.



Let f, r, and t denote 180° clockwise turns of the front, right, and top, respectively (cubie 8 is held fixed; Dorst (1989) shows that this is equivalent to factoring by the Euclidean group in three dimensions). Note that this cubie numbering is just a shorthand for labeling each cubie by its unique coloring. This holds true for the Cube with only 180° twists, as position determines orientation.

Finding Serial Algorithms for Tasks Represented by Groups

Finite groups can be reformulated utilizing group representation theory to find coset decompositions. This is illustrated on the 2x2x2 Cube. We use group representation theory to represent f, r, and t as matrices:

$$f = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$r = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

$$t = \begin{pmatrix} 0&1&0&0&0&0&0 \\ 1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1 \end{pmatrix}$$

These matrices are 7-dimensional, corresponding to the 7 unsolved cubies. The reformulation method consists of finding eigenvectors of eigenvalue 1; these are the invariants. Any invariant of all the actions is irrelevant for the task, and can be removed, by first changing the coordinate system so that the invariant eigenvectors are axes, and then projecting to the noninvariant subspace, removing all irrelevant information at once. In this case, the eigenvectors are:

$$r: \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \text{for } \lambda = 1, \text{ and } \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} \text{for } \lambda = -1$$

$$f: \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \text{for } \lambda = 1, \text{ and } \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \text{for } \lambda = -1$$

$$t: \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \text{for } \lambda = 1, \text{ and } \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix} \text{for } \lambda = -1$$

and the common invariant eigenvector is:

$$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Note that we have abbreviated the above eigenvectors to save space; they are actually 7-vectors. We then change the basis, yielding the new representations for r, f, and t:

$$r = \begin{pmatrix} 1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0 \\ 0&0&\frac{1}{2}&\frac{\sqrt{3}}{2}&0&0&0 \\ 0&0&\frac{\sqrt{3}}{2}&-\frac{1}{2}&0&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&-1 \\ 0&0&0&0&0&-1&0 \end{pmatrix}$$

$$f = \begin{pmatrix} 1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0 \\ 0&0&\frac{1}{2}&-\frac{\sqrt{3}}{2}&0&0&0 \\ 0&0&-\frac{\sqrt{3}}{2}&\frac{-1}{2}&0&0&0 \\ 0&0&0&0&0&0&-1 \\ 0&0&0&0&0&1&0 \\ 0&0&0&0&-1&0&0 \end{pmatrix}$$

$$t = \begin{pmatrix} 1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0 \\ 0&0&-1&0&0&0&0 \\ 0&0&0&1&0&0&0 \\ 0&0&0&0&0&-1&0 \\ 0&0&0&0&-1&0&0 \\ 0&0&0&0&0&0&1 \end{pmatrix}$$

This procedure computes the irreducible invariants of a group. The irreducible factors of dimension 1, 1, 2, and 3 are found along the diagonals of the matrices. Projecting to these subspaces yields two interesting subproblems:

$$r = \begin{pmatrix} \frac{1}{2} & \frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & -\frac{1}{2} \end{pmatrix} \qquad f = \begin{pmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} \\ -\frac{\sqrt{3}}{2} & -\frac{1}{2} \end{pmatrix}$$

$$t = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$$

On cubelets 1, 2, and 3, the subgroup generated is {i. r. f. t. rt. tr}.

$$r = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{pmatrix} \qquad f = \begin{pmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix}$$

$$t = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

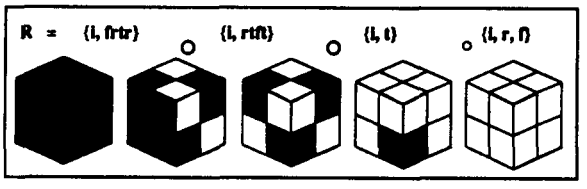
On cubelets 4, 5, 6, and 7, the subgroup generated is

{i, r, f, t, rf, rt, fr, ft, tr, tf, rfr, rft, rtr, rtf, frt, ftr, ftf, trf, tfr, rfrt, rftr, rftf, rtrf, rtfr}.

Using each set of matrices as generators, we get two subgroups of actions, the second of which is a faithful representation of the whole group. The first subgroup moves cubies 1,2, and 3, while holding 4,5,6, and 7 in position. The second subgroup moves cubies 4,5,6, and 7 while holding 1,2, and 3 in their positions. We then repeat this procedure on the first set of actions to obtain a full set of prime factors of the Rubik group.

These factors can be assembled in different ways to form serial algorithms. There is more than one way to decompose this group. This is analogous to the different ways of multiplying the prime factors of a number. Five serial algorithms are obtained in this way. We now examine two of them.

Serial Algorithm 1:



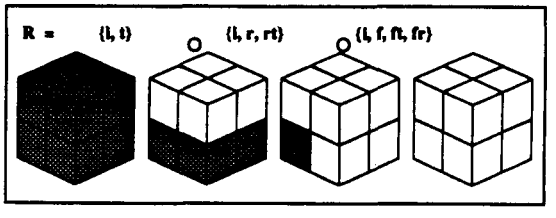
1. One of { i,r,f } brings cubelet 3 into cubicle 3.
2. One of { i,t } brings cubelet 1 and 2 into their places.
3. One of { i, rtft } brings (4,6) and (5,7) in the proper planes (`the front face looks right').
4. One of { i, frtr } finishes the Cube.

The above figure is read right-to-left; solved cubicles are shaded. "i" denotes the identity (null) action. The average number of moves required to solve the Cube in this way is 5.17.

Each step in the decomposition corresponds to bringing a feature to its goal value. Subsequent steps hold that value invariant. In this way, sensory planning is decomposed, i.e., the agent need only sense part of the Cube at each step. For example, the first step solves cubicle 3. Knowing the colors of the solved cubicle 8, we know the colors of cubicle 3 - it has the same color as the bottom of cubicle 8, and two new colors. There is only one such cubie, and it must be in one of three locations: in its goal position, or in cubicle 1 or 2. The agent need only look in those 3 locations to determine what action to take. Once cubicle 3 is solved, it need not be sensed again. The agent next solves cubicles 1 and 2; it need only sense either position to see if the proper cubie is there; if so, it does nothing, otherwise, it twists the top. Finally, the agent uses macros to solve the remaining four cubicles, by examining the front face to see if it's a uniform color, and then examining the top or right face to see it is of uniform color.

This reduction in the complexity of sensing (the input requirements) is one of the salient aspects of task decomposition. In large, realistic tasks, it is not possible to fully sense the world, e.g., in a changing environment one part of the world may change while the agent is sensing another part. Even when possible, it is often too expensive. A good decomposition can greatly reduce the sensory expense. This gain, however, is at the cost of suboptimality of the solution. The above decomposition has average cost of 5.17, whereas an optimal solution is of average length 2.46. There are better decompositions. We now examine the best decomposition.

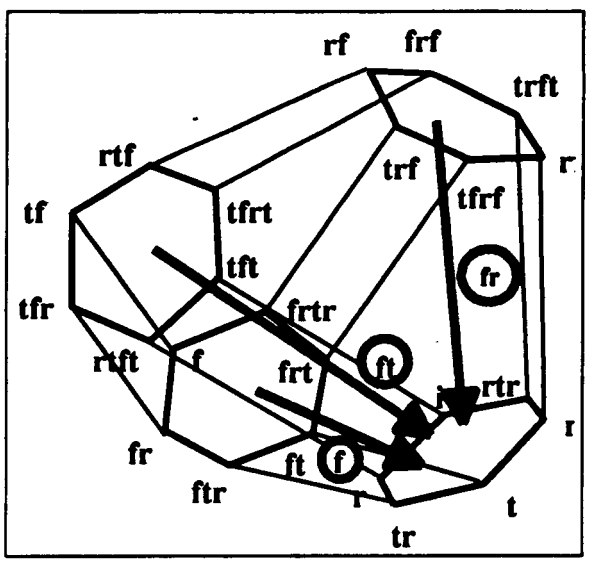
Serial Algorithm 2:



1. One of { i,f,fr,ft } brings cubelet 6 into cubicle 6.
2. One of { i,r,rt } brings 3,7 in place (bottom layer correct).
3. One of { i, t} finishes the Cube.

The average number of moves to solve the Cube using this decomposition is 2.75.

Each decomposition can be thought of as a coordinate system whose origin is the goal state. For example, the second serial algorithm can be thought of as a 3-dimensional coordinate system (a,b,c) where a is in {i,f,ft,fr}, b is in {i,r,rt}, and c is in {i,t} (Leo Dorst produced the geometric interpretation of this coordinate system):



The first coordinate brings us to the proper hexagon, the second coordinate to the proper pair of opposing vertices in the hexagon, and the third coordinate to the goal state.

In this coordinate system, each subproblem is obtained by projecting onto that coordinate. For example, projecting to the first coordinate yields a 4-element state space whose states are the hexagons. Reaching the goal state (hexagon) in this space is equivalent to the subproblem of bringing cubie 3 into its goal location.

From the Rubik's Cube example, we see that we can view a representation as a coordinate system whose axes are the components of the task. Using group representation theory, we represent the actions as matrices. Changing the basis so that invariant eigenvectors are axes eliminates irrelevant information, and identifies a good task decomposition. We now formalize this notion in a general way.

Coordinate Systems in Transformation Monoids

We are interested in the structure of transformation monoids, so a natural first step is to examine Green's relations (Lallement, 1979). Green's relations are defined as follows: given any semigroup S, we define the following equivalence relations on S:

$a \mathrel{R} b$ iff $aS^1 = bS^1$

$a \mathrel{L} b$ iff $S^1a = S^1b$

$H = R \cap L$

$D = R \vee L$

$a \mathrel{J} b$ iff $S^1aS^1 = S^1bS^1$

where S^1 denotes the monoid corresponding to S with an identity element adjoined.

Intuitively, we can think of these relations in the following way: aRb iff for any plan that begins with "a", there exists a plan beginning with "b" that yields the same behavior; aLb iff for any plan that ends with "a", there exists a plan ending with "b" that yields the same behavior; aHb indicates functional equivalence, in the sense that for any plan containing an "a" there is a plan containing "b" that yields the same behavior; two elements in different D-classes are functionally dissimilar, in that no plan containing either can exhibit the same behavior as any plan containing the other.

Let us examine these relations in a representation for the Towers of Hanoi. Let Q = {1,2,3,4,5,6,7,8,9} be the set of states for the 2-disk Towers of Hanoi. Let A be the semigroup of transformations generated by:

$$x = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 2 & 3 & 1 & 5 & 6 & 4 & 8 & 9 & 7 \end{pmatrix}$$

$$y = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ & 4 & & & & 8 & 3 & & \end{pmatrix}$$

"x" moves the small disk right one peg (wrapping around from peg 3 to peg 1), and "y" moves the large disk one peg to the left (wrapping around from peg 1 to peg 3). Then A is a semigroup with 31 elements. We name this representation TOH1. Each element of A is a partial function on the set of states. Green's relations in A are:

D0 **0**	D2	**x, xx, xxx** D1
y, yxxy, yxxyxxy	yx, yxxyx, yxxyxxyx	yxx, yxxyxx, **yxxyxxyxx**
xy, xyxxy, xyxxyxxy	xyx, xyxxyx, **xyxxyxxyx**	xyxx, xyxxyxx, xyxxyxxyxx
xxy, xxyxxy, **xxyxxyxxy**	xxyx, xxyxxyx, xxyxxyxxyx	xxyxx, xxyxxy, xxyxxyxxyxx

where the R-classes are horizontal and the L-classes are vertical. The D-classes model the structure of the task representation, in the sense that the n-th D-class is equivalent to the n-disk Towers of Hanoi. Adding additional disks merely adds additional D-classes. Each D-class Dn contains the macros that move all of the disks 1 through n.

Let us examine D2. There are three subgroups in this D-class, containing the idempotents (an idempotent is an element x such that xx = x). The idempotents are in bold type. These three H-classes are maximal subgroups of A and their generators are the macros for moving the large disk. Now we define a coordinate system for any semigroup.

Definition. Let R be an R-class of a semigroup, and let H_λ ($\lambda \in \Lambda$) be the set of H-classes contained in R. A *coordinate system* for R is a selection of a particular H-class, denoted H_1 contained in R, and of elements q_λ, $q'_\lambda \in S^1$ with $\lambda \in \Lambda$, such that the mappings $x \rightarrow xq_\lambda$ and $y \rightarrow yq'_\lambda$ are bijections from H_1 to H_λ and from H_λ to H_1, respectively. A coordinate system for R is denoted by $[H_1; \{(q_\lambda, q'_\lambda): \lambda \in \Lambda\}]$.

This says to choose an H-class in a D-class, and find 1-1 mappings to all other H-classes in the same R-class. We are justified in calling this a coordinate system for a D-class, as any two representations of coordinate systems for any two R-classes are isomorphic, giving 1-1 mappings to all the H-classes in the D-class (Lallement, 1979, p.46).

This definition of coordinate system provides an intuitive conceptual framework for homomorphic reformulation. The groups in the decomposition can be viewed as levels of an abstraction hierarchy, or subproblems in a serialization. Each such decomposition yields a coordinate system, which is the index of the group in the decomposition, together with the indices given by the decomposition of the group, as illustrated in Rubik's Cube. Change of representation involves generating a different decomposition for the given monoid, and thus is a change of coordinate systems. This fits perfectly with the intuition we developed in the cylinder example.

A good example of such reformulation is switching between serial algorithms for Rubik's Cube. Such reformulation is performed to match the unique characteristics of each decomposition to the characteristics of the agent and the requirements of the task, e.g., serial algorithm 2 has a lower expected search cost, whereas serial algorithm 1 never requires sensing cubicle 5.

Each coordinate system generates a Rees matrix representation for A, permitting us to change basis within a semigroup and find serial algorithms in a manner analogous to the Rubik's Cube example. The reader is referred to Lallement (1979) for details of Rees matrix representations. Unfortunately, application of this technique to general semigroups can be very expensive computationally. Even the

decomposition of small semigroups may require a large number of groups. The minimal number of groups required for a decomposition of a given semigroup is called the *group complexity* of that semigroup. It is not known whether the group complexity is decidable. This makes it very difficult to design good algorithms for finding such decompositions.

Even more seriously, this form of representation change is not fully general. Homomorphic reformulation techniques elucidate the structure of a transformation semigroup, and thus possess a serious limitation: they can only preserve the structure of the semigroup, which limits the components they can produce. Such techniques can only remove extraneous information to uncover existing structure in a given representation. If this structure is not appropriate for efficient problem-solving, then homomorphic reformulation will be of little use.

For example, in ABSTRIPS (Sacerdoti, 1974) the relevant predicates must already exist in the initial representation, or else numbers cannot be assigned to them. Another example is provided by Subramanian's work: if the theory is stated in such a way that the irrelevant information is distributed among the statements of the theory, rather than concentrated in a subset of the statements, then it cannot be dropped without rendering the theory incapable of solving the task. TOH2 is such a representation.

For these reasons, we utilize the technique described in this section only within group machines. In the next section, we will show how to extend this technique to handle a wider class of semigroups - inverse semigroups.

Related Work

Reformulating tasks in this way has been described in various ways in the literature. Sacerdoti (1974), Knoblock et al. (1990), and Unruh & Rosenbloom (1989), among others, describe this reformulation as building an abstraction hierarchy. For example, in ABSTRIPS an ordering was imposed on the state-description predicates; bringing the predicates to their goal values in this order was viewed as top-down search in a hierarchy of abstract problem descriptions.

Niizuma & Kitahashi (1985) and Banerji & Ernst (1977) describe this reformulation as projecting the states. In this view, an equivalence relation is imposed on the states, and the equivalence classes are the states in the quotient space. The only actions retained in the new representation are those that move between equivalence classes.

Zimmer (1990) and Benjamin et al. (1990) describe this reformulation as decomposing the actions. In this approach, the set of sequences of actions is decomposed into two sets: those that are most relevant (according to some criterion) for solving the problem, and those that are less relevant. This induces an equivalence relation on the set of states, as in the previously described approach; a difference is that sequences of actions (macros) are used, rather than actions. The decomposition procedure is then repeated on the less relevant actions.

A similar approach is taken by Subramanian (1987), who drops statements from a theory if the reduced theory can still derive the goal statement; the dropped statements are considered irrelevant. In these approaches, the state space is reduced by removing states that can no longer be reached by actions (statements) retained in the representation (theory). These approaches differ from the state projection approach mainly in the order in which states and actions are reformulated. In the state projection approach, a feature is chosen, inducing an equivalence relation that factors the states and decomposes the actions. In the action decomposition approach, the sequences of actions are decomposed according to some criterion, e.g., irrelevance (Subramanian) or enablement (Benjamin), which induces an equivalence relation on the states.

Korf (1983) and Riddle (1986) describe this reformulation as serializing the subgoals. Finding a set of serializable subgoals for a problem permits solution of the problem by solving each subgoal in order. Korf points out that this reduces the exponent of the search, possibly resulting in a big gain in efficiency.

Most of these authors refer to this type of reformulation in more than one of the above four ways. Also, this is not an exhaustive list of work on this type of reformulation. In the remainder of this paper, we refer to this type of representation change as *homomorphic reformulation*, as in Lowry (1990).

The General Reformulation Problem

Homomorphic reformulation changes the presentation of a semigroup, thus "re-presenting" it. The toughest cases of reformulation occur when the necessary problem-solving structures do not already exist, and involve transforming the semigroup into a *transformationally equivalent* semigroup with the desired structures. We call this the *general reformulation problem*. In keeping with our intuition that homomorphic reformulation is a coordinate change, we call non-homomorphic reformulation a *deformation*, because it changes the structure of the set of actions.

We begin our examination of the general reformulation problem by describing a representation for the Towers of Hanoi that lacks good decompositions. We then define transformational equivalence, and give an algorithm for computing transformational equivalence for a useful class of semigroups.

An Example: TOH2

In TOH2, the only feature available to the agent is what disk is on top of each peg. This is a sound and complete theory of the 2-disk Towers of Hanoi, just as TOH1 is. The search complexity of this representation is exactly the same as for TOH1, because the states are the same, the same number of actions are executable in any state, and the solutions are of the same length. Thus, we see the insufficiency of logical completeness, soundness, and worst-case complexity for evaluating representations.

The actions of TOH2 do not mention the disk that is moved, and no abstractions can be generated. We cannot find an abstraction hierarchy that first solves the large disk, then the small disk, because the set of actions cannot be partitioned into moves for each disk. Certainly the agent can first bring the large disk to the goal peg, then the small disk, but as was pointed out earlier, there is no structure in this representation of the set of actions that can be used to find that subgoal ordering, and the set of actions has no decomposition. No matter how we project these actions, we end up with all six of them. Thus, we cannot apply the type of reformulation we applied to the cylinder or to Rubik's Cube. No re-presentation of this transformation monoid will help; we need a new monoid of actions.

This is a different semigroup than in representation TOH1, and that its structure does not reflect the structure of the task in as helpful a manner. Relevant distinctions are not made, e.g., between moving the larger disk from peg1 to peg2 and moving the smaller disk between peg1 and peg2; irrelevant distinctions are made, e.g., between moving a disk from peg1 to peg2 and moving the same disk from peg2 to peg3.

This semigroup possesses only trivial (one-element) subgroups. We must find a way of transforming this semigroup to a better one.

Transformational Equivalence

Although the representations TOH1 and TOH2 are structurally dissimilar, they both have the Towers of Hanoi as a model, and thus map the states of the Towers of Hanoi in a logically equivalent fashion. We state this precisely with the following definitions:

Definition. Given two semigroups S1 and S2 acting on Q1 and Q2, respectively, a function f: Q1 $\rightarrow$ Q2 is said to be a *transformational reduction* if for all p,q $\in$ Q1, if q is reachable from p via S1 then f(q) is reachable from f(p) via S2.

Definition. Two semigroups S1 and S2 acting on Q1 and Q2, respectively, are said to be *transformationally equivalent* if there exist transformational reductions f: Q1 $\rightarrow$ Q2 and g: Q2 $\rightarrow$ Q1.

The preservation of reachability guarantees that any solution in one representation is a solution in the other. We will call a transformational reduction a *t-reduction*, and transformational equivalence will similarly be called a *t-equivalence*. Semigroup morphisms are t-reductions; however, not all t-reductions are semigroup morphisms. For example, TOH1 and TOH2 are t-equivalent, but there are no semigroup morphisms between them (there can be no function from A1 and A2, or from A2 to A1.) Neither is a simulation or abstraction of the other.

Computation of t-reductions can be extremely expensive. By restricting (specializing) and combining (by disjunction) elements of the semigroup A2 of representation TOH2, we can transform A2 in a general way to obtain any semigroup of actions that transforms Q2 in a similar manner; however, the number of ways of transforming a set of partial functions in this way is hyperexponential in the number of elements of A2. To make this problem tractable, we proceed by investigating one class of semigroups at a time. We examine the structure of semigroups of that class, and construct an algorithm that transforms that structure into t-equivalent semigroups of a class with superior computational properties. In the next section, we will describe such an algorithm, which transforms inverse semigroups into t-equivalent groups.

Transforming Inverse Semigroups into t-equivalent Groups

As we have seen, finite group representations possess an excellent matrix representation theory that permits efficient computation of serial algorithms. Finite group representations also possess another very useful property: all the actions in a transformation group are totally defined. The absence of partially defined actions means that there are no constraints on application of actions, and therefore a problem solver need not test actions for applicability when generating and testing possible actions. In the AI literature, this is referred to as "embedding the constraints in the generator." Testing partial actions is responsible for much of the time spent by search algorithms. For example, a production system spends much of its time attempting to instantiate rules that do not fully match. Thus, if a task admits a group representation, it is very desirable to find that representation. Consider a task that admits an inverse semigroup representation.

Definition. A semigroup S is an *inverse semigroup* if for any element a of S, there exists an element b of S such that aba = a.

Many interesting tasks admit inverse semigroup representations, including many AI tasks, e.g., the Towers of Hanoi, Rubik's Cube, the Missionaries and Cannibals, Fool's Disk, the Blocks World, and the 8-Puzzle. Also included are many motion and assembly tasks, e.g., parking a car. Intuitively, a task admits an inverse semigroup representation if it is true that whenever any sequence of actions s is performed in a state q, there exists a sequence of actions w that will return to state q.

We state the following theorem, but omit the lengthy proof to save space.

Theorem. Any task admits a finite inverse semigroup representation iff it admits a finite group representation.

In the next section, we will illustrate the procedure for transforming inverse semigroups to groups with two algorithms, which form the core of the proof of the theorem.

Algorithms

The transformation of inverse semigroups into groups is illustrated on various representations for the Towers of Hanoi.

Reformulating TOH1

Consider D2 of TOH1. The primitive idempotents are in nontrivial subgroups. The reformulation algorithm in this case is:

- Compute Green's relations.
- Find the primitive idempotents.

- Find the generators of the corresponding subgroups.
- Select a coordinate system originating at one of the subgroups.
- Map each generator and all its corresponding generators under the coordinate system to one new label.

The primitive idempotents are shown in bold type in Figure 12.The generators of the subgroups are xxy, xyx, and yxx. The renaming process in this case just relabels these three to one new label, forming the disjunctive macro:

Define z = case {

little disk left of large disk: xxy

little disk on large disk: xyx

little disk right of large disk: yxx }

This new action is globally applicable, and moves the two disks so that their relative position is unchanged. The identification of "the relative position of the two disks" as the discriminating feature is not addressed in this paper; it will be addressed in part 2, which will deal with the syntactic aspects of reformulation. The present paper is concerned only with the functions that features must compute, not with the formulae for computing these functions.

This construction gives a partial morphism from A to a group generated by z, with the relation zzz = 1. This partial map is defined only on the nine elements contained in the three group H-classes. Any such partial map can be extended with the identity map on all totally defined actions. In TOH1, this means mapping the action "x" to itself, giving a group G_1 generated by x and z, with the relations $x^3 = 1$, $z^3 = 1$, xz = zx. In this case, the result is a total morphism on A.

G_1	1	z	z^2
1	1	z	z^2
x	x	x z	$x z^2$
x^2	x	x^2 z	$x^2 z^2$

As this group is abelian, the set of actions of the Towers of Hanoi then decomposes in two ways:

- executing the z macro the necessary number of times to solve the large disk, then
- executing "x" the necessary number of times to solve the small disk;

or :

- executing "x" the necessary number of times to solve the small disk, then
- executing the z macro the necessary number of times to solve the large disk.

These decompositions do not lead to optimal solutions (they can be improved by including both right and left moves for both disks); however, they possess the usual advantage of task decompositions: they clarify and simplify the task, leading to reduced sensing and planning time. The partiality of the actions in TOH1 is *encapsulated within macros* in this new representation, thereby eliminating subgoal interference by moving the constraints to the generator.

Reformulating TOH2

Consider TOH2. The groups containing the primitive idempotents are all trivial. In this case, a reformulation algorithm is:

- Compute Green's relations.
- Find the primitive idempotents.
- Find a minimal word $x_1x_2x_3...x_n$ for one of the primitive idempotents.
- Map the set of functions $x_1x_2x_3...x_nx_1$, $x_2x_3x_4...x_nx_1x_2$, etc. to one new symbol.

All the primitive idempotents are mapped to the identity function. All primitive idempotents can be found by cyclically permuting a minimal word for a primitive idempotent. Also, this word gives a cycle of Q (executing the actions of the word visits each state of Q exactly once). We restrict these nine functions to single states by multiplying on the left by the appropriate primitive idempotents, and then map these nine functions to one new symbol v, giving a cycle that visits each element of Q exactly once, so that each v is a counterclockwise arrow around the state graph for the 2-disk Towers of Hanoi.

Notice that three of the elements of the original semigroup are not mapped; they are not necessary for reachability, but only for efficiency. This gives a cyclic group G_2 of order 9 generated by v, which decomposes into two cyclic groups of order three:

G_2	1	·	v^6
1	1	v^3	v^6
v	v	v^4	v^7
v^2	v^2	v^5	v^8

Once again, these actions are totally defined, so subgoal interference has been eliminated and constraints have been hidden by encapsulating them in macros.

This is isomorphic to the group found in the previous example from TOH2, with $z = v^3$, and $x = v^2$. But this group representation is not related to the group representation from TOH1 by a homomorphism of transformation monoids That this is so is evident from the way the two groups map the states. Group G_1 maps state 1 into state 3 via action x^2, but G_2 maps state 1 into state 4 via action v^2. This shows that non-homomorphic transformation groups can exist in the category of representations for a task. Although these two groups are isomorphic as abstract groups, they possess different computational properties when acting on the states of the task, e.g., the average path length between any two states in G_1 is shorter than in G_2. The morphisms of transformation monoids distinguish properly between these two representations, thus illustrating the usefulness of the formalism for reasoning about representations.

Summary

We have described a research program pursuing an algebraic approach to reasoning about representation change. There are three advantages to this approach. First, it ties in to an existing theory of semigroups that is general and intuitive. We hope that this paper has demonstrated the intuitive advantages of this approach, particularly in the use of coordinate systems to characterize reformulation.

Second, we can use this theory for classification. We classify representations by the structure of their transformation monoids, and classify tasks according to the representations they admit. We can also classify representation changes. For example, we have classified reformulations as coordinate transformations if they transform the presentation of the transformation monoid, and as deformations, if they transform the structure of the monoid.

Third, we can use this theory to construct algorithms for representation change. For example, we showed how to use group representation theory to automatically abstract a group representation, and we showed how to move constraints from the tester to the generator for inverse semigroups.

This paper has dealt with the semantics of representation change, as embodied in the structure of semigroups of actions. Part 2 will deal with the agent-dependent features used to encode states and actions, which are embodied in strings of symbols over alphabets.

Acknowledgments

This work has benefited greatly from discussions with Ranan Banerji, Leo Dorst, Jonathan Hodgson, Indur Mandhyan, and Madeleine Rosar. Leo and Madeleine did much of the work on Rubik's Cube.

References

Amarel, Saul, (1968). On Representations of Problems of Reasoning about Actions, in Michie (ed.) *Machine Intelligence*, chapter 10, pp. 131-171, Edinburgh University Press.

Arbib, Michael A., and Manes, Ernest G., (1974). Machines in a Category: An Expository Introduction, SIAM Review, Vol.16, No.2, pp.163-192, April, 1974.

Banerji, Ranan B. and Ernst, George W., (1977). A Theory for the Complete Mechanization of a GPS-type Problem Solver, IJCAI-77, pp.450-456.

Benjamin, D. Paul, Dorst, Leo, Mandhyan, Indur, and Rosar, Madeleine, (1990). An Introduction to the Decomposition of Task Representations in Autonomous Systems, in "Change of Representation and Inductive Bias", D. Paul Benjamin (ed.), Kluwer Academic Publishers.

Bobrow, Leonard S., and Arbib, Michael A., (1974). Discrete Mathematics, Saunders.

Chomsky, N., (1957). Syntactic Structures, Mouton, The Hague.

Dorst, Leo, (1989). Representations and Algorithms for the 2x2x2 Rubik's Cube, Philips Technical Report TR-89-041.

Doyle, Jon, and Patil, Ramesh S., (1991). Two theses of knowledge representation: language restrictions, taxonomic classification, and the utility of representation services, Artificial Intelligence **48**, pp. 261-297.

Eilenberg, Samuel, (1974). Automata, Languages, and Machines, Volumes A&B, Academic Press.

Howie, J. M., (1976). An Introduction to Semigroup Theory, Academic Press.

Knoblock, Craig A., Tenenberg, Josh D., and ßng, Qiang, (1991). Characterizing Abstraction Hierarchies for Planning, AAAI-91.

Korf, Richard E., (1983). Learning to Solve Problems by Searching for Macro-Operators, Ph.D. Thesis, Carnegie-Mellon University.

Lallement, Gerard (1979). Semigroups and Combinatorial Applications, Wiley & Sons.

Lowry, Michael, (1990). Homomorphic Reformulation, Proceedings of the Second International Workshop on Problem Reformulation, Price Waterhouse, Palo Alto, California.

Lowry, Michael, (1987). Algorithm Synthesis Through Problem Reformulation, AAAI-87.

Niizuma, S. and Kitahashi, T., (1985). A Problem-Decomposition Method Using Differences or Equivalence Relations between States, Artificial Intelligence **25**, pp.117-151.

Riddle, Patricia J., (1986). Exploring Shifts of Representation, in Mitchell, Carbonell, and Michalski (eds.), Machine Learning: A Guide to Current Research, Kluwer.

Sacerdoti, E., (1974). Planning in a Hierarchy of Abstraction Spaces, Artificial Intelligence 5(2), pp.115-135.

Simon, H.A., (1969). The Sciences of the Artificial, MIT Press, Cambridge, Mass.

Subramanian, Devika, and Genesereth, M.R., (1987). The Relevance of Irrelevance, IJCAI-87, pp.416-422.

Unruh, Amy, and Rosenbloom, Paul S., (1989). Abstraction in problem solving and learning, IJCAI-89, pp.681-687.

Zimmer, Robert M., (1990). Representation Engineering and Category Theory, in *Change of Representation and Inductive Bias*, D. Paul Benjamin (ed.), Kluwer Academic Publishers.

53-63

Specification Reformulation During Specification Validation

Kevin M. Benner
USC / Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
(310) 822-1511
Benner@isi.edu

Abstract

The goal of the ARIES Simulation Component (ASC) is to uncover behavioral errors by "running" a specification at the earliest possible points during the specification development process. The problems to be overcome are the obvious ones – the specification may be large, incomplete, underconstrained, and/or uncompilable. This paper describes how specification reformulation is used to mitigate these problems. ASC begins by decomposing validation into specific validation questions. Next, the specification is reformulated to abstract out all those features unrelated to the identified validation question thus creating a new specialized specification. ASC relies on a precise statement of the validation question and a careful application of transformations so as to preserve the essential specification semantics in the resulting specialized specification. This technique is a win if the resulting specialized specification is small enough so the user may easily handle any remaining obstacles to execution. This paper will (1) describe what a validation question is, (2) outline analysis techniques for identifying what concepts are and are not relevant to a validation question, and (3) identify and apply transformations which remove these less relevant concepts while preserving those which are relevant.

Introduction

Validation at the requirements level is often characterized as validation with respect to the client's or stakeholder's intent. The goal of specification validation is to identify those aspects of the specification which do not conform to the client's intent and then to make appropriate changes. More practically, this boils down to uncovering bugs in the specification and fixing them. The goal of this work is to address a specific subclass of specification errors that have not previously been satisfactorily addressed. In particular, this work addresses identifying errors in the dynamic behavior of a high-level specification. This work will distinguish itself from related works by being able to handle large, very-high-level specifications. This is done by making explicit specific validation questions which focus validation activities sufficiently enough so that traditional validation techniques, like simulation and direct execution are tractable.

The problem of identifying errors in the specification and the cost of finding these later during the development process is well documented [Boehm, 1981]. Among these errors, the most difficult to identify early on are those which concern behavior. In general these include: (1) inconsistency between specification components, (2) incompleteness with regard to known scenarios, and (3) inconsistency between requirements and their realization in the specification.

The work describe herein will uncover behavioral errors by "running" a specification at the earliest possible points during the specification development process. The problems to be addressed are the obvious ones – the specification may be large, incomplete, underconstrained, and uncompilable. These problems are addressed via a four step process. First, the validation activity is decomposed into specific validation questions. Second, the specification is reformulated to abstract out all those features unrelated to the identified validation question thus creating a new specialized specification. Third, the specialized specification is executed with the purpose of proving or disproving the validation question. And finally fourth, since the specialized specification was constructed in a disciplined manner, one may now infer the result of the validation question about the original specification.

The general feel of the interaction is more like a debugging sessions, particularly early in the development. The goal is to get something running quickly and easily so as to reveal behaviors implied by the specification and make them accessible to end users and stake holders for early validation (or more likely, early error identification). During a typical validation session, the specialized specification and its validation question are executed. The simulation system using the validation question will guide the execution toward satisfaction of the validation question. When this is

not possible the simulator will point out how the validation question has been violated. The stake holder and analyst will observe the execution. When a validation question is not satisfiable, the analyst will be able to explore the behavior space to understand why this is the case. Appropriate changes may then be made to the specification and the specialized specification construction process replayed. This is then followed by re-execution of the specialized specification. Naturally this process may be repeated.

The abstraction or reformulation process employed during specialized specification construction is the heart of the ARIES Simulation Component(ASC, pronounced "ask"). It relies on a precise statement of the validation question and a careful application of transformations so as to preserve the essential specification semantics in the resulting specialized specification. This technique is a win if the resulting specialized specification is small enough so that the user may easily handle any remaining obstacles to execution. This paper will (1) describe what a validation question is, (2) outline analysis techniques for identifying what concepts are and are not relevant to a validation question, and (3) identify and apply transformations which remove these less relevant concepts while preserving those which are relevant.

The work described in this paper is a component of a larger effort called ARIES [Johnson *et al.*, 1991] which is concerned with the overall task of requirements acquisition and specification development and validation. Requirements may be stated informally and then gradually formalized and elaborated. Validation is facilitated via a variety of graphical and textual presentations. Elaboration and refinement are supported via evolution transformations. Additionally, mechanisms for reuse and concept encapsulation have been provided.

The example used throughout this paper is drawn from the air traffic control domain, specifically behaviors concerning *handoff* — passing control of an aircraft from one air traffic controller to another. Some of the concepts included in the full specification are: control, physical location, sensors, tracks, maintaining tracks, flight plans, aircraft movement, agents within the air traffic control domain, etcetera.

Validation Questions

At the beginning of the requirements acquisition process, many requirements are not easily expressed as abstract, concise, declarative statements of stake-holder needs. Rather at this point, requirements are often more easily expressed informally as a mix of situations and experiences which the stake-holder wishes to have handled by the system to be specified.

Informally, a validation question is any question a user or stake-holder may have about the specified system. It could encompass anything he/she believes to be pertinent. The goal of a validation question is to provide a means to ask these questions. Fundamentally, validation questions are statements of user's or stake-holder's requirements. They are stated in a manner as similar as possible to the way they are manifest in the user's or stake-holder's real world. And they hopefully have little dependence on how these requirements may be realized in the specification. This section will show how these goals are attained by allowing the stake-holder to express his/her requirements via the following constructs: scenarios, to describe partial orderings of states and events using both abstract and concrete concepts, and assumptions, to support the implicit assumptions common in natural language and often used when stake-holders describe their needs.

Consider the following natural language questions. They are the intuitive basis from which we will evolve formal validation questions.

- VQ-1: Does handoff occur before the aircraft moves from its current airspace to an adjacent airspace?
- VQ-2: Once in-route to a particular location, is control maintained throughout?
- VQ-3: Will the system recognize when an aircraft is out of conformance with its flight plan?

Figure 1: Some Informal Validation Questions

The above questions illustrate several important characteristics of validation questions. First, validation questions often implicitly rely on scenarios and assumptions to provide a narrowed context. Second, validation questions often use concrete and/or qualitative instances to focus on specific, relevant attributes of the specified system. And finally third, validation questions often embody some expected interaction that the analyst is trying to stress. The remainder of this section will describe how validation questions are described in terms of scenarios and assumptions.

Scenarios

Scenarios are a partial ordering of events and/or states. They allow one to describe a complex sequence of activities at an arbitrary level of detail without necessarily making commitments regarding their causal relationships.

This section will define the semantics of a scenario with respect to state transition diagrams[1]. The specific semantics is determined by the scenario mode. Alternatives include comparative, restrictive, or prescriptive. The mode is selected by the analyst during formulation of the validation question. Each mode constrains the behavior space of the specification in a progressively more restrictive manner.

[1]Other notations for scenarios are also available and are often used. They are isomorphic with STDs.

- **Comparative** has no effect on the behavior space of the simulated specification, but acts as a watch dog informing the user as to the satisfaction or partial satisfaction[2] of a scenario during simulation. In this case satisfaction of a node determines the current node. The current node is neither necessary nor sufficient to advance to the next node.
- **Restrictive** means that nondeterminism within the behavior space is pruned so that if the scenario may be satisfied it will be. Basically, satisfaction of each node is necessary but not sufficient to advance to a following node. Informally this mode is best described as a procedural invariant.
- **Prescriptive** means that the simulator advances the scenario from one node to the next irrespective of the state of the simulation. More formally, the simulator treats the satisfaction of a node as necessary and sufficient for advancing to the next node. Operationally, advancing to a transition node implies that the corresponding event is invoked. When the event is completed the following state node is made true.

Main validation question The validation question labeled as *VQ-1* in figure 1 is formalized as the state transition diagram shown in figure 2. States *within-s1* and *within-s2* are the qualitative values for the aircraft being in sectors *s1* and *s2* respectively. Transitions *handoff* and *alert-controller* represent the events of the same name and the following states represent completion of these same events. Transitions *m-1* and *m-2* represent any event that would result in the final state *within-s2*. During simulation, ASC will drive this state transition diagram to reflect the state of the simulation. If an illegal transition occurs the analyst will be informed.

The goal here has been to express typical, critical situations that the user wants to be sure are handled in a specific way. This could have been done in terms of concept at any level of abstraction. Typically as the specification moves closer to completion, the validation question may become more complex and may be expressed in terms of lower-level concepts.

Driving scenarios Driving scenarios are prescriptive scenarios, typically used to model actions outside the scope of the current specialized specification. Figure 3 shows two driving scenarios that are used within this example. The driving scenario commits to manage specific concepts. In this case those concepts are *track-position* and the qualitative values derived from *track-position* (e.g., *top-of-block-altitude* and *within-accept-handoff-computed-point-distance*). When a driving scenario manages a concept it supersedes all other specification concepts which attempt to influence the same concept. This information will be used later during specification reformulation.

[2]Satisfaction of a state means the predicate associated with the state is true. Satisfaction of a transition means the event associated with the transition has been invoked. Satisfaction of a scenario means that the states and transitions of the scenario have been satisfied in the order specified by the scenario.

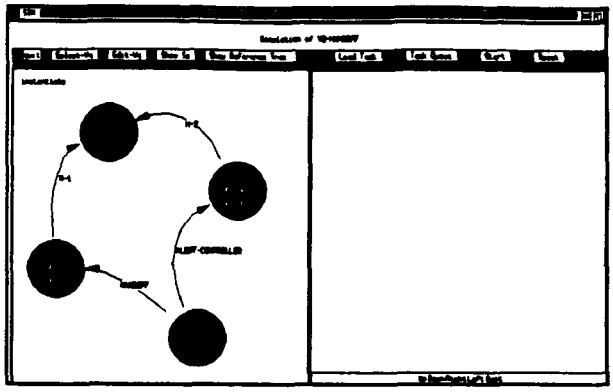


Figure 2: Validation Question – *VQ-Handoff*

```
procedure DS-LEVEL-3()
    := steps(
          insert within-s1 ac1;
          insert top-of-block-altitude ac1;
          delay();
          insert enter-new-airspace ac1)

procedure DS-LEVEL-4()
    := steps(
          insert within-s1 ac1;
          insert top-of-block-altitude ac1;
          delay();
          insert within-accept-handoff-computed-point-distance ac1;
          insert within-s2 ac1)
```

Figure 3: Driving Scenarios for *VQ-Handoff*

The level of abstraction at which driving scenarios operate is one of the primary influences on the level at which simulation will be done. Instead of driving *track-position*, the analyst could decide to drive *sensor-reports*. This would result in a larger, more detailed specialized specification which would include processing of *sensor-reports* into *tracks*.

Scenarios to constrain nondeterminism Specifications are highly underconstrained, particularly early in their development. ASC allows one to execute a specification in spite of this by providing various mechanism to constrain the nondeterminism within the context of a validation question. One of these mechanisms is s restrictive scenario which acts as procedural constraint on the behavior space.

One example of this is the *Handoff* transition in figure 2. *Handoff* is not actually an event but rather a *restrictive* scenario which is constraining the simulation to only consider handoffs consisting of an initiate and accept phase (see figure 4). This scenario precludes handoffs from being canceled or rejected. More

complex scenarios would deal with this after first performing validation on this simpler case.

```
scenario handoff(ac:track)
   := steps[
        automatic-init-handoff(ac);
        accept-handoff(ac, any controller, any controller)]
```

Figure 4: Restrictive Scenario

Assumptions

Assumptions allow the analyst to codify what are often implicit assumptions made by developers as they build rapid prototypes. The advantage of this approach is that it documents said assumptions. Once recorded, the analyst can now separately validated the assumptions with stake-holders within the context of the current validation question. This way the analyst can be sure that assumptions do not trivialize the basic intent of the validation question. A later section of this paper will show how assumptions are used during specification reformulation.

```
invariant FIXED-SET-OF-TRACKS
   for-all (t1:track) element-of(t1, {ac1, ac2})
```

Figure 5: Assumption for *VQ-Handoff*

Assumptions are expressed as invariants. Figure 5 contains one of the assumptions used in the current example. Since we are not validating *track-processing*, we can relax some of the constraints on tracks and for now constrain the number of tracks in the simulation model. *Ac1* and *ac2* are defined as tracks in an unshown initialization scenario.

Influence Analysis

The previous section described how a validation question is formalized. This section will show how the validation question is used to reformulate the current specification into a specialized specification which is simulatable. We begin with influence analysis.

Brooks in [Brooks, 1986] warns that descriptions of software that abstract away its complexity often abstract away its essence. Influence analysis is a means of allowing the analyst to see through this complexity to distinguish between concepts which are most relevant to the validation question and those which are not. Once identified, ASC provides reformulation transformations which remove those concepts which are not relevant.

In general, formally showing whether or not one concept effects another is undecidable. Even at an informal level, causality is a vary hard problem. Influences finesse this issue by relying on rules which are easily computable and which generate all potential influences rather than making claims about actual influences. As such, the resulting influence graph should be considered a conservative representation of concept influences – that is they may indicate influences which are not actually possible, but are safe, in that they will not fail to indicate the presence of an influence that does exist.

Once the initial influence graph is generated, more knowledge intensive approaches are applied to remove many of those potential influences which are not actual influences.

Another problem in extracting the influence graph is that influence paths could be through arbitrarily many intermediaries and in an incomplete specification would be inherently suspect. Rather than deal with this problem, ASC allows the analyst to limit the path length it will look at during analysis. Granted this has the horizon effect, but this can be minimized. (see section Horizon Effect Addressed)

Influence Definition

An influence identifies the conduits through which one concept effects another during execution. ASC divides these conduits up into three classes: information, control, and miscellaneous. This section will informally characterize each of these classes and then illustrate them via an example. ASC has operationally formalized these concepts based on the specification language Reusable Gist [Johnson and Feather, 1991].

- **Information influences** are concerned with the flow of information between concepts. Stated another way, when information changes, how does it percolate through the system? Some examples of these types of influences are: database updates, assignment statements, and definitional use of data declarations (i.e., relations, types, and instances) by other data declarations.
- **Control influences** are concerned with *if* and *when* behaviors may occur. Some examples of this class of influences are preconditions on an event, invocation of an event, invariants, and conditional statements (Note, granularity of influences is at the level of declarations. Thus influences on or by statements are reflected as influences on or by the event which contains the statement).
- **Miscellaneous influences** are concerned with influences on and by the the validation question. Most influences are *onto* validation questions with driving scenarios being the exception.

Figure 6 is the Reusable Gist definition of the event *accept-handoff*. Figure 7 shows a paraphrase of this event. The resulting primitive influence graph is shown in figure 8.

```
Demon ACCEPT-HANDOFF(track,
                     current-controller:controller,
                     receiving-controller:controller)
    precondition controlled(track, current-controller) and
                 handoff-in-progress(track,
                                     current-controller,
                                     receiving-controller)
    postcondition controlled(track, receiving-controller)
    := steps(track.controlled ← receiving-controller;
             remove handoff-in-progress(track,
                                        current-controller,
                                        receiving-controller);
             track.track-status ← 'normal)
```

Figure 6: Reusable Gist definition of the event *Accept-Handoff*

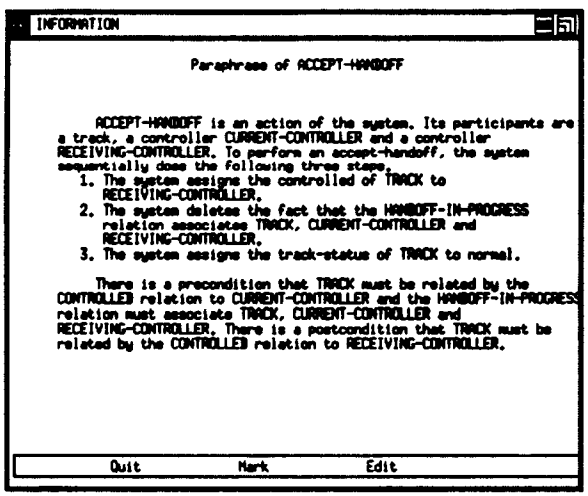


Figure 7: Paraphrase of the event *Accept-Handoff*

The influence graph of figure 8 is most similar to de Kleer's mechanism graph from his work in qualitative reasoning [de Kleer, 1986; de Kleer and Brown, 1986]. The mechanism graph shows the causal influences between concepts. A vertex contains an information value which represents a specific circuit component attribute (e.g. the voltage or current at a given component). Edges represent how a change in a vertex value is propagated to adjacent vertices. Edges are derived from either component models or domain specific heuristics. In influence graphs, a vertex represents a specification concept declaration (or fragment). Edges represent how a concept influences either the behavior or value of another concept.

The influence graph (figure 8) shows most of the immediate influences on and by *accept-handoff*. *Receiving-controller*, *current-controller*, and *track* are parameters of the event. *Enabling-pred-of-accept-handoff* is a composite node representing the precondition of the event. *Controlled*, *and*, and *handoff-in-progress* are relation referenced by the event. Edges within the graph represent the direction in which influences are propagated. Note that *how* each influence effects a given node is not represented. This is in fact outside the capability of influence analysis in ASC. None the less, this still provides a great deal of information to the analyst when creating a specialized specification as we will see later.

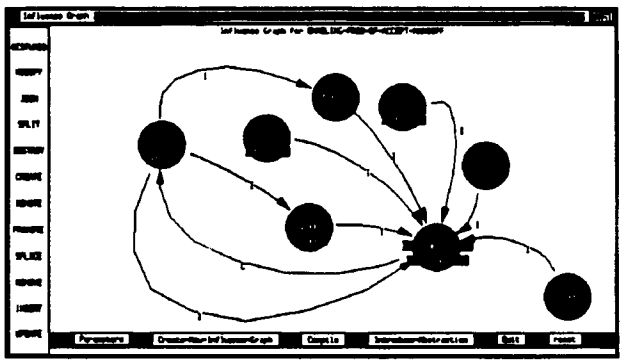


Figure 8: Primitive Influence Graph of the event *accept-handoff*

Automated Graph Abstraction

Though the graph in figure 8 could be used as is, it shows many influences which really do not drive the dynamic behavior of the specification. This section will describe some of the influence abstraction rules which are applied automatically by ASC. Figure 9 shows the resulting influence graph. It is this graph, not the previous one, which the analyst is first shown after influence analysis.

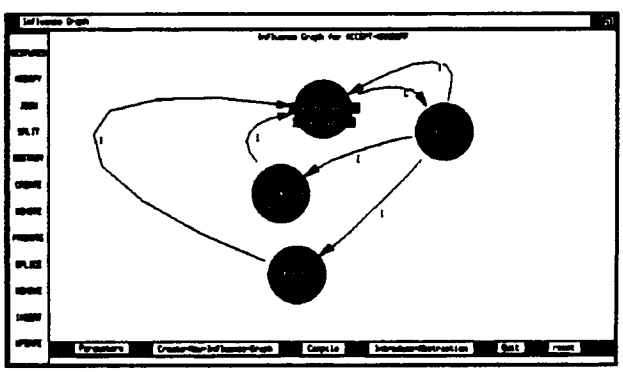


Figure 9: Influence Graph of the event *accept-handoff*

Below are some of the abstractions which are automatically applied during influence analysis.

- Remove influences on self.
- Remove influences from concepts in the Predefined folder (i.e., commonly used relations, e.g., *and*).
- Remove *static* concepts which have no influences on them[3].
- Remove variables and parameters which are not explicitly influenced.[4]

Interactive Graph Abstraction

Not all abstractions can be done in an automatic fashion. Typically, the presence of certain influences indicate either an error in the specification or an opportunity to apply an abstraction. The analyst must make these decisions. ASC identifies these cases during automatic influence abstraction and then posts notifications via an agenda mechanism. When the analyst is ready, he/she may view the agenda and the alternative actions recommended by ASC. Recommendations typically include suggested transformations which can cause the desired effect in either the evolving specialized specification or the underlying specification. Some of these interactive suggestions are:

- When there are no influences on a type or relation declaration, suggest that the concept should be declared static.
- When there is an influence on a type or relation declaration, suggest that the concept should be changed to dynamic (e.g., explicit or derived relation).
- When an influence node has only a single input information influence and only a single output information influence, suggest that the intermediate node be abstracted out and the input and output nodes be modified to be a direct influence.

For validation question *VQ-Handoff* of figure 2, influence analysis results in 224 influence nodes. After automatic abstraction this count is reduced to 97 influence nodes with 51 posted suggestions (most of which concern suggestions on declaring concepts as dynamic or static). After the analyst handles the most obvious suggestions, the influence node count is reduced to 77. Though an improvement over the starting point of 224 concepts, there are still a lot of concepts to compile for simulation.

Specification Reformulation

The previous section's analysis and reformulation were basically independent of the validation question. This section will suggest more drastic reformulations which take advantage of the knowledge implicit in the validation question.

Modification, whether motivated by errors discovered in the specification or by simplifying assumptions, are accomplished via the application of transformations. Since ASC is a component of ARIES, it is able to take advantage of an extensive library of evolution transformations [Johnson and Feather, 1990]. These transformations formally evolve a specification based on specific desired effects.

An important feature of the reformulation process is that not all the effort to build the specialized specification need be thrown away after doing validation. Many of the applied transformations are equally valid in both the specialized specification and the original specification. ASC allows the analyst to declare during reformulation if a transformation should be recorded and later applied to the original specification. This selective record of transformations provides an opportunity for these transformations to be replayed on the original specification. (This selective replay capability has not yet been implemented in ASC.)

Reformulations Motivated by the Validation Question

Reformulation based on a validation question is analogous to how partial evaluation (mixed computation) [Ershov, 1985] is able to generate an efficient residual program based on a more general program and a subset of its input parameters. This technique is potentially more powerful because a validation question is a richer source of knowledge than just a list of input parameters.

Reformulation based on assumptions We begin with *VQ-Handoff*'s assumption (see figure 5) that there will be a fixed number of tracks which already exist. Figure 10 shows the influence graph of *fixed-set-of-tracks* and all of the concepts in the specification it directly influences, i.e., *track*, *initiate-tracking*, and *extract-track-info*.

The analyst begins with the event *initiate-tracking*. Influence analysis shows there are no other influences on it. Visual inspection reveals to the analyst that it creates tracks. Since the assumption says there will be no new tracks, this event is superfluous and thus should be abstracted out.[5]

The analyst next handles the type *track*. The implication of the assumption is obvious. The type should be declared static. Note that this is not generally true within the air traffic control domain, but illustrates a common result of validation question assumptions.

The third influenced node is the event *extract-track-info*. The analyst attempts to handle it as he/she

[3] If there is an influence on a static concept post it as an error.

[4] Explicit information influence are various forms of assignment. Event parameters are almost always removed since the dominating influence is the control influence on the event (e.g., who the caller is).

[5] A theorem prover would be very useful here. Given the narrowed context, it might be tractable to prove the above conclusion automatically. This is outside the scope of ASC.

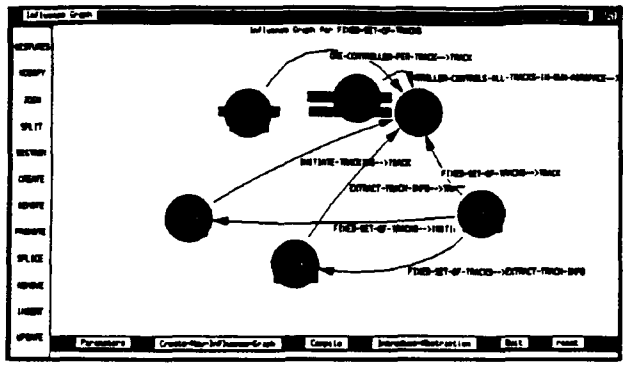


Figure 10: Influence Graph of the assumption *fixed-set-of-tracks*

handled *initiate-tracking*. In this case, visual inspection of the event shows that this event both creates *tracks* and assigns them a *track-position*. At this point the analyst needs to determine if he/she will specialize *extract-track-info* into a new event which only deals with *track-position* or if the event may be abstracted away completely. The analyst then displays a new influence graph as shown in figure 11. This influence graph shows that *extract-track-info* influences both *track* and *track-position*. Additionally, *track-position* is influenced by the driving scenario *ds-level-3*. At this point, remembering that driving scenarios have taken responsibility to be the sole maintainer of the concepts they influence, the analyst now removes *extract-track-info* from the specialized specification.

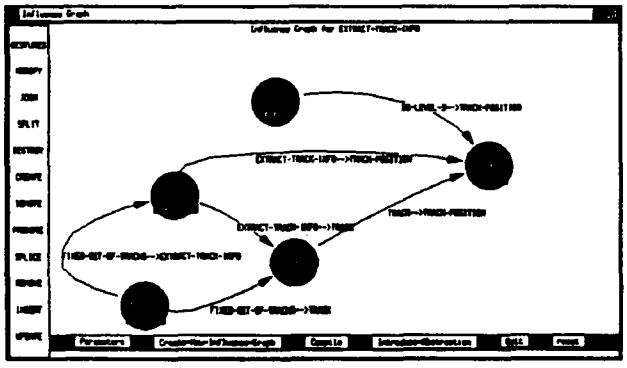


Figure 11: Influence Graph of the event *extract-track-info*

Reformulation based on driving Scenarios In the current example, influence analysis shows that the *next-controller* relation used by *automatic-init-handoff* relies on *paired* and *flight-plan* neither of which is fully defined. Additional analysis shows that *flight-plan* influences only a few other concepts including *conformance*. The analyst decides to abstract out *flight-plan* and *paired* and then model *next-controller* and *conformance* without them.

Two approaches are possible. One is to redefine *next-controller* such that it can be derived from concepts already within the specialize specification or to directly maintain the relation. In the spirit of picking the approach which is quick (and hopefully not too dirty), the latter is chosen. This is easy to do within the context of the validation question. The analyst simply includes an assertion as part of the driving scenario that *next-controller(ac-1, c1)*.

More reformulation based on assumptions *Conformance* is handled slightly differently. Since *VQ-Handoff* deals with handoffs which are initiated automatically, the analyst can take advantage of this specialized case knowledge and assume *conformance* is always true. To assume otherwise would imply one is not in an automatic handoff situation and thus would be outside the scope of *VQ-Handoff*.

Note that it might be tempting to abstract away *conformance* by assuming *inhibited-handoff* is false, but this would fail because *inhibited-handoff* also influences other events which are directly involved in the validation question (see figure 12). Such assumptions which influence system behavior with respect to the validation question are not acceptable.

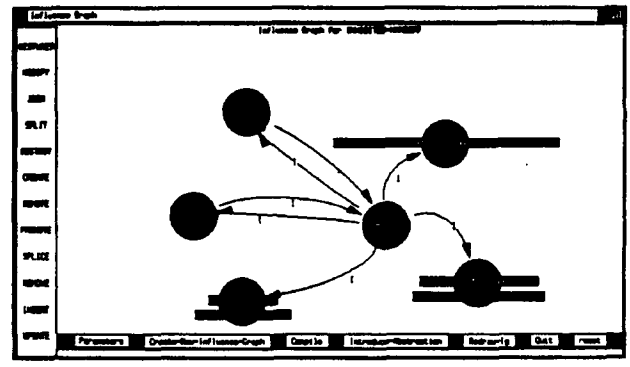


Figure 12: Influence Graph of the relation *Inhibited-Handoff*

Another use for assumptions is to decompose validation questions into smaller more manageable pieces similar to how assumptions are used in proofs to break them up into multiple, hopefully more manageable, pieces. In the case of automatic verses manual initiation of handoff, a well chosen assumption creates two distinct scenarios which are then handled separately.

Reformulation to introduce qualitative abstractions An earlier version of the validation question *VQ-Handoff* was expressed in terms of *track-position*. Likewise, many event preconditions were expressed in terms of *track-position*. Rather than describe scenar-

ios in terms of *track-positions*, the analyst decided to reformulate the specification to introduce qualitative abstractions. ASC facilitated this by showing what concepts where influenced by *track-positions*. The analyst was then able to use ARIES transformations to replace complex predicates about *track-position* with new predicates expressed in terms of newly defined relations. These new relations were *top-of-block-altitude*, *within-handoff-computed-point*, *within-accept-handoff-computed-point-distance*, and *within-accept-handoff-computed-point-time*. Each is a specializations of *track-position*. This now allowed the analyst to easily describe validation questions and scenarios in terms of these qualitative states rather than in terms of *track-position* which has many more states but which fall into one of these five qualitative states.

Reformulating scenarios as run-time constraints Not all optimizations are realizable during specialized specification construction. This problem is pointed out by Meyer in [Meyer, 1991] when applying partial evaluation to imperative languages. The problem is that compile time execution can result in side-effects which are not noticed at the appropriate time. This is because the side-effect could happen during specialized specification construction and not during simulation. The problem with this is that other parts of the specification which trigger on the side-effects of the partially evaluated scenario will now not have those side-effects to react to at run-time. As a result partial evaluation at specialized specification construction time must be constrained not do anything that causes triggering states to disappear.

In *VQ-Handoff*, the handoff scenario includes only *automatic-init-handoff* and *accept-handoff*. It excludes several other events that could be a part of a typical handoff scenario (e.g., *manual-init-handoff*, *reject-handoff*, and *cancel-handoff*). ASC translates these scenarios into a procedural invariant which ensures the appropriate behavior at run-time. The advantage of such a constraint is that one is not forced to deal with control issues regard the full set of events until the pairwise (in this case *automatic-init-handoff* and *accept-handoff*) interaction has first been resolved.

Horizon Effect Addressed

ASC mitigates the horizon effect by trying to create a specialized specification which defines a closed simulation model. In such a model there are no outside influences and all influence paths within the closed model are known, thus there is no horizon effect.

With respect to the current example, reformulation continues until a closed model is achieved. The final specialized specification contains 44 influence nodes.

Summary

The success of the ARIES Simulation Component may be measured with respect to the following criteria.

- ability to execute a specification where previously it could not be done.
- ability to execute a specification with less effort than was required before.
- ability to document requirements satisfaction.
- ability to make validation comprehensible to stakeholders.
- ability to provide a flexible approach for system validation.

At this point its too early to address most of these issues. I have only applied ASC to a few validation questions, all within the domain of ATC handoffs. The most quantifiable results to date concern the number of concepts involved in the specification of *VQ-Handoff*, figure 13.

	number of concepts
ATC Knowledge Base	1,400
Primitive Influence Graph	224
Initial Influence Graph	97
Final Specialized Specification	44

Figure 13: Number of Concept Declarations for Validation Question *vq-handoff*

This illustrates that only a fraction of the total number of possible concepts were actually needed to achieve executability. Additionally, the focus provided by the validation question provided direction on which concepts to flesh out next in order to achieve closure and executability. Granted these techniques could and have to some degree been applied by hand when one builds a rapid prototype. The difference is that ASC generates both a rapid prototype and a formal characterization of how it relates to the original specification.

As a sidebar, in the process of constructing the specialized specification I discovered several errors. Many of these were in fact errors in the specification which I believed was essentially correct. This seems good, in that error discovery is an important precursor to validation.

References

Boehm, B. 1981. *Software Engineering Economics.* Prentice Hall.

Johnson, W.L.; Feather, M.S.; and Harris, D.R. 1991. The KBSA requirements/specifications facet: ARIES. In *Proceedings of the 6th Knowledge-Based Software Engineering Conference.* to appear in *IEEE Expert.*

Brooks, F. P. 1986. No silver bullet: Essence and accidents of software engineering. *Computer* 20(4):10–19.

Johnson, W.L. and Feather, M.S. 1991. Reusable gist language description. Available from USC / ISI.

Kleer, J.de 1986. How circuits work. In *Qualitative Reasoning About Physical Systems.* MIT Press. 205–280.

Kleer, J.de and Brown, J. S. 1986. Qualitative physics based on confluences. In *Qualitative Reasoning About Physical Systems.* MIT Press. 7–83.

Johnson, W.L. and Feather, M.S. 1990. Using evolution transformations to construct specifications. In *Automating Software Design.* AAAI Press.

Ershov, A. P. 1985. On mixed computation: Informal account of the strict and polyvariant computation shemes. In *NATO ASI Series, Vol. F14 - Control Flow and Data Flow: Concepts of Distributed Programming.* Springer-Verlag. 107–120.

Meyer, U. 1991. Techniques for partial evaluation of imperative languages. In *Proceedings of Sympsium on Partial Evaluation and Semantic-Based Program Manipulation*, Yale Univ., New Haven, CT. 94–105.

Machine Learning Techniques in Optimal Design

Giuseppe Cerbone
Oregon State University
Computer Science Dept.
Corvallis, OR 97331 (USA)

Introduction

Many important applications can be formalized as constrained optimization tasks. For example, we are studying the engineering domain of two-dimensional (2-D) structural design. In this task, the goal is to design a structure of minimum weight that bears a set of loads.

Figure 1 shows a solution to a design problem in which there is a single load (L) and two stationary support points (S1 and S2). The solution consists of four members, E1, E2, E3, and E4 that connect the load to the support points. In principle, optimal solutions to problems of this kind can be found by numerical optimization techniques. However, in practice [Vanderplaats, 1984] these methods are slow and they can produce different local solutions whose quality (ratio to the global optimum) varies with the choice of starting points. Hence, their applicability to real-world problems is severely restricted.

To overcome these limitations, we propose to augment numerical optimization by first performing a symbolic compilation stage to produce (a) objective functions that are faster to evaluate and that depend less on the choice of the starting point and (b) selection rules that associate problem instances to a set of recommended solutions. These goals are accomplished by successive specializations of the problem class and of the associated objective functions. In the end, this process reduces the problem to a collection of independent functions that are fast to evaluate, that can be differentiated symbolically, and that represent smaller regions of the overall search space. However, the specialization process can produce a large number of sub-problems. This is overcome by deriving inductively selection rules which associate problems to small sets of specialized independent sub-problems. Each set of candidate solutions is chosen to minimize a cost function which expresses the tradeoff between the quality of the solution that can be obtained from the sub-problem and the time it takes to produce it. The overall solution to the problem, is then obtained by solving in parallel each of the sub-problems in the set and computing the one with the minimum cost.

In addition to speeding up the optimization process, our use of learning methods also relieves the expert from the burden of identifying rules that exactly pinpoint optimal candidate sub-problems. In real engineering tasks it is usually too costly to the engineers to derive such rules. Therefore, this paper also contributes to a further step towards the solution of the knowledge acquisition bottleneck [Feigenbaum, 1977] which has somewhat impaired the construction of rule-based expert systems.

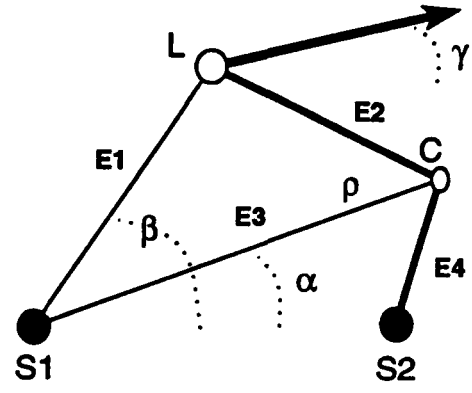


Figure 1: A solution to a 2-D structural design problem with given topology.

Our optimization schema differs from techniques currently used in the machine learning community. Our approach relies on the specialization of the problem via incorporation of constraints prior to optimization. Braudaway [Braudaway, 1988] designed a system along the same principle. However, to our knowledge, very little work has been done in using learning techniques to speedup numerical optimization tasks. In contrast, the current trend in the machine learning community focuses on methods, such as Explanation Based Learning (EBL) [Ellman, 1989], capable of generating rules. In addition, EBL methods have had little

success in the task of optimizing numerical procedures. We conjecture that one of the reasons is the dependence of EBL methods on the trace of the problem solver. The trace of a numerical optimizer gives little information on the structure of the problem. Therefore, in mathematical domains, EBL-derived rules are too detailed to produce any appreciable speedup.

The remainder of the paper is organized as follows. Section presents the 2-D structural design task. This is followed in Section by an overview of numerical optimization methods, their limitations, and our solution which is illustrated using a simple example. The machine learning methods are outlined in Section . These methods are then applied in Section which illustrates the experiments. These show that, for a certain family of problems, the compilation stage produces a substantial improvement in the performance of the optimization methods. Benefits and limitations of our strategy are summarized in Section , which also outlines future work.

Task description

Table 1 describes the 2-dimensional structural design task that we are attacking. Figure 1 shows an example problem in which L is the load and S1 and S2 are two supports. The so-called "topology" is given as a graph structure containing four edges (the members) and four vertices (the load, the two supports, and an intermediate connection point C). The topology does not specify the lengths of the members or the location of C. The topology and the position shown in the figure

Table 1: The 2-D Design Task.

Given:	A 2-dimensional region R A set of stable points (supports) A set of external loads with application points within R
Find:	The number of members, connectivity, and positions of all intermediate connection points such that the structure has minimum weight and is stable with respect to all external loads.

give the minimum-weight solution. In this solution, 4 members are used and E1 and E3 are in tension (they are being "stretched"), while members E2 and E4 are in compression. Tension members will be referred to as "rods" and indicated by thin lines. Compression members will be referred to as "columns" and indicated by thick lines. The type of members used in the solution is an abstraction that we have used throughout our work. To indicate a configuration of tensile and compressive members that consititutes a solution, we have defined the *stress state*. The stress state is an array of m elements in which each element corresponds to a member. The value of each element in the array is $+1$ if the member is tensile and -1 if the member is compressive.

The weight of a truss can be decreased in at least two ways. First, the engineer can use lighter material. Second, the "shape" can be designed in such a way that, for instance, it uses less material and, hence, it is lighter. In this paper we do not consider the (admittedly) important advances in the science of material but, instead, we focus on the synthesis of shapes that reduce the weight of a truss with a chosen construction material.

The task shown in Table 1 is actually only one step in the larger problem of designing good structures. In general, structural design proceeds in three steps [Palmer and Sheppard, 1970; Vanderplaats, 1984]. First, the problem solver chooses the topology, which specifies the locations of the loads and supports and the connectivity of the members. Then, the second step is to determine the locations of the connection points (and hence the lengths, locations, internal forces, and cross-sectional areas of the members) so as to minimize the weight of the structure. This is usually accomplished by numerical non-linear optimization techniques. The third and final step in the process optimizes the shapes of the individual members. This can often be accomplished by linear programming.

In addition to focusing only on the first two steps, we have introduced several simplifying assumptions to provide a tractable testbed for developing and testing machine learning methods. Specifically, we assume that structural members are joined by frictionless pins, only statically determinate structures are considered, the cross section of a column is square, columns and rods of any length and cross sectional area are available, and supports have no freedom of movement. A statically determinate structure contains no redundant members, and hence, the geometrical layout completely determines the forces acting in each member.

Given these assumptions, the weight of a candidate solution is usually calculated by a three-step process. The first step is to apply the *method of joints* [Wang and Salmon, 1984] to determine the forces operating in each member. Once this is known, the second step is to classify each member as compressive or tensile. This is important, because compressive and tensile members are composed of different materials and have different densities; e.g. concrete columns and high tensile steel rods. The third step is to determine the cross-sectional area of each member. The load that a member can bear is assumed to be linearly proportional to its cross-sectional area. Finally, the weight of each member can be computed as the product of the density of the appropriate material, the length of the member, and the cross-sectional area of the member.

The last two steps can be collapsed into a single parameter k: the ratio of the density per-unit-of-force-borne for compressive members to density per-unit-of-force-borne for tensile members. With this simplifica-

tion, instead of minimizing the actual weight, we can minimize the following quantity which, with an abuse of notation, we define as

$$Weight = \sum_{\substack{\text{tensile} \\ \text{members}}} \|F_i\| \, l_i + \sum_{\substack{\text{compressive} \\ \text{members}}} k \, \|F_j\| \, l_j .$$

F_i is the force in member i, and l_i is the length of member i. This is the initial objective function for the work described in this paper.

We conclude this section with a brief description of the method of joints, which is one of the methods used to calculate the F_i in statically determinate structures. The method of joints computes these forces by solving a system of linear equations as illustrated, for the problem in Figure 1, in Table 2. The matrix of coefficients is called [Wang and Salmon, 1984] the *axial* (or *static*) matrix and the vector of givens is defined as the *load vector*. In Figure 1, let C $= (x, y)$, S1 $= (x_1, y_1)$, and S2 $= (x_2, y_2)$, be the cartesian coordinates of the connection point, and the two supports, respectively. In addition, let (x_l, y_l) be the coordinates, and let p and γ be the magnitude and direction of the load L. The internal forces in each member are obtained by first constructing the axial matrix and load vector and then solving the system of equations for the unknown internal forces. Table 2 shows the symbolic system of equations for the example in Figure 1 with unknown forces F_1, F_2, F_3, and F_4 and with the coordinates of all the points explicitly substituted.

Now that we have defined the 2-dimensional design task and formulated it as a non-linear optimization problem, let us turn, in the next section, to a brief review of existing techniques for optimization and to the proposed methods.

Knowledge-based Optimization

Classical optimization textbooks [Vanderplaats, 1984; Papalambros and Wilde, 1988] present a comprehensive survey of optimization methods and of various techniques for conducting the search for an optimal solution. The schema illustrated in Figure 2 is typical of many domain independent non-linear optimization methods. The process is iterative. Starting at some initial point, the objective function is evaluated and the termination criteria are tested. If the test fails, a new point is generated by taking a step, of some chosen length in some chosen direction, away from the current point. Each point defines a set of values for the independent variables in the objective function.

Most optimization algorithms differ primarily in the criteria used to choose the direction along which to optimize. Some optimization methods (e.g., Powell's method [Vanderplaats, 1984]) choose the direction and step size using only evaluations of the objective function. Other methods, such as gradient descent and its variations [Papalambros and Wilde, 1988], require computation of the partial derivatives of the objective

Table 2: Method of Joints for the example in Figure 1. The product of the *axial matrix* and of the unknown forces F_i equals the *load vector*.

$$\begin{pmatrix} cos(\alpha_1) & cos(\alpha_2) & 0 & 0 \\ sin(\alpha_1) & sin(\alpha_2) & 0 & 0 \\ 0 & cos(\alpha_2 + 180) & cos(\alpha_3) & cos(\alpha_4) \\ 0 & sin(\alpha_2 + 180) & sin(\alpha_3) & sin(\alpha_4) \end{pmatrix} \times \begin{pmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{pmatrix} = \begin{pmatrix} Lcos(\gamma) \\ Lsin(\gamma) \\ 0 \\ 0 \end{pmatrix}$$

$$\begin{array}{llll} cos(\alpha_1) = (x_1 - x_l)/l_1, & cos(\alpha_2) = (x - x_l)/l_2 \\ cos(\alpha_3) = (x_1 - x)/l_3, & cos(\alpha_4) = (x_2 - x)/l_4 \\ sin(\alpha_1) = (y_1 - y_l)/l_1, & sin(\alpha_2) = (y - y_l)/l_2 \\ sin(\alpha_3) = (y_1 - y)/l_3, & sin(\alpha_4) = (y_2 - y)/l_4 \end{array}$$

and l_i's are Euclidean distances:

$$\begin{aligned} l_1 &= \sqrt{(x_1 - x_l)^2 + (y_1 - y_l)^2} \\ l_2 &= \sqrt{(x - x_l)^2 + (y - y_l)^2} \\ l_3 &= \sqrt{(x - x_1)^2 + (y - y_1)^2} \\ l_4 &= \sqrt{(x - x_2)^2 + (y - y_2)^2}. \end{aligned}$$

function to choose the new direction of optimization. Still other methods approximate the partial derivatives numerically by evaluating the objective function at many points.

The primary computational expense of numerical optimization methods is the repeated evaluation of the objective function. An advantage of gradient descent methods is that they need to evaluate the objective function less often, because they are able to take larger, and more effective steps. Of course, they incur the additional cost of repeatedly evaluating the partial derivatives of the objective function. Hence, they produce substantial savings only when the reduction in the number of function evaluations offsets the cost of evaluating the derivatives.

In engineering design, the objective function is typically very expensive to evaluate. This slows the numerical optimization process because the speed of numerical optimization is determined by the cost and frequency of evaluating the objective function. For the structural design domain to compute the objective function (volume of each structure) a system of linear equations must be solved. This is typically carried out by algorithms which are cubic in the number of unknowns. This number is usually large in real applications like bridge design. Furthermore, the fact that the constant k is applied only to compressive members makes it impossible to obtain a differentiable closed-form. The signs of the internal forces must be com-

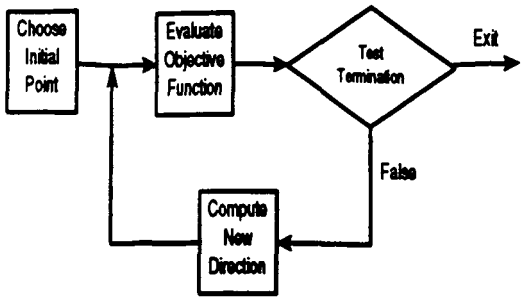


Figure 2: Traditional optimization schema.

puted before it is possible to determine which members are compressive. This prevents the use of gradient-based optimization methods that require fewer evaluations of the objective function – only slower function-based methods are applicable. One measure of the performance of a numerical optimizer is the time it takes to produce a solution. This quantity, however, depends on the choice of the starting point. Therefore, to obtain an accurate measurement, it is necessary to average the values obtained running the optimizer from different starting points.

Moreover, most engineering models are not unimodal. This directly affects the reliability of the solutions because numerical optimizers settle for local minima since they are unable to leap from one region to another to determine the global minimum. As shown in Figure 3, the objective function for the structural design domain is non unimodal. For instance, for the function in Figure 1 gradient methods started with $x = 1500$ and $y = 2000$ reach a local minimum in region R2 while the global minumum is in region R1. A measurement of the reliability can be obtained by taking the ratio (quality) of the local minimum and of the global minimum in controlled experiments in which the absolute minimum can be easily computed. Time and quality induce a tradeoff that can be exploited by defining the function:

$$\texttt{utility(solution)} = \texttt{CPUtime(solution)} * \texttt{CPUcost} + \texttt{quality(solution)}$$

where CPUcost is a positive constant that accounts for the cost of running the optimizer. We have used this definition in the learning stages of our approach to focus the attention of the optimization process on a few candidates that will produce solutions of maximum utility.

As shown in Figure 4, the increased reliability and speed are accomplished by augmenting the traditional run time optimization with a "compilation" stage prior to numerical optimization. The inputs to the compiler are (a) an high level description of the problem, (b)

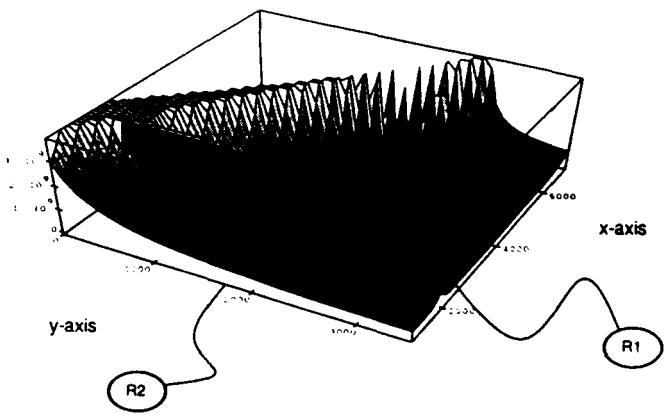


Figure 3: Volume of the structure in Figure 1.

domain knowledge about stress states, and (c) a procedure to generate training examples. Symbolic and inductive techniques are then used to (1) produce simplified versions of the objective function per each stress state, and (2) learn stress state selection rules which map problem instances into sets of candidate stress states of minimum cost.

First, the compiler produces one objective function for each topology and stress state. Each of these functions is a specialized version of the expression of the weight and it is faster to evaluate than the original, less specific, objective function. As an example, the function produced for the topology and stress state in Figure 1 is illustrated in Table 3. This expression is a closed form of the weight of a structure as a function of the two cartesian coordinates of connection point C restricted to region R1 in Figure 3. Moreover, these simplified expressions are differentiable and this permits the use of faster gradient-based optimization algorithms.

