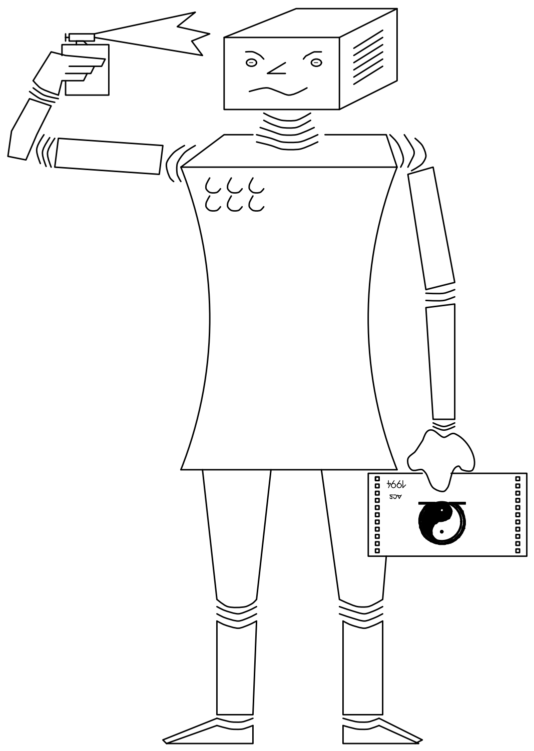

Reformulating Resolution Problems by Tactics

Manfred Kerber and Axel Präcklein

Published as: In Michel De Glas and Zdzislaw Pawlak, editors, *WOCFAI '95 – Proceedings of the Second World Conference on the Fundamentals of Artificial Intelligence*, pages 169–180, Paris, France, 1995.

Reformulating Resolution Problems by Tactics

Manfred Kerber
Fachbereich Informatik
Universität des Saarlandes
D-66041 Saarbrücken, Germany
e-mail: kerber@cs.uni-sb.de

Axel Präcklein
Neue Fahrstr. 13
D-75181 Pforzheim, Germany
e-mail: prckln@cs.uni-sb.de

Abstract

A straightforward formulation of a mathematical problem is mostly not adequate for resolution theorem proving. We present a method to optimize such formulations by exploiting the variability of first-order logic. The optimizing transformation is described as logic morphisms, whose operationalizations are tactics. The different behaviour of a resolution theorem prover for the source and target formulations is demonstrated by several examples. It is shown how tactical and resolution-style theorem proving can be combined.

Keywords: problem formulation, resolution, tactics, theorem proving.

1 Introduction

Solving mathematical problems with resolution-based theorem proving systems requires, in most cases, intelligence and ingenuity on the part of the user, since the final formulation of the problem is of essential importance for the problem-solving behaviour of the system. Often the main job is to formulate the task in a machine-friendly form, and once this is done, the system easily finds a proof. Of course this situation is far from being satisfactory, not at least, because the transformed rather than the original task, is solved; this is aggravated by the fact that the correspondence between both formulations is usually not clear. When the user modifies the problem until it can be mechanically solved, his reformulation efforts are typically not documented. Although this procedure is not convincing, in many situations some reformulations are necessary in order to obtain any automatically generated proofs at all. Therefore it is only reasonable to exclude reformulations for checking special features of a theorem prover but not for standard applications.

The general idea of changing the representation of a problem is classical: human mathematicians rephrase problems, often until the reformulation of the original problem is similar to that of a known problem. In fact, Pólya suggests such recasting of problems in his course on human mathematical problem solving [Pó65, vol.2, p.80]: "*Of course you want to restate the problem (transform it into an equivalent problem) so that it becomes more familiar, more attractive, more accessible, more promising.*"

In AI as in human problem solving the representation of a problem is essential to the behaviour of the problem-solving mechanisms. The example of the checkerboard without two opposite edges in [McCa64], [WOLB84, p.117] shows clearly that from the very beginning, the reformulation of problems has played a central role in the development of AI.

In the domain of automated mathematical problem solving, Bundy [Bu83, p.91] pleads for modifications of problems in order to obtain adequate formulations for computers. In section 4.3, he gives practical hints for reformulations making problems more digestible by theorem proving programs; in particular, he elaborates on the advantage of avoiding *all* function symbols.[1] Wos et al. [WOLB84, chapter 4] have also proposed several informal methods along these lines for representing various exercises for automated reasoning programs.

In order to formalize the procedure of reformulating problems, we will consider two different formulations of the problem: the original normally user-friendly formulation and a second formulation, into which the original formulation is to be transformed and which will then be given to the theorem proving system in order to obtain a solution. Such a transformation between formalizations is appropriate only if a proof for the second entails the existence of a proof for the original one. More precisely, we require that a proof for the original problem is constructible using the proof found for the transformed formulation. The best known methods to describe the human behaviour in mathematical problem solving are writing tactics as, for instance, those used in LCF [GMW79], Nuprl [Co86], and proof-planning [Bu88]. Tactics (or extensions of tactics) serve as the fundamental units comprising the entire problem-solving processes of such systems.

