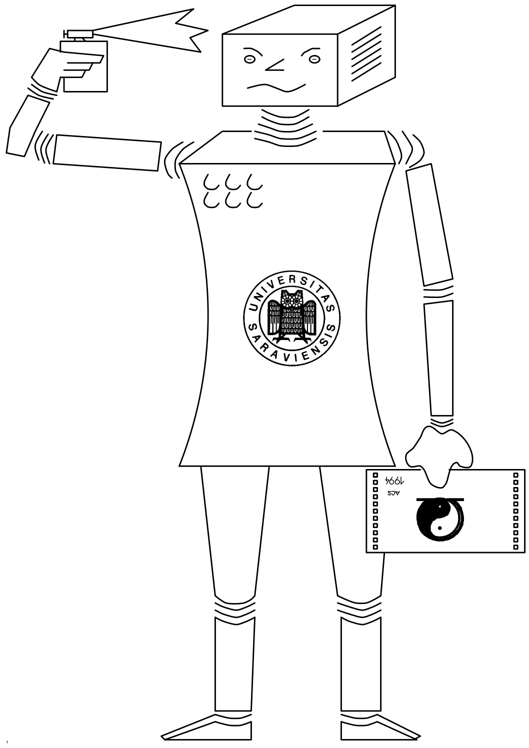
UNIVERSITÄT DES SAARLANDES
FACHBEREICH INFORMATIK
D-66041 SAARBRÜCKEN
GERMANY
WWW: http://js-sfbsun.cs.uni-sb.de/pub/www/



SEKI Report

Tactics for the Improvement of Problem Formulation in Resolution-Based Theorem Proving

Manfred Kerber and Axel Präcklein

SEKI Report SR-92-09

Tactics for the Improvement of Problem Formulation in Resolution-Based Theorem Proving

Manfred Kerber
kerber@cs.uni-sb.de

Axel Präcklein
prckln@cs.uni-sb.de

Fachbereich Informatik,
Universität des Saarlandes,
6600 Saarbrücken,
Germany

Abstract: We transform a user-friendly formulation of a problem to a machine-friendly one exploiting the variability of first-order logic to express facts. The usefulness of tactics to improve the presentation is shown with several examples. In particular it is shown how tactical and resolution theorem proving can be combined.

Keywords: problem formulation, theorem proving, tactics, resolution

Contents

1 Introduction

Solving mathematical problems with resolution based theorem proving systems requires in most cases intelligence and ingenuity on the part of the user, because 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 least, because not the original task is solved but a transformed one. The user modifies the problem until it can be solved, however, this reformulation efforts are never documented. The consequence could be to forbid changes of the problem representation and to compute the solution for the original formulation (whatever that might be). But this would only be reasonable to check special features of a theorem prover but not its problem-solving facility in general. When comparing the situation to that of a human mathematician it seems obvious that he also rephrases the problem. Often the process terminates when the question is transformed into a form similar to a known problem.

In fact this procedure was suggested by George Pólya in his course on human mathematical problem solving [21, 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.*" Alan Bundy [4, p.91] also pleads for modifications of the problems in order to get adequate formulations for computers: "*... we should look at alternative axiomatizations of mathematical theories, which utilize a language closer to the one with which we ourselves describe the problem.*" We do not agree with Bundy's hypothesis to restate problems in a human oriented language, but we share his opinion that the representation is very important for the solution. Bundy also gives practical hints how reformulations can be done to make problems more digestible for theorem proving programs [4, section 4.3]. Especially he elaborates on the advantage of avoiding *all* function symbols. One of our focal points is to examine classes of problems well-suited to the method of avoiding *some* function symbols. Bundy cites an example where the function symbols are eliminated. But using a completion-based equality theorem prover it is better to state it with function symbols. The main concern of this paper is to make explicit some informal methods like those which are elaborated by Larry Wos et al. [26, chapter 4] to represent various exercises for automated reasoning programs.

In order to by-pass the cheat mentioned above, it is necessary to have two different formulations of the problem: one original (user-friendly) formulation, which is transformed to a second logically sufficient formulation, to be given to the theorem proving system in order to be proved. This transformation is of course a process that requires intelligence. Hence it is an object of study in artificial intelligence. The best known methods to describe the human behaviour in mathematical problem solving are meta-reasoning and writing tactics as those used in LCF [11], Nuprl [8], and proof-planning [5, 6]. The transformation process presented here fits well into this framework.

2 Example for Different Normalizations

Our first example is presented in order to show that a simple preprocessing step on an original formulation can result in a significantly less difficult initial clause set to be given to the resolution theorem prover. The considered theorem is, that the composition operation for relations is associative, in other words, for all binary relations ρ, σ, τ over a fixed set: $(\rho \circ \sigma) \circ \tau = \rho \circ (\sigma \circ \tau)$ (the example is taken from [9, p.34]).

We write relations as subsets of the binary Cartesian product of the object level domain. The composition of two relations is defined by

$$\forall x, y\ (x, y) \in \rho \circ \sigma \leftrightarrow \exists z\ (x, z) \in \rho \land (z, y) \in \sigma.$$

We translate the problem by a standard method into first-order logic using "apply" and a so-called extensionality axiom as introduced in section 3. We continue our considerations with the translated formulation, because we use a first-order theorem prover (Markgraf Karl Refutation Procedure [18]) and the reformulation can be described in first-order, although higher-order is more adequate as starting point. For the second example we introduce some basic concept of higher-order logic, which are also necessary for investigations about the soundness of the procedure (compare section 5).

2.1 The Initial First-Order Formulation

The "apply"-predicate is written as `APPLY`. `IXITO` is the type of binary relations (`I` stands for ι, the type of individuals, `O` for o, the type of truth values, compare section 3). Instead of $(x, y) \in \rho$ we use the notion $\rho(x, y)$, which is translated into the first-order atom `APPLY(RHO X Y)`.

```
Formulae given to the editor
============================

Axioms:   * Sorts *
          SORT I,IXITO:ANY
          TYPE APPLY(IXITO I I)
          * Definition of Composition *
          TYPE COMP(IXITO IXITO):IXITO
          ALL RHO,SIGMA:IXITO ALL X,Y:I
              (EX Z:I APPLY(RHO X Z) AND APPLY(SIGMA Z Y))  EQV  APPLY(COMP(RHO SIGMA) X Y)
          * Extensionality Axiom *
          ALL RHO,SIGMA:IXITO  (ALL X,Y:I APPLY(RHO X Y) EQV APPLY(SIGMA X Y))  IMPL  RHO = SIGMA

Theorems: ALL RHO,SIGMA,TAU:IXITO  COMP(COMP(RHO SIGMA) TAU) = COMP(RHO COMP(SIGMA TAU))
```