Another obstacle to practical applications of numerical optimization methods is the high dimensionality (number of independent variables) of the problems. Our compilation strategy decreases the dimensionality of optimization problems by searching a set of training examples for relations (regularities) among independent variables. These relations are then used as constraints among variables and are incorporated into the specialized versions of the objective function. This procedure eliminates independent variables with the result of greatly simplifying the optimization process, of enlarging its scope of applicability, and of speed-

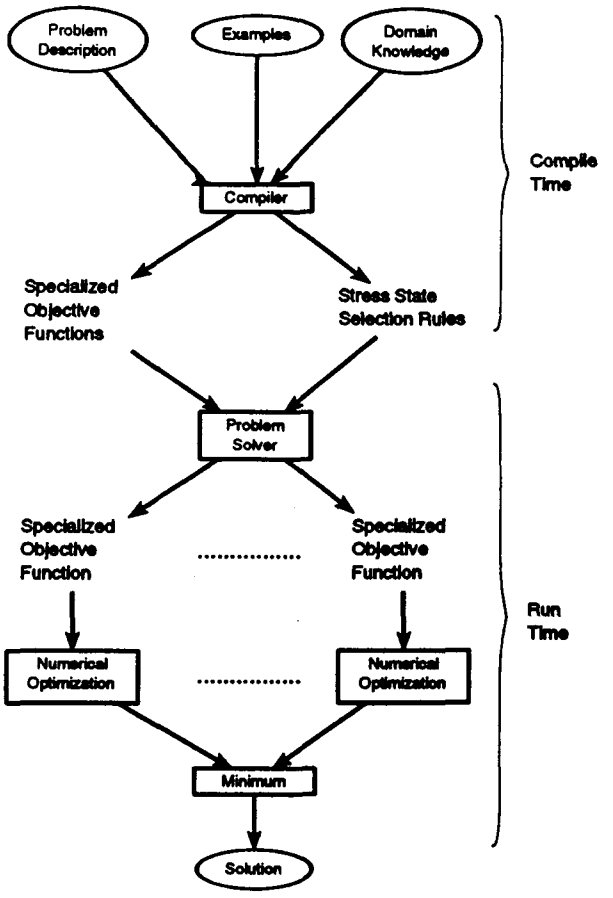


Figure 4: Proposed numerical optimization framework.

ing up run time optimization. For the region R1 in Figure 3, the compiler will determine that if the connection is expressed in polar coordinates ρ and α only the distance ρ from support S1 need be determined (see Figure 1.) This is because, in the analysis of the examples, it will discover that the angle α can be computed as one half of the angle β which is one of the givens of the problem. The final objective function is shown in Table 4 which contains only a single variable ρ vs. the two (x and y) in the expression in Table 3. This final expression indicates a reduction in dimensionality because, at run time, the numerical optimizer will only need to determine the value of ρ to compute the position of the connection point.

Finally, the compiler learns search control knowledge in the form of IF-THEN-ELSE rules. This is then used at run time to select stress states that lead quickly to quasi-optimal

Table 3: Partially evaluated objective function for the problem of Figure 1.

$$
\begin{aligned}
&Weight = \\
&\left(1.14\,10^{13}x - 5.66\,10^{9}x^2 + 8.16\,10^{5}x^3 + \right. \\
&\quad 3.28\,10^{13}y - 3.26\,10^{9}xy + 2.44\,10^{5}x^2y - \\
&\quad 6.70\,10^{9}y^2 + 8.16\,10^{5}xy^2 + 2.44\,10^{5}y^3 - \\
&\quad \left. 4.08\,10^{16}\right) / \\
&\left(1.28\,10^{1}xy - 2.56\,10^{4}x + 2.56\,10^{4}y - \right. \\
&\quad \left. 6.40\,y^2 - 2.56\,10^{7}\right)
\end{aligned}
$$

solutions. The set of stress states is chosen so that the *utility* of the stress states is maximized. The utility is a function that combines the time it takes to produce a solution with its expected quality (ratio to the global minimum.) This function introduces a trade-off between quality and time that is exploited by the learning algorithm [Cerbone and Dietterich, 1992]. As an example, for the design problem in Figure 1 whose objective function is shown in Figure 3, the compiler derives search control knowledge that allows the problem solver to focus the attention of the numerical optimizer on regions R1 and R2 when the load is directed toward support S2 and away from support S1.

Machine Learning Methods

This section describes in greater detail the symbolic and inductive learning techniques. Inductive learning techniques are used to (a) simplify the optimization process by reducing the number of independent variables and (b) derive the stress state selection rules. The inductive methods rely upon knowledge about the partitioning of the design space and upon a set of training examples that, for many engineering tasks, can be generated by the compiler. A complete discussion of the compilaton stages can be found in [Cerbone, 1992].

Symbolic Methods. Symbolic techniques are used to incorporate into the objective function knowledge about stress states and knowledge discovered during inductive analysis. The goal is to produce an highly simplified and specialized objective function. This is accomplished by partial evaluation [Futamura, 1971], and loop unrolling [Burstall and Darlington, 1977] – two techniques widely used in high-end optimizing compilers. Partial evaluation incorporates constant values for variables into functions (or programs) and simplifies them. Loop unrolling unfolds iterative con-

Table 4: Objective function for the structure in Figure 1 with reduced dimensionality.

$$
\begin{aligned}
&Weight_{simplified} \approx \\
&\left(1.16\,10^{13}\rho - 5.19\,10^{9}\rho^2 + 8.19\,10^{5}\rho^3 - 4.08\,10^{13}\right) / \\
&\quad 3.95\rho^2
\end{aligned}
$$

structs (e.g., `for` loops) and transforms them into sequential programs. These techniques have been implemented using the Mathematica programming language [Wolfram, 1988] and [Maeder, 1989] which is suitable to numerical problems.

As an example of specialization, we illustrate how domain knowledge is used to specialize the objective function. First the problem solver chooses the topology. This can be simply done by enumerating a few possible configurations. Once the topology is chosen, it can be incorporated into the objective function. This allows us to compute symbolically the axial matrix and the load vector (see Section). We then apply symbolic algorithms to solve and simplify the system of equations and to obtain a closed-form expression for the forces. In principle, an infinite number of topologies should be explored; however, Friedland [Friedland, 1971] experimentally demonstrated that only a few of them need be considered to achieve satisfactory solutions.

The second specialization step is to plug in the givens of the problem and partially evaluate the resulting mixed symbolic/numeric expression. For our examples, the givens of the problems are the loads and supports; however, one may wish to analyze a structure subject to different inputs such as various loading conditions or support locations. In such cases it is possible to leave those values in symbolic form and substitute their numerical values at run time.

The third compilation step is to split the objective function V into cases according to stress state. When the objective function is specialized according to stress state, the result is a collection of special-case objective functions $\{V_1, \ldots, V_n\}$. Because each V_j corresponds to one stress state, it is possible to tell, at compile time, which forces should be multiplied by k. Hence, each V_j is differentiable, and this enables us to employ gradient-based optimization techniques that, typically, are faster than methods based only on evaluating the objective function alone.

Reduction of independent variables. A further speedup and increase in reliability of the numerical optimizers is obtained using inductive methods to decrease the number of independent variables (*dimensionality*) in the numerical optimization problem. The compiler is given a series of examples and uses them to inductively determine which independent variables can be computed as functions of known quantities. For instance, in the design domain, when searching within a region it might turn out to be superfluous to search along all dimensions because there might exist a simple relationship between one of the coordinates and known quantities like the location of loads and supports. These relations are then used as constraints and are incorporated into the objective functions. The result is the reduction of the number of independent variables. This, in turn, produces an even simpler and faster optimization problem. For instance, the function shown in Table 3 has two independent variables while the corresponding inductively simplified version has only one independent variable and it is shown in Table 4. Hence, the final optimization problem entails a simple linear optimization while the original one has two dimensions.

The variables to be eliminated are determined using an EBL-like approach which employs:

- training examples
- a library of given geometry entities (points, angles, etc.)
- a geometrical domain theory
- known relationships among geometric entities
- *regularities* – a mixture of heuristics and statistical regression techniques.

Each unknown connection point is subject to a compile time heuristic search process that attempts to compute (reformulate) the location as a function of loads and supports.

To see how this works, let us consider again the example problem in Figure 1 which we shall refer to as the "bisector" example. In this example, the connection point `C` is the unknown and the givens are the load `L` and the supports `S1` and `S2`. Moreover, let us assume that a set of training examples has been either provided or derived by the system. The reformulation starts by identifying all geometric objects using the given domain theory. For the bisector example, the system identifies, among others, the following geometric objects:

`point(S1)`, `point(S2)`, `point(C)`, `point(L)`,
`angle(`β`, L, S1, S2)`, `angle(`α`, C, S1, S2)`,
`segment(SG1, S1, S2)`, ...

Predicates such as `point` and `angle` are basic elements of the given geometric domain theory. This means that, given a set of cartesian coordinates, the system is capable of computing each predicate. During the computation of each predicate, the system tags it as *given* or *unknown*. A predicate is *given* if all the entities used to compute it are either givens of the problem (loads or supports) or can be expressed a combination of given predicates. Otherwise, the predicate is tagged as *unknown*. For the bisector example, `point(C)` and all predicates that involve it in their derivation (e.g. `angle(`α`, C, S1, S2)`) are unknowns, all others are givens.

With this knowledge, the system then tries to relate the unknown geometric entity `point(C)` to as many other entities as possible with the ultimate goal of expressing it only using given geometric entities. This is accomplished by using a blend of EBL and discovery techniques. In the EBL jargon, the geometric knowledge base is the *domain theory*, `point(C)` is the target *concept*, and the operationality criterion is the fact that a concept must be expressed in terms of known geometric objects. To visualize this reformulation step, let us refer to the derivation tree in Figure 5.

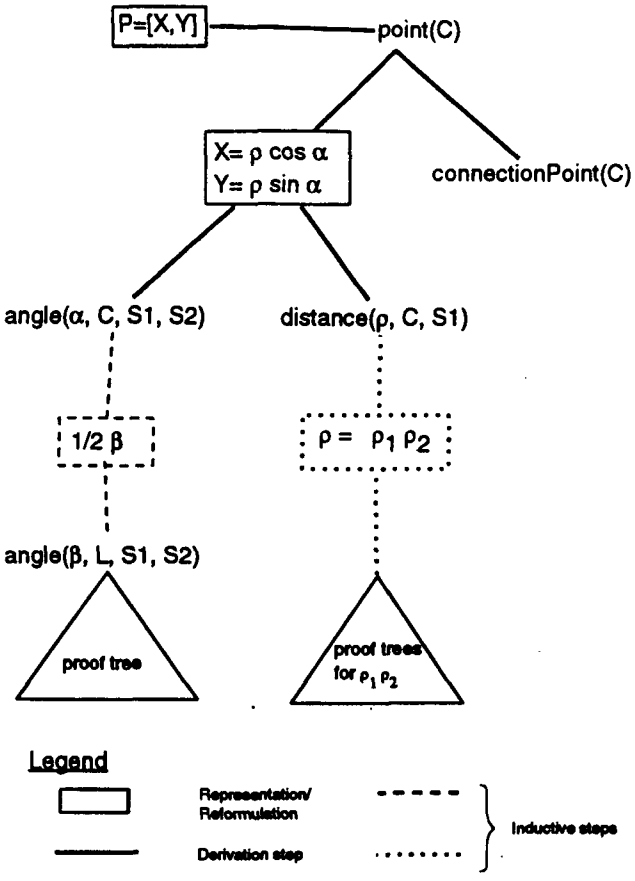


Figure 5: Decision tree to derive the *concept* `point(C)`.

The rightmost branch indicates that C is a connection point and, therefore, it is no longer explored. The leftmost branch, instead, uses a domain rule that reformulates a point in polar coordinates. Intuitively, the domain rule states that a point can be identified by its distance ρ from S1 and by the angle α between points C, S1, and S2. With this in mind, the system recursively tries to determine `angle(`α`, C, S1, S2)` and `distance(`ρ`, C, S1)`. After having exploited all proofs, the system concludes that it is not possible to re-express the `angle` and the `distance` in terms of known entities. If we were to follow EBL strictly, we should conclude that the domain theory is incomplete; that is, it is not powerful enough to bridge the gap between unknowns and givens. This, in turn, implies that the search would terminate concluding that `point(C)` cannot be re-expressed in terms of known geometric objects.

To overcome this problem we have used a discovery approach that fills these knowledge gaps with *eureka* [Burstall and Darlington, 1977] steps. Despite the name, however, in our strategy these steps are not arbitrary but inductive For the example in Figure 1, we determine that the angle α between points C, S1, and S2 is exactly one-half the angle β between points L, S1, and S2. Once this *regularity* is determined, in contrast with Burstall and Darlington's approach, we test the eureka step against all user provided examples to determine if it is a random occurance or a widespread phenomenon. In the former case, any use of this regularity is abandoned and others (if any) are tried. In the latter case, the regularity is assumed as a transformation of the unknown geometric entity. This is shown by the node in Figure 5 connected by the dashed lines. The system then subgoals on the geometric entities that were used to recognize the angle β. These are recognized as givens because they were derived from the position of the load and of the supports and the search terminates. The discussion of the branch identified by the dotted is similar to the one above and it is omitted for the sake of brevity.

The actual domain rules used in the geometric theory carry along also information that bridge the gap between the cartesian representation of a point and the polar one. This implies that the x and y coordinates of C can be expressed in terms of the angle α and of the distance ρ. In turn, the angle α is substituted by $\frac{\beta}{2}$ which can be computed from the given position of the load and supports. These transformations are considered as constraints and are incorporated into the objective function which is further simplified using the symbolic techniques. The result of the incorporation is shown in Table 4.

Rule derivation. The specialization steps discussed above greatly improve the running time of the optimizers on each objective function but they might introduce a large number of candidate solutions. These, in principle, can be exponential. To overcome this problem, we have devised a new inductive learning method to prune candidates that do not lead to optimal solutions. This method learns search control knowledge in the form of decision trees which can then be quickly transformed into `IF-THEN-ELSE` rules. These design rules associate features of the problem to a few regions in which the global minimum is believed to lie according to the examples given to the learning algorithms. The global solution is then obtained by running the optimizer on each of these regions and by taking the minimum solution.

We have found that most existing learning algorithms are not suitable for learning rules for optimization problems. The main obstacle is the absence of features that allow discrimination among classes. Algorithms like ID3 implicitely require independence of classes. Features with such discriminatory power are difficult to derive for many real application and especially for optimization tasks. On the other hand, it is relatively easy to provide *shallow* features which can circumscribe a set of possible solutions. Therefore, in devising our learning method we have assumed that all

features are *shallow* and proposed UTILITYID3, a novel learning algorithms. The algorithm resembles the well-known ID3 algorithm [Quinlan, 1987] in that it builds a decision trees and uses an information-theoretic heuristic to choose the feature on which to split at each recursive call. However, it is new in that the heuristic takes into consideration that the output is a set of recommended actions rather than a single discriminating class. This algorithm is fully described in [Cerbone, 1992] and [Cerbone and Dietterich, 1992].

In addition to the learning algorithm, we have introduced *maximum utility learning set*, a new learning framework. In this framework, a utility is associated to each candidate solution. The problem is to learn a set of actions of maximum utility that covers all given examples. For instance, in the design problem, the utility is a function of the time it takes the numerical optimizer to find a solution. The quality is measured with respect to the globally optimal design. It turns out that this learning problem is $\mathcal{NP}$ – *complete* [Garey and Johnson, 1979]. Hence, UTILITYID3 uses an approximation algorithm to determine a solution.

Experiments

To test the efficacy of this approach, we [Cerbone and Dietterich, 1991] have solved a series of design problems using an implementation based on Mathematica [Wolfram, 1988], and we have measured the impact of the compilation stages on the evaluation of the objective function, on the optimization task, and on the reliability of the optimization method. The measurements presented are averages over five randomly generated designs and, for each design, over 25 randomly generated starting points.

Objective function. The objective function of each design problem was evaluated in four different ways and, for each of them, we averaged the CPU[1] time over the different designs and starting points. The volume was first computed using the traditional, naive, numerical procedure with the method of joints. We then compiled the designs incorporating, in three successive stages, topological information, the givens of the problems, and the stress state. Figure 6 shows the time (per 100 runs) to evaluate the objective function at the various compilation stages. The biggest speedup was obtained with the numerical substitution of values into the symbolic closed form expression obtained and with the specialization to stress states. This suggests that the gain is related to the elimination of arithmetic operations from the original numerical problem.

Optimization. As indicated in Section , the running time of the optimizers is influenced by the number of function calls and by the time for each function evaluation. To present the benefits of our approach on the optimization task, we have experimented with two optimization algorithms (a) an optimizer based on Powell's method [Pike, 1986] that does not require gradient information and (b) the version of conjugate gradient descent [Press and others, 1988] provided by Mathematica. The graphs in Figures 7 and 8 report, respectively, the number of objective-function calls and the overall CPU time for each optimizer. The values connected by solid lines correspond to cases where the optimizer had no gradient information, while the values connected by dashed lines indicate averages utilizing the conjugate gradient descent method with alternative approximations for the gradient vector.

As expected, the number of evaluations remains constant throughout the compilation stages when the non-gradient is used, while it decreases drastically when we switch to the gradient-based optimization method.

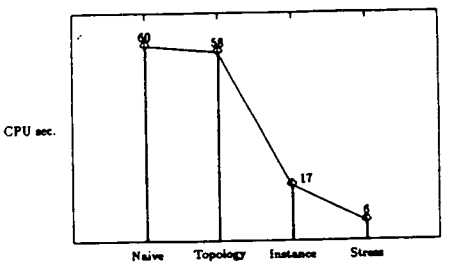


Figure 6: Influence of the compilation stage on the CPU time per function evaluation.

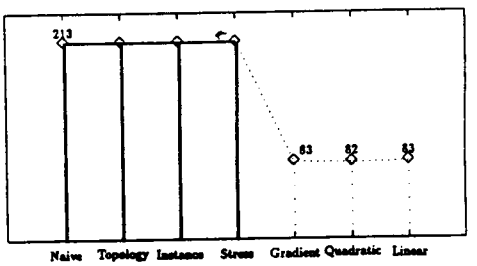


Figure 7: Influence of the compilation stage on the number of function calls.

The overall CPU time (Figure 8) steadily decreases as well. For the non-gradient method, the decrease is due to the progressive simplification of the objective

[1]The examples were run on a NeXT Cube with a 68030 board.

function itself, so that it is cheaper to evaluate. When we switch to the gradient method, there is initially no speedup at all, because the cost of evaluating the full gradient offsets the decrease in the number of times the objective function must be evaluated. However, additional speedups are obtained by approximating the objective function as a quadratic and as a linear function (by truncating its Taylor series).

We have found experimentally that there is no appreciable difference between the minima reached using the full gradient vector and the minima computed using quadratic approximations of the partial derivatives. However, the precision of the results obtained with the linear approximation is significantly reduced. Depending on the application, this trade of accuracy for speed may be acceptable. If not, the quadratic approximation should be employed.

Another possibility is to employ the linear approximation for the first half of the optimization search, and then switch to the quadratic approximation once the minimum is approached. In other words, the linear approximation can be applied to find a good starting point for performing a more exact search.

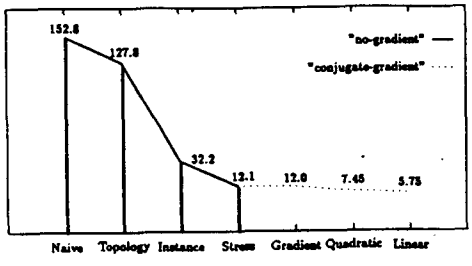


Figure 8: Influence of the compilation stage on the CPU time.

Reliability. An optimization method is reliable if it always finds the global minimum regardless of the starting point of the search. Unfortunately, as shown in Figure 3, the objective function in this task is not unimodal, which means that simple gradient-descent methods will be unreliable unless they are started in the right "basin." It is the user's responsibility to provide such a starting point, and this makes numerical optimization methods difficult to use in practice.

From inspecting graphs like Figure 3, it appears that, over each region corresponding to a single stress state, the objective function is unimodal. We conjecture that this is true for most of 2-D structural design problems. This means that optimization can be started from *any* point within a stress state, and it will always find the same minimum. If this is true, then our "divide-and-conquer" approach of searching each stress state in parallel will be guaranteed to produce the global optimum.

We have tested these hypothesis by performing 20 trials of the following procedure. First, a random starting location was chosen from one of the basins of the objective function that did not contain the global minimum. Next, two optimization methods were applied: the non-gradient method and the conjugate gradient method. Finally, our divide-and-conquer method was applied using, for each of the specialized objective functions V_j, a random starting location that exhibited the corresponding stress state. In all cases, our method found the global minimum while the other two methods converged to some other, local minimum.

Concluding Remarks

In this paper we have illustrated how machine learning techniques can be applied to optimal engineering design. This has been accomplished by tackling problems in two different areas:

- speeding up existing numerical methods
- learning a set of candidate optimal solutions.

Table 5 illustrates the correspondence between these problems and the machine learning techniques used in their solution. Our main contribution is to have shown that ML techniques can be effectively used to overcome some of the drawbacks of numerical optimizers and to increase their efficiency. Another contribution of this paper is to have shown that inductive techniques can complement traditional software engineering approaches in mathematical domains. This greatly reduces the need for knowledge transfer from experts to computer systems. In our approach, these results

Table 5: Rows enumerate problems in optimal design. Columns list Machine Learning paradigms. X's indicate the ML paradigm used to solve the problem.

	Symbolic Methods	*Inductive Learning*
Selection Rules		X
Speedup of Numerical Optimizers	X	X

required the use of a blend of novel and traditional optimization techniques. First, we have defined a new learning framework which is more appropriate to optimization tasks. This framework involves (a) the requirement that the output of the learning algorithm be a set of alternatives and (b) measures of the cost of obtaining solutions. The learning methods produce sets of minimum cost. Within this framework we have developed algorithms which output IF-THEN-ELSE rules that associate problem characteristics (features) to sets

of optimal solutions. This is a contribution to basic research in machine learning. Second, we have demonstrated that inductive methods can also be used to simplify numerical problems. In fact we employed a discovery approach to reduce the number of independent variables. Finally, we have used more traditional compiler optimization techniques in a learning framework and merged them with inductive methods. We have shown that the overall result is a drastic speedup of the numerical optimization techniques.

Our approach opens new research directions into the so far unexplored area of applications of machine learning to numerical optimization. It is our hope that, in the medium-to long-term, our techniques will allow the use of specialized numerical optimizers in real-time applications like intelligent CAD systems.

Acknowledgements

The author wishes to thank his advisor, Thomas G.Dietterich, for the discussions that lead to this paper, David G. Ullman and Prasad Tadepalli for comments on related papers, Igor Rivin for information on the internals of Mathematica, and Jerry Keiper for insights into `FindMinimum`□ in Mathematica version 1.2. Ullman is responsible for suggesting the term *stress state*. This research was supported by NASA Ames Research Center under Grant Number NAG 2-630.

References

Braudaway, Wesley 1988. Constraint incorporation using constrained reformulation. Tech.Rep. LCSR-TR-100 Computer Science Dept., Rutgers University.

Burstall, R.M. and Darlington, J. 1977. A transformation system for developing recursive programs. *Journal of the ACM* 24(1):44–67.

Cerbone, Giuseppe and Dietterich, Thomas G. 1991. Knowledge compilation to speed up numerical optimization. In *Proceedings of the Machine Learning Workshop*. 600–604.

Cerbone, Giuseppe and Dietterich, Thomas G. 1992. Inductive learning in engineering: A case study. In *Proceedings of the Adaptive and Learning Systems Conference of the IEEE Society for Optical Engineers.*

Cerbone, Giuseppe 1992. *Machine Learning in Engineering: Techniques to Speed up Numerical Optimization.* Ph.D. Dissertation, Oregon State University, Corvallis, OR.

Ellman, Thomas 1989. Explanation-based learning: A survey of programs and perspectives. *ACM Computing Surveys* 21(2):163–222.

Feigenbaum, E.A. 1977. The art of artificial intelligence 1: themes and case studies of knowledge engineering. Tech.Rep. STAN-CS-77-621, Stanford University, Dept. of Computer Science.

Friedland, L.R. 1971. *Geometric Structural Behavior.* Ph.D. Dissertation, Columbia University at New York, N.Y.

Futamura, Y. 1971. Partial evaluation of a computation process – an approach to a compiler-compiler. *Systems, Computers, and Controls* 2(5):45–50.

Garey, Michael J. and Johnson, David S. 1979. *Computers and Intractability, A Guide to $\mathcal{NP}$-completeness.* Freeman.

Maeder, Roman 1989. *Programming in Mathematica.* Redwood City, Calif. : Addison-Wesley, Advanced Book Program.

Palmer, A.C. and Sheppard, D.J. 1970. Optimizing the shape of pin-jointed structures. In *Proc. of the Institution of Civil Engineers.* 363–376.

Papalambros, Panos Y. and Wilde, Douglass J. 1988. *Principles of optimal design: modeling and computation.* Cambridge University Press.

Pike, Ralph W. 1986. *Optimization for Engineering Systems.* Van Nostrand.

Press, William H. and others, 1988. *Numerical Recipes in C: the art of scientific computing.* Cambridge University Press, Cambridge.

Quinlan, Ross J. 1987. Simplifying decision trees. *International Journal of Man-Machine Studies* 27:221–234.

Vanderplaats, Garret N. 1984. *Numerical Optimization Techniques for engineering design with applications.* New York: McGraw Hill.

Wang, Chu-Kia and Salmon, Charles G. 1984. *Introductory Structural Analysis.* Prentice Hall, New Jersey.

Wolfram, Steven 1988. *Mathematica.* Wolfram Research.

Reformulation Issues in Numerical Optimization

Giuseppe Cerbone*
Oregon State University
Computer Science Dept.
Corvallis, OR 97331

The initial goal of our research ([Cerbone, 1992], [Cerbone and Dietterich, 1992]) was to provide Machine Learning techniques to speed up numerical optimization. However, in hindsight, we have found opportunities to view part of our solution as a reformulation problem in mathematical domains.

Mathematical domains present a unique opportunity to develop and to test reformulation techniques. A typical mathematical task requires the solution of a set of equations subject to given constraints. As an example, in numerical optimization the problem is to determine the values of the independent variables that minimize an objective function subject to constraints on the values of the variables. Solutions to mathematical tasks that arise in engineering or in physics often cannot be found by algebraic manipulation and numerical methods must be employed to provide estimates. However, most optimization methods are slow and brittle. In fact, their speed and reliability depend on the choice of a starting point and on the formulation of the objective function. This prevents their applicability in large scale *real life* tasks.

To overcome these drawbacks, the user of numerical methods spends an enourmous amount of time reformulating the equations into a form appropriate for solution. In particular, for numerical optimization the goal is to determine a representation that allows to:

- Specialize the objective function to convex regions
- Decrease the number of independent variables

In the remainder of this overview, we outline the challenges and a few solutions to these representation changes in the design of lightweight frames to support loads.

A typical sequence of steps adopted by a problem solver to optimize a design is illustrated by the dashed lines in Figure 1 on the left. First, the engineer uses her/his own knowledge and experience to formulate a numerical task. Second, where possible, numerical optimization techniques are used to produce an optimal solution. Numerical optimization is typically a slow and brittle process. This is due to the fact that most numerical methods are hillclimbers. Thus, their speed is greatly affected by the number of evaluations of the objective function and by the time required for each evaluation. As shown in [Cerbone, 1992], we have devised techniques to reformulate the task to produce a faster optimization. Under many circumstances, the goal of speeding up the optimization conflicts with the goal of letting the engineer specify the objective function in a highly abstract format. In fact, while simplifying the formulation of the problem, the abstract specification can greatly slow down and decrease the reliability of the numerical solution. This is because optimizers have no knowledge of the problem domain. Therefore, at run time they use the same objective function provided by the engineer for all regions. In addition, some objective functions that arise in engineering are non differentiable. This prevents the use of powerful gradient-based numerical techniques – only slower function-based methods are applicable. This is especially true in the design task we are tackling.

Our research augments the traditional problem solving schema with the off-line knowledge compilation (or *learning*) stage illustrated by the solid lines in Figure 1 on the right. The compiler uses a blend of novel and traditional machine learning techniques to increase reliability and speed of the numerical optimization task. These results are accomplished by *reformulating* at compile time the design problem into subproblems and by deriving:

- Preprocessed objective functions for each subproblem
- Search control knowledge that allows the problem solver to focus only on a few subproblems.

The preprocessed functions contain fewer independent variables and have been greatly simplified. Therefore, they are faster to evaluate. At run time, the problem solver uses the search control knowledge derived during compilation to retrieve a few candidate solutions. Each of these candidates is then given as input to a numerical optimizer. However, in this case, the optimizer is given a simplified objective function. The net result is a speedup of as much as 95% over the run time of the traditional methods and a more reliable numerical optimization process. Being an off-line computation,

compilation does not introduce any overhead on the run time operations.

The specialization of the objective function is divided into two stages:

- Elimination of independent variables
- Identification of the "correct" abstraction to partition the function into convex regions.

Each of these two stages may require the reformulation of the objective function. In mathematical domains, the reformulation task consists in applying algebraic transformations to determine the "appropriate" format of an expression. To accomplish this task, algebraic operations are treated as operators that modify the function. Reformulation is also accomplished by a representational shift which changes the coordinate system from, say, polar to cartesian. Further reformulations takes place by choosing the appropriate origin, scale, and orientation of the coordinate system. Each of these operations can be considered as a reformulation of the original expression. As in most other reformulation tasks, an exhaustive search of all possible reformulations is unfeasible. Therefore, one must devise techniques to control the search.

In our research the need for reformulation arose during the elimination of independent variables. In some cases, the original objective function was given in cartesian coordinates. This representation did not allow any simplification. On the contrary, reformulating the function in a different coordinate system and performing algebraic simplifications allowed the elimination of independent variables from the optimization process. This was possible because the reformulation in polar coordinates revealed *regularities* among variables. The regularities were detected during the search and incorporated into the function represented in the new coordinate system. Regularities are detected by using a domain theory and heuristics such as `find equal angles`. These heuristics detect regularities only when the appropriate representation is chosen. In our solution ([Cerbone, 1992]), the search for the "correct" representation, is aided by classifying geometric entities (angles, points, lines, etc.) by type. These types are then related to changes in the representations.

A second important application of reformulation techniques to optimization problems is the automatic discovery of abstractions. In our solutions, we have used abstractions to partition the original optimization task into independent sub-problems over convex regions. In mathematical domains, the abstraction is usually a function of the independent variables that represents a change among stable states of the physical system. From a graphical standpoint, these changes correspond to multi-dimensional ridges that separate convex regions. To determine these abstractions the system must first find the singularities of the objective function and then synthesize these findings into a single expression. Given the parametric nature of expression, this task requires interpolation over a multidimensional parametric space. This is a complex task. However, if one takes into account the physical meaning of the regions of stability, it turns out that the determination of the state changes is a simpler process. In fact, we have used engineering intuition to partition the optimization problem into convex regions. Details of this process are contained in [Cerbone, 1992].

In conclusion, in this brief overview we have presented a few challenges that mathematical tasks present to reformulation. We claim that mathematical tasks are an ideal domain for reformulation techniques since they provide well-defined operators for representational shifts and the possibility of measuring the usefulness of a change of representation. On the other hand, mathematical domains pose formidable challenges that include a continuum search space and bridging the gap between numerical data and a higher level language that is closer to the experts' intuition.

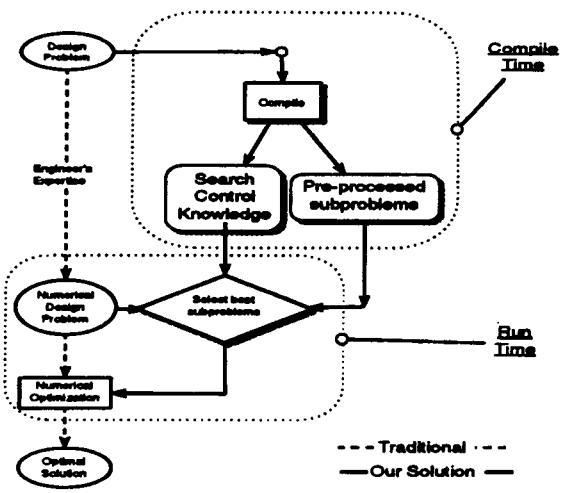


Figure 1: Problem solving strategies in numerical optimization.

References

Cerbone, Giuseppe and Dietterich, Thomas G. 1992. Inductive learning in engineering: A case study. In *Proceedings of the Adaptive and Learning Systems Conference of the IEEE Society for Optical Engineers.*

Cerbone, Giuseppe 1992. *Machine Learning in Engineering: Techniques to Speed up Numerical Optimization.* Ph.D. Dissertation, Oregon State University, Corvallis, OR.

Approximation, Abstraction and Decomposition in Search and Optimization

Thomas Ellman
Department of Computer Science
Rutgers University
ellman@cs.rutgers.edu

1. Synthesis of Search Control Heuristics

One portion of my research has focused on automatic synthesis of search control heuristics for constraint satisfaction problems (CSPs). I have developed techniques for automatically synthesizing two types of heuristics for CSPs: Filtering functions are used to remove portions of a search space from consideration. Evaluation functions are used to order the remaining choices. My techniques operate by first constructing exactly correct filters and evaluators. These operate by exhaustively searching an entire CSP problem space. Abstracting and decomposing transformations are then applied in order to make the filters and evaluators easier to compute. An abstracting transformation replaces the original CSP problem space with a smaller abstraction space. A decomposing transformation splits a single CSP problem space into two or more subspaces, ignoring any interactions between them. Both types of transformation potentially introduce errors into the initially exact filters and evaluators. The transformations thus implement a tradeoff between the cost of using filters and evaluators, and the accuracy of the heuristic advice they provide. I have shown these techniques to be capable of synthesizing useful heuristics in domains such as floor-planning and job-scheduling, among others. (See [Ellman, 1992].)

2. Synthesis of Hierarchic Problem Solving Algorithms

Another portion of my research is focused on automatic synthesis of hierarchic algorithms for solving constraint satisfaction problems (CSPs). I have developed a technique for constructing hierarchic problem solvers based on numeric interval algebra. My system takes as inputs a candidate solution space S and a constraint C on candidate solutions. The solution space S is assumed to be a cartesian product R^n where R is a set of integers. The constraint C is assumed to be represented in terms of arithmetic, relational and boolean operations. From these inputs the system constructs an abstract solution space S_a as a cartesian product R_a^n where R_a is a set of disjoint intervals that covers R. The system also constructs an abstract constraint C_a on abstract solutions. The abstract constraint C_a is obtained from the original constraint C by replacing ordinary arithmetic operations with interval algebra operations and replacing boolean operations with boolean set operations. The abstract space S_a and abstract constraint C_a are then used to build a hierarchic problem solver that operates in two stages. The first stage finds an abstract solution in the space S_a of intervals. The second stage refines the abstract solution into a concrete solution in the original search space S. I have shown this approach to be capable of synthesizing efficient problem solvers in domains such as floor-planning and job-scheduling, among others. (See [Ellman, 1992].)

3. Decomposition in Design Optimization

Another portion of my research is focused on automatic decomposition of design optimization problems. We are using the design of racing yacht hulls as a testbed domain for this research. Decomposition is especially important in the design of complex physical shapes such as yacht hulls. Exhaustive optimization is impossible because hull shapes are specified by a large number of parameters. Decomposition diminishes optimization costs by partitioning the shape parameters into non-interacting or weakly-interacting sets. We have developed a combination of empirical and knowledge-based techniques for finding useful decompositions. The knowledge-based method examines a declarative description of the function to be optimized in order to identify parameters that potentially interact with each other. The empirical method runs computational experiments in order to determine which potential interactions actually do occur in practice. We expect this approach to find decompositions that will result in faster optimization, with a minimal sacrifice in the quality of the resulting design. Implementation and testing of this approach are currently in progress. (I am pursuing this research in collaboration with Mark Schwabacher.) (See [Ellman *et al.*, 1992].)

4. Model Selection in Design Optimization

Another portion of my research is focused on intelligent model selection in design optimization. The model selection problem results from the difficulty of using exact models to analyze the performance of candidate designs. For example, in the domain of racing yacht design, an exact analysis of a yacht's performance would require a computationally expensive solution of the Navier-Stokes equations. Approximate models are therefore needed in order diminish the costs of analyzing and evaluating candidate designs. In many situations, more than one approximate model is available. For example, in the yacht design domain, the induced resistance of a yacht can be predicted by solving La Place's equation - an approximation of Navier-Stokes - or by using a simple algebraic formula. The two approximations differ widely in both the costs of computation and the accuracy of the results. Intelligent model selection techniques are therefore needed to determine which approximation is appropriate during a given phase of the design process.

We have attacked the model selection problem in the context of hillclimbing optimization. We have developed a technique which we call "gradient magnitude based model selection". This technique is based on the observation that a highly approximate model will often suffice when climbing a steep slope, because the correct direction of change is easy to determine. On the other hand, a more accurate model will often be required when climbing a gradual incline, because the correct direction of change is harder to determine. Our technique operates by comparing the estimated error of an approximation to the magnitude of the local gradient of the function to be optimized. An approximation is considered acceptable as long as the gradient is large enough, or the error is small enough, so that each proposed hillclimbing step is guaranteed to improve the value of the goal function. Implementation and testing of this approach are currently in progress. I am pursuing this research in collaboration with John Keane. (See [Ellman *et al.*, 1992].)

References

T. Ellman, J. Keane, and M. Schwabacher. The rutgers cap project design associate. Technical Report CAP-TR-6, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1992.

T. Ellman. Idealization-based methods for constraint satisfaction problems. Working Notes of the AAAI Workshop on Approximation and Abstraction of Computational Theories (Forthcoming), July 1992.

Abstraction and problem reformulation

Fausto Giunchiglia
Mechanized Reasoning Group
IRST
38050 Povo, Trento, Italy

In work done jointly with Toby Walsh, the author has provided a sound theoretical foundation to the process of reasoning with abstraction [GW90c; GW89; GW90b; GW90a]. The notion of abstraction formalized in this work can be informally described as:

[property 1] **the process of mapping a representation of a problem**, called (following historical convention [Sac74]) the "*ground*" representation, **onto a new representation**, called the "*abstract*" representation, which:

[property 2] **helps deal with the problem in the original search space by preserving certain desirable properties** and

[property 3] **is simpler to handle** as it is constructed from the ground representation by "throwing away details".

One desirable property preserved by an abstraction is provability; often there is a relationship between provability in the ground representation and provability in the abstract representation. Another can be deduction or, possibly inconsistency. By "throwing away details" we usually mean that the problem is described in a language with a smaller search space (for instance a propositional language or a language without variables) in which formulae of the abstract representation are obtained from the formulae of the ground representation by the use of some terminating rewriting technique. Often we require that the use of abstraction results in more efficient reasoning. However, it might simply increase the number of facts asserted (*eg.* by allowing, in practice, the exploration of deeper search spaces or by implementing some form of learning).

Among all abstractions, three very important classes have been identified. They relate the set of facts provable in the ground space to those provable in the abstract space. We call:

- TI abstractions all those abstractions where the abstractions of all the provable facts of the ground space are provable in the abstract space;
- TD abstractions all those abstractions where the "unabstractions" of all the provable facts of the abstract space are provable in the ground space;
- TC abstractions all those abstractions where a fact is provable in the ground space if and only if its abstraction is provable in the abstract space.

Historically the word abstraction has been mainly used with a much more restricted meaning which captures its use in problem solving and planning (for instance in Abstrips or Soar). Our notion of abstraction (and in particular the three classes defined above) turns out to capture and provide and unifying framework for describing work done in the definition of decision procedures (see for instance [DG79; Giu91]), in planning and problem solving (see for instance [Sac73; Ell90; MH91; Kno89]), explanation (see for instance [Doy86]), common sense reasoning (see for instance [Hob85]), qualitative and model based reasoning (see for instance [Moz90; Wel91]), approximate reasoning [Imi87]), analogy (see for instance [Ble90]) and reasoning with very large data bases (see for instance [Lev92]).

At a close look abstraction seems also very related to problem reformulation. In particular it seems that problem reformulation can be characterized as using some of the subclasses of TC and TD abstractions introduced in [GW90c]. A positive feedback on this intuition would allow to use the framework described in [GW90c; GW89] to put the work on problem reformulation on a more solid ground and, at the same time, to study and compare the techniques used in problem reformulation with the techniques used in all the other areas captured by the framework.

References

W.W. Bledsoe. A precondition prover for analogy. CS Dept. memo, 1990.

B. Dreben and W.D. Goldfarb. *The Decision problem - Solvable classes of quantificational formulas.* Addison-Wesley Publishing Company Inc., 1979.

R.J. Doyle. Constructing and refining causal explanations from an inconsistent domain theory. In *Proc.*

Fifth National Conference on Artificial Intelligence, Philadelphia, PA, 1986. AAAI.

T. Ellman. Mechanical generation of heuristics through approximation of intractable theories. In *Working Notes of AAAI Workshop on Automatic Generation of Approximations and Abstractions*, Boston, MA, 1990.

E. Giunchiglia. A set of hierarchically structured decision procedures for some subclasses of First Order Logic. In *Proceedings 3rd Scandinavian Conference on Artificial Intelligence*, Roskilde University, Denmark, 1991. Also available as MRG-DIST Technical Report 9101-01, University of Genova, Italy.

F. Giunchiglia and T. Walsh. Abstract Theorem Proving. In *Proc. IJCAI 89*, 1989. IRST Technical Report 8902-03. Also available as DAI Research Paper No 430, University of Edinburgh.

F. Giunchiglia and T. Walsh. Abstraction in AI. *AISB Quarterly*, 73:22–26, 1990.

F. Giunchiglia and T. Walsh. The inevitability of inconsistent abstract spaces. Technical Report 9006-16, IRST, Trento, Italy, 1990. Also available as DAI Research Paper, University of Edinburgh. Accepted to the Journal of Automated Reasoning.

F. Giunchiglia and T. Walsh. A Theory of Abstraction. Research paper no. 516, Dept. of Artificial Intelligence, University of Edinburgh, 1990. Also available as IRST-Technical Report 9001-14. To appear in Artificial Intelligence.

J.R. Hobbs. Granularity. In *Proc. 9th IJCAI conference*, pages 432–435. International Joint Conference on Artificial Intelligence, 1985.

T. Imielinski. Domain abstraction and limited reasoning. In *Proc. 10th IJCAI conference*, pages 997–1003. International Joint Conference on Artificial Intelligence, 1987.

C.A. Knoblock. Learning hierarchies of abstraction spaces. In *Proc. Sixth Intnl. Workshop on Machine Learning*, 1989.

A.Y. Levy. Irrelevance in problem solving. In *Submitted to AAAI-92*, 1992.

I. Mozetic and C. Holzbaur. Extending EBG by abstraction operators. In *Proceedings EWSL-91*, Porto, Portugal, 1991. Springer-Verlag.

I. Mozetic. Abstractions in model-based diagnosis. In *Working Notes of AAAI-90 Workshop on Automatic Generation of Approximations and Abstractions*, pages 64–75. AAAI, 1990.

E.D. Sacerdoti. Planning in a Hierarchy of Abstraction Spaces. In *Proc. 3rd IJCAI conference*. International Joint Conference on Artificial Intelligence, 1973.

E.D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

D.S. Weld. Reasoning about model accuracy. Technical Report 91-05-02, University of Washington, Dept. Computer Science and Engineering, 1991.

Queries for Bias Testing

Diana F. Gordon

Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory, Code 5510
Washington, D.C. 20375
gordon@aic.nrl.navy.mil

Abstract

Selecting a good bias prior to concept learning can be difficult. Therefore, dynamic bias adjustment is becoming increasingly popular. Current dynamic bias adjustment systems, however, are limited in their ability to identify erroneous assumptions about the relationship between the bias and the target concept. Without proper diagnosis, it is difficult to identify and then remedy faulty assumptions. We have developed an approach that makes these assumptions explicit, actively tests them with queries to an oracle, and adjusts the bias based on the test results.

1 Introduction

Bias is a fundamental aspect of any supervised concept learner. Numerous papers have noted this importance (e.g., Mitchell 1980; Haussler 1988). The type of bias that we discuss here is the choice of a hypothesis language. The hypothesis language defines the space of hypotheses. A *strong* bias defines a small hypothesis space; a *weak* bias defines a large hypothesis space; a *correct* bias defines a space that includes the target concept. A strong correct bias, e.g., one with fewer features, is generally desirable because it reduces the number of hypothesis choices and thereby promotes rapid convergence to the target concept.

The bias can be adjusted (shifted) dynamically during incremental concept learning by strengthening the bias when possible and weakening it to regain correctness. Recently, interest has grown in systems that dynamically shift the bias (e.g., Utgoff 1986; Rendell 1990; Spears & Gordon 1991). These systems, however, are limited in their ability to identify erroneous assumptions about the relationship between the bias and the target concept. Proper diagnosis aids in the recovery from faulty assumptions. We have developed an approach to bias adjustment that addresses this need for proper diagnosis. Our method consists of a bias tester and adjuster that can be added to an incremental concept learner to improve the learner's performance.

Unlike previous approaches to bias testing, our approach uses formal definitions of assumptions about the bias, called *biasing assumptions*, to guide an analysis of *why* the bias is inappropriate (e.g., too weak, or incorrect) for learning the target concept. An example of a biasing assumption is the irrelevance of a feature for learning the target concept. The bias tester performs this analysis (called a *biasing assumption test*) by actively testing the bias with queries to an oracle. Each query is a request to an instance generator for a new instance. For example, the irrelevance of a feature might be tested by querying an oracle for the class (positive/negative) of instances having different values of that feature. The bias adjuster then records the analysis results and adjusts the bias accordingly. If a biasing assumption holds, the adjuster strengthens the bias, e.g., by removing the irrelevant feature from the hypothesis language. Otherwise, the adjuster weakens the bias or allows the bias to stay the same if no adjustments are needed.

Our approach has three primary advantages. First, the bias tests are composed of queries. Queries can accelerate learning significantly (see Gordon 1990; 1992). Second, our approach is designed to be incorporated into an existing concept learner. Third, our approach diagnoses the bias to find and record specific erroneous biasing assumptions. This enables the bias to be *minimally weakened* as well as corrected. Minimal weakening is most advantageous when a stronger bias is desirable. In that case, bias strengthening along with minimal bias weakening can enable very rapid acquisition of the target concept (see Gordon 1990; 1992).

In our framework, the bias is the set of features and their values in the hypothesis language. These values appear in *value trees* (e.g., see Figure 1), which are input by a user or knowledge engineer who is somewhat familiar with the domain. Value trees are

typically called *generalization trees* because parent nodes are more general than their child nodes. Training instances are described in terms of leaf node values. Throughout this paper, we assume the concept learner begins with hypotheses described in terms of the instance language and evolves its hypotheses (perhaps using the value trees) in a specific-to-general direction. *Generalization* increases the generality of values within a particular hypothesis; *abstraction* increases the generality of the hypothesis language. The concept learner can use value trees for generalization. Our approach to bias testing and adjustment uses value trees for abstraction. Bias strengthening implies removal (i.e., abstraction) of a feature or feature value distinction from the hypothesis language. This shrinks the hypothesis space. Bias weakening implies the restoration of features or feature value distinctions. This weakening undoes abstraction and enlarges the hypothesis space. Bias weakening is defined to be minimized when the features and feature value distinctions that are restored to the language are restricted to those that *must* be restored to correct the bias.

The drawback of our approach is that it requires an oracle that can respond to queries during learning. The oracle can be either a human or the environment. In either case, it is not always practical to require an oracle. Humans may be too busy to answer questions. Furthermore, the use of the environment as an oracle is impractical if lives are at stake. For example, it is unreasonable to query whether a new chemical weapon is effective at killing people. On the other hand, it *is* practical to query whether small doses of Vitamin C cure the common cold.

Using Figure 1, we can see how a bias may be strengthened, weakened, and minimally weakened. Suppose the bias is all the trees in Figure 1, and the target concept states that **small bricks** are positive and instances of any other description are negative. The bias might be strengthened by removing all features other than "size" from the hypothesis language. This bias is incorrect because "shape" information is also needed to learn the target concept. One way to weaken and correct the bias is to restore the original language. Alternatively, we can minimally weaken the bias by restoring parts of the "shape" value tree but none of the "material" tree. Within the "shape" tree, we restore the "cube"/"brick" distinction and above, and restore the "curved-solid" node, but do not restore any child of the "curved-solid" node. Removing a distinction strengthens the bias to create an abstraction, whereas restoring it weakens the bias to undo the abstraction.

Section 2 formally defines two important biasing assumptions and then collapses them into one. Section 3 presents and analyzes algorithms to test the collapsed assumption. Section 4 summarizes and explains empirical results. Finally, Sections 5 and 6 present related work and a summary of the paper.

2 Biasing Assumptions

When supervised concept learners shift their bias, they typically make an implicit biasing assumption that the bias shift is correct for learning the target concept. Our approach makes each biasing assumption explicit, and associates each assumption with an abstraction operator. If the assumption holds, the corresponding abstraction operator can fire.

We assume two abstraction operators: *climb-value-tree(f,a)* and *remove-feature(f)*. The climb-value-tree(*f*,*a*) operator replaces values of feature *f* that are lower in the value tree (e.g., "cube" and "brick" in Figure 1) with a value *a* (e.g., "prism") that is higher in the tree throughout the hypothesis language. The remove-feature(*f*) operator eliminates feature *f* from the hypothesis language. We associate a *cohesion* assumption with climb-value-tree(*f*,*a*). Cohesion implies that the values below *a* in the value tree of *f* are unnecessary for predicting the target concept membership of instances. We associate an *irrelevance* assumption, which is equivalent to cohesion at the root node of a value tree, with drop-feature(*f*). Irrelevance implies that the feature to be removed is unnecessary for predicting the target concept membership of instances.

The following are the formal definitions of the two biasing assumptions. These definitions are tailored for an incremental concept learning context. We assume one new instance is accepted at a time and all previous instances are saved. Furthermore, we assume that the instances are not noisy and that the instance features are sufficient to distinguish positive from negative instances, though perhaps not ideal for learning the target concept. We abbreviate the set of all known positive instances at time t with POS(t), the set of all known negative instances at time t with NEG(t), the set of all positive instances with POS, and the set of all negative instances with NEG. We abbreviate the new instance at time t with $i(t)$, the target concept with TC, the irrelevance biasing assumption with $IRR(f,TC,t)$ for feature f, and the cohesion biasing assumption with $COH(a,TC,t)$ for value a.

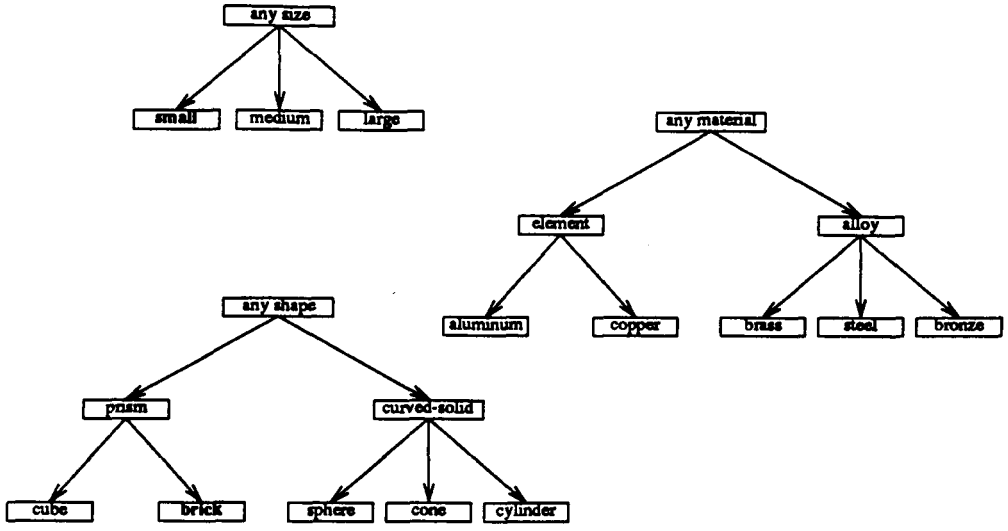


FIG. 1. Value trees.

For the following definitions, if $i(t)$ is positive, we let $L(t)$=(POS(t) $\cup$ $\{i(t)\}$) and L=POS or we let $L(t)$=NEG(t) and L=NEG. Likewise, if $i(t)$ is negative, we let $L(t)$=(NEG(t) $\cup$ $\{i(t)\}$) and L=NEG or we let $L(t)$=POS(t) and L=POS.

Let $\{f_1,...,f_n\}$ be the set of features considered relevant as of time (t - 1). Let $1 \le i \le n$, where i is the subscript used in the following definitions. Finally, we define $f_i(x,v_i)$ to mean that the value of feature f_i for instance x is v_i. Although the instance language consists of value tree leaf nodes, a nonleaf node can also be used to describe an instance, though not uniquely. We allow v_i to be either a leaf or nonleaf node in the following definitions. The formal definition of the irrelevance biasing assumption is:

$$IRR(f_i,TC,t) \leftrightarrow$$
$$((\forall v_1,...,v_n)(((\exists x \in L(t))(f_1(x,v_1) \,\&...\&\, f_n(x,v_n))) \rightarrow$$
$$((\forall w_i)(\forall y)((f_1(y,v_1) \,\&...\&\, f_i(y,w_i) \,\&...\&\, f_n(y,v_n))$$
$$\rightarrow (y \in L))))).$$

In other words, f_i is considered irrelevant to learning TC at time t if changing the value of f_i in any known instance x always yields a (new or old) instance whose classification (positive/negative) is the same as that of x.

Next, we define the cohesion biasing assumption. The cohesion of value a with respect to the target concept, $COH(a,TC,t)$, means that the descendent nodes $\{a_1,\ldots,a_l\}$ below value (node) a in the value tree appear to behave equivalently with respect to target concept membership. Let $A = \{a_1,\ldots,a_l\}$. Let $1 \le i \le n$ and $1 \le j,k \le l$. The formal definition is:

$$COH(a,TC,t) \leftrightarrow$$
$$((\forall v_1,...,v_n)(\forall a_j \in A)(((\exists x \in L(t))$$
$$(f_1(x,v_1) \,\&...\&\, f_i(x,a_j) \,\&...\&\, f_n(x,v_n))) \rightarrow$$
$$((\forall a_k \in A)(\forall y)((f_1(y,v_1) \,\&...\&\, f_i(y,a_k) \,\&...\&\, f_n(y,v_n))$$
$$\rightarrow (y \in L))))).$$

In other words, cohesion holds for value a for learning TC at time t if the replacement of one descendent value of a with another descendent value in any known instance x always yields a (new or old) instance whose classification is the same as that of x. Note that irrelevance is a special case of cohesion that occurs when a is the root node of a value tree. Therefore, these two assumptions can be collapsed into one. Let us call the collapsed assumption $IRR\text{–}COH(a,TC,t)$. The definition of this collapsed assumption is identical to that of $COH(a,TC,t)$.

3 Queries for Testing the Biasing Assumptions

The definition of $IRR\text{–}COH(a,TC,t)$ presented in the last section has been translated into algorithmic biasing assumption tests. This section presents the algorithms for these tests. There are two types of

biasing assumption tests, corresponding to the two times at which tests are executed. Each type is associated with a separate algorithm. One type of test (in Section 3.1) executes before bias shifting, and the other type (in Section 3.2) executes after bias shifting.

Like the definition on which they are based, our biasing assumption tests are tailored for incremental concept learning. If an assumption test is satisfied, then the corresponding biasing assumption is considered valid, and therefore it is "safe" to implement the abstraction corresponding to this assumption. If the test is not satisfied, corrective action may be required.

We assume that our approach to bias testing and shifting is added to an incremental concept learner that maintains two Disjunctive Normal Form (DNF) hypotheses: one that covers all previously seen positive instances and one that covers all previously seen negative instances. The flow of control begins when a query to the instance generator requests a new instance. When the new instance is received by the concept learner, the learner uses its hypotheses to predict the class of this instance. The learner then consults an oracle to find out the true class of the instance. If enough instances have been seen at this time to complete an assumption test, the bias is shifted according to the test results. Next, the learner updates its hypotheses to preserve *completeness* and *consistency*. Completeness implies the positive hypothesis covers all known positive instances and the negative hypothesis covers all known negative instances. Consistency implies the positive hypothesis covers no known negative instances and the negative hypothesis covers no known positive instances. These steps are repeated until the user decides the target concept has been learned. For more details see (Gordon 1992).

We introduce four types of queries to facilitate concept learning with bias shifts: *bias strengthening queries*, *bias weakening queries*, *counterexample queries*, and *random instance queries*. An assumption test is a sequence of bias strengthening queries or a sequence of bias weakening queries. Bias strengthening queries test abstractions before they are made; bias weakening queries retest abstractions after they have been made. The last two types of queries are not part of assumption tests, but they are useful for other reasons. The purpose of counterexample queries is to find out *whether* the bias is incorrect. If an incorrectness is found, the bias weakening queries then determine *why* the bias is incorrect. The purpose of the random instance queries is to generate instances for concept learning when none of the other queries applies. All queries except the bias weakening queries request instances not previously seen. Bias weakening queries try to use previously seen instances before generating new ones because they retest previously held assumptions, and the necessary instances to do this are often already present.

Random instance and counterexample queries are simple, so we describe them first. Random instance queries are requests to the instance generator for randomly generated (previously unseen) instances. Counterexample queries can provide counterexamples because they are requests for randomly generated (unseen) instances that are covered by one of the hypotheses. A negative instance covered by the positive hypothesis is a counterexample, and a positive instance covered by the negative hypothesis is a counterexample.

3.1 Bias Strengthening Queries

Bias strengthening queries test whether the biasing assumption associated with a potential abstraction holds. To do this, these queries test nodes of the value tree below the potential abstraction. If these values do not seem useful for distinguishing target concept membership, the abstraction is made.

The assumption tests that use bias strengthening queries and are executed prior to abstraction may be as rigorous as desired. Tests that use more queries are more rigorous. (Gordon 1990) describes a method for varying the rigor of these tests. Increasing the rigor can reduce the number of prediction errors, but it can also significantly increase the cost of the tests. For example, suppose we wish to test the cohesion of value a of feature f. Let us consider how we may vary the rigor of tests that are based on the formal definition of $IRR\text{-}COH(a,TC,t)$ in Section 2. We can increase the rigor with which we test the cohesion assumption by the following two methods: (1) Increase the number of values a_k from A to substitute for the original value a_j before assuming cohesion holds; (2) Increase the number of instances x whose f value is varied before assuming cohesion holds. (Note that each x corresponds to a unique choice of values for v_1 through v_n.) Either of these methods will increase the number of queries.

Here, we describe an algorithm that does not have very rigorous assumption tests and is therefore not excessively costly. It is not very rigorous because only one sibling of the original value is tested for each

hypothesis disjunct before making an abstraction, and because hypothesis disjuncts, rather than instances, have their values varied. There are typically far fewer disjuncts than instances. Our algorithm for generating bias strengthening queries to test biasing assumptions is the following:

Repeat the following until no more unseen, uncovered instances i' can be generated:
For each instance feature f do
Find the value a that is the parent node of the value of f in some (arbitrary) disjunct of one of the two hypotheses. The value a is a potential abstraction to be tested. If this value has been tested previously, then select another value for a from another disjunct.
For each (positive, negative) hypothesis h do
For each disjunct *disj* of h do
(1) Find $f(x,v)$, a conjunct of *disj*. If none exists, or if v is not a child of a, try another disjunct.
(2) Set SIBLINGS equal to the set of all siblings (which share a parent node a) of v in the value tree of f. These siblings are children of a.
(3) Select $s \in$ SIBLINGS.
(4) Replace "$f(x,v)$" in *disj* with "$f(x,s)$" to form d'. Then form a potential instance i' that satisfies d' by translating d' to the language of the instances, which consists of leaf values in the value trees. When translating to leaf values, the choice of a descendent of a higher level value is random. Check to see that i' is not already covered by the hypotheses and has not been seen yet. If i' is uncovered and unseen, request i' from the instance generator and continue. Otherwise, try other choices of leaf value descendents until they have all been tried. If the descendents have all been tried, then go to step (3) to find another sibling.
(5) Accept i' from the instance generator and consult the oracle for the class of i'.
Endfor
Endfor
(If the assumption test for abstraction a has succeeded at this point, the bias adjuster makes the abstraction.)
Endfor

This algorithm executes a sequence of biasing assumption tests. Each biasing assumption test corresponds to a test of an abstraction a. This test consists of a sequence of queries that vary the values in the hypothesis disjuncts. To ensure that these queries do not overlap with the other query types, bias strengthening queries only request instances that are not covered by either of the hypotheses and have not yet been seen. To form each bias strengthening query, this algorithm selects one feature f of one disjunct of one hypothesis h and alters the value of this feature. This is a form of *perturbation* (Porter & Kibler 1986). The value of f that is perturbed is a child, i.e., an immediate descendent, of the abstraction a in the value tree. The new value obtained through perturbation is a sibling of the original value, i.e., both values are children of a in the value tree. This new value is substituted for the old value in the hypothesis disjunct, and an instance that matches this description and has random values for unspecified features is requested.

Let us assume the algorithm is testing abstraction a. For each hypothesis h, suppose perturbing the value of f in all disjuncts of h yields only instances whose class is the same as that of h. Then nodes in the value tree below a do not seem to be useful for distinguishing positive from negative instances (though this assumption might later be proven wrong). Thus, the abstraction to a is permissible and can therefore be made by the bias adjuster. In other words, the biasing assumption test for abstraction a is satisfied. On the other hand, if an instance of a different class is generated, the abstraction cannot be made because the biasing assumption test is not satisfied.

Because we assume the hypotheses begin with the language of the instances, which consists of leaf values in the value trees, this algorithm tests abstractions one value tree level at a time, beginning one level up from the leaves. For example, when using the "material" tree of Figure 1, the language shift to "alloy" would be tested before the language shift to the root node "any material".

We can now see how this algorithm corresponds to the formal definition of $IRR\text{-}COH(a,TC,t)$ in Section 2. For each disjunct d of hypothesis h, we let x (from the assumption definition) be any instance covered by d. We also let v_1 through v_n be the feature values present in d. To perturb the value of feature f_i, our algorithm selects an a_k for the sibling value. If the substitution of a_k into d yields a new instance whose classification differs from that of x, the assumption being tested does not hold. On the other hand, if the

substitution of a_k's into *every* disjunct of h yields only instances of the same class as x, our assumption passes the test and is considered to hold. The perturbation values a_k are children of node a in a value tree, where a is the abstraction being tested. If a is the root node of the value tree, our algorithm tests the irrelevance of f. Otherwise, our algorithm tests the cohesion of a.

An upper bound on the number of bias strengthening queries generated by this algorithm is $O(F * D * d)$, where F is the number of instance features, D is the maximum number of disjuncts in the two hypotheses, and d is the maximum depth of a value tree. The branching factor of the value tree is not included in the cost of this algorithm because to test each abstraction only one sibling value is requested for each disjunct.

To illustrate bias strengthening queries, suppose we are testing the feature "size", where the features and trees of Figure 1 are used. Furthermore, suppose feature "material" is considered irrelevant and has been removed from the hypothesis language, and the current hypotheses are:

POS HYP: { x |
((size(x,small) & shape(x,brick))
v
(size(x,large) & shape(x,sphere))) }

NEG HYP: { x |
(size(x,medium) & shape(x,cylinder)) }.

Then a bias strengthening query to test the abstraction to "any size" is formed by using the first disjunct of the positive hypothesis to request a **medium brick** (or **large brick**) that has a randomly chosen value for "material". Then the second disjunct of the positive hypothesis is used to request a **small sphere** (or **medium sphere**), and the only disjunct of the negative hypothesis is used to request a **small cylinder** (or **large cylinder**), each with randomly chosen values for "material". If the first two instances are positive and the third instance is negative, then "size" is considered irrelevant and is removed from the hypothesis language by the bias adjuster. An abstraction is created when "size" disappears from the hypothesis language. On the other hand, if any of the requested instances has a different classification than the hypothesis from it was derived (e.g., the first instance is negative), then the abstraction is not created and "size" remains in the hypothesis language.

3.2 Bias Weakening Queries

Because the algorithm of Section 3.1 does not exhaustively test the biasing assumptions (e.g., only *one* sibling value per disjunct is tested before creating an abstraction), abstractions made after running this algorithm cannot be guaranteed to be correct. Therefore, bias weakening queries are needed to retest the abstractions after a prediction error in case the prediction error is due to an incorrect abstraction rather than a generalization error. These queries perturb the values of the description of the instance for which a wrong prediction has been made to isolate erroneous abstractions that might have caused the error.

Suppose a wrong prediction is made on a new instance i, and H is the hypothesis whose class differs from that of i. Then the following is our algorithm for generating bias weakening queries following a wrong prediction:

Form the set A of all abstractions present in the hypothesis language (and in H) that apply to i. These are the abstractions to be tested. Elements of A are feature-value pairs.
For each $(f,a) \in A$ that is not already known to be faulty do
  Let L be the set of all leaves in the value tree for f that are below a.
  For each $v \in L$ do
    (1) Substitute v for the corresponding feature value in the description of i to form a new description of i'. If i' has not yet been seen, ask the concept learner to predict then get the actual class (from the oracle) of i'. Otherwise, use the known class of i'.
    (2) If the class of i' is the same as that of i then loop again to find another element of L. Otherwise, record (f,a) as being faulty and abort this loop through the values of L to get another element of A.
  Endfor
Endfor

An abstraction applies to an instance if it has a value of feature f that is more general than the value of f in the instance. In other words, the abstraction must be a value tree ancestor of the value of f in the instance.

This algorithm retests the biasing assumption associated with each abstraction a. The values for perturbation are leaf nodes in the value tree below the abstraction a being tested. A feature value of i is varied by substituting a perturbation value into the

description of i and requesting an instance of this new description from the instance generator along with the class of the requested instance. If perturbing any of the values of i causes the generation of a new instance of a different class than i, then the abstraction being tested by perturbation is faulty. This abstraction is faulty because it removes a distinction that is below it in the value tree and that is necessary for predicting target concept membership.

Although bias weakening queries do not retest *all* biasing assumptions, they rigorously retest all biasing assumptions associated with abstractions that apply to i. Therefore, they identify all biasing assumption errors that caused the error in predicting the class of i. By doing so, these queries correct the bias in a way that enables the concept learner to regain consistency and completeness with respect to all previous instances, including i. We consider this testing to be rigorous because *all* descendent (leaf) values are tested until a value is found that disallows the abstraction. If none is found, the concept learner regains consistency and completeness without bias shifts.

Similarly to Section 3.1, we can see how this algorithm corresponds to the formal definition of *IRR–COH* (a,TC,t) in Section 2. This algorithm alters the value of f in the new instance i for which a wrong prediction has been made to create queries that request new instances. The algorithm then tests whether these newly-created instances have the same classification as i. The instance i plays the role of x in the definitions, and the perturbation values are the a_k's. If the abstraction a being tested is a root node value of a value tree, this algorithm tests the irrelevance of f. Otherwise, the algorithm tests the cohesion of a.

A is the set of all abstractions that apply to i. If F is the number of instance features, then the maximum size of A is F. This is because, for each instance feature f, only one ancestor (in the value tree) of the value of f in i will be in the hypothesis language at a particular time, and there are at most F features in the hypothesis language. In other words, for each f, there exists at most one $(f,a) \in A$. Furthermore, for each $(f,a) \in A$, this algorithm tests all leaf node descendents of a. There are at most b^d of these descendents, where d is the maximum depth and b is the maximum branching factor of any value tree. Therefore, an upper bound on the number of queries generated by this algorithm is $O(F * b^d)$. This algorithm is executed for each instance i for which the concept learner makes a wrong prediction.

To illustrate bias weakening queries, we continue with the example in Section 3.1. Suppose the bias strengthening queries cause "size" to be considered irrelevant and removed from the hypothesis language. The hypotheses are now:

POS HYP: { x | ((shape(x,brick)) v
(shape(x,sphere))) }

NEG HYP: { x | (shape(x,cylinder)) }.

If a **large copper brick** is not among the known instances, and a counterexample query requests one, and this instance is negative, then the concept learner will incorrectly predict the class of this instance. Bias weakening queries now perturb the description of this instance to determine the source of the prediction error. Perturbation to test the abstraction to "any size" might result in a request for an instance that is a **small copper brick**. If this example is positive, the assumption that "size" is irrelevant for distinguishing target concept membership is incorrect. If this instance is negative, the next query might be a request for a **medium copper brick**.

After bias weakening queries identify incorrect biasing assumptions, the bias adjuster weakens the bias to correct it. In the example just described, "size" would have to be restored to the hypothesis language to distinguish the **small copper brick** that is positive from the **large copper brick** that is negative. After the bias has been corrected, the concept learner relearns the instances to reform the hypotheses with the new language bias. The bias weakening queries enable bias weakening to be minimized because they identify the incorrect biasing assumptions. *All* assumptions not proven incorrect (in the past or the present) can be assumed to hold and can therefore be preserved during relearning.

3.3 Order of the Queries

The order for selecting the queries is as follows. The first query is a random instance query; the next query is a bias strengthening query. Bias strengthening queries continue as the default unless one of the following holds: (1) *complacency* occurs; (2) the concept learner makes a prediction error; or (3) no bias strengthening query can be formed.

Complacency is defined to occur when the concept learner has made four consecutive, correct predic-

tions.[1] A string of correct predictions indicates either that the concept has been learned or that counterexamples to the current hypotheses should be sought. Since it is not possible to know for certain whether the correct concept has been learned, counterexample queries occur in response to complacency. Bias weakening queries are the response to prediction errors. Once the diagnosis performed by these queries is completed, the bias strengthening queries resume until complacency occurs. If none of the other queries can be formed, random instance queries are generated. Counterexample queries can be formed only if they generate unseen instances that are covered by the current hypotheses. Bias strengthening queries can be formed only if they generate unseen instances that are *not* covered by the current hypotheses.

4 Results and Cost/Benefit Analyses

In this section, we summarize previously published empirical results. We then explain these results from three perspectives: system bias appropriateness, a query cost analysis, and a query benefit analysis. Finally, we present an example that illustrates why our approach is effective.

4.1 Empirical Results

We have added an implementation of our approach to bias testing and shifting to an incremental concept learner to form a system called PREDICTOR (Gordon 1990; 1992). In the experiments of (Gordon 1992), PREDICTOR's performance is compared with that of a baseline system called Iba's Algorithm Concept Learner (IACL), which is based on an algorithm from (Iba 1979). PREDICTOR is built on top of IACL by extending IACL's bias shifting capabilities. PREDICTOR is identical to IACL in all ways except two: the former system tests the bias prior to bias shifting whereas the latter does not, and the former system prefers a stronger bias than the latter system. Both systems consult an oracle to answer membership queries. IACL's membership queries are requests for randomly chosen instances. Also, both systems can shift the bias. However, IACL, like ID3 (Quinlan 1986) and a number of other concept learners, interleaves hypothesis selection and term (feature) selection. For IACL, bias shifting is not a deliberate, high-priority task as it is for PREDICTOR.

[1] The number of correct predictions needed was chosen by empirical tests.

The empirical experiments of (Gordon 1992) demonstrate that when its method is appropriate, PREDICTOR produces an order of magnitude improvement in the rate of convergence to the target concept and its negation over IACL. The empirical experiments of (Gordon 1990) demonstrate that, when appropriate, PREDICTOR has a better convergence rate than all the other systems with which it has been compared (IACL, a variant of ID3, and a version of AQ described in Michalski et al. 1986). If inappropriate, however, PREDICTOR produces a performance degradation with respect to IACL and the other systems. PREDICTOR's method is appropriate precisely when one would expect it to be - when bias shifting is the most expedient action to take to learn the target concept, i.e., there is a large disparity between the instance language and the language in which the target concept can be expressed most succinctly.

It is easy to see how this disparity would be likely to occur in many real-world learning situations. Most objects are described in terms of primitive features. It is reasonable to expect that a knowledge engineer, who is familiar with the domain in which concept learning will occur, would be aware of a number of potentially useful abstractions but would not be certain which abstractions are relevant for learning the concept. Therefore, this engineer might have the system begin with the known primitive features, but provide the system with potential abstractions. When provided with a set of potentially useful value trees, PREDICTOR's queries can isolate those abstractions which are correct and thereby expedite the learning process.

From an experimental study described in (Gordon 1992), we have learned that PREDICTOR has a synergistic effect between its bias shifting method and its bias tests. The system's bias shifting method promotes this synergy because few bias strengthening queries are required before an abstraction occurs. (Recall from Section 3.1 that the number is proportional to the number of hypothesis disjuncts.) Furthermore, the system minimally weakens the bias after a set of bias weakening queries (see Gordon 1992). In both cases, PREDICTOR is able to capitalize on the query results to gain and maintain a strong bias.

4.2 System Bias Appropriateness

In this paper, we use the term "bias" synonymously with "hypothesis language bias". This is the bias that PREDICTOR adjusts. In this section, we will discuss a meta-level bias, namely, the *system*

bias. To avoid confusion with the hypothesis language bias, hereafter, we refer to the system bias as the *system policy* or simply the *policy* as in (Provost & Buchanan 1992). At some level, all concept learning systems have some form of fixed system policy. For example, even the most adjustable system cannot implement all possible bias adjustment methods. The choice of bias adjustment methods is a fixed system policy. No one policy can be best for learning every target concept. Therefore, we need to develop an understanding of the appropriateness of system policies for learning different classes of concepts.

PREDICTOR's policy is its implicit assumption that abstraction is appropriate to try (by strengthening the language bias) and to retain (by minimally weakening the language bias) as much as possible. Strengthening and minimally weakening the bias are appropriate when they are the correct actions to take in the context of the current instance language and target concept.

PREDICTOR's bias strengthening and weakening queries, which form the majority of the system's queries, are geared entirely toward gathering information for bias shifting. This system uses its queries to reorder the instances to increase its information gain about the hypothesis language bias early in the learning process. Bias shifting is its priority. If this priority matches the task, PREDICTOR usually outperforms all other systems with which it is compared. This is because once PREDICTOR's queries have achieved their goal of a strong correct bias, the number of instances required to converge on the target concept is often significantly reduced by this bias shift (see Section 4.4).

Although IACL can adjust the hypothesis language, PREDICTOR generally far outperforms IACL when bias shifting is important. This is because IACL's policy places a higher priority on finding less general hypotheses and also on maintaining hypothesis consistency and completeness with previous training instances than it does on bias shifting (see Gordon 1992). This system only shifts the bias when it decides this is the best way to achieve its other priorities. IACL's queries are requests for randomly-generated instances. Therefore, this system does not favorably order the instances for gathering information about the bias. The other systems with which PREDICTOR has been compared have problems that are similar to IACL's.

In the next two sections, we present cost/benefit analyses that further explain the empirical results.

4.3 Query Costs

The upper bounds on PREDICTOR's queries, presented in Sections 3.1 and 3.2, seem somewhat high. Why does this system generate fewer queries (converge earlier) than IACL when bias shifting is appropriate for learning the target concept? Why does it generate more queries than IACL when it is inappropriate? In this section, we present a rough cost comparison between the number of queries generated by PREDICTOR and the number generated by IACL. To simplify our analysis and avoid a confusion between the effects of generalization and those of abstraction, let us suppose that no generalization is needed for learning the target concept.[2]

In Section 3.1, we show that an upper bound on the number of bias strengthening queries is $O(F * D * d)$, where F is the number of instance features, D is the maximum number of hypothesis disjuncts, and d is the maximum depth of any value tree. According to Section 3.2, an upper bound on the number of bias weakening queries generated in response to each wrong prediction on a new instance is $O(F * b^d)$, where b is the maximum branching factor of any value tree.

Suppose W wrong predictions are made prior to convergence.[3] Then an upper bound on the number of bias weakening queries generated to resolve W wrong predictions is $O(W * F * b^d)$. Therefore, an upper bound on the total number of bias strengthening and weakening queries prior to convergence to the target concept and its negation is $O((F * D * d) + (W * F * b^d))$.

The total number of PREDICTOR's queries also includes random instance and counterexample queries. However, in this analysis we ignore the cost of these two types of queries because at most four random instance queries occur before the counterexample queries activate (see Section 3.3), and the counterexample queries typically do not take long to find a counterexample. Once a counterexample is found, the bias weakening and then strengthening queries resume.

---

[2] This simplification does not significantly affect our analysis.

[3] PREDICTOR also executes bias weakening queries at another time besides when it makes a wrong prediction. Nevertheless, this does not significantly alter our cost estimates. We therefore omit a discussion of this topic. See (Gordon 1992) for details.

IACL's bias shifts are triggered by the order of the training instances (see Gordon 1992). A serendipitous order will enable the system to make the correct abstractions early. However, since this order is random, it cannot be guaranteed to be helpful. Furthermore, as mentioned in Section 4.2, IACL's bias shifts occur when IACL decides this is the best way to achieve its other goals. The main problem with this approach is that it does not offer much help to a system that needs to recover from incorrect abstractions. Often, what is really an abstraction problem is treated as a generalization problem by the system. As a result, incorrect abstractions often linger, thereby preventing correct abstractions from being made. In the worst case, this would cause the number of queries required for convergence to the target concept and its negation to equal the total number of instances, which is $O(b^{d} * F)$, where b^{d} is the maximum number of leaf nodes in any value tree.

Comparing the upper bounds for the two systems, we note that as F increases, IACL's performance should degrade much more rapidly than that of PREDICTOR. Furthermore, the upper bound cost for IACL depends on the data structures but not on the target concept. On the other hand, W and D in the formula for PREDICTOR's upper bound cost depend, at least in part, on the target concept.

D depends almost entirely on the target concept. W, on the other hand, depends both on the target concept and the bias appropriateness. If we assume the data structures and target concept are fixed, and we wish to analyze the effectiveness of bias shifting, then we need to focus on the W component of the cost formula. In particular, as W approaches zero, PREDICTOR's cost upper bound approaches a polynomial. W tends toward zero as a greater proportion of the biasing assumptions made by PREDICTOR are correct, e.g., most features are irrelevant and therefore the irrelevance assumption holds frequently. This is precisely the situation in which empirical experiments have shown PREDICTOR outperforms IACL.

Likewise, when most of the biasing assumptions are incorrect, W can become very large. In the worst case, we would have the same situation as we have with IACL, where all instances would have to be seen to learn the target concept. Again, empirical experiments confirm this analysis.

Our analysis of IACL's upper bound cost is much like that of the other systems with which PREDICTOR has been compared because these systems are also not designed for bias testing and shifting. Their system policies favor other tasks.