In this paper we show that tactics are also suitable as a basis for the problem transformation process outlined above. They will be used to optimize problem formulations for a resolution theorem prover by performing a preprocessing step. In the next section we formalize the notion of reformulation and in section 3 we present some examples. The examples show how reformulations can drastically improve the search behaviour of a theorem proving system. Finally, reformulation is operationally described by tactics in section 4. More details of our procedure can be found in [KePr95].

2 Basic Concepts for Problem Transformations

Since our examples are mathematical in nature, they are formulated originally in a language that is very close to higher-order logic, namely in Church's simple theory of types [Ch40]. We do not use any λ-expressions. In the following we formalize the notion of reformulation, and present a particular transformation that allows the transition from higher-order to first-order language.

[1]When using a completion-based equality theorem prover, however, it is usually better to state the problem *with* function symbols. One of our focal points is the examination of classes of problems that are well-served by the solution method of avoiding *some* function symbols.

2.1 A Description of Problem Transformations by Morphisms

In order to formally describe the translation of problem formulations we must define precisely the notions of "logic morphism" and "reformulation". A *problem formulation* is a pair $(\Gamma, \textsc{Thm})$ consisting of a formula set Γ of preconditions, and a formula $\textsc{Thm}$. The problem is to generate an inference $\Gamma \vdash \textsc{Thm}$.

Definition (Morphism of Logics): A *morphism of logics* Θ is a mapping from the signature $\mathcal{S}$ of a logic $\mathcal{F}^1(\mathcal{S})$ to the signature of a logic $\mathcal{F}^2(\Theta(\mathcal{S}))$, which maps every set of formulae in $\mathcal{F}^1(\mathcal{S})$ to a set of formulae in $\mathcal{F}^2(\Theta(\mathcal{S}))$. ■

Morphisms are extended to problem formulations in an obvious way. The source and target formulations of a transformation can be expressed in the same or in different logics. In order to relate the source and the target formulation, we consider only *sound* morphisms, in this paper.

Definition (Soundness): Let Θ be a morphism from $\mathcal{F}^1$ to $\mathcal{F}^2$. Θ is said to be *sound* iff for every formula set Γ in $\mathcal{F}^1$ the following holds: if Γ has a model in $\mathcal{F}^1$ then there is a model of $\Theta(\Gamma)$ in $\mathcal{F}^2$. ■

Assume Θ is a sound morphism. Then a solution for $(\Theta(\Gamma), \Theta(\textsc{Thm}))$ represents a solution for the original problem formulation $(\Gamma, \textsc{Thm})$, too.

To render precise the notion of a reformulation we need two additional properties of morphisms. In order to effectively perform the transformation, morphisms are required to be *computable*. While this assumption is indisputably appropriate, our second requirement, namely *retranslatability* of morphisms, is controversial. A retranslatable morphism is one with which a proof of the original problem formulation can be effectively computed from the proof of the transformed problem formulation. This property is necessary if communication of the proof to a human mathematician is desired. Our position is that of a "nominalist" [Pe91], that is, we hold the communication of a proof to be every bit as important as its actual derivation.

Definition (Reformulation): A *reformulation* is the application of a sound, computable, and retranslatable morphism to a problem formulation. ■

2.2 A Morphism from Higher-Order into First-Order Logic

Because we want to formulate mathematical problems in higher-order logic, but prove them in sorted first-order logic, we introduce a special class of morphisms, which translate higher-order expressions into the sorted first-order input logic of MKRP [OhSi91].

Definition (Quasi-Homomorphism): Let $\mathcal{F}^1(\mathcal{S}_1)$ and $\mathcal{F}^2(\mathcal{S}_2)$ be two logics. A mapping Θ that maps every formula and every term of $\mathcal{F}^1(\mathcal{S}_1)$ to a formula or to a term of $\mathcal{F}^2(\mathcal{S}_2)$, respectively, is called a *quasi-homomorphism* iff constants or variable x are mapped onto constants or variables $\Theta(x)$ and each composed term $f(t_1, \ldots, t_n)$ is either mapped homomorphical to $\Theta(f)(\Theta(t_1), \ldots, \Theta(t_n))$ or reified by $\boldsymbol{\alpha}^f(\Theta(f), \Theta(t_1), \ldots, \Theta(t_n))$, where $\boldsymbol{\alpha}^f$ stands for apply. For all other cases Θ is homomorphic. ■

In the remainder of this paper we consider injective quasi-homomorphisms from higher-order logic to sorted first-order logic since they are, in fact, reformulations (for a proof see [Ke92]).