2.2 Normalization of the Original Formulation

If we input this formulation into the Markgraf Karl Refutation Procedure (MKRP) (for the language see [25, 22]) we get:

```
  A1:  All x:Any + =(x x)
* A2:  All x,y:Ixito z,u:I + APPLY(y u f_1(y z u x))  - APPLY(comp(y x) u z)
* A3:  All x,y:I z,u:Ixito + APPLY(u f_1(z y x u) y)  - APPLY(comp(z u) x y)
* A4:  All x,y,z:I u,v:Ixito - APPLY(v z y)  - APPLY(u y x)  + APPLY(comp(v u) z x)
* A5:  All x,y:Ixito - APPLY(y f_2(y x) f_3(y x))  - APPLY(x f_2(y x) f_3(y x))  + =(y x)
* A6:  All x,y:Ixito + APPLY(y f_2(y x) f_3(y x))  + APPLY(x f_2(y x) f_3(y x))  + =(y x)
* T7:  - =(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
```

The system needs 1115 seconds to generate a proof with very complicated clauses. One of the last deduced clauses is depicted below[1]:

```
* R152:  - APPLY(c_2
                 f_1(c_1
                     f_3(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
                     f_2(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
                     comp(c_2 c_3))
                 f_1(c_2
                     f_3(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
                     f_1(c_1
                         f_3(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
                         f_2(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
                         comp(c_2 c_3))
                     c_3))
         - APPLY(c_3
                 f_1(c_2
                     f_3(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
                     f_1(c_1
                         f_3(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
                         f_2(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
                         comp(c_2 c_3))
                     c_3)
                 f_3(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3))))
```

2.3 Better Normalization

Looking on the input formulae and on the result of the normalization we see that it will be very useful to do a preprocessing step. The structure of the last axiom formula is $(A \leftrightarrow B) \rightarrow C$ where C matches the theorem C'. Hence we can construct a new theorem $A' \leftrightarrow B'$ according to this match. Starting with this "resolvent" as 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 splitparts of 18 and 13 seconds, respectively.

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

$$
\begin{array}{rl}
 & \boxed{\forall}\ (\ \boxed{\forall}\ A \leftrightarrow B) \rightarrow C \\
\rightsquigarrow & \boxed{\forall}\ \neg(\ \boxed{\forall}\ A \leftrightarrow B) \vee C \\
\rightsquigarrow & \boxed{\forall}\ \neg((\ \boxed{\forall}\ A \rightarrow B) \wedge (\ \boxed{\forall}\ B \rightarrow A)) \vee C \\
\rightsquigarrow & \boxed{\forall}\ \neg(\ \boxed{\forall}\ \neg A \vee B) \vee \neg(\ \boxed{\forall}\ \neg B \vee A) \vee C \\
\rightsquigarrow & \boxed{\forall}\ (\ \boxed{\exists}\ \neg(\neg A \vee B)) \vee (\ \boxed{\exists}\ \neg(\neg B \vee A)) \vee C \\
\rightsquigarrow & \boxed{\forall}\ (\ \boxed{\exists}\ A \wedge \neg B) \vee (\ \boxed{\exists}\ B \wedge \neg A) \vee C \\
\rightsquigarrow & \boxed{\forall}\ (\neg A \vee \neg B \vee C) \wedge (A \vee B \vee C)
\end{array}
$$

[1] The asterisk means that the clause is really used in the proof. - indicates negative and + positive literals.

The last formula corresponds to A5 and A6 in section 2.2. We have two clauses, where the As and Bs contain troublesome Skolem functions introduced in the last step because of quantifiers in the theorem and 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 f_2 and f_3. They are replaced by Skolem constants. In the formulation below one can see immediately 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 a complete proof of one of the splitparts follows.

```
Formulae given to the editor
============================

Axioms:   * Sorts *
          SORT I,IXITO:ANY
          TYPE APPLY(IXITO I I)
          * Definition of Composition *
          TYPE COMP(IXITO IXITO):IXITO
          ALL RHO,SIGMA:IXITO ALL X,Y:I
              (EX Z:I APPLY(RHO X Z) AND APPLY(SIGMA Z Y))  EQV  APPLY(COMP(RHO SIGMA) X Y)

Theorems: ALL RHO,SIGMA,TAU:IXITO ALL X,Y:I
          APPLY(COMP(COMP(RHO SIGMA) TAU) X Y)  EQV  APPLY(COMP(RHO COMP(SIGMA TAU)) X Y)

Set of Axiom Clauses Resulting from Normalization
=================================================

  A1: All x:Any + =(x x)
* A2: All x,y:Ixito z,u:I + APPLY(y u f_1(y z u x))  - APPLY(comp(y x) u z)
* A3: All x,y:I z,u:Ixito + APPLY(u f_1(z y x u) y)  - APPLY(comp(z u) x y)
* A4: All x,y,z:I u,v:Ixito - APPLY(v z y)  - APPLY(u y x)  + APPLY(comp(v u) z x)

Set of Theorem Clauses Resulting from Normalization and Splitting
=================================================================

Splitpart 1  * T5: - APPLY(comp(comp(c_4 c_1) c_2) c_3 c_5)
             * T6: + APPLY(comp(c_4 comp(c_1 c_2)) c_3 c_5)

Splitpart 2  * T7: + APPLY(comp(comp(c_9 c_6) c_7) c_8 c_10)
             * T8: - APPLY(comp(c_9 comp(c_6 c_7)) c_8 c_10)

Refutation of Splitpart 1:
==========================

Initial Clauses: A1: All x:Any + =(x x)
               * A2: All x,y:Ixito z,u:I + APPLY(y u f_1(y z u x))
                        - APPLY(comp(y x) u z)
               * A3: All x,y:I z,u:Ixito + APPLY(u f_1(z y x u) y)
                        - APPLY(comp(z u) x y)
               * A4: All x,y,z:I u,v:Ixito
                            - APPLY(v z y)  - APPLY(u y x)
                            + APPLY(comp(v u) z x)
               * T5: - APPLY(comp(comp(c_4 c_1) c_2) c_3 c_5)
               * T6: + APPLY(comp(c_4 comp(c_1 c_2)) c_3 c_5)


T6,1 & A3,2   --> * R1: + APPLY(comp(c_1 c_2) f_1(c_4 c_5 c_3 comp(c_1 c_2)) c_5)
```

```
T6,1 & A2,2   --> * R2: + APPLY(c_4 c_3 f_1(c_4 c_5 c_3 comp(c_1 c_2)))
R1,1 & A2,2   --> * R3: + APPLY(c_1
                               f_1(c_4 c_5 c_3 comp(c_1 c_2))
                               f_1(c_1 c_5 f_1(c_4 c_5 c_3 comp(c_1 c_2)) c_2))
R1,1 & A3,2   --> * R4: + APPLY(c_2 f_1(c_1 c_5 f_1(c_4 c_5 c_3 comp(c_1 c_2)) c_2)  c_5)
A4,3 & T5,1   --> * R5: All x:I - APPLY(comp(c_4 c_1) c_3 x)  - APPLY(c_2 x c_5)
R4,1 & R5,2   --> * R6: - APPLY(comp(c_4 c_1)
                               c_3
                               f_1(c_1 c_5 f_1(c_4 c_5 c_3 comp(c_1 c_2)) c_2))
R2,1 & A4,1   --> * R15:All x:I y:Ixito - APPLY(y f_1(c_4 c_5 c_3 comp(c_1 c_2)) x)
                                       + APPLY(comp(c_4 y) c_3 x)
R15,1 & R3,1  --> * R16:+ APPLY(comp(c_4 c_1)
                               c_3
                               f_1(c_1 c_5 f_1(c_4 c_5 c_3 comp(c_1 c_2)) c_2))
R16,1 & R6,1  --> * R17:[]

Time Used for Refutation of Splitpart 1: 18 seconds

Refutation of Splitpart 2:
==========================

Initial Clauses:
  A1: All x:Any + =(x x)
* A2: All x,y:Ixito z,u:I + APPLY(y u f_1(y z u x))  - APPLY(comp(y x) u z)
* A3: All x,y:I z,u:Ixito + APPLY(u f_1(z y x u) y)  - APPLY(comp(z u) x y)
* A4: All x,y,z:I u,v:Ixito - APPLY(v z y)
                           - APPLY(u y x)  + APPLY(comp(v u) z x)
* T7: + APPLY(comp(comp(c_9 c_6) c_7) c_8 c_10)
* T8: - APPLY(comp(c_9 comp(c_6 c_7)) c_8 c_10)



T7,1 & A3,2    --> * R18:+ APPLY(c_7 f_1(comp(c_9 c_6) c_10 c_8 c_7) c_10)
T7,1 & A2,2    --> * R19:+ APPLY(comp(c_9 c_6) c_8 f_1(comp(c_9 c_6) c_10 c_8 c_7))
R19,1 & A3,2   --> * R20:+ APPLY(c_6
                                f_1(c_9 f_1(comp(c_9 c_6) c_10 c_8 c_7) c_8 c_6)
                                f_1(comp(c_9 c_6) c_10 c_8 c_7))
R19,1 & A2,2   --> * R21:+ APPLY(c_9
                                c_8
                                f_1(c_9 f_1(comp(c_9 c_6) c_10 c_8 c_7) c_8 c_6))
A4,3 & T8,1    --> * R22:All x:I - APPLY(c_9 c_8 x)
                                - APPLY(comp(c_6 c_7) x c_10)
R18,1 & A4,2   --> * R29:All x:I y:Ixito - APPLY(y x f_1(comp(c_9 c_6) c_10 c_8 c_7))
                                        + APPLY(comp(y c_7) x c_10)
R29,2 & R22,2  --> * R30:- APPLY(c_6
                                f_1(c_9 f_1(comp(c_9 c_6) c_10 c_8 c_7) c_8 c_6)
                                f_1(comp(c_9 c_6) c_10 c_8 c_7))
                         - APPLY(c_9
                                c_8
                                f_1(c_9 f_1(comp(c_9 c_6) c_10 c_8 c_7) c_8 c_6))
R30,1 & R20,1  --> * R31:- APPLY(c_9
                                c_8
                                f_1(c_9 f_1(comp(c_9 c_6) c_10 c_8 c_7) c_8 c_6))
R31,1 & R21,1  --> * R32:[]


Time Used for Refutation of Splitpart 2: 13 seconds
```