4.4 Query Benefits

One of the goals of our method for bias testing and shifting is to reduce the number of features in the hypothesis language. Relevant results from computational learning theory can provide a rough estimate of the benefits of strengthening the bias in this way.[4] The *sample complexity* is the number of instances required to converge to the target concept. (Haussler 1988) has shown that sample complexity relates directly to the Vapnik-Chervonenkis (VC) dimension of a hypothesis space, which is a measure of the expressiveness of the hypothesis language. A less expressive language implies a stronger bias and a lower VC-dimension. Haussler measures convergence in the Probably Approximately Correct (PAC) framework of learnability, which assumes the error $\varepsilon > 0$ and the confidence $(1 - \delta) < 1$. In this framework, a concept is expected to be learned approximately with high probability.

According to (Haussler 1988), given a fixed ε and δ the minimum sample complexity is directly proportional to the VC-dimension. Furthermore, if the concept hypothesis is in k-DNF (which is true for many concept learners), then

$$VC\text{-}dim(H) \leq 4ks \log(4ks\sqrt{n}),$$

where H is the hypothesis space, n is the number of features in the instance language, s is a bound on the number of terms (disjuncts), $k \leq n$, and $s \leq \binom{n}{k}$.

We can now apply this theoretical estimate of sample complexity to partially explain the empirical results summarized in Section 4.1. In our framework, empirical performance is measured in terms of *absolute convergence* to the target concept, rather than PAC convergence. Absolute convergence implies the error ε is 0 and the confidence $(1 - \delta)$ is 1. In other words, we require that the target concept be learned precisely. We also require that the negation of the target concept be learned precisely for absolute convergence to hold. (These requirements are meaningful for applications where the cost of making an error is high.)

In our framework, the maximum number of literals per term, k, is equal to the number of (relevant)

[4] Only a "rough" estimate can be provided because the computational learning theory makes assumptions, such as instance selection from a fixed distribution, that are not met in our framework.

features in the hypothesis language. Also, we use a different bound on s for our framework because when there are irrelevant features, n no longer affects the number of disjuncts. If we assume there is a maximum of v values for any of the k relevant features, $s \leq v^k$. When PREDICTOR discovers irrelevant features and strengthens the bias (by reducing k), Haussler's inequality (with the new upper bound on s) predicts that the system will reduce the VC-dimension and thus reduce the sample complexity. With the new bound on s, Haussler's inequality also predicts a reduction in sample complexity when v is reduced by abstractions that are made based on cohesion assumptions. Furthermore, PREDICTOR's ability to minimally weaken the bias enables it to *retain* a hypothesis space with a low VC-dimension.

In summary, when bias strengthening is desirable, the cost of using queries to gather information about the bias can be offset by the benefit of a reduction in the sample complexity gained by having a stronger bias. In Section 4.3, we showed how in this situation the costs are also reduced. So, when appropriate, PREDICTOR's method can yield lower costs and increased benefits and a better performance in comparison with IACL and other systems. When inappropriate, the costs increase and the benefits are reduced and system performance degrades with respect to the performance of IACL and other systems.

4.5 Illustrative Example

Let us examine a very simple illustration of how the queries and bias shifts together can result in a synergy that reduces the convergence rate. We will focus primarily on the value of the bias weakening queries and minimal bias weakening. Assume we have two concept learners, CL-Q and CL. They differ only in that CL-Q uses the bias weakening queries and performs minimal bias weakening, whereas CL does not. CL's method for bias weakening is to make the hypothesis language equal to the instance language. CL's motivation for bias weakening is the same as that of CL-Q, namely, to resolve prediction errors. Other than the bias weakening queries of CL-Q, we assume both systems request random instances from an oracle.

For simplicity, let us further suppose that both systems are given a strong bias beforehand by the system implementor and their only bias shifting task is to weaken the bias if they discover (by a wrong prediction) that the bias is incorrect. Also, for simplicity, we assume neither system generalizes. They only learn concepts using bias adjustments.

The data structures given to both systems are the value trees of Figure 1, except that feature "size" is now restricted to having the values "small" and "large", and the value "curved-solid" has no child values. The target concept, as in Section 1, states that **small bricks** are positive and instances of any other description are negative. Finally, both systems begin with a hypothesis language bias which states that "shape" is the only relevant feature.

Both systems begin by requesting the same two instances: a **small aluminum brick**, which is positive, and a **large steel curved-solid**, which is negative. The current hypothesis now held by both systems is:

POS HYP: { x | (shape(x,brick)) }

NEG HYP: { x | (shape(x,curved-solid)) }.

If the next instance requested by both systems is a **large bronze brick**, and it is negative, both systems will make a prediction error, which triggers bias weakening.

CL-Q responds to the prediction error by using its bias weakening queries. It requests one instance to retest the abstraction to "any size" - a **small bronze brick**, which is positive. Since the class is different than that of the **large bronze brick**, CL-Q decides the abstraction to "any size" is faulty and removes it. CL-Q then requests a **large aluminum brick**, a **large copper brick**, a **large brass brick**, and a **large steel brick** to retest the abstraction to "any material". Since these instances have the same class as the **large bronze brick**, this abstraction is considered permissible. CL-Q now minimally weakens the bias to obtain the hypotheses:

POS HYP: { x |
(size(x,small) & shape(x,brick)) }

NEG HYP: { x |
((size(x,large) & shape(x,brick))
v
(size(x,large) & shape(x,curved-solid))) }.

The system now needs one more instance, a **small curved-solid** of any material, to converge on the target concept and its negation precisely. The odds are high that a random choice will request this instance before all instances have been seen. (Note that if the bias strengthening queries were used, this would be the next instance requested.)

After making a prediction error on the **large bronze brick**, CL behaves differently. It does not make the extra five queries that CL-Q made. Neither does it have the extra information CL-Q obtained. Rather than *minimally* weakening the bias, CL weakens the bias to the instance language. CL, therefore, requires *all* remaining 27 instances to precisely learn the target concept and its negation. In this case, it is clear that the information gained by CL-Q from the queries have offset their cost.

5 Related work

Our approach is related to theoretical research on irrelevance (e.g., Subramanian 1989) and relevance (Grosof & Russell 1989). Subramanian's definition of irrelevance is similar to ours. However, her definition is tailored for reformulating a problem solver's language to increase the problem solver's efficiency. Our definition, on the other hand, is tailored for incremental concept learning. Grosof and Russell have created a theory of shifting bias as nonmonotonic reasoning. They use the notion of relevance to motivate bias shifts. Prior to learning, biases are ordered from stronger (i.e., having fewer relevant components) to weaker (i.e., having more relevant components). Learning begins with a strong bias and shifts to weaker biases as needed. Unlike the bias shifts described here, Grosof and Russell's bias shifts are not motivated by an analysis of biasing errors and therefore they are unable to guarantee that the bias can be minimally weakened.

Our approach is also related to approaches that form abstractions based on equivalence classes. If cohesion holds for a value a of feature f, then a forms an equivalence class in terms of the target concept membership of instances having this value of feature f. Kokar's COPER is an example of another system that uses equivalence classes for concept learning (Kokar 1990). COPER uses the concept of invariance and an expectation of equivalence classes to indicate when constructive induction is needed. Constructive induction is the dynamic generation of new features. PREDICTOR could use a similar approach to COPER's to decide when to invent new abstractions. For example, if cohesion does not hold for some abstraction a in a value tree, this could be considered an indication of the need to split a into two separate abstractions for which cohesion does hold.

The use of active learning in our approach is related to literature on queries for concept learning, both theoretical (e.g., Angluin 1988) and experimental (e.g., Sammut & Banerji 1986; Muggleton 1987). Our approach is most similar to that of the few systems that query an oracle *and* shift the bias. Notable examples include the MARVIN system of Sammut & Banerji (1986), Gross's CAT (Gross 1991), Muggleton's Duce (1987), and the CLINT system described in De Raedt & Bruynooghe (1990). MARVIN shifts the bias by learning the definitions of new user-selected terms and then uses these new terms for further learning. This system also queries an oracle to test generalizations within the term definitions. These queries involve a form of perturbation similar to that used here. Nevertheless, MARVIN's queries are not for the purpose of deciding how to shift the bias.

CAT, Duce, and CLINT are all systems that query an oracle to make bias shifting decisions. Of all these approaches, our approach is most similar to that of CLINT, which uses irrelevance queries for bias strengthening. Nevertheless, the latter system does not use irrelevance queries to select a weaker bias. Furthermore, our approach is unlike that of CLINT and all other systems because our choice of a weaker bias is determined by bias tests that diagnose errors in such a way as to guarantee that the bias can be minimally weakened.

6 Summary

This paper presents a unique approach to bias shifting. Rather than performing unjustified bias shifts, as most concept learning systems do, we first test assumptions about the relationship between the bias and the concept being learned. We also *re*-test these assumptions in light of new, possibly contradictory evidence, i.e., a prediction error. These tests are performed with queries to an oracle. By using this approach, a system can both strengthen and minimally weaken the bias.

In addition to presenting our method for bias testing, this paper also summarizes empirical results. The empirical results are then explained by a cost/benefit analysis. Both the empirical and analytical results indicate that when bias strengthening is desirable, the query costs are lower and the benefits of a stronger bias, namely, a reduced sample complexity, are increased. Likewise, when bias strengthening is not appropriate, the query costs are increased and the benefits are reduced. The low costs and increased benefits have produced a large performance gain over other systems when bias strengthening is appropriate, and the high costs and decreased benefits have produced a performance loss with respect to other systems

when bias strengthening is inappropriate.

Acknowledgements

I thank Bill Spears for his constant help, encouragement, and insightful suggestions throughout this entire project. Also, thanks to Jaime Carbonell, Don Perlis, and Chinoo Srinivasan for their guidance, and to Alan Schultz for his helpful comments on previous drafts.

References

Angluin, D. (1988). Queries and concept learning. *Machine Learning, 2*, 319-342.

De Raedt, L. & Bruynooghe, M. (1990). Indirect relevance and bias in inductive concept learning. *Knowledge Acquisition, 2*, 365-390.

Gordon, D. (1990). *Active bias selection for incremental, supervised concept learning.* Ph.D. thesis, University of Maryland, College Park. Also (Technical Report UMIACS-TR-90-60 CS-TR-2464) University of Maryland, Department of Computer Science.

Gordon, D. (1992). *Actively testing and minimally weakening the inductive bias.* (NCARAI Technical Report AIC-91-011). Also, submitted to *Machine Learning.*

Grosof, B. & Russell, S. (1989). *Shift of bias as non-monotonic reasoning.* (Technical Report 14620) IBM, Yorktown Heights.

Gross, K. (1991). *Concept acquisition through attribute evolution and experiment selection.* Ph.D. thesis, Carnegie-Mellon University, Pittsburgh.

Haussler, D. (1988). Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence, 36*, 177-221.

Iba, G. (1979). *Learning disjunctive concepts from examples.* (A.I. Lab Memo 548). Massachusetts Institute of Technology.

Kokar, M. (1990). Semantic equivalence in concept discovery. In D. Benjamin (Ed.), *Change of Representation and Inductive Bias.* Boston: Kluwer.

Michalski, R., Mozetic, I., Hong, J. & Lavrac, N. (1986). *The AQ15 inductive learning system: An overview and experiments.* (Technical Report UIUCDCS-R-86-1260). University of Illinois, Department of Computer Science.

Mitchell, T. (1980). *The need for biases in learning generalizations.* (Technical Report TR CBM-TR-117). Rutgers University, Department of Computer Science.

Muggleton, S. (1987). Duce, an oracle based approach to constructive induction. In *Proceedings of the Tenth International Conference on Artificial Intelligence.* Milan, Italy: Morgan Kaufmann.

Porter, B. & Kibler, D. (1986). Experimental goal regression: A method for learning problem-solving heuristics. *Machine Learning, 1*, 249-286.

Provost, F. & Buchanan, B. (1992). Inductive policy. In *Proceedings of the Tenth National Conference on Artificial Intelligence.* San Jose: Morgan Kaufmann.

Quinlan, J. (1986). Induction of decision trees. *Machine Learning, 1*, 81-106.

Rendell, L. (1990). Feature construction for concept learning. In D. Benjamin (Ed.) *Change of Representation and Inductive Bias.* Boston: Kluwer.

Sammut, C. & Banerji, R. (1986). Learning concepts by asking questions. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Spears, W. & Gordon, D. (1991). Adaptive strategy selection for concept learning. *Proceedings of the First International Workshop on Multistrategy Learning.* Harpers Ferry: George Mason University.

Subramanian, D. (1989). *A theory of justified reformulations.* Ph.D. thesis, Stanford University, Stanford.

Utgoff, P. (1986). Shift of bias for inductive concept learning. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos: Morgan Kaufmann.

Reformulating Non-Monotonic Theories For Inference and Updating

Benjamin N. Grosof
IBM T. J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598
(914) 784-7100 ; Internet: grosof@watson.ibm.com

Abstract

We aim to help build programs that do large-scale, expressive non-monotonic reasoning (NMR): especially, *"learning agents"* that store, and revise, a body of conclusions while continually acquiring new, possibly defeasible, premise beliefs. Currently available procedures for forward inference and belief revision are *exhaustive*, and thus impractical: they compute the entire non-monotonic theory, then re-compute from scratch upon updating with new axioms. These methods are thus badly intractable. In most theories of interest, even backward reasoning is combinatoric (at least NP-hard). Here, we give theoretical results for prioritized circumscription that show how to reformulate default theories so as to make forward inference be *selective*, as well as *concurrent*; and to restrict belief revision to a *part* of the theory. We elaborate a detailed divide-and-conquer strategy. We develop concepts of *structure* in NM theories, by showing how to reformulate them in a particular fashion: to be conjunctively *decomposed* into a collection of smaller "part" theories. We identify two well-behaved special cases that are easily recognized in terms of syntactic properties: disjoint appearances of predicates, and disjoint appearances of individuals (terms). As part of this, we also *definitionally* reformulate the global axioms, one by one, in addition to applying decomposition. We identify a broad class of prioritized default theories, *generalizing default inheritance*, for which our results especially bear fruit. For this *asocially monadic* class, decomposition permits reasoning to be *localized* to individuals (ground terms), and reduced to *propositional*. Our reformulation methods are implementable in *polynomial time*, and apply to several other NM formalisms beyond circumscription.

Introduction

Large-Scale, Expressively Rich, Learning Agents: We aim in this work[1] to help build agents that do large scale, expressive non-monotonic reasoning (NMR). We are interested especially in what we call *learning agents*: automatic programs that store, and revise, a body of conclusions while continually acquiring new, possibly defeasible, premise beliefs.

In many applications, information about which defaults take precedence over others (have greater prioritization) is important and available. [2] Many applications need the ability to express fairly arbitrary first-order forms of default beliefs (e.g., induction, law, natural language, communication), as well as fairly arbitrary (finite) partial orders of precedence (e.g., specificity, reliability, and authority are not "layered" (a.k.a. "stratified"))). [Grosof, 1991] defines and discusses the importance of non-layered priority. Non-layered priority is needed, for example, to adequately represent *default inheritance*.

In these applications, we regard as desirable for many reasons, especially validation (both intuitive and rigorous), that a NM formalism be "expressively rich" not only in the above senses, but also that it be equipped with a relatively strong model-theoretic semantics (e.g., cf. Default Logic [Reiter, 1980], circumscription [McCarthy, 1986] [Lifschitz, 1984], and Autoepistemic Logic [Moore, 1985]). In this connection, we also are interested in skeptical or cautious, rather than credulous or brave, entailment.

Current Incapabilities: Currently, expressively rich NMR[3] has found virtually no application on a large scale (more than order of ten defaults), except for the rather special cases of Prolog-style logic programs and simple inheritance cf. AI frame-based systems.

Part of the problem is that there do not yet ex-

[1] part of forthcoming PhD dissertation [Grosof, 1992b]

[2] Note, however, that most of the discussion and results, e.g., about disjoint describability and definitional reformulation and asocially monadic theories, in this paper also apply to the basic case where there are only two "priority levels": for-sure and defeasible.

[3] In [Grosof, 1992b], we make this more precise; here, let us just consider circumscription, Default Logic, and Autoepistemic Logic.

ist practical inference mechanisms to support storing and revising a limited body of conclusions as a working theory. Currently, for expressively rich NMR [4], the only procedures for *forward*[5] inference are *exhaustive*: they compute the entire non-monotonic theory (or, even worse, all credulous extensions). Also, currently, there are no procedures for performing *belief revision* on a body of conclusions, upon receiving new, asserted axioms (an update), beyond the exhaustive method of re-computing everything from scratch. (By "axiom", we mean a premise belief.) [6] By belief revision, we mean modifying the stored conclusions to retract those that are no longer entailed by the newly augmented axiom set.[7] By updating, we mean belief revision plus possibly the inference and storing of some additional conclusions.

Strategy and Summary: In this work, we attack these problems at the level of logical understanding (rather than, say, domain-dependent control of reasoning). Our analytic perspective is that a prime underlying difficulty in the tasks of inference and updating, as well as in specification, is the *logical globality* of NMR: in general, conclusions depend on the *whole* of the axiom set. The exhaustiveness of current methods is, in effect, a manifestation of their caution in dealing with (conflicting) interaction.

We define the concept of a *prioritized database*, using circumscription, as the logical representation of a learning agent that performs *sound*, but incomplete, expressively rich NMR. By database, we mean a subset of a (NM) theory. Prioritized circumscription meets our prime expressive concerns, offers mathematical convenience, and has inference procedures currently available.

We elaborate a detailed "divide and conquer" strategy. We develop concepts of, and results about, *structure* in prioritized circumscriptive theories, by showing how to reformulate them in a particular fashion: to be *conjunctively "decomposed"* hierarchically into a collection of smaller "part" theories, i.e., sub-theories which we call *slices*. We show that it is possible, and useful, to slice within slices. In this way, we map groups of axioms to groups of conclusions. We use the decompositions to analyze the interaction between defaults / parts in a NM theory. Much technical difficulty and trickiness arises from the expressive need to consider non-layered prioritization.

We give theorems that *localize* entailment and thus show how to make forward inference be *selective*, as well as *concurrent*. Exhaustive inference on a slice generates only a part of the global theory. Inferences *within* each slice (sub-theory) can be performed in parallel with inference *within* every other slice. All non-monotonic inference can be localized to the slices; only *monotonic* inference is required between the slices. We give theorems that localize retraction and thus show how to make belief revision be *partial* in the sense that, for a given update, the arena of potential retraction is known to be restricted to a particular part of the previous database.

Our results enable the exploitation of other results on inference and belief revision that are limited to expressive special cases, say to do exhaustive forward inference in polynomial time (e.g., the "sympathetic-solitary" case in [Grosof, 1992b] that generalizes predicate completion [Clark, 1978] and the Closed World Assumption). These special case results can be applied to one, or several, slices, even when they do not apply to the global theory.

Our results are about well-behaved special cases that are easily recognized in terms of syntactic properties. The first *"cleanly slice-able"* property is disjointness of mentioned predicates. We show that if the for-sure and default axioms can be partitioned into groups which are disjoint in terms of the predicate symbols they mention, then non-monotonic inference based on each partition can proceed without considering the axioms in the other partitions: those other axioms are irrelevant in an important sense, as far as that partition is concerned. We show this implies that updating with new for-sure and default axioms that span only some of the previous partitions does not require retracting previous conclusions based purely on the remaining partitions: they are *safe*.

Most large practical applications, however, do not display such perfect partitionability of mentioned predicates. The real power from our result about disjointness of predicates comes when it is *combined* with another kind of reformulation: of the axioms in a given global axiom set, not just of the global axiom set into decomposed constituent axiom sets. We define a concept of *disjoint describability*: syntactic partitionability *after* definitional reformulation of the axioms. As part of this, we give a logical *definition* of a particular kind of definitional (i.e., equivalence-preserving) reformulation with respect to a background theory, modifying the standard logical idea of a conservative extension. We also discuss, and use, another kind of reformulation: to break up open defaults (i.e., schema-type, as opposed to closed, i.e., propositional) into cases. An important difference from definitionally reformulating monotonic theories is that two default axioms $D1$ and $D2$ cannot, in general, be equivalently replaced by the default axiom corresponding to the conjunction $D1 \wedge D2$ the way that two for-sure axioms can always be equivalently replaced by their conjunction $B1 \wedge B2$. This is why we need to consider reformulation of the

[4] including even the propositional special case and the special case of stratified logic programs with negation [Lifschitz, 1987] [Przymusinski, 1988]

[5] bottom-up. By "backward", we mean totally goal-directed cf. query-answering.

[6] NM formalisms, e.g., JTMS's [Doyle, 1979], having such procedures lack our desired expressive properties.

[7] For simplicity, we assume that these are the only ones removed from storage.

axioms one-by-one.

Using these definitional and default-cases reformulations, we arrive at our second cleanly slice-able, yet syntactically recognizable, property: disjointness of mentioned individuals. We show that a fairly broad class (*"asocially monadic"*) of prioritized default circumscriptions is cleanly slice-able into one slice theory per named individual (ground term in the language) plus a remainder-case slice. Each of these individual-wise slices is propositional, and is, essentially, much simpler than the global, in several ways: number of axiom instances (especially, potential primitive default conclusions), availability of inference methods, and availability of known computational complexity results. (Unfortunately, we do not have space to discuss NM inference methods and complexity results in detail. But see the final section.) The asocially monadic class **includes, as a special case, default inheritance networks** of the kind studied by [Touretzky, 1986] and used in many AI frame-based systems. The asocially monadic class is more general, however: it permits more than one antecedent in default rules, free use of negation, and freer use of disjunction.

While definitional reformulation is hard in general, we have a **polynomial-time algorithm** (omitted in this draft to save space and preserve focus) to perform the recognition and exploitation of this asocial-monadic reformulation and decomposition. More precisely, the algorithm is $O(n^3)$, where n is the size of the (global) axiom set (which is, moreover, typically much smaller than the whole theory, of course).

We show that our conjunctive decomposition results imply safeties in belief revision. We illustrate the problems of scale in learning agents with an extended example of a prioritized database and show that our safety theorems capture much of the preponderant stability (i.e., most beliefs are preserved after each update) that this database displays through its sequence of updates. We show, using the example, that decompositions on these two bases combine *synergistically*, as well as *hierarchically*: it is useful to slice within slices.

Finally, we observe that our formal reformulation methods are implementable at reasonable cost, and apply to several other NM formalisms. We have **polynomial-time algorithms** (again, omitted here due to space and focus) for disjoint predicates, as well as for asocially monadic, also in $O(n^3)$, where n is the size of the (global) axiom set.

A Motivating Example

Next, we give an extended example of a learning agent, in the domain of common-sense default reasoning, that illustrates issues of selective forward inference and partial belief revision on a large scale. We present it first at an intuitive level, and formalize it later.

We adopt the following notation. A •> prefix indicates that the sentence that follows is a *base* axiom, i.e., has for-sure (non-defeasible) belief status. A :> prefix indicates that the formula that follows is a *default* axiom (roughly, a normal default without pre-requisite in Default Logic). Its label, e.g. $(d1)$, serves as a tag for defining prioritization-type precedence between defaults via $\mathcal{PREFER}$ (*prioritization*) axioms. These define a strict partial order of precedence, via transitive closure. $\mathcal{PREFER}(d1, d2)$, for example, means that the default axiom with label $(d1)$ has strictly greater precedence (priority) than the default axiom with label $(d2)$.

We make the Uniqueness of Names Assumption (consider it included as a for-sure axiom). As a shorthand for conjunctions of for-sure assertions of positive or negative literals, we list the satisfying objects, or, more generally, tuples. Often, in this context, we use "..." to indicate that there are additional satisfying tuples not shown explicitly; for simplicity's sake, we assume these objects are distinct from all other explicitly-shown objects.

In this example, the agent starts with no beliefs, then accumulates axioms by receiving updates. After each update, the agent draws a bunch of conclusions (say, ground first-order sentences), both monotonically and non-monotonically, and retracts some of its previous conclusions. Each U_i indicates an update, consisting of one or more axioms. Axioms are numbered. In addition, we show explicitly with $\models$ and $\not\models$ a few of the more interesting NM conclusions and retractions, respectively, about which discussion will revolve. Note that, by "conclusion", we always mean in the skeptical sense.

The first update consists of a default axiom, that bats have two legs, together with some for-sure axioms. Non-monotonic (default) conclusions include that known bats are two-legged. The second update consists of another, default axiom, that mammals have four legs, together with the precedence axiom, that this new default has lower priority than the previous, more specific one. The third update consists of two default axioms about emergency disaster situations, plus some associated for-sure information. Intuitively, since the axioms in this new update are about a totally different topic than the previous axioms, they should not result in having to retract any of the previous conclusions. Moreover, intuitively, the agent should be able to draw the conclusions from these new axioms without even having to consider the previous ones in detail.

The fourth update consists of some for-sure information about two named individuals, *Joe* and *Spot*, that violates some previous default conclusions. Intuitively, since there is no information that "connects" any other named individuals to *Joe* and *Spot*, these new axioms should not result in having to retract any of the previous conclusions that are *not* about those named individuals: e.g., that are about some other named individuals. For example, the previous default conclusion $2legs(Betsy)$ should not have to be retracted.

Later, we will show how to capture these intuitions

EXAMPLE'S AXIOMS AND SAMPLE CONCLUSIONS

$\mathcal{U}_1$: **Mammals Taxonomy plus: Bats are Two-Legged**

[1] $\bullet\!\!>\ \forall x.\ bat(x) \supset mammal(x)$

[2] $\bullet\!\!>\ \forall x.\ dog(x) \supset mammal(x)$

[3] $\bullet\!\!>\ +bat:\ Betsy, Joe, June, Jackie, \ldots$

[4] $\bullet\!\!>\ +dog:\ Fido, Spot, Siccem, Jumper, \ldots$

[5] $\bullet\!\!>\ \forall x.\ \neg(2legs(x) \wedge 4legs(x))$

[6] $(d1) :>\ bat(x) \supset 2legs(x)$

$\&_i\mathcal{U}_i \quad \not\approx\ 2legs(Betsy) \wedge 2legs(Joe) \wedge \ldots$

$\mathcal{U}_2$: **Lower-Priority Default about Legged-ness**

[7] $(d2) :>\ mammal(x) \supset 4legs(x)$

[8] $\mathcal{PREFER}(d1, d2)$

$\&_i\mathcal{U}_i \quad \not\approx\ 4legs(Fido) \wedge 4legs(Spot) \wedge \ldots$

$\mathcal{U}_3$: **Emergencies (cf. [Grosof, 1991])**

[9] $(d3) :>\ fire(place, day) \wedge person(x) \supset leave(x, place, day)$

[10] $(d4) :>\ earthquake(place, day) \wedge person(x) \supset leave(x, place, day)$

[11] $\bullet\!\!>\ +person:\ Sue, Andy, Ed, Peg, Maggie, Eileen, Chang, \ldots$

[12] $\bullet\!\!>\ +fire:\ \langle Baltimore, 2/4/03\rangle, \langle Watts, 8/2/67\rangle, \ldots$

[13] $\bullet\!\!>\ +earthquake:\ \langle SF, 4/8/06\rangle, \langle MexicoCity, 5/3/87\rangle, \ldots$

[14] $\bullet\!\!>\ \forall x, place, day.\ leave(x, place, day) \supset \neg attend\_work(x, place, day)$

$\&_i\mathcal{U}_i \quad \not\approx\ leave(Sue, SF, 4/8/06) \wedge leave(Andy, Watts, 8/2/67) \wedge \ldots$

$\mathcal{U}_4$: **Legged-ness: Selective Defeat For Individuals**

[15] $\bullet\!\!>\ \neg 2legs(Joe) \wedge \neg 4legs(Joe) \wedge \neg 2legs(Spot) \wedge \neg 4legs(Spot)$

$\&_i\mathcal{U}_i \quad \not\approx\ 2legs(Joe) \quad ; \quad \not\approx\ 4legs(Spot)$

$\mathcal{U}_5$: **Work Attendance (cf. [Grosof, 1991])**

[16] $(d6) :>\ weekday(d) \wedge reg\_employ(person, place) \supset attend\_work(person, place, d)$

[17] $(d7) :>\ flu(person, day) \supset \neg attend\_work(person, place, day)$

[18] $\mathcal{PREFER}(d7, d6)$

[19] $\mathcal{PREFER}(d3, d7)\ \&\ \mathcal{PREFER}(d4, d7)\ \&\ \mathcal{PREFER}(d5, d7)$

$\mathcal{U}_6$: **Ed is Ill; Conflict Resolved by Prioritization (cf. [Grosof, 1991])**

[20] $\bullet\!\!>\ +weekday:\ Today, 11/12/91, \ldots$

[21] $\bullet\!\!>\ +reg\_employ:\ \langle Ed, BldgA\rangle, \ldots$

[22] $\bullet\!\!>\ +flu:\ \langle Ed, Today\rangle, \ldots$

$\&_i\mathcal{U}_i \quad \not\approx\ \neg attend\_work(Ed, Today)$

$\mathcal{U}_7$: **Miscellany: Meetings and Attendance (cf. [Grosof, 1991])**

[23] $\bullet\!\!>\ +Tuesday:\ Today, 11/12/91, \ldots$

[24] $\bullet\!\!>\ +in\_group(p, 4321):\ Ed, Peg, Maggie, \ldots$

[25] $\bullet\!\!>\ \forall person.\ in\_group(person, 4321) \supset reg\_employ(person, BldgA)$

[26] $\bullet\!\!>\ \neg vacation(Boss(4321), d):\ Today, \ldots$

[27] $\bullet\!\!>\ \forall p, d.\ group\_meeting(p, d) \wedge in\_group(p, 4321) \supset attend\_work(p, BldgA, d)$

$\mathcal{U}_8$: **Group Meetings; Non-Layered Conflict (cf. [Grosof, 1991])**

[28] $(d9) :>\ in\_group(p, 4321) \wedge Tuesday(d) \supset group\_meeting(p, BldgA, d)$

[29] $(d10) :>\ in\_group(p, 4321) \wedge vacation(Boss(4321), d) \supset \neg group\_meeting(p, BldgA, d)$

[30] $\mathcal{PREFER}(d10, d9)$

[31] $\mathcal{PREFER}(d3, d10)\ \&\ \mathcal{PREFER}(d4, d10)\ \&\ \mathcal{PREFER}(d5, d10)$

$\&_i\mathcal{U}_i \quad \not\approx\ \neg attend\_work(Ed, Today) \quad ; \quad \not\approx\ attend\_work(Ed, Today)$

as formal guarantees.

Formal Definitions: Prioritized Circumscription

We **define** our notation for axioms from section 2 as a meta-language (the Circumscriptive Language of Defaults, or CLD for short) that, at any point in the update sequence, specifies a *prioritized "default" circumscription* of the form:

$$PDC(B; D; R; fix\ W; Z) \stackrel{\text{def}}{\equiv} B[Z] \wedge \neg\exists Z'.\ B[Z'] \wedge Z \prec_{(D;R)} Z' \wedge W = W'$$

Here, B is the conjunction of the sentence parts of all of the for-sure axioms. D is the tuple of the default axioms' formula parts. R is a strict partial order of precedence (priority). It is the transitive closure of the precedence relation specified by the pairwise comparisons in the $\mathcal{PREFER}$ axioms. Its domain, accordingly, is the set of default axiom labels. Z is the tuple of all mentioned predicate symbols; e.g., in the example, $\langle bat, dog, mammal, 2legs, 4legs, fire, \ldots\rangle$. $W \subset Z$ is the tuple of predicates that are *fixed*. Fixing is a standard notion in the circumscription and non-monotonic reasoning literature. Fixing is part of the specification of non-monotonic reasoning. Intuitively, fixing some symbols implies that any formula that mentions only those symbols is *immune* to the circumscription operation in the sense that it can be concluded non-monotonically, i.e., from the circumscription, only if it can be concluded "monotonically", i.e., from the for-sure axioms B alone. For simplicity, we also *fix* (do not vary and second-order quantify over) all function symbols. This assumption can easily be relaxed. This assumption is typical in the circumscription literature. Uniqueness of Names, plus Domain Closure, implies that functions are effectively fixed, for example. For the sake of simplicity, in this paper, we for the most part do not consider fixing of predicates, only of functions: W is empty. We omit further details about fixing to save space and to preserve focus; see [Grosof, 1992b] for more.

Prioritized default circumscription is a slight generalization of prioritized predicate circumscription cf. [Grosof, 1991]. We employ it and CLD to clarify the definitions of axiom sets and of updating, and the intuitive relationship to other formalisms for default reasoning. [Grosof, 1992b] shows as a **theorem** the equivalence of any prioritized default circumscription to a corresponding, abnormality-style, prioritized predicate circumscription, generalizing a previous result that appeared in [Lifschitz, 1984]. Note that our definition can express minimizing predicates as a special case: e.g., $:> ab_i(x)$, where ab_i is an abnormality predicate.

We let N stand for the index tuple of D: it is just (isomorphic to) the tuple of the labels of the default axioms. I.e., in the example, after the second update, the elements of $D[Z]$ are:

$$\lambda x.\ bat(x) \supset legs2(x)\ ,$$

$$\lambda x.\ mammal(x) \supset 4legs(x)$$

and $N = \langle d1, d2\rangle$. $R(j,i)$ means that the default with label j has strictly higher priority than the default with label i. $\prec_{(D;R)}$ is defined as the strict version $(\preceq_{(D;R)} \wedge \neg\succeq_{(D;R)})$ of the prioritized "formula" preorder $\preceq_{(D;R)}$:

$$Z \preceq_{(D;R)} Z' \stackrel{\text{def}}{\equiv} \forall i \in N.\ [\forall j \in N.\ R(j,i) \supset (\forall x.\ Dj[Z,x] \equiv Dj[Z',x])] \supset (\forall x.\ Di[Z,x] \supset Di[Z',x])$$

Here Dj and Di refer to the j^{th} and i^{th} members, respectively, of the tuple D.[8] We **define** the corresponding circumscriptive prioritized default *theory* as the set of all conclusions entailed (model-theoretically, in second-order logic) by the prioritized default circumscription. [9] [10] We **define** a *prioritized database* (PDB) to be a pair, consisting of a CLD axiom set $\mathcal{A}$ (in the example, the current collection of the updates $\mathcal{U}_i$'s)); and an associated *prioritized database theory* $\mathcal{DB}$, which is some subset of the prioritized default circumscriptive theory $\mathcal{C}(\mathcal{A})$ specified by $\mathcal{A}$. Here, $\mathcal{C}$ is the non-monotonic theory operator for the CLD formalism.

Decomposition: Concepts

As part of our strategy, we need to develop a strong idea of a *part* of a non-monotonic theory. This is important for several reasons: 1) to define safe versus unsafe zones for belief revision; 2) to define relevant versus irrelevant context for inference (and for specification); and 3) to define the structure and organization of an overall ("global") prioritized database. In classical logic, we take for granted such an idea of a part of theory. However, the dependence of entailment on, in general, the entire *global* axiom set means that we have to "work for it" in NM logical systems.

Our general concept of decomposition is applicable to many NM logical systems. A global theory $\mathcal{T}$ can be obtained *either* directly by applying the NM theory operator $\mathcal{C}$ to the global axiom set $\mathcal{A}$, *or* indirectly (but equivalently) via decomposition. In decomposition, the global axiom set $\mathcal{A}$ is *decomposed* into an associated set of "constituent" axiom sets (the $\mathcal{SA}_i$'s). The global theory $\mathcal{T}$ is then equivalent to the *combination* of the corresponding sub-theories (the $\mathcal{ST}_i$'s), where each sub-theory is the result of applying $\mathcal{C}$ to a constituent axiom set: $\mathcal{ST}_i \stackrel{\text{def}}{=} \mathcal{C}(\mathcal{SA}_i)$.

[8] For notational simplicity, we ignore the potentially different arities of the various open formulas Di.

[9] See [Grosof, 1991] and [Grosof, 1992b] for more discussion of how prioritized circumscriptions are defined. Note that the prioritization p.o. R is not necessarily layered (stratified) (indeed, in our example, it is not) as it was in [Lifschitz, 1985].

[10] In section 5, we generalize the definition above to include the explicit "fixing" of a set of formulas, e.g., a subset of the predicates. [Grosof, 1992b] gives details.

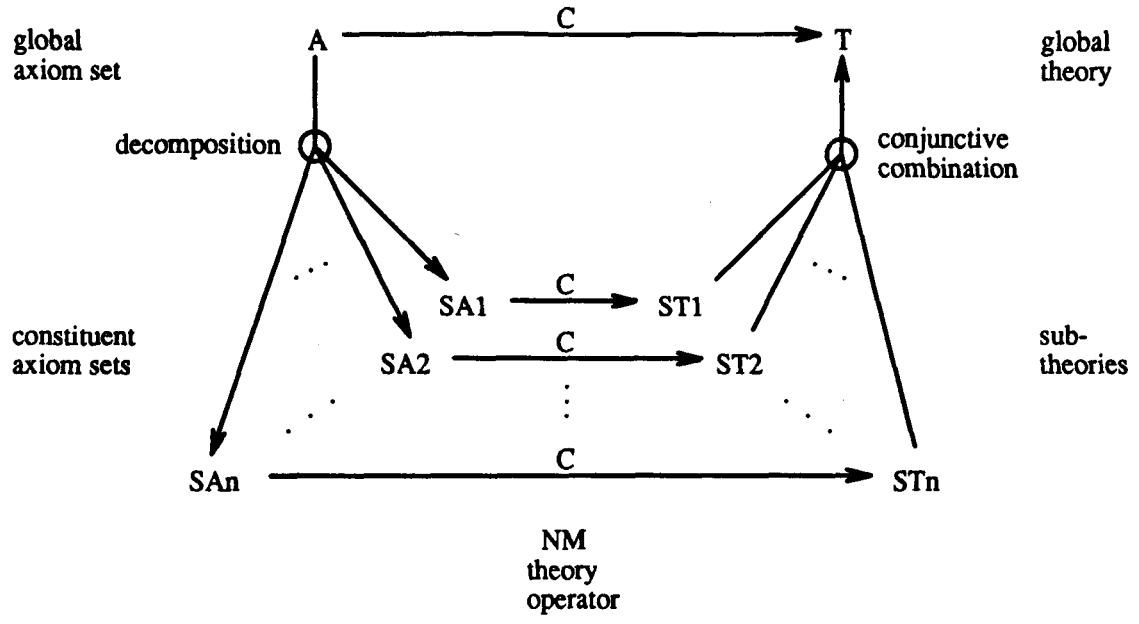


Figure 1: Conjunctive Decomposition: a conceptual flow diagram. A global theory $\mathcal{T}$ can be obtained *either* directly by applying the NM theory operator $\mathcal{C}$ to the global axiom set $\mathcal{A}$, *or* indirectly (but equivalently) via decomposition. In decomposition, the global axiom set $\mathcal{A}$ is *decomposed* into an associated set of *constituent* axiom sets (the $\mathcal{SA}_i$'s). The global theory $\mathcal{T}$ is then equivalent to the *conjunctive combination* of the corresponding sub-theories (the $\mathcal{ST}_i$'s), where each sub-theory is the result of applying $\mathcal{C}$ to a constituent axiom set: $\mathcal{ST}_i \stackrel{\text{def}}{=} \mathcal{C}(\mathcal{SA}_i)$.

In CLD, we **define** $\mathcal{T}$ to be the result of *conjunctive* combination when $\mathcal{T}$ is $Cn(\bigcup_{i=1,\ldots,n} \mathcal{ST}_i)$; where Cn is the *monotonic* consequence (theory) operator in classical logic. When the corresponding axiom sets are understood, we will say that the global *theory* is conjunctively decomposable into these *slice* sub-theories. [11]

In terms of the circumscriptions, we have:

$$\mathcal{PDC}(\mathcal{A}) \equiv \bigwedge_{i=1,\ldots,n} \mathcal{PDC}(\mathcal{SA}_i)$$

Again, when the corresponding axiom sets are understood, we will also speak of a *circumscription* being conjunctively decomposable into slice circumscriptions, e.g., for $n = 2$:

$$PDC(B; D; R; Z) \equiv PDC(SB1; SD1; SR1; Z) \wedge PDC(SB2; SD2; SR2; Z)$$

Figure 1 illustrates conjunctive decomposition with a flow diagram.

Conjunctive decomposition is thus a kind of* reformulation *or representation change. The global axiom set and theory $\langle A, T\rangle$ are transformed into a collection of constituent axiom sets and slice sub-theories: $\langle\langle \mathcal{SA}_1, \mathcal{ST}_1\rangle, \ldots, \langle \mathcal{SA}_n, \mathcal{ST}_n\rangle\rangle$.

Most Subsets Do Not Qualify As Constituents for Decomposition: Note that, in general, in non-monotonic reasoning, one cannot blithely partition a global axiom set into a bunch of (distinct, or, more generally, overlapping) subsets (whose union is the global axiom set) any old way and get a conjunctive decomposition. This is because the axioms in one subset may conflict with those in another.

E.g., consider the classic Quaker-Republican example of conflict in default reasoning: there are two default axioms, one saying that Quakers are typically Pacifists, and another saying that Republicans are typically non-Pacifists. In addition, there are two for-sure axioms: that Nixon is a Quaker, and that he is a Republican. Suppose we consider two subsets: one containing the Quaker axioms, and another containing the Republican axioms. Treating a subset as a constituent axiom set means drawing non-monotonic conclusions from it as if there were no other axioms around. Doing so, from the first (with Quaker) one gets the default conclusion that Nixon is a Pacifist; from the second, one gets the default conclusion that Nixon is a non-Pacifist. Taking the conjunction of these two "sub-theories" thus results in garbage: inconsistency. Yet the actual global theory is consistent: neither conclusion about Pacifism is sanctioned. Figure 2 illustrates.

[11] *Serial* combination has the flavor of a cascade: there is a series of phases of adding axioms and drawing conclusions, where the previous stage's conclusions are treated as for-sure. Many NM inference procedures can be described in this manner. Details about serial decomposition are omitted due to considerations of space and focus. See [Grosof, 1992b] for more.

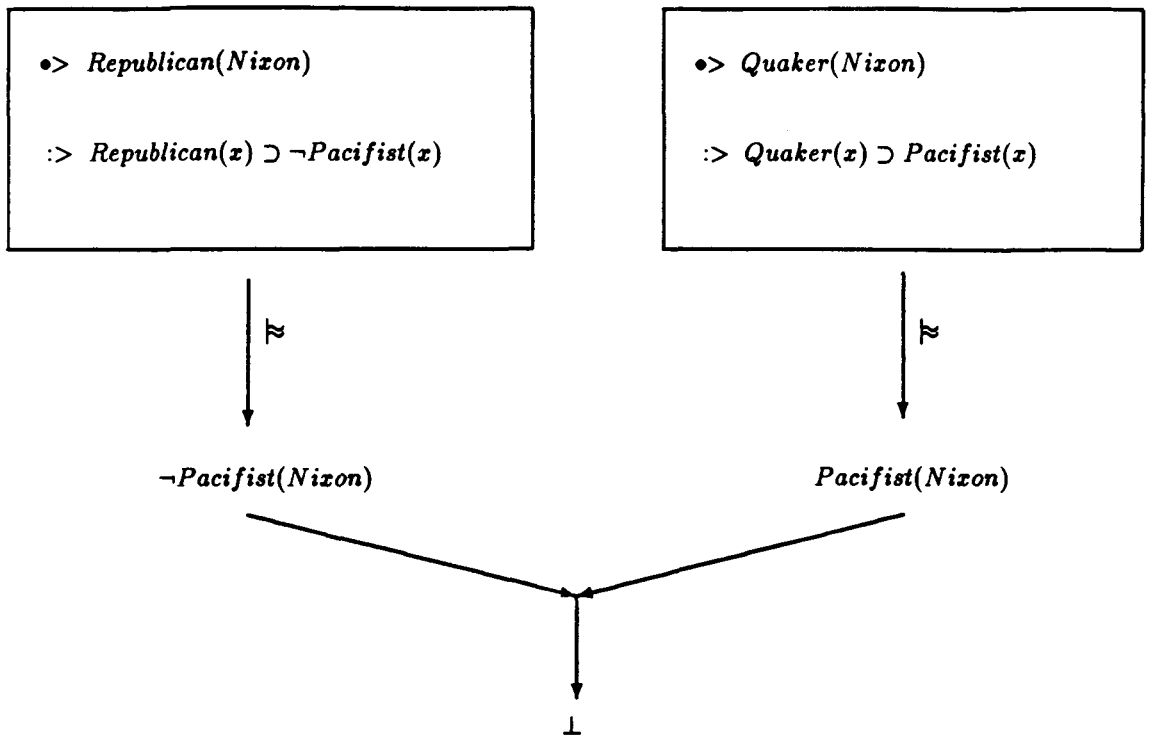


Figure 2: Non-modularity: Quakers and Republicans. (Default axiom labels not shown.)

Our perspective is that, in general, non-monotonicity means a kind of **logical non-modularity**: when attempting to draw conclusions from a subset of the global axiom set, one must keep in mind the context of the remainder of the global axiom set. If one considers that remainder as an "internal" update, then that update may be non-monotonic. Another way to view this situation is that non-monotonicity means **logical globality**: in general, a non-monotonic conclusion cannot be drawn until the *entirety* of the global axiom set is considered.

Locality:
Suppose we can find a conjunctive decomposition in which for some i, the slice's axiom set is a subset of the global, i.e., $\mathcal{SA}_i \subseteq \mathcal{A}$. In this case, we say that the slice is a *clean* slice. Then we know that all the remaining axioms $(\mathcal{A} - \mathcal{SA}_i)$ in the global axiom set are *irrelevant context*, in an important sense, relative to the slice's axiom set $\mathcal{SA}_i$. In this case, one can soundly, and in an important sense completely, perform inference *locally*: considering only the axioms in $\mathcal{SA}_i$, and using whatever standard procedures are available generally for the NM formalism. This is sound, because $\mathcal{C}(\mathcal{SA}_i)$ is then a subset of the global theory. This is complete, in a sense, because the contribution of $\mathcal{SA}_i$ to the global consequences requires only *monotonic* inference beyond its own local (NM) consequences $\mathcal{C}(\mathcal{SA}_i)$. By "irrelevant" above, then, we mean that one does not need to consider the remainder of the global axioms in order to do the essential non-monotonic aspect of the reasoning from $\mathcal{SA}_i$.

In the rest of this paper, we will be only considering decompositions that are clean. ([Grosof, 1992b], however, discusses the usefulness of decompositions that are not clean, e.g., decompositions on the basis of higher versus lower priority.)

Observe that in clean slicing, the constituent axioms sets are each smaller, and thus simpler, than the global axiom set. In prioritized default circumscription, and in other expressively rich NM formalisms, the computational complexity of non-monotonic reasoning (including, full forward inference and belief revision) is worse than monotonic reasoning. Non-monotonic reasoning (full forward inference and belief revision) in each slice, and via monotonic conjunctive combination, is thus **computationally less complex** than non-monotonic reasoning in the global theory.

Partitioning Axioms As Kind of Reformulation:
Our perspective, therefore, is that, in non-monotonic reasoning, decomposing, e.g., partitioning (see Theorems 1 and 12), a global axiom set into constituent axiom sets is a quite non-trivial kind of reformulation. This is very different from the situation in classical

monotonic reasoning.

Safeties of Updating:
Suppose, in a conjunctive decomposition, that $\{\mathcal{SA}_1, \ldots, \mathcal{SA}_k\}$ are present both before and after an update $\mathcal{U}$. I.e., suppose that some of the constituent axiom sets in a decomposition after an update $\mathcal{U}$ are unchanged from (i.e., are the same as) in a decompositon before that update. Then we know that all of the conclusions in the conjunctive combination of their associated slices are safe under the update.

Hierarchy:
We can view the conjunctive combination of a set of slice sub-theories as being, in turn, a sub-theory. When those slices are clean, then this sub-theory is itself well-defined as a clean slice: its axiom set is simply the union of those slices'. Thus we can often choose grain size hierarchically during conjunctive decomposition.

Sequencing of Inference: See section 1 about concurrency.

Disjoint Predicates

Our results will all make use of the following idea of decomposing the specified prioritization.

Composing Prioritization:
The concept of prioritization over groups of defaults is natural in the specification process for many applications: often a group of defaults corresponds to a topic. [Grosof, 1991] introduced, and [Grosof, 1992b] elaborates, this idea of *"composing"* prioritization, in which an overall prioritization p.o. R over the domain of individual defaults is equivalent to the result of composing an *external* prioritization p.o. RE, defined over groups, with a tuple RI of prioritization p.o.'s, one (RIi) per group, that each represent the prioritization *internal* to that group: $R = RE{*}RI$. Groups may, in turn, be composed of groups. Thus we may define prioritizations of prioritizations, in hierarchical or recursive fashion. Our example displays this structure.

Our first result is about decomposition on the basis of syntactic disjointness of predicates. It captures a basic case of the intuition that syntactically "having nothing to do with each other" should imply strong irrelevance of the kind we discussed in the last section.

Theorem 1
(Clean Decomposition, given Disjoint Predicates)
Let $PDC(B; D; R; Z)$ be a global PDC. Let $\{B1[Z1], \ldots, Bk[Zk]\}$ be a partition of the base axioms $B[Z]$, and let $\{D1[Z1], \ldots, Dk[Zk]\}$ be a partition of the default formulas $D[Z]$, where the predicate tuples $Z1, \ldots, Zk$ are a (disjoint) partition of Z. I.e., in terms of CLD, let there be a partition, of the base and default axioms, where the predicates mentioned in each element of the partition are disjoint. If a certain condition (0) (see below) on the prioritization R is satisfied, then

$$PDC(B; D; R; Z) \equiv \bigwedge_{j=1}^{k} PDC(Bj; Dj; RIj; Z)$$

(Note that the Z on the right-hand side can be equivalently replaced by Zj.) Condition (0) is defined as: either, R is the composition of some prioritization RE with the tuple RI of the internal prioritizations of each partition; or, R is layered (stratified). The composition condition for non-layered R corresponds, intuitively, to a kind of a partitionability of the prioritization. Note the special case of empty R satisfies (0).

Proof Overview: Surprisingly non-trivial. The essence is to use the ability to separate existential quantifiers in the right-hand-side part of the circumscription formula (cf. section 3). Non-layered prioritization makes this tricky: hence the prioritization conditions in the theorem. □

In terms of CLD, Theorem 1 tells us that syntactic disjointness implies *irrelevance* in the sense that we discussed in the last section; the decomposition by syntactic partition is a clean slicing.

Theorem 1 immediately yields a powerful result about inference.

Theorem 2
(Locality of Inference, given Disjoint Predicates)
In Theorem 1, each slice j is sound and complete, relative to the global theory, for inference over its corresponding sub-language (partition of the predicates). That sub-language consists of the formulas that mention only the predicates Zj. This locality holds both for forward inference, and for backward inference (query-answering). Note that to perform inference using any subset Y of the predicates Z, one need only work in the conjunctive combination of those slices whose predicates cover that subset Y.

Theorem 1 also immediately yields a powerful result about belief revision.

Theorem 3
(Safety of Updating, given Disjoint Sub-Languages)
In CLD, let the previous axiom set be partitionable according to Theorem 1. Let an update $\mathcal{U}$ consist of base, default, and prioritization axioms, such that the formula parts of the base and default axioms mention only predicates from a (possibly empty) subset of the previous partitions, and such that the global prioritization condition (0) is still met. Then all of the previous conclusions derived solely from the rest of the partitions' slices do not require retraction.

Application to Main Example:
The above theorems capture the first intuition that we discussed in section 2. At each point in the sequence of updates, Theorem 2 implies that inference can be localized: inferences about legged-ness can be performed in the slice that contains only the axioms about leggedness, and likewise for meetings. Figure 3 illustrates the

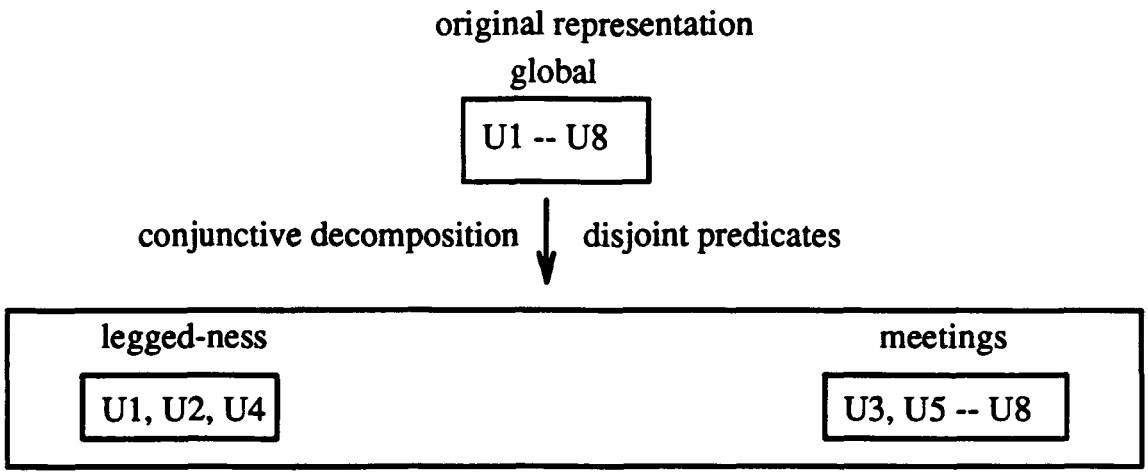


Figure 3: Conjunctive Decomposition using Disjoint Predicates: In our main motivating example (section 2), we conjunctively decompose the global axiom set (after the last update U_8) into two slices by employing the disjoint predicates result (Theorem 1): one slice about legged-ness, and the other slice about meetings. In the bottom half, each inner box stands for a constituent axiom set.

conjunctive decomposition cf. Theorem 1 after the last update. Theorem 3 guarantees that after each meetings update, all of the previous conclusions drawn from the legged-ness slice are safe, and vice versa.

Generalizations:
Theorems 1, 2, and 3 generalize in several directions. Firstly, predicate (and function) symbols may overlap between the constituent axiom sets as long as they are fixed in the circumscription (see earlier discussion about fixing in section 3). Intuitively, it is OK to specify some predicate (and function) symbols as fixed if it is OK not to infer any default conclusions expressible purely in terms of those symbols. Secondly, the prioritization condition can be relaxed somewhat.

Definitional Reformulation of Axioms:
Thirdly, and most interestingly (see discussion toward end of section 1 about source of power), one can decompose with irrelevance (slice cleanly) as long as one can definitionally *reformulate* the global axiom set to meet the partitionability condition. (See Theorem 12.) One interesting such case is reasoning about one individual object, e.g., *Joe* in our example, at a time. (See Theorem 16.) Often (e.g., for the legged-ness axioms in our example), such re-formulability is easily (time polynomial in the number of axioms) detectable syntactically. We pursue all this in the next two sections.

Basic Definitional Reformulation of Axioms, One-by-One

Next, we define a particular kind of definitional reformulation. This kind of reformulation maps each formula in one formulation into a correspondent formula in another formulation, while preserving equivalence, i.e., without loss of information. Our motivation for considering this limited kind of reformulation is our intended application: to disjoint describability and its asocial-monadic special case. Why do we do the reformulation "one axiom at a time", i.e., *one-by-one*? Much of the reason is that there is an important difference between default / NM reasoning and monotonic reasoning.

We take for granted in monotonic logics that a collection of for-sure (base) axioms $B1, \ldots, Bm$ can be equivalently replaced by the axiom $B1 \wedge \ldots \wedge Bm$. In prioritized default circumscription and most other expressively rich NM formalisms, however, one cannot, in general, equivalently replace the pair of default axioms (whose default formulas are) $D1$ and $D2$ by the default axiom (whose default formula is the conjunction) $D1 \wedge D2$ (even in the case without priorities, i.e., when the prioritization is empty). Informational "grain size" of the defaults is important: having the two separate defaults means that, for example, $D2$ may "succeed" (i.e., be concluded non-monotonically from the defaults) even if $D1$ is "defeated" (e.g., is violated by the for-sure information), unlike if the only default present is $D1 \wedge D2$. We will need equivalence-preserving (and information-preserving) reformulation in order to apply the decomposition on the reformulated representation back onto the original representation.

Circumscription is defined in terms of second-order logic. We thus find it convenient and natural to define the kind of definitional reformulation we will need in terms of second-order logic, as well. We build on the standard idea of a conservative extension, drawn from the classical logical literature. In this and the next section, we then develop several, increasingly complex variations of definitional reformulations, in order to handle

the grouping structure in various stages of our reformulations: groups of predicates, groups of individuals, groups of formulas.

In this paper, we mainly address reformulations oriented around disjointness of mentioned (predicate and function) symbols. It is thus convenient to define our changes of representation in terms of changes in the symbols mentioned.

First Cut at Definitional Reformulation:
What does it mean to definitionally reformulate theories (or formulas) while preserving equivalence? At first glance, it simply means to introduce some definitions (of new symbols) which logically imply (entail) the equivalence of a theory expressed in an original set of symbols (i.e., an original representation) to a new theory expressed in those new symbols (i.e., a new representation). E.g., let $A1[P]$ be the original theory, let $U[P,Q]$ be some definitions of new symbols Q in terms of the old symbols P, e.g., a conjunction of explicit definitions:

$$U[P,Q] \stackrel{\text{def}}{\equiv} (Q1 \equiv E1[P]) \wedge \ldots \wedge (Qm \equiv Em[P])$$

(where m is the length of the tuple Q), and let $A2[Q]$ be a new theory that is equivalent to $A1[P]$ given $U[P,Q]$:

$$U[P,Q] \models A1[P] \equiv A2[Q]$$

More generally, we can permit the new representation to use some of the old symbols; let W be the overlap symbols between the old and the new. Suppose $A1[W,Y]$ is the original theory, $A2[W,Y,Z]$ is the new theory, and $U[W,Y,Z]$ is the (conjunction of) definitions of new symbols Z in terms of W and Y, e.g.,

$$U[W,Y,Z] \stackrel{\text{def}}{\equiv} (Z1 \equiv E1[W,Y]) \wedge \ldots \wedge (Zm \equiv Em[W,Y])$$

(where m is the length of the tuple Z); and suppose

$$U[W,Y,Z] \models A1[W,Y] \equiv A2[W,Z]$$

Then we call U a *"putative"* definitional *reformulator*.

Observation 4
(Subtlety: Uninformativeness and Consistency)
However, there is a subtlety. To us, part of the intuition behind the idea of an definitional reformulation is that the equivalence is non-spurious, i.e., that the definitions themselves are not introducing information. Unfortunately, merely requiring U to be a conjunction of explicit definitions allows spuriosity and informativeness.

Consider the following **example.** Let W be empty. Let $Y \stackrel{\text{def}}{\equiv} \langle Y1, Y2\rangle$, where $Y1$ and $Y2$ are 0-ary predicates.[12] Let $A1[W,Y]$ be defined as $Y1 \wedge \neg Y1$. Let the definitions U be $(Z1 \equiv Y1) \wedge (Z2 \equiv \neg Y1)$, where $Z1$ and $Z2$ are 0-ary predicates. Let $Z \stackrel{\text{def}}{\equiv} \langle Z1, Z2\rangle$, and let $A2[W,Z]$ be defined as $Z1 \wedge Z2$. Then U implies that $A1$ is equivalent to $A2$. Yet this contravenes our intuition of a reasonable definitional reformulation. $A1$ is inconsistent, i.e., is equivalent to $False$. $A2$, by contrast, is consistent.

Viewing the direction of reformulation from $A2$ to $A1$, in effect U is introducing some information, namely that $Z1 \equiv \neg Z2$. The source of this problem is that, even though U is a conjunction of explicit definitions, U is itself not always consistent when it is viewed in this "return direction" of the reformulation (i.e., from $A2$ to $A1$). Yet, to us, any notion of equivalence-preserving definitional reformulation ought to be *symmetric*, i.e., kosher in *both* directions: from $A1$ to $A2$ *and* from $A2$ to $A1$. We would, therefore, like to impose some kind of additional constraint on U to guarantee intuitive uninformativeness and non-spuriosity of the equivalence between the two representations. Below, we do this by formalizing U's consistency and its relationship to directionality more precisely.

The idea of a conservative extension, standard in the classical logical literature, provides a nice notion of uninformativeness in terms of mentioned symbols.

Definition 5 (Conservative Extension)
Let $A1[P]$ be a formula[13] mentioning only (the tuple of symbols) P. Let Q be (a tuple of symbols) distinct from P. Let $A2[P,Q]$ be a formula mentioning only $P \cup Q$. Then we say that $A2[P,Q]$ is a *conservative extension* of $A1[P]$ when:

$$\forall P.\ [(\exists Q.\ A2[P,Q]) \equiv A1[P]]$$

or, equivalently, when both:

$$\forall P, Q.\ A2[P,Q] \supset A1[P]$$

$$\forall P.\ [A1[P] \supset (\exists Q.\ A2[P,Q])]$$

Another way to view the idea of conservatism in this definition is that $A2$ "says" exactly as much about P as $A1$ does. $A2$ in addition says stuff about Q. I.e., for any formula $G[P]$ mentioning only P:

$$A2[P,Q] \models G[P] \iff A1[P] \models G[P]$$

Suppose that

$$A2[P,Q] \stackrel{\text{def}}{\equiv} A1[P] \wedge U[P,Q]$$

Then we say that $U[P,Q]$ is a *conservatively extending update* to $A1[P]$.

Intuitively, we can thus view a conservatively extending update $U[P,Q]$ as uninformative in a precise sense, namely about the old symbols P.

Notation:
Let $D \leq E$ stand for the universally quantified implication $\forall x.\ D(x) \supset E(x)$, where D and E are open formulas with the same arity of free variables (i.e., are similar), and x stands for a tuple of free individual (object) variables. Let $D{=}E$ then be defined analogously as the universally quantified equivalence $\forall x.\ D(x) \equiv E(x)$. We also apply this notation to tuples $D = \langle D_1, \ldots, D_m\rangle$ and $E = \langle E_1, \ldots, E_m\rangle$: e.g.,

[12] We do not use $Y2$ immediately, but we will use it later when we continue this example in the discussion after Definition 7.

[13] in (higher-order) classical logic

$D{=}E$ stands for
$$D_1{=}E_1 \wedge \ldots \wedge D_m{=}E_m \ .$$

Fact 6
(Explicit Definitions Are Conservative)
(Conjunctions of) explicit definitions of new symbols (e.g., predicates) are always conservatively extending updates. I.e., in Definition 5, suppose $U[P,Q]$ is a conjunction of explicit definitions of each symbol in Q:
$$Q \ = \ E[P]$$
(Here, we are using the tuple = notation introduced above, and applying it also to functions and terms.) Then $U[P,Q]$ is a conservatively extending update, for any $A1[P]$.

Conservative Extension, Uninformativeness, and Directionality:
Equipped with the idea of a conservative extension, we are now ready to return to the question of refining the basic idea of definitional reformulation. In our "first cut" above, we found a need to formalize the constraint that the putative definitional reformulator U be uninformative, in both directions of the reformulation. In Definition 5, we observed that the property that a "definitional" reformulator $U[P,Q]$ is a conservatively extending update precisely expresseses U's uninformativeness, in the direction of $A1$ to $A2$, i.e., about P. There, however, U is not really quite a reformulator in the sense we discussed in the "first cut", since $A2$ mentions not just the new symbols Q, but also the old symbols P. However, we can *extract* the notion of uninformativeness present there, i.e., the "conservatism" in the idea of a conservative extension.

The property that U is a conservatively extending update is: $\quad A1[P] \ \models \ \exists Q.\, U[P,Q]$
which we can also write as:
$$\models \ (\forall P.\, A1[P] \supset \exists Q.\, U[P,Q])$$
One can view the right-hand-side as a satisfiability (i.e., consistency) property. This satifiability / consistency is conditional on $A1$.

We take this conservativeness property as the basis for uninformativeness of a (putative) definitional reformulator U. However, we need the "return direction" uninformativeness as well:
$$A2[Q] \ \models \ \exists P.\, U[P,Q]$$
which we can also write as:
$$\models \ (\forall Q.\, A2[Q] \supset \exists P.\, U[P,Q])$$

Definition 7
(Definitional Reformulator — Basic Case)
We say that $U[W,Y,Z]$ is a *definitional reformulator* (basic case) between two formulas $A1[W,Y]$ and $A2[W,Z]$ (where W, Y, and Z are distinct tuples of symbols) when:

1. U implies the equivalence of $A1$ and $A2$:
$$\models \ U[W,Y,Z] \supset (A1[W,Y] \equiv A2[W,Z])$$
2. U is uninformative, i.e., conservative, in both directions of the reformulation, i.e., with respect to $A1$ and with respect to $A2$:
$$\models \ (\forall W,Y.\, A1[W,Y] \supset \exists Z.\, U[W,Y,Z])$$
$$\models \ (\forall W,Z.\, A2[W,Z] \supset \exists Y.\, U[W,Y,Z])$$

Discussion; Directionality:
Having the second direction, in addition to the first direction, of the conservativeness property in Definition 7 rules out the nastily-behaved example that we discussed in Observation 4. However, the conservativeness property in Definition 7 reassuringly does permit, for **example**, the following, more intuitively reasonable basic-case definitional reformulator:
$$U[W,Y,Z] \ \stackrel{\text{def}}{\equiv} \ (Z1 \equiv Y1) \wedge (Z2 \equiv \neg Y2)$$
(where the symbols are as in the example discussed in Observation 4) for any $A1$, $A2$.

The property that U consists exclusively of (a conjunction of) explicit definitions ensures, in general, only *one* direction of conservativeness.

Conditionality Versus Unconditionality of Conservativeness:
Definition 7 is perhaps too "custom" in one regard, however. The conservativeness property is conditional: it depends on the particular $A1$ and $A2$. This is perhaps unsatisfactory intuitively, at least for some purposes, as a notion of "definitional" in "definitional reformulator".

Alternative Definition of Conservativeness: Unconditional Version:
As an alternative definition of the basic case of definitional reformulator, we **observe** that one can use a stronger (i.e., more strongly constrained, special case) notion of conservativeness instead:
$$\models \ \forall W,Y.\, \exists Z.\, U[W,Y,Z]$$
$$\models \ \forall W,Z.\, \exists Y.\, U[W,Y,Z]$$
to replace the conservativeness property (2.) in Definition 7. This *"unconditional"* version of the conservativness property does *not* depend on $A1$ and $A2$: i.e., it implies that the *"conditional"* conservativeness property (2.) in Definition 7 holds for *any* $A1$ and $A2$.

Alternative Definition of Conservativeness: Backgrounded Version:
As an intermediate position between the conditional and unconditional versions of the conservativeness property, we **observe** that one can formulate conditionality in a somewhat abstracted fashion: in terms of the symbols W that are in common between the two representations. We will find it convenient for our later definitions to employ a notion of a *background* $G[W]$ to the reformulation. One can view $G[W]$ as, in effect, included in both $A1[W,Y]$ and $A2[W,Z]$. We then define the *"backgrounded"* version of the conservativeness property as:
$$G[W] \ \models \ \forall Y.\, \exists Z.\, U[W, Y, Z]$$
$$G[W] \ \models \ \forall Z.\, \exists Y.\, U[W, Y, Z]$$

In the remainder of this paper, we will use this last, "backgrounded" version of the conservativeness property. We do so in order to formally simplify our later definitions of more complex kinds of definitional reformulators and reformulations, which are oriented towards particular uses. However, the "conditional" version of the conservativeness property is more fundamental and general, we believe, and is interesting to explore: we plan to do so in the future.

No Requirement of Explicitness:
Note that in Definition 7, we did *not* require U to be in the form of a conjunction of explicit definitions of new symbols in terms of old symbols. *We formalized / summarized the "definitional" flavor of the reformulator as, simply, its conservativeness.* Our definition of definitional reformulator thus allows U to consist of implicit definitions (e.g., with recursion) and partial definitions (i.e., necessary and sufficient conditions). (Later, in our result about the asocial monadic special case of disjoint describability (Theorem 16), the reformulator will consist exclusively of explicit definitions, however.)

Next, we define a definitional reformulation of a group of formulas, using a single common reformulator: one-by-one, into a new group of formulas. For this purpose, it is convenient to be able to abstract away from conditionalizing conservativeness on each of those formulas: we thus use the backgrounded version of conservativeness.

Definition 8 (Group Reformulator)
Let $ET1[W,Y]$ and $ET2[W,Z]$ each be a similar[14] tuple of formulas; these formulas may be open or closed. We call each tuple a *group*. Let $U[W,Y,Z]$ and $G[W]$ be closed formulas. We say that $U[W,Y,Z]$ is a *group reformulator between* $ET1[W,Y]$ *and* $ET2[W,Z]$, *given the background* $G[W]$ when:

1. U is conservative (given the background) with respect to Y and also with respect to Z:
$$G[W] \models \forall Y.\, \exists Z.\, U[W,Y,Z]$$
$$G[W] \models \forall Z.\, \exists Y.\, U[W,Y,Z]$$
2. U reformulates each formula in either group into the corresponding formula in the other group. I.e., U implies the equivalence of corresponding member formulas (subscripted by j) in the two groups:
$$U[W,Y,Z] \wedge G[W] \models \forall j.\, ET1j[W,Y] = ET2j[W,Z]$$

Disjoint Describability and Disjoint Individuals

Next, we show how to use definitional reformulation to generalize the disjoint predicate special case: to the more general case of disjoint describability. More precisely, we use definitional reformulation to transform a disjointly describable global axiom set into a representation that has disjoint predicates, and then to transform back again after decomposition. Figure 4 illustrates. We show that the disjoint describability case, like disjoint predicate case, has a clean, partitioning conjunctive decomposition, which, moreover, implies interesting localities of inference and safeties of updating. We then identify an interesting special case of disjoint describability (asocial-monadic) that, like the disjoint predicate case, is easily recognizable in terms of the syntax of the starting global axiom set.

We begin with some preliminaries.

Definition 9 (Syndicate Reformulator)
We define a *syndicate* reformulator as a tuple of group reformulators that obeys an extra *syndication property*: their conjunction is also conservative.

More precisely: Let $ETT1[W,Y]$ and $ETT2[W,Z]$ each be a similar *tuple of tuples* of formulas; these formulas may be open or closed. Each element of the top level of tupling is itself a tuple of formulas cf. Definition 8. The top level tuple is thus a *syndicate* whose elements are groups of formulas.

Let $UT[W,Y,Z]$ be a tuple of closed formulas, of the same length as the top level tuples above. I.e., let it consist of one formula per group. Let $G[W]$ be a closed (background) formula, as in Definition 8.

Below, we use i to subscript groups, and j to subscript formulas within groups.

We say that $UT[W,Y,Z]$ is a syndicate reformulator between $ETT1[W,Y]$ and $ETT2[W,Z]$, given the background $G[W]$ when:

1. For each group i, UTi is a group reformulator between $ETTi$ and $ETT2i$ (given the background):
$$\forall i.\, UTi[W,Y,Z] \wedge G[W] \models \forall j.\, ETT1ij[W,Y] = ETT2ij[W,Z]$$
2. the conjunction $UC \stackrel{\text{def}}{\equiv} \bigwedge_i UTi$ is conservative (given the background) with respect to Y and also with respect to Z:
$$G[W] \models \forall Y.\, \exists Z.\, UC[W,Y,Z]$$
$$G[W] \models \forall Z.\, \exists Y.\, UC[W,Y,Z]$$

The reason we call the above a *syndicate* reformulation is the linkage between the different groups imposed by the conjunction's (UC's) conservative extension property. This implies, but is not implied by, the conjunction of the conservative extension properties for each group's reformulator UTi.

Definition 10
(Partitioning Syndicate Reformulator)
We say that a syndicate reformulator cf. Definition 9 is *Z-partitioning* when:

$$\forall i.\, UTi[W,Y,Z] \stackrel{\text{def}}{\equiv} Ui[W,Y,Zi]$$

$$\forall i,j.\, ETT2ij[W,Z] \stackrel{\text{def}}{\equiv} ETT2ij[W,Zi]$$

where $\forall j \neq k.\, Zj \cap Zk = \emptyset$, i.e., the appearances of the symbols Z are partitioned by group.

[14] **Terminology**: By "similar", we mean of same length, and with same arities for their members.

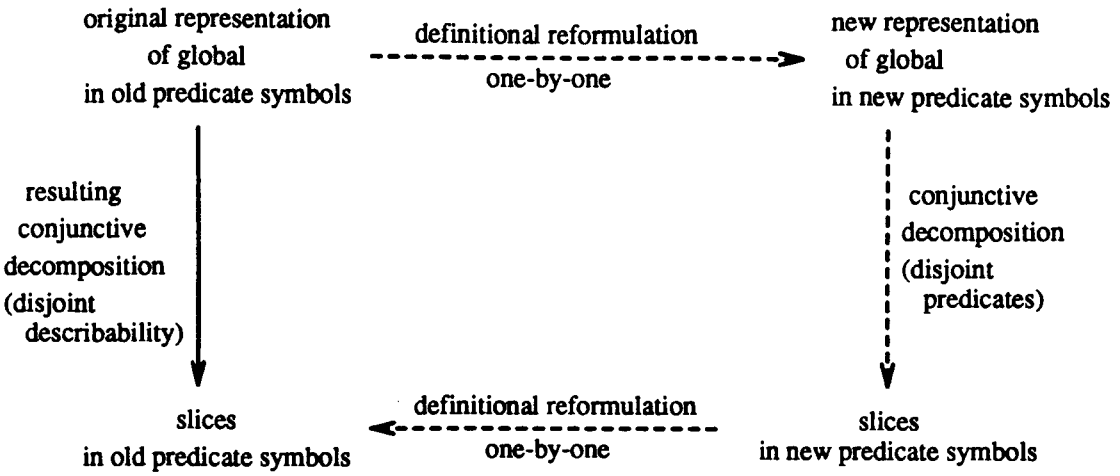


Figure 4: Disjoint Describability: a flow diagram of the reformulation steps involved.

Definition 11 (Disjoint Describability)
Suppose that UT is a Z-partitioning syndicate reformulator as in Definition 10, where for each group i, $ETT1i$ is defined as the concatenation of a (closed, base) formula $B1i$ with a tuple of (open, default) formulas $D1i$, and similarly, $ETT2i$ is the concatenation of $B2i$ and $D2i$.

Let $B1$ stand for the conjunction of the $B1i$'s; and $D1$ stand for the concatenation of the $D1i$'s. Let $B2$ and $D2$ be defined similarly.

Suppose also that $PDC(B2; D2; R; \mathit{fix}\ W; W, Z)$ fulfills the conditions in Theorem 1 (disjoint predicates), where the grouping, and the partition there on Z, is the same as in UT.

Then we say that $PDC(B1; D1; R; \mathit{fix}\ W; W, Y)$ is *disjointly describable* under (definitional) reformulation by $UT[W, Y, Z]$, given $G[W]$.

Theorem 12
(Clean Decomposition, given Disjoint Describability)
If a PDC is disjointly describable, then it is cleanly conjunctively decomposable into slices corresponding to the partitioning grouping employed in the reformulation. I.e., then the grouping employed in the reformulation forms the basis for a clean slicing.

More precisely: Suppose
$PDC(B1; D1; R; \mathit{fix}\ W; W, Y)$ is *disjointly describable* under (definitional) reformulation by $UT[W, Y, Z]$, given $G[W]$, as in Definition 11. Then

$$PDC(B1; D1; R; \mathit{fix}\ W; W, Y) \equiv \bigwedge_i PDC(B1i; D1i; Ri; \mathit{fix}\ W; W, Zi)$$

where $Ri \stackrel{\text{def}}{=} R^{Ni}$ is the internal prioritization of the group of defaults $D1i$, whose index set (tuple) is Ni. (Equivalently, the Zi on the right hand side could be replaced by Z.)

Proof Overview: Theorem 1 plus some lemmas about definitional reformulation of circumscriptions. Figure 4 illustrates the logical flow of the proof. □

Theorem 12 immediately yields results about **locality of inference**, using Theorem 2, and about **safety** of updating, using Theorem 3.

Next, we consider a special case of disjoint describability: asocial-monadic.

Theorem 13
(Fixed Cases Reformulation of Defaults)
In PDC, defaults can be reformulated by relativizing them to fixed (-formula) cases.

More precisely: In a $PDC(B; D^N; R; \mathit{fix}\ W; Z)$, suppose that

$$\forall i \in N.\ B[Z] \models \forall \boldsymbol{xi}.\ \bigvee_{j=1}^{mi} Fij[Z, \boldsymbol{xi}]$$

where $\boldsymbol{xi}$ is a (possibly empty) tuple of individual (object) variables, and where, for each i, j, the (possibly) open (elementary) formula $Fij[Z, \boldsymbol{xi}]$ is fixed relative to the circumscription (e.g., it mentions only function symbols; remember all functions are fixed). For each default index i, we call each Fij a *fixed case*. Suppose also that

$$B[Z] \models \forall i, j.\ \forall \boldsymbol{xi}.\ Eij[Z, \boldsymbol{xi}] \equiv (Fij[Z, \boldsymbol{xi}] \supset Di[Z, \boldsymbol{xi}])$$

I.e., suppose that each Eij is equivalent to the default Di relativized to the fixed case Fij. Then

$$PDC(B; D; R; \mathit{fix}\ W; Z) \equiv PDC(B; E; RR; \mathit{fix}\ W; Z)$$

where the tuple E stands for the concatenation of all the Eij's, and where RR is defined as the composition of R (as external prioritization) with a tuple $\emptyset T$ of empty prioritization p.o.'s. Each of $\emptyset T$'s elements is an empty prioritization p.o. $\emptyset Ti$ that is of size m and corresponds to (i.e., has as domain) the index set of the (sub-) tuple Ei.

Proof Overview: The key is that each original default pre-order is equivalently reformulated, in the

context of the circumscription's "augmentation" (i.e., second-order-quantified part in its definition cf. section 3), into a parallel default pre-order corresponding to Ei. □

Definition 14 (Asocially Monadic)
We say that a prioritized default circumscription $PDC(B; D; R; \mathit{fix}\ W; Z)$, or a corresponding CLD axiom set, is *asocially monadic* when:

1. All predicates in Z are monadic, i.e., 1-ary (a.k.a., unary).
2. The base sentence B has the form of a conjunction of universal[15] formulas. We will refer to these as the base formulas (axioms).
3. Every default formula (axiom) in D is quantifier-free.
4. No base sentence (axiom) in B, and no default formula (axiom) in D, "mixes" individuals. I.e., in their clausal forms, no clause contains two literals with different arguments. **Intuition:** different individuals "don't want to have anything to do with each other", i.e., they are *"asocial"*.
5. All terms appearing in the base and default formulas are ground, except for primitive variables.
6. The prioritization R is either layered (e.g., parallel), or it is *point-modular* (see definition below).
7. All (explicit) fixtures are of predicate symbols (W), rather than of arbitrary formulas. (In addition, as usual, all function symbols are fixed.)
8. Uniqueness of Names Axioms (UNA): The base B includes axioms enforcing the distinctness of all terms that appear in the base and default axioms.
9. Besides in the UNA, equality does not appear in the base or default formulas. (Remember, equality, when viewed as a predicate, is binary, not monadic.)