3 Effect of Problem Transformations on Some Examples

In this section we give two examples illustrating the benefits of reformulations. We begin by describing how to read the computer generated proofs of the MKRP-system [OhSi91]. The preconditions Γ of a problem formulation are named `Axioms`, the THM is called `Theorems`. The axioms and the negated theorem are transformed into clause normal form. The names of clauses stemming from `Axioms` begin with an `A`, those from the theorem begin with a `T`. Resolvents are labelled with an `R`. The asterisk means that the clause is really used in the proof. Positive literals are marked with a `+`, negative ones with a `-`. The equality predicate is written as an infix operator in the input and as a prefix operator in the output.

3.1 Avoiding Functions by Preprocessing

Our first example shows how simple preprocessing steps on a given problem formulation can result in a significantly less difficult initial clause set to be input to the resolution theorem prover. The examined theorem is the associativity of the composition operation for relations, that is, that for all binary relations ρ, σ, τ over a fixed set, $(\rho \circ \sigma) \circ \tau = \rho \circ (\sigma \circ \tau)$. We write relations as subsets of the binary Cartesian product of the object level domain. The composition of two relations is defined by

$$\forall x, y\ (x, y) \in \rho \circ \sigma \Leftrightarrow \exists z\ (x, z) \in \rho \wedge (z, y) \in \sigma.$$

Furthermore, an extensionality axiom is needed for the proof. We translate the problem by an injective quasi-homomorphism (as described in subsection 2.2) into first-order logic and continue our considerations with the translated formulation.

3.1.1 The Initial First-Order Formulation

The "apply" predicate $\boldsymbol{\alpha}$ from 2.2 is written as `@`. The type of binary relations is denoted by `REL` and the type of individuals by `DOM`. Instead of $(x, y) \in \rho$ we use the notion $\rho(x, y)$, which is translated into the first-order MKRP-atom `@(RHO X Y)`.

```
Axioms: SORT DOM,REL:ANY               * Sorts *
        TYPE @(REL DOM DOM)
        TYPE COMP(REL REL):REL         * Definition of Composition *
        ALL RHO,SIG:REL ALL X,Y:DOM (EX Z:DOM @(RHO X Z) AND @(SIG Z Y))
                                    EQV  @(COMP(RHO SIG) X Y)
        * Extensionality Axiom *
        ALL RHO,SIG:REL (ALL X,Y:DOM @(RHO X Y) EQV @(SIG X Y)) IMPL RHO = SIG
Theorems: ALL RHO,SIG,TAU:REL COMP(COMP(RHO SIG) TAU) = COMP(RHO COMP(SIG TAU))
```

3.1.2 Normalization of the Original Formulation

If we input this formulation to the MKRP-system we obtain the following clauses with the harmless-looking Skolem functions `f2` and `f3`:

```
  A1:  All x:Any + =(x x)
* A2:  All x,y:REL z,u:DOM + @(y u f1(y z u x))  - @(comp(y x) u z)
* A3:  All x,y:DOM z,u:REL + @(u f1(z y x u) y)  - @(comp(z u) x y)
* A4:  All x,y,z:DOM u,v:REL - @(v z y)  - @(u y x)  + @(comp(v u) z x)
* A5:  All x,y:REL - @(y f2(y x) f3(y x))  - @(x f2(y x) f3(y x))  + =(y x)
* A6:  All x,y:REL + @(y f2(y x) f3(y x))  + @(x f2(y x) f3(y x))  + =(y x)
* T7:  - =(comp(comp(c1 c2) c3) comp(c1 comp(c2 c3)))
```

The system needs 1115 seconds (on a UX1200 Symbolics machine) to generate a proof with very complicated clauses.

3.1.3 Reformulation

Looking at the input formulae and at the result of the normalization we see the possibility for a preprocessing step. The structure of the last axiom formula is $(A \Leftrightarrow B) \Rightarrow C$ where C matches the theorem C'. We can therefore construct a new theorem $A' \Leftrightarrow B'$ according to the match. Starting with this "resolvent" as the theorem we can avoid the unfolding during normalization, and can additionally split the theorem into the two parts $A' \Rightarrow B'$ and $B' \Rightarrow A'$ with computation times for the split parts of 18 and 13 seconds, respectively.

But this splitting is not the main reason for the improvement in performance (time without splitting: 114 seconds). The problem with the first formulation is the unfolding during normalization:

$$
\begin{array}{rl}
 & \boxed{\forall}\,(\,\boxed{\forall}\,A \Leftrightarrow B) \Rightarrow C \\
\rightsquigarrow & \boxed{\forall}\,\neg(\,\boxed{\forall}\,A \Leftrightarrow B) \vee C \\
\rightsquigarrow & \boxed{\forall}\,\neg((\,\boxed{\forall}\,A \Rightarrow B) \wedge (\,\boxed{\forall}\,B \Rightarrow A)) \vee C \\
\rightsquigarrow & \boxed{\forall}\,\neg(\,\boxed{\forall}\,\neg A \vee B) \vee \neg(\,\boxed{\forall}\,\neg B \vee A) \vee C \\
\rightsquigarrow & \boxed{\forall}\,(\,\boxed{\exists}\,\neg(\neg A \vee B)) \vee (\,\boxed{\exists}\,\neg(\neg B \vee A)) \vee C \\
\rightsquigarrow & \boxed{\forall}\,(\,\boxed{\exists}\,A \wedge \neg B) \vee (\,\boxed{\exists}\,B \wedge \neg A) \vee C \\
\rightsquigarrow & \boxed{\forall}\,(\neg A' \vee \neg B' \vee C) \wedge (A' \vee B' \vee C)
\end{array}
$$