3 Logic

In this section we introduce higher-order logic, because it is a more adequate language to formulate mathematical problems than first-order logic. Then we define the notion of a sound logic morphism in order to give a formal transformation from higher-order to first-order logic. In the example above the transformation is just a preparatory application of a calculus rule such that we can stay in first-order logic, even within the same signature. For the next example, however, we need a more sophisticated consideration of a higher-order representation to construct the transformation.

3.1 Basic Notions

We begin with the definition of higher-order logic based on CHURCH's simple theory of types [7].

Definition (Types): ι is the type of the individuals. o is the type of the truth values. If $\tau_1, \ldots, \tau_m$, and σ are types not equal to o (with $m \geq 1$) then $(\tau_1 \times \cdots \times \tau_m \rightarrow \sigma)$ is the type of m-ary functions with arguments of types $\tau_1, \ldots, \tau_m$, respectively, and value of type σ. If $\tau_1, \ldots, \tau_m$ are types not equal to o (with $m \geq 1$) then $(\tau_1 \times \cdots \times \tau_m \rightarrow o)$ is the type of m-ary predicates with arguments of types $\tau_1, \ldots, \tau_m$, respectively.

Definition (Signature): The signature of a logic in $\mathcal{L}^\omega$ is a set $\mathcal{S} = \bigcup_\tau \mathcal{S}_\tau^{const} \cup \bigcup_\tau \mathcal{S}_\tau^{var}$ where each set $\mathcal{S}_\tau^{const}$ is a (possibly empty) set of constant symbols of type τ and $\mathcal{S}_\tau^{var}$ a countable infinite set of variable symbols of type τ. A logic in $\mathcal{L}^\omega$ is defined by its signature $\mathcal{S}$ and is denoted $\mathcal{L}^\omega(\mathcal{S})$.

Definition (Terms):

1. Every variable or constant of a type τ is a term.
2. If $f_{(\tau_1 \times \cdots \times \tau_m \rightarrow \sigma)}, t_{\tau_1}, \ldots, t_{\tau_m}$ are terms of the types indicated by their subscripts, then we get a term of type σ by $f_{(\tau_1 \times \cdots \times \tau_m \rightarrow \sigma)}(t_{\tau_1}, \ldots, t_{\tau_m})$.

Definition (Formulae):

1. Every term of type o is a formula.
2. If φ and ψ are formulae and x is a variable of any type, then $(\neg\varphi)$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \rightarrow \psi)$, $(\varphi \leftrightarrow \psi)$, $(\exists x \varphi)$, and $(\forall x \varphi)$ are formulae.

The standard semantics is due to ALFRED TARSKI [24] and has been extended by LEON HENKIN [12] to the general model semantics used in the sequel: The class of models is enlarged so that it is possible to find sound and complete calculi for higher-order logic based on this non-standard semantics. In particular every proof found in such a calculus is also a proof with respect to the standard semantics. In order to approximate the possible models in general model semantics to those of standard semantics we require that certain axioms are fulfilled, the so-called comprehension axioms, which guarantee the existence of certain functions and predicates. For these axioms compare [1, p.156].