Definition 15
(Point-Modular Prioritization)
Point-modular prioritization generalizes (i.e., the class includes) the prioritization that is typical in default inheritance networks. By "point" here, we mean an individual in the logical language, either named (a ground term, e.g. Ed) or unnamed (e.g., referred to by a first-order variable, e.g., x in $\mathit{bat}(x) \supset 2\mathit{legs}(x)$). (This idea of a point can be straightforwardly generalized to a *tuple* of individuals (e.g., $\langle \mathit{Boss}(4321), d \rangle$) to handle predicates / formulas with arity more than one; but we are only considering here the unary case in the context of the asocially monadic case.) By point-modular, we mean that the overall prioritization is equivalent to the composition of some external prioritization (over the points) composed with a tuple of internal prioritizations, one per point. Point-modularity results when the prioritization is only specified between the *same* instantiations of different defaults. E.g., when the bat default has higher priority than the mammal default at each point: (the default axiom whose default formula is) $\mathit{bat}(\mathit{Betsy}) \supset 2\mathit{legs}(\mathit{Betsy})$ takes precedence over (the default axiom whose default formula is) $\mathit{mammal}(\mathit{Betsy}) \supset 4\mathit{legs}(\mathit{Betsy})$, $\mathit{bat}(\mathit{Joe}) \supset 2\mathit{legs}(\mathit{Joe})$ takes precedence over $\mathit{mammal}(\mathit{Joe}) \supset 4\mathit{legs}(\mathit{Joe})$, $\mathit{bat}(\mathit{Fido}) \supset 2\mathit{legs}(\mathit{Fido})$ takes precedence over $\mathit{mammal}(\mathit{Betsy}) \supset 4\mathit{legs}(\mathit{Betsy})$, etc., but there is no precedence between the defaults at different points, e.g., between $\mathit{bat}(\mathit{Betsy}) \supset 2\mathit{legs}(\mathit{Betsy})$ and $\mathit{mammal}(\mathit{Joe}) \supset 4\mathit{legs}(\mathit{Joe})$. Unfortunately, we do not have space to define point-modularity in further detail here; it requires discussing "pointwise" prioritization somewhat similar to that in [Lifschitz, 1988], and generalizing CLD to increase its expressivity with respect to prioritization. Note, however, that many point-modular prioritizations can be expressed in CLD. See [Grosof, 1992a] for more.

[15] **Terminology:** By *universal*, we mean without existential quantifiers.

Theorem 16
(Decomposition by Reformulation, Individual-Wise)
Suppose the $PDC(B0; D0; R0; \mathit{fix}\ W; Z)$ is asocially monadic cf. Definition 14. Then the circumscription can be cleanly sliced, i.e., conjunctively decomposed, into its *individual-wise* reformulation:

$$PDC(B0; D0; R0; \mathit{fix}\ W; Z) \equiv \bigwedge_{j=1}^{m+1} PDC(B1j; D1j; Rj; \mathit{fix}\ W; Z)$$

This individual-wise reformulation is **defined** as follows.

The basic idea of the reformulation is to divide the base and default axioms into groups: one group per named individual, plus a catch-all "remainder" group for all other, unnamed individuals. Some reformulation, of a relatively simple kind that is different from decomposition and one-by-one definitional reformulation, is involved in order to break up the quantified base axioms and the open defaults into these cases. Figure 5 illustrates this logical flow. The details of the overall reformulation are, however, a bit involved to define; bear with us.

To begin with, we partition the base and default formulas according to which arguments appear in them.

Let $J \stackrel{\text{def}}{=} \{1, \ldots, m\}$ index the set of all ground terms aj that appear in the base or default formulas.

Let $B0j$ stand for the tuple of base formulas that mention aj. Each of its members we write as $B0jk[Z]$.

Let $B0V$ stand for the tuple of base formulas, other than the UNA, that mention a free variable (all of these are universally quantified). Each of its members we write as $\forall x.\ B0Vk[Z, x]$. Here x is a single (free) individual variable.

We treat the default formulas similarly to the base. Let $D0j$ stand for the tuple of default formulas that mention aj. Each of its members we write as $D0jk[Z]$.

Let $D0V$ stand for the tuple of default formulas that mention a free variable (i.e., that are open; all of these

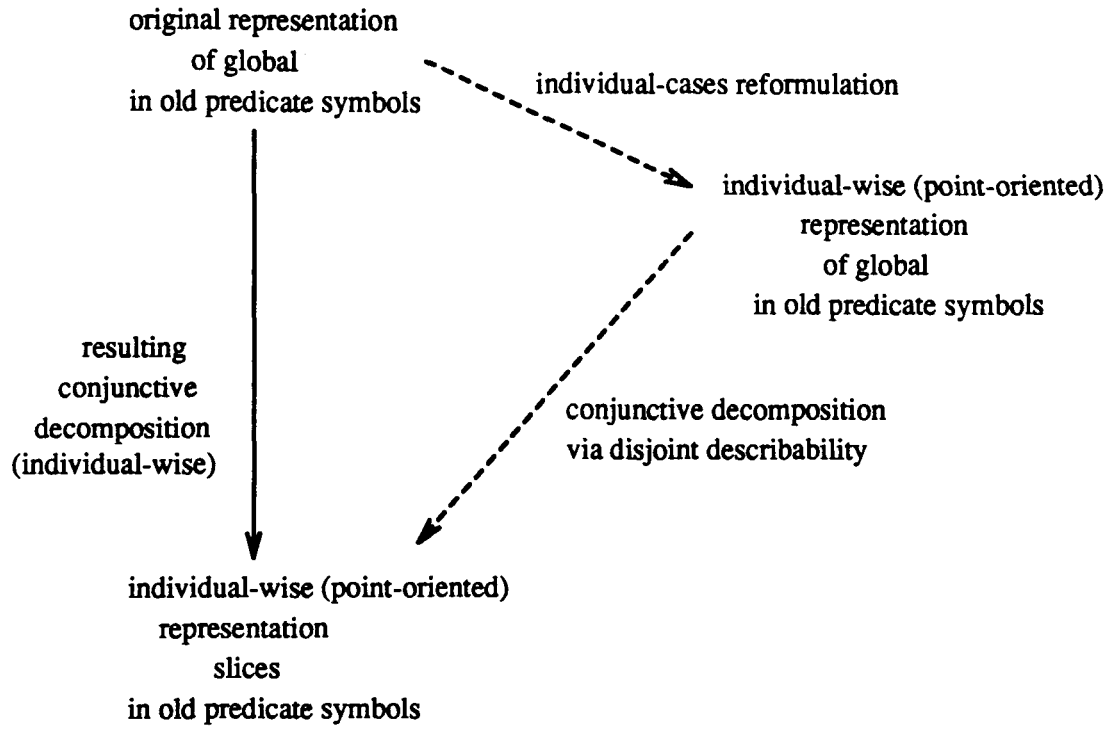


Figure 5: Asocially Monadic: a flow diagram of the reformulation steps involved. See also Figure 4.

are quantifier-free). Each of its members we write as $D0Vk[Z,x]$. Here x is a single (free) individual variable.

Next, we reformulate the base and default formulas that mention a variable.

For each $j \in J$, let $B1Vj$ stand for the instantiation of the quantified base formulas $B0V$ to aj. Each of its members $B1Vjk[Z]$ is defined as the formula $B0Vk[Z,aj]$.

Let $UNNAMED[x]$ stand for the formula $\bigwedge_{j \in J} x \neq aj$.

Let $B2V$ stand for the tuple of quantified base formulas after relativization to the unnamed case. Each of its members $B2Vk[Z]$ is defined as:

$$\forall x.\, UNNAMED[x] \supset B0Vk[Z,x]$$

For each $j \in J$, let $D1Vj$ stand for the instantiation of open default formulas $D0V$ to aj. Each of its members $D1Vjk[Z]$ is defined as the formula $D0Vk[Z,aj]$.

Let $D2V$ stand for the tuple of open default formulas after relativization to the unnamed case. Each of its members $D2Vk[Z]$ is defined as:

$$UNNAMED[x] \supset D0Vk[Z,x]$$

For each $j \in J$, Let $B1j$ stand for the conjunction of (all members of) the tuples $B0j$ and $B1Vj$.

For $j = m+1$ (i.e., the unnamed case), let $B1m+1$ stand for the conjunction of (all members of) the tuple $B2V$ plus the UNA.

For each $j \in J$, Let $D1j$ stand for the concatenation of the tuples $D0j$ and $D1Vj$.

For $j = m+1$ (i.e., the unnamed case), let $D1m+1$ stand for the tuple $D2V$.

Let Rj be defined as the prioritization internal to $D1j$, i.e., as R^{Nj}, where, for each $j = 1, \ldots, m+1$, Nj is the index tuple of $D1j$.

Proof Overview: We use a first stage of reformulation employing Theorem 13. This stage involves what we called above an "extra" kind of reformulation: e.g., to reformulate each open default axiom and each quantified base axiom into a collection of "point"-case (individual-case) axioms, plus a remainder-case (unnamed case) axiom. Then we use a second stage partitioning syndicate reformulation into disjoint describability, employing Theorem 12. In that second stage of reformulation, we treat the UNA as background. There, the newly introduced predicates are all 0-ary, except for those corresponding to the catch-all case. **The definitional reformulator consists of the explicit definitions of these newly introduced predicates. There is one new predicate for each ground atom in the original representation.** Note that the second stage itself combines two kinds of reformulation: definitional reformulation, to transform into a representation with disjoint predicates, and conjunctive decomposition. □

Figure 5 illustrates the logical flow of the reformula-

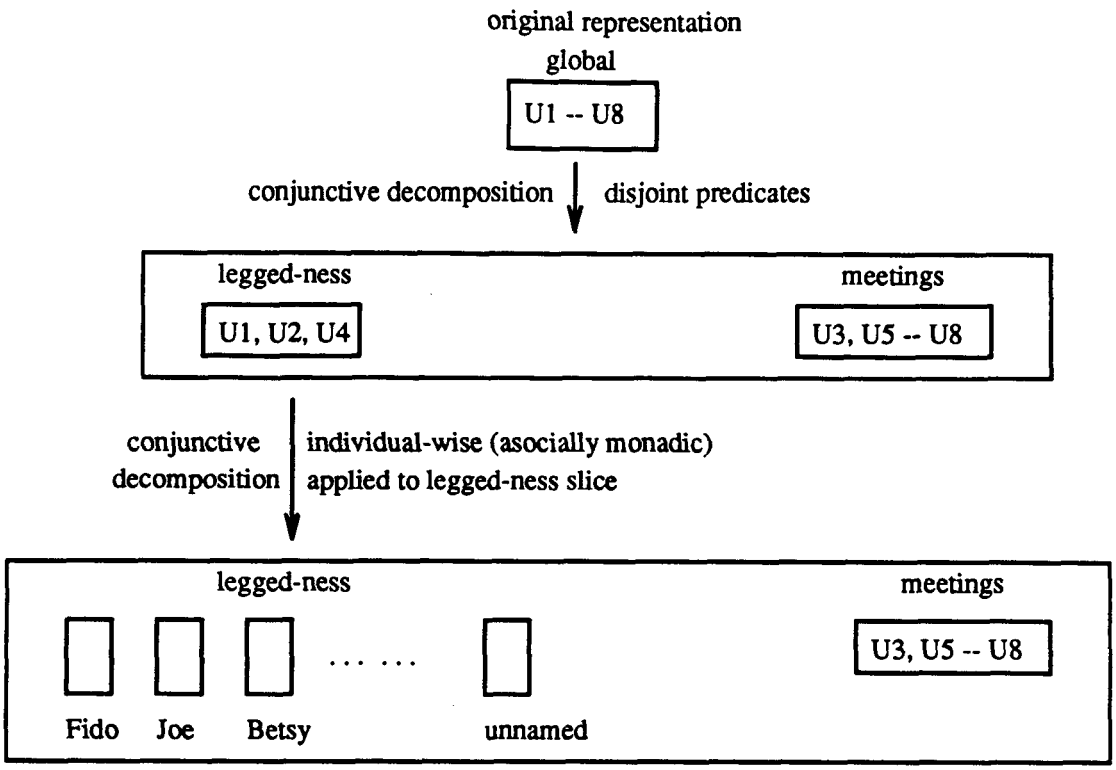


Figure 6: Conjunctive Decomposition using Asocially Monadic and Disjoint Predicates: In our main motivating example (section 2), we can conjunctively decompose the global axiom set (after the last update U_8) into two slices by employing the disjoint predicates result (Theorem 1): one slice about legged-ness, and the other slice about meetings. This first-stage decomposition is the same as in Figure 3. We can conjunctively decompose the legged-ness slice, individual-wise, by employing the asocially monadic result (Theorem 16). That is, in a second stage, we slice (more finely) within a slice that arose from the first stage. The second stage thus yields a second, finer-grain decomposition of the global axiom set, containing the meetings slice (unchanged from the first stage) plus each of the individual-case legged-ness slices. Together, the two stages exemplify the ability to decompose hierarchically / recursively. Each of the named-individual / "point" slices in the second stage contains a set of axioms that correspond to the instantiation / particularization of the original legged-ness axioms (U_1, U_2, and U_4) to (the case of) one named individual, e.g., *Joe*. Each outer box stands for a decomposition. Each inner box stands for a constituent axiom set.

tion steps involved; it builds upon Figure 4.

Application to Main Example: (Continued from the discussion in section 5:) Consider our main motivating example (about legged-ness and meetings, from section 2). There, after the final update U_8 (and, indeed, at any earlier point in the sequence of updates), the legged-ness slice, i.e., the set of axioms about legged-ness (U_1, U_2, and U_4) is asocially monadic. It can thus be conjunctively decomposed cleanly, individual-wise. Figure 6 illustrates and explains this decomposition. As we discussed earlier, the definitional reformulation involved in the individual-wise decomposition cf. Theorem 16 introduces a new 0-ary predicate for each ground atom in the original representation; in this example, two such new predicates are:

$$nbatJoe \equiv bat(Joe)$$
$$n2legsJoe \equiv 2legs(Joe)$$

Theorem 17
(Individual-wise Locality of Inference, when Asocially Monadic)
In Theorem 16, each slice j, where j is the (index of) a named individual (cf. statement of that Theorem), is sound and complete, relative to the global theory, for inference over its corresponding sub-language. That sub-language consists of the ground formulas (sentences) in which the only individual mentioned is j (e.g., *Betsy*). This locality holds both for forward inference, and for backward inference (query-answering). Note that to perform inference using any subset SJ of the named individuals J, one need only work in the conjunctive combination of those slices corresponding

to SJ.

For query-answering about a new named individual b (named in the query), just introduce the new term b into the set of terms that are indexed by J in the theorem. The only additional requirement is that the UNA ensure its distinctness from the other named individuals.

Application to Main Example: Thus after each update, inferences about any named individual's (e.g., *Joe*'s) legged-ness can be made by working in a slice axiom set that has been instantiated / particularized to that individual (*Joe*). One advantage of this is that simpler inference algorithms are available for such an expressively simpler axiom set. In this case, there is a decidable polynomial-time procedure (see "total-propositional" case results in [Grosof, 1992b]). By contrast, there is no general inference procedure, even for query-answering, yet available for the full example (i.e., including the meetings aspect). (See next section for discussion of inference procedures available for prioritized circumscription.) This illustrates that decomposition-type reformulation is useful to exploit available / known tractable special cases to do part of the inference in a NM theory, even when the overall theory is intractable or undecidable (see next section for more discussion of this point.)

Theorem 16 also immediately yields a powerful result about belief revision.

Theorem 18
(Safety of Updating, when Asocially Monadic) In CLD, let the previous axiom set be asocially monadic. Let an update $\mathcal{U}$ consist of base, default, and prioritization axioms, such that the formula parts of the base and default axioms are ground and mention only a set of named individuals IU. Then all of the previous conclusions derived solely from the rest of the named individuals' slices (i.e., the slices according to Theorem 16) are safe under the update.

Application to Main Example:
E.g., after update $\mathcal{U}_4$ (mentioning only *Joe* and *Spot*), this theorem tells us that we do not have to re-consider whether the previous conclusion *2legs(Betsy)* is still sanctioned: it must be preserved. Thus we can know, with relatively little computational work (see discussion of complexity in next section), that most of the previous NM conclusions are safe.

Disjoint Groups of Individuals:
Definition 14 and Theorems 16, 17, and 18 also generalize straightforwardly to considering disjoint *groups* of individuals, where any syntactic mixing in the axioms involves only individuals within the same group.

Discussion, Conclusions, and Future Work

Proof Procedures: Prioritized default circumscription is expressively reducible to prioritized predicate circumscription (see section 3). There exist several backward proof procedures for fairly expressive classes of prioritized predicate circumscription, including for layered (stratified) prioritization [Przymusinski, 1989] [Ginsberg, 1989] [Baker and Ginsberg, 1989] [Inoue and Helft, 1990] [Inoue *et al.*, 1991]. More interestingly, [Geffner, 1989] contains a proof theory and proof procedures which promise to be easily adaptable (using an equivalence theorem reported in [Grosof, 1991], detailed in [Grosof, 1992b]) to circumscription with *non*-layered prioritization.

Related Work: Note that we emphasize updating with new defaults, not just new for-sure axioms, unlike the conditional approaches to NMR (e.g., [Kraus *et al.*, 1990]). The ideas and results here apply to other NM formalisms, e.g.: Default Logic and Poole's [1988] and Brewka's [1989] systems, via the equivalence result in [Lifschitz, 1990]; as well as Geffner's [1989] system. The closest idea to conjunctive decomposition in the previous NMR literature is [Rathmann, 1990], who focussed, however, on conjunctively integrating heterogeneously-specified circumscriptive theories. He considered, moreover, only layered-priority predicate circumscriptions. Rathmann's and our work was developed independently. We are unaware of any other applications of reformulation to non-monotonic reasoning.

More Decompositions and Safeties: We did not have space here to report a number of additional results [Grosof, 1992b] about decompositions and their implications for safeties of updating, including about higher prioritization, hypotheticals, syntactic positivity, "serial" decompositions, weaker forms of irrelevance; and about the relationship of decompositions to specification and backward inference.

Algorithms and Automation of Our Results: In future work, we plan to automate *recognition* of decompositions and safeties of updating cf. our theorems, and the actual performance of the according *reformulation*. For the disjoint-predicates and asocial-monadic cases, we have **polynomial-time algorithms** to perform this: $O(n^3)$ time, where n is the size of the CLD axiom set.

Exploiting Truth Maintenance: Such recognition establishes "monotonicity" (i.e., implication) relationships between theories and sub-theories (e.g., theory after update versus theory before update; or theory versus slice). We plan also to automate a generalized ATMS-style [de Kleer, 1986] high-level architectural book-keeping scheme to exploit such stored monotonicity relationships to support inference and belief revision in a prioritized database. [Grosof, 1992b] gives details.

More General Cases of Disjoint Describability: In future work, we aim to find cases of disjoint describability that are more general than asocial-monadic, but are still easily recognizable syntactically (in terms of the syntax of the global axiom set). E.g., in

our main example, it would be nice to be able to particularize the Meetings slice to the individual Ed, in the same way that the asocial monadic result guarantees one can particularize the Legged-ness slice to the individual Joe. Right now, we can show this particularization about Ed is legitimate, but our proof method is by hand. We would like to be able to formalize and automate a class of decompositions for which this (Ed etc.) is an instance.

Conclusions I: See **Strategy and Summary** in section 1.

Conclusions II: Analyzing Computational Advantages of Reformulation: In future work, we also plan to analyze in detail the computational advantages and trade-offs involved in our decompositions and definitional reformulations. You may be wondering why we did not give any such computational complexity analysis in this paper. The main reason is that the picture is quite complicated for non-monotonic reasoning.

Even for query-answering in propositional default theories without priorities, current results show worst-case is **exponential** (NP-hard) [Selman and Kautz, 1989] [Kautz and Selman, 1989] [Selman and Levesque, 1989]. Thus: *Divide-to-conquer, i.e., seeking locality, is clearly desirable.*

But the basic complexity results for any kind of forward reasoning with priorities, for any kind of belief revision, and even for most kinds of backward (query-answering) reasoning are not available for circumscription, or other NM formalisms. Known tractable cases are highly restricted. ([Selman and Kautz, 1989] [Kautz and Selman, 1989] give polynomial-time backward procedures for special cases, including restrictions of Horn, of propositional default reasoning in their model-preference logic and in Default Logic. Delgrande [1991] gives a polynomial-time backward procedure for a Horn propositional case of his conditional logic.)

However, we believe that as these results become available, we will be able to show that decomposition and reformulation are advantageous. Our aim has been to develop methods that will be broadly applicable, and to break off a piece of the overall hard problems of non-monotonic reasoning. In current work, we are addressing how to relate our results to currently known tractable and intractable cases.

One clear advantage is that for many cases with quantification, for which worst-case is **undecidable** ([Reiter, 1980] [Kolaitis and Papadimitriou, 1988]): *We are able to reformulate some of the reasoning to become propositional, hence decidable.* E.g., when reasoning about individuals, for the asocially monadic class of theories (see Theorem 16).

Acknowledgements

Thanks to Devika Subramanian, Vladimir Lifschitz, and Michael Lowry for long-ago useful discussions on logical aspects of definitional reformulation. Thanks to Leora Morgenstern, Hector Geffner, and two anonymous reviewers for their comments on previous drafts.

References

A. Baker and M. Ginsberg. A theorem prover for prioritized circumscription. *Proceedings IJCAI-89*, pages 463–467, Detroit, MI., 1989.

G. Brewka. Preferred subtheories: An extended logical framework for default reasoning. *Proceedings IJCAI-89*, pages 1043–1049, Detroit, Michigan, 1989.

K. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

J. de Kleer. An assumption-based truth maintenance system. *Artificial Intelligence*, 28:280–297, 1986.

James P. Delgrande. Incorporating nonmonotonic reasoning in horn clause theories. *Proceedings of AAAI-91*, pages 405–411, 445 Burgess Drive, Menlo Park, CA 94025, 1991. AAAI Press.

J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

H. Geffner. *Default Reasoning: Causal and Conditional Theories.* PhD thesis, Computer Science Department, UCLA, Los Angeles, CA, 1989. Revised version published by MIT Press, 1992.

M. Ginsberg. A circumscriptive theorem prover. *Artificial Intelligence*, 39:209–230, 1989.

Benjamin N. Grosof. Generalizing prioritization. *Proceedings of the Second International Conference on Principle of Knowledge Representation and Reasoning*, pages 289–300, April 1991. Also available as IBM Research Report RC15605, IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598.

Benjamin N. Grosof. Generalizing prioritization ii (working title). Working paper., 1992.

Benjamin N. Grosof. *Updating and Structure in Non-Monotonic Theories.* PhD thesis, Computer Science Dept., Stanford University, Stanford, California 94305, 1992.

Katsumi Inoue and Nicolas Helft. On theorem provers for circumscription. Working paper. This is a revised version of a paper appearing under the same title in the Proceedings of the Canadian Conference on Computer Science and Artificial Intelligence '90, Ottawa, Canada, May 1990., Apr 1990.

Katsumi Inoue, Nicolas Helft, and David Poole. Query answering in circumscription. *Proceedings of IJCAI-91*, pages 426–431, San Mateo, California, 1991. Morgan Kaufmann.

H. Kautz and B. Selman. Hard problems for simple default logics. *Proceedings of the First International Conference on Principle of Knowledge Representa-*

tion and Reasoning, pages 189–197, Toronto, Ontario, 1989.

Phokion G. Kolaitis and Christos H. Papadimitriou. Some computational aspects of circumscription. *Proceedings of AAAI-88*, pages 465–469, San Mateo, California, 1988. Morgan Kaufmann. Held Minneapolis, MN.

S. Kraus, D. Lehmann, and M. Magidor. Preferential models and cumulative logics. *Artificial Intelligence*, 44:167–207, 1990.

Vladimir Lifschitz. Some results on circumscription. *Proceedings of the First AAAI Non-Monotonic Reasoning Workshop*, pages 151–164, Oct 1984. Held New Paltz, NY.

V. Lifschitz. Computing circumscription. *Proceedings IJCAI-85*, pages 121–127, Los Angeles, CA. 85.

Vladimir Lifschitz. On the declarative se cs of logic programs with negation. In Matthe Jinsberg, editor, *Readings in Nonmonotonic oning.* Morgan Kaufmann, San Mateo, CA, 1987.

V. Lifschitz. Circumscriptive theories: a logic-based framework for knowledge representation. *Journal of Philosophical Logic*, 17:391–441, 1988.

Vladimir Lifschitz. On open defaults. *Proceedings Symposium on Computational Logic*, Brussels, Belgium, 1990.

J. McCarthy. Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

R. Moore. Semantical considerations on nonmonotonic logics. *Artificial Intelligence*, 25:75–94, 1985.

D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27–47, 1988.

Teodor Przymusinski. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann, San Mateo, CA., 1988.

T. Przymusinski. An algorithm for circumscription. *Artificial Intelligence*, 38:49–73, 1989.

Peter K. Rathmann. *Nonmonotonic Semantics for Partitioned Knowledge Bases.* PhD thesis, Computer Science Dept., Stanford University, Stanford, California 94305, Jun 1990.

R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 12:81–132, 1980.

B. Selman and H. Kautz. The complexity of model preference default theories. In M. Reinfrank *et al.*, editor, *Proceedings of the Second International Workshop on Non-Monotonic Reasoning*, pages 115–130, Berlin, Germany, 1989. Springer Lecture Notes on Computer Science.

B. Selman and H. Levesque. The tractability of path-based inheritance. *Proceedings IJCAI-89*, pages 1140–1145, Detroit, MI., 1989.

D. Touretzky. *The Mathematics of Inheritance Systems.* Pitman, London, 1986.

Scare Tactics: Evaluating Problem Decompositions Using Failure Scenarios

B. Robert Helm and Stephen Fickas

Department of Computer and Information Science
University of Oregon
bhelm@cs.uoregon.edu fickas@cs.uoregon.edu

Abstract

Our interest is in the design of multi-agent problem-solving systems, which we refer to as *composite systems*. We have proposed an approach to composite system design by decomposition of problem statements. An automated assistant called Critter provides a library of reusable design transformations which allow a human analyst to search the space of decompositions for a problem.

In this paper we describe a method for evaluating and critiquing problem decompositions generated by this search process. The method uses knowledge stored in the form of *failure decompositions* attached to design transformations. We suggest the benefits of our critiquing method by showing how it could re-derive steps of a published development example. We then identify several open issues for the method.

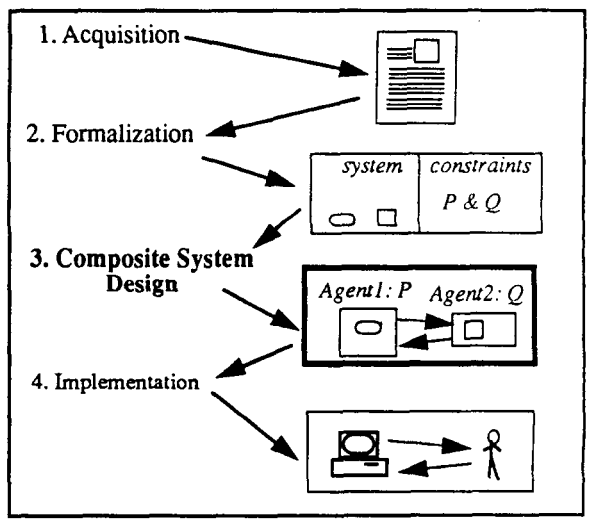
Introduction

Our group is interested in the design of *composite systems*, ones that encompass multiple agents cooperating in an ongoing activity [Fickas & Helm, 1992][1]. We arrived at this interest while studying the processes of software development. Systems analysts in the domains we studied [Fickas and Nagarajan, 1988] focused on policies and concerns which cut across human, hardware and software components. In composite system design, software agents are treated the same as human and physical agents, as components to be integrated together to solve larger system constraints. We have developed a design model, called Critter, to help a human designer create a composite system design [Fickas and Helm, 1992].

Figure 1 shows the place of composite system design within the more general system lifecycle we envision. We view the design process of a system as composed of four phases:

1. Acquisition. The designer acquires an initial, informal statement of the problem in terms of text descriptions and diagrams.
2. Formalization. The designer creates an initial formulation of the problem in terms of *system* and *constraints*. The initial system formally describes a minimal set of assumptions about possible behavior of the target system. The constraints formally describe the desirable behavior in terms of the initial system.
3. Composite system design. Given the formulation of the problem as initial constraints and system, the designer uses Critter to build a formal specification of a *composite system* for the problem. A composite system is a set of interacting, reactive components called *agents*. Each agent is associated with a set of *responsibilities*, constraints which the agent's behavior must satisfy. If all agents behave according to their responsibilities, the composite system will solve the desired problem.
4. Implementation. The agents of the composite system are implemented in the appropriate "technology" according to their specifications. This could mean producing software or manufacturing hardware. It might also involve writing legal statutes or training manuals describing the responsibilities of humans playing the role of an agent.

Figure 1. Context of composite system design.



In designing a library circulation system, for example, the designer first acquires an informal statement of assumptions about the system, and constraints such as "Library patrons get the books they want" and "Every book is accounted for". The designer formalizes the system and constraints. The designer then uses Critter to design a composite system rep-

1. This work was supported by the National Science Foundation under grant CCR-880485.

resenting the library. This formally specifies the responsibilities of agents such as the online catalog ("Report catalog entry if book title found"), the antitheft devices ("Sound alarm when magnetized book passes through the gate"), and even the library patrons ("Look in the online catalog if the book title is known"). Finally, the library agents are implemented. For the online catalog and antitheft device, this would involve writing or acquiring software and hardware. "Implementing" the library patron implies writing regulations and guidebooks to inform patrons of their role.

We have begun to formalize an approach to phase 3, composite system design, by decomposing problem statements. The designer incrementally decomposes the global constraints in the initial problem statement into the conjunction of more manageable subconstraints. The designer then *assigns responsibility* for these constraints to particular agents. For example, the designer of a library system could decompose the global constraint "Library patrons get the books they want" into "Library patrons can find the books they want" and "Library patrons can get the books they find." The patron and the online catalog agents are assigned responsibility for the former constraint; the patron and the library staff agents are assigned responsibility for the latter. [Feather, 1987] illustrates the approach by an informal example.

Critter includes a library of formally-represented composite system design tactics, and a suite of tools for automated evaluation and critiquing of the designs generated. To incorporate the decomposition method into Critter, we need to (1) identify and formalize general tactics for decomposing problem statements, and (2) identify knowledge which Critter could use to critique problem decompositions.

This paper focuses on the latter problem, that of critiquing problem decompositions. We illustrate a method for generating critiques, by showing how it rationalizes specific steps in a published development example [Feather, 1987]. In that example, Feather informally derived an elevator system design from the global constraints of never unnecessarily delaying passengers, and never moving passengers further from their destination. The development was guided by Feather's intuitions of the problem, and his domain knowledge. We show how we can capture some of this knowledge, in the form of a library of *failure scenarios*. We then discuss the research issues raised by this example.

Our work addresses the workshop in two respects:

1. We propose general techniques for evaluating problem decompositions in multi-agent systems. These techniques may find use beyond our interests, in formulating problems for multi-agent planning or for distributed AI systems.
2. The evaluation approach we propose in this paper requires techniques for storing and using compiled abstractions, specifically abstract plans. This workshop may identify research we can apply to our approach.

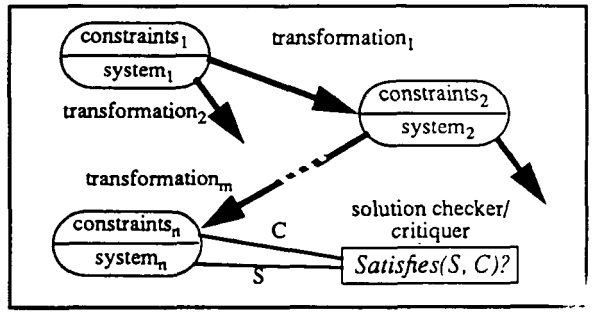
Searching for decompositions

In this section, we outline the Critter composite system design model, and its support for synthesizing problem decompositions.

Critter treats composite system design as search in a state space (Figure 2). A typical search algorithm has the following components:

- A state space representation
- A set of search operators for moving between states
- A solution checker which recognizes satisfactory states.
- A heuristic evaluator which identifies promising states.
- A search manager which maintains a record of states visited and operators applied.

Figure 2 Composite design search.



Each state in Critter's search space represents a complete composite system design for the problem at hand. The "search operators" which move from state to state are design transformations stored in Critter's knowledge base. The solution states in the search are acceptable composite system designs -- Critter provides critiquing tools to help identify these.

The last two components, heuristic evaluation and search management, are beyond the scope of our research at present. For heuristic evaluation, we rely on the human designer. Our studies of composite system design heuristics [Feather, Fickas, and Helm, 1991] [Fickas, Feather, and Helm, 1991] suggest that this task will have to remain with the designer in the foreseeable future. Support for human evaluation of design operators is the focus of other research [Johnson and Feather, 1991]. As for search management, Critter is implemented using an extended form of IBIS [Conklin and Begeman, 1988] that provides for separate design states. Critter provides functions for searching and backtracking in this space. In our current

implementation, new states are generated by hand-simulating operator application using an editor.

In the remainder of this section, we discuss Critter's support for the first three search components: state representation, transformations, and solution checking.

Design states

Figure 3 informally represents an initial state for the elevator design problem we use to illustrate our design approach. A state (hereafter "design state") in Critter's design search space has two parts:

1. *System.* The system portion defines the space of possible behaviors of the current composite system design.
2. *Constraints.* The constraint portion of a design state defines the subset of possible behaviors which are viewed as legal or desirable.

The system portion of a design state represents the space possible behaviors of the composite system. It specifies a set of objects, a set of primitive relations, and a set of actions which can add or delete object tuples from the relations. The system is thus similar to a planning domain for a STRIPS-like planner.

Relations and actions in the system portion are also labelled by *agents*. Agents in our model are simple reactive components. A relation labelled by an agent can be sensed by that agent; an action labelled by an agent can be controlled by that agent.

A behavior is a sequence of actions, each action labelled by its controlling agent. A prefix of a behavior represents the intermediate state of the composite system during its operation; to avoid confusion with design states, we will refer to execution states of the composite system as "points." As with planning domains, the system portion is non-deterministic; more than one action may be possible at a given point.

The system portion in Figure 3 includes two classes of agents, an elevator and set of passengers. Each passenger controls its own actions of entering and exiting elevators. A passenger can sense which floor it is on, and whether or not it is in a given elevator. Passengers also have a destination (not shown in the figure), which they know. The unique elevator controls its action of moving from floor to floor. It also can wait at a floor (not shown in the figure). The elevator can sense whether it is on a given floor.

The constraint portion of a design state is composed of a set of constraints. Each constraint is a predicate which is true or false for each behavior generated by the system portion. A constraint may refer to either relations or actions in the system portion.

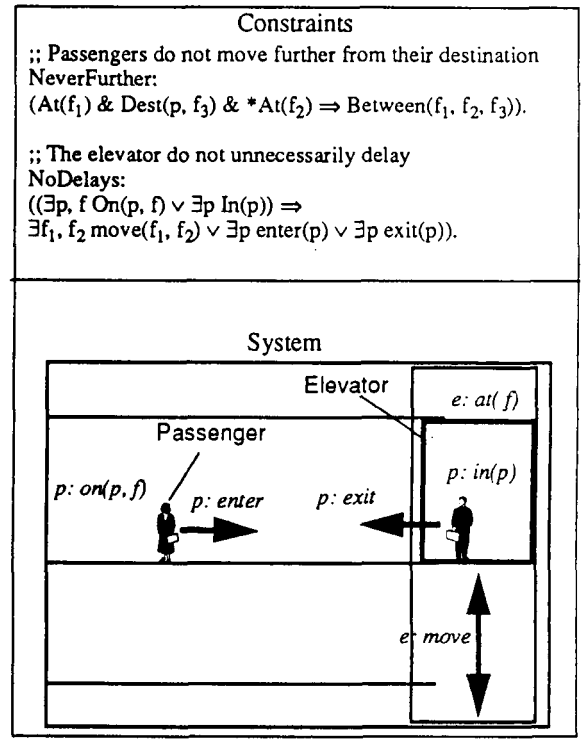
The constraint portion of Figure 3 includes two constraints:

1. NeverFurther: Elevator passengers should not move further from their destinations.
2. NoDelays: Passengers should not be unnecessarily delayed. This means that at each point in the elevator's behavior, it must either move, take on, or drop off passengers, unless no passengers exist.

Agents in the system portion can be assigned *responsibility* for constraints. If an agent has been assigned responsibility for a constraint, that agent must act to satisfy the constraint. The agent must control its actions so that all of the behaviors it generates satisfy the constraint, regardless of the actions of other agents. We call a constraint which is the responsibility of some agent an "assigned constraint."

The legal behaviors of a composite system design are all sequences of actions which can be generated by the agents in its system portion, and which satisfy all of the constraints and responsibility assignments of the constraint portion.

Figure 3 Initial state of the elevator problem.



We have represented the system portion informally, which is adequate for the purposes of this paper. Critter represents the system portion of design states are expressed in a Numerical Petri Net [Wilbur-Ham, 1985] notation, extended to include agents.

The constraints are written in a linear-time, quantified temporal logic extended to include constructs for responsibility assignment [Dubois, 1990]. For the most part, the constraint notation used in this paper is simply the predicate calculus, except on the following points:

- Variables appearing in a constraint are universally quantified unless otherwise indicated.
- Actions appear as predicates in constraints. The expression move(f_1, f_2) in the NoDelays constraint, for instance, states that "The elevator moves from f_1 to f_2 at the current point of the system's behavior." Ordinary predicates are capitalized to distinguish them from actions.
- Temporal logic operators reference future and past points in the system's behavior. The only construct we use in this paper is the * operator, which denotes the next point. Thus, the expression At(f_1) & Dest(p, f_3) & *At(f_2) can be read "The elevator is at f_1 and passenger p has destination f_3 and at the next point the elevator is at f_2."
- Constraints can include responsibility assignment operators. [Feather, 1987] and [Dubois, 1990] give a formal semantics for this construct; we use it informally throughout.
- The notation C[t/t'] denotes the constraint C with all occurrences of t replaced by t'. Thus, the expression NeverFurther[p/p_1] denotes the NeverFurther constraint with all occurrences of p replaced by p_1.

Design transformations

Critter has a library of design transformations that function as operators in the search for an acceptable composite system design. Each design transformation has a pattern which matches against parts of an existing design state, a result which generates in a new design state, and a list of conditions called domain assumptions that must hold for the transformation to apply (we do not discuss domain assumptions in this paper). We will represent the pattern and result of transformations as Prolog-like clauses.

Transformations are applied interactively. The human designer selects a transformation to apply, and matches the pattern of the transformation to components of the current design state. The system then generates a new design state incorporating the result of the transformation.

In design by problem decomposition, most of the transformations applied are of the following form:
pattern: constraint(C).
result: constraint(C_1 & ..., & C_n).

C in the pattern is a constraint. The transformation generates a new state where C is replaced by a new constraint C_1 & ... & C_n that entails C. This in turn may be decomposed into subconstraints.

When the designer judges that the constraints have been decomposed into sufficiently simple subconstraints, she assigns responsibility for each of the subconstraints to a single agent. As described above, assigning responsibility for a constraint C to an agent requires that agent to limit its actions so that C is met, regardless of the actions of other agents in the system.

Finally, the designer applies transformations to unfold the assigned constraints onto the preconditions of actions in the system portion. The designer may also have to use low-level design editing transformations to change the details of actions and relations in the system.

Our main interest is in the transformations for decomposition of constraints and assignment of responsibility. As an example, one class of decomposition transformation used in this paper is *Zone Defense*. Intuitively, Zone Defense decomposes a constraint by

1. Selecting an object.
2. Dividing the object's lifetime into "zones", and
3. Splitting the constraint into subconstraints based on the "zone" the object is in.

More formally, given a constraint C and a universally quantified variable v, we decompose C into subconstraints based on possible states of objects to which v can be bound. The application of Zone Defense to the NeverFurther constraint of the elevator problem is as follows:
pattern: constraint(NeverFurther), uv(p, NeverFurther).
result: constraint((p_1)
$\exists$f enter(p_1, f) $\Rightarrow$ NeverFurther[p/p_1]
&
$\exists f_1, f_2$ move(f_1, f_2) $\Rightarrow$ NeverFurther[p/p_1]
&
$\exists$f exit(p_1, f) $\Rightarrow$ NeverFurther[p/p_1]
&
($\exists$f enter(p_1, f) $\vee$ $\exists f_1, f_2$ move(f_1, f_2)
$\vee$ $\exists$f exit(p_1, f))).

Intuitively, to ensure that passengers never move further from their destination, we can ensure that the constraint holds when the passenger enters an elevator, when an elevator moves, and when the passenger exits the elevator.

Having broken NeverFurther into more manageable subconstraints, the designer can next assign responsibility for one of the subgoals to the elevator. The only action the elevator controls are "move" and "wait", so we separate these subconstraints of the decomposition, and assign them to the elevator with the *Limit Each Action* transformation. The instantiation of this transformation on the move action reads as follows:
pattern: constraint((p_1)
$\exists f_1, f_2$ move(f_1, f_2) $\Rightarrow$ NeverFurther[p/p_1]
agent(elevator)).
result: constraint((p_1)
responsible(c1,
$\exists f_1, f_2$ move(f_1, f_2) $\Rightarrow$ NeverFurther[p/p_1]),
agent(elevator)).

This transformation requires that the elevator control each move action so that NeverFurther holds, regardless of the actions of the passengers. Unfortunately, there is no way for both NeverFurther and NoDelays to be met if Never-

Further is assigned to the elevator as shown here. Two passengers going in different directions can enter the elevator and leave the elevator no choice but to either violate NeverFurther or NoDelays. We discuss this example further below.

Detecting solution states

A solution state in Critter's search is a design state where the system portion does not generate any behaviors which violate the constraints in the constraint portion. Critter includes analysis tools to help the analyst identify solution states. In [Fickas and Helm, 1992], we discuss several of these analysis tools and trade-offs between them. In this paper, we will discuss mainly the OPIE planning tool [Anderson and Fickas, 1989]. The system portion of a design state is effectively a planning domain. OPIE is a planner which shows that a design state is not a solution by producing a plan incorporating actions from the system portion for violating one or more constraints. We refer to such a plan as a *failure scenario.*

For example, to show that the initial elevator design state in Figure 3 is not a solution state, OPIE can generate a plan for violating the NeverFurther constraint from an initial point supplied by the analyst (+ indicates a relation added, - indicates a relation deleted):

Initial. On(p, 1), At(1), D(p, 2);
1. enter(p, 1): -On(p, 1) +In(p);
2. move(1, 3): -At(1) + At(3);

>>Violation of NeverFurther <<

At(1) & In(p) & D(p, 2) & *At(3)
& ¬Between(1, 3, 2)

This illustrates the general style of solution testing in Critter; we focus on identifying classes of behaviors or scenarios which violate the constraints, rather than verifying that the constraints are met. In the next section, we discuss some of the benefits of this approach. We also identify some of its limitations, and suggest how to address those limitations in design by decomposition.

Critiquing with failure scenarios

Critiquing composite system design states by failure scenarios offers two benefits for design:

1. *Diagnosis.* A scenario is a specific behavior of the system which violates a constraint. The designer can use this behavior to diagnose the problems of the current design state and identify potential solutions.
2. *Validation.* The system portion of a design state is effectively a model of what is possible in the design domain. If a scenario generated from that model is counterintuitive or unlikely in the domain, this is a hint that the model is too weak.

Our goal is to gain these benefits for design by problem decomposition. In this section, we suggest an approach to critiquing problem decompositions, and demonstrate the approach by showing how it could reproduce steps taken in a published composite system design derivation.

Synthesizing an approach

Planning over the system portion is not necessarily the best way to generate failure scenarios for decompositions, or for composite system designs in general. The planner cannot tell how likely, or how important a failure scenario it generates is. Consequently, it generates many scenarios with marginal value for design. More seriously, a designer can miss important failure scenarios in a design problem by "naive specification" of the problem. The planner relies entirely on the information in the design state to generate critiques. This knowledge may be incomplete or incorrect with respect to the design domain. The designer can exclude a particular failure, even a common one, by not including actions in the system portion which allow the planner to generate that failure. For example, the designer of a library can miss the possibility of books being stolen, by not encoding a "steal book" action in the initial design state.

A critic with domain knowledge can focus more quickly on serious problems, and can recognize problems even in naive specifications. We describe a domain-specific critic called SKATE for library design in [Fickas and Nagarajan, 1988]. SKATE has a case base of 1) library designs, 2) constraints they meet or violate, and 3) failure scenarios for those designs. Given a proposed design and a set of constraints, SKATE retrieves designs from its case base that match features of the proposed design, and that violate the proposed constraints. It then runs failure scenarios from the retrieved designs to demonstrate the problems. Given a library design including unrestricted checkout of books, for instance, and a constraint "users have a large selection of books to choose from", SKATE retrieves a design case with unrestricted checkout. It then executes a stored failure scenario of a "run" on the library, in which unrestricted checkout is used to strip the shelves bare.

SKATE's case base points it directly to well-known library failure scenarios, avoiding the problem of generating marginally useful scenarios. SKATE also avoids the problem of naive specifications. The failure scenarios SKATE generates are not restricted to using the actions and relations specified in the proposed library design. They can also include "environment" actions such stealing or destroying books, which a designer might not specify but which are known to cause problems in the library domain.

SKATE, however, suffers from a limited ability to match designs against cases. In general, it is hard to match the features of one arbitrary specification to another [Robinson, 1990]. SKATE requires the user to manually map features of the proposed library design into features used in SKATE's case base. This task is onerous and error-prone; important critiques can be missed by user mistakes in the mapping process.

One solution proposed by Fickas and Nagarajan is to integrate matching more closely with the process of producing designs. They suggested that the proposed design be generated by domain-specific editor, equipped with a collection of library components appearing in the case base. In effect, this limits the designer to producing designs SKATE knows how to critique.

Based on these considerations, we propose the following approach which integrates the approaches of OPIE and SKATE:

1. We will use Critter's transformation library in place of the case base of SKATE. Each decomposition transformation has an attached set of failure scenarios representing its typical defects. Critter thus plays the role of the domain-specific editor proposed by Fickas and Nagarajan.
2. Critter matches failure scenarios when it applies a transformation. Matching is simpler, compared to SKATE, because the instantiation of the transformation itself guides the matching process.
3. Critter critiques a design state using the OPIE planner. OPIE produces plans by specializing and refining previously matched failure scenarios.

This approach addresses the problem of marginally useful scenarios by storing a library of typically useful scenarios on transformations, and using these scenarios to focus the planner. Our study of failures in multi-agent systems [Fickas, Feather, & Helm, 1991] suggests that we can find such characteristic failure scenarios for problem decompositions. The approach also addresses the naive modelling problem by allowing failure scenarios to introduce new relations and actions into the design state being critiqued. As in SKATE, these "environment" components represent knowledge of well-known problems that crop up in multi-agent systems.

To illustrate this approach, we next show how critiques generated this way could anticipate two design steps which occurred in the composite system design development described in [Feather, 1987].

Focusing on a decomposition failure

Recall that Feather's elevator design problem had two initial constraints:

1. Passengers should never move further from their destination (NeverFurther).
2. Passengers should not be unnecessarily delayed (NoDelays).

From the constraint that passengers never move further from their destination, the designer in Feather's example "chooses the implication" that passengers in the same elevator must be travelling in the same direction. We show how a failure scenario can focus the planner to reproduce this design step.

Earlier we showed a development step which assigned the NeverFurther goal to the elevator. This step used a transformation called Limit Each Action. As noted above, this assignment requires the elevator to satisfy NeverFurther for all combinations of passengers and floors, regardless of prior actions of the passengers involved. Critter can generate an interesting counterexample to this constraint using a scenario attached to the Limit Each Action transformation. The attached scenario is called "incompatibility conspiracy". The abstract incompatibility conspiracy scenario requires that:

1. There are two agents in the system portion whose state can affect the truth of the constraint assigned by the transformation.
2. These two agents can act to reach a state S where an application of the action A will fail to satisfy the constraint for either one agent, or for the other. For the assigned constraint C and limited action A, we can compute the conditions on the state S the conspiring agents must reach. Specifically, we regress $\exists a1, a2 \neg (C[a1] \& C[a2])$ through the action A.

Instantiating the scenario on the application of Limit Each Action, we get a goal of generating a state where:

- There are two passengers in an elevator on a floor f_1
- The two passengers have destinations f_3, f_4
- No floor f_2 exists such that Between(f_1, f_2, f_3) & Between(f_1, f_2, f_4)

It remains for the planner, OPIE, to try to extend this minimal "scenario" into a plan. This requires a considerable effort on OPIE's part. If such a plan can be found, however, it provides a motivation for the requirement that passengers only enter the elevator with compatible passengers -- passengers travelling in the same direction.

Using an abstract failure scenario thus allowed the planner to recognize a critical deficiency, one which Feather deduced informally in his example.

Critiquing a naive communication model

In another step of Feather's development of the elevator problem, passengers have been assigned to enter the elevator when a suitable one arrives at the passenger's floor. The elevator has been assigned to take passengers to their destination when they enter. From this, the designer in Feather's example derives the constraint that the passengers communicate their presence on entering the elevator. We show how an abstract failure scenario could lead a

designer to this communication protocol, by introducing environment actions and relations which cause a stereotypical breakdown of communication.

The starting point for this development is the NoDelays goal, which requires that the elevator must either move or load and unload passengers when any passenger is present. The designer applies a macro-transformation called *Sequential Split* to the NoDelays goal. This transformation combines a Zone Defense operator with responsibility assignment. It subdivides the task of moving passengers into sequential zones, based on the status of the passenger. In particular, the designer uses Sequential Split to make passengers responsible for NoDelays when the elevator arrives at a floor. Responsibility passes sequentially to the elevator once the passenger enters. The instantiated version of Sequential Split expresses this formally:

pattern: constraint(
    On(p, f) & At(f) ⇒ NoDelays),
    agent(p), agent(elevator).
result: constraint(
    On(p, f) & At(e, f) ⇒ Responsible(p, enter(p, f))
    &
    (In(p) & At(f_1) ⇒
    Responsible(elevator, ∃ f_2 move(f_1, f_2)))))
    agent(p), agent(elevator).

Note that the requirement that the elevator moves, coupled with the NeverFurther constraint, ensures that the passenger will eventually arrive at its destination.

Our studies of transportation system failures suggest that sequential decompositions, while common, frequently fail due to "hand-off errors". In one hand-off failure scenario, for instance, the agent responsible for the second half of a sequential decomposition fails to pick up where the first agent leaves off, because it does not recognize it has become responsible. Translating this to the current problem, the elevator may fail to move, because it does not recognize that the passenger has entered and thus handed off responsibility for NoDelays.

This sequence of events is encoded as an abstract scenario attached to Sequential Split. Instantiated with the Sequential Split transformation above, it asks the planner to expand a sequence of states where:

1. ∃p, f On(p, f) & At(f);
2. In(p) & At(f) & ¬ElevatorResponsibleForMove

Note that the abstract scenario introduces a new binary relation ElevatorResponsibleForMove. This relation represents the elevator's internal model of the condition that activates its responsibilities. The failure scenario also introduces actions for asserting and deleting this relation. As with SKATE scenarios, abstract scenarios in Critter can add environment actions and relations to the design state for use in generating critiques. In this example, the new components allow OPIE to generate a plan in which a passenger enters the elevator, but the elevator does not recognize this (ElevatorResponsibleForMove is false), and so does not move.

Environment components introduced by attached scenarios allow OPIE to avoid the naive modelling problem. They force the designer to consider behavior which is typical for a class of problem decompositions, even if the designer has neglected to include components which support such behavior in the initial design state.

Returning to our example, the designer acknowledges the scenario, and designs a communication protocol to prevent it. The passenger becomes responsible for notifying the elevator when it enters the elevator. The elevator will acknowledge the handoff. This can be implemented by a familiar interface: passengers hit a button on entry to the elevator, and the button lights in response.

The handoff failure scenario thus produces and rationalizes an interface component developed in the Feather example. This step also shows how a failure scenarios incorporating environment components can expose naive assumptions about inter-agent communication, and lead to more realistic agent interfaces as a result.

Conclusions and Issues

We have proposed an approach to composite system design based on problem decomposition. To evaluate designs generated by the approach, we have proposed a method of scenario-based critiquing using compiled knowledge of typical failures of problem decompositions. Our method combines the approaches of earlier plan-based and case-based design critics we have developed. It addresses the problem of matching cases which stymied the case-based critic. It also helps solve the problems of unfocused search and naive modelling which were the principle drawbacks of the plan-based critic.

There remain numerous open research issues for the approach. Two issues in particular may be of interest to this workshop.

First, can we store scenarios on transformations which are specific enough to be more useful than simply running the planner? For example, the incompatibility conspiracy scenario was extremely general, and costly to instantiate. OPIE could possibly find the associated plan just as quickly by directly analyzing the design state. One rejoinder is that the transformation associated with the incompatibility conspiracy scenario, responsibility assignment, is too general to have useful scenarios associated with it. Increasing the grain size of transformations, and placing scenarios only on the large-grained transformations, might give better results on evaluation, but at a cost of increasing the size of Critter's transformations and complicating their application. The research issue: how can we evaluate the trade-off between more effective evaluation knowledge, versus more general problem decomposition methods?

Related to the issue of transformation versus scenario grain size is the question of combining multiple failure scenarios. For example, consider the step which split the NoDelays goal. In that step, we applied Sequential Split, which combined three smaller transformations (Zone Defense and two responsibility assignments). The result was tested by scenarios stored on Sequential Split. Suppose instead we had applied the three primitive transformations. How should we merge the separate stored failure scenarios into a combined scenario; or, alternatively, how can we decide which of the scenarios is the most important to run?

Related Work

Our work extends and formalizes that of Feather [1987], who proposed the concept of responsibility assignment and informally demonstrated a development methodology based on decomposition and assignment of constraints. [Dubois, 1990] developed a constraint formalism, and a development methodology incorporating responsibility assignment, which has influenced our own work.

The decomposition design process can be viewed as a multi-agent extension of "operationalization" [Mostow, 1983]. Mostow's FOO and BAR systems designed problem-solving programs by decomposing and weakening constraints until they were expressible in terms of easily computable functions. The problem-solving systems we are designing, however, incorporate a broad range of social, hardware, and software systems. Consequently, it is difficult to state a compact operationality criterion for a given design problem. We rely on the human analyst to judge operationality. Similarly, constraint violations in our design problems may have consequences ranging from trivial to life-threatening. Weakening and approximating constraints therefore is much more problematic; we do not attempt to address it with our current research.

[Steier and Kant, 1985] argue for the importance of execution in designing algorithms. Our style of critiquing is motivated by similar considerations. The approach we propose grows out of our previous work on case-based [Fickas and Nagarajan, 1988] and planner-based [Anderson and Fickas, 1989] critics. [Dubois and Hagelstein, 1988] propose a slightly different approach to critiquing: derive implications by forward inference over the constraints, and present them to the user for validation. A critic using this approach requires knowledge to decide which deductions to make; abstract failure scenarios provide our method with this guidance.

Critter's critiquing task is similar to that of failure critics in planning systems such as CHEF [Hammond, 1989].The failure critics of CHEF attempt to steer CHEF's planner away from two types of failures:

1. Planning failures. These occur when the planner generates a plan that does not meet its goals, due to a false move by the part of the planner e. g. misordering two interacting steps.
2. Expectation failures. These occur when the planner generates a plan which does not meet its goal when executed in the environment of interest. Expectation failures arise when the planner's knowledge of its domain is incomplete or incorrect.

CHEF includes mechanisms for learning new failure critics from past planning or expectation failures. It also automatically indexes failures to planning moves that avoid those failures, and to moves which repair those failures.

In Critter, the "planner" is the user, and the "planning moves" are the transformations in Critter's library. The failure scenarios on a transformation identify both planning failures and expectation failures which could arise from using that transformation.

Critter does not, however, automatically learn failure scenarios from failures when they are encountered, due to the generality of its transformation library. CHEF was designed to operate within a fairly specific task domain (its example domain was Szechuan cooking). Consequently, it did not have to be too "finicky" in its choice of failures to learn [Minton, 1990]. In contrast, we hope to reuse Critter's knowledge base across diverse domains, such as transportation systems, network applications, and resource management systems. This makes it more difficult to automatically decide whether a given failure scenario is worth storing, and at what level of abstraction it should be stored. Our initial focus is thus on automated reuse of handpicked failure scenarios; learning the scenarios from previous design effort is a topic for future work.

Similarly, Critter does not automatically index from failures to avoidance or repair transformations. The "plans" (formal specifications) that Critter produces are allowed to contain more complex operators -- iterative, conditional, and uninstantiated operators, for example -- than the plans of CHEF. This makes it harder to explain a failure, assign blame for the failure to specification components, and index through those components to relevant transformations. For the present, we rely on the designer to perform indexing, but view it as an important area for future research.

References

[Anderson and Fickas, 1989] J. S. Anderson and S. Fickas, "A proposed perspective shift: viewing specification design as a planning problem," in *Proc. 5th International Workshop on Software Specification and Design* (Pittsburgh, PA, USA). Los Alamitos, CA: IEEE Computer Society, 1989, p. p. 177-184.

[Conklin and Begeman, 1988] J. Conklin and M. Begeman, "gIBIS: a hypertext tool for exploratory policy discussion," *ACM Trans. Office Information Systems*, Vol. 6, No.4, p. p. 303-331, October 1988.

[Dubois, 1990] E. Dubois, "Supporting an incremental elaboration of requirements for multi-agent systems," in *Proceedings: International Conference on Cooperating Knowledge-based Systems* (Keele, UK), 1990. Springer-Verlag: 1990, p. p. 130-134.

[Dubois and Hagelstein, 1988] E. Dubois and J. Hagelstein, "A logic of action for goal-oriented elaboration of requirements," in *Proc. 5th International Workshop on Software Specification and Design* (Pittsburgh, PA). Published as *ACM SIGSOFT Engineering Notes*, Vol. 14, No. 3, p. p. 160-168, May 1988.

[Feather, 1987] M. S. Feather, "Language support for the specification and development of composite systems," *ACM Trans. Programming Languages and Systems*, Vol. 9, No. 2, p. p. 198-234, November 1987.

[Feather, Fickas, and Helm, 1991] M.S. Feather, S. Fickas, and B. R. Helm, "Composite system design: the good news and the bad news," in *Proc. Fourth Annual KBSE Conference* Syracuse, NY, October 1991.

[Fickas and Helm, 1992] S. Fickas, B. R. Helm, "Knowledge representation and reasoning for the design of composite systems," *IEEE Transactions on Software Engineering*, Vol. 18, No. 6, June 1992, to appear.

[Fickas, Feather, & Helm, 1991] S. Fickas, B. R. Helm, and M. S. Feather, "When things go wrong: predicting failure in multi-agent systems," Dept. of Comp. and Info. Sci., Univ. of Oregon, Tech. Report CIS-TR-91-15, 1991 (Presented at the Niagara Workshop on Intelligent Information Systems, Niagara, NY, July 1991).

[Fickas and Nagarajan, 1988] S. Fickas, P. Nagarajan, "Being suspicious: critiquing problem specifications," in *Proceedings: AAAI-88 The Seventh National Conference on Artificial Intelligence* (St. Paul, Minn), 1988. AAAI Press, 1988, p. p. 19-24.

[Hammond, 1989] K. J. Hammond, *Case-Based Planning: Viewing Planning as a Memory Task*. Boston, MA: Academic Press, Inc., 1989.

[Johnson and Feather, 1991] W. L. Johnson, M. S. Feather, "Using evolution transformations to construct specifications", in M. Lowry and R. McCartney (eds.), *Automating Software Design*. AAAI Press, 1991.

[Minton, 1990] S. Minton, "Quantitative results concerning the utility of explanation-based learning," *Artificial Intelligence*, Vol. 42, p. p. 363-392. Also in J. W. Shavlik and T. G. Dietterich (eds.), *Readings in Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1990, p. p. 573-587.

[Mostow, 1983] J. Mostow, "A problem solver for making advice operational," in *Proceedings: AAAI -83*. Morgan Kaufman, 1983, p. p. 279-283.

[Robinson, 1990] W. Robinson, "A multi-agent view of requirements," in *Proc. 12th International Conference on Software Engineering* (Nice, France), 1990, p. p. 268-276.

[Steier and Kant, 1985] D. Steier, E. Kant, "The roles of execution and analysis in algorithm design," *IEEE Transactions on Software Engineering*, Vol. 11, No. 11, Nov. 1985.

[Wilbur-Ham, 1985] M. C. Wilbur-Ham, "Numerical petri nets -- a guide", Telecom Australia Research Laboratories, Report 7791, 1985.

Learning Impasses in Problem Solving

J.P.E. Hodgson
Center For Machine Learning
Department of Mathematics and Computer Science
St. Joseph's University
Philadelphia. PA 19131
jhodgson@sju.edu

Abstract

Problem Solving systems customarily use backtracking to deal with obstacles that they encounter in the course of trying to solve a problem. This paper outlines an approach in which the possible obstacles are investigated prior to the search for a solution. This provides a solution strategy that avoids backtracking.

Introduction

Many weak methods of problem solving are based upon the idea that a problem can be solved by choosing a sequence of goals and satisfying them in some order. GPS (Newell and Simon 1972) was amongst the first to set out this approach. Since then the work of Ernst and Goldstein (Ernst and Goldstein 1982), Korf (Korf 1985), and Guvernir (Guvernir 1987) has built upon this idea. The culmination of this kind of approach is, in some ways, the Soar system, which through the creation of a large production system with learning capabilities is able to incorporate many of the weak problem solving systems into a single system.

If one compares Soar and Korf's system they take quite distinct approaches to the problem of what should be learned and when it should be learned. Korf's system is able to specify in advance exactly what macros it needs to learn. This will yield benefits in the system's ability to determine which macro to use at a given point in the solution, at the price of requiring long searches for some of the more complex macros. Soar on the other hand learns only the solutions to the difficulties that actually arise. This conservative attitude toward learning means that the system can encounter problems in matching expensive chunks that do not arise in Korf's situation.

This paper looks for a half way house between these two strategies. We would like to obtain the benefits of easier pattern matching afforded by Korf's system without having to pay the price of the large amount of search that his system needs. Our approach is to show that for a substantial number of problems one can anticipate the impasses that will be encountered by a problem solver. These can then be modeled and solved in small pieces of the larger problem, thus avoiding the deep searches required in Korf.

Problems, Strategies and Impasses

We review briefly the definitions that we will need. A consensus as to the appropriate definitions seems to be emerging (Banerji 1983), (Benjamin et al. 1989), and (Niisuma and Kitahashi 1985). Our definitions follow this trend. Some of them have appeared previously in (Hodgson 1989).

Problems and Subproblems

Our definition of a problem is based upon the idea of an action.

Definition 1 *A* free problem P *is a triple* (S, Ω, a) *where* a *is a partial map*

$$a : S \times \Omega \to S$$

The set S is called the state space of the problem and the set Ω is the move set of the problem. The map a represents the effect of the moves on the state space. The effect of a move ω on the state s is to give the state $a(s, \omega)$. The element $a(s, \omega)$ fails to exist precisely when (s, ω) is not in the domain of a; that is when ω cannot be applied to the state s. A sequence $\Sigma = (\omega_1, \ldots, \omega_k)$ of moves on P is called *admissible* at s if the composition

$$a(s, \Sigma) = a(a(\ldots(s, \omega_1), \omega_2) \ldots, \omega_k) \ldots)$$

exists.

We need a notion of maps between problems.

Definition 2 *Given two problems* $P_1 = (S_1, \Omega_1, a_1)$ *and* $P_2 = (S_2, \Omega_2, a_2)$, *with a pair of maps* $f : S_1 \to S_2$ *and* $g : \Omega_1 \to \Omega_2$ *The pair (f,g) defines a* strict homomorphism $F : P_1 \to P_2$ *provided that*

1. *Given two points* s_1 *and* s_2 *such that* $f(s_1) = f(s_2)$, *then if the move* ω *applies to* s_1 *it also appplies to* s_2 *and*
2. *The equation*

$$f(a_1(s, u)) = a_2(f(s), g(u))$$

is satisfied in the sense that whenever the right-hand side exists so does the left-hand side.

A strict homomorphism F is a **monomorphism** *if the underlying maps f and g are one to one.*

We now turn to the notion of (strict) isomorphism.

Definition 3 *Two problems P_1 and P_2 are* **strictly isomorphic** *if there exists a pair of mutually inverse strict homomorphisms $F : P_1 \Rightarrow P_2$ and $H : P_2 \Rightarrow P_1$ between them.*

We can use monomorphisms in a natural way to define subproblems.

Definition 4 *Let $P = (S, \Omega, a)$ be a problem. A problem $P_0 = (S_0, \Omega_0, a_0)$ is a* **(strict) subproblem** *of P if there exists a problem monomorphism $(f, g) = F : P_0 \Rightarrow P$.*

It is worth noting that the requirement that the underlying state map be a monomorphism has the effect that even the weaker definitions of homomorphism such as the weak homomorphism of Niisuma and Kitahashi (Niizuma and Kitahashi 1985) lead to the same subproblems.

As an example of the concepts developed here we can take the sliding tile puzzles. In particular we might take the fifteen puzzle. The state space is then the set of all legal arrangements of the fifteen tiles and the blank in the 4×4 array. The move set can be given by the set $\Omega = \{U, D, L, R\}$ where the letter indicates the direction in which a tile is moved. A typical subproblem can be obtained by restricting one's attention to the tiles in the top half, (assuming that the blank lies in the top half). Moves on this subproblem are restricted to be those in which the blank remains in the top half of the array.

Strategies

So far we have not recognised that problems are supposed to represent things that are to be solved. To do this we define a **problem instance** for a problem P as a triple (P, σ, G) where σ is a state of P called the start state, and G is a subset of the state space called the goal set. A *solution* to the problem instance is a sequence Σ of moves which is admissible at σ and such that $a(\sigma, \Sigma) \in G$.

Informally a strategy is a sequence of intermediate subproblem instances between the initial state and the goal. We can distinguish two classes of strategies. In the first the successive state spaces overlap; we call these ample strategies. In the other the successive state spaces are disjoint; we call these abutment strategies.

Definition 5 *An* **ample strategy** *for a problem instance $(S, \Omega, a, \sigma, G)$ is a sequence $\{P_0, \ldots, P_k\}$ of subproblems of $P = (S, \Omega, a)$ such that the state spaces of successive subproblems have non-trivial intersection, that is $S_{i-1} \cap S_i \neq \emptyset \; \forall i \in 1, \ldots, k$. Furthermore $\sigma \in S_0$ and $G \subseteq S_k$.*

An **abutment strategy** *for a problem instance is a sequence $\{P_0, \ldots, P_k\}$ of subproblems of P such that*

1. $\sigma \in S_0$,
2. $G \subseteq S_k$,
3. $S_{i-1} \cap S_i = \emptyset \; \forall i \in 1, \ldots k$,
4. *for each $i \in 1, \ldots k$ there exists at least one pair (x_{i-1}, x_i) of points of S such that there is a move $\omega \in \Omega$ for which $a(x_{i-1}, \omega) = x_i$.*

A solution is **based upon** *a strategy if it is obtained by concatenating a sequence of solutions to the intermediate subproblems.*

We illustrate the two kinds of strategies with examples. For our example of an ample strategy we consider the case of Fool's disk. This problem has been discussed by Ernst and Goldstein (Ernst and Goldstein 1982). It consists of four concentric rings each of which is free to rotate about the common center. Each ring has eight numbers on it, appearing at 45 degree intervals around the ring. The goal of the problem is to rotate the rings so that the sum of each radius is 12. The standard strategy is as follows:

- By using only rotations through 45 degrees, make the sum on each pair of perpendicular diameters 48. P_0 thus has the same state space as P but a smaller move set.
- By using only rotations through 90 degrees, make the sum along each diameter 24. P_1 has as state space a set of states in which the sum on each pair of perpendicular diameters is 48. The move set is again a subset of the original one.
- By using only rotations through 180 degrees make the sum along each radius 12. P_2 has as state space a set of states in which the sum along each diameter is 24 and once again the move set is a subset of the original.

This strategy, when successful (about which more later), reduces the amount of search from 8^3 moves (the center ring can be kept fixed) to 8×3 moves.

Our second example is an elegant solution of the five puzzle that has been presented by Banerji (Banerji 1990). He observes that there is a way to represent the states of the five puzzle by points on the faces of a dodecahedron. The sequence of moves that circulates the blank around the the circumference of the the puzsle moves through all the states on one face. Passage from one face to another is effected by the moves that slide the blank up and down in the centre column. The strategy in this case consists of choosing the sequence of faces (each of which is a subproblem) through which one must pass from the start to the goal.

There is an important difference between these two examples. In the second example once the strategy is chosen no backtracking over the solutions to the intermediate problems is necessary but in the case of the fool's disk it may be necessary to backtrack since it is possible that the first arrangement in which the sum on all the diameters is 24 may not lead to a solution and another arrangement is needed. Niizuma and Ki-

tahashi (Niizuma and Kitahashi 1985) give a sufficient condition for this not to occur.

Proposition 6 *Suppose that for each subproblem occurring in a strategy it is the case that any instance of the subproblem can be solved then no back tracking will be needed to construct a solution to the original problem following the strategy.*

It may seem that the restriction on the state spaces of the intermediate problems is unduly restrictive. Yet it is exactly this that is needed to avoid backtracking. Thus one aim of our approach is to find strategies for which this hypothesis is true.

Impasses

At any stage in the execution of the strategy one has a subproblem instance $(P_i, \Omega_i, a_i, \sigma_i, G_i)$ where in the case of an ample strategy G_i is $S_i \cap S_{i+1}$ or in the case of an abutment strategy G_i is the set of points of S_i from which a move to S_{i+1} is possible. We have seen that the strategy proceeds smoothly as long as these intermediate problems can be solved.

Definition 7 *An intermediate problem for which there is no solution is called an* **impasse** *for the strategy.*

This terminology follows one case of the use of the term in the **Soar** system, in so doing we are also following the usage of Ruby in (Ruby and Kibler 1989).

It is important to note that our definition of an impasse in a problem is dependent upon the strategy chosen to solve the problem. Thus in the Banerji solution to the five puzzle there are no impasses since each intermediate goal is attainable. By contrast in the more usual method in which the tiles are brought into position in a prearranged order there are impasses.

Learning the Impasses

Our approach to finding impasseless strategies is to improve an existing strategy by modifying the subproblems so that they do not contain any impasses. As an example we show that in the case of Fool's disk we can do this by enlarging the intermediate problems. This need not always be the case as we shall see in some of the examples that we discuss.

For the Fools' disk case we can consider the intermediate problems defined as follows:

- By using only rotations through 45 degrees, make the sum on each pair of perpendicular diameters 48. P_0 thus has the same state space as P but a smaller move set.
- Let P_1 have as state space the set of all states in which the sum on each pair of perpendicular diameters is 48. The move set is again a subset of the original one. It may contain some moves through 45 degrees.
- Let P_2 have as state space the set of all states in which the sum along each diameter is 24. The move set may contain moves through 90 or even 45 degrees.

It is clear that for these problems no backtracking into earlier problems is necessary.

Finding the Impasses

Problem solvers such as **Soar** (Laird et al. 1986) and the stepping stone method (Ruby 1989) find the impasses in the course of attempting to satisfy the current goal. A search procedure is then invoked to resolve the impasse and the resolution of the impasse become part of the problem solver's knowledge about the problem. This is an accurate representation of much human problem solving, but it does not tell the whole story. Often faced with a problem a human will actively consider difficulties that may arise in the course of the resolution of the problem to see if they can be solved. One advantage of such an approach offers is that it allows one to take advantage of efficient storage techniques once one has determined that a small group of chunks will be adequate to solve the problem. This addresses in some measure the problem of expensive chunks (Tambe et al. 1990).

We give here a recognition criterion that forms the basis for an algorithm that can be used to produce impasses in problems. The criterion will be stated for the cases where the strategy is based upon the idea of reducing a set of features to their goal values. We begin by formalising this notion.

Given a problem P a feature on P is a map

$$f : S \rightarrow T(f)$$

between the state space of P and some finite set $T(f)$ called he target of of the feature. A set $\mathcal{F}$ of features is called **discriminating** if given any two state s_0 and s_1 of P there is some feature $f \in \mathcal{F}$ such that $f(s_0 \neq f(s_1)$. The set is called **adequate** for a goal G if given any state s which is not a goal state there is some feature f such that $f(s)$ is not a member of $f(G)$.

The strategy associated to an ordering $\{f_1, \ldots, f_k\}$ of a set of adequate features is the sequence of subproblems $P_i = (S_i, \Omega, a_i, \sigma_i, S_{i+1})$ where S_i is the set of all states for which the features $f_1, \ldots, f_{i-1}$ have goal values, a_i is the restriction of a to $a^{-1}S_i \cap (S_i \times \Omega)$. For these strategies we can give a recognition criterion for impasses.

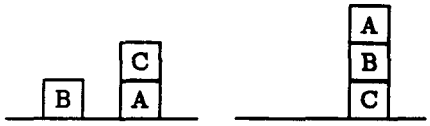
Proposition 8 *Let $P_i = (S_i, \Omega, a_i, \sigma_i, S_{i+1})$ be an intermediate problem for a strategy based upon an adequate set of features, then P_i is an impasse instance if either*

- *No move changing the value of f_i applies to σ_i, or*
- *Any sequence of moves on P that reduces f_i from σ_i must change the value of at least one of $f_1, \ldots f_{i-1}$.*

From this point forward the argument goes as follows. First, find an impasse. Second, produce a "smaller" example of the same impasse. Thirdly, expand the example to a subproblem in which the impasse can be resolved. Finally, show that the problem has a strategy based upon the new set of subproblems.

Examples of Impasses

To obtain an impasse of the first kind we can turn to Sussman's anomaly in the blocks world. The impasse can be succinctly described by the following figure.



Sussman's Anomaly

Although one can get "closer" to the goal by putting B on top of C in the position on the left hand side it will be necessary to undo this since the goal of putting A on B requires that the top of A be clear. Thus no move that will achieve the desired position for A is available.

To get an example of the second kind we consider the fifteen puzzle with the initial strategy of moving the tiles into position in the order $1, 2, 3, 4, 5, 6, 7, 8, 9, 13, 10, 14, 11, 12, 15$ (the ordering at the end is chosen to be a good one, we do not need to go this far though).

1	2	3	5
15	6		4
9	10	12	11
13	14	8	7

A Fifteen Puzzle Position

In the diagram above we find an impasse when we come to try and locate tile 4. The smallest subproblem in which this impasse appears is the 2×2 upper right hand corner in the diagram where which we place $3, 5, 4$, blank reading clockwise from the top right. (The choice of 5 is not significant.) This can be solved in the five puzzle that is obtained when we restrict attention to the top two rows and rightmost three columns of the puzzle. Furthermore we can cover the state space of the fifteen puzzle with copies of the five puzzle in the way that will be detailed in the next section and obtain an impasse free strategy.

In fact the recognition criterion given in proposition 8 permits one to write a simple program that will generate impasses in both these cases. Furthermore the expansion of the subproblem described in the example of the sliding tile puzzle will provide the means for resolving the impasses. This is the subject of the next section.

Atlases: Solving the problem

In this section we will describe a modified version of the notion of a strategy. In some sense it is a meta-strategy in that it is designed to produce an impasse free strategy for a problem by choosing the sequence of subproblems from a set of subproblems whose image cover the whole of the state space. The basic idea is that one determines what impasses may arise in the problem and then expands them to subproblems that resolve the impasses. These impasse resolving problems are then used to cover the state space of the problem giving rise to a new strategy.

Charts

It is convenient to introduce two auxiliary notions. These are chart and atlas. The idea is that chart are pieces of a problem that are all modeled on some common subproblem. The important charts will be the ones that contain the resolutions of impasses.

Definition 9 *Let P be a problem and s a state in P. Then a* **chart** *for P based upon a problem P_0 is a problem monomorphism $P_0 \rightarrow P$ whose image contains s.*

An **atlas** for a problem P is a finite collection $\mathcal{A}$ of charts such that every point in the state space of P is in the image of some chart of $\mathcal{A}$.

We define the images of two charts $f_1 : P_1 \rightarrow P$ and $f_2 : P_2 \rightarrow P$ to be *incident* if either

1. $f_1(P_1) \cap f_2(P_2)$ contains at least one move common to both subproblems, or
2. there exists a state $s_1 \epsilon f_1(P_1)$ and a state $s_2 \epsilon f_2(P_2)$ such that there is a reversible move ω with $a(s_1, \omega) = s_2$.

The *abstraction* of a problem associated to an atlas is the graph whose vertices correspond to the embedded charts of the atlas with an edge between each pair of incident charts. A sequence of pairwise adjacent charts is called a *chain*.

We will want to distinguish between two types of abstraction. An abstraction in which the charts overlap will be called an *ample* atlas. One in which all the charts are incident but do not overlap will be called an *abutment* atlas.

We give two examples of abstractions associated to an atlas. The first is based upon the earlier solution of the five puzzle. Here the charts consist of the images of the sub-problem of the five puzzle consisting of those states that are obtainable by moving the blank around the circumference of the puzzle. As Banerji remarks

(Banerji 1990) this represents the accessible states of the five puzzle on the faces of a dodecahedron. The faces of this are the points of the abstraction and the edges (which correspond to the move of the blank up or down in the middle column) correspond to the edges.

We can obtain an abstraction of the blocks world by "welding together" adjacent blocks so that we have only three big blocks to consider. Each big block is itself a blocks world and the three block world already contains the generating example of the impasse.

These two examples suggest that the correct choice of an atlas will allow one to give an impasse free strategy.

The Atlas Meta-strategy

Atlases serve as abstractions of a problem. Given a problem instance and an atlas on a problem we can define a problem instance on the atlas. The problem is to find a chain joining a chart containing the start position to a chart whose image intersects the goal.

Definition 10
Given an impasse $I = (P_0, \Omega_0, a, \sigma_0, G_0)$ on a problem, a chart $f : P_1 \rightarrow P$ is said to resolve *the impasse if there is a monomorphism of P_0 into P_1 and if the instance I can be solved in P_1.*

The main ideas of this paper can be summed up in the following.

Proposition 11 *Let $\{P_0, \ldots P_k\}$ be a strategy for a problem P and let $\mathcal{I}$ denote the set of impasses for this strategy. Let $\{Q_1, \ldots, Q_n\}$ be a set of charts of P such that each impasse is resolved in at least one of the Q_i. There is an atlas $\mathcal{A}$ based upon the charts Q_i whose associated meta-strategy gives impasseless strategies for P.*

The next section outlines a proof of this result and to a result on the length of the solutions that it produces.