The last formula corresponds to the clauses `A5` and `A6` above. There the A's and B's contain troublesome Skolem functions, introduced in the last step because of quantifiers in the theorem, which must be resolved in a difficult manner. By the preprocessing step we replace the three-literal clauses `A5` and `A6`, as well as the theorem clause `T7`, by the simple unit clauses `T5` through `T8` in the proof below and hence avoid the Skolem functions `f2` and `f3`. Really complex clauses are no longer possible, because these Skolem functions are replaced by the Skolem constants `c3` and `c5` (or `c8` and `c10` for the second split part, respectively), which cannot, of course, be nested. In the formulation below one can immediately see the five variables that are Skolemized.

The general *explicit* formulation of the extensionality axiom is replaced by a special *implicit* one. The alternative formulation together with the clause set is as follows:

```
Axioms:   SORT DOM,REL:ANY              * Sorts *
          TYPE @(REL DOM DOM)
          TYPE COMP(REL REL):REL        * Definition of Composition *
          ALL RHO,SIG:REL ALL X,Y:DOM
              (EX Z:DOM @(RHO X Z) AND @(SIG Z Y))  EQV  @(COMP(RHO SIG) X Y)
Theorems: ALL RHO,SIG,TAU:REL ALL X,Y:DOM
          @(COMP(COMP(RHO SIG) TAU) X Y)  EQV  @(COMP(RHO COMP(SIG TAU)) X Y)

Set of Axiom Clauses Resulting from Normalization
  A1: All x:Any + =(x x)
* A2: All x,y:REL z,u:DOM + @(y u f1(y z u x))  - @(comp(y x) u z)
* A3: All x,y:DOM z,u:REL + @(u f1(z y x u) y)  - @(comp(z u) x y)
* A4: All x,y,z:DOM u,v:REL - @(v z y)  - @(u y x)  + @(comp(v u) z x)

Set of Theorem Clauses Resulting from Normalization and Splitting
Splitpart 1  * T5: - @(comp(comp(c4 c1) c2) c3 c5)
             * T6: + @(comp(c4 comp(c1 c2)) c3 c5)
Splitpart 2  * T7: + @(comp(comp(c9 c6) c7) c8 c10)
             * T8: - @(comp(c9 comp(c6 c7)) c8 c10)
```

3.2 Explicit versus Predicative Formalization

Our second example deals with a more fundamental change of representation that relies on an explicit formulation. Again, the explicit statement of the problem is closer to a textbook formulation, it can be instantiated in the examined case to a very simple formulation. The task is to prove that the intersection of two equivalence relations is also an equivalence relation.

3.2.1 Higher-Order Formulation

The problem is easily stated in higher-order logic by the following formulae:

Definition of Intersection: $\forall\rho_{(\iota\times\iota\to o)},\sigma_{(\iota\times\iota\to o)}\ \ \forall a_\iota,b_\iota\ \ (\rho\cap\sigma)(a,b)\Leftrightarrow\rho(a,b)\ \wedge\ \sigma(a,b)$

Definition of Reflexivity: $\forall\rho_{(\iota\times\iota\to o)}\ \ \mathrm{ref}(\rho)\Leftrightarrow(\forall a_\iota\ \ \rho(a,a))$

Definition of Symmetry: $\forall\rho_{(\iota\times\iota\to o)}\ \ \mathrm{sym}(\rho)\Leftrightarrow(\forall a_\iota,b_\iota\ \ \rho(a,b)\Rightarrow\rho(b,a))$

Definition of Transitivity: $\forall\rho_{(\iota\times\iota\to o)}\ \ \mathrm{trans}(\rho)\Leftrightarrow(\forall a_\iota,b_\iota,c_\iota\ \ \rho(a,b)\wedge\rho(b,c)\Rightarrow\rho(a,c))$

Definition of Equivalence Relation: $\forall\rho_{(\iota\times\iota\to o)}\ \ \mathrm{eqv}(\rho)\Leftrightarrow\mathrm{ref}(\rho)\wedge\mathrm{sym}(\rho)\wedge\mathrm{trans}(\rho)$