Definition (Comprehension Axioms): The comprehension axioms Υ are the following formulae:

Υ^f For every term t of type $\tau \neq o$ of which the free variables are at most the different variables $x_1, \ldots, x_m, y_1, \ldots, y_k$ of type $\tau_1, \ldots, \tau_m, \sigma_1, \ldots, \sigma_k$:
$\forall y_1 \ldots \forall y_k \;\; \exists f_{(\tau_1 \times \cdots \times \tau_m \to \tau)} \;\; \forall x_1 \ldots \forall x_m \;\; (f(x_1, \ldots, x_m) \equiv t)$.

Υ^p For every formula φ of which the free variables are at most the different variables $x_1, \ldots, x_m, y_1, \ldots, y_k$ of type $\tau_1, \ldots, \tau_m, \sigma_1, \ldots, \sigma_k$:
$\forall y_1 \ldots \forall y_k \;\; \exists p_{(\tau_1 \times \cdots \times \tau_m \to o)} \;\; \forall x_1 \ldots \forall x_m \;\; (p(x_1, \ldots, x_m) \leftrightarrow \varphi)$.

In addition we need so-called extensionality axioms, which ensure that functions and predicates are equal if they agree on all arguments.

Definition (Extensionality Axioms): The extensionality axioms Ξ are the following formulae:

Ξ^f For all function symbols f, g of type τ equal to $(\tau_1 \times \cdots \times \tau_m \to \sigma)$, $\sigma \neq o$:
$\forall f \;\; \forall g \;\; (\forall x_{\tau_1} \ldots \forall x_{\tau_m} \;\; f(x_{\tau_1}, \ldots, x_{\tau_m}) = g(x_{\tau_1}, \ldots, x_{\tau_m}) \to f = g$

Ξ^p For all predicate symbols p, q of type τ equal to $(\tau_1 \times \cdots \times \tau_m \to o)$:
$\forall p \;\; \forall q \;\; (\forall x_{\tau_1} \ldots \forall x_{\tau_m} \;\; p(x_{\tau_1}, \ldots, x_{\tau_m}) \leftrightarrow q(x_{\tau_1}, \ldots, x_{\tau_m})) \to p = q$

3.2 Sorted Logics

Formulae, which are given to the MKRP-system are written in PLL [25, 22], a language for order-sorted first-order logic with equality predicate. PLL is our target language, but in this article we just use a flat sort structure. Sorts are introduced by SORT-declarations. Domain and range of functions and predicates are related to sorts by so-called TYPE-statements.

3.3 Logic Morphisms

Now we shall define those concepts that are necessary to describe the translations between formalizations in different logics.

Definition (Morphism of Logics): A morphism Θ is a mapping that maps the signature $\mathcal{S}$ of a logic $\mathcal{F}^1(\mathcal{S})$ to a signature of a logic $\mathcal{F}^2(\Theta(\mathcal{S}))$ and that maps every formula set in $\mathcal{F}^1(\mathcal{S})$ to a formula set in $\mathcal{F}^2(\Theta(\mathcal{S}))$.

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

The soundness of a morphism means: a proof that the translated problem is unsatisfiable entails that the original problem is unsatisfiable.

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 $\mathcal{F}^2(\mathcal{S}_2)$ is called a quasi-homomorphism iff the following conditions are satisfied:

1. For all terms:
   - 1.1 if x is a variable of $\mathcal{F}^1(\mathcal{S}_1)$ then $\Theta(x)$ is a variable of $\mathcal{F}^2(\mathcal{S}_2)$.
   - 1.2 if c is a constant of $\mathcal{F}^1(\mathcal{S}_1)$ then $\Theta(c)$ is a constant of $\mathcal{F}^2(\mathcal{S}_2)$.
   - 1.3 if $f(t_1, \ldots, t_m)$ is a term of $\mathcal{F}^1(\mathcal{S}_1)$ then
     $\Theta\left(f(t_1, \ldots, t_m)\right) = \theta\left(\Theta(f), \Theta(t_1), \ldots, \Theta(t_m)\right)$
     with $\theta(a, a_1, \ldots, a_m) = \begin{cases} a(a_1, \ldots, a_m) & \text{or} \\ \boldsymbol{\alpha}_a(a, a_1, \ldots, a_m) \end{cases}$
     The constants $\boldsymbol{\alpha}$ have to be chosen appropriately out of $\mathcal{S}_2$, especially they have to be *new*, that is, there must be no element $\alpha' \in \mathcal{S}_1$ so that $\boldsymbol{\alpha}_a = \Theta(\alpha')$. The case which is chosen can depend only on the type of a, not on the $a_1, \ldots, a_m$. ($\boldsymbol{\alpha}$ stands for *apply*.)
2. For all formulae φ_1, φ_2 and for all variables x:
   - 2.1 $\Theta(\varphi_1 \wedge \varphi_2) = \Theta(\varphi_1) \wedge \Theta(\varphi_2)$
   - 2.2 $\Theta(\neg\varphi) = \neg\Theta(\varphi)$
   - 2.3 $\Theta(\forall x \varphi) = \forall \Theta(x) \Theta(\varphi)$
3. All terms that are not formulae of $\mathcal{F}^1(\mathcal{S}_1)$ are mapped to terms that are not formulae of $\mathcal{F}^2(\mathcal{S}_2)$.

Now we give a sufficient criterion for the soundness of translations of formulae of $\mathcal{L}^\omega$ into formulae of PLL, which is strong enough to cover most requirements.

Theorem: If Θ is an injective quasi-homomorphism from $\mathcal{L}^\omega(\mathcal{S})$ to PLL, then Θ is sound (for a proof see [13]).

4 Explicit Versus Predicative Formalization

The second example treats a more fundamental change of a typical explicit representation. Again the explicit way to state the problem is more general in nature and 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 (this example is also taken from [9, p.37]).

4.1 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:
  $\forall \rho_{(\iota\times\iota\to o)}\ \ \mathrm{sym}(\rho) \leftrightarrow (\forall a_\iota, b_\iota\ \ \rho(a,b) \to \rho(b,a))$
- Definition of Transitivity:
  $\forall \rho_{(\iota\times\iota\to o)}\ \ \mathrm{trans}(\rho) \leftrightarrow (\forall a_\iota, b_\iota, c_\iota\ \ \rho(a,b)\ \wedge\ \rho(b,c) \to \rho(a,c))$
- 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:
  $\forall \rho_{(\iota\times\iota\to o)}, \sigma_{(\iota\times\iota\to o)}\ \ \mathrm{eqv}(\rho)\ \wedge\ \mathrm{eqv}(\sigma) \to \mathrm{eqv}(\rho \cap \sigma)$