Solutions and Their Length

The ideas required to construct the impasseless strategy are outlined below. The details have been worked out for the sliding tile puzzles, the Tower of Hanoi and the blocks world but in a manner that is somewhat problem dependent. Future work involves unifying the implementation so that it applies in a more problem independent way.

Resolving the Impasses

Let $P_i = (S_i, \Omega_i, a_i, \sigma_i, S_{i+1})$ be an impasse arising from the strategy based upon the set $\{f_1, \ldots, f_k\}$ of features on a problem P. The following sequence of steps is used to resolve the impasse.

SHRINK

The goal of this step is to remove from consideration those features that are not required to construct the impasse. In general given a set of features on a problem we can restrict to the moves that affect only these features. The required shrinking takes place by eliminating the features which are both fixed and whose value does not figure in the creation of the impasse.

ENLARGE

Moves that effect the remaining features are now added to produce a subproblem in which the impasse can be solved. At each step the move added should affect the smallest possible number of additional features.

An Example

We can illustrate this process with the example of the fifteen puzzle. We saw that an impasse can be reached when the first three tiles have been placed. The SHRINK process reduces this to an example equivalent to a three puzzle in which the tiles appear in the order 3, x, 4, blank, when read clockwise from the top left hand corner, (x denotes one of the possible tile values other than those already used.) We can then EXPAND to a five puzzle, which can be either horizontal or vertical in which the impasse is resolvable.

The next step is to determine whether there is an atlas for the problem whose charts are isomorphic to the set of subproblems obtained by resolving the impasses. If this is the case we then replace the original strategy by the following one. We suppose as before that we have a problem P with an adequate set of features $\{f_1, \ldots, f_k\}$. In addition we assume that there is an atlas $\mathcal{A}$ whose charts are isomorphic to the impasse resolving subproblems obtained by the process outlined above.

Using the same ordering of features that was used for the original strategy that produced the impasses.

1. Set as the current subgoal the reduction of the next feature to its goal value.
2. As each feature comes up for reduction find a chain of minimal length joining the current state to a state in the current subgoal.
3. Extract the move sequence joining the current point to one in which the feature has been reduced.

Since the atlas contains a resolution of all impasses this method will solve the problem whenever there is in fact a solution.

The Length of a Solution

We can now give an estimate for the length of a solution found using this method. We need some preliminary definitions.

L will stand for the maximum chain length required to perform the reduction of a feature.

D will stand for the maximum distance between two states in a chart. When a particular chart C is referred to we will use $D(C)$ for the distance on the chart. Note that this number can be infinite if the chart is an impasse chart.

N will be the number of features on the problem.

n will be the number of chains required to reduce all the features.

The first result is the following

Theorem 12 *Let P be a problem with an ample atlas and features with values of L, D, n as above. Then the algorithm given above finds a solution of length at most $L \times D \times n$.*

Proof. For each feature the length of chain required to reduce it does not exceed L, furthermore one each component of the chain the length of the move sequence required is less than D.□

The corresponding result for abutment atlases is the following. The proof is similar.

Theorem 13 *Let P be a problem with an abutment atlas and features with values of L, D, n as above. The algorithm above supplies a solution of length at most $n \times (L \times D + L - 1)$.*

Although these results are quite simple they give quite good estimates. For example in the case of the fifteen puzzle if we use the estimate of 22 as the maximum distance on the five puzzle (Banerji 1990) we get an estimate of $(22 \times 3 \times 15) + 3$ for the length of a solution. A more perspicuous version of the argument yields $(19 \times 22) + 3$.

Summary and Conclusions

This paper has presented a method for solving problems that constructs the impasses associated to an initial strategy in order to be able to find a new strategy in which impasses will not arise.

The method can be applied to produce short solutions to the sliding tile puzzles as well as to the blocks world. Though the implementation is at this stage still very problem dependent. Future work will produce a version that is more general.

Acknowledgements I am grateful to Ranan Banerji for his reading of earlier drafts of this paper and for numerous helpful comments during the course of this work.

References

R.B. Banerji 1983. *Artificial Intelligence: A Theoretical Approach.* North-Holland.

R.B. Banerji 1990. Heuristics: Alternatives to minimum distance and a strange state of the five puzzle. In *Intelligent Systems State of the Art and Future Directions.* (Ras and Zemankova Eds. Ellis Horwood Chichester UK .

P. Benjamin, L. Dorst, I. Mandhyan, M. Rosar 1989. An Introduction to the Decomposition of Task Representations in Autonomous Systems. Technical Report Philips Labs.

G.W. Ernst and M.M Goldstein 1982. Mechanical Discovery of Classes of Problem Solving Strategies. J ACM 29 1 1-23.

H.A. Guvenir 1987. Learning Problem Solving Strategies Using Refinement and Macro Generation. Ph.D. Thesis. Case University.

J.P.E. Hodgson 1988. Solving Problems by Subproblem Classification. Proceedings ISMIS Torino.

J.P.E. Hodgson 1989. Automatic Generation of Heuristics. In *Formal Methods in Artificial Intelligence: A Sourcebook.* Ed. R.B. Banerji. Elsevir.

R. Korf 1985. *Learning to Solve Problems by Macrogeneration.* Pitman.

John Laird, Paul Rosenbloom, and Alan Newell 1986. *Universal Subgoaling and Chunking.* Kluwer Academic.

A. Newell and H.A. Simon 1972. *Human Problem Solving.* Prentice- Hall. Englewood Cliffs. NJ.

S. Niisuma and T. Kitahashi 1985. A Problem Solving Method Using Differences or Equivalence Relations Between States. *Artificial Intelligence* 25 117-151.

David Ruby and Dennis Kibler 1989. Learning Subgoal Sequences for Planning. Proc 11th ICJAI. 609-614.

M. Tambe, A. Newell and P.S. Rosenbloom 1990. The Problem of Expensive Chunks and its Solution by Restricting Expressiveness. *Machine Learning.* 5 299-348.

When does Changing Representation Improve Problem-Solving Performance ?

Robert Holte
Computer Science Dept.
University of Ottawa
Ottawa, Ontario
Canada K1N 6N5
holte@csi.uottawa.ca

Robert Zimmer and Alan MacDonald
Dept. of Electrical Engineering
Brunel University
Uxbridge, Middx.
England UB8 3PH

Abstract

The aim of changing representation is the improvement of problem-solving efficiency. For the most widely studied family of methods of change of representation it is shown that the value of a single parameter, called the expansion factor, is critical in determining (1) whether the change of representation will improve or degrade problem-solving efficiency and (2) whether the solutions produced using the change of representation will or will not be exponentially longer than the shortest solution. A method of computing the expansion factor for a given change of representation is sketched in general and described in detail for homomorphic changes of representation. The results are illustrated with homomorphic decompositions of the Towers of Hanoi problem.

Definitions

The following definitions of the basic elements of problem solving are used throughout the paper. Only the definition of "solution" is non-standard.

A **state** is an atomic object.
An **action**, or **operator**, is a function mapping states to states.
A **plan**, or **path**, is a sequence of actions.
A **goal** is a set of states.
A **problem** is a pair consisting of an initial state and a goal.
A **problem space** is a pair consisting of a set of states and a set of actions.[1]

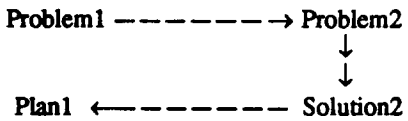
A **solution** (of Problem <S0,G>) is a sequence
S0-A1-S1-A2-...-S(n-1)-An-Sn
where Si is a state, Ai is an action mapping S(i-1) to Si, and Sn is in the goal G.

A **solution plan** (of Problem <S0,G>) is a sequence of actions mapping S0 to a state in G.

[1] This definition assumes all sets of states are allowable as goals, that all State-Goal pairs are allowable as Problems, that all sequences of composable actions are allowable as Plans, etc.. This assumption is not essential to any of the results that follow.

Change of Representation

Problem space Pspace2 is a change of representation of problem space Pspace1 if there exists a relation, R, between the two problem spaces that "preserves" solutions in the following sense. If R maps a problem, Problem1, in Pspace1 to Problem2 in Pspace2, and Solution2 is a solution of Problem2, and R maps Solution2 to Plan1 (a plan in Pspace1), then Plan1 is a solution plan of Problem1. This definition is depicted in the following diagram.

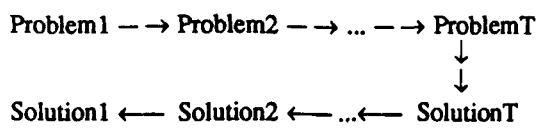


This is an extremely general definition, presupposing nothing about the nature of the indivudal mappings, nor anything about how problem spaces or the mappings are implemented.

The net effect of change of representation is to "decompose" problem-solving in Pspace1 into a three step computation:
(1) translating a given problem into a problem in Pspace2
(2) problem-solving in Pspace2, and
(3) translating the solution back into Pspace1.

Change of representation may be applied recursively to any of these three steps. Most commonly (for example in "hierarchical" problem-solving (Knoblock,1991)), it is applied repeatedly to step (2) until this step becomes

trivial. Diagrammatically, this may depicted as:



The rightmost problem space in this diagram can always be assumed to be the trivial space consisting of one state and one operator (the identity). In this way the explicit problem-solving step is entirely eliminated: a problem is solved by being translated into the trivial problem space and then translating the solution back into the original problem space. The total cost of problem-solving, then, is the sum of the costs of the two translations.

Solution Refinement

Within the preceding general strategy for problem-solving by change of representation there are many possible variations. One of these variations, called "solution refinement" is the subject of this paper. Solution refinement is defined by the two following properties. First, the only complex computation is the translation of a solution in Pspace(K) to a solution in Pspace(K-1). This computation is called refinement. Second, refinement preserves the structure of the solutions, in the following sense. Suppose the solution in Pspace(K) is

S0-A1-S1-A2-...-An-Sn

A refinement of this solution must have the form

X0-I0-RS0-RA1-X1-I1-RS1-RA2-...-Xn-In-RSn

where

- RSi is a state in Pspace(K-1) corresponding to state Si,
- RAi is an action in Pspace(K-1) corresponding to action Ai and defined on RSi,
- Xi is the result of applying action RAi to state RS(i-1) (X0 is the start state in the problem to be solved in Pspace(K-1)),
- and Xi-Ii-RSi is a solution to the problem of getting from Xi to RSi (if it happens that Xi=RAi, then Ii is empty).

Every action in a solution has a counterpart in the refinement of the solution, and usually there will be new actions added (the non-empty Ii). Therefore, a refinement will usually be longer, and can never be shorter, than the solution it is based on. In other words, as the initial "trivial" solution is translated back to become a solution in Pspace1 it grows longer and longer — it expands each time it is refined. The "expansion factor" (pp.10-11, (Stefik & Conway,1982)) is defined as the average ratio of the length of a refinement to the length of the solution from which it was derived. An equivalent definition, which will be useful later, is that the expansion factor is the average number of states in the segments Xi-Ii-RSi.

Solution refinement, in various forms, is the oldest and most widely studied method of change of representation. It is most often associated with the use of "abstractions", as in ABSTRIPS (Sacerdoti,1974), NOAH (Sacerdoti,1977), ALPINE (Knoblock,1991), and ABTWEAK (Yang & Tenenberg,1990). But solution refinement, as a strategy for problem-solving, is equally applicable to many ways of decomposing a problem space, not only to abstraction. For example, in our research (Zimmer et al.,1991), Pspace2 may be any refinable homomorphic image of Pspace1: that is, the mapping between Pspace1 and Pspace2 may be any many-to-one mapping of states to states and operators to operators[2] such that: (1) the behaviour of operators is preserved, and (2) there exists a refinement of every solution in Pspace2. Examples of solution refinement and homomorphic decompositions are given in the next section.

The Towers of Hanoi Problem

Although there are several different ways to define the Towers of Hanoi problem space, in this paper we will follow the standard definition. A state is defined by naming the peg on which each of the disks sits. There are 3 pegs, and any disk may be on any peg, so if there are D disks there are 3^D states. An operator is defined by naming a disk and a direction (clockwise or anticlockwise): thus there are 2D operators, given D disks. The effect of an operator is to move the specified disk from its current peg to the next peg in the specified direction. An operator is defined on a state only if all the disks smaller than the disk to be moved are on the peg that is

[2] We are currently exploring the use of many-to-many mappings of operators, called "distributed representations" in (Holte,1988).

not affected by the operator.

In the 2-disk Towers of Hanoi problem there are 9 states, { <S,L> | S, L ∈ {1,2,3}}, where S indicates the peg of the smaller disk and L the peg of the larger disk. The 4 operators are SC, SA, LC, LA, where S (L) indicates the smaller (larger) disk, and C (A) indicates clockwise (anticlockwise).

In the 1-disk Towers of Hanoi problem, there are 3 states {<1>,<2>,<3>}, indicating the position of the lone disk, and two operators, C and its inverse A.

The Standard Decomposition

There are exactly four homomorphisms of the 2-disk space, as defined above, onto the 1-disk space. The standard one can be summarized in English as: "ignore the position of the smallest disk". Formally, in this decomposition state <S,L> is mapped to state <L>, operators LC and LA are mapped to C and A, respectively, and operators SC and SA are mapped to the identity.

To illustrate the use of this decomposition in problem-solving, consider the problem in which the start state is <1,1> and the goal is <3,3>. This 2-disk problem is mapped into the 1-disk space. The translated problem has <1> as a start state and <3> as a goal. The 1-disk solution is <1>-A-<3>. This solution implicitly has identity operators acting on states <1> and <3>. Refinement must now map this solution into a solution to the original problem. Operator A maps back uniquely to the operator LA, but the states do not map back uniquely, nor do the implicit identity operators. For example, the identity operators on state <1> map back to any sequence of operators which, when applied to a state in which the larger disk is on peg 1, lead to a state in which the larger disk is on peg 1.

Our refinement algorithm works by translating a solution in Pspace(K), SolutionK, into a sequence of subproblems to be solved in Pspace(K-1). Each State-Operator fragment in SolutionK is translated into a goal to be solved starting at the final state of the previously solved subproblem (or, in the case of the first goal of this form, starting at the given start state of Pspace(K-1)). In the present example, SolutionK is <1>-A-<3>: this is translated into the goal "<1>-A", whose meaning is "reach a state in which operator LA is applicable and the larger disk is on peg 1". Problem-solving commences from the start state in Pspace(K-1), <1,1>, and proceeds, as usual, until this goal is satisfied. In this case the solution is <1,1>-SC-<2,1>. This solution is the refinement of the <1>- segment of SolutionK. Note that the one state in SolutionK has been expanded into 2 states in this refinement: this expansion factor is the key in determining whether this decomposition will improve or degrade the efficiency of problem-solving.

Continuing with the example, operator LA is added to the solution, along with the state (<2,3>) to which it leads from <2,1>. The state <2,3> will be the start state for the next subproblem in the refinement process. Because we have finished with all the operators in SolutionK, only the final refinement subproblem remains: the goal is to reach <3,3>, the goal state in Pspace(K-1). Problem-solving commences from the state <2,3> and finds the solution <2,3>-SC-<3,3>. This is the expansion of the -<3> segment of SolutionK: as before, there is an expansion factor of 2. The final solution to the original problem is created by linking together all the solutions to the refinement subproblems, giving <1,1>-SC-<2,1>-LA-<2,3>-SC-<3,3>.

Non-Standard Decompositions

In the first non-standard decomposition, state <S,L> is mapped to state <P> if S and L are both equal to P or if both are different than P. Thus, states <1,1>, <2,3>, and <3,2> map to state <1>, states <2,2>, <1,3>, <3,1> map to state <2>, and states <3,3>, <1,2>, <2,1> map to state <3>. Operators SA and LC both map to operator A, and SC and LA both map to C.

In the second non-standard decomposition, state <S,L> is mapped to state <S>: that is, the position of the larger disk is ignored. Thus, states <1,1>, <1,2>, and <1,3> map to state <1>, states <2,1>, <2,2>, <2,3> map to state <2>, and states <3,1>, <3,2>, <3,3> map to state <3>. Operators LA and LC map to A and C, respectively, and SA and SC both map to the identity.

In the final decomposition, state <S,L> is mapped to state <S-L+1>, where the subtraction is done modulo 3. In other words, the mapping is based on the relative positions of the two disks. States in which the two disks are on the same peg — <1,1>, <2,2>, and <3,3> — are mapped to state <1>. States in which the smaller

disk is one peg "ahead" of the larger disk — <2,1>, <3,2>, and <1,3> — are mapped to state <2>. And states in which the smaller disk is one peg "behind" the larger disk — <1,2>, <2,3>, and <3,1> — are mapped to state <3>. Operators LC and SC both map to operator C, and operators LA and SA both map to operator A.

When any of these decompositions is applied to the N-disk Towers of Hanoi problem space the resulting space is isomorphic to the (N-1)-disk space. Hence the same decomposition can be applied repeatedly to produce a sequence of successively smaller problem spaces ending with the trivial problem space.

Problem-solving Efficiency

The aim of all kinds of change of representation, including solution refinement, is to improve the efficiency of problem-solving. Consequently, it would be useful to be able to predict the change in problem-solving efficiency that would result by making a particular change of representation. This ability would enable a system to select the most efficient among a set of possible changes of representation — for example, to select the best of the four decompositions of the 2-disk Towers of Hanoi problem. And, accompanied by an estimate of the problem-solving efficiency of the original problem representation, this ability would enable a system to determine whether any of the changes of representation is actually an improvement.

It is not difficult to analyze the efficiency of solution refinement methods under the assumption that the expansion factor at every level is the same. Let A be the number of nontrivial problem spaces, and X be the expansion factor. Then the length of the final solution is X^A. If W[X] denotes the amount of "work" required to refine a single state-operator solution fragment, then the total amount of work required to create the final solution is $W[X]*(X^A-1)/(X-1)$.

In his thesis (Knoblock,1991), Craig Knoblock observes that if X is a constant and A is proportional to the logarithm of the optimal solution length, then the work required by solution refinement is exponentially less than the work required by a brute force problem-solver in the original (undecomposed) problem space. These circumstances hold when the standard decomposition is used to solve Towers of Hanoi problems in which all disks are initially on the same peg.

This formula for "work" provides a direct way to evaluate the efficiency of different decompositions of a problem space, providing that one can compute W[X] and measure the values of X and A for a given decomposition. In fact, the only real difficulty is the calculation of X. The number of non-trivial problem spaces is normally self-evident, and the term W[X] is almost always negligible compared to X^A. Note that with the values of X and A we can calculate the expected length of a solution as well as the expected amount of work required to create it.

To see how to calculate X, recall that X, the expansion factor, is (by definition) equal to the average number of states in the segments "Xi-Ii-RSi" that are inserted during refinement. If the method used to change representation imposes constraints on the possible values of Xi and RSi, then these constraints may provide enough information to compute an expected value, or at least an upper bound, on X. For example, in homomorphic decompositions it must be the case that Xi and RSi are "equivalent", i.e. that they are mapped to the same state by the homomorphism. Given this fact, the expected value of X is simply the "average" length of the shortest path (operator sequence) between each possible pair of equivalent states. "Average" is in quotes because the actual probability of encountering each of the <Xi,RSi> pairs in practice is normally unknown.

To illustrate this computation, consider the standard decomposition of the 2-disk Towers of Hanoi problem space. 9 different <Xi,RSi> pairs can be constructed from the 3 states that map to <1>. Of these 9 pairs, 3 are of the form <S,S>, 3 are of the form <S,SC(S)>, and 3 are of the form <S,SA(S)>. The shortest path connecting S to S has a length (number of states) of 1, and the shortest path connecting S to SC(S) or SA(S) has a length of 2. The same analysis holds for the the states that are mapped to <2>, and for those that are mapped to <3>. Therefore the expected value of X, assuming all pairs of equivalent states are equiprobable, is (3*1 + 6*2)/9, or 5/3. This turns out to be impossibly low — in the N-disk Towers of Hanoi problem X must be larger than twice the Nth root of 2/3 — an indication that all pairs are not actually equiprobable. Nevertheless, this value may still be useful to

compare with the value of X computed in the same manner for the other decompositions.

The three other decompositions all have the same expected value of X, namely 2.56. This is considerably larger than the value for the standard decomposition. Thus we expect that the standard decomposition will produce shorter solutions with less work than the other decompositions. To test this prediction, the standard decomposition and the second non-standard decomposition were used to solve all N-disk problems in which all disks are initially on peg 1. Work is measured as the number of arcs traversed by a breadth-first problem-solver before finding a solution[3]. The results of this experiment are:

# of Disks	WORK	
(N)	Standard	Non-Standard #2
2	8.7	12.8
3	25.4	47.1
4	63.0	133.0
5	142.0	328.0

# of Disks	SOLUTION LENGTH	
(N)	Standard	Non-Standard #2
2	3.0	3.2
3	5.7	6.7
4	11.0	14.3
5	21.7	29.9

The ratio of successive solution lengths gives a true indication of the actual expansion factors of the two decompositions: X is optimal (slightly less than 2.0) for the standard decomposition and 2.1 for the non-standard decomposition. The difference in expansion factors is much smaller than predicted, but still results in a significant difference in solution lengths and the work required.

If a formula is available to compute the expected amount of work required for problem-solving in the original (undecomposed) problem space, then this can be compared to the work formula for solution refinements to determine whether a given decomposition will improve or degrade efficiency. In the N-disk Towers of Hanoi problem space the expected amount of work is half the number of arcs in the entire space (assuming that the problem-solver never traverses the same arc twice), which is given by the formula $3*(3^N-1)/2$. Because this formula has the same form as the work formula for solution refinement, it follows immediately that a decomposition will degrade performance on the N-disk Towers of Hanoi if and only if its expansion factor is 3 or greater.

In the same way that the work required with and without a change of representation can be compared, so too can solution length be compared. A breadth first problem-solver always finds a minimal length solution. In the N-disk Towers of Hanoi problem space, the minimum solution length, for the average problem in which all disks are initially on peg 1, is $(2^{(N+1)}+1)/3$. Comparing this to the expected solution length for solution refinement, it follows that a decomposition will produce exponentially longer solutions whenever its expansion factor is greater than 2.

The fact that the critical value of the expansion factor is different for solution length and work-required leads to the apparent paradox that some decompositions will construct exponentially longer solutions and yet do exponentially less work. In fact, the second non-standard decomposition exhibits this phenomenon, as the following data shows (the experimental conditions are the same before).

# of Disks	WORK	
(N)	Original Space	Non-Standard #2
2	10.8	12.8
3	37.8	47.1
4	118.8	133.0
5	361.9	328.0

# of Disks	SOLUTION LENGTH	
(N)	Original Space	Non-Standard #2
2	3.0	3.2
3	5.7	6.7
4	11.0	14.3
5	21.7	29.9

[3] Unlike the problem-solver in Knoblock's analysis, this problem-solver never traverses the same arc twice. This simple bookkeeping usually results in an exponential reduction in the work required.

Conclusion

The aim of changing representation is the improvement of problem-solving efficiency. For the most widely studied family of methods of change of representation it has been shown that the value of a single parameter, called the expansion factor, is critical in determining (1) whether the change of representation will improve or degrade problem-solving efficiency, and (2) whether the solutions produced using the change of representation will or will not be exponentially longer than the shortest solution. A method of computing the expansion factor for a given change of representation has been sketched in general and described in detail for homomorphic changes of representation. The results have been illustrated with homomorphic decompositions of the Towers of Hanoi problem.

References

Holte, R. C. (1988), "An Analytical Framework for Learning Systems", Ph.D. dissertation, Brunel University. also TR-AI88-72, Computer Sciences Dept., Univ. of Texas at Austin.

Knoblock, C.A. (1991), "Automatically Generating Abstractions for Problem Solving", technical report CMU-CS-91-120, Computer Science Department, Carnegie-Mellon University.

Knoblock, C.A., J.D. Tenenberg, and Q. Yang (1991), "Characterizing Abstraction Hierarchies for Planning", Proc. IJCAI, pp.692-697.

Sacerdoti, E. (1974), "Planning in a hierarchy of abstraction spaces", Artificial Intelligence 5(2):115-135.

Sacerdoti, E. (1977), "A Structure for Plans and Behaviour", American Elsevier Publishers.

Stefik, M. and Lynn Conway (1982), "Towards the Principled Engineering of Knowledge", The AI Magazine, vol.3, no.3, pp. 4-16.

Yang, Q. and J.D. Tenenberg (1990), "Abtweak: Abstracting a nonlinear, least commitment planner", Proc. AAAI, pp.204-209.

Zimmer, R., Alan MacDonald, and Robert Holte (1991), "Reasoning about Representations: Towards the Automation of Representation Change", Proceedings of the Florida Artificial Intelligence Research Symposium, pp. 201-205.

Domain and Specification Models for Software Engineering

Neil Iscoe, Zheng-Yang Liu, Guohui Feng
EDS Research, Austin Laboratory
1601 Rio Grande, Ste 500
Austin, Texas 78701
iscoe@austin.eds.com

Abstract

This paper discusses our approach to representing application domain knowledge for specific software engineering tasks. Application domain knowledge is embodied in a domain model. Domain models are used to assist in the creation of specification models. Although many different specification models can be created from any particular domain model, each specification model is consistent and correct with respect to the domain model. One aspect of the system—hierarchical organization is described in detail.

Introduction

Creating, maintaining and evolving software systems requires an understanding of both programming knowledge and application domain knowledge. Programming knowledge is relatively well understood. It is formal, modeled in a variety of ways, explicit enough to be taught to novices, and general enough to apply across many domains. Although empirical field studies (Curtis, et al., 1988) have shown that application domain knowledge is critical to the success of large projects, this knowledge is rarely modeled as needed. It is usually implicitly embodied in the application code rather than explicitly recorded and maintained separately from the code. Even when the knowledge is recorded, it is generally stored in voluminous natural language documents in an informal rather than a formal manner. Although problem-specific languages partially remedy this situation, they still capture domain knowledge in an ad hoc rather than a systematic manner. Furthermore, these languages are generally not designed in such a way that the results can be generalized or even replicated.

Application domain models are representations of relevant aspects of application domains that can be used for different operational goals in support of specific software engineering tasks or processes. Domain models determine what there is in the world for reasoning about given application domains and sanction the types of inferences allowed.

Operational goals are always implicit in the construction of a domain model and are essential to understanding the form and content of that model. Unlike generalized knowledge representation projects such as Cyc (Lenat, 1990) that attempt to provide a basis for modeling encyclopedic knowledge, domain modeling explicitly acknowledges the commonly held view (Amarel, 1968) that representations are designed for particular purposes. These purposes–the operational goals–inherently bias any particular solution and dictate the final form of the model. As real-world domains are infinitely rich and diverse, we inevitably adopt particular perspectives in deciding what is relevant with respect to given tasks when formulating models (Liu and Farley, 1991). Even within the field of domain modeling, many different operational goals and modeling projects are being pursued (Iscoe, et al. 1991).

In the next section, we give an overview of the domain modeling research at EDS and our corresponding operational goals. We then introduce a model reformulation concept—the generation of multiple specification models from a single domain model. The remainder of the paper focuses on one of the mechanisms which allows a specification designer to rapidly construct specification models that are consistent and correct with respect to the original domain model.

Domain Modeling Research

EDS specializes in creating software for a variety of industries. Each industry area such as utilities, finance, or health insurance has an associated body of knowledge which is critical to the understanding of specification and implementation of software systems. Domain expertise is acquired by personnel over a period of years, and the company is organized into strategic business units (SBUs) so that knowledge about a particular industry can be maintained over time.

At the EDS Austin research laboratory, we are attempting to create a domain modeling system which can achieve the following operational goals:

- Requirements & Specifications—Eliciting, verifying, and formalizing software requirements and specifications,
- Program Transformation/Generation—Transforming a specification into efficient executable code,
- Reverse Engineering—Identifying the semantics of existing code in terms of a partial specification,
- Explanation, Education & Communication—Capturing and communicating application domain knowledge.

The realization of these operational goals is consistent with our long-term plan for creating knowledge-based tools to support programming-in-the-large (Barstow, 1988) development. The domain modeling approach provides ample opportunities for investigating and creating new development paradigms.

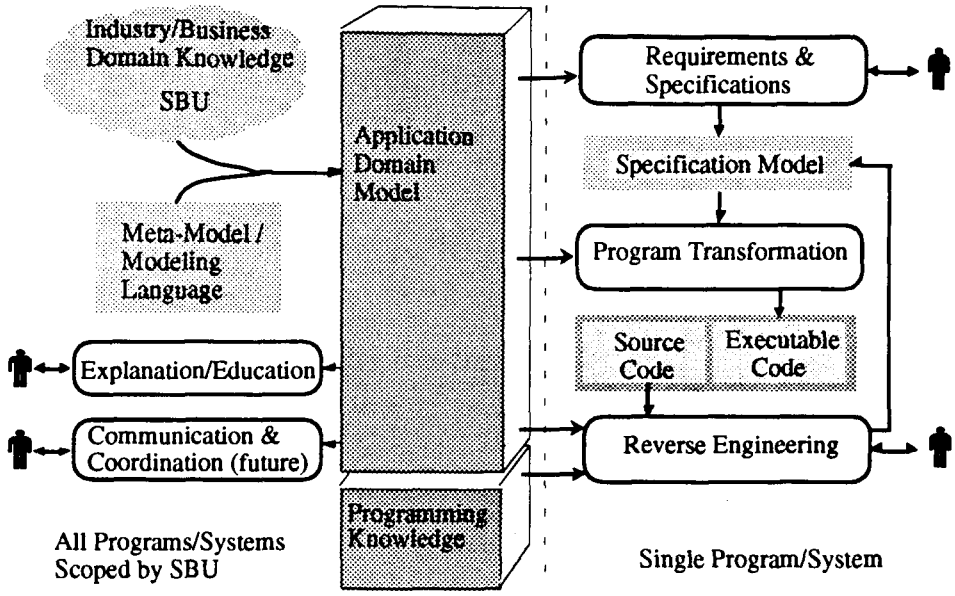


Figure 1. Domain Modeling with Operational Goals

Figure 1 illustrates the context in which we model. The industry knowledge for each SBU is instantiated into a domain model, which then serves as a source of knowledge for programs (the ovals) to achieve our operational goals. In the figure, the *specification model* (rectangle) contains the specification for a specific system within an application domain. Because one of our goals is to generate executable code, we require that any particular specification model be consistent. A very large but finite number of specification models can be created which are consistent and are correct with respect to a particular domain model.

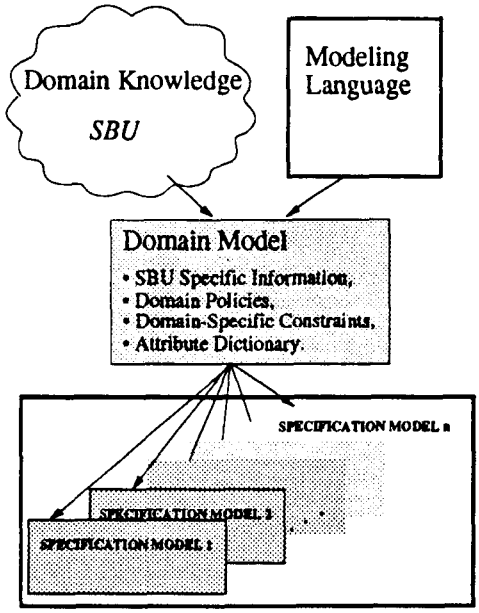


Figure 2. Instantiating Specification Models

Figure 2 illustrates the two separate modeling tasks required by our approach. Domain experts interact with a system to store their knowledge in terms of a domain model. Specification designers then use the system to build specification models which satisfy constraints in the domain model.

In order to create a specification model, the designer selects a set of relevant policies and constraints from the domain model that must be included and enforced in the specification model. The constraints include intra-attribute as well as inter-attribute relationships within and across entities relevant to the task at hand.

Dynamic Knowledge Structure

The remainder of this paper presents one aspect of our meta-model representation that is relevant to this workshop—dynamic restructuring of a hierarchically organized domain knowledge.

While most would agree that hierarchical organizational strategies provide a reasonable way to structure knowledge within complex domains, the creation of a hierarchical structure, like any type of representational scheme, imposes a particular view of the world. Unfortunately, there is no particular view that is optimal for every application. Although the programs within a particular application share the same legal, physical, and economic constraints, the construction of any particular specification model depends upon a set of policy decisions that determine how cases are handled. Furthermore, *software in the large* systems are continually changing in such a manner that the concept of a static hierarchy is insufficient to capture the process of system evolution.

Consider software systems that manage the payment of health insurance claims. Although conceptually simple, these systems handle hundreds of thousands of different

cases. One way to represent these cases is to enumerate the leaf nodes of the hierarchies created by the appropriate partitioning of attributes such as gender, age, family_status, previous_condition, employment, deductibles, copayments, prognosis, and so on. Unfortunately, the tree structure created by case expansion not only obscures relevant and interesting cases, but is also a monolithic structure. It is a paradox of object-oriented approaches that well-adapted structures are not adaptable to new situations.

Because of the combinatorial explosion of the leaf nodes, it makes sense to handle the cases at as high a level as possible. Term subsumption systems such as CLASSIC (Borgida, et al. 1989) automate this process by determining the place in a hierarchy in which terms are subsumed. But subsumption systems assume a single structure in which all sub-models can belong. In the case of applications such as health insurance, individual modules may have different hierarchical structures and still maintain the integrity and constraint rules of the domain model.

Attribute Definitions

Attributes are normally considered as data values or slot fillers within a class or frame. However, the standard treatment of attributes as lists of data values with some underlying machine representation fails both to capture sufficient semantic information from the application domain and to state definitions with sufficient formality to allow semantics-related consistency checks.

Attributes are functions which define how a set of objects is mapped within a class. One type of attribute has a value set represented by a nominal scale which consists of a set of values, $\mathcal{V}(A) = \{C_1, \ldots C_n\}$.

The semantics of an application domain are maintained by creating categories in such a way that items to be categorized with respect to a particular attribute are as homogeneous as possible within a category and as heterogeneous as possible between categories. Examples of nominal scales abound and map cleanly to the notion of enumerated type as shown below:

```
(Colors
   :type     nominal_scale
   :values   (Red Yellow Green Blue)
```

The next type of attribute is an ordinal scale—a nominal scale in which a total ordering exists among the categories. Interval and ratio scales are the more quantitative scales and add definitions of dimensions, units, and granularity.

This brief description of attribute type was included to allow the reader to understand the examples in this paper. Attributes have additional types and a number of other properties which are explained in (Iscoe, et al 1992).

Hierarchical Decomposition

Hierarchies are a natural way to view and organize information and, at some level of abstraction, are a part of most object-oriented and knowledge representation languages. Unfortunately, the simplicity of these concepts can sometimes obscure the semantics that a model is attempting to capture. That one's needs dictate one's ontological choice is a fundamental premise of knowledge engineering. The ability to systematically define a new set of attributes by partitioning the value sets of old attributes and then using these new attributes to reclassify the domain in accordance with the new requirements is a fundamental aspect of our attribute characterization. By preserving the "ontological map" as a component of the attribute, the domain modeler can shift between the differing paradigms modeled by various classes of objects.

Attribute characterization provides a representation and systematic methodology for the partitioning of attributes that facilitates the way they are organized, subdivided, and built into hierarchies. An attribute restriction is a new attribute whose value set and set of applicable relations are subsets of the original attribute.

Creating a new attribute serves the dual purpose of creating a set of views on the old attribute as well as creating a new attribute. Often, new attributes are defined in terms of old attributes by partitioning the original value set and then equating each new attribute value with an element of the partition. As an example, an accounts receivable (AR) system may use the attribute days_to_payment whose value is the average number of days it takes for the client to pay a bill.

```
(days_to_payment:
:type          ratio_scale
:dimension     time
:unit          days
:min           0
:max           360)
```

From the standpoint of AR applications, a more useful attribute might be:

```
(type_of_payer:
:type          Ordinal_scale
:Ordered_by    lateness of paymen
:values  (pays_on_time  slow_pay  dead_beat))
```

This new attribute will be defined by partitioning the value set of days_to_payment, Vp by subdividing the range of values, then equating each value with one of the elements of the partition as illustrated in figure 3 and described as follows:

```
(type_of_payer
  :mapped_from  days_to_payment
    (pays_on_time   (<=30)
    (slow_pay
         (AND (> 30) (< 90)))
    (dead_beat        (>= 90))))
```

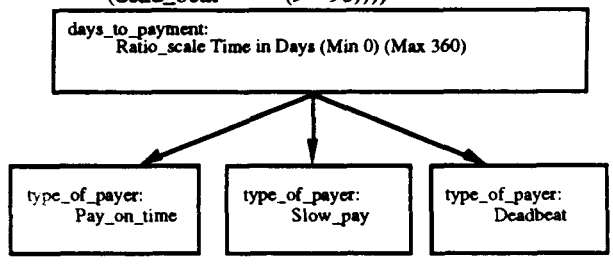


Figure 3 — Partitioning days_to_payment

Note that the days_to_payment attribute is based on a ratio scale while the type_of_payer attribute is based on an ordinal scale. In general a defined attribute represents a loss of information (in this example, the number of days overdue) in return for a more useful and inherently less detailed category.

Using Population Parameters

Population parameters facilitate the formation of new attributes. For example, some graduate admissions committees use interval-scaled GRE scores to separate applicants into acceptance categories. Population parameters allow designers to create new attributes based on restrictions to the original attribute as shown below:

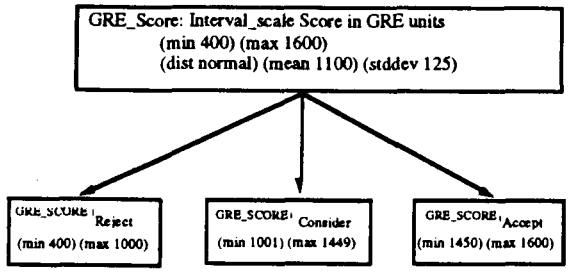


Figure 4 — Using Population Parameters to Restrict an Attribute

Figure 4 shows the GRE score as an attribute which could be attached to a student. Understanding the distribution of values within the value set of GRE scores allows application designers to create partitions in any one of a variety of ways. For example, assume that an application designer wanted to create an initial partition based on the requirement *"accept all students who score in the top x% on the GRE, consider those who score between x% and y%, and reject those who score in the bottom y%."* Given this type of requirement, the domain model contains the appropriate information to use and an algorithm to produce the correct raw score numbers to achieve such a partition.

Another way that these requirements are sometimes stated is to build a partition based on an absolute raw score. For example, a requirement like "*accept all students who score above 1450 on the GRE*" can be easily incorporated. Furthermore, this type of specification can be used interactively so that the designer can juggle between raw scores and percentiles until the partitions appropriate for the application domain are produced.

Domain and Specification Models

In this section we focus on relations between attributes within a single domain model class. For the purposes of this discussion we define the following attributes:

```
(name          :type  identifier)
(eye_color     :type nominal_scale
    :values  (brown, blue, green))
(Gender        :type  nominal_scale
    :values (male female))
(Hysterectomy  :type  ordinal_scale
    :values (Y N))
(Medicare_payment :type  ratio_scale
   :dimension  (money)
   :unit          (dollar)
   :granularity  (.01))
(Age_m  type:      ordinal_scale
   :values          (under65  65_and_over)
   :mapped_from    age
                  (under65       (< 65))
                  (65_and_over  (>= 65)))
```

Although other constraints exist, domain model classes can be regarded as consisting of sets of attributes which are either required or might be included within a particular domain model. These constraints are expressed as follows:

must_have(c, a, cond) — attribute *a* must be used in class *c* in a model if condition *cond* evaluates to true.

applicable(c, a, cond) — attribute *a* can be used in class *c* a model if condition *cond* evaluates to true.

Within any particular specification model, an attribute is simply classified as used within a class.

used(m, c, a, cond) — within model m, attribute *a* is used in class *c* in model m if condition *cond* evaluates to true.

The most straight-forward relationship between a domain model and a specification model is that must_have attributes are used in all specification models and applicable attributes are selected by the specification designer.

must_have(c, a, cond) $\leftrightarrow \forall$m used(m, c, a, cond)
applicable(c, a, cond) $\leftrightarrow \exists$m used(m, c, a, cond)
thus
must_have(c,a, cond) $\rightarrow$ applicable(c,a, cond)

For example, in a domain model, name might be required for all specification models, while eye_color could be selected only if it were appropriate for the particular specification model.

```
(person
   :must_have ((Name ())
   :applicable ((eye_color ())
   ...)
```

The application of these constraints when *cond* is vacuously true is fairly standard feature in most modeling languages of this type. However, name and eye_color are attributes which are total and are not as interesting as the cases that occur when the attributes are partial functions.

Conditions for Function Evaluation

Recalling that an attribute is a function which maps objects to a particular property, *cond* can be interpreted as the condition which must be satisfied for the attribute to be a total instead of a partial function. In other words, *cond* defines the subset which is the domain of applicability of

the partial function. For example for a person class hysterectomy is only applicable if the gender is female.

(applicable person Hysterectomy
(= Gender female))

The domain modeling system is designed so that the conditions required to establish the proper domain for an attribute are automatically maintained. These conditions are constrained in such a way that tractability is maintained and are of the form $((p_1\ a_1\ v_1)1...\ (p_n\ a_n\ v_n))$, where p_i is the name of a predicate, a_i is the name of an attribute, and v_i is a value of the attribute.

When conditions exist, the following axiom is needed:

(applicable c a cond1) →
[(used m c a cond2) → (cond1→ cond2)] (1)

A user can create a specification model with any particular class hierarchy as long as the domain policies and constraints are satisfied.

Domain and specification model consistency is maintained by a specialized theorem prover. The theorem prover, *STR+VE*, is an upgraded version of the prover presented in (Bledsoe 1980) for proofs of theorems in general inequalities. A TMS is being constructed to interface between the modeling system and the theorem prover

We are currently experimenting with ways to capture and verify domain modeling constraints by presenting redundant information in a variety of ways. We believe that many of the specification problems in large systems are created when value set changes cause a single case to be changed but fail to correct cases that were identified from a previous inference.

For example, if we assume that hysterectomy is applicable to females, the system can infer that hysterectomy cannot apply to males by using axiom 1, the definition of applicable, and the definition of gender to derive a contradiction.

applicable(c, a, cond) ↔ ∃m used(m, c, a, cond)
applicable(P, hys, [(= gender m)])
¬(= Gender, M) →(= Gender, F)
(= Gender, M) → ¬(= Gender, F)

A key point is that when people are presented with value sets they automatically and unconsciously perform substitutions such as the ones listed above. This is a reasonable way to build a model until a value set changes. In large systems, value sets are frequently changed. Consequently, conclusions that were drawn by using negation to infer values become invalid. We use the applicability of conditions and the system's knowledge of value sets to attempt to provide the proper cases for the domain modeler to check when condtions change.

Discussion

In this paper, we have presented the concept of modeling application domains in order to achieve the operational goals of program specification, code generation, and reverse engineering. The main concept is that multiple specification models can be created that are consistent and "correct" with respect to a domain model. Domain models represent information about a particular industry area. Specification models represent information about a particular system.

Domain and specification models are constructed by using a graphical interface to interactively create a set of rules based on attribute value set partitions and the preceding axioms. The system is being implemented using Motif GUI on SPARC workstations. Although it is currently operating in a single user mode, it is being designed to be accessed simultaneously by multiple domain modelers. We are also trying to accelerate the knowledge capture process by reverse engineering data models that have been captured by an existing EDS case tool and instantiating them into the appropriate domain models.

References

Amarel, S. 1968. "On Representations of Problems of Reasoning About Actions," in *Machine Intelligence 3*, D. Michie. Ed., American Elsevier, New York pp. 131-171.

Barstow, D. 1985. "Domain-Specific Automatic Programming," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 11, pp. 1321-1336.

Barstow, D. 1988. "Artificial Intelligence and Software Engineering," in Shrobe, H., ed., *Exploring Artificial Intelligence*. AAAI. Morgan Kaufmann, San Mateo, CA.

Bledsoe, W. W., and Hines, L. M. 1980. "Variable Elimination and Chaining in a Resolution-Base Prover for Inequalities," *Proceedings of the 5th Conference on Automated Deduction*, Les Arcs, France, Springer-Verlag, pp. 70-87.

Borgida, A., Brachman, R.J., McGuinness, D.L., and Resnick, L.A. 1989. "CLASSIC: A structural data model for objects," in *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pp. 59-67.

Curtis, B., Krasner, H. and Iscoe, N. 1988. "A Field Study of the Software Design Process for Large Systems," *Communications of the ACM*, vol. 31, no. 11, pp. 1268-1287.

Davis, R. 1991. "Knowledge Representation: Broadening the Perspective," AAAI-91 Panel, Anaheim, CA.

Iscoe, N, Browne, J.C., Werth, J. , and Liu, Z.Y, 1992 "Attributes – Building Blocks for Modeling Application Domains," Submitted to IEEE TSE

Iscoe, N., Williams, G. and Arango, G., Eds. 1991. *Domain Modeling for Software Engineering, Proceedings of Domain-Modeling Workshop*, Austin, Texas.

Lenat, D.B. , Guha, R.V., Pittman, K., Pratt, D., and Shepherd, M. 1990. "Cyc: Toward Programs with Common Sense," *CACM*, vol. 33, no. 8, pp. 30-49.

Liu, Z.-Y. and Farley, A. 1991. "Tasks, Models, Perspectives, Dimensions," *The 5th International Workshop on Qualitative Reasoning* Austin, Texas, pp. 1-12.

Research Summary

Craig A. Knoblock
University of Southern California
Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
knoblock@isi.edu

Reducing search in problem solving is a central issue in building systems to solve complex and interesting problems. One approach to reducing search is through the use of abstraction. My research has focused on three closely related issues: identifying the properties that comprise a useful abstraction, developing techniques for automatically generating abstractions, and making effective use of these abstractions in problem solving.

An abstraction space is formed by ignoring details of a problem space. An important property of an abstraction space is that a plan produced in that space can be refined without undoing the work performed in the abstract space. [Knoblock *et al.*, 1991b] provides a formal characterization of this property, called *ordered monotonicity*, which forms the basis of an algorithm for generating abstractions spaces.

Based on the ordered monotonicity property, I implemented a system called ALPINE that automatically generates abstractions for problem solving [Knoblock, 1990b, Knoblock, 1991a]. The system takes both a problem and an initial problem space and produces a hierarchy of abstract problem spaces that is tailored to the particular problem to be solved. ALPINE produces abstractions in a variety of domains [Knoblock, 1990a, Knoblock, 1991a, Knoblock, 1992] and then uses them for hierarchical problem solving.

The abstractions generated by ALPINE produce significant performance improvements [Knoblock, 1991b]. The hierarchical problem solver is implemented as an extension to the PRODIGY system [Carbonell *et al.*, 1991], making it possible to experiment on existing domains and both combine and contrast the use of abstraction with other types of learning. [Knoblock *et al.*, 1991a] describes the integration of ALPINE with explanation-based learning. We also plan to integrate ALPINE with the learning by analogy component in PRODIGY.

References

[Carbonell *et al.*, 1991] Jaime G. Carbonell, Craig A. Knoblock, and Steven Minton. PRODIGY: An integrated architecture for planning and learning. In Kurt VanLehn, editor, *Architectures for Intelligence*, pages 241–278. Lawrence Erlbaum, Hillsdale, NJ, 1991. Available as Technical Report CMU-CS-89-189.

[Knoblock *et al.*, 1991a] Craig A. Knoblock, Steven Minton, and Oren Etzioni. Integrating abstraction and explanation-based learning in PRODIGY. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, CA, 1991.

[Knoblock *et al.*, 1991b] Craig A. Knoblock, Josh D. Tenenberg, and Qiang Yang. Characterizing abstraction hierarchies for planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, CA, 1991.

[Knoblock, 1990a] Craig A. Knoblock. Abstracting the Tower of Hanoi. In *Proceedings of the Workshop on Automatic Generation of Approximations and Abstractions*, Boston, MA, 1990.

[Knoblock, 1990b] Craig A. Knoblock. Learning abstraction hierarchies for problem solving. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 923–928, Boston, MA, 1990.

[Knoblock, 1991a] Craig A. Knoblock. *Automatically Generating Abstractions for Problem Solving.* Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, 1991. Available as Technical Report CMU-CS-91-120.

[Knoblock, 1991b] Craig A. Knoblock. Search reduction in hierarchical problem solving. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, CA, 1991.

[Knoblock, 1992] Craig A. Knoblock. An analysis of ABSTRIPS. In J. Hendler, editor, *Artificial Intelligence Planning Systems: Proceedings of the First International Conference (AIPS92).* Morgan Kaufmann, San Mateo, CA, 1992.

Localization vs. Abstraction: A Comparison of Two Search Reduction Techniques

Amy L. Lansky

Sterling Software
NASA Ames Research Center (AI Research Branch)
MS 269-2, Moffett Field, CA 94035
LANSKY@PTOLEMY.ARC.NASA.GOV

Abstract

There has been much recent work on the use of *abstraction* to improve planning behavior and cost. Another technique for dealing with the inherently explosive cost of planning is *localization*. This paper compares the relative strengths of localization and abstraction in reducing planning search cost. In particular, localization is shown to subsume abstraction. Localization techniques can model the various methods of abstraction that have been used, but also provide a much more flexible framework, with a broader range of benefits.

1 Introduction

Over the years, several research results have appeared on the use of *abstraction* to guide and improve planning performance [2, 3, 4, 5, 11, 12]. Abstraction techniques restructure a problem and the problem-solving process into a set of "abstraction levels." At the top level of abstraction, the problem is described and solved at the most coarse-grained level of detail. Each successive level is made more concrete than its predecessor by incrementally adding information into the problem description. The use of abstraction can benefit planning if the solution found at an abstract level serves as a good starting point for problem-solving at the next level of detail. Thus, abstraction may be viewed as a heuristic for ordering which pieces of the overall planning problem are solved first, and which later. At least two methods have been used within the planning community for creating levels of abstraction: (1) creation of more *concrete* levels of detail by incrementally decomposing abstract actions into more concrete subactions (*operator abstraction*); and (2) creation of more *abstract* levels by incrementally eliminating required action preconditions (*state abstraction*).

Recent work has also appeared on the use of domain *localization* or *decomposition* to structure a problem description and thereby guide and improve planner performance. In this case, search savings are attained via a "divide and conquer" approach to reasoning. A domain and problem description (its actions, definitions, goals, preconditions, and any other constraints or properties) are divided up into *regions*. Semantically, regions define the precise "scopes of interaction" between domain properties and actions. Each region consists of a subset of the overall set of actions and the various properties and goals that pertain to those actions.[1] The localization structure of a domain is then used to break the planning space into a set of smaller reasoning spaces (each constructing a plan for a particular region) and to determine how these spaces are searched. In [9], a localized search algorithm is described, along with analytical and empirical results that demonstrate how exponential savings in search cost can be achieved.

This paper compares the relative strengths of localization and abstraction as heuristics for reducing planning search cost. In particular, localization is shown to subsume abstraction; localization can model abstraction "levels," but also provides a more flexible framework for domain partitioning, with a broader range of planning benefits. Section 2 begins with a characterization of the planning search space and the relative search benefits achievable via localization and abstraction. Section 3 then provides a description of the localized reasoning frameworks of two planners – GEMPLAN [6, 7, 8, 9] and COLLAGE, a new system that builds upon the ideas in GEMPLAN. Analytical and empirical results that describe the cost savings attainable by utilizing lo-

[1] Problem reduction (translation of a goal into subgoals) may also be used to decompose a planning problem [1]. However, this kind of technique may more properly be viewed as a problem solving method rather than a search reduction technique, though search savings may occur as a result.

calization are also summarized, as well as tradeoffs in its use. Next, Section 4 shows how localization can encode the various commonly used methods of abstraction. Finally, Section 5 concludes with further discussion of the strengths and weaknesses of the two techniques.

2 The Planning Search Space

Consider a search space in which each node is associated with a plan and each arc is associated with a plan-construction operation that transforms a plan into a new plan (typically via the addition of actions, relations, or variable bindings). Such a tree directly reflects plan-space search, but can also be mapped onto state-space search. In the latter case, the plan P associated with a node is mapped onto the "state" reached after executing P, and each arc operation is mapped onto the action (i.e., the reasoning the planner must perform in order to add that action) that takes it from one state into the next state. Given this characterization of planning search, we can see that the cost of both state-based or plan-based search can be improved in at least three ways:

1. *Lowering operation cost* – i.e., reducing the cost of each arc or plan-construction operation. Since most planning algorithms are NP-complete in the size of the plan, reducing plan size is one way of lowering operation cost.

2. *Operation ordering* – i.e., choosing a good order in which to apply plan-construction operations. Goal-ordering is one example of this, as are other heuristics for determining how a planning space is searched. A good operation ordering may result in less backtracking, but may also improve solution quality.

3. *Reducing implicit search space size*, typically by lowering the branching factor of the search space. Of course, decreasing necessary backtracking via operation ordering may reduce how much of a space is actually searched. But limiting the applicable operations at each node absolutely reduces the total size of the space.

Both localization and abstraction may be viewed as problem-solving heuristics for reducing planning search cost. Alternatively, they may be viewed as ways of reformulating or recasting a planning problem so that the cost of search required to solve that problem is reduced. Abstraction techniques explicitly break the problem-solving process up into "abstraction levels." At each level, more information is added into the problem definition (e.g., actions are decomposed or preconditions are added) to create a more complex planning problem. Since abstraction levels inherently control the order in which pieces of the problem are tackled, it is a heuristic for *operation ordering*. In earlier stages of the reasoning process, only "higher" level operations, which involve high-level actions or conditions, are applied. This set is expanded as the problem and domain definition is expanded. Although abstraction also initially limits the set of applicable operations at each search node, an inherent reduction of applicable operations is not a guarantee of the abstraction technique once the domain is fully expanded. Rather, it is the job of abstraction-derivation techniques to form abstraction hierarchies that guarantee properties like *monotonicity* [4], which limit interaction between the actions and states of the various abstraction levels.

Rather than dividing a problem definition into abstraction levels, localization divides a problem according to the inherent scope of its actions, properties, and goals. A particular localization or domain decomposition provides a planner with a semantic definition of the scope of all domain actions and properties. Each region may be viewed as a "scope" of reference, with an associated set of actions, definitions, goals, etc. As a result, a localization can be used to determine which domain actions and properties interact, and which are independent. Localization then forms a valid basis for partitioning the planning search space into a set of smaller spaces (one for each region), for focusing the application of plan-construction operations to specific pieces of the plan, and for triggering those operations at appropriate times.

Moreover, unlike abstraction, localization can be used to encapsulate domain information based on any criterion, not just "abstractness." The region divisions are based on the particular qualities and scopes of the domain rather than a particular "abstraction-inducing" technique such as operator or state abstraction. Thus, abstraction-based localizations might be used, but also physically-based, process-based, or temporally-based partitionings, which may be more compelling.

Finally, and perhaps most importantly, localization allows for domain regions that overlap and interact. While it is often difficult to attain a clean partitioning into abstraction levels (often resulting in a collapse of levels or a great deal of interaction between levels), localization embraces the notion that real-world decompositions cannot be neatly decomposed and will naturally entail regional overlap. Thus, the localization technique explicitly provides methods for coping with regional interaction.

In terms of the potential search benefits described above, localization can achieve all three:

1. Plan-construction operations are applied to much smaller *regional* plans. Thus, localization *reduces operation cost.* While abstraction may provide a way of initially working on smaller plans at higher levels of abstraction, ultimately, the scope of the planning algorithms becomes the most detailed plan. Thus, abstraction does little to partition the scope of reasoning and does not inherently improve operation cost.

2. The localized search technique directs search flow so that only "relevant" operations are applied at each point in the reasoning process (i.e., those operations relevant to regions whose plans have been modified). Thus, localization controls *operation ordering.*

3. Since only region operations are applicable at each region search node, localization reduces search space size by *limiting the branching factor at each search node.*

3 Localized Representation and Reasoning

We now explain the localization technique by describing its instantiation in two localized planners, GEMPLAN and COLLAGE. In both systems, a region R is defined by a *region description*:

$$<actions(R), definitions(R), constraints(R), subregions(R)>$$

Each region R is associated with a search tree, $tree(R)$, whose role is to construct a plan, $plan(R)$, that satisfies all regional constraints, given available actions and definitions. Each plan is a partially ordered set of actions. The set $actions(R)$ defines a set of *action types* which are considered to belong directly to R and instances of which may occur within $plan(R)$. (Note that $plan(R)$ may also include actions belonging to subregions of R.) The set $definitions(R)$ includes any definitions pertaining to activity in $plan(R)$. The set $constraints(R)$ includes "constraints" that must be satisfied by $plan(R)$. Finally, $subregions(R)$ consists of regions belonging to R.[2]

The regions comprising a domain may take on any structural configuration – they may be disjoint, form hierarchies, or even overlap. Semantically, a particular decomposition defines the *scope* of domain properties; the scope of each definition and constraint associated with R is $plan(R)$ – which may be composed only of actions in R and its subregions. It is the role of the domain describer to ensure that these scoping semantics are correct; the planner assumes that they are. The only required criterion for domain decomposition is that each constraint and definition belong to a region that includes *at least* the entire "scope of applicability" of that definition or constraint (but possibly more).

As an example, consider the small construction domain depicted in Figure 1. It has been partitioned into regions that include the activities of an electrician and plumber. These regions include subregions that contain activities at specific walls. Each *wall* region would be associated with definitions and constraints that are relevant only to the actions that can take place at that wall. In contrast, *electrician* (or *plumber*) definitions pertain to *all* activity directly within *electrician* (*plumber*), as well as all activity at its wall subregions. Since *wallA* is shared by the electrician and plumber, both the electrical and plumbing constraints apply to the activity within *wallA*. The constraints directly associated with *wallA* itself would probably include those relating to coordination of the plumber and electrician activities at that wall. The figure also shows search trees for these regions. Each tree is concerned with building a plan for its region that satisfies all regional constraints. The planning process may thus be viewed as a set of "mini-planners," tied together by the structural relationships between regions.

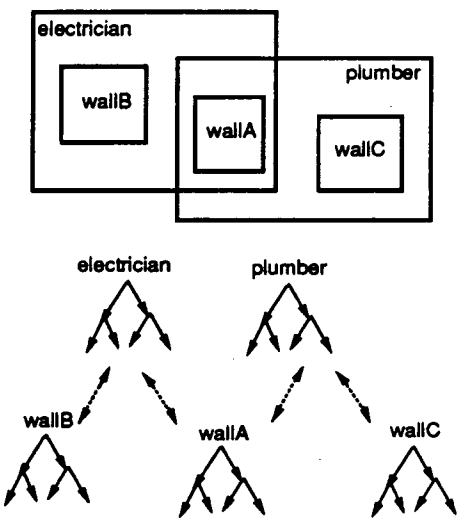


Figure 1: A Localized Construction Domain

3.1 Localizing Traditional Planning Representations

One important distinction between the planning representation of GEMPLAN and COLLAGE and that of traditional planners is the encoding of domain information in terms of "actions," "definitions," and "constraints" rather than STRIPS-like

[2] Section 3.1 describes the relationship between "actions, definitions, and constraints" and more traditional planning representations.

operator descriptions. One reason for this is that it allows domain information to be more easily localized. In a traditional planning representation language, an "action description" is bound up with action preconditions and effects. The "definition" of a particular state predicate is essentially a side-effect of the set of action descriptions within the domain and is thus "distributed" throughout the domain description. Whether or not a literal P is true at some point in the plan is determinable by examining the actions within the plan, along with their defined preconditions and effects, and seeing whether they "combine" to achieve P. The goals that a traditional planner attempts to achieve are a combination of user-provided top-level goals and subgoals that are posted to fulfill action preconditions.

In contrast, the GEMPLAN/COLLAGE framework separates the definition of actions from their preconditions and effects. *Action-type definitions* are simply descriptions of the action types themselves – an action "name" with a set of parameters. For instance, in a blocks world domain, *pick(block)* would define an action type, an instance of which is *pick(a)*. A state predicate is defined separately by an explicit *predicate definition*. In the GEMPLAN implementation of the blocks world, the following definition of *clear(B)* is used:[3]

```
strips_definition(clear(B),
                  [adder(pick(Y),on(Y,B)),
                   adder(put(B,_),true),
                   deleter(put(_,B),true),
                   deleter(pick(B),true)]).
```

A predicate definition includes a list of conditional adder and deleter descriptions. The first parameter of "adder" or "deleter" is an action type which adds or deletes the predicate, under the condition in the second parameter. For example, an action of type *pick(Y)* adds *clear(B)* if *on(Y, B)* is necessarily true just before it, *put(B, _)* always adds *clear(B)*, and *put(_, B)* or *pick(B)* always delete *clear(B)*. Separating predicate definitions from actions descriptions allows actions and predicates to be individually localized (see Section 4). It also makes conditional effects easy to describe; for example, that an action adds a particular literal P in some contexts and another literal Q in others.

In the GEMPLAN/COLLAGE framework, action preconditions and top-level goals are also described as separate entities – they are explicitly defined as *constraints*. For example, in the blocks world domain description, we have:

```
constraint(precondition(pick(B),clear(B))
constraint(precondition(put(X,B),clear(B))
```

Such constraints can be easily localized. Also, note how the separation of precondition constraints from predicate definitions clearly distinguishes between necessary action preconditions and those conditions utilized only for describing conditional effects.

Given a framework of actions, definitions, and constraints, planning may be viewed as "constraint satisfaction" rather than backwards and/or forwards chaining on state-based goals and conditions. In GEMPLAN and COLLAGE, a "constraint" is simply any property that the planner knows how to test and make true. The standard STRIPS-based algorithms form only a subset of the possible methods of plan construction in GEMPLAN and COLLAGE – many other kinds of constraint forms and satisfaction algorithms are provided by the two systems. Any of these constraint forms may be used to encode domain properties, and all constraints are appropriately scoped by the localization structure of a domain. Thus, in many ways, both systems may be viewed as general constraint-based reasoners rather than strictly as planners.

3.2 Localized Search

Once a domain has been localized, its regional structure guides how localized search is performed. As described earlier, each *tree(R)* is concerned with constructing a *plan(R)* that satisfies *constraints(R)* given *actions(R)*, *definitions(R)*, and the actions and definitions of all subregions (and subsubregions, etc.) of R. Each tree node is associated with the region plan constructed up to that point in the search, and each tree arc is associated with a plan modification that transforms a region plan into a new region plan. Upon reaching a node, the planner must choose which region constraint to check next. (Thus, an implicit branching factor in the search space is the set of all region constraints at each node.) If the chosen constraint is not satisfied by the plan at that node, constraint satisfaction algorithms must be applied, resulting in a set of new region plans at the next level down in the tree. A constraint satisfaction algorithm typically adds new actions, relations, and variable bindings to a region plan, and may also generate new subregions. For example, in order to satisfy a precondition constraint, one option is to add an action and appropriate relations that establish that action as an "adder" of the precondition.

Because it is partitioned into regional search trees, localized search is more complicated than the traditional global search utilized by most planners. The localized search algorithm described in [9] has two basic functions: (1) *global correctness*: making sure

[3] Capitalized tokens (or the character "_") represent variables. Lowercase is used for constants. Thus, notation of the form *pick(X)* or *pick(_)* is used to denote any *pick* action with a single parameter.

that all constraints that need to be checked are checked and that appropriate shifts occur between between regional search spaces; and (2) *global consistency*: making sure that all of the plan fragments being constructed (especially those shared by more than one super-region plan) are consistent with each other. This second function is much like that of a distributed database and is ensured by updating all relevant plans for ancestor regions of R, each time search exits from $tree(R)$. Global correctness is ensured, first, by making sure that all regions are searched at least once, and second, by making sure that search eventually occurs for a region R whenever R's plan has been affected by some previous plan modification. GEMPLAN uses a fixed strategy for controlling search flow and consistency maintenance, but COLLAGE allows for more flexible approaches.

In some senses localized search control may be viewed as a TMS-like strategy for maintaining constraint satisfaction – only "affected" constraints need to be rechecked. Unlike a true TMS, however (which also tries to capture "what affects what"), domain localization is a broad-brush heuristic strategy that need not be accompanied by perpetual and expensive reasoning to update those dependencies. The domain decomposition provides a "cut" at defining scope and interactions; the planner uses it, but never needs to verify it or update it. In this respect, localization provides the same level of heuristic information as abstraction, providing a "useful" partitioning of domain information. However, localization can encapsulate information based on many, perhaps mixed, criteria. Some regions may capture physical structures, others may represent processes, and others may represent abstraction hierarchies within these or overlaid with these. For instance, the construction domain of Figure 1 includes regions that are physically-based (the walls) as well as those that represent contractor "processes." One might view localization as having the ability to capture both "horizontal" as well as "vertical" decomposition.

3.3 Localized Search Benefits and Tradeoffs

In [9] a detailed complexity analysis is provided that highlights the potential benefits and tradeoffs of localized search. That paper also provides some initial empirical results that support this theoretical analysis. This section summarizes these benefits and tradeoffs.

Since the cost of localized search for a particular domain is very dependent on the particular constraints, structure, and problem specification for that domain, the "general" complexity analysis described in [9] was performed on a somewhat idealized domain scenario. The search cost of a global, non-localized domain was compared with that of the same domain, partitioned into a set of m subregions each of which overlaps by some factor k with another region g. An original set of n_c constraints was partitioned among these $m+1$ regions. Table 1 summarizes provides the results of this analysis. Complexity results were calculated assuming that all constraints were either constant, linear, quadratic, or exponential in cost relative to the size of the plan. The table also compares the cost of best-case or worst-case search. Best-case measures the cost of one path through the search space (no backtracking), and worst-case measures the cost of the entire potential space. The term s is the size of the final plan. The term n_f is the number of potential fixes for each constraint. Finally, C is the cost of maintaining consistency, which is assumed to be $O(m^2k)$.

These results show that localized search is nearly always better than non-localized search – in most cases much better. The only exceptions are constant-complexity best-case search (when there is no reduction in the amount of the space searched *nor* in constraint algorithm cost) or when the cost of consistency maintenance overshadows the cost of the search. The amount by which localized search wins over non-localized search is proportional to m (the amount of decomposition), but inversely proportional to mk (the amount of overlap). Thus, increased decomposition is always worthwhile, except for the cost of increased overlap. The gains of localized search become exponential as the complexity of the constraint algorithms increases and the amount of the space actually searched increases. These gains come from three sources, which correspond directly to the three factors described in Section 2:

1. The *cost* of each arc – i.e., *operation cost.* Even if the absolute size of the non-localized and localized search spaces are the same, expensive constraint algorithms are applied to much smaller plans in the localized case.

2. The *search heuristics* provided by localization – i.e., *operation ordering.* Because of the semantic information provided by a localized domain description, the most relevant constraints tend to be applied at the right time, enabling a reduction in the amount of the search space that actually needs to be searched.

3. The *size* of the search space – i.e., *branching factor reduction.* This is because only regional constraints are relevant at each node.

complexity of c(i) and f(i)	*Non-Localized (best-case)*	*Localized (best-case)*	*Non-Localized (worst-case)*	*Localized (worst-case)*
constant (b)	$2bs$	$2b(s+mk)+C$	$b(n_c n_f)^s$	$b(m(\frac{n_c n_f}{m+1})^{\frac{s}{m}} + (\frac{n_c n_f}{m+1})^{mk}) + C$
linear (ib)	bs^2	$b(\frac{s^2}{m} + (mk)^2) + C$	$bs(n_c n_f)^s$	$b(s(\frac{n_c n_f}{m+1})^{\frac{s}{m}} + (\frac{n_c n_f}{m+1})^{mk}) + C$
quadratic (i^2)	$\frac{2}{3}s^3$	$\frac{2}{3}(\frac{s^3}{m^2} + (mk)^3) + C$	$s^2(n_c n_f)^s$	$\frac{s^2}{m}(\frac{n_c n_f}{m+1})^{\frac{s}{m}} + (mk)^2(\frac{n_c n_f}{m+1})^{mk} + C$
exponential (b^i)	$2b^s$	$2(mb^{\frac{s}{m}} + b^{mk}) + C$	$(bn_c n_f)^s$	$m(\frac{bn_c n_f}{m+1})^{\frac{s}{m}} + (\frac{bn_c n_f}{m+1})^{mk} + C$

Table 1: Complexity Results

Empirical tests were also carried out which bolster these results. In [9], several decompositions of a building-construction domain were compared, as well as the effects of varying the size of the actual building plan constructed. In this domain, constraint cost was fairly low (close to linear for most constraints) and there was no backtracking. Even so, the search cost of the best localization was less than 50% of the non-localized domain configuration. The results also show that increased localization provides increased benefit, except for the added expense due to increased regional overlap. However, as also predicted by the complexity results, the detrimental effects of increased overlap become overshadowed as plan size and search space size increases.

One of the focuses of the COLLAGE project is to flesh out our understanding of localized search by performing many more controlled experiments. The new COLLAGE search control architecture features a constraint-activation and consistency-activation agenda mechanism that allows for various aspects of the search strategy to be easily modified and reconfigured. Using this architecture, we plan to test a suite of search strategies over a suite of problem types that vary in the amount of backtracking required, constraint algorithm difficulty, as well as domain localization structure and problem size. Finally, we also hope to come up with a *localization learning* approach that automatically discovers domain-dependent and domain-independent localization heuristics.

4 Modeling Abstraction With Localization

In order to model traditional planning-based abstraction methods in a localized framework, we must create "levels" of reasoning, representing incrementally more detailed descriptions of the domain. Referring to the characterization of a region description in Section 3, we can see that this can be achieved by incrementally adding regions and/or subregion links, constraints, action types, and definitions. The addition of this information can be done by a special search step that introduces the next "level" of reasoning. Both GEMPLAN and COLLAGE already incrementally add regions during planning, and both systems access relevant domain information in such a way that makes incremental addition of other types of information trivial to accommodate.[4] In addition, the incremental addition of subregion containment relationships adds an interesting "twist" to the types of abstraction levels attainable; a region may be initially visible to some super-regions and then incrementally become a subregion of additional regions, resulting in "mix-and-match" levels of abstraction.

4.1 Operator Abstraction

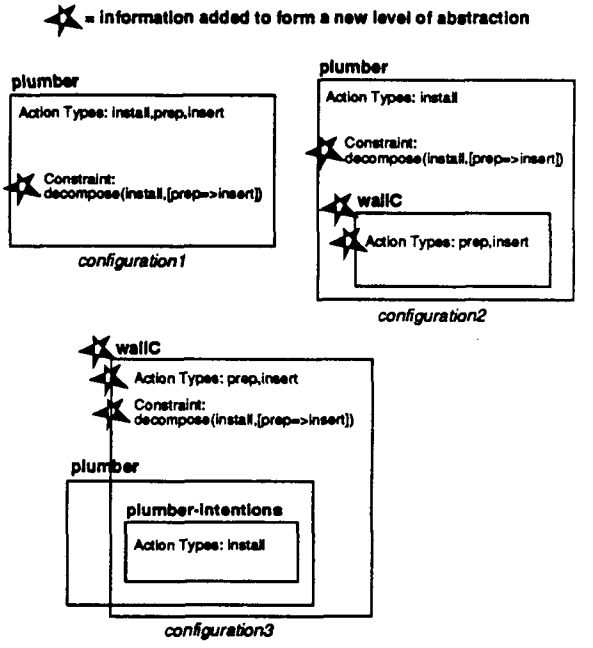


Figure 2: Operator Abstraction

One of the constraint forms available in GEMPLAN and COLLAGE is the *decompose* constraint, which requires that actions of a specified type be [conditionally] decomposed into one of a set of possible patterns of interrelated subactions. Operator abstraction can be modeled in a localized framework by incrementally adding such action de-

[4] All domain information accessed by the plan-construction operations is represented and accessed in plan-relative fashion. As a result, new constraints, actions, regions, and definitions can be "added" to a plan and thus become newly accessible to the reasoning mechanism.

composition constraints and, optionally, incrementally adding regions which contain the subactions at the "next level down." Different degrees of interaction between "levels" may be achieved, depending on the localization configuration used.

Figure 2 provides three sample configurations for modeling operator abstraction in the construction domain of Figure 1.[5] In all three cases, an *install* action of the plumber is decomposed into two subactions at a lower level of detail, *prep* and *insert*. In `configuration1`, no wall subregion exists. Instead, operator abstraction is achieved by simply adding a decompose constraint to *plumber*. In `configuration2`, the subaction types *prep* and *insert* and a subregion *wallC* containing them are also added, thus creating a new level that includes new action types and a new subregion. In `configuration3`, region *wallC* contains the decomposition constraint and subactions, but overlaps with *plumber* only at the point of the higher level action *install*. Note how, in `configuration2`, the lower-level actions in *wallC* become subject to *plumber*'s constraints, introducing potential interaction between "abstraction levels." In `configuration3`, this interaction does not exist except at region *plumber-intentions*. If only region *plumber* adds *install* actions, no planning interaction will occur once region *wallC* is added (i.e., there will be no need to recheck the constraints in *plumber*), thereby guaranteeing monotonicity in the reasoning process. For more discussion of monotonicity and related properties, see Section 5. Also note how, in general, constraints may refer to actions at mixed levels of detail. Unlike many hierarchical planners, GEMPLAN and COLLAGE allow both actions and their subactions to be present within a plan simultaneously.

4.2 State Abstraction

A localized framework can also model state abstraction in several ways, depending on the desired effect. In Figure 3, three possible configurations are given in which various preconditions and definitions affecting the *install* action and its subactions are incrementally added. `Configuration1` illustrates how action preconditions (or top-level goals) can be added on a per-action basis, by simply incrementally adding precondition (or goal) constraints. If we wished a specific predicate to be completely unavailable until a certain "abstraction level" (achieving a "partitioned hierarchy" [4]), is predicate definition and all precondition or goal constraints that utilize that predicate would not be added until that "level"

[5]The constraint syntax used in Figures 2 and 3 has been simplified for illustrative purposes.

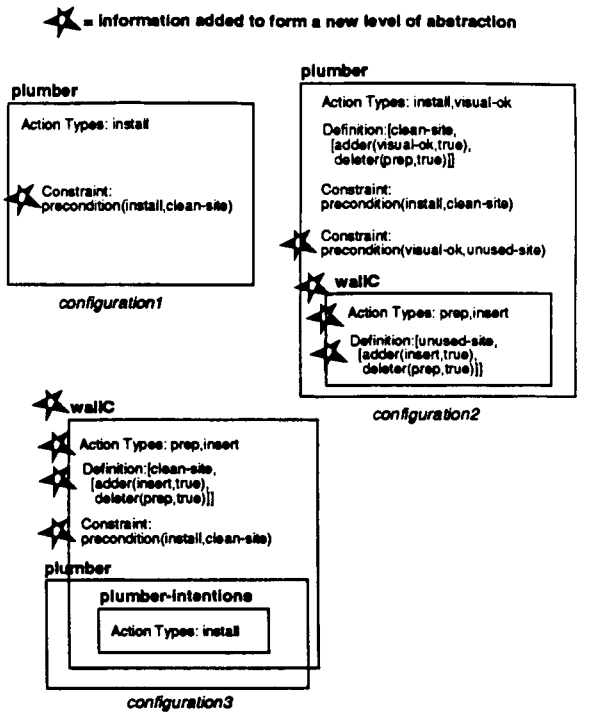


Figure 3: State Abstraction

in the reasoning process is reached. Another option is to incrementally add available "lower level" action types that have been defined to be adders or deleters of a predicate. In this way, action effects (rather than just preconditions) may be incrementally added to the [illegible] of reasoning. In `configuration2`, a literal *clear-site* cannot be "deleted" until the lower-level action type *prep* is added to the domain. This next level also includes a new precondition for *visual-ok* (*unused-site*), as well as a new definition for *unused-site* based on the lower level actions *prep* and *insert*.

One can achieve a strict, noninteracting partition of predicates and actions into levels (i.e. *monotonicity*), by utilizing the strategy depicted in `configuration3`. Here, the new region *wallC* is added which contains a new precondition constraint, actions, and definitions at the next level down. In this case, region *wallC* overlaps with region *plumber* rather than being strictly contained within it. Thus, if we adhere to a regimen in which only region *plumber* adds actions of type *install* (and only region *wallC* can add actions of type *prep* and *insert*), a strict separation of effect would be achieved – changes within region *wallC* would not trigger search within *plumber*, thereby guaranteeing monotonicity.

5 Discussion

The primary point of this paper has been to show that localization is more general than abstraction – it can capture the same kind of heuristic information, but can also express other forms of encapsulation, with potentially greater benefits. This section discusses the impact of localization on such properties as *monotonicity* and tries to shed some light on other plusses and minuses of localization and abstraction.

In [4], several properties are described that provide a useful basis for the formation of abstraction hierarchies. These include the *upward solution property*, *monotonicity*, and *ordered monotonicity*. By constructing abstraction hierarchies in a way that ensures these properties, guarantees can be made about the completeness of an abstracted search space and the amount of backtracking that will be necessary. In particular, the upward solution property guarantees that decomposing the problem into abstraction levels will not remove completeness from the search space. Monotonicity properties additionally remove the need to backtrack into higher levels of the reasoning space.

If localization is used to represent abstraction, what effect does this have on these properties? Does strictly controlling the ordering of constraint application remove possible solutions? Once actions, constraints, and regions are added into the domain specification, will backtracking to a point in the reasoning space *before* this addition be required? These are precisely the kinds of questions that localization addresses. A localization structure captures the defined semantics of interaction between actions and constraints. If two constraints do not apply to the same pieces of the growing plan, they do not interact and their relative constraint ordering does not make a difference. Likewise, if actions, constraints, or regions incrementally added to the planning problem do not cause triggering of previously defined constraints, a pure refinement strategy is possible – no backtracking will be necessary. And even if the regional configuration of a localization does not by itself *guarantee* independence, search heuristics (that encode knowledge about such guarantees) can be used to block unnecessary backtracking or constraint rechecking.

In sum, if localization is used to capture exactly and *only* the forms of abstraction available in the various systems outlined in [4], then localization will manifest the same properties as those systems. Guarantees about such things as monotonicity are a function of the abstractions or localizations chosen for a specific domain. The techniques used by Knoblock [3] to learn abstractions that guarantee ordered monotonicity, or those used by Christensen [2], could also be used within a localization framework.

