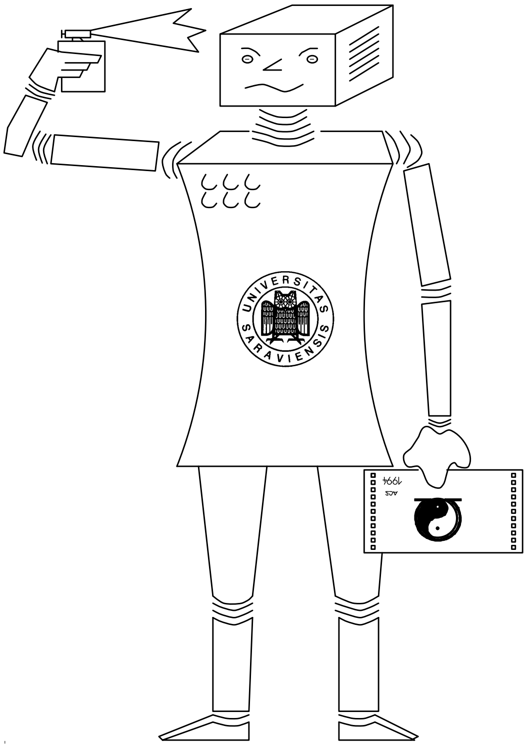
UNIVERSITÄT DES SAARLANDES
FACHBEREICH INFORMATIK
D-66041 SAARBRÜCKEN
GERMANY
WWW: http://jswww.cs.uni-sb.de/pub/www/



SEKI Report

Adaptation of Declaratively Represented Methods in Proof Planning

Xiaorong Huang   Manfred Kerber
Lassaad Cheikhrouhou

SEKI Report SR-95-12

Adaptation of Declaratively Represented Methods in Proof Planning

Xiaorong Huang Manfred Kerber Lassaad Cheikhrouhou
Fachbereich Informatik
Universität des Saarlandes
D-66041 Saarbrücken
Germany
{huang|kerber|lassaad}@cs.uni-sb.de

Abstract

The reasoning power of human-oriented plan-based reasoning systems is primarily derived from their domain-specific problem solving knowledge. Such knowledge is, however, intrinsically incomplete. In order to model the human ability of adapting existing methods to new situations we present in this work a declarative approach for representing methods, which can be adapted by so-called meta-methods. Since apparently the success of this approach relies on the existence of general and strong meta-methods, we describe several meta-methods of general interest in detail by presenting the problem solving process of two familiar classes of mathematical problems. These examples should illustrate our philosophy of proof planning as well: besides planning with the current repertoire of methods, the repertoire of methods evolves with experience in that new ones are created by meta-methods which modify existing ones.

1 Introduction

Machine-oriented theorem provers like those based on resolution, paramodulation, rewriting have been successfully applied in different fields of logic and mathematics (see, e.g., [WOLB84, chapters 9,10]). The strength of these systems is truly remarkable, but the general complexity results demonstrate clearly that no algorithm can be constructed to practically solve arbitrary tasks, even propositional logic is in a class that is generally considered intractable. The success of human mathematician is largely ascribed to the fact that they are generally specialized in some fields and can rely on domain-specific problem solving techniques they have accumulated throughout their professional experiences. This kind of problem solving behavior is first supported in interactive theorem proving systems (e.g. LCF [GMW79] or Nuprl [Con86]), which contain specific problem solving knowledge called tactics. In order to model the dynamic search process as well, tactics have been extended to so-called methods by adding specifications. Methods, in turn, have been successfully incorporated into a planning framework (e.g. CLAM [BvHHS90, BSvH+93]). Intuitively speaking, a method contains a piece of knowledge for solving or simplifying problems or transforming them into a form that is easier to solve.

The reasoning power of such systems is not derived from a complete underlying reasoning calculus, but relies on domain-specific problem solving knowledge. Such knowledge is, however, inevitably incomplete, which leads to a limited reasoning power of the corresponding systems. While this holds for plan based systems and human mathematicians alike, the latter often go beyond their specialized knowledge. They have the ability to adapt existing methods to novel situations, which is one of the main features contributing to our problem solving competence (see [Pól45] for mathematical reasoning and [Van89] for general problem solving). Although very important, this issue remains widely unaddressed in traditional proof planning systems. Actually in a previous framework where tactics are pure procedures, this task is quite formidable and equals to the problem of mechanically modifying procedures.

