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.

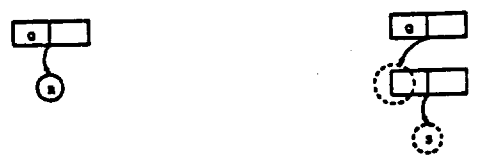
We present only a brief description of the DRAT algorithm here. A detailed description of an implementation can be found in [VanBaalen89] or [VanBaalen92].

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 struc-

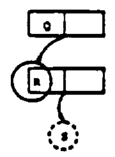


Figure 3: Composition of the structures in Figure 3.

tures representing each fact and then composing them. The conjunction is satisfiable just in case no contradiction is signalled in the composition process.

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 [illegible] to be a satisfiability procedure. Another important [illegible]perty of schemes is that they can be used as buildi[illegible]locks to construct "larger" satisfiability procedur[illegible] 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 [illegible] each scheme computes $\{u = v \mid u, v \in C \wedge \Sigma$ [illegible] $\models u = v\}$ 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 ← $\emptyset$
$T_I \leftarrow \emptyset$
$T'_C \leftarrow T_C$
UNTIL empty($T'_C$) DO
      instance ← choose-instance($T'_C$)
      IF null(instance) THEN EXIT-WITH failure
      instances ← union(instance, instances)
      $T_I$ ← 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 this 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

$$\Sigma = \begin{bmatrix} f(a) = \{b, g\} \wedge f(c) = \{d, e\} \wedge a = c \wedge \\ g \neq d \wedge g \neq e \wedge b \neq d \wedge b \neq e \end{bmatrix}$$

is satisfiable. First Σ is split into

$$\Sigma_1 = \begin{bmatrix} a = c \wedge g \neq d \wedge g \neq e \wedge b \neq d \wedge b \neq e \wedge \\ f(a) = c_1 \wedge f(c) = c_2 \end{bmatrix}$$

$\Sigma_2 = [c_1 = \{b, g\} \wedge c_2 = \{d, e\}]$.

$\mathcal{E}$ 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-symbols*(T), the set of relation and function symbols of T. The *rf-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-symbols*$(P) \subseteq$ *rf-symbols*(Θ) and *rf-symbols*$(Q) \subseteq$ *rf-symbols*(Ψ). (4) *rf-symbols*(Θ) is disjoint from *rf-symbols*(Ψ). (5) Θ and Ψ have the same variables.

Rules are symmetric in the sense that [illegible]ir biconditionals can be used to introduce new symb[illegible] 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:
$< [x \in F(y) \Rightarrow F(y) = \{x\}], ,$
$\quad [x \in F(y) \Leftrightarrow x \neq \perp \wedge x = F'(y)] >$.[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
    $ref\text{-}pairs$ ← choose-ref-pairs($T'_C$)
    IF null($ref\text{-}pairs$) THEN EXIT-WITH failure
    $symbols, rule$ ← choose($ref\text{-}pairs$)
    $instantiated\text{-}rule$ ← instantiate($rule, symbols$)
    [illegible] ← $\mathcal{R}(instantiated\text{-}rule, T'_C)$
    [illegible] ← $\lambda(t).\mathcal{R}(instantiated\text{-}rule, \mathcal{R}^*(t))$

    [illegible]$stance$ ← choose-instance($T'_C$)
    [illegible] null($instance$) THEN EXIT-WITH failure
    $instances$ ← union($instance, instances$)
    $T_I$ ← 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 instan[illegible]of that schem[illegible] by looking for binary relations in T[illegible]at have the [illegible]mmetry property.

Ch[illegible]e-ref-rule [illegible]ses the identified properties of n[illegible]logical 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 [illegible]arantees that Q follows.

As [illegible]ore, if DRAT exits with T'_C empty, it has succeede[illegible] finding a T_I equivalent to T_C; Otherwise, it has fa[illegible]l.

[1]The symbol $\perp$ is used in specifying axioms about partial functions, $F(a) = \perp$ 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:

$r_1 = <, , R(x,y) \Leftrightarrow y \in F_R(x) >$

$r_2 = < x \in F(y) \Rightarrow F(y) = \{x\}, ,$
$\quad [x \in F(y) \Leftrightarrow x \neq \bot \wedge x = F'(y)] >$

$r_3 = < (x \neq \bot \wedge y \neq \bot) \Rightarrow x = F(y) \Leftrightarrow y = F(x), ,$
$\quad [x = F(y) \wedge x \neq \bot \Leftrightarrow F'(y) = \{x,y\} \wedge x \neq y] >$

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

$x \neq spouse(x) \vee x = \bot$,
$x = spouse(y) \wedge x \neq \bot \Rightarrow y = spouse(x) \wedge y \neq \bot$
$x = spouse(y) \wedge x \neq \bot \wedge y = spouse(z) \wedge y \neq \bot$
$\quad \Rightarrow x \neq spouse(z) \vee x = \bot$
$y = spouse(x) \wedge y \neq \bot \wedge z = spouse(x) \wedge z \neq \bot$
$\quad \Rightarrow y = z$

Note that the second and fourth clauses in this set follow from instances of $\hat{\mathcal{F}}$ 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

$couple(x) \neq \{x, x\} \vee x = x$,
$couple(x) = \{x, y\} \wedge x \neq y \Rightarrow$
$\quad couple(y) = \{y, x\} \wedge y \neq x$,
$couple(x) = \{x, y\} \wedge x \neq y \wedge$
$\quad couple(y) = \{y, z\} \wedge y \neq z \Rightarrow$
$\qquad couple(x) \neq \{x, z\} \vee x = z$,
$couple(y) = \{x, y\} \wedge y \neq x \wedge$
$\quad couple(z) = \{z, x\} \wedge z \neq x \Rightarrow y = z$

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-symbols*(Ψ) 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-symbols*(Θ) are disjoint from *rf-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-symbols*(Ψ) are removed from T or (2) no new occurrences of Ψ can be found, even though symbols in *rf-symbols*(Ψ) 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. Th[illegible]s the case if and only if $\Sigma \cup T_C \cup \neg\phi$ is unsatisfiable. [illegible] $\neg\mathcal{R}^*(\phi)$ is a disjunction of literals, the procedure is [illegible]d 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 pro[illegible]ng, Γ and $\neg\phi$ are converted to sets of clauses which w[illegible] be called Γ' and $\neg\phi'$ respectively. Let T_C be the se[illegible]f nonground clauses in Γ'. As before, DRAT is used t[illegible] 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*.