But an advantage of using a localized framework is that it can be used to capture *much more* than abstraction. Depending on the domain, physically- or process-based localizations might reap even greater search benefits than abstraction-based localizations. Even though "levels" of reasoning can be modelled, they form only a small portion of the structuring capabilities of localization. While properties such as ordered monotonicity may be useful, they come at a price. Since monotonicity requires noninteraction between levels, it may result in a collapse of the hierarchy. Indeed, this might be fairly common, since real-world problems rarely lend themselves to pure refinement strategies. In a localized framework, there is no need to collapse levels or regions into each other if they are not strictly independent. A localization need not be organized hierarchically and does not necessarily have to engender separate planning "phases." Interactions are handled as a basic mechanism of the search process which directs the flow of reasoning, without necessarily invoking backtracking into a "previous level" of reasoning. Finally, in a localized framework, actions, definitions, constraints, and regions may be incrementally added in flexible ways. "Levels of detail" may be mixed among constraints. The addition of subregion relationships can incrementally and selectively increase the scope of constraints.

Of course, just as for abstraction, the trick is to find a *good* localization that reaps as many search benefits as possible. As discussed earlier, a research focus in COLLAGE is automatically learning such localizations. The key is to find a decomposition that balances decomposition and interaction. Increased decomposition results in finer-tuned localization of constraints, but also results in increased regional overlap and accompanying increases in consistency maintenance costs and potential "thrashing" between regional search spaces. The tradeoff between locality and overlap mirrors the abstraction tradeoff between increasing the number of abstraction levels and increasing the amount of interaction between levels.

Admittedly, the cost of dealing with regional overlap and the complexity of localized search is a limitation of the localization technique. Because abstraction simply partitions the search into ever-growing levels of detail, it can still use global search methods. The management of regional search in a localized framework requires more work. Other problem reformulation techniques may also be more feasible in an abstraction-based framework, where the problem may be "reformulated" before search proceeds at each level. However, this might also be accomplished

in a localized framework via incremental *modification* of domain constraints, actions, and definitions.

Finally, localization is also applicable to other kinds of tasks. Localization can be used to encapsulate any kind of domain information – not just STRIPS preconditions, goals, and action decomposition. The technique can be used by any kind of reasoning that can be cast in terms of constraints applied to a partitionable frame of reasoning. Methods based on localized search have already been incorporated into a scheduler [13], another planner that uses abduction as the primary plan-construction mechanism [10], and an image understanding framework [14]. Localization can also aid replanning and plan reuse. If certain pieces of a plan become faulty during run-time, localization provides a good first-cut at which pieces of the plan can be reused and which constraints must be rechecked. Localization-based replanning and reuse is another focus of COLLAGE.

Acknowledgments

Thanks to Steve Minton, Rich Keller, Andrew Philpot, Peter Friedland, John Allen, Smadar Kedar, Mark Drummond, and the referees who have reviewed this paper for their useful comments and encouragement.

References

[1] Bresina, J., Marsella, S. and C. Schmidt. "Predicting Subproblem Interactions," Technical Report LCSR-TR-92, LCSR, Rutgers University (February 1987).

[2] Christensen, J. "A Hierarchical Planner that Generates Its Own Hierarchies," in *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI90)*, Boston, Massachusetts, pp. 1004–1009 (1990).

[3] Knoblock, C.A. "Learning Abstraction Hierarchies for Problem Solving," in *Seventh International Workshop on Machine Learning*, pp. 923-928 (1990).

[4] Knoblock, C.A., Tenenberg, J.D., and Q. Yang. "A Spectrum of Abstraction Hierarchies for Planning," *Proceedings of the 1990 Workshop on Automatic Generation of Approximations and Abstractions*, Boston, Massachusetts, pp.24–35 (1990).

[5] Korf, R.E. "Planning as Search: A Quantitative Approach," *Artificial Intelligence (33,1)*, pp. 65-88 (1987).

[6] Lansky, A.L. "A Representation of Parallel Activity Based on Events, Structure, and Causality," in *Reasoning About Actions and Plans*, M. Georgeff and A. Lansky (editors), Morgan Kaufmann, pp. 123-160 (1987).

[7] Lansky, A.L. "Localized Event-Based Reasoning for Multiagent Domains," *Computational Intelligence Journal, Special Issue on Planning (4,4)* (1988).

[8] Lansky, A.L. "Localized Representation and Planning," in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (editors), Morgan Kaufmann (1990).

[9] Lansky, A.L. "Localized Search for Multiagent Domains," *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, Sydney, Australia, pp. 252–258 (1991).

[10] Missiaen, L. "Localized Abductive Planning for Robot Assembly," in *Proceedings of the 1991 IEEE Conference on Robotics and Automation*, pp. 605–610 (April 1991).

[11] Sacerdoti, E. "Planning in a Hieararchy of Abstraction Spaces," *Artificial Intelligence, 5*, pp. 115–135 (1974).

[12] Wilkins, D.E. *Practical Planning*, Morgan Kaufmann Publishers (1988).

[13] Personal communication with M. Zweben about the NASA Ames AI Research Branch scheduling project.

[14] Personal communication with Framentec Applied Artificial Intelligence Group, Cedex 16, 92084 Paris La Defense, France.

Irrelevance in Problem Solving

Alon Y. Levy

Knowledge Systems Laboratory
Stanford University
701 Welch Road, Bldg. C, Palo Alto, CA 94304
alevy@cs.stanford.edu

Abstract

The notion of irrelevance underlies many different works in AI, such as detecting redundant facts, creating abstraction hierarchies and reformulation and modeling physical devices. However, in order to design problem solvers that exploit the notion of irrelevance, either by automatically detecting irrelevance or by being given knowledge about irrelevance, a formal treatment of the notion is required.

In this paper we present a general framework for analyzing irrelevance. We discuss several properties of irrelevance and show how they vary in a space of definitions outlined by the framework. We show how irrelevance claims can be used to justify the creation of abstractions thereby suggesting a new view on the work on abstraction.

Introduction

Meta-level reasoning has received a lot of attention from researchers in artificial intelligence as a means of guiding problem solvers in their search for solutions [Hayes, 1973; Genesereth, 1988; Smith and Genesereth, 1985; Clancey, 1983]. A common of meta-level strategy is to avoid using knowledge that is irrelevant to the goal at hand. In fact, the notion of irrelevance has been a common theme in many research works, but its formal analysis has received attention only from few researchers such as Subramanian and Genesereth [Subramanian and Genesereth, 1987; Subramanian, 1989]. The ability to give a problem solver advice about what parts of a knowledge base are irrelevant to a specific problem solving goal is a powerful method to reduce its search. For example, consider a domain in which we are trying to find routes between cities in the country, using flights, trains and busses. For some goals, we might want to advise the problem solver that rules and facts about flights are irrelevant, either because the minimal price of flights is known to be greater than is required for the specific goal or because we know that flights will not yield an optimal solution. By giving this advice, we significantly reduce the size of the search space explored by the problem solver.

The notion of irrelevance also plays a key role in work on abstractions and change of representation. Intuitively, when we want to create a simpler or abstract representation we remove some irrelevant detail. If the removed detail was indeed irrelevant then the solution to the problem in the abstract theory will map back to a solution in the original theory. Therefore, if we can provide the system with knowledge about irrelevance or relative irrelevance of knowledge, the system can exploit it to automatically create abstractions. Methods for mechanically detecting relevance can be used to automatically create abstractions.

However, both in order for a user to be able to state such claims to a system in a principled manner and for the system to make proper use of given claims, a better analysis of the notion of irrelevance in problem solving is required. This paper describes a general framework for analyzing the notion of irrelevance. We define a space of possible definitions of irrelevance by identifying several axes along which irrelevance claims differ. Several important properties of irrelevance concerning their usage in problem-solving are outlined and we show how varying the definition of irrelevance in our space affects the satisfaction of these properties. Next, we discuss how irrelevance claims can serve as justifications for creating an abstraction. The case of irrelevance of a distinction between properties (represented as predicates) is examined in detail and we show how such a claim serves as a justification for predicate abstraction [Plaisted, 1981; Tenenberg, 1987].

This framework makes several contributions. First, it clarifies the issues involved in the notion of irrelevance therefore enabling us to better exploit the notion in works that rely on it, such as the work on detecting redundant facts or creating abstraction hierarchies. The properties of irrelevance that we outline provide guidance in building a system that incorporates such claims. Giving precise definitions of irrelevance formalizes the problem of automatically deducing irrelevance facts, thereby enabling us to automatically cre-

ate abstractions, based on deduced irrelevance claims. Moreover, since our framework provides a language to express knowledge about irrelevance, we can use this language to express knowledge about the domain that can help reduce the size of the search or justify creating an abstraction.

Preliminaries

Assume our theory of the domain is represented by a knowledge base of first order predicate calculus formulas, Δ. A problem solving goal (or query) is represented by a formula ψ. The goal is to find whether ψ is implied by Δ (or if ψ has free variables, we want to know which assignments to the variables result in a formula that is entailed from Δ.). Our aim is to identify facts that are *irrelevant* to ψ in order to reduce the search space generated for ψ. Formalizing the concept of irrelevance can be done in several levels. For example, one can formalize irrelevance in terms of the models of Δ and ψ, i.e., a semantic level analysis. Irrelevance can also be analyzed in terms of the facts in the theory Δ, a so called *meta-theoretic* analysis [Subramanian, 1989]. Alternatively, one can give a proof-theoretic analysis of irrelevance, in terms of the actual set of derivations the problem solver can explore in the search to solve ψ. Although these levels are by no means independent, it is important to distinguish between them when defining irrelevance or comparing between definitions.

The goal of this paper is to define notions of irrelevance that enable us to optimize actual problem solving. Therefore, we analyze irrelevance from the system's view of the problem-solving process which is a proof-theoretic one. The system does not actually see the world as the user sees it nor does it see the conceptualization of the world. Instead, it sees the set of symbols used to describe the domain and the set of derivations it can generate.

Example 1: Suppose we are using a resolution theorem prover on a knowledge base in clause form[1]. Consider the following two theories:

$T_1 = \{f \Rightarrow g, \neg f \Rightarrow g\}$

$T_2 = \{g\}$.

T_1 and T_2 are satisfied by the same set of models. In each the value assigned to f does not affect the value of g, and therefore we might consider f to be irrelevant to g. However, in T_1, the theorem prover will have to resolve on the symbol f to derive g, and therefore as far as it is concerned, it can't ignore the symbol f. ∎

Note that we are not claiming that irrelevance relations in the domain are not useful to control problem solving; quite the contrary. Most irrelevance facts are based on properties of the domain. However, a relevance relation in the domain will only be useful if it is reflected in the representation.

In particular, for a problem solver to exploit irrelevance claims, the following properties of irrelevance claims will be of interest. Assume $IR(\phi, \psi, \Delta)$ denotes that the fact (or set of facts) ϕ is irrelevant to the goal ψ with respect to the theory Δ.

- What can the problem solver do given the irrelevance claim? Can it ignore a fact that is deemed irrelevant? Can it ignore any fact that contains it as a subexpression?
- Do irrelevance claims add up? If $IR(\phi_1, \psi, \Delta)$ and $IR(\phi_2, \psi, \Delta)$ hold, does that imply that $IR(\{\phi_1, \phi_2\}, \psi, \Delta)$ holds? If so, we can use all the relevance claims that are available to us at a given instant. However, if not, we can only use one at a time, and then we must check that the others still hold in the resulting theory.
- Is irrelevance a monotonic property? I.e., if we add more facts to the knowledge base, can irrelevant facts become relevant or vice versa?
- Does the irrelevance of a subject imply the irrelevance of a subject which is syntactically related to it? E.g., Does $IR(\phi, \psi, \Delta)$ imply $IR(\neg\phi, \psi, \Delta)$ or $IR(\phi \vee \phi_1, \psi, \Delta)$? Such properties will enable us use a given set of irrelevance claims to deduce additional ones.
- Can irrelevance claims be found automatically by examining the KB?

An important issue in a definition of irrelevance is the *subject* of irrelevance, i.e., the type of entity being deemed irrelevant to the goal. So far we discussed only the irrelevance of a fact (or set of facts) to a problem solving goal, but the subject may be any kind of entity in the representation, such as the objects-constants, predicate-symbols and functions. The irrelevance subject can also be more abstract such as a decision to distinguish between a set of predicates or objects in the representation. The following is an example of the irrelevance of a predicate distinction.

Example 2: Consider the knowledge base with the following facts.

$r_1 : SportsCar(x) \Rightarrow Vehicle(x)$
$r_2 : FamilyCar(x) \Rightarrow Vehicle(x)$
$r_3 : SportsCar(x) \Rightarrow HighRiskInsurance(x)$
$r_4 : FamilyCar(x) \Rightarrow \neg SportsCar(x)$
$r_5 : FamilyCar(Camry)$

In order to solve the query $Vehicle(x)$, the distinction between the relations $SportsCar$ and $FamilyCar$ is irrelevant. Intuitively, all that matters for the proof is that x is some kind of car. Therefore, we can remove the distinction in the representation by *predicate abstraction* [Tenenberg, 1987]. We express the theory using an abstract predicate, Car, as follows:

[1] For clarity, in this document we do not use clause form notation but assume the problem solver gets formulas in clause form.

$s_1 : Car(x) \Rightarrow Vehicle(x)$
$s_2 : Car(Camry)$

r_1 and r_2 were abstracted to s_1, while r_5 was abstracted to s_2. r_3 on the other hand was a rule specific to $SportsCar$, because
$FamilyCar(x) \Rightarrow HighRiskInsurance(x)$
does not hold, and therefore we cannot abstract it to
$Car(x) \Rightarrow HighRiskInsurance(x)$.
Consequently, it is removed from the theory. Similarly, r_4 is a formula that distinguishes between the relations $FamilyCar$ and $SportsCar$ and therefore is removed from a theory that ignores the distinction between these relations. ∎

A final issue that factors into a definition of irrelevance is the space of possible changes of the representations and the theory (or *weakenings* of the theory [Subramanian and Genesereth, 1987]) we are considering in order to remove the irrelevancy. In example 1, we only considered changing the theory by removing facts and therefore we could not justifiably say that f is irrelevant to g. However, had we considered changing the theory by adding some of its logical consequences (e.g., g), we could deem f irrelevant to g. In example 2, the irrelevancy was removed by predicate abstraction, i.e., replacing the predicates $FamilyCar$ and $SportsCar$ by an abstract predicate Car.

A Space of Irrelevancies

To capture the various properties of irrelevance we define a space of possible definitions of irrelevance. The space of definitions revolves around the set of possible derivations of the goal. Let Δ be a knowledge-base, ψ be a goal and $\mathcal{D}$ be the set of derivations of ψ from Δ. A definition of irrelevance of ϕ (which can be any irrelevance subject) to ψ is composed of the following choices:

A1. Defining irrelevance of ϕ with respect to a single derivation, $D \in \mathcal{D}$.

A2. A subset $\mathcal{D}_0$ of $\mathcal{D}$ over which to quantify A1.

A3. The method of quantification over $\mathcal{D}_0$, i.e., existentially or universally.

Formally, if D is a derivation of a goal ψ from a knowledge base, Δ, we denote the choice for A1 by $Ir(\phi, \psi, D)$, i.e., that ϕ is irrelevant to the derivation D of the goal ψ. If Φ is a set of facts, $Ir(\Phi, \psi, D)$ holds if $Ir(\phi_i, \psi, D)$ for all $\phi_i \in \Phi$.

Definition 3: Let $\mathcal{D}_0$ be a set of derivations of a goal ψ from the knowledge base Δ[2]. ϕ is said to be *weakly irrelevant* to ψ with respect to $\mathcal{D}_0$, denoted by $WI(\phi, \psi, \mathcal{D}_0)$[3], if $Ir(\phi, \psi, D)$ holds for some $D \in \mathcal{D}_0$. ϕ is said to be *strongly irrelevant*, denoted by $SI(\phi, \psi, \mathcal{D}_0)$, if $Ir(\phi, \psi, D)$ holds for every $D \in \mathcal{D}_0$. ∎

Note that in Definition 3, the knowledge base, Δ does not appear explicitly in WI (SI), but is implicit in the set $\mathcal{D}_0$. For every choice of Ir and of $\mathcal{D}_0$, we get a definition for weak and strong irrelevance. Except for $\mathcal{D}_0 = \mathcal{D}$, examples of $\mathcal{D}_0$ include the set of all minimal derivations[4], or all derivations bounded by some resource constraints. The following example clarifies some of these distinctions.

Example 4: Consider a knowledge base with the following rules:
$r_1 : E(x) \Rightarrow Q(x)$
$r_2 : R(x) \Rightarrow Q(x)$
$r_3 : P(x) \Rightarrow Q(x)$
$r_4 : E(x) \Rightarrow P(x)$
$r_5 : Q(x) \Rightarrow P(x)$

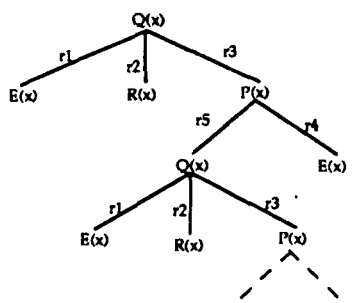


Figure 1: Search space for a goal $Q(x)$

The knowledge base also contains a set of ground facts but only for the predicate E. Figure 1 shows the possible derivations that can be generated for Q from this theory. Suppose we define $Ir(r, g, D)$ to hold if the rule r does not appear in the derivation D. Let $\mathcal{D}$ be the set of all derivations of $Q(a)$[5]. $WI(r_3, Q(a), \mathcal{D})$ holds since whenever $Q(a)$ is derivable, there will be a derivation of $Q(a)$ using only r_1. $SI(r_2, Q(a), \mathcal{D})$ holds because r_2 cannot be part of a proof of $Q(a)$. $SI(r_5, Q(a), \mathcal{D})$ does not hold, however, if we consider the set of non-redundant derivations $\mathcal{D}_0$[6], then $SI(r_5, Q(a), \mathcal{D}_0)$ holds. ∎

Irrelevance of a Fact

In this section we briefly consider the case in which the relevance subject is a single fact, and show how varying the choices for A1-A3 affects the properties of the resulting irrelevance claims. The definitions consider a specific problem solver, hence our discussion assumes we are using resolution theorem prover. A derivation

[2] If ψ is a set of goals (e.g., a goal with free variables) we consider a set of derivations for every element of ψ. The definitions below hold if they hold for every element of ψ.

[3] Note that the knowledge base Δ is implicit in the third argument of WI and SI.

[4] Given some criteria of minimality of deductions.

[5] which will be empty if $E(a)$ is not in the knowledge base.

[6] A derivation tree is redundant if it has two identical nodes n_1 and n_2 such that n_1 is an ancestor of n_2.

is a resolution tree of clauses, where the goal clause (or the empty clause in case of a refutation proof) is the root, and the children of every clause are the two clauses that were resolved in order to get it. The leaves of the tree are clauses from the knowledge base, and they are denoted by $Base(D)$.

Consider three choices for A1. In the first, a fact is irrelevant to a derivation of the goal if it is not one knowledge-base facts used in it:

Definition 5: $Ir_1(\phi, \psi, D)$ if $\phi \notin Base(D)$. ∎

A stronger definition requires that ϕ is irrelevant to a derivation if it appears nowhere in the derivation:

Definition 6: $Ir_2(\phi, \psi, D)$ iff there does not exist a substitution σ such that $\phi\sigma$ is a subclause of a clause in D. ∎

Subramanian [Subramanian, 1989] defines ϕ to be irrelevant to ψ with respect to a theory Δ, if there is a subset of Δ that entails ψ but is non-committal on ϕ. In our space, we can formalize this as follows:

Definition 7: $Ir_3(\phi, \psi, D)$ if $Base(D) \not\models \phi$ and $Base(D) \not\models \neg\phi$. ∎

Using Ir_3, for a refutation resolution theorem prover, $WI(\phi, \psi, \mathcal{D})$ is equivalent to the definition given in [Subramanian, 1989].

Figure summarizes the different properties that hold for the definitions described above. The following show how the properties of weak irrelevance differ from those of strong irrelevance.

Observation 8: Whenever irrelevance adds up on a single derivation, it will add up for strong irrelevance, i.e., if

$$Ir(\Phi_1, \psi, D) \wedge Ir(\Phi_2, \psi, D) \Rightarrow Ir(\{\Phi_1, \Phi_2\}, g, D)$$

hold for any D, then for any choice of $\mathcal{D}_0$,

$$SI(\Phi_1, \psi, \mathcal{D}) \wedge SI(\Phi_2, \psi, \mathcal{D}) \Rightarrow SI(\{\Phi_1, \Phi_2\}, g, \mathcal{D})$$

This property does not hold for weak irrelevance. ∎

Observation 9: The converse holds for weak irrelevance too, i.e., whenever

$$Ir(\{\Phi_1, \Phi_2\}, \psi, D) \Rightarrow Ir(\Phi_1, \psi, D) \wedge Ir(\Phi_2, \psi, D)$$

holds for any D, then for any choice of $\mathcal{D}_0$,

$$WI(\{\Phi_1, \Phi_2\}, \psi, \mathcal{D}_0) \Rightarrow WI(\Phi_1, \psi, \mathcal{D}_0) \wedge WI(\Phi_2, \psi, \mathcal{D}_0)$$

$$SI(\{\Phi_1, \Phi_2\}, \psi, \mathcal{D}_0) \Rightarrow SI(\Phi_1, \psi, \mathcal{D}_0) \wedge SI(\Phi_2, \psi, \mathcal{D}_0)$$

∎

Observation 10: For any definition of Ir such that $Ir(\phi, \psi, D) \Rightarrow Ir_1(\phi, \psi, D)$, if we add facts to the knowledge base, irrelevance can change as follows. A fact that was weakly irrelevant will still be weakly irrelevant. A fact that was strongly irrelevant will be at least weakly irrelevant. A fact that was not weakly irrelevant might become weakly irrelevant. ∎

	Ir_1	Ir_2	Ir_3
P_1	√	√	√
P_2		√	
P_3		√	
P_4	√	√	√
P_5			√
P_6		√	

P_1: $WI(\phi, \psi, \mathcal{D})$ implies that $\Delta \setminus \phi \vdash \psi$.
P_2: $WI(\phi, \psi, \mathcal{D})$ implies that the problem solver can ignore any derivation that contains ψ.
P_3: $WI(\phi, \psi, \mathcal{D})$ implies that the problem solver can ignore any derivation that contains ψ as a subexpression.
P_4: *Adding up* -
$Ir(\Phi_1, \psi, D) \wedge Ir(\Phi_2, \psi, D) \Rightarrow Ir(\{\Phi_1, \Phi_2\}, g, D)$.
P_5: *Transfers through equivalence* -
$Ir(\phi_1, \psi, D) \wedge (\phi_1 \equiv \phi_2) \Rightarrow Ir(\phi_2, \psi, D)$.
P_6: If ϕ is a subclause of ϕ_1, then $Ir(\phi_1, \psi, D) \Rightarrow Ir(\phi_1, \psi, D)$.

Figure 2: Properties of Irrelevance

Deducing Irrelevance Claims

Varying the definition of irrelevance has drastic effects on the ability to automatically derive irrelevance claims. Given a knowledge base Δ and a goal ψ, we would like to derive all (or part of) the facts in Δ that are irrelevant to ψ. In general, looking at the whole knowledge base to determine irrelevance will be more costly than solving the query. A more interesting question is whether irrelevance claims can be derived by looking at only a small and stable part of the knowledge base. For example, in example 4, we were able to determine irrelevance by merely looking at the structure of the proof space created by the rules, regardless of the specific ground facts for the predicate E.

We examine this question for knowledge bases comprised of a set of Horn rules with no function symbols (Datalog, [Ullman, 1989]), and a database of ground atomic facts. We distinguish between two sets of predicates in the knowledge base, the *extensional predicates* (EDB predicates) which are those that appear only in the database and in antecedents of rules, and the *intensional predicates* (IDB predicates) which are the predicates appearing in the consequents of the rules, i.e., the predicates that are being defined by the EDB predicates and the rules. A query is an IDB predicate, i.e., to find all the derivable facts for that predicate. Every derivable instance of the goal has a (perhaps more than one) derivation tree. A *derivation tree* is a tree consisting of *goal-nodes* and *rule-nodes*. A goal

node is labeled by a ground atom, and it has a single child, which is an instantiated rule-node. The head of an instantiated rule-node is identical to its parent goal-node. A rule-node has a child goal-node for each one of its subgoals. The leaves of a derivation tree are goal-nodes labeled by ground atoms from the EDB. A derivation is *not minimal* (or *redundant*) if there are two identical goal-nodes n_1 and n_2, such n_1 is an ancestor of n_2. A rule r is irrelevant to a derivation D (i.e., $Ir(r, \psi, D)$) if none of the rule nodes in D are instances of r (note that this is equivalent to Ir_1 and Ir_2).

The question we address is the following. Given a set of rules, $\mathcal{P}$, a query q and a definition of irrelevance, can we determine whether a rule $r \in \mathcal{P}$ is irrelevant to query for *any* possible set of ground facts in the knowledge base. We consider two choices for A2, the set of all derivations of the goal q, denoted by $\mathcal{D}$, and the set of all minimal derivations, $\mathcal{D}_0$.

Finding irrelevant rules enables us to significantly prune the search space for the query. In example 4, rule r_2 will not appear in any derivation of Q, therefore $SI(r_2, Q(x), \mathcal{D})$ holds. r_5 will appear only in redundant derivations of Q and therefore $SI(r_5, Q(x), \mathcal{D}_0)$ holds. Since $Q(x)$ can always be derived using either r_1 or $\{r_3, r_4\}$, both $WI(r_1, Q(x), \mathcal{D})$ and $WI(\{r_3, r_4\}, Q(x), \mathcal{D})$ hold. Consequently, identifying the various kinds of irrelevance can enable us to compute Q using only r_1. Considering constraint literals in the rules enables us to derive additional irrelevance claims:

Example 11: Consider the following knowledge base:

$s_1 : Q(x,z) \wedge Q_1(z,y) \wedge x < z \Rightarrow P(x,y)$
$s_2 : Q(z,x) \wedge Q_1(x,y) \wedge x < y \Rightarrow P(x,y)$
$s_3 : E_1(x,y) \wedge x < 3 \Rightarrow Q(x,y)$
$s_4 : E_2(x,y) \wedge x > 1 \Rightarrow Q_1(x,y)$

If the query is $P(x,y)$, all rules are relevant. However, if the query is $P(x,y) \wedge (y < 1)$, then s_2 is strongly irrelevant, i.e., $SI(s_2, P(x,y) \wedge (y < 1), \mathcal{D})$. ∎

Finding all rules which are weakly irrelevant, i.e., $WI(r, g, \mathcal{D})$, is precisely the rule redundancy problem shown to be undecidable by Shmueli [Shmueli, 1987]. Consequently, determining $WI(r, g, \mathcal{D}_0)$ is also undecidable. For strong irrelevance, if the rules contain no constraint literals and no object constants, determining $SI(r, g, \mathcal{D}_0)$ is equivalent to the rule reachability problem that has an easy polynomial time solution [Kifer, 1988]. [Levy and Sagiv, 1992] gives an algorithm for detecting $SI(r, g, \mathcal{D}_0)$ and $SI(r, g, \mathcal{D})$ even when constraint literals are present. It also establishes an exponential-time lower bound on the problem of determining $SI(r, g, \mathcal{D}_0)$.

Using Irrelevance to Justify Abstractions

Much of the work in AI on creating abstraction hierarchies relies on the intuition that by creating an abstract theory we are removing some irrelevant detail. If the detail removed is indeed irrelevant, then a solution to the problem in the abstract theory will map back to a solution in the original theory (also referred to as the *ground theory*). Otherwise, we will have to backtrack between abstraction levels. Although this has been the motivation underlying work on abstractions, the formal connection between irrelevance and abstractions has received little attention (e.g., [Subramanian, 1989]). For example we can view *predicate abstraction* as being justified by the irrelevance of a distinction between predicates; object aggregation can be justified by irrelevance of a granularity distinction. Identifying abstraction with the notion of irrelevance offers several advantages:

- We make explicit what is being abstracted (i.e., the subject of irrelevance).
- We make clear the strength of the justification for the abstraction (by the strength of the type of irrelevance claim that holds).
- We formalize the problem of automatically creating abstractions by translating it to the problem of automatically finding irrelevance claims.

In this section we briefly discuss how irrelevance claims that are justifications for abstractions can be formulated in our framework. We identify several *irrelevance subjects* that account for many abstractions discussed in the literature. As a consequence we get an expressive language to state knowledge about the domain that can affect the creation of abstractions. We define a notion of irrelevance that best justifies abstractions and mention several weaker notions.

The first assumption underlying a formalization of irrelevance is that removing irrelevant detail should not enable us to reach new conclusions about the set of goals we are interested in, i.e., any conclusion reached in the abstract theory should be an abstraction of one in the base theory (this is also known as a TD property of abstractions [Giunchiglia and Walsh, 1991] or the downward solution property [Tenenberg, 1987]). The justification for this claim is that by removing irrelevant detail, we are effectively ignoring some of our knowledge, and therefore, we can not come to new conclusions[7]. For example, when we remove some irrelevant detail in a planning problem (e.g., action precondition), if the resulting abstract plan can not be mapped back to a base-level plan, the detail we have removed was not truly irrelevant to the problem[8]. Sec-

[7] As long as the our reasoning has no form of nonmonotonicity.

[8] Note that this does not necessarily mean that the abstraction is not useful!

ond, the abstract theory should not prevent us from solving the goal, i.e., if the original theory had a solution to the goal, then the abstract one should too. Finally, in order for the abstraction to be computationally effective, the solutions that are preserved by the abstraction should be the cheaper ones.

These criteria are naturally formulated in our framework. Recall that in order to define irrelevance of a subject α, we must give a definition for $Ir(\alpha, \psi, D)$, i.e., when the subject α is irrelevant to a derivation D. Given a theory Δ, we denote the abstract theory resulting from removing the irrelevancy α by $f_\alpha(\Delta)$. For example, if α is a distinction between predicates, $f_\alpha(\Delta)$ is the theory resulting from predicate abstraction. The exact form of $f_\alpha(\Delta)$ is discussed in the next section. We base our definition of Ir on a mapping h_α from the derivations of ψ in Δ, denoted by $\mathcal{D}_1$, to the derivations of $f_\alpha(\psi)$ in $f_\alpha(\Delta)$, $\mathcal{D}_2$. The only requirement from h_α is that it is onto $\mathcal{D}_2$. h_α need not be a total mapping on $\mathcal{D}_1$, i.e., there might be derivations of ψ that will not be mapped to the abstract theory, and it need not be 1-1. Other constraints on h_α will yield stronger forms of irrelevance and therefore stronger justifications for the abstraction, (for example, h_α will be called a *simplifying mapping* if for any $D \in \mathcal{D}_1$, the cost of $h(D)$ is no more than the cost of D[9]). Given h_α, $Ir(\alpha, \psi, D)$ is defined as follows:

Definition 12: $Ir(\alpha, \psi, D)$ is true iff $h_\alpha(D)$ is not empty. ∎

Note that in this definition h_α is dependent on α and ψ. Definitions of weak and strong irrelevance are obtained by quantifying the definition of Ir over a chosen set of derivations. The following states that the first two requirements of an abstraction are satisfied by weak irrelevance.

Observation 13: If $\mathcal{D}_0$ is a set of derivations in $\mathcal{D}_1$, and $WI(\alpha, \psi, \mathcal{D}_0)$ holds then ψ is provable from Δ if and only if $f(\psi)$ is provable from $f_\alpha(\Delta)$. ∎

In order satisfy the third requirement, we must impose a restriction on $\mathcal{D}_0$:

Observation 14: If $\mathcal{D}_0$ is a set of derivations that contains all minimal derivations and h_α is a simplifying mapping, then if $SI(\alpha, \psi, \mathcal{D}_0)$ holds, $f_\alpha(\psi)$ will have a solution in the abstract theory if and only if it has one in the original theory, and at least one of abstract-level solutions will cost no more than that cheapest solution in the original theory. ∎

This condition is a sound justification for creating the abstraction. Imposing more constraints on h_α will give us even stronger justifications. For example, we can require that $h_\alpha(D)$ effectively break up D into subproblems of equal size. Knoblock [Knoblock, 1990; Knoblock *et al.*, 1991] shows how this constraint along with other effects the ability to achieve savings when employing hierarchical planning.

Weaker irrelevance claims can also be given to the system. For example, we can state a distinction α_1 is more relevant than a distinction α_2, i.e., whenever α_1 is justifiably abstracted, so is α_2. Another kind of claim is one a probabilistic one, i.e., stating to the system that in most cases α is irrelevant to ψ. The system can then use this claim and succeed in most cases and backtrack in others. By stating irrelevance claims declaratively we can also state under what conditions the relevance claim holds.

In the next section we examine the case of predicate abstractions and show they are justified by irrelevance of a predicate distinction.

Irrelevance of Predicate Distinctions

When designing a representation, a decision has to be made about the detail with which to conceptualize the world. In some cases, identifying a property P (e.g., $Car(x)$) will suffice. In other cases we need to refine P to subclasses $\mathcal{P} = \{P_1, \ldots, P_n\}$ (e.g., $SportsCar(x)$, $FamilyCar(x)$, etc.) For some goals, the finer distinction of properties is irrelevant, and therefore, reasoning will be more efficient if we change the theory by abstracting the distinction. We would like to be able to give the system knowledge about the domain that will guide it in deciding when a predicate distinction is relevant. To define the meaning of such an irrelevance claim in the framework, we first must define the abstract theory resulting from removing the predicate distinction and the mapping of derivations between the original and abstract theories.

The Abstract Theory

Suppose we have a theory Δ, consisting of a set of predicates $\mathcal{P} = \{P_1, \ldots, P_n\}$, and we want to abstract the distinction between them by replacing them by a predicate P that represents their union (e.g., we want to replace $\{FamilyCar, SportsCar\}$ by the predicate Car). Intuitively, to abstract the theory Δ, we replace every occurrence of a predicate in $\mathcal{P}$ in every formula in Δ by P (e.g., abstract $FamilyCar(x) \Rightarrow Vehicle(x)$ by $Car(x) \Rightarrow Vehicle(x)$). However, doing so for every formula in Δ might result in an inconsistent theory or in a theory that will entail conclusions that were not entailed by the original one. In example 2, abstracting rule r_4 will result in a contradiction ($Car(x) \Rightarrow \neg Car(x)$), and abstracting r_3 will result in a fact that is not entailed by the theory (i.e., $Car(x) \Rightarrow HighRiskInsurance(x)$ does not follow from the theory). In order to assure that our derivation mapping will be onto, we need the abstract theory to be consistent with the ground one. Tenenberg [Tenenberg, 1987; Tenenberg, 1988] discusses predicate abstractions and defines the maximal set of formulas that can be included in the abstract theory such that the abstract theory will be consistent with the original one. His

[9] Given some cost model for derivations such as the number of nodes in the proof tree.

definition is based on the interpretation of the abstract predicate, which is the union of the interpretations of the predicates in $\mathcal{P}$. However, as Tenenberg notes, this set is usually infinite even when the ground theory is finite. Therefore, the abstract theory we consider is a finite subset of the one defined by Tenenberg. Our abstract theory consists of the abstractions of the formulas in the base theory that are *independent* of the predicate distinction. Intuitively, a formula is independent if its abstraction is consistent with the theory. In the formal definition, we assume that formulas are represented as clauses. A literal in a clause is negative if it is a negation of an atomic formula (e.g., $\neg P(x)$ is a negative literal, while $P(x,y)$ is a positive literal). $Neg(C)$ ($Pos(C)$) denotes the set of negative (positive) literals in a clause C.

Definition 15 : Independence - Let $\mathcal{P} = P_1, \ldots, P_n$, and suppose $Neg(C)'$ is the result of substituting every occurrence of an element of $\mathcal{P}$ in $Neg(C)$ by some other predicate in $\mathcal{P}$ using a mapping f_1. (Two occurrences of the same predicate need not have the same mapping under f_1.) A clause C is independent of a predicate distinction $\mathcal{P}$ with respect to a ground theory Δ, if for any such f_1 there exists a mapping, f_2 of the occurrences of elements of $\mathcal{P}$ in $Pos(C)$ to elements of $\mathcal{P}$, such that $Pos(C)' = f_2(Pos(C))$ and $\Delta \vdash Pos(C)' \cup Neg(C)'$[10] ∎

Note, that a clause that contains only positive literals from $\mathcal{P}$ will be independent whenever it is provable from the theory. The problem arises with the negative literals. In example 2, all rules but r_3 are independent of the distinction $\{FamilyCar, SportsCar\}$.

Lemma 16: *A clause C is independent of a predicate distinction $\mathcal{P}$, if and only if $f(C)$ would be included in the abstract theory as defined by Tenenberg in [Tenenberg, 1990].*

The Derivation Mapping

Given the abstract theory produced by removing the predicate distinction, we can define the mapping of derivations in the base-theory to those in the abstract one. Recall that we require that the mapping be an onto mapping. Intuitively, given a derivation in the abstract theory, a base-level derivation that is mapped to it should be obtainable by reversing the abstraction function on the formulas in the derivation. However, as the following example shows, this cannot always be done.

Example 17: Consider the following knowledge base:
$r_1 : P_1(x) \Rightarrow Q(x)$
$r_2 : P_2(x) \Rightarrow R(x)$
$r_3 : R(x) \Rightarrow P_1(x)$
$r_4 : P_2(a)$

[10] Notice, that in the definition we use $\vdash$, which assumes a simple case where the base-level reasoner and the meta-level reasoner are the same. However, in general, they need not be the same.

Suppose we want to abstract P_1, P_2 by an abstract predicate P. The resulting abstract theory will be:
$s_1 : P(x) \Rightarrow Q(x)$
$s_2 : R(x) \Rightarrow P(x)$
$s_3 : P(a)$

s_1 is included in the abstract theory because r_1 is independent of the predicate distinction (because $P_2(x) \Rightarrow Q(x)$ is derivable from the theory).

The (single) derivation of $Q(a)$ in the abstract theory cannot be trivially mapped to a base-level derivation. The reason is that it uses s_1 and s_3, and they are abstractions of of r_1 and r_4 which do not yield a base level derivation of $Q(a)$. ∎

The source of the problem is that some reasoning was done in the process of creating the abstract theory. In this case, s_1 already represented a base-level chain of reasoning that derived $P_2(x) \Rightarrow Q(x)$.

Informally, we define the derivation mapping, h_α, by specifying all the base-level derivations that map to a given abstract-level derivation D. The mapping is defined in two steps as follows. Given D, we first construct all the possible mappings in which occurrences of P in D are mapped to elements of $\mathcal{P}$, such that the resulting derivation is a valid one. For example, in Figure 3, the abstract-level derivation (a) has two such possible mappings (b) and (c). Next try to complete each of the resulting derivations such that they will be valid derivation in the base-level theory. In our example, (b) cannot be completed because $P_1(a)$ does not follow from our original theory. (c) however, can be completed, as shown in (d). Any such complete base-level derivation is mapped to D under the mapping h_α. In Figure 3 only (d) is mapped to the abstract level derivation (a) (i.e., $h_\alpha(d) = a$).

In order to show that h_α is onto, we must show that at least one of the intermediate derivations can be completed to a valid derivation from the base-level theory.

We prove this by defining one mapping $\mathcal{M}$, from the occurrences of P in D to $\mathcal{P}$. $\mathcal{M}$ will have the property that when we apply it to D, the resulting derivation is guaranteed to have a completion to a valid base-level derivation. Let $\mathcal{C}$ be the leaves of the abstract level derivation, D that contain the predicate P. We define $\mathcal{M}$ on the occurrences of P in $\mathcal{C}$ such that two literals that are resolved somewhere in D are assigned the same predicate in $\mathcal{P}$. That ensures that $\mathcal{M}$ can be extended to all the occurrences of P in D. For clarity, we assume that P does not appear in the root of D, and that D did not have any non-trivial factoring (see [Genesereth and Nilsson, 1987]). We define a partial order $<$ on the clauses in $\mathcal{C}$, and make assignments to clauses in the topological order induced by $<$.

Definition 18: For every $C_i, C_j \in \mathcal{C}$, $C_i < C_j$ iff an ancestor of C_i is resolved with an ancestor of C_j on a literal in $\mathcal{P}$, and the ancestor of C_i contributes the positive literal to the resolution. ∎

Lemma 19: *The relation $<$ is acyclic.*

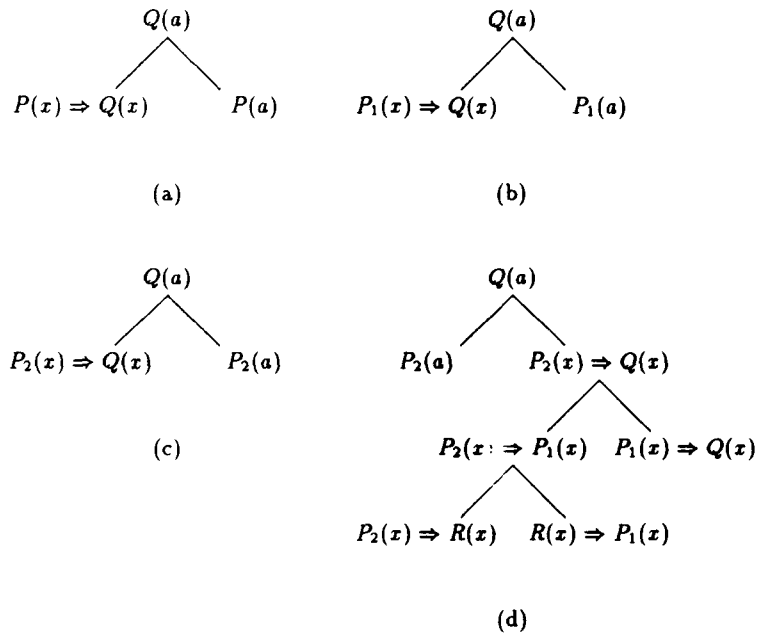


Figure 3: Mapping base-level derivations to abstract-level derivations

Also note that if C_i is minimal in the order $<$, (i.e., there is no C_j such that $C_j < C_i$), then C_i contains only positive appearances of P.

We define $\mathcal{M}$ on the occurrences of P in C_i only after we have defined the mapping for all its occurrences in clauses C_j such that $C_j < C_i$, as follows:

- If C_i contains only positive appearances of P, we map the occurrences of P such that that the resulting clause is entailed from the base-level theory. Note that by the definition of the abstract theory, there must be at least one such mapping for C_i.
- If C_i contains negative literals of P, we do the following. For any negative occurrence of P, the positive literal with which it is resolved in D has been already mapped previously (by the definition of $<$). Hence we map it to the same element of $\mathcal{P}$ to which its counterpart was mapped. As for the positive literals, any assignment for them such that the resulting clause is derivable from the base theory is a valid assignment. The definition of the abstract theory (i.e., all elements of $\mathcal{C}$ are abstractions of independent base-level clauses), guarantees that at least one such assignment exists.

The mapping $\mathcal{M}$ guarantees that every leaf of the tree is either in the knowledge base or is derivable from it. Therefore, the resulting tree can be completed to a full base-level derivation.

Theorem 20: *The derivation mapping h_α is well defined and onto (i.e., every derivation in the abstract theory has at least one derivation in the base theory that maps to it), and is a simplifying abstraction.*

Properties of the Irrelevance Definition

Given the definition of irrelevance, the question arises whether given the original theory and the abstract one, it is possible to decide if the predicate distinction is irrelevant to the goal. The following provides a first step in that direction by identifying a class of derivations that are preserved by the abstraction.

Theorem 21: *If $\mathcal{D}_0$ is a set of derivations of the goal such that for any $D \in \mathcal{D}_0$, all the facts in Base(D) are independent of the predicate distinction $\mathcal{P}$, then $SI(\mathcal{P}, \psi, \mathcal{D}_0)$ holds.*

Observation 22: The converse does not hold. I.e., ψ can have a derivation in the abstract theory, but not have one in the base theory only from independent facts. Example 17 illustrates that.[11] ∎

[11] Note that if we change the definition of independence to require $Pos(C)' \cup Neg(C)' \in \Delta$ instead of $\Delta \vdash Pos(C)' \cup$

From this condition and the algorithms described in [Levy and Sagiv, 1992] we can construct an algorithm for detecting irrelevance of predicate distinctions in the following case:

Corollary 23: *Given a Datalog theory, Δ and $f_{\mathcal{P}}(\Delta)$ which is the abstract theory resulting from removing the distinction between predicates in $\mathcal{P}$, there is an algorithm to determine whether $SI(\mathcal{P}, \psi, \mathcal{D}_0)$ holds for any given set of ground facts, where $\mathcal{D}_0$ is the set of all non-redundant derivations of ψ from Δ.*

Note that creating the abstract theory is in general undecidable because it entails solving the rule redundancy problem. Methods for detecting some classes of redundant rules (e.g., [Sagiv, 1988]) can be used to construct a subset of the theory.

Other Relevance Subjects

The same technique described above can be used to define irrelevance of other kinds of relevance subjects. [Levy, 1992] discusses the following subjects:

- **Object aggregations:** We replace a set of object constants by an aggregate object. E.g., replace the subparts of a component by one object representing the component. For example, in the Missionaries and Cannibals problem [Amarel, 1981], we can replace the sets of missionaries and cannibals by objects denoting their sets.
- **Object distinction:** We replace a set of object constants by a representative object that has only the properties common to all elements of the set (i.e., we replace a set $\mathcal{O} = \{o_1, \ldots, o_n\}$ by an object o, such that $P(o)$ holds iff $P(o_i)$ holds for every $o_i \in \mathcal{O}$. For example, when reasoning about chemical reactions, it is enough to consider only one representative molecule of every type in the chemical formula and that suffices to describe the complete reaction between the substances.
- **Predicate representative:** We replace a set of predicates $\mathcal{P}$ by an abstract predicate that represents their *intersection*.
- **Macro rule:** We replace a set of facts S by a logical consequence, s of S.

Conclusions

We presented a general formal framework for analyzing the notion of irrelevance. The framework contains a space of possible definitions of irrelevance claims that enabled to formalize previous definitions (e.g., [Subramanian, 1989]) and present new ones. We identified several important properties of irrelevance claims and demonstrated how these properties change as we move in the space of definitions. The framework enabled us to irrelevance claims that serve as justifications for abstractions, thereby providing a new view on work in abstractions. Justifying abstractions by irrelevance claims provides a *first principles* [Subramanian, 1989] account of abstractions, elucidating questions such as automatically creating abstractions, creating abstractions that are specific for a given goal and using domain knowledge to guide the creation of abstractions. This paper presents only initial work on in this direction and much remains to be explored.

$Neg(C)'$, we will get the converse direction too, i.e, if a goal has a proof in the abstract theory, it will have one in the ground theory in which all facts are independent of the predicate distinction.

Acknowledgements

I would like to thank several people for discussions on the topics discussed in this paper: Adnan Darwiche, Ed Feigenbaum, Richard Fikes, Fausto Giunchiglia, Pat Hayes, Yumi Iwasaki, Hiroshi Motoda, Pandu Nayak and Shuky Sagiv. Much of this work was done while I was visiting the Hitachi Advanced Research Laboratory, and I would like to thank Hitachi, and especially Hiroshi Motoda for their generous support.

References

Amarel, Saul 1981. On representations of problems of reasoning about actions. In Webber, Bonnie L. and Nilsson, Nils J., editors 1981, *Readings in Artificial Intelligence*. Morgan Kaufmann, Los Altos, CA.

Clancey, William J. 1983. The advantages of abstract control knowledge in expert system design. In *Proceedings of the Third National Conference on Artificial Intelligence*, Los Altos, CA. Morgan Kaufmann. 74–78.

Genesereth, Michael R. and Nilsson, Nils J. 1987. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, Los Altos, CA.

Genesereth, Michael R. 1988. Introspective fidelity. In *Meta-Level Architectures and Reflection*. Elsevier Science Publishers B.V. (North Holland). 75–86.

Giunchiglia, Fausto and Walsh, Toby 1991. A theory of abstraction. Submitted to Journal of Artificial Intelligence.

Hayes, Patrick J. 1973. Computation and deduction. In *Proceedings of the 1973 Mathematical Foundations of Computer Science Symposium, Czechoslovakian Acadamy of Sciences.*

Kifer, M. 1988. On safety, domain independence, and capturability of database queries. In *Proceedings of the International Conference on Data and Knowledge Bases, Jerusalem.*

Knoblock, Craig; Tenenberg, Josh D.; and Yang, Qiang 1991. Characterizing abstraction hierarchies for planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, Cambridge MA. MIT Press. 692–697.

Knoblock, Craig A. 1990. Learning abstraction hierarchies for problem solving. In *Proceedings of the*

Eighth National Conference on Artificial Intelligence, Los Altos, CA. Morgan Kaufmann.

Levy, Alon Y. and Sagiv, Yehoshua 1992. Constraints and redundancy in datalog. In *To appear in the Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Diego, CA.*

Levy, Alon Y. 1992. Creating abstractions using relevance claims. In preparation.

Plaisted, D. 1981. Theorem proving with abstraction. In *Artificial Intelligence.* Vol. 16, pp. 47–108.

Sagiv, Yehoshua 1988. Optimizing datalog programs. In Minker, Jack, editor 1988, *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann, Los Altos, CA. 659–698.

Shmueli, Oded 1987. Decidability and expressiveness aspects of logic queries. In *Proceedings of the 6th ACM Symposium on Principles of Database Systems.* 237–249.

Smith, David E. and Genesereth, Michael R. 1985. Ordering conjunctive queries. In *Artificial Intelligence.* 26(2) pp. 171–215.

Subramanian, D. and Genesereth, M.R. 1987. The relevance of irrelevance. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Los Altos, CA. Morgan Kaufmann.

Subramanian, Devika 1989. A theory of justified reformulations. In *Ph.D thesis, Dept. of Computer Science, Stanford University.* Stanford, CA.

Tenenberg, Josh D. 1987. Preserving consistency across abstraction mappings. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Los Altos, CA. Morgan Kaufmann. 1011–1014.

Tenenberg, Josh D. 1990. Abstracting first order theories. In Benjamin, Paul, editor 1990, *Change of Representation and Inductive Bias.* Kluwer, Boston, Mass.

Ullman, Jeffery D. 1989. *Principles of Database and Knolwedge-base Systems, Volume I, II.* Computer Science Press, Rockville MD.

Alon Levy

Computer Science Department
Stanford University,
Stanford, CA, 94305,
email: levy@cs.stanford.edu

Research Summary

My research is focussed on studying the notion of irrelevance in different areas of problem solving. I have developed a general framework in which one can study the issues concerning irrelevance and how it can be applied to problem solving. The framework has been applied to three different areas.

The first is in the context of deductive databases [LS92]. There we have shown how analyzing the notion of irrelevance allows us to identify new kinds of redundancies in Datalog programs (Horn rules without function symbols), and we have given new algorithms for detecting such redundancies.

The second is in work on abstractions. Much of the work in AI on creating abstraction hierarchies and problem reformulation is based on the intuition that we abstract a problem by removing *irrelevant* detail. [Lev92] shows how such irrelevance claims can be formalized in my framework. Doing so enables one to explicitly state justifications for abstractions, therefore enabling the system to reason about the abstraction hierarchies it creates for a given goal. In addition, algorithms for automatically deducing irrelevance can be used to automatically create abstractions.

The framework has been applied to the problem of modeling physical devices [LIM92]. When a system chooses a model for a device given some task such as diagnosis, design or simulation, it needs to make decisions about which aspects of the device are relevant to the current task. We have shown how heuristics for guiding model selection can be stated as irrelevance claims in the language provided by the framework. We were able to express heuristics that have been discussed in the literature and to come up with new ones motivated by the vocabulary existing in the framework.

References

[Lev92] Alon Y. Levy. Irrelevance in problem solving. Knowldedge System Laboratory Technical Report, Computer Science Department, Stanford University., 1992.

[LIM92] Alon Y. Levy, Yumi Iwasaki, and Hiroshi Motoda. Relevance reasoning to guide compositional modelling. Knowledge System Laboratory Technical Report. Computer Science Department, Stanford University, 1992.

[LS92] Alon Y. Levy and Yehoshua Sagiv. Constraints and redundancy in datalog. In *To appear in the Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Diego, CA.*, 1992.

Generation and Exploitation of Aggregation Abstractions for Scheduling and Resource Allocation

Theodore A. Linden
Advanced Decision Systems
1500 Plymouth St.
Mountain View, CA 94043
linden@ads.com

Michael R. Lowry
Kestrel Institute
3260 Hillview Avenue
Palo Alto, CA 94306
lowry@kestrel.edu

Abstract

Our research is investigating abstraction of computational theories for scheduling and resource allocation. These theories are represented in a variant of first order predicate calculus, parameterized multi-sorted logic, that facilitates specification of large problems. A particular problem is conceptually stated as a set of ground sentences that are consistent with a quantified theory. We are mainly investigating the automated generation of aggregation abstractions and approximations in which detailed resource allocation constraints are replaced by constraints between aggregate demand and capacity. We are also investigating the interaction of aggregation abstractions with the more thoroughly investigated abstractions of weakening operator preconditions. The purpose of the theories for aggregated demand/capacity is threefold: first, to answer queries about aggregate properties, such as gross feasibility; second, to reduce computational costs by using the solution of aggregate problems to guide the solution of detailed problems; and third, to facilitate reformulating theories to approximate problems for which there are efficient problem solving methods. We also describe novel methods for exploiting aggregation abstractions.

Motivation

Domain specific planning and scheduling systems have achieved a modicum of real world success, and current efforts are aimed at vastly increasing the size and complexity of problems which can be handled with knowledge-based technology. We believe that much of the power of domain-specific planning and scheduling systems comes from their use of specialized algorithms at different levels of abstraction. For example, a resource allocation problem can often be approximated as linearized upper and lower bounds at a high level of abstraction, and solved using linear programming methods in order to identify bottleneck resources. Domain-specific scheduling systems use many different kinds of abstraction, not just the abstraction hierarchies defined by dropping literals from operator preconditions, as is the case for ABSTRIPS and most of its progeny. In particular, for large scheduling and resource allocation problems whose computational complexity is characterized by resource contention between many separate tasks, aggregation abstractions of demand and resource capacity play a more dominant role than abstraction of operator preconditions. An example is to aggregate all transportation capacity into a single linear quantity - total cargo volume. However, the drawback of domain-specific systems is their lack of flexibility and the necessity of hand-coding the knowledge.

The objective of our research is to develop the technology for dynamically 'compiling' domain-specific scheduling systems from declarative specifications and the subgoals and constraints that arise during planning and scheduling. The goal is to achieve the efficiency of hand-coded domain-specific systems but at the same time maintain the benefits of domain independent systems which interpret declarative problem specifications. The benefits of the latter arise from their generality: because the assumptions are explicit rather than hard-coded, the system is more widely applicable, the declarative representation is more transparent and thus more trusted and more easily validated, and furthermore the representation is more easily modified as requirements evolve. The automated synthesis and selection of abstractions is a key component to enabling domain-specific systems to be compiled from declarative specifications.

The next section of this paper describes the underlying semantics we are using for abstractions, approximations, and aggregations. The subsequent section describes the techniques we are developing for generating abstractions and approximations. The final section describes new techniques for exploiting aggregation approximations and abstractions.

Semantics of Abstractions, Approximations, and Aggregations

Semantically, we define an abstraction as a (possibly partial) mapping from the models of one theory to the models of another theory. We assume that these mappings are transitive and reflexive. If a mapping is total and can be inverted, then the two theories it relates are isomorphic.

The intended semantics of a theory determine the appropriate constraints on a valid abstraction. We define the appropriate abstraction constraints through the converse mapping: implementation. For *loose* specifications, the intended semantics is any model satisfying the theory, hence a valid implementation is a mapping from the models of the implementing theory into (but not necessarily onto) the models of the loose specification. This is compatible with definitions of abstraction as theory generalization, i.e. a widening of the class of models. For this type of semantics, implementation is the same as theory refinement.

For *tight* specifications, the intended semantics is a minimal model (up to isomorphism). Minimal model semantics correspond to the operational semantics of most types of logic programming. Minimal model semantics are also useful for succinctly axiomatizing models with inductive types, the simplest being the natural numbers. Hence the appropriate constraint on abstraction is a mapping from a single concrete model to a single abstract model. A third type of specification is *parameterized*, consisting of a parameter theory, which has many interpretations, and a body which extends the parameter theory. The usual intended semantics for this type of specification is to take *all* the models of the parameter theory, and then to extend each one with additional objects, functions and relations such that this extension is minimal with respect to all possible extensions consistent with the body. This third type of semantics is best seen as a tight specification for each model of the parameter theory. This type of semantics is most useful when a general specification is given for a whole class of problems.

We have axiomatized various types of generic resources using parameterized specifications: consumable resources (such as fuel), reusable non-shareable resources (such as a landing strip), synchronized-shareable resources (such as a cargo ship), and independent-shareable resources (such as a parking lot). A particular domain theory is built up by composing instantiations of these generic parameterized resource theories with particular resources.

Syntactically, an abstraction is defined through two theories and a set of definitions for abstraction functions from the objects and operations of the concrete model(s) to the objects and operations of the abstract model(s). The abstraction functions are defined in the syntax of the union of the abstract and concrete theories.

Approximations arise from weakening or strengthening the criteria for models of a theory. In the context of our research, this weakening/strengthening is always with respect to queries or goals. For example, if the goal is to transport cargo from one country to another given a certain set of resources, then the satisfaction of a strengthened approximation guarantees the transportation feasibility of the original, while the non-satisfaction of a weakened approximation guarantees the transportation infeasibility of the original. Strengthening and weakening occur not only with respect to the truth of sentences but also with respect to any partial order, such as the total order on the reals (true/false defines a partial order on the booleans). For example, approximations can also be upper and lower bounds on resources required for transportation feasibility. Given a complex query or goal it is necessary to map strengthening/weakening of the whole into strengthening/weakening of the parts. The polarity analysis for sentences [Manna & Waldinger 86] has been extended to a polarity analysis for any type of formula ranging over any domain with a partial order [Smith 92]. Thus given a complex query with a specified direction of strengthening or weakening, the constraints on the strengthening/weakening of the functions and relations in the query can be mechanically derived.

Aggregations are mappings from collections of objects with their individual attributes to a whole representing the collection with attributes for the collective. In theory aggregations can arise as equivalent conditions for satisfaction of a goal. For example, in order for a chemical reaction to occur in a solution the individual molecules must have sufficient kinetic energy. This constraint on the attributes of individual molecules can be reformulated into an equivalent constraint on the temperature of the whole solution. In the context of the research reported in this paper aggregations are most often approximations with respect to a query or goal.

Generation of Aggregation Abstractions

We are using two techniques for automatic generation of aggregation approximations. The first is based on analysis of behavioral equivalence: given a goal, two objects are behaviorally equivalent if they can be mutually substituted for each other in the achievement of the goal. For example, for the goal of transporting a rifle division, two small cargo planes are behaviorally equivalent to one large cargo plane. However, for transporting a heavy armor division only the large cargo plane has a sufficient girth for tanks and heavy artillery. Thus for transporting heavy armor divisions two small cargo planes are not behaviorally equivalent to a single large cargo plane. This simple example illustrates that abstractions such as total lift capacity must be dependent on context in order to be useful.

The result of behavioral equivalence analysis is the definition of an equivalence relation on the objects of a domain. In an abstract theory, behaviorally equivalent objects are identified. Syntactically, the equivalence relation in the concrete theory is transformed to an equality relation in the abstract theory. A number of issues arise in ensuring that the transformation from a behavioral equivalence relation to an equality relation is semantically well defined, particularly for inductively defined types. These issues are addressed in [Lowry 1989, Lowry 1990].

When behavioral equivalence analysis is applied to a set of goals, or to a complex domain theory with many constraints, the result will be a set of behavioral equivalences. (The behavioral equivalence for the conjunction of the goals is the intersection of the individual equivalence relations.) This set can be ordered by inclusion, defining a partial order on behavioral equivalence relations.

Upper and lower bounds exist for each pair of behavioral equivalence relations; a lattice is defined by including the universal equivalence relation (all objects equivalent) and the identity equivalence relation (each object is equivalent only to itself). This lattice can be more densely ordered by inheriting ordering relations on the goals and constraints. For example, ABSTRIPS type orderings on literals in the preconditions of operators derived through various programs [Tenenberg 89; Knoblock 89, 90] can be used to order their corresponding behavioral equivalence relations.

The second technique for generation of aggregation approximations is through the use of bounding approximations: given a goal or query, and an abstract domain theory obtained through behavioral equivalence analysis, an extended polarity analysis is applied to the goal(s) with respect to the abstract domain theory. Various kinds of symbolic bounding approximations are derived by KIDS through this polarity analysis, which is currently implemented through a transitive rewriting technique on formulas called directed inference [Smith 90]. These approximations include necessary conditions, sufficient conditions, symbolic upper bounds, and symbolic lower bounds. These approximation functions are composed with the abstraction functions derived through behavioral equivalence analysis to yield the abstraction (approximation) functions that map the concrete domain onto the abstract domain.

To derive aggregation approximations, the domain theory must include generic axioms defining the relation between aggregate constraints and constraints on individuals. Most of the resources we are considering are linear: resources have time-varying capacities which at each instance cannot be exceeded by the sum of the consumers assigned to that resource. The axioms for these kinds of generic aggregation constraints, together with a particular domain theory, are transformed by the polarity analysis into definitions of possible aggregation approximations.

Like the behavioral equivalence abstractions, the aggregation approximations derived through polarity analysis can be ordered by strength. Furthermore, directed inference, because it is a transitive rewriting technique, automatically generates part of this ordering relation. The composition of the lattice of behavioral equivalence relations and the ordering on aggregation approximations again yields a lattice. We are investigating techniques for this composed lattice to be implicitly defined rather than explicitly generated.

Exploiting Resource Abstractions

For very large resource allocation and scheduling problems that are solved interactively, some form of greedy algorithm is often appropriate. In this section, we illustrate the use of resource abstraction hierarchies to enhance the opportunities for finding a linear ordering of allocation and scheduling decisions that achieves good results within the context of a greedy algorithm. The same resource abstraction hierarchies can also be used to enhance the variable ordering heuristics used with other search strategies.

The example described later in this section shows that resource abstractions can enable a successful linear ordering of resource allocation decisions where no such ordering exists without the abstractions. The resource abstractions allow allocation decisions (assignments of resources to operations) to be made in steps down the resource abstraction hierarchy; first an abstract resource is allocated to an operation, later, this decision is refined to a more specific resource. Each allocation of an abstract resource is essentially a reservation for an unspecified instance of that abstract group of resources. By making the reservation early while deferring more specific resource choices, the deferred decisions can take advantage of information that becomes available as other decisions are made.

The cost of using resource abstraction hierarchies is the additional checking needed to maintain consistency of the allocation as it is built incrementally. Each allocation decision must be checked to ensure that it does not overuse the resource that is being assigned. With abstract resources, each decision must be checked not only against the resource being assigned but also against the more abstract resources.

To formalize these concepts, assume we have a resource abstraction hierarchy in the form of a lattice (**R**, »=) where **R** is a set of resource types and »= is the **extends relation** which is a partial ordering defined on **R** meaning "is more specific than (or equal to)." In the example discussed later in this section, the specific resources are seaports and **R** consists of individual seaports and selected sets of seaports. The lattice relation on **R** is membership or subset. For example, Norfolk » tank-loading seaports » East Coast seaports. Similarly, Baltimore » mid-Atlantic seaports.

Given a set of specific resources, the power set of these resources is a candidate lattice; however, this is typically a bad candidate because it would involve exponential cost in checking resource assignments against the more abstract nodes. While a narrow lattice can be developed at design time; it is likely to be far more effective to choose abstractions that are tailored to the specific problem instance using the techniques discussed previously of behavioral equivalence analysis and aggregation approximations.

Figure 1 depicts a small portion of a simplified problem involving military crisis action deployments. A large number of force modules (such as all the equipment associated with a brigade) are to be shipped through East Coast ports. Two of the hundreds of such force modules (FM15 and FM35) are identified in the figure. The problem addressed in this example is to plan the deployment without excessive congestion at the ports (other aspects of the overall problem involve scheduling of transports and other resources).

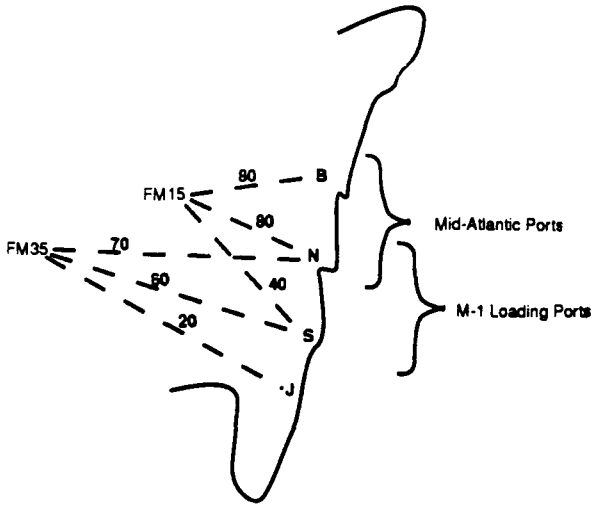


Figure 1: Problem of assigning seaports to transportation tasks

The two force modules can be shipped through any of 4 different ports, labeled B, N, S, and J. The dashed lines between the force modules and the ports that are suitable for shipment are labeled with the utility of using this route. That utility summarizes the current state of information about the cost of ground transportation to the port, the locations of available transport ships, characteristics of the ports, and other factors.

If this port selection problem could be isolated from other aspects of the overall crisis action planning problem, it could be solved by existing OR algorithms. Unfortunately, there are many complex dependencies between the port selection and the scheduling of other resources that require the port selection decisions to be interleaved with other decisions within an overall multi-user, interactive construction paradigm. Within this context, the port selection is done with a greedy algorithm, and a goal is to use the locally available information to order the allocation decisions.

This example focuses on deciding whether to choose a port for FM15 before or after choosing one for FM30. Without abstraction levels, neither ordering can be successful under all interactions with other demands on the port resources. If we use the simple greedy approach and assign to FM15 first because it can achieve a higher utility, then the arbitrary choice that has be made between B and N may prevent FM35 from obtaining its best choice. (We can't choose B knowing that FM35 prefers N because we are assuming there are many other force modules that are also competing for both B and N, and many of them may have higher priority than FM35.) On the other hand, if the assignment is made to FM35 first, it may choose port N, but that could be the cause of FM15 not getting either of its good choices. This problem of ordering these two allocation decisions (and the decisions about all other pairs of force modules) might be avoided by using statistical look-ahead techniques [Fox 89] that project the contention at the ports and may allow FM15 to choose first and make a non-arbitrary choice between B and N. Separating the decisions across abstraction level gives additional opportunities to be successful with a greedy algorithm, and appears to be especially useful in conjunction with statistical look-ahead techniques. Note that no technique can make a greedy algorithm successful all the time.

Two abstractions on the seaports are shown in Figure 1—mid-Atlantic ports and M-1 loading ports. For the abstractions to be effective, the utility function should often be more homogeneous across different instances of the abstract resource than across the entire domain of the resources. This can be a goal when creating abstractions dynamically.

The mid-Atlantic port abstraction enables an ordering of the port allocation decisions for these two force modules that will almost always be successful:

1) Reserve a mid-Atlantic port for FM15 (assuming it competes successfully with other force modules for these ports).

2) Reserve an M-1 loading port for FM35 (assuming it competes successfully with other force modules for these ports).

3) Assign N to FM35 assuming it preserves all reservations for both mid-Atlantic and M-1 loading ports and competes successfully for N with the other force modules.

4) Assign to FM15 whichever instance of a mid-Atlantic port is left over. (The reservation guarantees that some mid-Atlantic port will be left for FM15)

The reservation that FM15 has for a mid-Atlantic port allows a lower priority force module to be given precedence over the higher priority module in step three—as long as the reservation is preserved.

This four step ordering of the decisions about these two force modules often achieves a good solution without backtracking. Similar reasoning for other pairs of force modules can be used to order the other decisions made by a greedy algorithm—but there are no guarantees that a good ordering can be found for all pairs of decisions.

Bibliography

[Fox 89] Mark S. Fox et al, "Constrained Heuristic Search", Proc. IJCAI-89, Morgan Kaufmann Publ.

[Knoblock 89] Craig A. Knoblock, "A Theory of Abstraction for Hierarchical Planning", *Change of Representation and Inductive Bias*, ed. D.P. Benjamin, Kluwer 1989

[Knoblock 90] Craig A. Knoblock, "Learning Problem-Specific Abstraction Hierarchies", Proc. of Workshop on Change of Representation and Problem Reformulation; Menlo Park, CA March 1990

[Linden 89] Theodore A. Linden, "Planning by Transformational Synthesis," *IEEE Expert*, 4,2 Summer, 1989, pp. 46-55.

[Linden 90] Theodore A. Linden, "Transformational Synthesis: A Paradigm for Building Large-Scale Planning Applications," *Planning Systems for Autonomous Mobile Robots*, ed. D. P. Miller and D. J. Atkinson, Prentice Hall, 1989.

[Lowry 89] Michael R. Lowry, "STRATA: Problem Reformulation and Abstract Data Types", in Change of Representation and Inductive Bias, Edited by Paul Benjamin, Kluwer Academic Publishers 1989

[Lowry 89] Michael R. Lowry, "Algorithm Synthesis through Problem Reformulation", PhD Thesis, Stanford University, 1989

[Lowry 90] Michael R. Lowry, "Abstracting Domains with Hidden State", in Proc. of Workshop on Automatically Generating Abstractions and Approximations, AAAI-90

[Manna & Waldinger 86] Manna, Z. and Waldinger, R. Special Relations in automated deduction. *Jorunal of tthe ACM 33, 1 pp.* 1-59.

[Smith-Lowry-89] Smith, D.R. and Lowry, M.R., Algorithm Theories and Design Tactics, in Proceedings of the International Conference on Mathematics of Program Construction, LNCS-375, Springer-Verlag, Berlin, June 1989, 379-398 (to appear in Science of Computer Programming, 1990).

[Smith-90] Smith, D.R., KIDS: A Semi-Automated Program Development System, in IEEE Transactions on Software Engineering special issue on Formal Methods, September 1990.

[Smith-92] Smith, D.R., Connecting Specification Morphisms, in Proceedings of the International Conference on Mathematical Theory of Computation.

[Tenenberg 89] Josh Tenenberg, "Abstracting First-Order Theories", *Change of Representation and Inductive Bias*, ed. D.P. Benjamin, Kluwer 1989

Symmetry as Bias: Rediscovering Special Relativity

Michael R. Lowry

AI Branch
M.S. 269-2
NASA Ames Research Center
Moffett Field, CA 94035
lowry@pluto.arc.nasa.gov

Abstract

This paper describes a rational reconstruction of Einstein's discovery of special relativity, validated through an implementation: the Erlanger program. Einstein's discovery of special relativity revolutionized both the content of physics and the research strategy used by theoretical physicists. This research strategy entails a mutual bootstrapping process between a hypothesis space for biases, defined through different postulated symmetries of the universe, and a hypothesis space for physical theories. The invariance principle mutually constrains these two spaces. The invariance principle enables detecting when an evolving physical theory becomes inconsistent with its bias, and also when the biases for theories describing different phenomena are inconsistent. Structural properties of the invariance principle facilitate generating a new bias when an inconsistency is detected. After a new bias is generated, this principle facilitates reformulating the old, inconsistent theory by treating the latter as a limiting approximation. The structural properties of the invariance principle can be suitably generalized to other types of biases to enable *primal-dual* learning.

Introduction[1]

Twentieth century physics has made spectacular progress toward a grand unified theory of the universe. This progress has been characterized by the development of unifying theories that are then subsumed under even more encompassing theories. Paradigm shifts are nearly routine, with the postulated ontology of the universe changing from the three dimensional absolute space of Newtonian physics, to the four dimensional space-time of relativistic physics, and through many other conceptual changes to current string theories embedded in ten dimensions. Theoretical physicists attribute much of the success of their discipline to the research strategy first invented by Einstein for discovering the theory of relativity [Zee 86].

At the heart of Einstein's strategy was the primacy of the principle of *invariance:* the laws of physics are the same in all frames of reference. This principle applies to reference frames in different orientations, displaced in time and space, and moreover to reference frames in relative motion. This principle also applies to many other aspects of physics, including symmetries in families of subatomic particles. The application of the invariance principle to "two systems of coordinates, in uniform motion of parallel translation relatively to each other" was Einstein's first postulate: the principle of special relativity [Einstein 1905].

Einstein's genius lay in his strategy for using the invariance principle as a means of unifying Newtonian mechanics and Maxwell's electrodynamics. This strategy of unifying different areas of physics through the invariance principle is responsible for many of the advances of theoretical physics. In the parlance of current machine learning theory, Einstein's strategy was to combine the principle of special relativity with his second postulate, the constancy of the speed of light in a vacuum, to derive a new bias. (This second postulate was a consequence of Maxwell's equations; [Einstein 1905] notes that experimental attempts to attribute it to a light medium were unsuccessful.) This new bias was designed and verified to be consistent with Maxwell's electrodynamics, but was inconsistent with Newton's mechanics. Einstein then reformulated Newton's mechanics to make them consistent with this new bias. He did this by treating Newton's mechanics as a limiting approximation, from which the relativistic laws were derived through generalization by the new bias.

Einstein's strategy is a model for scientific discovery that addresses a fundamental paradox of machine learning theory: to converge on a theory from experimental evidence in non-exponential time, it is necessary to incorporate a strong bias [Valiant 84], but the stronger the bias the more likely the 'correct' theory is excluded from consideration. Certainly any conventional analysis of what could be learned in polynomial time would exclude a sufficiently unified theory of physics. The paradox can be avoided

[1]This research was supported in part by a sabbatical while the author was a member of the Kestrel Institute, and in part by a subcontract through Recom Technologies to NASA Ames Research Center.

machine learning algorithms that have capabilities for reasoning about and changing their bias. Even if a strong bias is ultimately 'incorrect', it is still possible to do a great deal of useful theory formation before the inconsistencies between the bias and empirical facts becomes a limiting factor. The success of the Galilean/Newtonian framework is an obvious example. To avoid the paradox, a machine learning algorithm needs to detect when a bias is inconsistent with empirical facts, derive a better bias, and then reformulate the results of learning in the incorrect bias space into the new bias space [Dietterich 91]. The Erlanger program described in this paper is such an algorithm.

Einstein's strategy is essentially a mutual bootstrapping process between two interrelated hypothesis spaces: a space for biases, and a space for physical theories. The invariance principle defines the space of biases; each bias is a different postulated set of symmetries of the universe, formalized through a group of transformations. The invariance principle also defines a consistency relationship that mutually constrains the bias space and the space for physical theories. The hypothesis space for biases has a rich lattice structure that facilitates generating a new bias when a shift of bias is necessary. The hypothesis space for physical theories has an approximation relation between theories (limit homomorphisms) that, after a shift in bias, facilitates generating a new theory from an old (approximate) theory and the new bias. The entire process converges if learning in the bias space converges.

This paper builds upon the considerable body of literature on relativity and the role of symmetry in modern physics. Its contribution includes identifying and formalizing the structural relationships between the space of biases and the old and new theories that enabled Einstein's strategy to succeed, in other words, made it computationally tractable. The tactics for carrying out the components of this strategy have been implemented in the Erlanger program, written in Mathematica v.1.2.

The next section of this paper presents an overview of Einstein's strategy. The following section introduces the invariance principle, which determines the consistency relationship between a bias and a physical theory. It also describes the procedure for detecting inconsistency. The following section presents the tactic for computing a new bias using the invariance principle. It takes the reader through the Erlanger program's derivation of the Lorentz transformations. The section after defines *limit homomorphisms*, a formal semantics for approximation. The following section describes BEGAT: BiasEd Generalization of Approximate Theories, an algorithm that uses the invariance principle and the semantics of limit homomorphisms to generate components of the new theory. The paper concludes with a generalization of Einstein's strategy called *primal-dual* learning, which might be applied to other types of biases.

Overview of Einstein's Strategy

Einstein's strategy for deriving special relativity will first be explained through an analogy with symmetries and tangents of geometric figures. Then the structural components of the invariance principle interrelating the bias space and the space of physical theories will be outlined and the overall research strategy described with respect to these components. The next section will describe the mathematics of the invariance principle as it applies to theories of physics.

Symmetry and Group Theory

The symmetries of a geometric figure are invertible transformations that map the figure to itself. For example, a square is mapped to itself by various transformations about its center: horizontal reflections, vertical reflections, and ninety degree rotations. Because these transformations are invertible, they form a group.

A group is any set with a constant identity element, a binary operation defined on any two elements, and an inverse operation mapping any element to its inverse. A transformation group consists of elements which are transformations of some other set S; each transformation is a bijection from S to S. A transformation T defined on S is an *automorphism* of a subset $F \supseteq S$ iff $T(F) = F$. Hence if S is the two dimensional plane and F is a geometric figure such as a square, then the symmetries of F are those transformations T such that $T(F) = F$. Restrictions can be placed on the transformations considered; for example, transformations that preserve topological structure are called homeomorphisms while transformations that preserve distance are called isometries. The isometries of a square include horizontal reflections, vertical reflections, and multiples of ninety degree rotations about it center.

Symmetries can be represented through transformation equations; for example, the equations for a rotation of θ degrees about the origin in two dimensions define new primed coordinates for each point in terms of the original coordinates: $x' = x\cos\theta - y\sin\theta$, $y' = x\sin\theta + y\cos\theta$. If θ is a constant, then these equations represent a single transformation. If θ is a parameter, then these equations represent a set of transformations. Note that for any θ, a circle with its center at the origin is mapped to itself. Hence these equations denote a set of automorphisms of all origin-centered circles. One way to prove this algebraically is to solve for the equation of a circle, i.e., reduce $x^2 + y^2 = r^2$ to a set of functions for y in terms of x for different quadrants, plug the definitions of these functions into the transformation equations, and then show that the new points also satisfy these equations.

The method implemented in the Erlanger program is slightly different because it is based upon an equivalent but alternative approach to defining symmetries. (See [Friedman 83] for a thorough analysis of the relation

between these two approaches, as applied to space-time theories.) Instead of viewing the transformations as mappings from points to points within a single reference frame, the transformations are viewed as mappings between reference frames. A figure is symmetric if it appears exactly the same in the new reference frame as it does in the old reference frame. In this alternative approach, the transformation equations are the same except that the sign of the parameter is inverted, because rotating the reference frame θ about the origin is equivalent to rotating the figure by $-\theta$ about the origin: $x' = x\cos\theta + y\sin\theta$, $y' = -x\sin\theta + y\cos\theta$.

An Analogy to Einstein's Strategy

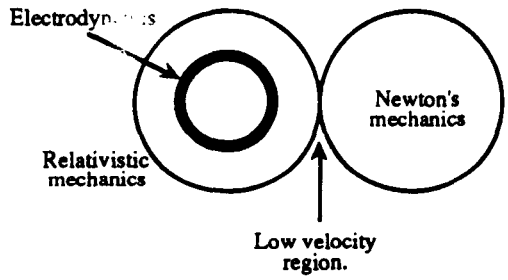


Figure 1.