In this work we present a *declarative* approach in order to address the mechanical adaptation of methods. The key idea is to declaratively represent methods, and thereby to enable their mechanical adaptation by so-called meta-methods. The success of this approach heavily relies on the existence of general and strong meta-methods. Therefore, besides the theoretical framework, we describe in detail several of them with the help of the problem solving process of two familiar classes of mathematical problems. These trace protocols of problem solving should illustrate our philosophy of proof planning: instead of using a strong but fixed set of methods, the repertoire of methods evolves with the experience.

The technical report proceeds as follows: in the next section we introduce the basic ideas of our approach. In section 3 we present our first example showing how methods can be adapted to not directly fitting situations. We start with a proof for the theorem that the non-empty intersection of two subsemi-groups of a semi-group remains a subsemi-group. Then we modify the corresponding method by the meta-method `connective-to-quantifier` in order to come up with a method for proving that the non-empty intersection of a family of subsemi-groups of a semi-group remains a subsemi-group. We briefly illustrate how the approach is realized in the Ω-MKRP system. In section 4 we exemplify our approach by describing the evolution of diagonalization methods. This example class is interesting because of its historical importance in mathematics. In particular, although it will be difficult for a proof planning system to invent a famous method as Cantor did in this case, it is an interesting test if it can carry out the subsequent modifications expected from a mathematician with some training. We start with the proof of the theorem that the powerset of a set has a greater cardinality than the set itself. Then we stepwise modify the corresponding method to cope with the proofs of the theorem that the continuum has a greater cardinality than the natural numbers and of the halting problem. Besides generating new specific methods for new problems, meta-methods are also applied to generalize existing methods. In section 5 we discuss the results and briefly describe the future development.

2 A Declarative Approach toward Proof Planning

In this section we present our approach to proof planning, show how methods can be declaratively represented and mechanically modified.

The Planning Framework

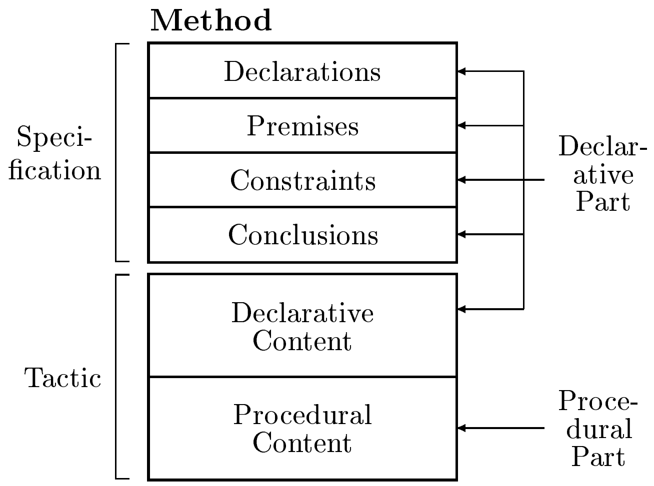
The work in this technical report should be understood in the setting of a computational model that casts the entire process of theorem proving, from the analysis of a problem up to the completion of a proof, as an *interleaving process* of proof planning, method execution and verification. In particular, this model ascribes a reasoner's reasoning competence to the existence of methods together with a planning mechanism that uses these methods for proof planning.

To understand the proof planning process, please remember that the goal of proof planning is to fill gaps in a given partial proof tree by forward and backward reasoning [HKKR94]. Thus from an abstract point of view the planning process is the process of exploring the search space of *planning states* that is generated by the *plan operators* in order to find a complete *plan* (that is a sequence of instantiated plan operators) from a given *initial state* to a *terminal state*.

Concretely a *planning state* contains a subset of lines in the current partial proof that correspond to the boundaries of a gap in the proof. This subset can be divided into *open lines* (that must be proved to bridge the gap) and *support lines* (that can be used as premises to bridge it). The initial planning state consists of all lines in the initial problem; the assumptions are the support lines and the conclusion is the only open line. The terminal planning state is reached when there is no more open line in the planning state.