Theorem: $\forall\rho_{(\iota\times\iota\to o)},\sigma_{(\iota\times\iota\to o)}\ \ \mathrm{eqv}(\rho)\wedge\mathrm{eqv}(\sigma)\Rightarrow\mathrm{eqv}(\rho\cap\sigma)$

3.2.2 First-Order Formulation

As with our first example, these higher-order formulae are translated by an injective quasi-homomorphism (compare subsection 2.2) into the MKRP input language.

```
Axioms:   SORT DOM,REL:ANY              * Formulation with Variable Relations *
          TYPE @(REL DOM DOM)
          TYPE INTER(REL REL):REL       * Definition of Intersection *
          ALL RHO,SIG:REL ALL A,B:DOM @(INTER(RHO SIG) A B) EQV @(RHO A B) AND @(SIG A B)
          TYPE REF(REL)                 * Definition of Reflexivity *
          ALL RHO:REL REF(RHO) EQV (ALL A:DOM @(RHO A A))
          TYPE SYM(REL)                 * Definition of Symmetry *
          ALL RHO:REL SYM(RHO) EQV (ALL A,B:DOM @(RHO A B) IMPL @(RHO B A))
          TYPE TRANS(REL)               * Definition of Transitivity *
          ALL RHO:REL TRANS(RHO) EQV
           (ALL A,B,C:DOM @(RHO A B) AND @(RHO B C) IMPL @(RHO A C))
          TYPE EQ.REL(REL)              * Definition of Equivalence Relation *
          ALL RHO:REL EQ.REL(RHO) EQV REF(RHO) AND SYM(RHO) AND TRANS(RHO)
Theorems: ALL RHO,SIG:REL EQ.REL(RHO) AND EQ.REL(SIG) IMPL EQ.REL(INTER(RHO SIG))
```

3.2.3 Reformulation

The following variant of the above problem is expressed with constant predicates. Instead of proving the theorem for all ρ and σ we show it for arbitrary new constants ρ and σ according to the inference rule "AE" (All-Einführung, Universal Generalization) of Gentzen's natural deduction calculus [Ge35]. After the expansion of all definitions we obtain a formula set that is entirely first-order, with the exception of the intersection function whose domain consists of predicates. In order to eliminate this function symbol we introduce a new predicate constant RHOSIG for the term $\rho \cap \sigma$. The explicit definitions of the original formulation are now implicitly given in the theorem. We give the problem formulation of this reformulation whose proof consists of six trivial splitparts.

```
Axioms:   SORT DOM:ANY              * Formulation with Constant Relations *
          TYPE RHO(DOM DOM)
          TYPE SIG(DOM DOM)
          TYPE RHOSIG(DOM DOM)
          ALL A,B:DOM RHOSIG(A B) EQV RHO(A B) AND SIG(A B)
Theorems:    ((ALL A:DOM     RHO(A A))
          AND (ALL A,B:DOM   RHO(A B) IMPL RHO(B A))
          AND (ALL A,B,C:DOM RHO(A B) AND RHO(B C) IMPL RHO(A C))
          AND (ALL A:DOM     SIG(A A))
          AND (ALL A,B:DOM   SIG(A B) IMPL SIG(B A))
          AND (ALL A,B,C:DOM SIG(A B) AND SIG(B C) IMPL SIG(A C)))
       IMPL  ((ALL A:DOM     RHOSIG(A A))
          AND (ALL A,B:DOM   RHOSIG(A B) IMPL RHOSIG(B A))
          AND (ALL A,B,C:DOM RHOSIG(A B) AND RHOSIG(B C) IMPL RHOSIG(A C)))
```

3.2.4 Proof Statistics

In order to prove that the intersection of two equivalence relations is again an equivalence relation, we have presented two different formulations of the problem – a relatively direct translation of the higher-order formulation and a more subtle formulation. Runtime behaviour of the MKRP-system (measured in seconds) for different option settings is compared in the following table. "Depth" is the maximally allowed term depth in the generated clauses. "Splitting" means that the theorem may be divided into several parts for the proof. The "Terminator" is a special proof tool for unit resolution [AnOh83]. The term depth has not been limited for the runs using the terminator.

	Depth			Depth with Splitting			Terminator	
	∞	2	1	∞	2	1	Standard	Splitting
Variant 1	∞	2105	unsolvable	269	65	22	∞	10
Variant 2	46	46	47	5	5	5	23	5

In all settings, the second variant is superior to the first one. The difficulty with the first formulation results primarily from the possibility of nesting the intersection function. Note that for this example, the first variant has an infinite search space, while the search space of the second is finite.

Although the two examples suggest that eliminating function symbols leads to more efficient deductions, this need not always be the case. Sometimes precisely this elimination method is used to construct sets of test examples with increasing complexity for theorem provers. Consider, for example, the pigeonhole problem as presented in [Pe86, example 72], which can be proved in higher-order logic for all $n \in \mathbb{N}$, but which requires increasing amount of resources in propositional logic for increasing n (for details see [KePr95]).