4.2 Generated First-Order Formulation

These higher-order formulae are translated by a quasi-homomorphism (compare the definition above) into first-order logic and is formulated in PLL [25, 22]. We have to employ "apply" in representing relations, which are now object variables.

```
Formulae given to the editor
============================

Axioms:   * Formulation with Variable Relations *
          SORT I,IXITO:ANY
          TYPE APPLY(IXITO I I)
          * Definition of Intersection *
          TYPE INTER(IXITO IXITO):IXITO
          ALL RHO,SIGMA:IXITO ALL A,B:I APPLY(INTER(RHO SIGMA) A B)
                                  EQV  APPLY(RHO A B) AND APPLY(SIGMA A B)
          * Definition of Reflexivity *
          TYPE REF(IXITO)
          ALL RHO:IXITO REF(RHO) EQV (ALL A:I APPLY(RHO A A))
          * Definition of Symmetry *
          TYPE SYM(IXITO)
          ALL RHO:IXITO SYM(RHO) EQV (ALL A,B:I APPLY(RHO A B) IMPL APPLY(RHO B A))
          * Definition of Transitivity *
          TYPE TRANS(IXITO)
          ALL RHO:IXITO TRANS(RHO)
              EQV (ALL A,B,C:I APPLY(RHO A B) AND APPLY(RHO B C) IMPL APPLY(RHO A C))
          * Definition of Equivalence Relation *
          TYPE EQU.REL(IXITO)
          ALL RHO:IXITO EQU.REL(RHO) EQV REF(RHO) AND SYM(RHO) AND TRANS(RHO)

Theorems: ALL RHO,SIGMA:IXITO EQU.REL(RHO) AND EQU.REL(SIGMA) IMPL EQU.REL(INTER(RHO SIGMA))
```

4.3 Better First-Order Formulation

The following variant 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 GERHARD GENTZEN's natural deduction calculus [10, 15]. After the expansion of the definition we obtain a formula set that is first-order except for the intersection function, whose domain consists of predicates. In order to eliminate this function symbol we introduce for the term $\rho \cap \sigma$ a new predicate constant RHOSIGMA. Now the definitions are implicit in the formulation.

```
Formulae given to the editor
============================

Axioms:   * Formulation with Constant Relations *
          SORT I:ANY
          TYPE RHO(I I)
          TYPE SIGMA(I I)
          TYPE RHOSIGMA(I I)
          ALL A,B:I RHOSIGMA(A B) EQV RHO(A B) AND SIGMA(A B)

Theorems:    ((ALL A:I     RHO(A A))
          AND (ALL A,B:I   RHO(A B) IMPL RHO(B A))
          AND (ALL A,B,C:I RHO(A B) AND RHO(B C) IMPL RHO(A C))
          AND (ALL A:I     SIGMA(A A))
          AND (ALL A,B:I   SIGMA(A B) IMPL SIGMA(B A))
          AND (ALL A,B,C:I SIGMA(A B) AND SIGMA(B C) IMPL SIGMA(A C)))
       IMPL  ((ALL A:I     RHOSIGMA(A A))
          AND (ALL A,B:I   RHOSIGMA(A B) IMPL RHOSIGMA(B A))
          AND (ALL A,B,C:I RHOSIGMA(A B) AND RHOSIGMA(B C) IMPL RHOSIGMA(A C)))

Set of Theorem Clauses Resulting from Normalization and Splitting
=================================================================
```

```
Splitpart 1   T1: All x:Any + =(x x)
            * T2: All x:I + RHO(x x)
              T3: All x:I + SIGMA(x x)
            * T4: - RHO(c_1 c_1)
Splitpart 2   T5: All x:Any + =(x x)
              T6: All x:I + RHO(x x)
            * T7: All x:I + SIGMA(x x)
            * T8: - SIGMA(c_2 c_2)
Splitpart 3   T9: All x:Any + =(x x)
             T10: All x:I + RHO(x x)
             T11: All x:I + SIGMA(x x)
           * T12: + RHO(c_4 c_3)
             T13: + SIGMA(c_4 c_3)
           * T14: - RHO(c_3 c_4)
           * T15: All x,y:I - RHO(y x)
                            + RHO(x y)
Splitpart 4  T16: All x:Any + =(x x)
             T17: All x:I + RHO(x x)
             T18: All x:I + SIGMA(x x)
             T19: + RHO(c_4 c_3)
           * T20: + SIGMA(c_4 c_3)
           * T21: - SIGMA(c_3 c_4)
             T22: All x,y:I - RHO(y x)
                            + RHO(x y)
           * T23: All x,y:I - SIGMA(y x)
                            + SIGMA(x y)
```

```
Splitpart 5  T24: All x:Any + =(x x)
             T25: All x:I + RHO(x x)
             T26: All x:I + SIGMA(x x)
           * T27: + RHO(c_6 c_7)
             T28: + SIGMA(c_6 c_7)
           * T29: + RHO(c_7 c_5)
             T30: + SIGMA(c_7 c_5)
           * T31: - RHO(c_6 c_5)
             T32: All x,y:I - RHO(y x)  + RHO(x y)
             T33: All x,y:I - SIGMA(y x)  + SIGMA(x y)
           * T34: All x,y,z:I - RHO(z y)  - RHO(y x)
                              + RHO(z x)
Splitpart 6  T35: All x:Any + =(x x)
             T36: All x:I + RHO(x x)
             T37: All x:I + SIGMA(x x)
             T38: + RHO(c_6 c_7)
           * T39: + SIGMA(c_6 c_7)
             T40: + RHO(c_7 c_5)
           * T41: + SIGMA(c_7 c_5)
           * T42: - SIGMA(c_6 c_5)
             T43: All x,y:I - RHO(y x)  + RHO(x y)
             T44: All x,y:I - SIGMA(y x)  + SIGMA(x y)
             T45: All x,y,z:I - RHO(z y)  - RHO(y x)
                              + RHO(z x)
           * T46: All x,y,z:I - SIGMA(z y)  - SIGMA(y x)
                              + SIGMA(z x)
```

```
Initial Operations on Theorems
==============================

Splitpart 1  T4,1 & T2,1    -->* R1:  []

Splitpart 2  T8,1 & T7,1    -->* R2:  []

Splitpart 3  T15,2 & T14,1  -->* R3:  - RHO(c_4 c_3)
             R3,1 & T12,1   -->* R4:  []

Splitpart 4  T23,2 & T21,1  -->* R5:  - SIGMA(c_4 c_3)
             R5,1 & T20,1   -->* R6:  []

Splitpart 5  T29,1 & T34,2  -->* R7:  - RHO(c_6 c_7)  + RHO(c_6 c_5)
             R7,1 & T27,1   -->* R8:  + RHO(c_6 c_5)
             R8,1 & T31,1   -->* R9:  []

Splitpart 6  T41,1 & T46,2  -->* R10: - SIGMA(c_6 c_7)  + SIGMA(c_6 c_5)
             R10,1 & T39,1  -->* R11: + SIGMA(c_6 c_5)
             R11,1 & T42,1  -->* R12: []
```

4.4 Proof Statistics

In the following table we compare the runtime behaviour of Markgraf Karl (measured in seconds) computing our example with different option settings. “Depth” is the maximal depth of terms in clauses that are allowed to be deduced. “(Split)” means that the theorem may be divided into several parts for the proof. The “Terminator” is a special proof tool for unit resolution [2].

