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.

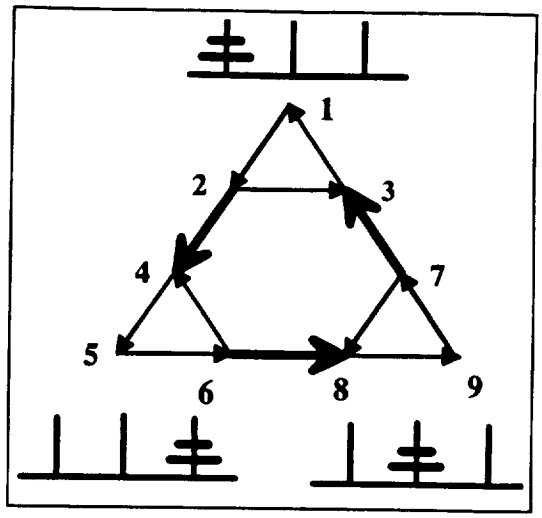
As a result of these three limitations, we are led to the conclusions that representation change is a necessary capability of any autonomous intelligent system, and that AI needs a fuller understanding of representations and their properties. In the next section, we consider the types of properties that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

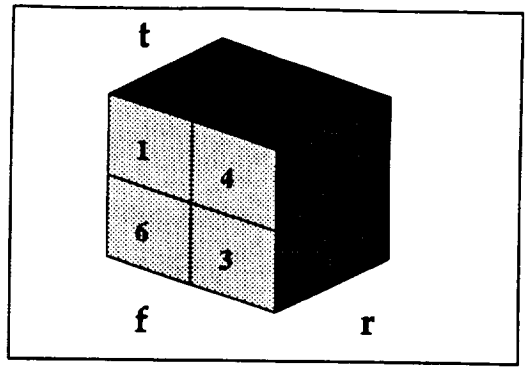
The subproblems are obtained by projecting the cylinder onto the x-axis and z-axis, respectively. In the new coordinates, good subproblems are obtained by projection. In the original coordinates, this is not the case; projecting onto any coordinate axis or coordinate plane yields a subproblem that is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

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

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

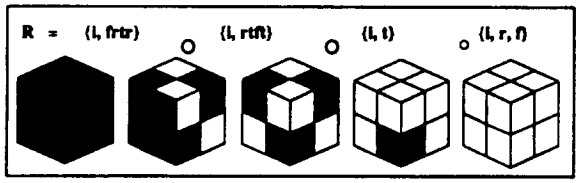
On cubelets 4, 5, 6, and 7, the subgroup generated is

{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:



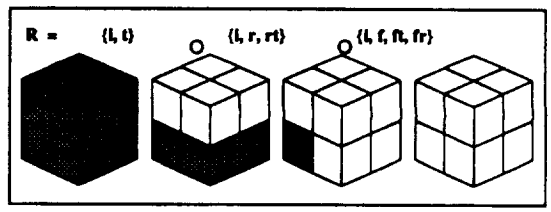
1. One of { i,r,f } brings cubelet 3 into cubicle 3.
2. One of { i,t } brings cubelet 1 and 2 into their places.
3. One of { i, rtft } brings (4,6) and (5,7) in the proper planes (`the front face looks right').
4. One of { i, frtr } finishes the Cube.

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.

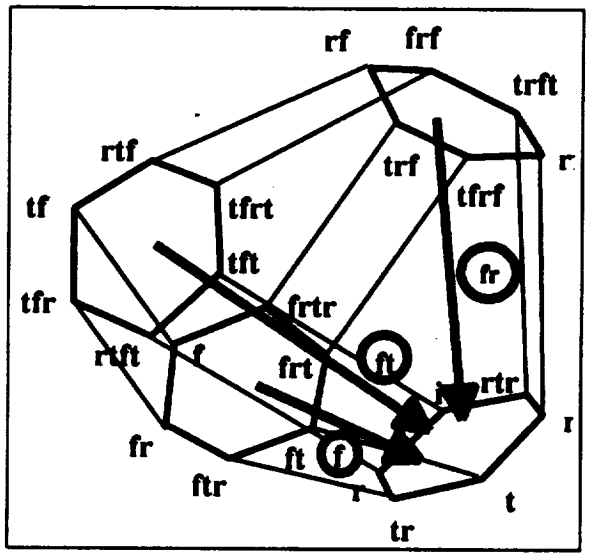
Serial Algorithm 2:



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:

D 0 **0**	D2	x, xx, **xxx** D 1
y, yxxy, yxxyxxy	yx, yxxyx, yxxyxxyx	yxx, yxxyxx, **yxxyxxyxx**
xy, xyxxy, xyxxyxxy	xyx, xyxxyx, **xyxxyxxyx**	xyxx, xyxxyx, xyxxyxxyxx
xxy, xxyxxy, **xxyxxyxxy**	xxyx, xxyxxyx, xxyxxyxxyx	xxyxx, xxyxx, 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 $q_\lambda, q'_\lambda \in S^1$ 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.