Einstein's strategy for deriving special relativity is illustrated through the simple geometric analogy in Figure 1. Newton's mechanics is represented by the circle on the right, its set of symmetries are all rotations and reflections about its center. This set of symmetries is inconsistent with the invariance of the speed of light, a deductive consequence of Maxwell's electrodynamics that is represented by the small bold circle on the left.

Einstein derived the set of symmetries consistent with the constant speed of light by first generalizing from the particular circle representing Newton's mechanics to symmetries for all possible circles, i.e., rotations and reflections about all possible centers. He then specialized this set of all possible circular symmetries by solving for the center of the circle consistent with the constant speed of light. This new symmetry was verified to be consistent with Maxwell's electrodynamics.

Einstein then derived relativistic mechanics, represented by the larger left circle, through two constraints: that it be circularly symmetric around the same center as electromagnetic phenomena, and that it be tangent to Newton's mechanics as relative velocities approach 0. Note that Newton's mechanics had only been empirically verified at low velocities compared to light; the rest of the circle was assumed from the originally postulated symmetries dating back to Galileo. In this manner Einstein unified electromagnetism and mechanics under the same set of symmetries while still accounting for the wealth of experimental confirmations of Newton's theory at low velocities compared to the speed of light. Although simplistic, this geometric analogy captures the essential extensional relationships between Newton's mechanics, Maxwell's electromagnetism, and relativistic mechanics.

One of the crucial facts about symmetry as bias is that the groups corresponding to different figures form a lattice ordered by the subset relation. (More generally, the ordering is defined through group homomorphisms.) There is a contravariant relation between the complexity of an object and its set of symmetries. For example, a square is more complex than a circle, hence the group of transformations for a square is a subset of the group of transformations for a circle. As explained in the next section, this relation between geometric figures and their symmetries also holds between theories of physics and their symmetries. This contravariant relation is essential to the bootstrap learning of Einstein's strategy.

Structural Relations Exploited in Einstein's Strategy

Figure 2 illustrates the structural relations between the bias space and the space for physical theories that was exploited by Einstein, and indicates how these same structural relations might be exploited in other types of bias.

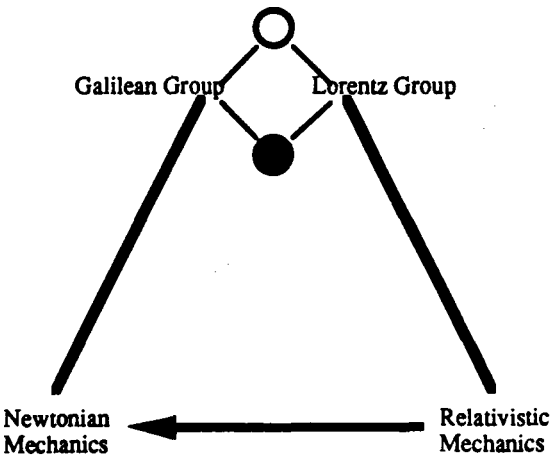


Figure 2

1. The diamond represents a space of biases for physical laws. The biases are different postulated symmetries of the universe. As modern physics has evolved, the bias has evolved. Each bias in this space is formalized as a transformation group.

2. The consistency relationship between a bias (a transformation group) and a physical theory is represented by a solid black line. The diagram illustrates that

Newtonian mechanics is consistent with the Galilean transformation group.

3. When an inconsistency is detected between an experimental fact and the current bias, then a new bias is computed. The new bias is computed by combining the new observation, an upper bound (represented by a hollow circle) and lower bounds (represented by a solid black circle). The upper bound is a superset of transformations that constrains the types of transformations that are considered. The transformations in this superset that are consistent with the new observation are selected for the new bias. This selection is done by symbolically solving for those transformations that are consistent with the new observation, rather than enumerating over all the transformations in the upper bound. The calculation is simplified through the use of lower bounds. Einstein derived the Lorentz transformations through this procedure.

4. Laws in the new hypothesis space are constrained to be consistent with the new bias and also to have, as a limiting approximation, the laws in the old hypothesis space. This limiting approximation is indicated by the arrow from relativistic mechanics to Newtonian mechanics. In fact, the new laws can often be derived from the old laws by using the new bias to reformulate the old laws. This was the method Einstein used to generate relativistic mechanics.

The power of Einstein's strategy is that his framework scales up from special relativity through the history of twentieth century physics, although the mathematics becomes considerably more complex. From the viewpoint of machine learning, the power of Einstein's strategy is his mutual bootstrapping between the bias space and the hypothesis spaces by exploiting the structural relationship between them: the invariance principle.

Symmetry as Bias: the Invariance Principle

Symmetry is a unifying aesthetic principle that has been a source of bias in physics since ancient times. In modern physics this principle is stated as: 'the laws of physics are invariant for all observers.' An invariance claim is a universally quantified statement of the form 'For all events/histories of type F, for all reference frames of type R, Physical Theory P holds'. An invariance claim implies that a group of transformations mapping measurements between different observers also maps physical theory P onto itself. Such a group of transformations defines the postulated symmetries of the universe, and is the type of bias used by theoretical physicists. The transformations are parameterized by the relation between two different observers, such as their relative orientation or velocity. For example, Galileo defined the following transformation equations relating measurements for observers in constant relative velocity v parallel to the x-axis: $\{x' = x - vt, t' = t\}$ These transformations are consistent with Newton's theory of mechanics.

The invariance principle defines a consistency relationship between physical theories and groups of transformations. The following definitions are standard and sufficient for our purpose of understanding and implementing Einstein's strategy for deriving special relativity. However, the reader should be aware that these definitions are a simple starting point for a deep, well developed mathematical theory that has had a profound impact on theoretical physics. (A good mathematical exposition focused on special relativity is [Aharoni 65], a more sophisticated philosophical and foundational treatment is [Friedman 83].)

Below, $\mathcal{G}$ is a transformation group. An invariant operation is a special case of a covariant operation. Laws are invariant if they define the same relation after they are transformed by the action of the transformation group. A sufficient condition for a theory to be invariant with respect to a transformation group $\mathcal{G}$ is if all the operations are covariant and all the laws are invariant.

Invariance of an operation or form:

$$\text{Invariant}(op,\mathcal{G}) \Leftrightarrow \forall(g \in \mathcal{G}, x1 \ldots x_n)$$
$$op(x_1,x_2,\ldots,x_n) = op(g(x_1,x_2,\ldots,x_n))$$

Covariance of an operation or form:

$$\text{Covariant}(op,\mathcal{G}) \Leftrightarrow$$
$$\forall(g \in \mathcal{G}, x1 \ldots x_n)$$
$$op(g(x_1,x_2,\ldots,x_n)) = g(op(x_1,x_2,\ldots,x_n))$$
$$\Leftrightarrow \forall(g \in \mathcal{G}, x1 \ldots x_n)$$
$$op(x_1,x_2,\ldots,x_n) = g^{-1}(op(g(x_1,x_2,\ldots,x_n)))$$

Invariance of a physcial law expressed as a universally quantified equation:

$$\text{Invariant}(\forall(\ldots)\ t1(\ldots) = t2(\ldots),\ \mathcal{G}) \Leftrightarrow$$
$$\forall(g \in \mathcal{G}, x1 \ldots x_n)$$
$$t1(x_1,x_2,\ldots,x_n) = t2(x_1,x_2,\ldots,x_n)$$
$$\equiv t1(g(x_1,x_2,\ldots,x_n)) = t2(g(x_1,x_2,\ldots,x_n))$$

More generally, a theory is invariant with respect to a transformation group $\mathcal{G}$ iff all the transformations in the group are automorphisms of the models of the theory. This is equivalent to proving that the theory and the transformation equations together imply the same theory in other frames of reference (though see [Friedman 83] for qualifications).

Invariance of a theory $\mathcal{T}$:

$$\text{Invariant}(\mathcal{T},\mathcal{G}) \Leftrightarrow$$
$$\forall(g \in \mathcal{G})\,\text{Models}(\mathcal{T}) \equiv g(\text{Models}(\mathcal{T}))$$
$$\equiv \forall(g \in \mathcal{G})\,\mathcal{T} \models \mathcal{T}/g \text{ and } \mathcal{T}/g \models \mathcal{T}$$

where $\mathcal{T}/g$ denotes substituting variables with the terms defined by the transformation equations.

Because of the inverse property of groups, the two conjunctions imply each other:

$$\forall(g \in \mathcal{G})\,\mathcal{T} \models \mathcal{T}/g^{-1}$$

implies $\forall(g \in \mathcal{G})\,\mathcal{T}/g \models \left(\mathcal{T}/g^{-1}\right)/g$

implies $\forall(g \in \mathcal{G})\,\mathcal{T}/g \models \mathcal{T}$

To check an invariant predicate, the Erlanger program back-substitutes transformation equations into a form or law and then compares the result to the original form or law. If the function or relation are the same, then the

invariant predicate is true. In essence the Erlanger program assumes the law holds good in the original reference frame and then transforms the law into measurements that would be observed in a new frame of reference. (This can be done independent of whether the law is invariant.) If these measurements agree with the law stated in the new frame of reference, then the law is invariant. The steps of the algorithm are described below and illustrated with the example of determining whether the Galilean transformations are consistent with the constant speed of light, Einstein's second postulate. The input is the definition of the law for the constant speed of light, and the transformation equations relating variables in the original frame of reference to the variables in the new (primed) frame of reference:

Invariant($x^2 = c^2t^2$, $\{x = x' + vt', t = t'\}$)

1. Solve the law in the new frame of reference to derive expressions for dependent variables (This turns a relation between variables into a disjunction of sets of substitutions.): $\{\{x' = ct'\}, \{x' = -ct'\}\}$

2. Use the parameterized transformation equations to substitute expressions in the new frame of reference for variables in the old frame of reference; this yields a new law relating measurements in the new frame of reference: $(x' + vt')^2 = c^2t'^2$

3. The substitutions derived in step 1 are applied to the new law derived in step 2:

$\{(ct' + vt')^2 = c^2t'^2, (-ct' + vt')^2 = c^2t'^2\}$

4. If the law(s) derived in step 3 is a valid equality(ies), then the law(s) is invariant. For this example they are not, so the Erlanger program determines that Einstein's second postulate is inconsistent with the Galilean transformations.

Deriving a New Bias

The invariance principle can be used not only to verify that a physical law is consistent with a particular bias, but also to generate a new bias when a physical law is inconsistent with the current bias, as when the constant speed of light is inconsistent with the Galilean transformations. There are important structural aspects of the invariance principle that enabled this aspect of Einstein's strategy to succeed. In particular, the consistency relationship is contravariant: a weaker physical theory is consistent with a larger set of transformations. (For the purposes of this paper, 'weaker' can be thought of as 'fewer deductive consequences', though this is not entirely correct. This only holds if each law transforms into itself.) Thus when an inconsistency is detected between a bias represented by a set of transformations and an evolving physical theory, the physical theory can be relaxed, leading to an enlarged set of transformations. This enlarged set is then filtered to compute the new bias.

Assume that a physical theory T (e.g. Newton's mechanics) is consistent with a transformation group G (e.g. the Galilean group). Further assume that G is the largest transformation group consistent with T. Then a new empirical fact e is observed (e.g. the constant speed of light), such that e is not consistent with G. Then T is relaxed to T' (e.g. Newton's first law), thereby enlarging G to G' (e.g. the set of all linear transformations). The new bias is the subset of G', i.e. G''(e.g. the Lorentz group), such that T' with e is consistent with G''. Then the laws in $(T - T')$ are transformed so that they are consistent with G'' and have as limiting approximations the original laws. This section describes an implemented algorithm for deriving G'', while the next sections describe transforming the laws in $(T - T')$. These same algorithms can also be used when trying to unify theories with different biases, such as Newton's mechanics and Maxwell's electromagnetism.

The Lorentz group is a set of transformations that relate the measurements of observers in constant relative motion. The Lorentz group is a sibling to the Galilean group in the space of biases. Einstein's derivation of the Lorentz transformations implicitly relied upon structural properties of the lattice of transformation groups. In particular, Einstein constrained the form of the transformations with an upper bound, derived from Newton's first law: a body in constant motion stays in constant motion in the absence of any force. This is his assumption of inertial reference frames, an assumption he relaxed in his theory of general relativity. The largest set of transformations consistent with Newton's first law are the four dimensional linear transformations. Of these, the spatial rotations and spatial/temporal displacements can be factored out of the derivation, because they are already consistent with Einstein's second postulate. (The Erlanger program does not currently have procedures implemented to factor out subgroups of transformations - these are under development.) This leaves an upper bound for a subgroup with three unknown parameters (a,d,f) whose independent parameter is the relative velocity (v):

$$x = a(x' + vt') \quad t = dx' + ft'$$

This upper bound includes both the Galilean transformations and the Lorentz transformations. The DeriveNewBias algorithm takes the definition of an upper bound, such as the one above, including lists of the unknown and independent parameters, a list of invariants, a list of background assumptions, and information on the group properties of the upper bound. When this algorithm is applied to Einstein's second postulate of the constant speed of light, the derivation of the Lorentz transformations proceeds along roughly the same lines as that in Appendix 1 of [Einstein 1916]. This derivation and others are essentially a gradual accumulation of constraints on the unknown parameters of the transformations in the upper bound, until they can be solved exactly in terms of the independent parameter which defines the relation between two reference frames. The algorithm is described below, illustrated with the example of deriving the Lorentz transformations.

The input in this example to the DeriveNewBias algorithm is the upper bound given above, two invariants for a pulse of light - one going forward in the x direction

and one going backwards in the x direction $\{x = ct, x = -ct\}$, the assumptions that the speed of light is not zero and that the relative velocity between reference frames is less than the speed of light, and information for computing the inverse of a transformation. The steps of the DeriveNewBias algorithm:

1. Constraints on the unknown parameters for the transformation group are derived separately from each individual invariant. This step is similar to the procedure which checks whether a law is invariant under a transformation group. However, instead of steps 3 and 4 of that procedure, the system of equations from steps 1 and 2 are jointly solved for constraints on the unknown parameters. For the two invariants for a pulse of light, the derived constraints are:

$$a = (-c^2 d + cf)/(c - v), \quad a = (c^2 d + cf)/(c + v)$$

2. The constraints from the separate invariants are combined through Mathematica's SOLVE function. In the example of the Lorentz derivation, this reduces the unknown parameters to a single unknown (f):

$$a = f, \quad d = (fv)/c^2$$

3. In the last step, the group properties are used to further constrain the unknown parameters. Currently the implemented algorithm only uses the inverse property of a group, but the compositional property is another source of constraints that could be exploited. First, the constraints on the unknown parameters are substituted into the upper bound transformation definition, yielding a more constrained set of transformations. For the Lorentz example this yields:

$$x = f(x' + vt') \qquad t = ft' + fvx'/c^2$$

Second, the inverse transformations are computed. The information given to the algorithm on the group properties of the upper bound define how the independent parameter for the transformation is changed for the inverse transformation. For relative velocity, this relation is simply to negate the relative velocity vector . This then yields the inverse transformations:

$$x' = f(x - vt) \qquad t' = ft - (fvx)/c^2$$

The inverse transformations are then applied to the right hand side of the uninverted transformations, thereby deriving expressions for the identity transformation:

$$x = f\left(f(x - vt) + v\left(ft - \frac{fvx}{c^2}\right)\right)$$

$$t = f\left(ft - \frac{fvx}{c^2}\right) + \frac{fvf(x - vt)}{c^2}$$

These expressions are then solved for the remaining unknown parameters of the transformation (e.g. f), whose solution is substituted back into the transformations:

$$x = (x' + vt')\frac{\sqrt{c}}{\sqrt{2}}\sqrt{\frac{1}{c - v} + \frac{1}{c + v}}$$

$$t = \left(t' + \frac{v}{c^2}x'\right)\frac{\sqrt{c}}{\sqrt{2}}\sqrt{\frac{1}{c - v} + \frac{1}{c + v}}$$

The result is the new bias, which in this example is equivalent to the standard definition of the Lorentz transformations (the definitions above are in Mathematica's preferred normal form).

Limit Homomorphisms: Approximations between Theories.

Once a new bias is derived, a learning algorithm needs to transfer the results of learning in the old bias space into the new bias space. Unless the relationship between the old bias and the new bias can be exploited, in the worst case this means running the learning algorithm with the new bias over all the examples used to derive the old theory. The shift in bias from the Galilean transformation group to the Lorentz transformation group required a global reformulation of all the theories of physics, from kinematics to fluid dynamics, and later quantum mechanics. Yet in all these reformulations, the relativistic theory was derived from its non-relativistic counterpart without exhaustively considering the experimental evidence justifying the non-relativistic theory. This was done by treating the non-relativistic theory as an approximation to the new, unknown relativistic theory; and combining this constraint with the Lorentz transformations to derive a corresponding relativistic theory.

A theory such as Newton's mechanics that has a high degree of experimental validation over a range of phenomena (e.g. particles interacting at low velocities compared to the speed of light), represents a summary of many experimental facts. If a new theory is to account for these same experimental facts, it must agree with the observable predictions of the old theory over the same range of phenomena. Hence the old theory must approximate, to within experimental error, the new theory over this range of phenomena (and vice versa). By showing that an old theory is a limiting approximation to a new theory, it is unnecessary to exhaustively reconsider all the experimental evidence justifying the old theory. This approximation criteria for partially validating a new theory is well accepted, both within scientific communities and within the philosophy of science. However, the development of relativity theory went beyond a *post-hoc* verification of this approximation criteria: the approximation criteria was used to *derive* the new theory.

Various notions of "approximation" have been developed in AI to support reasoning between approximate theories, and even generating approximate theories from detailed theories [Ellman 90,92]. The problem of generating a new theory from an approximate theory and a new bias requires a precise definition of approximation with a well defined semantics. This section describes *limit homomorphisms*, which are homomorphisms that only hold in the limiting value of some parameter. Limit homomorphisms can be viewed as an extension of fitting parameter approximations [Weld 92] with additional algebraic structure that adds the constraints needed to

derive the new theory, and not just model the approximation relation.

A limit homomorphism is a map m from one domain to another such that for corresponding functions f_1 and f_2 the following equality converges as the limit expression goes to the limiting value:

$$\lim_{expr \to value} m(f_1(x_1 \ldots x_n)) = f_2(m(x_1) \ldots m(x_n))$$

Another motivation for this definition of approximation is to resolve a fundamental disagreement between Kuhn's view of the paradigm shift from Newtonian to relativistic physics, and the view of most physicists. Most physicists agree with the logical positivists: Newtonian physics is a limiting approximation of Einstein's physics. Kuhn argues that this is spurious [Kuhn 62, pg. 102], because the corresponding concepts in the relativistic and Newtonian mechanics are different. A limit homomorphism combines a map between corresponding concepts and a limiting approximation, thus achieving a limiting approximation between two different conceptual domains.

A well known type of limit homomorphism within computer science is Ω-order computational complexity. For example, the Ω-order computational complexity of a sequence of program statements is the maximum of the Ω-order computational complexity of the individual statements:

$$\lim_{input \to \infty} \Omega(S_1; S_2; \ldots S_n) = \mathrm{Max}(\Omega(S_1), \Omega(S_2), \ldots \Omega(S_n))$$

To determine the Ω-order computational complexity of a program, this limit homomorphism is recursively applied to the definition of a program. Similarly, to determine the non-relativistic quantity corresponding to a relativistic quantity, the appropriate limit homomorphism is recursively applied to the definition of the relativistic quantity.

Within physics, limit homomorphisms define the relationship between new, unified theories and the older theories they subsume. If the mapping function m is invertible, then a limit homomorphism can be defined in the reverse direction. The limit homomorphisms between Newton's mechanics and different formulations of relativistic mechanics are invertible. Thus from an *a priori*, mathematical viewpoint neither Newtonian mechanics nor relativistic mechanics is intrinsically more general than the other - the mathematical relationship is symmetric; each is a limit homomorphism of the other. These theories agree on their predictions when velocities are low, but diverge as velocities approach the speed of light. Relativistic mechanics is *a posteriori* more general because its predictions agree with experimental facts for high velocities, hence the theory is more generally applicable. Relativistic mechanics is also extrinsically more general in the sense that its bias is consistent with electrodynamics, and hence relativistic mechanics and electrodynamics can be unified.

BEGAT: (BiasEd Generalization of Approximate Theories)

While the intrinsic mathematical relationship between Newtonian and relativistic physics is not one of generalization [Friedman 83], the *process* of generating relativistic mechanics from Newtonian mechanics is one of generalization. This section describes the mathematics justifying this process, and an implemented algorithm based on these mathematics that derives relativistic kinematics. Extensions currently undergoing implementation are described that will enable it to derive different formulations of relativistic dynamics.

It is clear from a reading of [Einstein 1905] that Einstein derived relativistic mechanics from Newtonian mechanics, by treating the latter as a limiting approximation that was valid in low velocity reference frames and applying the Lorentz transformations in order to generalize to the relativistic laws. For example, in section 10, paragraph 2 of [Einstein 1905]: "If the electron is at rest at a given epoch, the motion of the electron ensues in the next instant of time according to the equations [Newton's equations of motion] ... as long as its motion is slow." Einstein then generalized to the relativistic equation of motion by applying the Lorentz transformations to Newton's equations of motion. Einstein even constrained the laws of relativistic dynamics to have the same form as Newtonian dynamics.

This point needs to be made because [Kuhn 62], which many in AI take as a definitive source on scientific revolutions, argues otherwise with respect to the genetic relationship between Newtonian and relativistic mechanics [Kuhn 62, pg. 103]: "Though an out-of-date theory can always be viewed as a special case of its up-to-date successor, it must be transformed for the purpose. And the transformation is one that can be undertaken only with the advantages of hindsight, the explicit guidance of the more recent theory. .... But it [the old theory] could not suffice for the guidance of research." The first sentence is true, but the remaining part of the paragraph is demonstrably false as applied to Einstein's derivation of relativistic mechanics. As is clear from the selection of Einstein's paper in the preceding paragraph, Einstein not only used Newton's theory to guide his search for the proper relativistic laws, he transformed, with foresight, the old (Newtonian) laws to obtain the new (relativistic) laws. Few physicists or philosophers/historians of science currently subscribe to Kuhn's interpretation.

When both the old theory and the new theory comply with the invariance principle, then the difference in the biases will determine the limit point, i.e. the range of phenomena over which they must agree. The following mathematical sketch explains what this limit point must be, when the theories postulate the same number of dimensions. The two biases will share some subgroups in common (e.g. the spatial rotations) and differ in other subgroups (e.g. the subgroup for relative velocity). For the

subgroups that differ, the identity transformations will be the same. Hence the value of the parameter (e.g. relative velocity) that yields the identity transformation must be the limit point (e.g. 0 relative velocity). Furthermore, assuming that the transformations in the differing subgroups are a continuous and smooth function of their parameter(s), and that the functions in the respective theories are smooth and continuous, then the bounding epsilon-delta requirements for a limit are satisfied.

Thus, given a new bias, the new theory must be derived so that it satisfies two constraints: the theory is invariant under the new bias, and the old theory is a limit homomorphism of the new theory. The limit homomorphisms between Newtonian physics and relativistic physics can be defined through the composition of tupling (or projections) that are invertible, with Lorentz transformations applied to the various entities of the theory. Because the Lorentz transformations are also invertible, the composition is invertible. In other words, the limit homomorphism is defined through a standard homomorphism at the limit point, which will be denoted h, and Lorentz transformations denoted g.

The two constraints on the new theory, that it be invariant under the new bias and that it have as a limiting approximation the old theory, can be solved to generate the new theory when the limit homomorphism is invertible. The new theory and the limit homomorphism are derived in tandem. In essence, the transformations in the new bias are used to 'rotate away' from the limit point, as Einstein 'rotated' a description of Newton's equations for an electron initially at rest to reference frames in which it was not at rest. (Here 'rotate' means applying the transformations in the subgroups of the new bias not contained in the old bias, e.g. the Lorentz transformations.)

For the operations of the new theory, these two constraints can often be directly combined as follows:

1. New, unknown operation is covariant wrt new bias:

$$\mathrm{op}(g(x_1, x_2, \ldots, x_n)) = g(\mathrm{op}(x_1, x_2, \ldots, x_n))$$

Equivalently: $\mathrm{op}(x_1, x_2, \ldots, x_n) = g^{-1}(\mathrm{op}(g(x_1, x_2, \ldots, x_n)))$

2. New, unknown operation has limit homomorphism to old operation op′:

$$\lim_{\substack{x_1,\ldots,x_n \to \\ \text{limit point}}} h(\mathrm{op}(x_1, x_2, \ldots, x_n)) = \mathrm{op}'(h(x_1), h(x_2), \ldots, h(x_n))$$

Thus: $\mathrm{op}(x_1, x_2, \ldots, x_n) =$

$$g^{-1}(h^{-1}(\mathrm{op}'(h(g(x_1)), h(g(x_2)), \ldots, h(g(x_n)))))$$

where $g(x_1, x_2, \ldots, x_n) =$ limit point

In words, the new operation is obtained by :

1. Finding a transformation g that takes its arguments to a reference frame where the old operation is valid.

2. Applying the inverse transformation to define the value of the new operation in the original reference frame.

Applying BEGAT to derive the laws of the new theory is a similar two step process: first, a transformation is determined that takes the variables to a reference frame in which the old laws are valid, and then the inverse transformations are symbolically applied to the equations for the old laws.

The algorithm is underconstrained, because of the interaction of the definition of the new (unknown) operation and the definition of the (unknown) homomorphism h. In parts of [Einstein 1905], Einstein assumes that h is the identity, for example in his derivation of the relativistic composition of velocities (described below), and then derives an expression for the new operation. In other parts of [Einstein 1905], he assumes that the old operation and the new operation are identical, for example in his derivation of the relativistic equation of motion. In that derivation he kept the same form as the Newtonian equation (i.e. force = mass * acceleration) and then solved for a relativistic definition of inertial mass, and hence h. To his credit, Einstein recognized that he was making arbitrary choices [Einstein 1905 section 10, after definition of transverse mass]: "With a different definition of force and acceleration we should naturally obtain other values for the masses."

The following illustrates how the BEGAT algorithm works for a simple operation when h is the identity.

Note that when h is the identity: $\mathrm{op}(x_1, x_2, \ldots, x_n) =$

$$g^{-1}(\mathrm{op}'(g(x_1), g(x_2), \ldots, g(x_n))$$

where $g(x_1, x_2, \ldots, x_n) =$ limit point

BEGAT takes as input the definition of the old operation, the list of transformations for the new bias, and a definition of the limit point. For the composition of velocities, the old operation is simply the addition of velocities:

Newton - Compose($v1, v2$) = $v1 + v2$ where:

$v1$ is the velocity of reference frame R_1 w.r.t. R_0

$v2$ is the velocity of object A w.r.t. reference frame R_1

and the output is defined in reference frame R_0

The transformations are the Lorentz transformations derived earlier. The limit point is when R_1 is the same as R_0, i.e. $v1 = 0$. The first part of the reasoning for the BEGAT algorithm is at the meta-level, so it is necessary to understand some aspects of the notation used in the Erlanger program. Variables are represented by an uninterpreted function of the form:

var[event, component, reference-frame]. This form facilitates pattern matching. Transformations have representations both as lists of substitutions and as a meta-level predicate of the form:

Transform[start-frame, end-frame, independent-parameter] The independent parameter for relative velocity has the form: var[end-frame,relvelocity,start-frame]. Thus $v1$ is represented as var[R_1,relvelocity,R_0] and $v2$ as var[A,velocity,R_1].

1. BEGAT first solves for g, the transformation which takes the arguments to the limit point. This transformation maps the reference frame for the output to the reference frame for the limit point. The result is obtained by applying a set of rewrite rules at the meta-level:

Transform[R_0,R_1,var[R_0,relvelocity,R_1]]

This transformation maps reference frame R_0 to reference frame R_1.

2. BEGAT next solves for the value of the variables which are given to the old operation, i.e. $g(v1)$, $g(v2)$. For $g(v1)$ it symbolically solves at the meta-level for:

Apply[Transform[R_0,R_1,var[R_0,relvelocity,R_1]], var[R_1,relvelocity, R_0]],

obtaining var[R_1,relvelocity,R_1], i.e. $g(v1)=0$

For $g(v2)$ it symbolically solves at the meta-level for:

Apply[Transform[R_0,R_1,var[R_0,relvelocity,R_1]], var[A,velocity, R_1]],

obtaining var[A,velocity,R_1], i.e. $g(v2)=v2$ since $v2$ is measured in R_1.

This meta-level reasoning about the application of transformations is necessary when the input variables and the output variables are defined in different reference frames.

3. BEGAT next symbolically applies the old operation to the transformed variables:

$$\text{Newton - compose}(g(v1), g(v2)) = 0 + v2 = v2$$

4. BEGAT finally applies the inverse transformation to this result to obtain the definition for the relativistic operation: Relativistic-compose($v1$,$v2$) =

Apply[Transform[R_1,R_0,var[R_1,relvelocity,R_0]], var[A,velocity, R_1]]

The transformation derived previously for velocities is now applied to var[A,velocity, R_1], yielding the definition of the operator for relativistic composition of velocities: so BEGAT calls DeriveCompositeTransformation with the definition for velocity (i.e.$v = \Delta x/\Delta t$), and the Lorentz Transformations for the components of the definition of velocity - namely the transformations for the x co-ordinate and the time co-ordinate derived earlier. DeriveCompositeTransformation then symbolically applies these transformations to the components of the definition, and then calls Mathematica's SOLVE operation to eliminate the Δx, Δt components from the resulting expression. The result is the same definition as Einstein obtained in section 5 of [Einstein 1905]:

$$\text{Relativistic} - \text{compose}(v1, v2) = (v1 + v2)/(1 + (v1v2)/c^2)$$

Deriving Relativistic Dynamics

This subsection describes how the invariance principle can be used to derive other components of the new theory and the limit homomorphism, illustrated with one derivation of relativistic dynamics. Different background assumptions lead to different limit homomorphisms m and different formulations of the equations for relativistic dynamics. In his original paper, Einstein reformulated the Newtonian equation by measuring the force in the reference frame of the moving object and the inertial mass and acceleration in the reference frame of the observer. (In essence, Einstein did not complete step 2, for reasons too complex to explain here.) This leads to a projection of the Newtonian mass into separate transverse and longitudinal relativistic masses.

A subsequent formulation of relativistic dynamics consistently measures masses, accelerations, momentum and energy in the reference frame of the observer, resulting in a single relativistic mass that varies with the speed of the object. In this formulation the mass of a system is the sum of the masses of its components, and is conserved in elastic collisions. The modern formulation of relativistic dynamics, based on Minkowski's space-time and Einstein's tensor calculus, requires that components that transform into each other be tupled together. Thus because time coordinates transform into spatial coordinates, time and space are tupled into a single 4-vector. Consequently energy and momentum are also tupled together. In this case m maps Newtonian inertial mass to rest mass, and maps Newtonian acceleration and forces to their 4-vector counterparts.

In all three cases the derivation strategy is based directly on the invariance principle and the principle that the non-relativistic theory be a limiting approximation to the relativistic theory. The strategy is to assume that the laws of dynamics are invariant under the Lorentz transformations, and then to solve for the limit homomorphism that makes them invariant. (If it is not possible to consistently solve for the limit homomorphism, then the theory cannot be invariant.) These limit homomorphisms are composed of two maps: first a tupling or projection map from the components of the original theory to components of the new theory ($\mathcal{H}$), and second of Lorentz transformations for components of the new theory($\mathcal{G}$). These two maps are generated by the derivations.

Derivations based on the tensor calculus are the most elegant because the tensor calculus is essentially a syntactic encoding of the invariance principle, as applied to biases defined by groups of linear homogenous transformations. However, an explanation of the group-theoretic basis of the tensor calculus is beyond the scope of this paper. Instead we will describe the justification and strategy that applies to the first two derivations of relativistic dynamics, and then illustrate it with part of Einstein's original derivation. This derivation has been partially simulated in interactive mode with Mathematica 1.2. The justification and steps of this derivation are also the same as that for relativistic electrodynamics; more specifically, the derivation of the Lorentz transformations for electric and magnetic fields.

Recall the definition of the invariance of a theory under a transformation group $\mathcal{G}$, where $\mathcal{NT}$ is the new theory:

$$\text{Invariant}(\mathcal{NT}, \mathcal{G}) \Leftrightarrow \forall (g \in \mathcal{G}) \mathcal{NT} / g \models \mathcal{NT}$$

This is combined with the constraint that the old theory is a limit homomorphism of the new theory, where $\mathcal{LH}$ is the definition of the components of the old theory in terms of the components of the new theory:

$$\mathcal{NT} \cup \mathcal{LH} \models \mathcal{OT}$$

When the limit homomorphism is invertible, we also have:

$\mathcal{OT} \cup IL\mathcal{H} \models \mathcal{NT}$

Because this inverse limit homomorphism can be factored into a tupling/projection map $\mathcal{H}$ and the new bias $\mathcal{G}$, this last constraint can be combined directly with the invariance principle to yield a single constraint between the old theory and the new theory. :

$\forall(g \in \mathcal{G})(\mathcal{OT} \cup \mathcal{H}) / g \models \mathcal{NT}$

By the definition of a limit homomorphism, the old theory is defined with respect to the reference frame for which the limiting value holds (e.g. zero relative velocity). The transformations in $\mathcal{G}$ take the result of applying the tupling/projection map $\mathcal{H}$ to this reference frame and transform it to all other reference frames. The constraint is satisfied when the new theory, defined with respect to any reference frame $\mathcal{R}$, is a consequence of the old theory, the tupling/projection map $\mathcal{H}$, and the transformation g from the reference frame for the old theory to the reference frame $\mathcal{R}$. We will now show how this constraint can be used to derive the new theory, illustrated with Einstein's derivation of relativistic dynamics.

In all derivations of relativistic dynamics, it is assumed that the new equation has the same form as the Newtonian equation, but that the definition of the components might be different; according to $\mathcal{H}$ and $\mathcal{G}$. Thus if $\mathcal{H}$ and $\mathcal{G}$ are partially known, say $\mathcal{H}'$ and $\mathcal{G}'$ are defined for some of the components, then the remaining parts of $\mathcal{H}$ and $\mathcal{G}$ are derived by setting up the following unified constraint and solving for the remaining parts of the limit homomorphism:

$\forall(g \in \mathcal{G})(\mathcal{OT} \cup \mathcal{H}') / g' \models \mathcal{OT}'$

where $\mathcal{OT}'$ has the same form as the Newtonian theory but with new variables which are functions of corresponding variables in $\mathcal{OT}$ and the parameters of the transformation group $\mathcal{G}$.

Einstein's derivation of relativistic dynamics proceeded as follows. First, the old theory ($\mathcal{OT}$) was Newton's dynamics relating a particle's inertial mass, acceleration, and the force exerted upon the particle (Einstein considered the case where the force was exerted by an electric field with a particle of charge ε). This law is valid in the reference frame of the particle:

$$\mathcal{OT} \equiv m\frac{d^2x}{dt^2} = \varepsilon E_x \qquad m\frac{d^2y}{dt^2} = \varepsilon E_y \qquad m\frac{d^2z}{dt^2} = \varepsilon E_z$$

Through previous derivations, $\mathcal{H}'$ and $\mathcal{G}'$ were known for space, time, and electromagnetic fields; though Einstein did not use the transformations for the electromagnetic field. The map $\mathcal{H}'$ for space and time was the identity, while $\mathcal{G}'$ was the Lorentz transformation equations for space and time generated by DeriveNewBias. (A different background assumption where $\mathcal{H}'$ tuples space and time into a single 4-vector would yield the tensor formulation of relativistic dynamics). Thus Einstein needed to solve for the relativistic definition of inertial mass as a function of the non-relativistic mass and the parameter of the Lorentz transformation group; namely, the relative velocity between reference frames. Because the relative velocity is a vector quantity with x,y,z components; the definition of the inertial mass is also set up with x,y,z components. These components of the inertial mass might later be identified. In the following, v is the relative velocity between the reference frame of the particle and an observer moving in the positive x direction, and β is a term defined with respect to the magnitude of v: $\beta = 1/\sqrt{1 - v^2/c^2}$. The unprimed variables are in the reference frame of the particle, while the primed variables are in the reference frame of the observer. The constraint relating Newton's dynamics, the Lorentz transformations, and relativistic dynamics is instantiated from the unified constraint above:

$$\forall v \ \mathcal{OT} \cup \left\{ \begin{array}{c} x = \beta(x' + vt') \\ y = y' \\ z = z' \\ t = \beta\left(t' + x'v^2/c^2\right) \end{array} \right\} \vdash \left\{ \begin{array}{l} m'_x(m,v)\dfrac{d^2x'}{dt'^2} = \varepsilon E_x \\ m'_y(m,v)\dfrac{d^2y'}{dt'^2} = \varepsilon E_y \\ m'_z(m,v)\dfrac{d^2z'}{dt'^2} = \varepsilon E_z \end{array} \right.$$

Note that Einstein defines the force in the reference frame of the particle, even on the right hand side. The equations for Newton's dynamics are then partially transformed into the reference frame of the observer by applying the Lorentz transformations, yielding a simplified constraint:

$$\left. \begin{array}{l} m\beta^3\dfrac{d^2x'}{dt'^2} = \varepsilon E_x \\ m\beta^2\dfrac{d^2y'}{dt'^2} = \varepsilon E_y \\ m\beta^2\dfrac{d^2z'}{dt'^2} = \varepsilon E_z \end{array} \right\} \vdash \left\{ \begin{array}{l} m'_x(m,v)\dfrac{d^2x'}{dt'^2} = \varepsilon E_x \\ m'_y(m,v)\dfrac{d^2y'}{dt'^2} = \varepsilon E_y \\ m'_z(m,v)\dfrac{d^2z'}{dt'^2} = \varepsilon E_z \end{array} \right.$$

This constraint is then solved for definitions of the relativistic inertial mass in terms of the Newtonian inertial mass and the parameter between the reference frame of the particle and the observer. Solving this constraint is a simple directed inference problem [Smith 91]; reasoning backwards from the right hand side a match is derived between the variables for the relativistic inertial mass and terms on the left hand side:

$m'_x(m,v) = m\beta^3$

$m'_y(m,v) = m\beta^2$

$m'_z(m,v) = m\beta^2$

The definitions for the y and z components of the inertial mass are identical, so they can be combined into a single `transverse' inertial mass. In alternative derivations of relativistic dynamics, all the components of the inertial mass are identical.

While the particular derivation tactics currently implemented or undergoing implementation in the BEGAT algorithm might not be directly applicable to other types of biases, it is likely that analogues can be found. Research

toward generalizing BEGAT is described after a review of related work.

Related Research

Within AI, this research is related to scientific discovery and theory formation [Shrager and Langley 90], qualitative physics [Weld and de Kleer 90], change of bias in machine learning [Benjamin 90a], and use of group theory [Benjamin 90b]. The research in this paper appears to be the first addressing the automated rediscovery of scientific revolutions of twentieth century theoretical physics. Most of the work in scientific theory formation has been on incremental theory revision (normal science). Previous research on scientific revolutions includes conceptual and qualitative accounts of the geological revolution in plate tectonics [Thagard and Nowak 90] and the chemical revolution of the oxygen theory [O'Rork, Morris, and Schulenburg 90]. Recently, [Thagard 92] has addressed automating the comparison of competing theories, and applied it to comparing Einstein's relativity theories with competing theories.

The notions of approximation within qualitative physics are closely related to limit homomorphisms. The well known calculii for qualitative physics reasoning usually include some sort of homomorphism from the reals [Forbus 84] [Kuipers 86]. The use of limits (fitting parameters) to define approximation relations between models is described in [Weld 89]. Within machine learning, research on declarative representations and reasoning about bias is most important, see the collection of papers in [Benjamin 90a]. The research described in this paper is one approach to addressing an open problem presented in [Dietterich 91]: analytically comparing biases. The declarative bias used in theoretical physics is group theory. A good collection of papers, many of which focus on the use of group theory in AI reasoning and problem solving, is in the workshop proceedings [Benjamin 90b].

The mathematical model and the research strategy presented in this paper are consistent with the physics literature. References accessible to the layman include [Zee 86] and [Davies and Brown 88]. With respect to that literature the chief innovations of this paper are the result of focusing on the structure of derivations with the aim of formalizing them. This focus is peculiar to AI; to the best of my knowledge it has not been addressed before. The closest previous works may be various pedagogical explanations found in textbooks such as [Skinner 82], [Taylor and Wheeler 66], and [French 68].

Conclusion: Toward Primal-Dual Learning

A hypothesis of this research is that Einstein's strategy for mutually bootstrapping between a space of biases and a space of theories has wider applicability than theoretical physics. Below we generalize the structural relationships of the invariance principle which enabled the computational steps of Einstein's derivation to succeed. We conjecture that there is a class of *primal-dual* learning algorithms based on this structure that have similar computational properties to primal-dual optimization algorithms that incrementally converge on an optimal value by alternating updates between a primal space and a dual space.

Let $\mathcal{B}$ be a set of biases with ordering relation $\lhd$ that forms a lattice. Let $\mathcal{T}$ be a set of theories with ordering relation $\prec$ that forms a lattice. Let C be a consistency relation on $\mathcal{B}\times\mathcal{T}$ such that:

$$C(b,t) \text{ and } b' \lhd b \Rightarrow C(b',t)$$
$$C(b,t) \text{ and } t' \prec t \Rightarrow C(b,t')$$

This definition is the essential property for a well-structured bias space: As a bias is strengthened, the set of theories it is consistent with decreases; as a theory is strengthened, the biases it is consistent with decreases. Hence C defines a contravariant relation between the ordering on biases and the ordering on theories.

Let $\mathcal{U}$ be the most general bias function from $\mathcal{T} \rightarrow \mathcal{B}$ such that $C(\mathcal{U}(t),t)$ and $\neg C(b,t) \Rightarrow b \lhd \mathcal{U}(t)$. Let $\mathcal{D}$ be a function from $\mathcal{B}\times\mathcal{T} \rightarrow \mathcal{B}$ such that $\mathcal{D}(b,t) = b \wedge \mathcal{U}(t)$, where $\wedge$ is the lattice meet operation.

$\mathcal{D}$ is the DeriveNewBias function, which takes an upper bound on a bias and filters it with a (new) theory or observation to obtain a weaker bias. (For some applications of primal-dual learning, $\mathcal{D}$ should take a lower bound on a bias and filter it with a new theory or observation to obtain a stronger bias.) $\mathcal{D}$ is well-defined whenever $\mathcal{B}$, $\mathcal{T}$, and C have the properties described above. However, depending on the type of bias, it might or might not be computable. If it is computable, then it defines the bootstrapping from the theory space to the bias space when an inconsistency is detected.

The bootstrapping of BEGAT from a new bias to a new theory that has a limiting approximation to the old theory requires two capabilities. First, given the old bias and the new sibling bias, the restriction of the old theory to those instances compatible with the new bias must be defined and computable. Second, given this restriction, its generalization by the new bias must also be defined and computable.

As an example of BEGAT with a different type of bias, consider the problem of learning to predict a person's native language from attributes available in a data base. We will assume that one's native language is the same as the language spoken by one's mother, but that the mother's language is not in the data base. A declarative representation for biases that includes functional dependencies was presented in [Davies and Russell 87] and subsequent work. Let the original bias be that the native language is a function of the birth place. This bias would likely be consistent with data from Europe, but might be inconsistent with the data from the U.S. because of its large immigrant population. Assume that a function $\mathcal{D}$ derives a new bias where the native language is a function of the mother's place of origin. The following limit

homomorphism formalizes the intersection of the original bias and the new bias:

$$\lim_{\#immigrants \to 0} \text{mother's} - \text{origin}(x) = \text{birth} - \text{place}(x)$$

The restriction of the original theory to concepts derived from the limiting value (e.g. non-immigrant data) is compatible with this new bias. Furthermore, the concepts learned from this restricted set can be transferred directly to the new theory by substituting the value of the birth place attribute into the value for the mother's place of origin.

Future research will explore the theory and application of primal-dual learning to theoretical physics and other domains. Given the spectacular progress of twentieth century physics, based on the legacy of Einstein's research strategy, the computational advantages of machine learning algorithms using this strategy might be considerable.

Acknowledgments

Thanks to Robert Holte, Laura Jones, Hugh Lowry, Thomas Pressburger, and Jeffrey Van Baalen for their many helpful comments on improving this paper.

References

Aharoni, J. 1965. *The Special Theory of Relativity*. New York: Dover.

Benjamin, P. editor. 1990a. *Change of Representation and Inductive Bias*. Boston: Kluwer.

Benjamin, P. editor. 1990b. *Workshop Proceedings for Algebraic Approaches to Problem Solving and Perception*. June 1990.

Davies, P.C.W. and J. Brown 1988. *Superstrings: A Theory of Everything?* Cambridge University Press.

Davies, T. R. and Russell, S.J. 1987. A Logical Approach to Reasoning by Analogy. In IJCAI-87.

Dietterich, T. 1991. Invited Talk on Machine Learning at AAAI91, Los Angeles, CA.

Einstein, A. 1905. On the Electrodynamics of Moving Bodies. In *The Principle of Relativity, A Collection of Original Memoirs on the Special and General Theory of Relativity*, contributors H.A. Lorentz, A. Einstein, H. Minkowski, and H. Weyl. NewYork: Dover (1952).

Einstein, A. 1916. *Relativity: The Special and General Theory. A clear explanation that anyone can understand*. New York: Crown.

Ellman, T. editor. 1990. *AAAI90 Workshop proceedings on Automatic Generation of Abstractions and Approximations*. Boston, MA.

Ellman, T. editor. 1992. *AAAI902Workshop proceedings on Automatic Generation of Abstractions and Approximations*. San Jose, CA.

Forbus K.D. 1984. Qualitative Process Theory. *Artificial Intelligence*(24):85-168.

French, A. P. 1968. *Special Relativity*. New York: Norton.

Friedman, M. 1983. *Foundations of Space-Time Theories: Relativistic Physics and Philosophy of Science*. New Jersey: Princeton University Press.

Kuhn, T. S. 1962. *The Structure of Scientific Revolutions*. Chicago: University of Chicago Press.

Kuipers, B. 1986. Qualitative Simulation. *Artificial Intelligence(29): 289-388*.

Minkowski, H. 1908. Space and Time. In *The Principle of Relativity, A Collection of Original Memoirs on the Special and General Theory of Relativity*, contributors H.A. Lorentz, A. Einstein, H. Minkowski, and H. Weyl. NewYork: Dover (1952).

O'Rorke, P., Morris, S. and D. Schulenburg 1990. Theory Formation by Abduction: A Case Study Based on the Chemical Revolution. In *Computational Models of Scientific Discovery and Theory Formation*, editors J. Shrager and P. Langley.

Shrager, J. and Langley, P. eds. 1990. *Computational Models of Scientific Discovery and Theory Formation*, editors. San Mateo: Morgan Kaufmann.

Skinner, R. 1982. *Relativity for Scientists and Engineers*. New York: Dover.

Smith, D.R. 1982. Derived Preconditions and Their Use in Program Synthesis. In *Sixth Conference on Automated Deduction*, ed. D.W. Loveland, 172-193. Lecture Notes in Computer Science, volume 138. Berlin: Springer-Verlag.

Taylor, E.F. and Wheeler, J.A. 1966. *Spacetime Physics*. San Francisco: Freeman.

Thagard, P. and G. Nowak 1990. The Conceptual Structure of the Geological Revolution. In *Computational Models of Scientific Discovery and Theory Formation*, editors J. Shrager and P. Langley.

Thagard, P. 1992. *Conceptual Refvolutions*. Princeton, New Jersey: Princeton University Press.

Valiant, L.G. 1984. A Theory of the Learnable. In *CACM* (27):1134-1142.

Weld,D.S. 1989. *Automated Model Swithching: Discrepency Driven Selection of Approximation Reformulations*. University of Washington Computer Science Department Technical Report 89-08-01.

Weld, D.S. and de Kleer, J. eds. 1990. *Readings in Qualitative Reasoning about Physical Systems*. editors. San Mateo, CA: Morgan Kaufmann.

Weld, D.S. 1992. *Reasoning about Model Accuracy*. University of Washington Computer Science Department Technical Report To appear in Artificial Intelligence.

Zee, A. 1986. Fearful Symmetry: The Search for Beauty in Modern Physics. New York: Macmillan.

Becoming Reactive By Concretization

Armand Prieditis and Bhaskar Janakiraman
Department of Computer Science
University of California
Davis, CA 95616

Abstract

One way to build a reactive system is to construct an action table indexed by the current situation or stimulus. The action table describes what course of action to pursue for each situation or stimulus. This paper describes an incremental approach to constructing the action table through achieving goals with a hierarchical search system. These hierarchies are generated with transformations called *concretizations*, which add constraints to a problem and which can reduce the search space. The basic idea is that an action for a state is looked up in the action table and executed whenever the action table has an entry for that state; otherwise, a path is found to the nearest (cost-wise in a graph with cost-weighted arcs) state that has a mappring from a state in the next highest hierarchy. For each state along the solution path, the successor state in the path is cached in the action table entry for that state. Without caching, the hierarchical search system can logarithmically reduce search. When the table is complete the system no longer searches: it simply reacts by proceeding to the state listed in the table for each state. Since the cached information is specific only to the nearest state in the next highest hierarchy and not the goal, inter-goal transfer of reactivity is possible. To illustrate our approach, we show how an implemented hierarchical search system can completely reactive.

1 Introduction and Motivation

Intelligent interaction with the world can be viewed as a combination of planning to achieve some goal and of reaction to external stimuli in the course of executing a plan. A pure planning system produces a complete plan of actions before executing it [4, 8, 3, 5]. In contrast, a pure reactive system quickly selects and executes a single action based on an external stimulus [2, 1, 6]. Planning systems appear to work well when the predictability of the world is precisely captured in the planner's actions, whereas reactive systems appear to work well in worlds that are fraught with uncertainty or unpredictability—where plans have little chance of succeeding in their entirety, where the ability to plan to completion is not a virtue. This paper describes how a planning system can incrementally become more reactive through interaction with its world, By becoming more reactive, the system reduces its decision-making time.

Previous approaches to building reactive systems from non-reactive ones include compilation and learning from examples. Firby [5] and Rosenschein [13] show how to compile high-level input descriptions of actions and goals into reactive systems. Similarly, Rosenschein and Kaelbling describe a technique to compile constraint expressions into directly executable circuits for a robotic control system [14]. Mitchell uses Explanation-Based Learning to incrementally learn the general conditions under which a particular action, which helps achieve a particular planning goal, should be applied [10]. If the conditions are matched, the same action is applied—irrespective of the system's current goal. The advantage of learning over compiling is that examples focus on those parts of the environment with which an intelligent agent actually interacts; only those actions that are relevant to that interaction are compiled for reactivity.

The problem with the Explanation-Based Learning approach is that multiple goals can lead to multiple action suggestions for the same state, which results in deliberation as to which action to apply and therefore less reactivity. This anomaly is commonly called the *wandering bottleneck* problem in the machine learning literature: as a result of eliminating one time bottleneck (e.g. time taken to react) another one unexpectedly arises (e.g. time taken to decide how to react). More precisely, in a problem with n problem-solving states, each state can have as many as n possible action suggestions since there can be as many as n goals from which the action suggestions are learned. Moreover, to store such a network of states and actions can require as much as $O(n^2 \log n)$ space over n goals and n states, since $O(\log n)$ space is required to store each action suggestion. If n is an exponential of problem size, then this approach is generally not feasible.

This paper describes a technique to avoid the wandering bottleneck problem by hierarchically organizing the state space such that at most one action is learned for each state. As a side-benefit, this hierarchical organization reduces the worst-case space requirements by a factor of n.

The rest of this paper is organized as follows. Section 2 defines the notion of a concretization and derives several important properties of concretizations in

search. Section 3 describes our approach to becoming reactive by concretization. Section 4 presents experimental results of applying our approach. Finally, Section 5 summarizes the conclusions of this work and discusses a few promising avenues of future research.

2 Concretizations

Intuitively, a concretization of a problem is one that has added constraints. The importance of these added constraints is that they reduce the branching factor during search. To formalize this intuitive notion requires a definition of search. The definition that we will assume is standard in the AI literature [11]. A search problem can be thought of as consisting of a graph of nodes, which represent states, and directed arcs that represent the application of an operator. These arcs are typically weighted to represent the cost of applying the corresponding operator. Search can be thought of as finding a finite path in the graph from a node representing a given initial state to a node representing a given goal state. The graph can be specified explicitly or implicitly. In an explicit specification, the nodes and arcs with associated costs might be supplied in a table that includes every node in the graph and a list of its successors and the costs of associated arcs. This information might also be specified by a matrix that stores the costs of associated arcs for every pair of nodes (an infinite cost arc represents the absence of an arc). In an implicit specification, only that portion of the graph that is sufficient to include a goal node is made explicit by applying operators using a search algorithm such as A^* [11]. For example, in the Eight Puzzle problem, the set of states consists of all tile permutations and operators only allow swapping the blank with an adjacent tile (i.e. the cost function on a pair of states returns 1 if one state is reachable from the other by swapping the blank with an adjacent tile, and ∞ otherwise). The goal state might specify that the tiles are in a particular order.

More formally, let a search problem be a 3-tuple $\langle S, c \rangle$, where S is a set of states describing situations of the world; $c : S \times S \rightarrow \Re$ is a positive cost function that represents the cost of applying the corresponding action from one state to another; and $G \subseteq S$ is a set of goal states. An *instance* of a search problem includes a 2–tuple $\langle i, g \rangle$ where $i \in S$ is the initial state and $g \in S$ is the goal state (for simplicity, we assume that there is only one goal state). The objective is to find a finite length finite cost path from i to g.

A problem $\langle S', c' \rangle$ is a *concretization* of another problem $\langle S, c \rangle$ with respect to $\phi : S' \rightarrow S$ iff ϕ *reduces cost:* $(\forall s', t' \in S') c(\phi(s'), \phi(t')) \leq c(s', t')$.

For example, Figure 1 shows a concretization of the Towers of Hanoi problem. The original problem is composed of operators that stack smaller disks on top of larger disks from pin to pin; states are simply disks stacked in increasing size on various pins. The initial and goal states for a typical three disk instance of the Towers of Hanoi problem are also shown in the figure. If the disks are numbered from top to bottom and then the operators are constrained such that they never place an odd-numbered disk on an even-numbered disk and vice versa, then this new problem is concretization of the original problem with respect to a mapping function that ignores disk parity. The reason is because the cost is reduced: operators apply more often in the original problem. Notice that any solution in the concrete space is guaranteed to be a solution in the original space because the concretized problem is more restricted. Since the branching factor will be lower for the concretized problem, solution generation will be more efficient (though slightly longer solutions will generally result). This property, which we call *solution-soundness*, is perhaps most powerful when a problem can be concretized into one for which an efficient solution generator exists. Any solution to the concretized problem can then be directly mapped onto a solution to the original problem. For example, a Blocks World problem with three table locations can be concretized into a Towers of Hanoi problem, which has an associated divide-and-conquer algorithm, by assigning a "size" to each block (say, small to large for each block on every stack, consistent in the initial and goal states). Any solution to the corresponding Towers of Hanoi problem can be mapped onto a solution to the original problem simply by ignoring size.

Tenenberg describes a similar property, which he calls the *downward solution property*, in the context of planning with a certain type of operator representation [16]. In his terminology, a transformed problem has the downward solution property if every solution in the transformed space can be mapped onto one for the original problem. Solution soundness is a generalization of the downward solution property since it does not depend on specific operator representations.

Despite the solution-soundness property of concretizations, a solvable problem in the original space

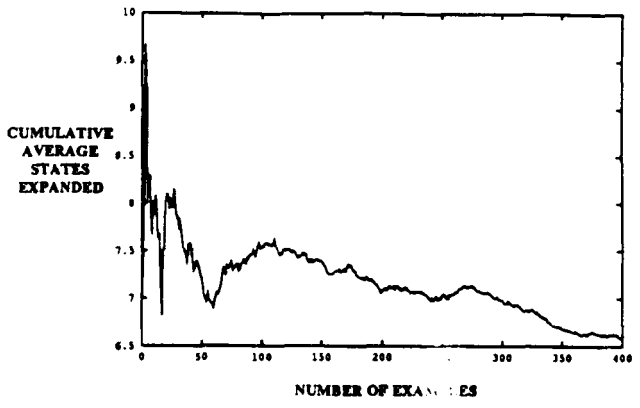


Figure 4 Becoming Reactive Through Interaction with the World

only those states that are most frequently encountered or apply learning techniques to reduce table size. In particular, we are currently investigating applying our ideas to a less artificial problem (a robot routing task), which includes explicitly specified operators with inputs from external sensors such as in a robot routing task. It might be possible to apply Explanation-Based or inductive learning to learn the *class* of states that lead to the nearest state in the next highest hierarchy.

Another problem is that constructing concretization hierarchies is generally a difficult problem. However, a catalog of problem transformations such as those of Absolver II [12] might prove helpful. Another method might be to use clustering algorithms to group similar states into equivalence classes. Problem-solving performance with more meaningful groupings—those that exploit the structure of the search graph and similarity of states—should be improved over the results we obtained with random hierarchical groupings.

Ultimately, we would like to test our ideas in a dynamic world where an intelligent agent's plans to achieve goals are continually thwarted by unforeseen events to which the agent has to react immediately, recover, and then proceed towards achieving the goal. We believe that a hierarchical learning system of the sort described here may be especially suited for such worlds. We are currently modeling a dynamic world and testing this hypothesis.

References

[1] P. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proceedings AAAI-87*, pages 268–272. Seattle, WA, 1987. American Association for Artificial Intelligence.

[2] R.A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1), March 1986.

[3] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–378, 1987.

[4] R. Fikes, P. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4):251–288, 1972. Also in *Readings in Artificial Intelligence*, Webber, B. L. and Nilsson, N. J., (Eds.).

[5] J. Firby. *Adaptive Execution in Complex Dynamic Worlds*. PhD thesis, Yale University, 1989.

[6] L Kaelbling. An architecture for intelligent reactive systems. In M. Georgeff and A. Lansky, editors, *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*. Morgan Kaufmann, 1986.

[7] L. Kleinrock and F Kamoun. Hierarchical routing for large networks. *Computer Networks*, 1:155–174, 1977.

[8] R. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(2):97–109, 1985.

[9] R. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33(1):65–88, 1987.

[10] T. Mitchell. Becoming increasingly reactive. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, July 1990. American Association for Artificial Intelligence.

[11] N. J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, Palo Alto, CA, 1980.

[12] A. Prieditis. Machine discovery of effective admissible heuristics. *Machine Learning*, October 1992. To Appear.

[13] S. Rosenschein. Formal theories of knowledge in ai and robotics. *New Generation Computing*, (3):345–357, 1985.

[14] S. Rosenschein and L. Kaelbling. The synthesis of digital machines with provable epistemic properties. In *Theoretical Aspects of Reasoning about Knowledge*, pages 83–98, San Mateo, CA, 1988. Morgan Kaufmann.

[15] E. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[16] J. Tenenberg. *Abstraction in Planning*. PhD thesis, University of Rochester, 1988.

Reformulating Constraints for Compilability and Efficiency

Chris Tong, Wesley Braudaway, Sunil Mohan, and Kerstin Voigt
Department of Computer Science
Rutgers University
New Brunswick, NJ 08903

Abstract

KBSDE is a knowledge compiler that uses a classification-based approach to map solution constraints in a task specification onto particular search algorithm components that will be responsible for satisfying those constraints (e.g., local constraints are incorporated in generators; global constraints are incorporated in either testers or hillclimbing patchers). Associated with each type of search algorithm component is a subcompiler that specializes in mapping constraints into components of that type. Each of these subcompilers in turn uses a classification-based approach, matching a constraint passed to it against one of several schemas, and applying a compilation technique associated with that schema.

While much progress has occurred in our research since we first laid out our classification-based approach [Ton91], we focus in this paper on our reformulation research. Two important reformulation issues that arise out of the choice of a schema-based approach are:

- *Compilability.* Can a constraint that does not directly match any of a particular subcompiler's schemas be reformulated into one that does?
- *Efficiency.* If the efficiency of the compiled search algorithm depends on the compiler's performance, and the compiler's performance depends on the form in which the constraint was expressed, can we find forms for constraints which compile better, or reformulate constraints whose forms can be recognized as ones that compile poorly?

In this paper, we describe a set of techniques we are developing for partially addressing these issues.

Introduction

Because we have described KBSDE more extensively elsewhere [Ton91], our introduction to the basic ideas behind KBSDE will be relatively brief.

Rooms lengths must be at least minValue1.	(MINL)
Rooms widths must be at least minValue2.	(MINW)
Rooms must be inside the floorplan.	(INS)
Rooms must be adjacent to the floorplan boundary.	(ADJ)
Rooms must not overlap.	(NONOV)
The rooms must completely fill the floorplan space.	(FILL)

Figure 1: Constraints on house floorplans

Task specifications. KBSDE accepts task specifications that can be expressed in the form:

Synthesize($i : I, o : O$) | $P(o)$

where i is the input defining a particular problem, o is a candidate solution, O (the type of the object o to be synthesized) defines the space of candidate solutions, and P(o) is a predicate on o that must be satisfied. P(o) is expressed as a conjunction of simpler constraints.

Many design tasks can be specified in this manner. For example, the specification for a simple house floorplanning task might look like:

Synthesize($< l : houseLength, w : houseWidth, n : NbrRooms >, fp : Floorplan$) | $AcceptableFloorplan(fp)$

where $AcceptableFloorplan(fp)$ is the conjunction of the constraints listed in Figure 1.

Algorithm descriptions. KBSDE's top-level classification partitions the conjoined constraints in $P(o)$ into (mutually exclusive) subsets $P_i(o)$ of constraints, where each subset is to be satisfied by a distinct algorithm component (either a constrained generator, a tester, or a hillclimbing patcher). Prototype heuristics for assigning constraints to algorithm components are discussed in [Ton91].

One example of a partitioning of the constraints in Figure 1 among a set of algorithm components is:

Synthesize($< l : HLength, w : HWidth,$
$n : NbrRooms >, fp : Floorplan$)
Generate($s \mid MINL(fp) \wedge MINW(fp)$
$\wedge INS(fp) \wedge ADJ(fp)$);
if Test($fp \mid NONOV(fp) \wedge FILL(fp)$) fails
then Patch($fp \mid NONOV(fp) \wedge FILL(fp)$);
Return(fp).

The intended semantics of this syntax is: generate an s; if the tested constraints fail, then try to patch; if patching fails, chronologically backtrack to the generator. Later references to Tests in this paper are intended to have the same semantics with respect to test failure and backtracking to preceding generators.

Algorithms themselves can be partitioned across several *levels of abstraction*. For example,

Synthesize($< l : HLength, w : HWidth,$
$n : NbrRooms >, fp : Floorplan$)
Generate($< a_1 : area(Room), ...,$
$a_n : area(Room) >$);
Test($< a_1, ..., a_n > | \; l * w = a_1 + a_2 + ... + a_n$);

high level

---

low level

For i = 1 **to** n **do**
Generate($r_i : Room \mid MINL(r_i)$
$\wedge MINW(r_i) \wedge INS(r_i)$
$\wedge ADJ(r_i) \wedge Area(p_i) = a_i$);
$rooms(fp) \leftarrow < r_1, ..., r_n >$;
if Test($fp \mid NONOV(fp) \wedge FILL(fp)$) fails
then Patch($fp \mid NONOV(fp) \wedge FILL(fp)$);
Return(fp).

Constraints are thus partitioned across levels as well as across the algorithm components within each level. In addition, inter-level constraints (such as $Area(p_i) = a_i$) are dynamically posted to ensure that a solution generated at the next level down is a *refinement* of the solution at the current level.

Reformulating constraints for incorporation into generators

The RICK subcompiler (the subject of Wes Braudaway's Ph.D. work [Bra91b, Bra91a]) of KBSDE specializes in incorporating local solution constraints c into generators of all instances s of a given type T. The result is a *constrained generator*, whose computed range of values is guaranteed to include only those instances of T that satisfy c; this is typically accomplished by either changing the lower or upper bounds of the range, or pruning out inconsistent values and caching the remainder. Note that T can be non-numerical, and hierarchical in structure (e.g., a rectangular floorplan is composed of rectangular rooms; each room is defined in terms of 4 parameters: $< x, y, l, w >$).

Compilability

The structure mismatch problem. The generator structure of a an algorithm is its internal organization of generator components. Different generator structures can be constructed for the same task. Some generator structures do not allow the incorporation of all constraints. We refer to this as the *structure mismatch problem.*

The structure mismatch problem is actually a family of problems, as illustrated in Figure 2, which indicates all the activities involved in RICK's process of designing a constrained generator. Each such decision ((a) through (h)) can be "bollixed" by its own unique structure mismatch problem. For example, decision (a) - partitioning requirements - takes a set of requirements R(s) and decides which will be treated as local, type-defining constraints T(s), which will be considered as semi-local constraints C(s) on interfacing parts of s, and which will be considered global constraints P(s); which constraints can be treated as type-defining depend on the known datatypes.

The RICK subcompiler *avoids* the structure mismatch problems associated with decision (e) (the choice of low-level object structure), decision (g) (the choice of composition architecture), and decision (h) (the choice of control flow) by using a least commitment approach to *top-down refinement* of the generators that is constrained by the constraints to be incorporated [Bra91b].

Thus for theses decisions, RICK avoids the need for a reformulation-based approach to the structure mismatch problem. However, RICK does require reformulation for a particular special case of decision e) which we now describe.

Reformulation to eliminate terminological mismatch between constraint and generator. If a constraint c refers to an object obj that is semantically a dynamically generated part of solution o (e.g., the *points* inside rectangle R) but that does not appear syntactically as a part or parameter of o (according to the given type definition T), then c cannot be directly incorporated into the generator of all instances of type T (since the hierarchical structure of the generator procedure is directly lifted from the syntactic structure of T). Incorporation is enabled by using reformulation techniques that reexpress c solely in terms of the defined parts and parameters of T.

For example, constraint INS, "Rooms must be inside the floorplan", might originally be expressed as "If a point is inside a room, than that point is also inside the floorplan":

$\forall R, P[[Room(R) \wedge Point(P) \wedge x(sw(R)) \leq x(P)$
$\leq x(ne(R)) \wedge y(sw(R)) \leq y(P) \leq y(ne(R))] \rightarrow$
$[x(sw(floorplan)) \leq x(P) \leq x(ne(floorplan))\wedge$

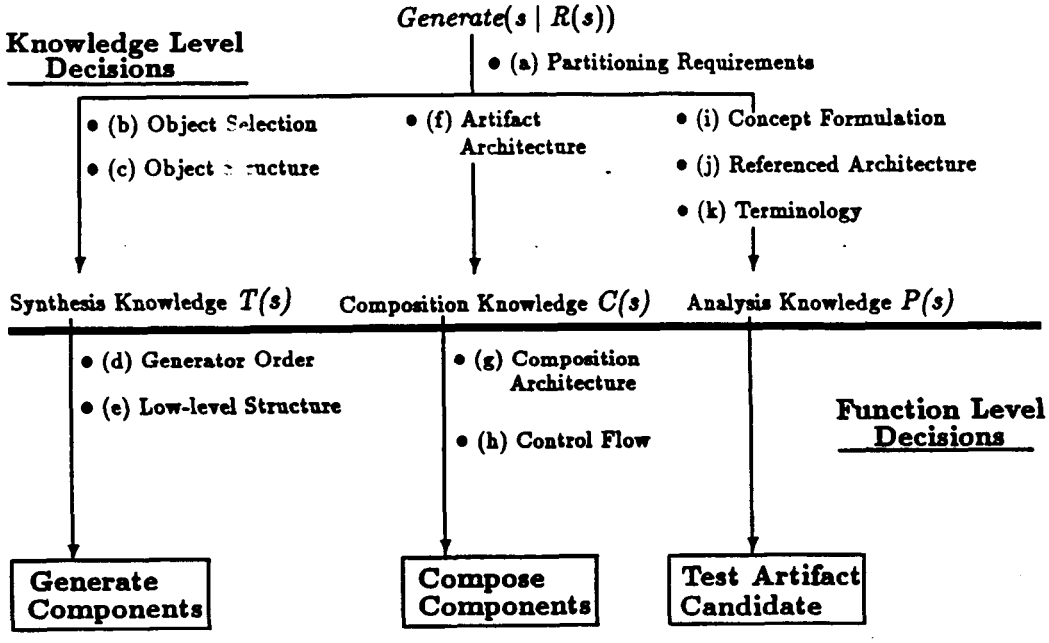


Figure 2: Design decisions defining a family of structure mismatch problems

$y(sw(floorplan)) \leq y(P) \leq y(ne(floorplan))]]$

The problem is that T, the part decomposition for objects of type Floorplan, does not include points P in the interior of a room. The RICK system uses the transitivity of the "$\leq$" relation to *hypothesize* a plausible reformulation of the above constraint that does not refer to points P, and instead, simply constrains the extreme points of room R:

$\forall R[Room(R) \rightarrow$
$[x(sw(floorplan)) \leq x(sw(R)) \leq x(ne(floorplan)) \wedge$
$y(sw(floorplan)) \leq y(sw(R)) \leq y(ne(floorplan)) \wedge$
$x(sw(floorplan)) \leq x(ne(R)) \leq x(ne(floorplan)) \wedge$
$y(sw(floorplan)) \leq y(ne(R)) \leq y(ne(floorplan))]]$

RICK then uses a standard theorem-proving technique to *verify* that this hypothesized reformulation is a necessary condition for the original constraint.

Reformulation by eliciting simplifying assumptions. Because RICK uses the simplex method to check the consistency of a set of constraints (a necessary step along the way to constructing a constrained generator satisfying those constraints), all such constraints must ultimately be expressible in a linear algebraic form in order for compilation to proceed. Sometimes, however, constraints that could be reformulated as linear constraints depend for their reformulation upon knowledge not available in RICK's knowledge base.

For example, in our house floorplanning example, floorplans and rooms are defined to be rectangular. Rectangles in general need not be aligned horizontally or vertically in the Cartesian plane; thus, the type definition for a general rectangle will have associated the nonlinear constraint:

$[x(nw(R)) - x(sw(R))] * [x(se(R)) - x(sw(R))] =$
$[y(nw(R)) - y(sw(R))] * [y(se(R)) - y(sw(R))]$

However, were RICK to know that:

$x(nw(R)) = x(sw(R))$

i.e., we can consider the rectangle to be vertically aligned with the y axis, then because (non-degenerate) rectangles also have the associated constraint:

$y(sw(R)) < y(nw(R))$

the constraint could be reduced to the linear constraint:

$y(se(R)) = y(sw(R))$

i.e., "the rectangle is horizontally aligned with the x axis."

If at any point, compilation comes to a halt because the only constraints left to compile are nonlinear, RICK consults the user, by presenting a list of plausible *simplifying assumptions*. These simplifying assumptions are generated by heuristics that examine a nonlinear constraint and consider what would have to be true in order for it to be reducible to a linear constraint. Thus if $(x - y)$ appears in a product, "$x = y$" would simplify the constraint if it were true; if xy appears in a sum, $x = 1/y$ would simplify the constraint.

The generation (and selection/verification by the user) of such simplifying assumptions is intended to mimic the form of mathematical reasoning, "Without loss of generality, let us assume ...". The user is involved in this process because, in general, actual verification of such simplifying assumptions requires knowledge that is not available in the system's knowledge base.

Efficiency improvement

RICK's task is to construct, for a given constraint c(o), where o is of type T, a constrained generator of objects of type T that are guaranteed to satisfy c. Thus, no matter how it chooses to represent solutions or incorporate the constraint, if RICK fully incorporates c into the generator, the set of solutions generated will always be the same. Since RICK does not reason about "low-level" issues such as choice of data structure for solutions, the primary issue regarding efficiency is whether the constructed constrained generator - which sequentially produces all the members of the set $\{o \mid c(o)\}$ - is producing duplicate candidate solutions in that sequence.

One reformulation technique used by RICK to help reduce the construction of redundant solutions is based on knowing that if RICK is passed a constraint of the form:

$$\forall x, y[P \rightarrow x = y]$$

it will "operationalize" this by constructing generators for objects x and y, generate one object first (say, x), and then construct y to be an exact copy of x. RICK avoids this undesirable behavior by looking for such constraints, forming their logical contrapositive (except for the type-defining terms), and then reexpressing the constraint in a canonical form.

For example, the NONOV constraint, "Rooms must not overlap", might originally be expressed as:

$\forall R1, R2, P[room(R1) \wedge room(R2) \wedge point(P)$
$\wedge strictlyInside(P, R1) \wedge$
$strictlyInside(P, R2) \rightarrow R1 = R2]$

The contrapositive is:

$\forall R1, R2, P[room(R1) \wedge room(R2) \wedge R1 \wedge point(P)$
$\wedge R1 \neq R2 \rightarrow$
$\sim strictlyInside(P, R1) \vee \sim strictlyInside(P, R2)]$

which is then put in canonical form:

$\forall R1, R2, P[room(R1) \wedge room(R2) \wedge R1 \wedge point(P)$
$\wedge strictlyInside(P, R1)$
$\wedge R1 \neq R2 \rightarrow \sim strictlyInside(P, R2)]$

Reformulating constraints for more efficient function evaluation

The MENDER subcompiler (the subject of Kerstin Voigt's Ph.D. work [VT89]) of KBSDE specializes

in incorporating global constraints c into hillclimbing patchers; these patchers take a candidate solution s that fails test c, and iteratively modifies s, one parameter value at a time, in such a way that improvement occurs with respect to an evaluation function f. f is constructed by MENDER so that there is some value k such that when hillclimbing reaches $f(s) \geq k$, c(s) is simultaneously satisfied. For example, the FILL constraint, "The rooms must completely fill the floorplan space" is completely satisfied when $f(s) = HLength * HWidth$, where f(s) is the number of 1x1 "unit tiles" in the floorplan that are covered by rooms.

MENDER handles those global constraints that can be viewed as *resource assignment problems* (RAPs) involving assigning a fixed set of resource units to a dynamically generated set of consumers in a specified way. For example, the FILL constraint can be viewed as a RAP wherein the the resources are unit tiles in the rectangular floorplan, the consumers are rectangular rooms and the required assignment is that each resource unit be assigned (at least) one consumer. (Note that other constraints such as NONOV and INS ensure that each resource unit is assigned *exactly* one consumer.)

Compilation is based on a taxonomy of resource assignment problem *schemas*, where what is varying across the schemas is the nature of the assignment (one-to-one, onto, etc.) Associated with each schema is a method for constructing an evaluation function appropriate for that kind of RAP. Thus the schema for an "onto" assignment (such as FILL) has an associated evaluation function which counts the total number of resource units "covered" by consumers in a particular state.

Efficiency improvement

The RAP schemas can be organized into a specialization lattice. More specialized schemas have more constraints on the assignment; because, therefore, more is known about such RAPs, they also often have more efficiently evaluatable functions. For example, the most specialized RAP, where the relation between resource units and consumers is "one-to-one" and "onto" (e.g., as between unit tiles in the floorplan and unit tiles in the room rectangles), can take advantage of the fact that all the consumers must have associated resource units assigned to them. (The nature of the overall search algorithm architecture in which the hillclimbing patcher is embedded guarantees that the patcher will be passed a candidate solution that satisfies the "one-to-one" constraint, though not necessarily the "onto" constraint.) It further relies on the common occurrence of a strictly hierarchical structure in the consumer organization (e.g., the consumer unit tiles are grouped into rectangular rooms). On the basis of these facts, the associated evaluation function can count the total number of "covered" resource units by counting the total number of assigned consumer units, which is the same as summing the *sizes* of the (mutually exclusive) consumer groups into which the consumer units are partitioned. Thus, if the FILL constraint were viewed as an instance of this RAP schema, the associated evaluation function would add the *areas* of the placed rooms (which are architecturally gauranteed to be inside the house and non-overlapping).

Such specialized schemas match a *conjunction* of constraints (e.g., the most specialized RAP matches FILL ∧ INS ∧ NONOV). Initially, a global constraint is completely successfully matched against one of the less specialized RAP schemas. Each of the specializations of the RAP schema constitutes a potential reformulation opportunity. Such an opportunity is processed in a goal-directed fashion, in the sense that domain-specific instances of the additional constraints which must also be true to match the more specialized RAP schema are then sought among the conjuncts of P(o) (or proven to be antecedents for P(o)).

Reformulating constraints for designing abstraction levels

The HiT subcompiler (the subject of Sunil Mohan's Ph.D. work [Moh91]) of KBSDE specializes in dividing the search algorithm architecture into two or more levels (if two, they are called the "base level" and the "abstract level"). Each of these levels has an associated search algorithm, configured from (constrained) generators, testers, and hillclimbing patchers (see, e.g., the earlier two-level example).

A (generally global) constraint P(s) can serve as the basis for constructing an abstraction level in the following sense: An abstraction function mapping solutions s into abstractions f(s) is constructed (e.g., f might abstract a "room" into a "room area"). An abstract generator Generate(i:input,a:range(f)) can then be constructed which generates all elements a in the range of f(s). P(s) can be abstracted into test P'(a), which is applicable to abstract candidate solutions. Thus one possible search algorithm for the abstract level is:

> Generate($i : input, a : range(f)$);
> Test($a \mid P'(a)$)

HiT currently is organized around two schemas representing constraint types whose very form makes it easy to construct an abstraction function:

1. *Functional constraints: P(F(s)).* Based on such a constraint, the abstract level generates $\{z \mid z = F(s)\}$ and the base level is then responsible for ensuring that P(refinement(z)) holds.
2. *Disjunctive constraints:* $\forall s, t[solution(s) \wedge partType(t, s) \rightarrow \forall p : T[part(p, s) \rightarrow P1(p) \vee P2(p) \vee ... \vee Pn(p).]]$ The abstract level generator selects *one* of these disjuncts to be true by fiat (i.e., it posts the disjunct as a constraint). The base level must then construct an s

that satisfies that disjunct.

Compilability

Reformulating a constraint into disjunctive form. Several general rules are used to carry out this reformulation (some of which are similar to those used to transform a predicate logic assertion into conjunctive normal form). These include: "Move negations inward", "Reformulate the expression as a disjunction", and "Remove variables that refer to non-solution objects."