As it is not possible in general to determine the best formulation of a problem in advance, it is necessary to decide heuristically whether or not reformulation is adequate. For instance, a propositional logic formulation should be avoided when it is very large compared to the corresponding first-order formulation.

4 Tactics for Problem Transformations

In order to operationalize our morphisms we use the tactic formalism, which was invented during the development of the LCF proof system [GMW79]. In LCF and other advanced proof-checking systems like Nuprl [Co86], tactics are basically abbreviations for sequences of calculus rules. Whereas all operations in these systems are performed in the same logical language, for us a tactic is just the operationalization of a reformulation, which may link two different languages.

In Nuprl *elementary tactics* correspond to the application of calculus rules. More powerful *composed tactics* are obtained using *tacticals* to combine tactics: **IF** *Expression* **THEN** *Tactic*, **IF** *Expression* **THEN** $\textit{Tactic}_1$ **ELSE** $\textit{Tactic}_2$, **REPEAT** *Tactic*, **WHILE** *Expression* **DO** *Tactic*, **COMPOSITION** $\textit{Tactic}_1 \cdots \textit{Tactic}_n$. Originally *Expression* is a boolean expression written in the programming language ML. We use an informal logical language below to specify the expressions that are necessary to specify the problem transformations.

4.1 Tactics for the Examples

Informally a tactic transforming our first example from its initial formulation into the form that is more appropriate for resolution might be: If certain conditions are fulfilled (namely, a theorem THM is to be proved from a formula set Γ, the theorem is an atom, Γ contains an implication, in which the antecedent is not an atom and the succedent matches the theorem, and the predicate of the theorem does not occur anywhere else in the problem formulation), then the theorem and one of the axioms are replaced by a simpler formula. The replacement corresponds to the backward application of modus ponens (modulo matcher), where the implication is valid.

```
ELIM_THM_PRED :=
IF      $\text{TO\_PROVE}(\Gamma \vdash \text{THM})$
        $\wedge\ \text{ATOM}(\text{THM})$
        $\wedge\ \exists \text{AX}\ \text{AX} \in \Gamma\ \wedge\ \text{AX} = (\mathit{Antecedent} \Rightarrow \mathit{Succedent})$
        $\wedge\ \neg\ \text{ATOM}(\mathit{Antecedent})$
        $\wedge\ \exists \sigma : \mathit{Matcher}\ \text{THM}\ = \sigma(\mathit{Succedent})$
        $\wedge\ \neg\text{DOES\_OCCUR}(\text{PREDICATE}(\text{THM}), (\Gamma - \{\text{AX}\}\ \cup\ \{\mathit{Antecedent}\})$
THEN    $\text{TO\_PROVE}(\Gamma - \{\text{AX}\} \vdash \sigma(\mathit{Antecedent}))$
```

We neglect the existence of universal quantifiers in this description of the tactic. Of course, the variables in the domain of the matcher must be universally quantified in the corresponding formula.

In the second example several different tactics are used to obtain the final formulation of the problem from the initial first-order one. The first tactic serves to replace universally quantified variables in the theorem by constants:

```
ELIM_THM_VARS := IF    $\text{TO\_PROVE}(\Gamma \vdash \text{THM})$
                       $\wedge\ \text{THM}\ =\ \forall x_1, \ldots, x_n\ \varphi$
                       $\wedge\ \sigma := \{x_1 \leftarrow c_1, \ldots, x_n \leftarrow c_n\}$
                       $\wedge\ \forall i, j\ i, j \in \{1, \ldots, n\} \wedge c_i = c_j \Longrightarrow i = j$
                       $\wedge\ \forall c\ c \in \{c_1, \ldots, c_n\} \Longrightarrow \neg\text{DOES\_OCCUR}(c, \Gamma \cup \{\varphi\})$
                 THEN  $\text{TO\_PROVE}(\Gamma \vdash \sigma(\varphi))$
```

After the application of this tactic to the initial problem formulation (compare subsection 3.2.2), the problem is transformed into the following one (where the dots mean that the rest of the problem formulation is unchanged):

```
Axioms:   ...
          TYPE RHO,SIG:REL
          ...
Theorems: EQ.REL(RHO) AND EQ.REL(SIG) IMPL EQ.REL(INTER(RHO SIG))
```

The second tactic is used to expand the definitions occurring in the preconditions:

```
EXPAND_DEF := IF    $\text{TO\_PROVE}(\Gamma \vdash \text{THM})$
                    $\wedge\ \exists \text{AX}\ \text{AX} \in \Gamma\ \wedge\ \text{AX} = \forall x_1, \ldots, x_n\ P(x_1, \ldots, x_n) \Leftrightarrow \varphi$
                    $\wedge\ \neg\text{DOES\_OCCUR}(P, \varphi)$
              THEN  $\text{TO\_PROVE}(\text{SUBST\_ALL}(P, \varphi, \Gamma - \{\text{AX}\} \vdash \text{THM}))$
```

for a tactic SUBST_ALL(*pred, formula, in*) replacing all occurrences of the predicate P in the form $P(t_1, \ldots, t_n)$ by $\sigma(\varphi)$, where σ is the matcher $\sigma = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$, which matches the occurrence of P in *formula* with the definition of P.