A Declarative Representation for Methods

Formally, a method in our approach is defined as a 6-tuple with the components:



Declarations: A signature that declares the meta-variables used in the method,

Premises: Schemata of proof lines which are used by this method to prove the conclusions,

Constraints: Additional restrictions on the premises and the conclusions, which can not be formulated in terms of proof line schemata (described in [HKRS94]),

Conclusions: Schemata of proof lines which this method is designed to prove,

Declarative Content: A piece of declarative knowledge interpreted by the procedural content. This slot is currently restricted to schemata of partial proofs,

Procedural Content: Either a standard procedure interpreting the declarative content, or a special purpose inference procedure.

Methods can be very general, for instance consist essentially of one application of a rule at the calculus level. More specific methods comprise proof ideas, for instance, to use a homomorphy property, to apply mathematical induction or to use diagonalization. Such methods are typically formulated in terms of proof schemata. The most specific methods consist of full proofs for specific problems.

Modification of Methods

The standard practice of proof planning is to use a fixed repertoire of methods in order to produce a proof plan [BSvH+93]. Difficulties arise when new problems are encountered exceeding the power of the existing methods. To overcome this, a main feature contributing to the problem solving competence of mathematicians becomes crucial, namely that they can extend their current problem solving repertoire by adapting existing methods to suit novel situations.

The intention of our work can be compared to Ireland's approach of proof critics [Ire92]. While proof critics are specific and attached to single methods, meta-methods embody general problem independent procedures for adapting arbitrary methods which meet some applicability conditions. The work of Giunchiglia and Traverso [GT94] to represent tactics in a logical meta-language has a similar motivation as well, namely to mechanically adapt existing tactics. Their formalism is more expressive since they can represent procedural aspects like loops on a logical meta-level too. In our approach, the declarative part of methods basically consists of a proof schema. This leads to a more natural representation and enables an easier transformation in some cases.

By adopting a declarative approach for formulating methods, it is firstly possible to extract methods from proofs, secondly it is also feasible to formulate meta-methods adapting existing methods. Currently, a meta-method is essentially a procedure which takes as input some methods and some additional parameters, and produces a new method. We have already identified a variety of meta-methods such as: the generalization of methods in order to apply them in less specific situations or the syntactic adaptation of methods to bridge syntactic gaps, for instance, arities of predicates. In this technical report we examine the meta-methods `connective-to-quantifier`, `Cut-Submethod`, `Abstract`, and `set2func`.

3 From Conjunctions to Universal Quantifications

In this section we illustrate a concrete modification, which translates connectives into quantifiers. The initial method has been extracted from a proof showing that the intersection of two subsemi-groups remains a subsemi-group:
Let U and V be subsemi-groups of a semi-group G, then the intersection $U \cap V$ is also a subsemi-group of G, if the intersection is not empty.
The informal proof goes as follows:

$$\begin{aligned} u, v \in U \cap V &\rightarrow u, v \in U \wedge u, v \in V \\ &\rightarrow u \cdot v \in U \wedge u \cdot v \in V \\ &\rightarrow u \cdot v \in U \cap V \end{aligned}$$

In a previous work [Ker89], we have presented informally how this proof can be transformed into a corresponding proof concerning an arbitrary number of subsemi-groups. A

Declarations	—			
Premises	$\cap$-Def,SubSGrpDef,NonemptyDef			
Constraint	—			
Conclusions	33			
Declarative Content	2.	2	$\vdash$ subsemigrp$(U_0, G_0, \cdot) \wedge$ subsemigrp$(V_0, G_0, \cdot) \wedge$ nonempty$(U_0 \cap V_0)$	Hyp
	3.	3	$\vdash x_0 \in (U_0 \cap V_0) \wedge y_0 \in (U_0 \cap V_0)$	Hyp
	5.	3	$\vdash x_0 \in U_0 \wedge x_0 \in V_0$	$\cap$-Def(4)
	6.	3	$\vdash x_0 \in U_0$	And-E(5)
	7.	3	$\vdash x_0 \in V_0$	And-E(5)
	9.	3	$\vdash y_0 \in U_0 \wedge y_0 \in V_0$	$\cap$-Def(4)
	10.	3	$\vdash y_0 \in U_0$	And-E(9)
	11.	3	$\vdash y_0 \in V_0$	And-E(9)
	14.	2	$\vdash \forall x_\blacksquare \forall y_\blacksquare x \in U_0 \wedge y \in U_0 \rightarrow x \cdot y \in U_0$	SubSGrpDef(13)
	17.	2	$\vdash \forall x_\blacksquare \forall y_\blacksquare x \in V_0 \wedge y \in V_0 \rightarrow x \cdot y \in V_0$	SubSGrpDef(16)
	20.	2,3	$\vdash x_0 \cdot y_0 \in U_0$	14(6,10)
	21.	2,3	$\vdash x_0 \cdot y_0 \in V_0$	17(7,11)
	22.	2,3	$\vdash x_0 \cdot y_0 \in U_0 \wedge x_0 \cdot y_0 \in V_0$	And-I(20,21)
	33.		$\vdash \forall G, \cdot, U, V_\blacksquare$(subsemigrp$(U, G, \cdot) \wedge$ subsemigrp$(V, G, \cdot) \wedge$ nonempty$(U \cap V))$ $\rightarrow$ subsemigrp$(U \cap V, G, \cdot)$	Forall-I(32)
Procedural Content	`schema-interpreter`			