	Depth			Depth (Split)			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 is caused above all by the possibility to nest the intersection function.

4.5 Universality of Avoiding Functions

The above example also suggests that it is always a good idea to eliminate function symbols. But this need not be the case. Sometimes just this elimination method is used to construct sets of test examples with increasing complexity for theorem provers as the pigeonhole problem [19, Example 72] shows.

An informal higher-order representation of the general problem is the following:

- All pigeons are in holes:
  $\forall i\!:\!\text{pigeon}\ \exists j\!:\!\text{hole}\ \ In(i,j)$
- Only one pigeon in a hole:
  $\forall i,j\!:\!\text{pigeon}\ \ \forall k\!:\!\text{hole}\ \ i \neq j \rightarrow\ \neg In(i,k) \vee \neg In(j,k)$
- More pigeons than holes:
  $Card(\text{pigeon}) = S(Card(\text{hole}))$
- Mathematical background:
  $\forall n\!:\!\mathbb{N}\ \ S(n) > n$

  $\forall p,h\!:\!\text{sort}\ \ Card(p) > Card(h) \rightarrow \neg\exists\varphi\!:\!map\ \ Inj(\varphi,p,h)$

  $\forall\varphi\!:\!map\ \ \forall p,h\!:\!\text{sort}\ \ Inj(\varphi,p,h) \leftrightarrow \forall x,y\!:\!p\ \ \varphi(x)=\varphi(y) \rightarrow x=y$

This problem can be translated into first-order logic and was solved by MKRP in 17 seconds.

For each $n \in \mathbb{N}$ we can construct from this formulation a propositional logic problem with the number of clauses cubic depending on n. This is done via the substitution of universal quantification by finite conjunction and of existential quantification by finite disjunction in the first two formulae. The task for $n = 2$ has the following form with pigeon $= \{$1,2,3$\}$ and hole $= \{$a,b$\}$:

```
In_1a or In_1b        not In_1a or not In_2a        not In_1b or not In_2b
In_2a or In_2b        not In_1a or not In_3a        not In_1b or not In_3b
In_3a or In_3b        not In_2a or not In_3a        not In_2b or not In_3b
```

The translation to first-order logic is:

```
Formulae given to the editor
============================

Axioms:   SORT NAT,PRED,THING,MAP : ANY
          TYPE GT (NAT NAT)
          TYPE CARD (PRED) : NAT
          TYPE S (NAT) : NAT
          TYPE P,H : PRED
          TYPE A (PRED THING)
          TYPE IN (THING THING)
          TYPE INJ (MAP PRED PRED)
          TYPE AF (MAP THING) : THING
          TYPE PHI : MAP
          ALL I : THING A (P I) IMPL A (H AF (PHI I)) AND IN (I AF (PHI I))
          ALL I,J: THING A (P I) AND A (P J)
                   IMPL (ALL K : THING A (H K) IMPL (NOT I = J IMPL
                                                     (NOT IN (I K) OR NOT IN (J K))))
          CARD (P) = S (CARD (H))
          ALL N : NAT GT (S (N) N)
          ALL P,H : PRED GT (CARD (P) CARD (H)) IMPL (NOT EX PHI : MAP INJ (PHI P H))
          ALL PHI: MAP ALL
                     P,H : PRED INJ (PHI P H)
                     EQV (ALL X,Y : THING A (P X) AND A (P Y) AND AF (PHI X) = AF (PHI Y)
                          IMPL X = Y)

Set of Axiom Clauses Resulting from Normalization
=================================================

          A1:   All x:Any + =(x x)
        * A2:   + =(card(p) s(card(h)))
        * A3:   All x:Nat + GT(s(x) x)
        * A4:   All x:Thing - A(p x)  + A(h af(phi x))
        * A5:   All x:Thing - A(p x)  + IN(x af(phi x))
        * A6:   All x:Map y,z:Pred - GT(card(z) card(y))  + A(z f_2(x z))
        * A7:   All x:Map y,z:Pred - GT(card(z) card(y))  + A(z f_1(x z))
        * A8:   All x:Map y,z:Pred - GT(card(z) card(y))  + =(af(x f_2(x z)) af(x f_1(x z)))
        * A9:   All x:Map y,z:Pred - GT(card(z) card(y))  - =(f_2(x z) f_1(x z))
        * A10:  All x,y,z:Thing - A(p z)  - A(p y)  - A(h x)  + =(z y)  - IN(z x)  - IN(y x)


Refutation:
===========

Initial Clauses:     A1:   All x:Any + =(x x)
                   * A2:   + =(card(p) s(card(h)))
                   * A3:   All x:Nat + GT(s(x) x)
                   * A4:   All x:Thing - A(p x)  + A(h af(phi x))
                   * A5:   All x:Thing - A(p x)  + IN(x af(phi x))
                   * A6:   All x:Map y,z:Pred - GT(card(z) card(y))  + A(z f_2(x z))
                   * A7:   All x:Map y,z:Pred - GT(card(z) card(y))  + A(z f_1(x z))
                   * A8:   All x:Map y,z:Pred - GT(card(z) card(y))
                                              + =(af(x f_2(x z)) af(x f_1(x z)))
                   * A9:   All x:Map y,z:Pred - GT(card(z) card(y))  - =(f_2(x z) f_1(x z))
                   * A10:  All x,y,z:Thing - A(p z)  - A(p y)  - A(h x)  + =(z y)  - IN(z x)
                                           - IN(y x)



A2,1 & A3,1    --> * P1:   + GT(card(p) card(h))
```

```
P1,1 & A6,1    --> * R2:   All x:Map + A(p f_2(x p))

P1,1 & A7,1    --> * R3:   All x:Map + A(p f_1(x p))
P1,1 & A8,1    --> * R4:   All x:Map + =(af(x f_2(x p)) af(x f_1(x p)))
P1,1 & A9,1    --> * R5:   All x:Map - =(f_2(x p) f_1(x p))
R4,1 & A4,2    --> * P6:   + A(h af(phi f_1(phi p)))  - A(p f_2(phi p))
P6,2 & A6,2    --> * R7:   + A(h af(phi f_1(phi p)))  - GT(card(p) card(h))
R7,2 & P1,1    --> * R8:   + A(h af(phi f_1(phi p)))
R4,1 & A5,2    --> * P9:   + IN(f_2(phi p) af(phi f_1(phi p)))  - A(p f_2(phi p))
P9,2 & A6,2    --> * R10:  + IN(f_2(phi p) af(phi f_1(phi p)))  - GT(card(p) card(h))
R10,2 & P1,1   --> * R11:  + IN(f_2(phi p) af(phi f_1(phi p)))
A7,2 & A5,1    --> * R20:  All x:Map y:Pred - GT(card(p) card(y))
                                        + IN(f_1(x p) af(phi f_1(x p)))
R20,2 & A10,6  --> * R21:  All x:Pred - GT(card(p) card(x))  - A(p f_2(phi p))  - A(p f_1(phi p))
                                  - A(h af(phi f_1(phi p)))  + =(f_2(phi p) f_1(phi p))
                                  - IN(f_2(phi p) af(phi f_1(phi p)))
R21,2 & R2,1   --> * R22:  All x:Pred - GT(card(p) card(x))  - A(p f_1(phi p))
                                  - A(h af(phi f_1(phi p)))  + =(f_2(phi p) f_1(phi p))
                                  - IN(f_2(phi p) af(phi f_1(phi p)))
R22,2 & R3,1   --> * R23:  All x:Pred - GT(card(p) card(x))  - A(h af(phi f_1(phi p)))
                                  + =(f_2(phi p) f_1(phi p))
                                  - IN(f_2(phi p) af(phi f_1(phi p)))
R23,2 & R8,1   --> * R24:  All x:Pred - GT(card(p) card(x))  + =(f_2(phi p) f_1(phi p))
                                  - IN(f_2(phi p) af(phi f_1(phi p)))
R24,2 & R5,1   --> * R25:  All x:Pred - GT(card(p) card(x))  - IN(f_2(phi p) af(phi f_1(phi p)))
R25,2 & R11,1  --> * R26:  All x:Pred - GT(card(p) card(x))
R26,1 & P1,1   --> * R27:  []


Time Used for Refutation: 17 seconds
```