Repeated application of this tactic SUBST_ALL yields a new problem formulation, where the definitions of `REF`, `SYM`, `TRANS`, and `EQ.REL` are expanded. (Note, that the definition of intersection is not a "simple" definition and cannot be expanded by this tactic):

```
Axioms:   SORT DOM,REL:ANY
          TYPE @(REL DOM DOM)
          TYPE RHO,SIG:REL
          * Definition of Intersection *
          TYPE INTER(REL REL):REL
          ALL RHO,SIG:REL ALL A,B:DOM @(INTER(RHO SIG) A B)
                                  EQV @(RHO A B) AND @(SIG A B)
Theorems: (ALL A:DOM @(RHO A A)) AND
          (ALL A,B:DOM @(RHO A B) IMPL @(RHO B A)) AND
          (ALL A,B,C:DOM @(RHO A B) AND @(RHO B C) IMPL @(RHO A C))
     AND  (ALL A:DOM @(SIG A A)) AND
          (ALL A,B:DOM @(SIG A B) IMPL @(SIG B A)) AND
          (ALL A,B,C:DOM @(SIG A B) AND @(SIG B C) IMPL @(SIG A C))
     IMPL (ALL A:DOM @(INTER(RHO SIG) A A)) AND
          (ALL A,B:DOM @(INTER(RHO SIG) A B) IMPL @(INTER(RHO SIG) B A)) AND
          (ALL A,B,C:DOM @(INTER(RHO SIG) A B) AND @(INTER(RHO SIG) B C)
                     IMPL @(INTER(RHO SIG) A C))
```

The third tactic is used to instantiate universally quantified variables by constants:

```
INST_VARS :=
IF      $\text{TO\_PROVE}(\Gamma \vdash \text{THM})$
        $\wedge\ \exists t\ \text{DOES\_OCCUR}(t, \text{THM}) \wedge \text{GROUND}(t) \wedge t = f(t_1, \ldots, t_n)$
          $\Longrightarrow \forall s\ \text{DOES\_OCCUR}(s, \text{THM}) \wedge s = f(s_1, \ldots, s_n) \Longrightarrow s = t$
              $\wedge\ \forall \text{AX}\ \text{AX} \in \Gamma \Longrightarrow \forall s\ \text{DOES\_OCCUR}(s, \text{AX}) \wedge s = f(s_1, \ldots, s_n)$
                $\Longrightarrow \text{AX} = (\forall x_1, \ldots, x_n\ \varphi) \wedge s_1 = x_1 \wedge \ldots \wedge s_n = x_n$
        $\wedge\ \sigma := \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$
THEN    $\text{TO\_PROVE}(\text{SUBST}(\forall x_1, \ldots, x_n\ \varphi, \sigma(\varphi), \Gamma) \vdash \text{THM})$
```

where SUBST(*from to in*) replaces all occurrences of *from* by *to* in *in*.

The term t that must occur in the theorem according to the specification of the tactic is `INTER(RHO SIG)`. Hence the universally quantified variables `RHO` and `SIG` in the axiom can be instantiated to the constants `RHO` and `SIG` of the theorem.

The fourth tactic replaces the `@` constructs by newly generated predicates:

```
ELIM_APPLY :=
IF      $\text{TO\_PROVE}(\Gamma \vdash \text{THM})$
        $\wedge\ \forall \varphi\ \text{ATOM}(\varphi) \wedge \text{DOES\_OCCUR}(\varphi, \Gamma \cup \{\text{THM}\}) \wedge \varphi = @(s, s_1, \ldots, s_n)$
          $\Longrightarrow \text{GROUND}(s)$
THEN    $\text{TO\_PROVE}(\text{SUBST}(@(s, s_1, \ldots, s_n), s(s_1, \ldots, s_n), \Gamma \vdash \text{THM}))$
```

with the ground terms converted to new predicate symbols s. In applying the fourth tactic, the ground term `INTER(RHO SIG)` is replaced by a new constant `RHOSIG`. Moreover, the `@`-constructs are eliminated using new predicates instead of constants. This results in the reformulation of subsection 3.2.3.

By building the presented tactics together we get the following composed tactic, which performs the reformulation of example 3.1:

```
ELIM := COMPOSITION ELIM_THM_VARS
                    REPEAT EXPAND_DEF
                    INST_VARS
                    ELIM_APPLY
```

The tactic INST_VARS is rather complicated but very general in nature. In our context this tactic is only applied to generate the preconditions for the ELIM_APPLY-tactic. The tactic ELIM_APPLY should be applied when the “Apply”-construct appears only in the presence of universal quantification.