Figure 1: Method `subsemigroup-conjunction`

method solving the first problem is given in the method in figure 1, which basically consists of a proof generated within the proof development environment Ω-MKRP [HKK+94] (to abstract away from details we only show the key steps). In the following we illustrate how this method can be transformed by a meta-method to a method for the case of arbitrarily many subsemi-groups. The new theorem can be viewed as a generalization of the previous one:

Let $\{U_i : i \in I\}$ be a family of subsemi-groups of a semi-group G, then the intersection $\bigcap_{i \in I} U_i$ is also a subsemi-group of G, if the intersection is not empty.

Actually, the proof of this theorem is analogous to the one above. The analogy can be derived from the correspondence between "$\wedge$" and "$\cap$" on the one hand and "$\forall$" and "$\bigcap$" on the other hand. The proof can therefore be sketched as:

$$\begin{aligned} u, v \in \bigcap_{i \in I} U_i &\rightarrow \forall i \in I_\blacksquare u, v \in U_i \\ &\rightarrow \forall i \in I_\blacksquare u \cdot v \in U_i \\ &\rightarrow u \cdot v \in \bigcap_{i \in I} U_i \end{aligned}$$

The corresponding method for this proof is given in figure 2 and can be constructed out of the method `subsemigroup-conjunction` in figure 1 by applying the meta-method `connective-to-quantifier`, which contains the following parts[1]:

- parameters: a (possibly empty) list of pairs of related formula patterns

[1] Meta-variables are indicated by overlining.

Declarations	—
Premises	$\bigcap$-Def,SubSGrpDef,NonemptyDef
Constraint	—
Conclusions	33
Declarative Content	1. 1 $\vdash \forall n_\blacksquare \mathrm{subsemigrp}(U_0(n), G, \cdot) \wedge \mathrm{nonempty}(\bigcap U_0)$ Hyp 2. 2 $\vdash x_0 \in \bigcap U_0 \wedge y_0 \in \bigcap U_0$ Hyp 5. 2 $\vdash \forall n_\blacksquare x_0 \in U_0(n)$ $\bigcap$-Def(3) 6. 2 $\vdash x_0 \in U_0(n_0)$ Forall-E(5) 7. 2 $\vdash \forall n_\blacksquare y_0 \in U_0(n)$ $\bigcap$-Def(4) 8. 2 $\vdash y_0 \in U_0(n_0)$ Forall-E(7) 13. 1 $\vdash \forall x, y_\blacksquare (x \in U_0(n_0) \wedge y \in U_0(n_0)) \rightarrow x \cdot y \in U_0(n_0)$ SubSGrpDef(12) 20. 1,2 $\vdash x_0 \cdot y_0 \in U_0(n_0)$ 13(6,8) 21. 1,2 $\vdash \forall n_\blacksquare x_0 \cdot y_0 \in U_0(n)$ Forall-I(20) 33. $\vdash \forall G, \cdot, U_\blacksquare (\forall n_\blacksquare \mathrm{subsemigrp}(U(n), G, \cdot) \wedge \mathrm{nonempty}(\bigcap U)) \rightarrow \mathrm{subsemigrp}(\bigcap U, G, \cdot)$ Forall-I(32)
Procedural Content	`schema-interpreter`