5 Tactics to Cope with the Presented Situations

Tactics were explicitly used for the first time in the LCF proof system [11]. The proof-checker Nuprl [8], which is based on Automath [3] and Per Martin-Löf's type theory [16], incorporates the possibility to write tactics as well as the programming language ML of LCF. Tactics can be viewed as mappings from proofs to proofs. Instead of proving $\Gamma \vdash \text{THEOREM}$ it is sufficient to show several parts $\Gamma_1 \vdash \text{THEOREM}_1, \ldots, \Gamma_n \vdash \text{THEOREM}_n$ (soundness of tactics).

Elementary tactics correspond in Nuprl to the application of calculus rules. In order to get powerful tactics there are some possibilities to combine tactics to composed tactics by the so-called tacticals: `IF` *Expression* `THEN` *Tactic*, `IF` *Expression* `THEN` $Tactic_1$ `ELSE` $Tactic_2$, `REPEAT` *Tactic*, `WHILE` *Expression* `DO` *Tactic*, `COMPOSITION` $Tactic_1$ $Tactic_2$. Originally *Expression* is a boolean expression written in ML. In the sequel we use an informal logical language to specify our expressions.

5.1 Tactics for the Examples

In a more informal style a tactic to transform our first example from the initial formulation into the form more appropriate for resolution would be: If certain conditions are fulfilled, the theorem and an axiom 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 $\quad$ TO_PROVE($\Gamma \vdash$ THEOREM)
$\wedge$ ATOM(THEOREM)
$\wedge\ \exists$ AX AX $\in \Gamma \wedge$ AX $= (Antecedent \rightarrow Succedent)$
$\wedge\ \neg$ ATOM($Antecedent$)
$\wedge\ \exists \sigma : Matcher$ THEOREM $= \sigma(Succedent)$
$\wedge\ \neg$ DOES_OCCUR(PREDICATE(THEOREM), $(\Gamma - \{$AX$\} \cup \{Antecedent\})$)
THEN $\quad$ TO_PROVE($\Gamma - \{$AX$\} \vdash \sigma(Antecedent)$)

In this description of the tactic we neglect the existence of universal quantifiers. 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 from the initial first-order one.

ELIM_THM_VARS :=
IF $\quad$ TO_PROVE($\Gamma \vdash$ THEOREM)
$\wedge$ THEOREM $= \forall x_1, \ldots, x_n\ \varphi$
$\wedge\ \sigma := \{x_1 \leftarrow c_1, \ldots, x_n \leftarrow c_n\}$
$\wedge\ \forall i, j\ i, j \in \{1, \ldots, n\} \wedge c_i = c_j \Longrightarrow i = j$
$\wedge\ \forall c\ c \in \{c_1, \ldots, c_n\} \Longrightarrow \neg$ DOES_OCCUR($c, \Gamma \cup \{\varphi\}$)
THEN $\quad$ TO_PROVE($\Gamma \vdash \sigma(\varphi)$)

EXPAND_DEF :=
IF $\quad$ TO_PROVE($\Gamma \vdash$ THEOREM)
$\wedge\ \exists$ AX AX $\in \Gamma \wedge$ AX $= \forall x_1, \ldots, x_n\ P(x_1, \ldots, x_n) \leftrightarrow \varphi$
$\wedge\ \neg$ DOES_OCCUR(P, φ)
THEN $\quad$ TO_PROVE(SUBST_ALL($P, \varphi, \Gamma - \{$AX$\} \vdash$ THEOREM))

with a tactic SUBST_ALL(*pred, formula, in*) replacing all occurrences of the predicate P : $P(t_1, \ldots, t_n)$ by $\sigma(\varphi)$ with the unifier $\sigma = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$.

INST_VARS :=
IF $\quad$ TO_PROVE($\Gamma \vdash$ THEOREM)
$\wedge\ \exists t$ DOES_OCCUR(t,THEOREM) $\wedge$ GROUND(t) $\wedge\ t = f(t_1, \ldots, t_n)$
$\Longrightarrow \forall s$ DOES_OCCUR(s,THEOREM) $\wedge\ s = f(s_1, \ldots, s_n) \Longrightarrow s = t$
$\wedge\ \forall$AX AX$\in \Gamma \Longrightarrow \forall s$ DOES_OCCUR(s,AX) $\wedge\ s = f(s_1, \ldots, s_n)$
$\Longrightarrow$ 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 $\quad$ TO_PROVE(SUBST($\forall x_1, \ldots, x_n\ \varphi, \sigma(\varphi), \Gamma) \vdash$ THEOREM)

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

ELIM_APPLY :=

$$
\begin{array}{ll}
\texttt{IF} & \textsc{to\_prove}(\Gamma \vdash \textsc{theorem}) \\
 & \wedge\ \forall\varphi\ \textsc{Atom}(\varphi) \wedge \textsc{does\_occur}(\varphi, \Gamma \cup \{\textsc{theorem}\}) \wedge \varphi = \mathit{Apply}(s, s_1, \ldots, s_n) \\
 & \quad \Longrightarrow \textsc{ground}(s) \\
\texttt{THEN} & \textsc{to\_prove}(\textsc{subst}(\mathit{Apply}(s, s_1, \ldots, s_n), s(s_1, \ldots, s_n), \Gamma \vdash \textsc{theorem}))
\end{array}
$$

with the ground terms converted to new predicate symbols s. Hence after the application of this tactic the predicate symbol *Apply* is completely eliminated.