An alternative procedure would be to begin the reformulation process with a higher-order formulation and to use the following compound tactic, thus avoiding the tactics INST_VARS and ELIM_APPLY:

```
ELIM := COMPOSITION ELIM_THM_VARS
                    REPEAT EXPAND_DEF
                    TRANSLATE
```

This is in contrast to the first procedure, where the translation is carried out as the first step and then the ELIM tactic is applied.

4.2 Soundness Considerations

Of course the correctness of all involved tactics has to be proved. But as in, for example, Nuprl, this follows for most of our tactics immediately from the fact that they are iterations of calculus rules: the tactic ELIM_THM_PRED is a combination of instantiation and modus ponens, ELIM_THM_VARS is universal generalization, EXPAND_DEF combines instantiation and substitution, and INST_VARS is just a restricted form of instantiation.

TRANSLATE is a transition between different logics. The correctness is ensured by the theorem cited at the end of subsection 2.2. Applying the tactic ELIM_APPLY preserves the logic of the problem, but changes its signature. This change is not essential, because the proof steps can be mapped injectively. All $@$ appear together with constants c in the form $@(c, \ldots)$ and are transformed to $c(\ldots)$. Such transformations are possible for all proof steps.

5 Conclusion

We have shown how tactics can be used to reformulate problem descriptions so that they are stated more appropriately for solution by a resolution theorem prover. Reformulated problems can often be solved much more efficiently. The examples given above support the thesis that a resolution-based system alone is not adequate as a tool in the daily work of a human mathematician.

Perhaps one day the automatic generation of tactics from new examples will be possible by exploiting the knowledge represented in rules, which are encoded in such a system. In that case, it would be possible to distinguish classes of problem formulations and reformulations. It is our contention that the explicit formulation of such tactics reveals the structure and contents of the otherwise implicitly coded know-how.

References

[AnOh83] G. Antoniou and H. J. Ohlbach, Terminator, *Proceedings of the 8th IJCAI*, Karlsruhe, Germany, 1983, pp. 916–919.

[Bu83] A. Bundy, *The Computer Modelling of Mathematical Reasoning*, Academic Press, 1983.

[Bu88] A. Bundy, The use of explicit plans to guide inductive proofs, *Proceedings of the 9th CADE*, Argonne, Illinois, USA, 1988, pp. 111–120.

[Co86] R.L. Constable et al., *Implementing Mathematics with the Nuprl Proof Development System*, Prentice Hall, 1986.

[Ch40] A. Church, A formulation of the simple theory of types, *Journal of Symbolic Logic*, Vol. 5, 1940, pp. 56–68.

[Ge35] G. Gentzen, Untersuchungen über das logische Schließen I & II, *Mathematische Zeitschrift*, Vol. 39, 1935, pp. 176–210, 572–595.

[GMW79] M. Gordon, R. Milner, and C. Wadsworth, *Edinburgh LCF: A Mechanized Logic of Computation*, Springer Verlag, 1979.

[HKK+94] X. Huang, M. Kerber, M. Kohlhase, E. Melis, D. Nesmith, J. Richts, and J. Siekmann, Ω-MKRP – a proof development environment, *Proceedings of the 12th CADE*, Nancy, France, 1994, pp. 788–792.

[Ke92] M. Kerber, *On the Representation of Mathematical Concepts and their Translation into First Order Logic*, PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, Germany, 1992.

[KePr95] M. Kerber and A. Präcklein, Using tactics to reformulate formulae for resolution theorem proving, *Annals of Mathematics and Artificial Intelligence*, forthcoming.

[McCa64] J. McCarthy, A tough nut for proof procedures, AI Project Memo 16, Stanford University, Stanford, California, USA, 1964.

[OhSi91] H. J. Ohlbach and J. H. Siekmann, The Markgraf Karl Refutation Procedure, in *Computational Logic – Essays in Honor of Alan Robinson*, J.-L. Lassez and G. Plotkin, eds., MIT Press, 1991, pp. 41–112.

[Pó65] G. Pólya, *Mathematical Discovery – On Understanding, Learning, and Teaching Problem Solving*, Princeton University Press, 1962/1965.

[Pe86] F. J. Pelletier, Seventy-five problems for testing automatic theorem provers, *Journal of Automated Reasoning*, Vol. 2, 1986, pp. 191–216.

[Pe91] F. J. Pelletier, The philosophy of automated theorem proving, *Proceedings of the 12th IJCAI*, Sydney, Australia, 1991, pp. 538–543.

[WOLB84] L. Wos, R. Overbeek, E. Lusk, and J. Boyle, *Automated Reasoning – Introduction and Applications*, Prentice Hall, 1984.