Figure 2: Method `subsemigroup-univ-quantification`

- transformation rules:
  1. Merge two proof lines with formulae which are different only for one single constant or variable, that is, with formulae $\psi[\overline{U}]$ and $\psi[\overline{V}]$, where $\psi[\overline{U}] = \mathsf{subst}(\psi[\overline{V}], \overline{U}, \overline{V})$. $\psi[\overline{U}]$ denotes that $\overline{U}$ occurs in ψ:

     $$\left.\begin{array}{ll} L_1 & \psi[\overline{U}] \\ L_2 & \psi[\overline{V}] \end{array}\right\} \mapsto L_3 \quad \psi[\overline{U}(\overline{n})]$$

  2. Transform certain conjunctions into universal quantifications and certain disjunctions into existential quantifications in formulae:

     $$\begin{array}{llll} L_1 & \Phi[\varphi[\overline{U}] \wedge \varphi[\overline{V}]] \mapsto L_2 & \Phi[\forall \overline{n}_\blacksquare \varphi[\overline{U}(\overline{n})]] \\ L_1 & \Phi[\varphi[\overline{U}] \vee \varphi[\overline{V}]] \mapsto L_2 & \Phi[\exists \overline{n}_\blacksquare \varphi[\overline{U}(\overline{n})]] \end{array}$$

  3. Execute the transformation according to the formula patterns in the parameter slot.

The last transformation rule gives the user the possibility to specify additional problem-specific transformations, here $(\overline{U} \cap \overline{V} \mapsto \bigcap \overline{U})$, which results in the transition

$$L_1 \quad \Phi[U \cap V] \mapsto L_2 \quad \Phi[\bigcap U].$$

Such optional transformation is not always necessary, please see the different formulations of the pigeon-hole principle [KP96].

We have not mentioned the adaptation of the justification in the transformation rules above. In general the elimination rules of logical connectives are transformed into the corresponding rules for the instantiation of quantifications. The justification And-E(5) in the first method becomes Forall-E(5) in the second, for instance.

The transformation above does not formulate an unambiguous characterization of the mappings to be carried out, since the first rule may be applied to wrong candidates, such as lines 6 & 10, as well as to right one, such as lines 6 & 7. Which lines should be merged by this rule, can be determined by traversing the proof tree backwards from the theorem. Whenever a line having a conjunction (or a disjunction) as formula is justified by applying a connective introduction to the two conjuncts (or disjuncts, respectively) dual lines of the corresponding subproofs are considered. Identical dual lines modulo the specified difference by the rule are merged. The rest of these subproofs should be cut and the premise list of the method should be accordingly adapted.

The result of applying this meta-method with the additional transformation pattern as argument is the method in figure 2. The four pairs of lines (6,7), (10,11), (14,17), and (20,21) are merged by rule one to the new lines 6, 8, 13, and 20. For instance

$$\left.\begin{array}{l} 6\ x_0 \in U_0\ \text{And-E}(5) \\ 7\ x_0 \in V_0\ \text{And-E}(5) \end{array}\right\} \mapsto 6\ x_0 \in U_0(n_0)\ \text{Forall-E}(5)$$

Rule three is applied to lines 2, 3, and T, thereby we get from line 3 in the source proof line 2 in the target proof (the other two lines are also manipulated by rule two). By rule two, lines 2, 5, 9, 22, and T are mapped to lines 1, 5, 7, 21, and T, respectively.

We have encoded the method `subsemigroup-conjunction` in the Ω-MKRP system [HKK+94] and implemented the meta-method `connective-to-quantifier` as described above. Ω-MKRP uses a typed higher-order input language which corresponds roughly to Church's λ-calculus [Chu40], and a natural deduction calculus as its main output proof formalism. This meta-method creates the method `subsemigroup-univ-quantification` by adapting the previous method. Concerning the control, up to now we have to manually interrupt the planning process and provide the meta-methods with appropriate instantiations for meta-variables.

As can be seen from the example above, further automation is possible by comparing the source and the target theorems: The correspondence of the subformula $\text{subsemigrp}(U, G, \cdot) \wedge \text{subsemigrp}(V, G, \cdot)$ in the first theorem and the subformula $\forall n_\bullet \text{subsemigrp}(U(n), G, \cdot)$ in the second theorem suggests to apply the meta-method `connective-to-quantifier`. In the same manner the parameter can be constructed from the remaining differences of the two theorems by mapping the term $U \cap V$ to $\bigcap U$.