Built together this leads to the following composed tactic:

ELIM :=
COMPOSITION ELIM_THM_VARS
REPEAT EXPAND_DEF
INST_VARS
ELIM_APPLY

The tactic INST_VARS is rather complicated but very general in nature. When the "Apply"-construct appears only together with universal quantification as described in the tactic it seems possible to eliminate it in a similar way.

An alternative procedure would be to begin with higher-order 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 contrasts to the first procedure, where the translation is done at first:

ELIM :=
COMPOSITION TRANSLATE
ELIM_THM_VARS
REPEAT EXPAND_DEF
INST_VARS
ELIM_APPLY

5.2 Soundness of Tactics and Proof Presentation

Of course the correctness of all involved tactics has to be proved. Normally such a proof is obvious, because tactics 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 at the end of section 3.

For ELIM_APPLY we stay in the same logic but change the signature. This change is not essential, because the proof steps can be mapped one-to-one. All *Apply*s appear together with constants c in the form $Apply(c, \ldots)$ and are transformed to $c(\ldots)$. This transformation is also possible in all proof steps.

ELIM_APPLY is just one instance of the variability of representation in a logical language. Another example for this variability is constituted by the numerous representations of the equality predicate.

Soundness considerations for transformations between variants of problem descriptions are not really important because we take the position of a "Nominalist" [20]: As desirable as it is to find a proof as desirable it is to be able to communicate it. Of course it is very reasonable to present the proof in the language, which has been input by the user, because he is familiar with it.

In our case we have a first-order proof procedure, and the final proof can be translated back into higher-order logic, because the quasi-homomorphisms are all bijective.

6 Conclusion

We showed how tactics can be used to reformulate problem descriptions, such that they are more appropriately stated for a resolution theorem prover. Reformulated this way they can be solved much faster. The examples above support our opinion that resolution alone is not adequate for a system that can serve as tool in the daily work of a mathematician. The presented work will be further developed in the project "Ω-MKRP" [23], where among other things tactics will be designed for the combination of tactical and resolution theorem proving. We hope that hundreds of tactics like those presented previously will be the result of some years of experimentation and will lead to a useful system.

May be one day the automatic generation of tactics from new examples will be possible by exploiting the knowledge represented in these rules. As the example of the checkerboard without two opposite edges [17] shows, the reformulation of problems plays a central role in the development of AI right from its roots (for this example see also [26, p.117]). Techniques developed in AI [14] may be adaptable to mathematical tasks.

We think that the explicit formulation of such tactics opens the eye for the structure and contents of the coded know-how.

References

[1] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof.* Academic Press, Orlando, Florida, USA, 1986.

[2] Gregor Antoniou and Hans Jürgen Ohlbach. Terminator. In *Proceedings of the 8th IJCAI*, Karlsruhe, Germany, 1983. Morgan Kaufman, San Mateo, California, USA.

[3] Nicolaas Govert de Bruijn. A survey of the project AUTOMATH. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry - Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 579–606. Academic Press, London, United Kingdom, 1980.

[4] Alan Bundy. *The Computer Modelling of Mathematical Reasoning.* Academic Press, London, United Kingdom, 1983.

[5] Alan Bundy. The use of explicit plans to guide inductive proofs. In *Proceedings of the 9th CADE*, Argonne, Illinois, USA, 1988. Springer Verlag, Berlin, Germany. LNCS 310.

[6] Alan Bundy, Frank van Harmelen, Christian Horn, and Alan Smaill. The OYSTER-CLAM system. In Mark E. Stickel, editor, *Proceedings of the 10th CADE*, pages 647–648, Kaiserslautern, Germany, 1990. Springer Verlag, Berlin, Germany. LNAI 449.

[7] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, **5**:56–68, 1940.

[8] R.L. Constable, S.F. Allen, H.M. Bromley, W.R. Cleaveland, J.F. Cremer, R.W. Harper, D.J. Howe, T.B. Knoblock, N.P. Mendler, P Panangaden, J.T. Sasaki, and S.F. Smith. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1986.

[9] Peter Deussen. *Halbgruppen und Automaten*, volume 99 of *Heidelberger Taschenbücher.* Springer Verlag, Berlin, Germany, 1971.

[10] Gerhard Gentzen. Untersuchungen über das logische Schließen II. *Mathematische Zeitschrift*, **39**:572–595, 1935.

[11] Michael Gordon, Robin Milner, and Christopher Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation.* LNCS 78. Springer Verlag, Berlin, Germany, 1979.

[12] Leon Henkin. Completeness in the theory of types. *Journal of Symbolic Logic*, **15**:81–91, 1950.

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

[14] Richard E. Korf. Toward a model of representation changes. *Artificial Intelligence*, **14**:41–78, 1980.

[15] Christoph Lingenfelder. *Transformation and Structuring of Computer Generated Proofs.* PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany, 1990.

[16] Per Martin-Löf. Constructive mathematics and computer programming. In *6th International Congress for Logic, Methodology, and Philosophy of Science*, pages 153–175. North Holland, Amsterdam, Netherlands, 1982.

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

[18] Hans Jürgen Ohlbach and Jörg H. Siekmann. The Markgraf Karl Refutation Procedure. SEKI Report SR-89-19, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany, 1989.

[19] Francis Jeffry Pelletier. Seventy-five problems for testing automatic theorem provers. *Journal of Automated Reasoning*, **2**:191–216, 1986.

[20] Francis Jeffry Pelletier. The philosophy of automated theorem proving. In John Mylopoulos and Ray Reiter, editors, *Proceedings of the 12th IJCAI*, pages 538–543, Sydney, 1991. Morgan Kaufman, San Mateo, California, USA.

[21] George Pólya. *Mathematical Discovery – On understanding, learning, and teaching problem solving*. Princeton University Press, Princeton, New Jersey, USA, 1962/1965. Two volumes, also as combined edition, 1981, John Wiley and Sons, New York, USA.

[22] Axel Präcklein, editor. The MKRP User Manual. SEKI Report, forthcoming, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 1992.

[23] Jörg H. Siekmann et al. Ω-MKRP. SEKI Report, forthcoming, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 1992.

[24] Alfred Tarski. Der Wahrheitsbegriff in den formalisierten Sprachen. *Studia philosophia*, **1**:261–405, 1936.

[25] Christoph Walther. The Markgraf Karl Refutation Procedure. PLL – a first-order language for an automated theorem prover. Interner Bericht 35/82, Fakultät für Informatik, Universität Karlsruhe, Karlsruhe, Germany, 1982.

[26] Larry Wos, Ross Overbeek, Ewing Lusk, and Jim Boyle. *Automated Reasoning – Introduction and Applications*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1984.