Using these transformations, a constraint such as NONOV:

> For all pairs of rooms R1 and R2, it is not that the case there exists a point p that is both inside room R1 and inside some other room R2.

or:

$$\forall R1, R2[Room(R1) \wedge Room(R2) \wedge R1 \neq R2 \rightarrow \sim (\exists p[Inside(p, R1) \wedge Inside(p, R2)])$$

is eventually re-expressed as:

$$\forall R1, R2[Room(R1) \wedge Room(R2) \wedge R1 \neq R2 \rightarrow [\sim L(R1, R2) \vee \sim B(R1, R2) \vee \sim R(R1, R2) \vee \sim A(R1, R2)]]$$

where the predicates are defined as follows:

- L(R1,R2): The x coordinate of the right side of R1 is less than or equal to the x coordinate of the left side of R2.
- B(R1,R2): The y coordinate of the top side of R1 is less than or equal to the y coordinate of the bottom side of R2.
- R(R1,R2): The x coordinate of the left side of R1 is greater than or equal to the x coordinate of the right side of R2.
- A(R1,R2): The y coordinate of the bottom side of R1 is greater than the y coordinate of the top side of R2.

At this point, the constraint matches the "disjunctive constraint schema" and HiT can now proceed to construct an abstract level where, for every pair of rooms $< R1, R2 >$, one of the four above relationships is generated as a constraint to be satisfied.

Efficiency improvement

Deriving composition laws for the disjunctive case. As is readily noticed, picking topological relations at random between pairs of rooms is not likely to very rapidly converge on an abstract solution that is actually concretely realizable.

Fortunately, because the predicates of disjuncts can be viewed as defining relations, we can sometimes exploit known or provable properties of relations such as transitivity, reflexivity, or symmetry. Such properties can be viewed more constructively as *composition rules*. Thus for the L(R1,R2) relation, the following two composition rules can be shown to hold:

- *Transitivity.* If $L(R1, R2)$ and $L(R2, R3)$, then $L(R1, R3)$.
- *No reflexivity.* $\sim L(R1, R1)$.

These composition laws can then be made operational in several ways, including: incorporating them into the abstract generators using RICK; using them to dynamically prune the ranges of the abstract generators; using them as abstract tests (supplementing the original test).

Discussion and conclusions

Summary

In this paper, we have briefly described a number of reformulation techniques for use during knowledge compilation, either to make constraints compilable in the first place, or to put them in a form that compiles into a more efficient search algorithm. The reformulation techniques described here are schema-specific; matching of a constraint to a given subcompiler's schema can be aided by a reformulation technique, or a constraint that (already) matches a particular schema can be put in a form (possibly one that matches another schema) that will allow it to be more efficiently satisfied.

Implementation status

At this point, the reformulation techniques discussed for use in the RICK subcompiler have been implemented; the ones associated with the MENDER and HiT subcompilers are the subject of ongoing research.

Related work

Antecedent derivation. A schema-specific approach to schema-matching is usefully contrasted with a general-purpose approach, such as Smith's antecedent derivation method [Smi82]. One difference (we believe) is that the "antecedent derivation" process for a given schema can be restricted to using a specified (small) set of inference rules associated with that schema. A second difference is that in some cases, our schema-specific reformulation technique is a "normalization" process that works in the *forward* direction (e.g., to put a constraint in a disjunctive form).

DRAT. Like KBSDE, another schema-oriented approach that is more specialized than Smith's antecedent derivation method is Van Baalen's DRAT system [BD88]. KBSDE's target is an efficient generate-test-patch architecture; in contrast, DRAT's target is an efficient (object-oriented) forward-chaining theorem prover. Both systems take a classification-based approach to assigning specified constraints to schemas. However, KBSDE's schemas correspond to generic search algorithm components such as generator or patcher types, whereas DRAT's schemas correspond to (efficient implementations for) generic forward-chaining rules.

In KBSDE, "incorporating a constraint" in a constrained generator means that the constraint need no longer be represented explicitly in the problem solver; the generator is guaranteed to only produce acceptable solutions. Similarly, in DRAT, "capturing a constraint" in a rule implementation also means that that constraint need not be explicitly mentioned in the problem solver. KBSDE's ideal is to *incorporate* all constraints in a single (hierarchically organized) *constrained generator*, which produces completely correct solutions in polynomial time. Since this ideal is seldom achieved (it would require finding a solution representation in which all constraints are localizable to solution parts), KBSDE has a set of fallback strategies: incorporate the "leftover" global constraints in a patcher, in new abstraction levels, or (least preferred) in a tester.

DRAT's analogue to our reformulation techniques for enabling compilability is called concept introduction; by considering alternative formulations for one of the task's concepts, DRAT can sometimes find a representation that allows more constraints to be capturable. A process called operationalization then tries to capture the "leftover", uncaptured constraints by writing procedures and using these to further specialize the already selected representations.

Code optimization.. Our reformulations associated with efficiency improvement are similar in spirit to intermediate code optimization in standard compiler technology, in the sense that such optimizations are done: (a) at a level of abstraction higher than the target level (in our case, reformulating constraints into other constraints); and (b) based on a thorough knowledge of how the compiler to the target level works.

Research directions

The set of reformulation techniques presented here is under development. Some of the areas still in need of further development are: techniques for reformulating constraints to match RAP schemas; techniques for matching the functional constraint schema; techniques for improving the efficient processing of functional constraints; and an elaboration of how to best exploit derived composition laws for newly constructed abstraction levels.

Acknowledgements

I am grateful to Mark Shirley and Rob Holte for reviewing an earlier draft of this paper. I am especially grateful to my Spiritual Teacher, Sri Da Avabhasa. The research reported here was supported in part by the National Science Foundation (NSF) under Grant Numbers IRI-9017121 and DMC-8610507, in part by the Defense Advanced Research Projects Agency (DARPA) under DARPA-funded NASA Grant NAG2-645, in part by the DARPA under Contract Number N00014-85-K-0116, and in part by the Center for Computer Aids to Industrial Productivity (CAIP), Rutgers University, with funds provided by the New Jersey Commission on Science and Technology and by CAIP's industrial members. The opinions expressed in this paper are those of the authors and do not reflect any policies, either expressed or implied, of any granting agency.

References

J. Van Baalen and R. Davis. Overview of an approach to representation design. In *Proceedings of the AAAI*, pages 392–397, St. Paul, MN, August 1988.

W. Braudaway. Automated synthesis of constrained generators. In M. Lowry and R. McCartney, editors, *Automating Software Design*. AAAI Press, 1991.

W. Braudaway. *Knowledge compilation for incorporating constraints*. PhD thesis, Rutgers University, New Brunswick, NJ, December 1991.

S. Mohan. Constructing Hierarchical Solvers for Functional Constraint Satisfaction Problems. In *Proceedings of the AAAI Spring Symposium*, New Brunswick, NJ, Spring 1991. Also available as AI/Design Project Working Paper #172.

D. R. Smith. Derived preconditions and their use in program synthesis. In D. W. Loveland, editor, *Proceedings of the Sixth Conference on Automated Deduction*, pages 172–193. Springer-Verlag, New York, 1982. Lectures Notes in Computer Science 138.

C. Tong. A divide-and-conquer approach to knowledge compilation. In M. Lowry and R. McCartney, editors, *Automating Software Design*. AAAI Press, 1991. Also available as Rutgers AI/Design Project Working Paper #174.

K. Voigt and C. Tong. Automating the construction of patchers that satisfy global constraints. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1446–1452, Detroit, MI, August 1989.

The Role of Reformulation in the Automatic Design of Satisfiability Procedures

Jeffrey Van Baalen
Computer Science Department
University of Wyoming
jvb@moran.uwyo.edu

Abstract

Recently there has been increasing interest in the problem of *knowledge compilation* [Selman&Kautz91]. This is the problem of identifying tractable techniques for determining the consequences of a knowledge base. We have developed and implemented a technique, called DRAT, that given a *theory*, i.e., a collection of first-order clauses, can often produce a type of decision procedure for that theory that can be used in the place of a general-purpose first-order theorem prover for determining the many of the consequences of that theory. Hence, DRAT does a type of knowledge compilation. Central to the DRAT technique is a type of reformulation in which a problem's clauses are restated in terms of different nonlogical symbols. The reformulation is isomorphic in the sense that it does not change the semantics of a problem.

INTRODUCTION

Recently there has been increasing interest in the problem of *knowledge compilation* [Selman&Kautz91]. This is the problem of identifying tractable techniques for determining the consequences of a knowledge base. Most interesting knowledge bases are written in highly expressive languages for which the general problem of complete inference is intractable (e.g., at least NP-hard, usually undecidable). Even though the general inference problem in such a language is intractable, given a particular knowledge base, it is often possible to identify a tractable inference procedure that is complete for the inferences required in that knowledge base.

We have developed and implemented a technique, called DRAT, that given a *theory*, i.e., a collection of first-order clauses, can often produce a type of decision procedure for that theory. This type of procedure is called a *literal satisfiability procedure*. Such a satisfiability procedure for a theory T decides whether or not a conjunction of ground literals is satisfiable in T. A literal satisfiability procedure for a theory can be used in the place of a general-purpose first-order theorem prover for determining the many of the consequences of that theory. Hence, DRAT does a type of knowledge compilation.

Obviously, we are better off using a satisfiability procedure for determining the consequences of a theory than we are using a general-purpose theorem prover because the satisfiability procedure is guaranteed to halt. However, under what circumstances should we consider such a procedure tractable? A straightforward way to define tractability is polynomial-time worst-case complexity and for some theories DRAT can produce a satisfiability procedure that has this property. For many other theories, the satisfiability procedures produced are exponential in the worst case. Note that DRAT can determine whether a satisfiability procedure it produces has polynomial or exponential worst-case behavior. In either case, the procedures are usually much more efficient than a general theorem prover because the complexity of the theorem prover proving that a fact F follows from a theory T is a function of the sum of the size of $F \cup T$, while the complexity of the satisfiability procedure is a function of the size of F.

Even when DRAT cannot produce a literal satisfiability procedure for an entire theory it is often an improvement to use a procedure for a subset of an input theory because such a procedure can be interfaced with a general-purpose theorem prover in such a way that the procedure and the theorem prover work together to determine the consequences of the theory.

In practice, so long as a procedure can be found for a significant subset of the theory, the resulting inference systems are much more efficient than the theorem prover alone because many of the inferences that the theorem prover would have to do are done more efficiently by the satisfiability procedure.

Let Ψ be the set of axioms of a problem and let S be the satisfiability procedure that DRAT designs for Ψ', some subset of Ψ. The theorem prover restricts its manipulation of the statements in Ψ', using S instead whenever possible. This paper presents a formalization of DRAT and proves that it is complete, i.e., that for any first-order statement ϕ, if $\Psi \models \phi$, S combined with the theorem prover will prove ϕ. We show that DRAT's reformulation greatly increases its effectiveness and that a solution to a reformulated version of a problem is

guaranteed to be a solution to the original problem.

We present only a brief description of the DRAT algorithm here. A detailed description of an implementation can be found in [VanBaalen89] or [VanBaalen92].

DRAT was inspired by human problem solving performance on analytical tasks of the type found on graduate level standardized admissions tests. An example problem is given in Figure 1.

Given: M, N, O, P, Q, R, and S are all members of the same family. N is married to P. S is a grandchild of Q. O is a niece of M. The mother of S is the only sister of M. R is Q's only child. M has no brothers. N is a grandfather of O.
Query: Who are the siblings of S?

Figure 1: The FAMILIES Analytical Reasoning Problem

We analyzed human problem-solving behavior on a number of these problems and found the prevalent use of diagrams to assist in problem solving. Figure 2 illustrates the typical diagrams people use to solve the problem in Figure 1.

"R is the only child of Q" "S is a grandchild of Q"
(Divided rectangles represent couples; circles represent sets of children of the same couple: full circles are closed sets, broken circles are sets all of whose members may not be known; the directed arc represents the "children-of" function between couples and their sets of children.)

Figure 2: Two statements in a representation commonly used by people.

These diagrams were found to contain a common set of structures (across different people and different problems). The arcs in Figure 2 are an example of such a structure. They represent the 1–1 function between a married couple and their set of children. Each common structure was also found to have a standard set of procedures for manipulating it. For example, one procedure associated with the arcs in Figure 2 ensures that they behave like a 1–1 function. It reads roughly as, "If two objects are equal and they appear at the same end of two separate 1–1 function arcs with the same function symbol, the arcs and the objects at their other end can be composed." This procedure is among those used to compose the structures in Figure 2 to yield the diagram in Figure 3.

People use these diagrams to test the satisfiability of a particular collection of facts by creating the structures representing each fact and then composing them. The conjunction is satisfiable just in case no contradiction is signalled in the composition process.

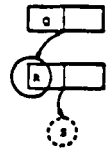


Figure 3: Composition of the structures in Figure 3.

DRAT has a library of procedures called *schemes*. These schemes model people's diagrammatic structures and their manipulations. Schemes were found to have a number of important properties which are described in this paper. Perhaps the most important of these properties is that each scheme turns out to be a satisfiability procedure. Another important property of schemes is that they can be used as building blocks to construct "larger" satisfiability procedures. DRAT uses this property to construct satisfiability procedures for input problems.

The implementation of DRAT includes the schemes found in analyzing the diagrams that people used on thirty analytical tasks. It has been tested on twelve of these problems stated in a sorted first-order logic. The problems vary in size from thirty to sixty sorted first-order statements. The performance of the theorem prover/satisfiability procedure combinations that DRAT produces for these problems was at least two orders of magnitude better than the performance of the theorem prover alone. For example, our general theorem prover took 988,442 resolutions— three hours and five minutes—to solve the problem shown in Figure 1. The satisfiability procedure that DRAT produced was able to solve the problem entirely without the theorem prover and did so in less than three seconds.

PRELIMINARIES

Each scheme is a *tractable literal satisfiability procedure* for a *theory*.

Definition 1 A *theory* is a set of statements in first-order predicate calculus with equality.

Definition 2 A *literal satisfiability procedure* for a theory T is a procedure that decides for any conjunction of ground literals Σ whether or not $\Sigma \cup T$ is satisfiable.

Each scheme is *tractable* in the sense that, given any Σ containing n literals, the scheme for a theory T decides the satisfiability of $\Sigma \cup T$ in time polynomial in n.

Given a particular Σ, in addition to determining literal satisfiability in some theory, each scheme computes $\{u = v \mid u, v \in C \wedge \Sigma \models u = v\}$ where C is the set of constant symbols appearing in Σ. As detailed in section , these equalities are communicated between schemes in a way that allows the combination

of schemes to determine satisfiability for the union of their theories.

One important result of this research is the particular library of schemes we have developed from the observation of human problem solving of analytical tasks. However, in the formal characterization that follows, we abstract away from the detail of the current scheme library, identifying the properties of schemes required for the completeness of DRAT.

This paper first takes a simplified view of what DRAT will accept as an input problem and also assumes that DRAT is only successful if it can produce a satisfiability procedure for an entire problem. In this setting, we prove that a combination of schemes is a satisfiability procedure for the union of the theories of the individual schemes. In section , the above restrictions are relaxed and it is shown how, in the more general setting, the procedures produced by DRAT are interfaced with a theorem prover.

DRAT requires that the formulas of schemes and the formulas of an input problem be converted to *clauses*, i.e., disjunctions of first-order literals. The remainder of the paper assumes that this has been done. However, the presentation will often use more intuitive forms for statements, when the conversion to clause form is straightforward.

The restricted definition of a problem taken first is:

Definition 3 A *problem* is a triple $< \Sigma, T_C, \Phi >$, where Σ and Φ are sets of ground literals and T_C is a set of clauses each of which contains at least one variable. Such a triple is interpreted as a question about whether or not for all the ground literals $\phi \in \Phi$, $\Sigma \cup T_C \models \phi$.

Here is an example problem:

$$\Sigma_1 = \left\{ \begin{array}{l} grandfather(O,N), married(N,P) \\ grandchild(S,Q), niece(O,M), \\ M \neq N, N \neq O, \ldots \end{array} \right\}$$

$$T_{C_1} = \left\{ \begin{array}{l} mother(S,x) \Leftrightarrow sister(M,x), \\ (sister(M,x) \wedge sister(M,y)) \Rightarrow x = y, \\ child(Q,x) \Leftrightarrow x = R, \\ \neg brother(M,x), \ldots \end{array} \right\}$$

$$\Phi_1 = \{sibling(O,S), child(N,M)\}$$

In addition to those axioms shown, Σ_1 also contains disequalities between all of the individual constants mentioned. T_{C_1} also contains definitions of concepts such a *grandchild* and formulas defining general properties of the family relation domain such as symmetry of *married*.

Given a problem $< \Sigma, T_C, \Phi >$, DRAT's objective is to design a literal satisfiability procedure for T_C. This procedure is used to solve the problem for the particular Σ and Φ. To determine whether for some $\phi \in \Phi$, $\Sigma \cup T_C \models \phi$, the satisfiability procedure for T_C is used to decide whether or not $\Sigma \cup T_C \cup \neg\phi$ is unsatisfiable. For example, DRAT tries to design a satisfiability procedure for T_{C_1}. If successful, the procedure is used to decide whether "O" is a sibling of "S" and "M is a child of "N" follow from $\Sigma_1 \cup T_{C_1}$ by determining the satisfiability of $\Sigma_1 \cup T_{C_1} \cup \neg sibling(O,S)$ and of $\Sigma_1 \cup T_{C_1} \cup \neg child(N,M)$.

Obviously, we are better off using a satisfiability procedure for T_C to solve a problem $< \Sigma, T_C, \Phi >$ than using a general theorem prover because the satisfiability procedure is guaranteed to halt. Perhaps less obvious is the fact that these procedures are usually much more efficient than a general theorem prover. The intuition behind this is that the complexity of the theorem prover solving the problem is a function of the size of the entire problem, while the complexity of the satisfiability procedure is a function of the size of $\Sigma \cup \Phi$. As pointed out in section , this intuition is substantiated by the performance of the procedures that DRAT has designed.

THE DRAT TECHNIQUE

We will call the relation, function and individual constant symbols in a theory the *nonlogical symbols* of that theory. The nonlogical symbols of each scheme's theory are treated as parameters to be instantiated with the nonlogical symbols of T_C. For example, the scheme $T_{symmetric}$ whose theory is $\{R(x,y) \Rightarrow R(y,x)\}$ is parameterized by R.

DRAT tries to find a set of scheme instances that can be combined to give a literal satisfiability procedure for T_C. Consider a set of scheme instances. Call the union of the theories of each scheme instance T_I. DRAT has succeeded in finding a satisfiability procedure when it finds a T_I that is logically equivalent to T_C. The following is an abstract description of this process:

instances $\leftarrow \emptyset$
$T_I \leftarrow \emptyset$
$T'_C \leftarrow T_C$
UNTIL empty(T'_C) DO
    instance $\leftarrow$ choose-instance(T'_C)
    IF null(*instance*) THEN EXIT-WITH failure
    instances $\leftarrow$ union(*instance*, *instances*)
    $T_I \leftarrow$ union(theory(*instance*), T_I)
    FOR EACH $\phi \in T'_C$
        WHEN $T_I \models \phi$ DO $T'_C \leftarrow T'_C - \phi$
    END FOR
END UNTIL

A set of scheme instances is built up incrementally and, simultaneously, the set of clauses in T'_C is paired down. Each time **choose-instance** is invoked, it inspects T'_C and chooses a scheme instance whose theory is entailed by T'_C. After the theory of *instance* is added to T_I, DRAT removes clauses from T'_C that are entailed by T_I.

DRAT uses the following procedure for computing satisfiability in T_I to determine the $\phi \in T'_C$ that follow from T_I. For each clause ϕ, it creates ϕ' by substituting a new individual constant for each unique variable in ϕ. If the satisfiability procedure for T_I reports that $\neg\phi' \cup T_I$ is unsatisfiable, $T_I \models \phi$.

If the algorithm is exited with T'_C empty, DRAT has succeeded in finding a T_I that is equivalent to T_C. To

see this, note that $T'_C \cup T_I \equiv T_C$ is an invariant of the loop. Adding theory(*instance*) to T_I does not violate the condition because $T'_C \models$ theory(*instance*). Removing from T'_C clauses ϕ such that $T_I \models \phi$ also does not violate the condition.

If the algorithm is exited because **choose-instance** returns nil, it has failed to find a T_I that is equivalent to T_C.

Note that this algorithm is nondeterministic because, in general, on a call to **choose-instance**, there are several instances from which to choose. The DRAT implementation searches for an appropriate collection of scheme instances. This search is reduced considerably by the fact that scheme instances in T_I may not share nonlogical symbols. As discussed in section , this restriction is required to allow schemes to be combined by the method described below. More detail on how the DRAT implementation controls this search can be found in [VanBaalen92].

A PROCEDURE FOR COMBINING SCHEMES

Since T_I is the theory of a set of scheme instances, so long as these instances do not share nonlogical symbols, DRAT has a satisfiability procedure for T_I. This procedure is the combination of schemes used to create T_I. DRAT's combination technique is the same technique as reported by Nelson & Oppen in [Nelson&Oppen79] and a more detailed description than what follows can be found there.

Let $\mathcal{L}(T)$ be the set of nonlogical symbols appearing in the clauses of T. We will often refer to $\mathcal{L}(T)$ as the language of T. Consider two scheme instances, T_1 and T_2, where $\mathcal{L}(T_1)$ is disjoint from $\mathcal{L}(T_2)$, and consider a conjunction of literals Σ in $\mathcal{L}(T_1 \cup T_2)$. The procedure for deciding the satisfiability of $\Sigma \cup T_1 \cup T_2$ begins by splitting Σ into two conjunctions of literals: Σ_1, with literals in $\mathcal{L}(T_1)$ and Σ_2, with literals in $\mathcal{L}(T_2)$ such that the conjunction of literals in Σ_1 and Σ_2 is satisfiable just in case Σ is.

When a literal in Σ contains nonlogical symbols from $\mathcal{L}(T_1 \cup T_2)$, remove each subterm whose function symbol is not in the language of the head symbol of the term. A subterm is removed by substituting a new constant symbol for that subterm in the literal and conjoining an equality between the term and the new symbol with the proper Σ_i. For example, suppose R is in $\mathcal{L}(T_1)$, f is in $\mathcal{L}(T_2)$ and Σ contains the literal $R(f(a))$. The embedded term is in the wrong language, so it is removed. This is done by substituting a new constant, say b, for $f(a)$ in $R(f(a))$ to obtain $R(b)$ and conjoining $b = f(a)$ with Σ_2.

For each literal in Σ, this technique is applied repeatedly to the right most function symbol in the wrong language until the literal no longer contains symbols in the wrong language. Then the literal is conjoined with the appropriate Σ_i. For instance, $R(b)$ from the example above contains no symbols in the wrong language so it is conjoined with Σ_1.

Next the scheme for T_1 is used to determine the satisfiability of $\Sigma_1 \cup T_1$. Recall that in so doing, this scheme also computes the set of equalities between constants in Σ_1 that follow from $\Sigma_1 \cup T_1$. Call this set E_1. The scheme for T_2 is used to determine the satisfiability of $\Sigma_2 \cup T_2 \cup E_1$. If it is satisfiable, E_2, the set of equalities that follow from $\Sigma_2 \cup T_2 \cup E_1$, is propagated back to T_1, i.e., T_1 is used to compute $\Sigma_1 \cup T_1 \cup E_2$.

This propagation of equalities continues until one of the schemes reports "unsatisfiable" or until no new equalities are computed. Note that since there are at most $n-1$ nonredundant equalities between n constant symbols, this process will terminate. Unless the scheme for T_1 or T_2 reports "unsatisfiable," the procedure for the combination returns "satisfiable."

A complication to this equality propagation procedure is that given a set of ground literals, many tractable schemes imply disjunctions of equalities between constants without implying any of the disjuncts alone, a property called *nonconvexity* in [Nelson&Oppen79]. An example of a convex scheme is one that determines satisfiability for the theory of equality with uninterpreted function symbols. An example of a nonconvex scheme is one for the theory of sets. To see this, note that $\{a,b\} = \{c,d\}$ implies $a = c \vee a = d$, but does not imply either equality alone.

A scheme associated with a nonconvex theory must compute disjunctions of equalities between constants that follow from a given conjunction of ground literals. The equality propagation procedure is extended to handle such schemes by case splitting when a nonconvex scheme produces a disjunction. When one of the component schemes produces the disjunction $c_1 = d_1 \vee \cdots \vee c_n = d_n$, the combined satisfiability procedure is applied recursively to the conjunctions $\Sigma_1 \cup \Sigma_2 \cup \{c_1 = d_1\}, \ldots, \Sigma_1 \cup \Sigma_2 \cup \{c_n = d_n\}$. If any of these is satisfiable, "satisfiable" is returned, otherwise "unsatisfiable" is returned.

As a simple example of this procedure, consider two schemes: $\mathcal{E}$ for the theory of equality with uninterpreted function symbols and $\mathcal{S}$ for the theory of finite sets. Now consider whether

$$\Sigma = \left[\begin{array}{l} f(a) = \{b,g\} \wedge f(c) = \{d,e\} \wedge a = c \wedge \\ g \neq d \wedge g \neq e \wedge b \neq d \wedge b \neq e \end{array} \right]$$

is satisfiable. First Σ is split into

$$\Sigma_1 = \left[\begin{array}{l} a = c \wedge g \neq d \wedge g \neq e \wedge b \neq d \wedge b \neq e \wedge \\ f(a) = c_1 \wedge f(c) = c_2 \end{array} \right]$$

$\Sigma_2 = [c_1 = \{b,g\} \wedge c_2 = \{d,e\}]$.

$\mathcal{E}$ is run on Σ_1 and determines that $c_1 = c_2$. $\mathcal{S}$ is run on $\Sigma_2 \cup \{c_1 = c_2\}$ which produces the disjunction $b = d \vee b = e$. The procedure is now invoked recursively for $\Sigma_1 \cup \Sigma_2 \cup \{b = d\}$ and $\Sigma_1 \cup \Sigma_2 \cup \{b = e\}$. In both calls, Σ_2 produces the disjunction $g = d \vee g = e$ which is unsatisfiable. Therefore, both calls return "unsatisfiable," hence $\Sigma \cup \mathcal{E} \cup \mathcal{S}$ is unsatisfiable.

We place one additional requirement on schemes

to make the equality propagation procedure practical. Schemes must be *incremental*. This means that a scheme must be able to save its "state" when a conjunction of literals is satisfiable and it must be able to use the saved state to determine the satisfiability of larger conjunctions at incremental cost.

REFORMULATION

The DRAT technique as described in section is severely limited by the way in which a problem is stated. Often, it is much more successful with an equivalent formulation of the problem stated in terms of a different collection of nonlogical symbols. For instance, recall the problem about family relations given in section . It was stated in terms of the binary relation *child*. It turns out that, given the current scheme library, the DRAT implementation is much more successful when the problem is stated in terms of *parents*, a function from an individual to his or her set of parents. One reason this formulation is better is that the library contains a scheme for a theory of fixed sized sets. DRAT discovers an instance of this scheme that allows it to remove several general clauses from the problem including one that limits the size of parent sets to two.

In an effort to circumvent this sensitivity to a problem's formulation, DRAT is able to reformulate a problem in terms of new nonlogical symbols without changing the "meaning" of the problem. **Choose-instance** is often able to find scheme instances in reformulated problems where it was unable to do so in the initial formulations. DRAT's reformulation technique is modeled after the reformulation that people do in solving analytical tasks. For an example of this refer again to the problem and diagrams given in section . In the diagrams appear concepts such as "married couples" and "sets of children of the same couple." These concepts are not present in the initial problem formulation — the problem has been reformulated.

DRAT does a particular kind of reformulation called *isomorphic reformulation* in [Korf80]. We formalize isomorphic reformulation as a relation between theories.

Definition 4 A *reformulation map* $\mathcal{R}^*_{\mathcal{L}_1,\mathcal{L}_2}$ *between two languages* $\mathcal{L}_1$ *and* $\mathcal{L}_2$ is a function from clauses in $\mathcal{L}_1$ to sets of clauses in $\mathcal{L}_2$.

Definition 5 A theory T_2 is an *isomorphic reformulation* of a theory T_1 just in case there exists a reformulation map $\mathcal{R}^*_{\mathcal{L}(T_1),\mathcal{L}(T_2)}$ such that
$T_1 \models \phi \Leftrightarrow T_2 \models \mathcal{R}^*_{\mathcal{L}(T_1),\mathcal{L}(T_2)}(\phi)$, for every clause ϕ in $\mathcal{L}(T_1)$.

If T_2 is an isomorphic reformulation of T_1, any question we have about what clauses are entailed by T_1 can be answered by theorem proving in T_2. Given the question, "does $T_1 \models \phi$?" we use $\mathcal{R}^*$ to translate ϕ into $\mathcal{L}(T_2)$ and then attempt to prove that $T_2 \models \mathcal{R}^*(\phi)$.

As a simple example of isomorphic reformulation, consider the following two theories:

$$T_1 = \left\{ \begin{array}{l} R(x,x), \\ R(x,y) \Rightarrow R(y,x), \\ R(x,y) \wedge R(y,z) \Rightarrow R(x,z) \end{array} \right\}$$

$$T_2 = \left\{ \begin{array}{l} x \in R\text{-}class(x), \\ x \in R\text{-}class(y) \Rightarrow y \in R\text{-}class(x), \\ x \in R\text{-}class(y) \wedge y \in R\text{-}class(z) \Rightarrow \\ \qquad\qquad x \in R\text{-}class(z) \end{array} \right\}$$

T_2 is an isomorphic reformulation of T_1. To show this, we exhibit an appropriate $\mathcal{R}^*_{\mathcal{L}(T_1),\mathcal{L}(T_2)}$. First, we introduce the function γ with $\gamma(R(x,y)) = x \in R\text{-}class(y)$ and $\gamma(\neg R(x,y)) = x \notin R\text{-}class(y)$.
The function γ is also defined in the obvious way for literals that are instances of the patterns $R(x,y)$ and $\neg R(x,y)$, i.e., given the constants a and b, $\gamma(R(a,f(b))) = a \in R\text{-}class(f(b))$.

Given the literals $\phi_1, \ldots, \phi_n$, $n \geq 1$
$\mathcal{R}^*_{\mathcal{L}(T_1),\mathcal{L}(T_2)}(\phi_1 \vee \cdots \vee \phi_n) = \{\gamma(\phi_1) \vee \cdots \vee \gamma(\phi_n)\}$.

Now $T_2 \equiv \mathcal{R}^*_{\mathcal{L}(T_1),\mathcal{L}(T_2)}(T_1)$, using the obvious extension of $\mathcal{R}^*$ to sets of clauses. Therefore,
$T_1 \models \phi \Leftrightarrow T_2 \models \mathcal{R}^*_{\mathcal{L}(T_1),\mathcal{L}(T_2)}(\phi)$. To see this, note that we can take any resolution proof of $T_1 \vdash \phi$ and uniformly apply $\mathcal{R}^*_{\mathcal{L}(T_1),\mathcal{L}(T_2)}$ to the clauses in each step of the proof to obtain a proof of $\mathcal{R}^*_{\mathcal{L}(T_1),\mathcal{L}(T_2)}(T_1) \vdash \mathcal{R}^*_{\mathcal{L}(T_1),\mathcal{L}(T_2)}(\phi)$. We can also define $\mathcal{R}^*_{\mathcal{L}(T_2),\mathcal{L}(T_1)}$ similarly to $\mathcal{R}^*_{\mathcal{L}(T_1),\mathcal{L}(T_2)}$ and use it to transform any proof $\mathcal{R}^*_{\mathcal{L}(T_1),\mathcal{L}(T_2)}(T_1) \vdash \mathcal{R}^*_{\mathcal{L}(T_1),\mathcal{L}(T_2)}(\phi)$ into a proof of $T_1 \vdash \phi$.

ADDING REFORMULATION TO DRAT

One strategy for finding a satisfiability procedure for a theory T_1 is to identify a theory T_2 with the following properties: (1) a satisfiability procedure is known for T_2, (2) we can find a reformulation map $\mathcal{R}^*_{\mathcal{L}(T_1),\mathcal{L}(T_2)}$ demonstrating that T_2 is an isomorphic reformulation of T_1 and (3) $\mathcal{R}^*_{\mathcal{L}(T_1),\mathcal{L}(T_2)}$ is a computable function.

The actual DRAT technique is an extension of the algorithm discussed in section to apply the above strategy. This extension enables DRAT to generate theories that are isomorphic reformulations of T_C while searching for a set of scheme instances that is a satisfiability procedure for T_C. DRAT has a library of reformulation rules, each of which is a reformulation map. These rules are applied to an input theory T_C to construct theories that are isomorphic reformulations of T_C. The extended algorithm searches for scheme instances in these isomorphic reformulations as well as in the original T_C.

Roughly, each reformulation rule is viewed as an axiom schema that can be instantiated with nonlogical symbols and used as a rewrite rule to reformulate a theory. To understand this view, consider the following axiom schema in which R is a parameter:

$R(x,y) \Leftrightarrow x \in F_R(y)$.
This states that for any binary relation, there is a projection function F_R that is a mapping from individuals to sets of individuals such that $F_R(y) = \{x \mid R(x,y)\}$.

DRAT can apply the above reformulation rule to binary relations in T_C. When the rule is applied to R in T_C, the new function symbol F_R is introduced and T_C is reformulated in terms of F_R. For instance, if this rule is applied to *child* in the family relations problem given earlier, it will introduce a function that we will call *parents*, from an individual to his or her set of parents. DRAT uses the formula introducing *parents*, i.e., $child(x,y) \Leftrightarrow x \in parents(y)$, to reformulate the problem, rewriting all occurrences of $child(x,y)$ to $x \in parents(y)$.

This example reformulation rule can be applied to any binary relation in any theory. More generally, DRAT's reformulation rules are conditional on *properties* of nonlogical symbols in a theory. A property of a nonlogical symbol is simply a first-order statement mentioning that symbol. Before giving the general form of reformulation rules, we introduce the function $rf\text{-}symbols(T)$, the set of relation and function symbols of T. The $rf\text{-}symbols(T)$ does not contain the symbols $=$ or $\in$, even if they are mentioned in T. These are treated as special (logical) symbols in the reformulation process.

The general form of reformulation rules is given in the following definition.

Definition 6 A triple $< P, Q, \Theta \Leftrightarrow \Psi >$ is a *reformulation rule* when it meets the following restrictions: (1) P and Q are conjunctions of clauses (both of which may be empty). (2) Θ and Ψ are conjunctions of literals. (3) $rf\text{-}symbols(P) \subseteq rf\text{-}symbols(\Theta)$ and $rf\text{-}symbols(Q) \subseteq rf\text{-}symbols(\Psi)$. (4) $rf\text{-}symbols(\Theta)$ is disjoint from $rf\text{-}symbols(\Psi)$. (5) Θ and Ψ have the same variables.

Rules are symmetric in the sense that their biconditionals can be used to introduce new symbols in "either direction." When the parameters in Θ are instantiated with symbols in a theory T, the rule is used to reformulate T in terms of the new symbols in Ψ. The conjunction of clauses P is the condition that must be true of a theory for the reformulation rule to be used to rewrite Θ as Ψ. When the parameters in Ψ are instantiated with the symbols in T, the rule is used to reformulate T in terms of the new symbols in Θ. In this case, Q is the condition that must be true for the rule to be used.

Here is an example of a conditional reformulation rule:
$< [x \in F(y) \Rightarrow F(y) = \{x\}],,$
$[x \in F(y) \Leftrightarrow x \neq \bot \wedge x = F'(y)] >$.[1]
This rule can be applied to any theory T containing a function F whose range elements are sets of size one, i.e., $P = [x \in F(y) \Rightarrow F(y) = \{x\}]$. When applied, the rule reformulates T in terms of a function F' such that $F'(y) = x$ just in case $x \in F(y)$. Q is empty in this rule because the rule can always be applied in the other direction.

The following is an abstract description of the DRAT algorithm extended to do reformulation:

instances $\leftarrow \emptyset$
$T_I \leftarrow \emptyset$
$T'_C \leftarrow T_C$
$\mathcal{R}^* \leftarrow \lambda(t).t$
UNTIL empty(T'_C) DO
  EITHER
    ref-pairs $\leftarrow$ choose-ref-pairs(T'_C)
    IF null(*ref-pairs*) THEN EXIT-WITH failure
    symbols, *rule* $\leftarrow$ choose(*ref-pairs*)
    instantiated-rule $\leftarrow$ instantiate(*rule*, *symbols*)
    $T'_C \leftarrow \mathcal{R}(instantiated\text{-}rule, T'_C)$
    $\mathcal{R}^* \leftarrow \lambda(t).\mathcal{R}(instantiated\text{-}rule, \mathcal{R}^*(t))$
    instance $\leftarrow$ choose-instance(T'_C)
    IF null(*instance*) THEN EXIT-WITH failure
    instances $\leftarrow$ union(*instance*, *instances*)
    $T_I \leftarrow$ union(theory(*instance*), T_I)
    FOR EACH $\phi \in T'_C$
      WHEN $T_I \models \phi$ DO $T'_C \leftarrow T'_C - \phi$
    END FOR
END UNTIL

DRAT nondeterministically either chooses a reformulation rule and reformulates T'_C or adds the theory of the new instance to T_I. **Choose-instance** identifies an *instance* by identifying properties of the nonlogical symbols in T'_C. It looks for properties that appear in the theories of schemes. For example, when the scheme library contains a scheme one of whose axioms is $R(x,y) \Rightarrow R(y,x)$, DRAT attempts to choose instances of that scheme by looking for binary relations in T_C that have the symmetry property.

Choose-ref-rule uses the identified properties of nonlogical symbols in T'_C to identify reformulation rules that can be applied to those symbols. Rules introduce new symbols as explained above. **Choose-ref-rules** returns a list of $< symbols, rule >$ pairs, where *symbols* is an ordered list of nonlogical symbols. Each pair in the list can be applied to T_C by instantiating the parameters of the rule with *symbols*. For a rule of the form
$< P, Q, \Theta \Leftrightarrow \Psi >$,
symbols can either be used to instantiate the parameters in Θ or in Ψ, but not both. Conditional rules are returned only when T'_C entails their condition. **Choose-ref-rule** guarantees that if *symbols* instantiates Θ then P follows from T'_C; If *symbols* instantiates Ψ, it guarantees that Q follows.

As before, if DRAT exits with T'_C empty, it has succeeded in finding a T_I equivalent to T_C; Otherwise, it has failed.

[1] The symbol $\bot$ is used in specifying axioms about partial functions, $F(a) = \bot$ means that $F(a)$ is undefined.

Again we have suppressed the issues of search by giving a nondeterministic procedure. The search conducted by the extended algorithm is over a much larger space than the search conducted by the simple algorithm described in section . The DRAT implementation with reformulation must compare alternative problem formulations. Fortunately, we have found some effective heuristics for controlling the search. See [VanBaalen89] or [VanBaalen91] for details.

The procedure **instantiate**, instantiates a *rule* with respect to the nonlogical symbols in *symbols* to produce an *instantiated-rule*. $\mathcal{R}$ is the reformulation procedure. We describe this procedure for the case where a rule of the form
$< P, Q, \theta_1 \wedge \cdots \wedge \theta_n \Leftrightarrow \Psi >$
is used to rewrite occurrences of $\theta_1 \wedge \cdots \wedge \theta_n$, the *from conjunct*, to occurrences of Ψ, the *to conjunct*. The procedure for applying the rule in the other direction is obtained by reversing the biconditional and replacing references to P by references to Q.

Each set of unit clauses in T'_C of the form $\{(\theta_1)\sigma, \ldots, (\theta_n)\sigma\}$, where σ is a substitution for the variables in the θ_i, is rewritten as the set of unit clauses $(\Psi)\sigma$. Each clause containing the literals $(\neg\theta_1)\sigma, \ldots, (\theta_n)\sigma$ is rewritten to contain $(\neg\Psi)\sigma$. After all possible occurrences are rewritten, the clauses in Q are added to the rewritten theory.

We call a rewriting produced by $\mathcal{R}$ *complete* when it removes all of the nonlogical symbols appearing in the from conjunct. $\mathcal{R}$ may or may not produce a complete rewriting. For example, given a right hand side of the form $R(f(x))$, rewriting will only be complete when R and f appear in a theory only in patterns of this form. If the rewriting process is not complete, $\mathcal{R}$ adds the instantiated $\Theta \Leftrightarrow \Psi$ to the rewritten theory.

As an example of applying $\mathcal{R}$, consider again the rule
$< [x \in F(y) \Rightarrow F(y) = \{x\}],,$
$[x \in F(y) \Leftrightarrow x \neq \perp \wedge x = F'(y)] >$.
As noted, the condition P must follow from a theory to reformulate F as F' in that theory. Since the condition Q is empty, there are no clauses to add to the resulting theory. If the rewriting is not complete, $[x \neq \perp \wedge x = F'(y) \Leftrightarrow x \in F(y)]$ is added to the rewritten theory. Since there is no condition Q, this rule can always be used, in the other direction, to reformulate F' as F. In this case, P is added to the rewritten theory. Again, the biconditional may need to be added to the rewritten theory.

To ensure that the extended DRAT algorithm generates only isomorphic reformulations, each reformulation rule must be shown to generate only isomorphic reformulations. To guarantee this, we require that, when instantiated, each reformulation rule be an *extending definition*.

Definition 7 A reformulation rule $< P, Q, \Theta \Leftrightarrow \Psi >$ is an *extending definition* if for all theories T the following conditions hold:

1. Whenever the $rf\text{-}symbols(\Theta) \subseteq rf\text{-}symbols(T)$, $rf\text{-}symbols(\Psi)$ is disjoint from $rf\text{-}symbols(T)$ and $T \models P$, then every model of T can be expanded to a model of $T \cup \{\Theta \Leftrightarrow \Psi\}$.
2. Whenever the $rf\text{-}symbols(\Psi) \subseteq rf\text{-}symbols(T)$, $rf\text{-}symbols(\Theta)$ is disjoint from $rf\text{-}symbols(T)$ and $T \models Q$, then every model of T can be extended to a model of $T \cup \{\Theta \Leftrightarrow \Psi\}$.

Section shows that for any reformulation rule *rule*, $\lambda(t).\mathcal{R}(rule, t)$ is a computable function and so long as *rule* is an extending definition, that whenever a theory T entails the appropriate condition of *rule*, $\mathcal{R}(rule, T)$ is an isomorphic reformulation of T.

The $\mathcal{R}^*$ produced by DRAT on the problem $< \Sigma, T_C, \Phi >$ is the composition of reformulation maps used by the algorithm to reformulate T_C. Since each reformulation map generates an isomorphic reformulation, $\mathcal{R}^*(T_C)$ is an isomorphic reformulation of T_C. Since each step is computable, $\mathcal{R}^*$ is a computable function.

Finally we point out that, since Ψ and Θ in the reformulation rule $< P, Q, \Theta \Leftrightarrow \Psi >$ are required to have the same variables, $\mathcal{R}^*(\Sigma)$ and $\mathcal{R}^*(\Phi)$ will always be ground. However, even though Σ and Φ are conjunctions of ground literals, $\mathcal{R}^*(\Sigma)$ and $\mathcal{R}^*(\Phi)$ may not be. To see this, suppose that Σ contains the literal $\neg\phi$ and $\mathcal{R}^*(\phi)$ is a conjunction. Then $\neg\mathcal{R}^*(\phi)$ will be a disjunction.

Section shows that when DRAT uses reformulation in designing a satisfiability procedure for a problem $< \Sigma, T_C, \Phi >$ and $\mathcal{R}^*(\Sigma)$ is a conjunction of literals, the problem can be solved by solving
$< \mathcal{R}^*(\Sigma), \mathcal{R}^*(T_C), \mathcal{R}^*(\Phi) >$. The fact that a satisfiability procedure for a reformulation of a problem requires $\mathcal{R}^*(\Sigma)$ to be a conjunction of literals is not a significant difficulty in the more general setting discussed in section in which satisfiability procedures are used in conjunction with a theorem prover.

AN EXAMPLE

In practice, we have found that adding reformulation to DRAT increases its effectiveness considerably. We illustrate this with a relatively simple example excerpted from the DRAT implementation design of a satisfiability procedure for the example problem given in section . We illustrate the implementation's behavior on the set T of clauses:
$\neg married(x, x)$,
$married(x, y) \Rightarrow married(y, x)$
$married(x, y) \wedge married(y, z) \Rightarrow \neg married(x, z)$
$married(y, x) \wedge married(z, x) \Rightarrow y = z$

There are three schemes in DRAT's library that are relevant to the example. The scheme $\hat{\mathcal{F}}$ for the theory of partial 1-1 functions with parameters F and F', which are inverse functions, and theory($\hat{\mathcal{F}}$)= $\{x = F(y) \wedge x \neq \perp \Leftrightarrow y = F'(x) \wedge y \neq \perp\}$; The scheme $\mathcal{S}_2$ for the theory of sets of size two with S as a parameter and theory($\mathcal{S}_2$)= $\{x_1 \in S \wedge x_2 \in S \wedge x_1 \neq x_2 \Rightarrow S =$

$\{x_1, x_2\}\}$; And, the scheme $\mathcal{E}$ for the theory of equality with uninterpreted function symbols.

The relevant reformulation rules are:

$r_1 =<, , R(x,y) \Leftrightarrow y \in F_R(x) >$

$r_2 =< x \in F(y) \Rightarrow F(y) = \{x\}, ,$
$[x \in F(y) \Leftrightarrow x \neq \perp \wedge x = F'(y)] >$

$r_3 =< (x \neq \perp \wedge y \neq \perp) \Rightarrow x = F(y) \Leftrightarrow y = F(x), ,$
$[x = F(y) \wedge x \neq \perp \Leftrightarrow F'(y) = \{x,y\} \wedge x \neq y] >$

As is typical in the implementation, these rules are normally used only in one direction. As noted in section , r_1 reformulates a binary relation in a theory as a function F_R onto sets: $F_R(x) = \{y \mid R(x,y)\}$. Also as noted in section , when applied to a theory containing a function F whose range elements are sets of size one, r_2 introduces a function F' such that $F'(y) = x$ just in case $x \in F(y)$. The rule r_3 reformulates an F that is its own inverse as a function F', mapping an individual into sets of size two such that $F'(x) = \{x, F(x)\}$.

Given the schemes above, DRAT is unable to design a satisfiability procedure for T without reformulation. In an effort to design a satisfiability procedure for all of T, the DRAT implementation repeatedly reformulates the problem, finally producing a formulation in terms of a function that we will call *couple*, mapping an individual to the married couple of which he or she is a member.

DRAT uses rule r_1 to reformulate T in terms of a function that we will call *spouses*, a mapping from an individual to the set of his or her spouses. $\mathcal{R}(r_1, T)$ is

$x \notin spouses(x)$,
$x \in spouses(y) \Rightarrow y \in spouses(x)$
$x \in spouses(y) \wedge y \in spouses(z) \Rightarrow x \notin spouses(z)$
$y \in spouses(x) \wedge z \in spouses(x) \Rightarrow y = z$

DRAT uses rule r_2 to reformulate $\mathcal{R}(r_1, T)$ in terms of a partial function that we will call *spouse*, a mapping from an individual to his or her spouse. $\mathcal{R}(r_2, \mathcal{R}(r_1, T))$ is

$x \neq spouse(x) \vee x = \perp$,
$x = spouse(y) \wedge x \neq \perp \Rightarrow y = spouse(x) \wedge y \neq \perp$
$x = spouse(y) \wedge x \neq \perp \wedge y = spouse(z) \wedge y \neq \perp$
$\Rightarrow x \neq spouse(z) \vee x = \perp$
$y = spouse(x) \wedge y \neq \perp \wedge z = spouse(x) \wedge z \neq \perp$
$\Rightarrow y = z$

Note that the second and fourth clauses in this set follow from instances of $\mathcal{F}$ and $\mathcal{E}$ respectively. Hence, if DRAT were to terminate at this point, T'_C would include only the first and third clauses.

DRAT uses rule r_3 to reformulate the above theory in terms of the function *couple*. The result is

$couple(x) \neq \{x,x\} \vee x = x$,
$couple(x) = \{x,y\} \wedge x \neq y \Rightarrow$
$couple(y) = \{y,x\} \wedge y \neq x$,
$couple(x) = \{x,y\} \wedge x \neq y \wedge$
$couple(y) = \{y,z\} \wedge y \neq z \Rightarrow$
$couple(x) \neq \{x,z\} \vee x = z$,
$couple(y) = \{x,y\} \wedge y \neq x \wedge$
$couple(z) = \{z,x\} \wedge z \neq x \Rightarrow y = z$

All of the clauses in this set follow from the combination of $\mathcal{S}_2$ and an instance of $\mathcal{E}$ containing the uninterpreted function symbol *couple*. Thus, through the use of reformulation, DRAT succeeds in designing a satisfiability procedure for the theory T. Without reformulation it is unable to design a procedure for any subset of T.

STEPS TOWARDS THE COMPLETENESS OF DRAT

This section proves two results towards the completeness of DRAT. First, we show that DRAT designs satisfiability procedures. If DRAT successfully designs a procedure for some set of axioms T_C, then that procedure can be used to decide the problem $< \Sigma, T_C, \Phi >$ for any conjunctions of ground literals Σ and Φ. Second, we consider the addition of reformulation to DRAT and show that a satisfiability procedure for $\mathcal{R}^*(T_C)$ can be used as a satisfiability procedure for T_C so long as $\mathcal{R}^*(\Sigma)$ is a conjunction of literals. These results are necessary preliminaries for the proof of completeness in section .

DRAT DESIGNS SATISFIABILITY PROCEDURES

Before proceeding to prove that DRAT designs satisfiability procedures, we recall properties of schemes presented thus far and discuss some additional required properties.

Recall that a scheme for a theory T is a procedure that decides the satisfiability of $\Sigma \cup T$, where Σ is a conjunction of ground literals. Given a particular Σ, each scheme also computes the set of equalities between constants in Σ that follow from $\Sigma \cup T$. If T is nonconvex, its scheme also computes disjunctions of equalities between constants in Σ that follow from $\Sigma \cup T$.

We call a first-order theory whose formulas contain no existential quantifiers a *quantifier-free* theory. An additional requirement on schemes is that their theories be quantifier-free. As a practical matter, this is not a serious restriction beyond restricting schemes to be tractable. See [Oppen80] for further discussion of this point.

The theories of schemes are also required to have infinite models. The equality propagation technique may not work if a theory has only finite models because, given a set of constant symbols larger than the set of individuals in the model's domain, such a theory implies the disjunction of equalities between those constant symbols. Theories with infinite models do not imply disjunctions of equalities between variables. Therefore, given a theory T with infinite models, such disjunctions can only follow from $T \cup \Sigma$, for some Σ whose satisfiability is being decided. Any disjunctions of equalities between constants that follow must involve only constants mentioned in Σ. This restriction to theories with infinite models does not appear to be significant. To date, we have not found any schemes that we could not include because they violated this restriction.

The theorem proved below is similar to the theorem given in [Nelson&Oppen79]. It differs in the addition of the requirement that each scheme's theory have infinite models. The theorem appearing in [Nelson&Oppen79] is incorrectly stated. The reason a different proof is included here is that the proof given in [Nelson&Oppen79] is incorrect.[2] We also include our proof because the technique is much more direct and serves as a foundation for research in progress to extend our results.

Theorem 1 *Let T_1 and T_2 be theories with no common nonlogical symbols. If there are schemes for T_1 and T_2, there is a scheme for $T_1 \cup T_2$.*

Proof: We prove that the procedure described in section for combining two schemes is a scheme for $T_1 \cup T_2$. If the scheme for T_1 or T_2 reports "unsatisfiability," clearly $\Sigma_1 \cup \Sigma_2 \cup T_1 \cup T_2$ is unsatisfiable and, since $\Sigma_1 \cup \Sigma_2$ and Σ are cosatisfiable, $\Sigma \cup T_1 \cup T_2$ is unsatisfiable. We must show that if the procedure of section reports "satisfiable," $\Sigma \cup T_1 \cup T_2$ is satisfiable. This is done by showing how to construct a model of $\Sigma \cup T_1 \cup T_2$ when the procedure reports "satisfiable."

Let $C = \{c_0, \ldots, c_n\}$ be the set of constant symbols appearing in Σ_1 or Σ_2. Let E be the set of equalities propagated by the procedure of section . As we will see, when the procedure halts, E contains all the $c_1 = c_2$ such that $c_1, c_2 \in C \wedge \Sigma_1 \cup \Sigma_2 \cup T_1 \cup T_2 \models c_1 = c_2$. E will also contain any equalities chosen when case splitting occurs.

Let $\overline{E} = \{c_1 = c_2 \mid c_1, c_2 \in C \wedge c_1 = c_2 \notin E\}$. Since the schemes for T_1 and T_2 reported "satisfiable," there are models of $\Sigma_1 \cup T_1 \cup E$ and $\Sigma_2 \cup T_2 \cup E$. Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be models of $\Sigma_1 \cup T_1 \cup E$ and $\Sigma_2 \cup T_2 \cup E$ respectively that agree on the interpretation of the equalities in $\overline{E}$. We show how to construct a model $\mathcal{M} \models \Sigma \cup T_1 \cup T_2$ from $\mathcal{M}_1$ and $\mathcal{M}_2$.

Before giving this construction, we show that it is possible to pick an $\mathcal{M}_1$ and $\mathcal{M}_2$ that agree on $\overline{E}$. First note that if $\overline{E}$ is empty, all $\mathcal{M}_1$ and $\mathcal{M}_2$ agree. Now suppose that $\overline{E}$ is not empty. In this case, there exists an $\mathcal{M}_1$and an $\mathcal{M}_2$ that do not satisfy any equality in $\overline{E}$. For suppose to the contrary. In particular, suppose that every $\mathcal{M}_1$satisfies some equality in $\overline{E}$. If $\overline{E}$ contains exactly one equality, $c_1 = c_2$, $\Sigma_1 \cup T_1 \cup E \models c_1 = c_2$ and $c_1 = c_2 \in E$, not $\overline{E}$. If $\overline{E}$ contains more than one equality, $\Sigma_1 \cup T_1 \cup E$ entails the disjunction of equalities in $\overline{E}$. But then $\Sigma_1 \cup T_1 \cup E$ is nonconvex which is impossible because, instead of returning satisfiable, the algorithm in section would have case split in this situation. This same argument can be made for $\mathcal{M}_2$ and, hence, there exists an $\mathcal{M}_2$ that does not satisfy any of the equalities in $\overline{E}$. Thus, we can choose an $\mathcal{M}_1$ and $\mathcal{M}_2$ that agree on the interpretation of the equalities in $\overline{E}$.

[2] A correct version of the theorem appears in [Nelson84], however, the proof given there is still incorrect.

Note that since $\mathcal{M}_1$ and $\mathcal{M}_2$ agree on the interpretation of the equalities in E and in $\overline{E}$, they agree on the interpretation of every equality between constants in C.

Let $\mathcal{M}_1 = < D_1, R_1, F_1, C_1 >$, where D_1 is the domain of $\mathcal{M}_1$, R_1 is the interpretation of relation symbols of $\mathcal{M}_1$ in D_1, F_1 is the interpretation of the functions symbols of $\mathcal{M}_1$ and C_1 is the interpretation of individual constant symbols in $\mathcal{M}_1$. Similarly, let $\mathcal{M}_2 = < D_2, R_2, F_2, C_2 >$.

We now construct $\mathcal{M}$ by merging $\mathcal{M}_1$ and $\mathcal{M}_2$ as follows. The domain of $\mathcal{M}$ is $D_1 \cup D_2'$, where D_2' is the domain of $\mathcal{M}_2'$, a modified version of $\mathcal{M}_2$. $\mathcal{M}_2'$ is obtained by replacing individuals in D_2 by individuals in D_1 when they are designated by the same constant symbol. For all constant symbols $c \in C$, replace every occurrence of $C_2(c)$ in D_2 by $C_1(c)$, i.e., $C_2'(c) = C_1(c)$ when c is a shared constant symbol and $C_2'(c) = C_2(c)$ otherwise. For all R in the domain of R_2, let $R_2'(R)$ be the set $R_2(R)$ modified by the above replacement procedure. Similarly, let F_2' be the new interpretation of the function symbols of $\mathcal{M}_2$. $\mathcal{M}_2' = < D_2', R_2', F_2', C_2' >$.

$\mathcal{M}_2$ and $\mathcal{M}_2'$ are isomorphic structures because $\mathcal{M}_1$ and $\mathcal{M}_2$ agree on the interpretation of every equality between constants in C. If $\mathcal{M}_1$ and $\mathcal{M}_2$ did not agree, then $\mathcal{M}_2$ and $\mathcal{M}_2'$ would not be isomorphic. For suppose, that $\mathcal{M}_1 \models c_1 = c_2$ but $\mathcal{M}_2 \not\models c_1 = c_2$. Then the two constant symbols designate the same individual in D_2' and different individuals in D_2 and, hence, $\mathcal{M}_2'$ is not isomorphic to $\mathcal{M}_2$.

To finish the construction of $\mathcal{M}$, we take $\mathcal{M} = < D_1 \cup D_2', R_1 \cup R_2', F_1 \cup F_2', C_1 \cup C_2' >$. Since $\mathcal{M}_1 \models \Sigma_1 \cup T_1$ and $\mathcal{M}_2' \models \Sigma_2 \cup T_2$, $\mathcal{M} \models \Sigma_1 \cup \Sigma_2 \cup T_1 \cup T_2$. Since $\Sigma_1 \cup \Sigma_2$ and Σ are cosatisfiable, $\mathcal{M} \models \Sigma \cup T_1 \cup T_2$ and the proof of the theorem is complete. □.

The fact that DRAT designs satisfiability procedures is a direct consequence of theorem 1. Since the result of combining two schemes is again a scheme, any number of schemes can be combined by this method.

DRAT DOES ISOMORPHIC REFORMULATION

This section includes the proofs of two properties of DRAT's reformulation procedure $\mathcal{R}$. These results are sufficient to show how a satisfiability procedure generated by DRAT for some reformulated theory can be used to solve the original problem.

Lemma 1 *If a reformulation rule (rule) is an extending definition in T of the form $< P, Q, \Theta \Leftrightarrow \Psi >$ and $T \models P$, then $\mathcal{R}(rule, T)$ is an isomorphic reformulation of T.*

Proof: The condition that must be met is that if $T \models P$, $T \models \phi \Leftrightarrow \mathcal{R}(rule, T) \models \mathcal{R}(rule, \phi)$,
for any clause $\phi \in \mathcal{L}(T)$. We prove the equivalent fact that if $T \models P$,

$SAT(T \cup \{\neg\phi\}) \Leftrightarrow SAT(\mathcal{R}(rule, T) \cup \neg\mathcal{R}(rule, \phi))$, where $SAT(T)$ means that T is satisfiable.

[$\Rightarrow$] If $SAT(T \cup \{\neg\phi\})$, $SAT(T \cup \{\Theta \Leftrightarrow \Psi\} \cup \{\neg\phi\})$ because, by the definition of extending definition, every model of T can be extended to a model of $T \cup \{\Theta \Leftrightarrow \Psi\}$. Therefore, there exists a model of $T \cup \{\Theta \Leftrightarrow \Psi\} \cup \{\neg\phi\}$. But
$T \cup \{\Theta \Leftrightarrow \Psi\} \cup \{\neg\phi\} \models \mathcal{R}(rule, T) \cup \neg\mathcal{R}(rule, \phi)$.
Hence every model of $T \cup \{\Theta \Leftrightarrow \Psi\} \cup \{\neg\phi\}$ is a model of $\mathcal{R}(rule, T) \cup \neg\mathcal{R}(rule, \phi)$. Since there exists a model of $T \cup \{\Theta \Leftrightarrow \Psi\} \cup \{\neg\phi\}$, there exists a model of $\mathcal{R}(rule, T) \cup \neg\mathcal{R}(rule, \phi)$ and hence, it is satisfiable.

[$\Leftarrow$] The proof in this direction is similar, with the added step of showing that every model of $\mathcal{R}(rule, T) \cup \neg\mathcal{R}(rule, \phi)$ can be extended to a model of $\mathcal{R}(rule, T) \cup \{\Theta \Leftrightarrow \Psi\} \cup \neg\mathcal{R}(rule, \phi)$. Since *rule* is an extending definition, every model of a theory T_1 that entails Q can be extended to a model of $T_1 \cup \{\Theta \Leftrightarrow \Psi\}$. By the definition of $\mathcal{R}$, the clauses of Q will appear in $\mathcal{R}(rule, T)$ and hence $\mathcal{R}(rule, T) \models Q$. Therefore, every model of $\mathcal{R}(rule, T)$ can be extended to a model of $\mathcal{R}(rule, T) \cup \{\Theta \Leftrightarrow \Psi\}$. Thus, if $\mathcal{R}(rule, T) \cup \neg\mathcal{R}(\phi)$ is satisfiable, so is $T \cup \{\neg\phi\}$. □

It follows directly from this lemma and the fact that extending definitions can be used in either direction, that a reformulation rule $(P \wedge Q) \Rightarrow [\Theta \Leftrightarrow \Psi]$ with the $rf\text{-}symbols(\Psi)$ instantiated in term of a theory T can be used to reformulate T in terms of Θ so long as $T \models Q$.

Lemma 2 *For any reformulation rule (rule), the function $\lambda(t).\mathcal{R}(rule, t)$ is computable.*

Proof: Suppose the biconditional of *rule* is $\Theta \Leftrightarrow \Psi$ and $\mathcal{R}$ applies rule to rewrite occurrences of Ψ to occurrences of Θ in T, as described in section . Since $rf\text{-}symbols(\Theta)$ are disjoint from $rf\text{-}symbols(T)$, a rewrite step can never introduce a pattern of literals to which *rule* can be applied a second time. The rewrite is applied repeatedly until one of the following events occurs: (1) all of the symbols in $rf\text{-}symbols(\Psi)$ are removed from T or (2) no new occurrences of Ψ can be found, even though symbols in $rf\text{-}symbols(\Psi)$ are still present. In either case, repeated application of the rewrite rule terminates. Hence, $\lambda(t).\mathcal{R}(rule, t)$ is computable. □

The two preceding lemmas are sufficient to show that a satisfiability procedure for $\mathcal{R}^*(T_C)$ can be used to solve the problem $< \Sigma, T_C, \Phi >$, so long as $\mathcal{R}^*(\Sigma)$ is a conjunction of ground literals. Assuming that $\mathcal{R}^*(\Sigma)$ is a conjunction, the satisfiability procedure is used to solve the problem by solving $< \mathcal{R}^*(\Sigma), \mathcal{R}^*(T_C), \mathcal{R}^*(\Phi) >$ as follows. For each $\phi \in \Phi$, if $\neg\mathcal{R}^*(\phi)$ is a conjunction of literals, we use the procedure to determine if $\mathcal{R}^*(\Sigma) \cup \mathcal{R}^*(T_C) \cup \neg\mathcal{R}^*(\phi)$ is unsatisfiable. This is the case if and only if $\Sigma \cup T_C \cup \neg\phi$ is unsatisfiable. If $\neg\mathcal{R}^*(\phi)$ is a disjunction of literals, the procedure is used to determine the satisfiability of $\mathcal{R}^*(\Sigma) \cup \mathcal{R}^*(T_C) \cup l$, for each literal $l \in \neg\mathcal{R}^*(\phi)$. If any of these is satisfiable, $\mathcal{R}^*(\Sigma) \cup \mathcal{R}^*(T_C) \cup \neg\mathcal{R}^*(\phi)$ is satisfiable; otherwise it is unsatisfiable.

THE COMPLETENESS OF DRAT

Two simplifying assumptions were made in the previous sections. First, in definition 3, it was assumed that a problem for DRAT was of a restricted form. Second, it was assumed that DRAT's success depended on designing a satisfiability procedure for all of T_C. Both of these assumptions are now relaxed and we show how a literal satisfiability procedure is interfaced with a resolution theorem prover in such a way that the procedure/theorem prover combination is complete.

A problem for DRAT is now taken to be a pair $< \Gamma, \phi >$, where Γ is a set of first-order formulas and ϕ is a first-order formula. A pair $< \Gamma, \phi >$ is interpreted as the question, "$\Gamma \models \phi$?"

As a typical preprocessing step for resolution theorem proving, Γ and $\neg\phi$ are converted to sets of clauses which will be called Γ' and $\neg\phi'$ respectively. Let T_C be the set of nonground clauses in Γ'. As before, DRAT is used to design a literal satisfiability procedure for T_C. However, instead of exiting with failure if it is unable to design a procedure for all of T_C, it returns the satisfiability procedure and T_C', those clauses not incorporated into the satisfiability procedure. Also, as before, DRAT returns the reformulation map $\mathcal{R}^*$.

The algorithm given in section refers to the set of clauses for which a literal satisfiability procedure has been designed as T_I. Here that procedure is referred to as $\mathcal{S}_{T_I}$. We show how $\mathcal{S}_{T_I}$ is used along with a resolution theorem prover to demonstrate the unsatisfiability of $Cl = \mathcal{R}^*(\Gamma') \cup \mathcal{R}^*(\neg\phi')$. The nonground clauses of Cl are manipulated by the theorem prover in the usual way, except that clauses in T_I are prohibited from resolving with ground clauses. These resolutions are unnecessary because $\mathcal{S}_{T_I}$ is a "compression" of any resolution steps that can result from such a resolvant.

$\mathcal{S}_{T_I}$ is used in the manipulation of ground clauses in Cl and ground clauses derived from Cl during theorem proving. It is interfaced to the theorem prover via *theory resolution*[Stickel85]. One type of theory resolution, called *total narrow* theory resolution, requires a decision procedure for a theory T, given a set of literals L, to compute subsets L' of L such that $L' \cup T$ is unsatisfiable. Such a procedure is used to compute *T-resolvants* of a set of clauses as follows. Consider the decomposition of the clauses into $K_i \vee L_i$, where each K_i is a single literal in $\mathcal{L}(T)$ and L_i is disjunction of literals (possibly empty). For each subset of the K_i, say $\{K_{i_1}, \ldots, K_{i_n}\}$, that is unsatisfiable in T, the clause $L_1 \vee \cdots \vee L_n$ is a T-resolvant.

The theorem prover constructs T_I-resolvants from ground clauses, using $\mathcal{S}_{T_I}$ to compute sets of ground literals that are unsatisfiable in T_I. Let GrL be the set of ground unit clauses in Cl and let $GrCl$ be the set of ground nonunit clauses in Cl. First, the ground clauses are separated into clauses that are in $\mathcal{L}(T_I)$ and clauses

that are not. This is accomplished for the clauses in GrL using the procedure described in section ; It is accomplished for clauses in $GrCl$ in a similar fashion.

If a ground clause c_1 contains a literal that is not in $\mathcal{L}(T_I)$ and a ground clause c_2 contains the negation of that literal, the theorem prover computes the resolvant of c_1 and c_2 in the normal way. T_I-resolvants are computed using $\mathcal{S}_{T_I}$ to compute sets of ground literals that are unsatisfiable in T_I as follows. Let GrL_{T_I} be the set of literals in GrL that are in $\mathcal{L}(T_I)$. Let $GrCl_{T_I}$ be the set of literals in $\mathcal{L}(T_I)$ appearing in clauses of $GrCl$. We input progressively larger subsets of $GrLits = GrL_{T_I} \cup GrCl_{T_I}$ to $\mathcal{S}_{T_I}$ as long as those sets are satisfiable in T_I. Once a set is unsatisfiable in T_I, all supersets of it will also be unsatisfiable. When the theorem prover deduces a new ground literal in GrL_{T_I}, it is added to $GrLits$. The smallest subsets of $GrLits$ found to be unsatisfiable in T_I are used to compute T_I-resolvants of ground clauses.

Theorem 2 *Given the problem $< \Gamma, \phi >$, let $\mathcal{S}_{T_I}$ be a literal satisfiability procedure for $T_I \subseteq \mathcal{R}^*(\Gamma)$. If $\Gamma \models \phi$, $\mathcal{S}_{T_I}$ combined with the theorem prover will demonstrate the unsatisfiability of Cl.*

Proof: In [Stickel85], Stickel shows that, given a set of clauses $K_i \vee L_i$, if a decision procedure for a theory T computes all subsets K_i that are *minimally* unsatisfiable in T, total narrow theory resolution is complete. We must show that the above procedure for computing T_I-resolvants computes all subsets of $GrLits$ that are minimally unsatisfiable in T_I. Clearly, so long as $\mathcal{S}_{T_I}$ is a literal satisfiability procedure, the above procedure computes all these subsets. Thus, the completeness result follows directly from the results of section . □

The procedure described above can be made much more efficient. There are several refinements used by the DRAT implementation to consider far fewer subsets for unsatisfiability in T_I. We discuss two of these here. One refinement is to distinguish between literals in GrL_{T_I} and $GrCl_{T_I}$. First, we consider the satisfiability of GrL_{T_I}. If this is unsatisfiable, we are done. Otherwise, we consider progressively larger sets of literals appearing in clauses in $GrCl_{T_I}$. For each such set s, $\mathcal{S}_{T_I}$ is used to determine whether or not $GrL_{T_I} \cup s$ is unsatisfiable in T_I.

Note that the subsets identified with this refinement are not always minimal: it is possible for a subset of $GrCl_{T_I}$ union a subset of GrL_{T_I} to be unsatisfiable in T_I. However, it turns out that completeness of theory resolution is retained in this case, since the extraneous literals are in GrL_{T_I} and, therefore, are unit clauses.

A second simpler refinement only considers subsets of $GrCl_{T_I}$ each of whose elements appears in a different clause in $GrCl$.

As a final point about the efficiency of the procedure for computing subsets that are minimally unsatisfiable in T_I, recall that schemes are required to be incremental. Because of this, $\mathcal{S}_{T_I}$ is used very efficiently to consider progressively larger sets of literals.

It is often most effective to leverage the use of $\mathcal{S}_{T_I}$ by doing as much of the theorem proving as possible at the "ground level." The DRAT implementation uses "set of support" strategy which is very effective in accomplishing this when $\neg\phi'$ is ground because it tends to produce ground resolvants.

Summary and Ongoing Work

We have presented a formalization of DRAT: a technique for automatic design of satisfiability procedures. We have shown how these procedures are interfaced to a theorem prover so that it can, in many cases, prove theorems more efficiently. Given Ψ, the set of axioms of a problem, and $\mathcal{S}_{\Psi'}$, a literal satisfiability procedure designed for $\Psi' \subseteq \Psi$, we have proven that for any first-order statement ϕ, if $\Psi \models \phi$, the theorem prover/$\mathcal{S}_{\Psi'}$ combination will prove ϕ.

The major steps of our argument were as follows:

1. We showed that a combination of satisfiability procedures with certain properties is again a satisfiability procedure.
2. We showed that the reformulation that is essential to DRAT's effectiveness is isomorphic reformulation and, therefore, a satisfiability procedure of a reformulated theory can be used to solve problems in the original theory.
3. We proved the completeness of our technique for combining literal satisfiability procedures with a theorem prover. In this combination, $\mathcal{S}_{\Psi'}$ is used to compute Ψ'-resolvants from ground clauses and the theorem prover is restricted so that it does not resolve ground clauses on literals in $\mathcal{L}(\Psi')$.

In our ongoing work, we are attempting to extend DRAT's scheme combination technique. As much as possible, we would like to remove the restriction on the sharing of nonlogical symbols between component scheme instances in combinations. We are exploring the conditions under which limited types of overlap between nonlogical symbols is allowed. When overlap is allowed, component schemes must propagate more information than just equalities between constant symbols. In most cases where overlap is allowed and in which the schemes propagate at least the set of equalities between constants, it is not difficult to show the completeness of a propagation technique. The major issue that arises is proving that the propagation terminates.

As an example, consider allowing two schemes to share function symbols. The schemes must propagate all equalities between ground terms involving shared function symbols. The proof technique used in section can be extended to prove that such schemes combined by an appropriately extended propagation technique will produce semi-decision procedures for the combinations of their theories. However, in general, it is not possible to prove that the propagation will terminate.

One situation in which overlap is allowed occurs when the theories of schemes are sets of clauses in a sorted first-order logic. In this case, a function symbol F whose range is disjoint from its domain can be shared between schemes because terms of the form $F(F(x))$ are not well formed and, hence, it is easy to show that propagation of terms involving F will terminate.

Acknowledgements Richard Fikes, Bob Nado, Mike Lowry, and David McAllester provided helpful comments on drafts. Dave McAllester pointed out the error in Nelson & Oppen's proof of the combination of satisfiability procedures and suggested the technique we used in our proof. Bob Nado participated in numerous discussions on many aspects of the paper.

References

Brachman, R.J., Fikes, R.E. and Levesque, H.J., "KRYPTON: A Functional Approach to Knowledge Representation," in Brachman, R.J and Levesque, H.J. (editors), Readings in Knowledge Representation, pp. 411-429, Morgan Kaufmann, 1985.

Cohn, A.G., "Many Many Sorted Logics," Workshop on Principles of Hybrid Reasoning, pp.63-78, 1988.

Korf, R.E., "Toward a Model of Representation Changes," *Artificial Intelligence*, 14, pp.41-78, 1980.

Loveland, D.W., Automated Theorem Proving: a logical basis, North Holland, 1978.

Selman,B. and Kautz, H., "Knowledge compilation using horn approximations," AAAI91, pp. 904-909, 1991.

Nelson, G. and Oppen, D.C., "Simplification by Cooperating Decision Procedures," *ACM Transactions on Programming Languages and Systems*, 1, pp.245-257, 1979.

Nelson, G. and Oppen, D.C., "Fast Decision Procedures Based on Congruence Closure," *Journal of the ACM*, 27, pp.356-364, 1980.

Nelson, G., "Combining Satisfiability Procedures by Equality-Sharing," in Bledsoe, W.W. and Loveland, D.W., Automated Theorem Proving: After 25 Years, American Mathematical Society, 1984.

Oppen, D.C., "Complexity, Convexity and Combinations of Theories," *Theoretical Computer Science*, 12, pp.291-302, 1980.

Stickel, M.E., "Automated Deduction by Theory Resolution," *Automated Reasoning*, 1, pp.333-355, Reidel Publishing Co., 1985.

Van Baalen, J., "Toward a Theory of Representation Design," MIT Artificial Intelligence Laboratory, Technical Report 1128, 1989.

Van Baalen, J., "The Completeness of DRAT, a Technique for the Automatic Design of Satisfiability Procedures," KR91, pp. 514-525, 1991.

Van Baalen, J., "Automated Design of Specialized Representations," to appear in *Artificial Intelligence*.

Sensor/Data Fusion Research Outline

Kyriakos P. Zavoleas
Department of Industrial Engineering and Information Systems
Northeastern University
kyriakos@nueng.coe.northeastern.edu

This article summarizes some aspects of the research carried out in the Center of Electomagnetics Research, Northeastern University, in the context of the Sensor/Data Fusion project, under the supervision or Prof. M. M. Kokar; our interest in change of represnetation originates by our application needs. Related issues of our research are briefly discussed in the following paragraphs.

Sensor/Data Fusion (SDF) is an application driven discipline, facing the problem of associating, combining and compromizing among various information sources. An information source can be a physical sensor, data from data/knowledge bases, or an external constraint or guidance either from the human or frome another system. Sensor/Data Fusion can be included in the more general field of Sensory Information Processing, where the term "sensor" is used in a broad sense.

Our research in the area of SDF lies in between theory and practice. The main goals set are the following:

1. Develop a systematic approach (a methodology) for designing, implementing, testing and maintaining sensor/data fusion systems that can:
   (a) interpret sensory information,
   (b) reason about situations using this information plus inputs from data bases (collections of known facts) and knowledge bases (collections of rules and heuristics) and other SDF systems.
2. Through theoretical and experimental investigations develop a theoretical framework in which designs of such systems can be formulated and analyzed.

The approach taken in this research is goal driven: first propose design requirements and specifications, then identify theoretical questions needed to be answered in order to carry out the proposed design, and then investigate these questions in a systematic way.

The prevailing idea in our conceptualization of a SDF system is to follow: data should be dynamically transformed, processed and represented into suitable forms, so as to facilitate combining and reasoning about them. To restrict the complexity issue of a Sensor/Data Fusion System, we have proposed a layered architecture [KZR91]. In establishing a good definition for the layers two factors are considered: the semantics of the data, and the form of representation used. To transform data from one layer to another a process of ***abstraction*** must be invoked; on the other hand a ***fusion*** process combines data of the same layer into data of the same or adjacent layers. While the system requirements of *autonomy, fidelity, consistency,* and *versatility* must be borne in the designer's mind, the main design issues are:

- what are the layers for a particular class of domains,
- what are the abstraction and fusion mechanisms,
- what should be the structure within the layers,
- how would the system dynamically choose among alternative processing paths.

An initial approach to guide the design of a Sensor/Data Fusion System makes use of logic [KZ92]. This provides for an idealized framework, which should be borne in mind in the design of a SDF system; however, in an actual design, issues of noise and uncertainty must be also dealt with. From the logical standpoint, we have to deal with *theories* and their *models*; collectively, they represent *formal systems*. *Theories* are collections of statements about the world expressed in a particular *formal language* (language of the theory). A theory includes a set of *inference rules* which are used for *deriving* new statements through the derivation process (theorem proving). Associated with a theory and a model is a set of mappings (*interpretation rules*) which assign correspondence between the theory and the model. This correspondence must fulfill the *adequacy postulate* – whatever statement can be derived in the theory, it can be shown to be true in the model. The model in our case is a set of data (including sensory data) about the world together with a set of operations on the data. Instead of deriving statements about the world using the theory, one can check whether the statement is fulfilled in the model; this process is called *model checking*. It is an open research question whether and when it is better to use theorem proving and when to use model checking. We intend to investigate such a question for the domain of sensor/data fusion.

The requirement of deriving consistent sets of conclusions must incorporate both models (real data about the world) and theories. In order to preserve consistency when implementing SDF system layers we need to make sure that both the abstraction process and the fusion process fulfill this requirement. As a consequence of this, in our logical framework we could formulate the following definition of fusion:

Fusion is a process of combining two formal systems (two models and two theories) into a new formal system in such a way that we obtain a new formal system, i.e., such that the requirement of logical adequacy is fulfilled.

Along with the theoretical investigations that we have carried out until now, we are designing and implementing a prototype experimental system [KZ91], which is composed of a number of layers (currently two), incorporating abstraction and fusion processes. We are experimenting with this system on a chosen domain and examining the results of abstraction and fusion processes at each layer. This will allow us to formulate guidelines on the payoffs of doing abstraction vs. fusion in particular layers, choosing an appropriate layer for fusing data, or performing fusion dynamically in a number of layers, and combining/fusing the results.

References

[KZ91] M. M. Kokar and K. P. Zavoleas. Sensor/data fusion research at cer: Progress report. Technical report, Center of Electromagnetics Research, Northeastern University, Nov. 1991.

[KZ92] M. M. Kokar and K. P. Zavoleas. A logical framework of sensor/data fusion. In *Proceedings of the European Robotics and Intelligent Systems Conference EURISCON 1991*. IFAC, 1992.

[KZR91] M. M. Kokar, K. P. Zavoleas, and S. A. Reveliotis. A new framework for sensor/data fusion. Technical Report CSE-MMK-1-91, Center of Electromagnetics Research, Northeastern University, Jan. 1991.