4 Diagonalization

Since the diagonalization method is well-studied in mathematics and since it applies to a class of problems of different syntactical structure, we have used it from the very beginning as a non-trivial example serving as a guideline to conceptualize our approach [HK91].

Proofs as Methods

First we illustrate how our first diagonalization method is extracted from a successful proof, constructed with the proof development environment Ω-MKRP. The first diagonalization problem we examine is the *powerset* theorem stating that the cardinality of the powerset of a set is greater than the cardinality of the set itself.

The whole problem consists of six assumptions and the conclusion **Powerset**.

TND	$\forall x_o. x \vee \neg x$
=-Ref	$\forall x_o. x = x$
=-Equiv	$\forall x_o. \forall y_o. x = y \rightarrow [x \leftrightarrow y]$
Surj-Def	$\forall f_{\iota\rightarrow(\iota\rightarrow o)}. \forall a_{\iota\rightarrow o}. \forall b_{(\iota\rightarrow o)\rightarrow o}. \mathrm{surj}(f, a, b) \leftrightarrow$ $(\forall x_{\iota\rightarrow o}. x \in b \rightarrow (\exists y_\iota. (y \in a \wedge x = f(y))))$
PSet-Def	$\forall a_{\iota\rightarrow o}. \forall x_{\iota\rightarrow o}. x \in P(a) \leftrightarrow x \subseteq a$
$\subseteq$-Def	$\forall a_{\iota\rightarrow o}. \forall b_{\iota\rightarrow o}. a \subseteq b \leftrightarrow \forall x_\iota. x \in a \rightarrow x \in b$
Powerset	$\forall M_{\iota\rightarrow o}. \neg\exists f_{\iota\rightarrow(\iota\rightarrow o)}. \mathrm{surj}(f, M, P(M))$

Note that based on the λ-calculus, Ω-MKRP makes no distinction between predicates and sets. $x \in a$ is only syntactic sugar for $a(x)$. "Tertium non datur", two obvious properties about equality, quite standard definitions of surjectivity, powerset, and subset are included in this formulation as premises. The theorem states that there is no surjective function f from a set M into the powerset $P(M)$. To improve readability we do not present the examples in the Ω-MKRP input syntax, but use a special TEX output facility of Ω-MKRP. In particular we suppress the type information in the proofs. In figure 3 a proof of the powerset problem can be found. It was first interactively generated with Ω-MKRP at the level of a higher-order natural deduction calculus and subsequently abstracted onto the so-called assertion level [Hua94], where justifications are largely given in terms of the application of assertions (such as definitions or theorems). By doing so, the current proof of 29 lines shown in figure 3 is obtained from the original one of 69 lines.

Although at a much more appropriate level of abstraction, this abstracted proof is still a full proof of this particular theorem. On account of this we can hardly expect that every step of such a particular proof will be useful for another proof. More likely, only some key ideas of an existing proof can be borrowed to solve a related new problem. The key steps in the current proof are listed below:

- the theorem (line 29), the three initial steps of simplifying the theorem by forall-introduction, not-introduction, and exists-elimination generating lines 28, 1, 27, 2, and 26,
- the central property that the diagonal is in the powerset (line 9), which is a key step in the indirect proof, and the two following lines, where the surjectivity definition is applied (lines 10 and 11), and
- line 16, which contains the contradiction in a concise form and the important lines of the case analysis in order to make the contradiction explicit (lines 17, 19, 20, 23, 24, and 25).

Eliminating the other intermediate steps particular to this problem we get the method `Diag-1` in figure 4.

Note that this method contains all essential steps in the original proof, and it is quite easy to fill the gaps automatically by some automated theorem prover or by further proof planning.

The task to write the initial methods has still to be carried out by the user (as in the CLAM system where all methods are user written). In our approach, there are mainly two ways to do this. Firstly a user can write them by hand. Many typical methods in our framework are written this way, such as the application of a definition, of a theorem, or of a certain property like homomorphy. Secondly initial methods can often be extracted from successful proofs. Although up to now this is done by the user too; this is quite easy,

1.	1	$\vdash$	$\exists f.\mathrm{surj}(f, M_0, P(M_0))$	(Hyp)
2.	1,2	$\vdash$	$\mathrm{surj}(f_0, M_0, P(M_0))$	(Hyp)
3.	3	$\vdash$	$x \in \lambda z.[z \in M_0 \wedge \neg[z \in f_0(z)]]$	(Hyp)
4.	3	$\vdash$	$[x \in M_0 \wedge \neg[x \in f_0(x)]]$	(LambdaE 3)
5.	3	$\vdash$	$x \in M_0$	(AndE 4)
6.		$\vdash$	$[x \in \lambda z.[z \in M_0 \wedge \neg[z \in f_0(z)]] \rightarrow x \in M_0]$	(ImpI 5 3)
7.		$\vdash$	$\forall x.x \in [\lambda z.[z \in M_0 \wedge \neg[z \in f_0(z)]] \rightarrow x \in M_0]$	(ForallI 6)
8.		$\vdash$	$\lambda z.[z \in M_0 \wedge \neg[z \in f_0(z)]] \subseteq M_0$	($\subseteq$-Def 7)
9.		$\vdash$	$\lambda z.[z \in M_0 \wedge \neg[z \in f_0(z)]] \in P(M_0)$	(PSet-Def 8)
10.	1,2	$\vdash$	$\exists y_0.[y_0 \in M_0 \wedge \lambda z.[z \in M_0 \wedge \neg[z \in f_0(z)]] = f_0(y_0)]$	(Surj-Def 2 9)
11.	1,2,11	$\vdash$	$[y_0 \in M_0 \wedge \lambda z.[z \in M_0 \wedge \neg[z \in f_0(z)]] = f_0(y_0)]$	(Hyp)
12.	1,2,11	$\vdash$	$\lambda z.[z \in M_0 \wedge \neg[z \in f_0(z)]] = f_0(y_0)$	(AndE 11)
13.		$\vdash$	$y_0 \in f_0(y_0) = y_0 \in f_0(y_0)$	(=-Ref)
14.	1,2,11	$\vdash$	$y_0 \in \lambda z.[z \in M_0 \wedge \neg[z \in f_0(z)]] = y_0 \in f_0(y_0)$	(= 12 13)
15.	1,2,11	$\vdash$	$[y_0 \in \lambda z.[z \in M_0 \wedge \neg[z \in f_0(z)]] \leftrightarrow y_0 \in f_0(y_0)]$	(=-Equiv 14)
16.	1,2,11	$\vdash$	$[[y_0 \in M_0 \wedge \neg[y_0 \in f_0(y_0)]] \leftrightarrow y_0 \in f_0(y_0)]$	(LambdaE 15)
			Case 1	
17.	1,2,11,17	$\vdash$	$y_0 \in f_0(y_0)$	(Case 1)
18.	1,2,11,17	$\vdash$	$\neg[y_0 \in f_0(y_0)]$	(16 17)
19.	1,2,11,17	$\vdash$	$\perp$	(NotE 18 17)
			Case 2	
20.	1,2,11,20	$\vdash$	$\neg[y_0 \in f_0(y_0)]$	(Case 2)
21.	1,2,11	$\vdash$	$y_0 \in M_0$	(AndE 11)
22.	1,2,11,20	$\vdash$	$y_0 \in f_0(y_0)$	(16 21 20)
23.	1,2,11,20	$\vdash$	$\perp$	(NotE 20 22)
			End of Case 2	
24.		$\vdash$	$[y_0 \in f_0(y_0) \vee \neg[y_0 \in f_0(y_0)]]$	(TND)
25.	1,2,11	$\vdash$	$\perp$	(OrE 24 19 23)
			End of Case Analysis	
26.	1,2	$\vdash$	$\perp$	(ExistsE 10 25)
27.	1	$\vdash$	$\perp$	(ExistsE 1 26)
28.		$\vdash$	$\neg[\exists f.\mathrm{surj}(f, M_0, P(M_0))]$	(NotI 27)
29.		$\vdash$	$\forall M.\neg[\exists f.\mathrm{surj}(f, M, P(M))]$	(ForallI 28)

Figure 3: ND-Proof of the **Powerset** Example

since in our approach methods essentially consist of a specification and a *declaratively* represented tactic. Such initial methods are then modified by meta-methods to suit analogous situations.

4.1 The First Generalization

In order to show how such a modification can be performed, let us look now at our second diagonalization example, namely the theorem stating that there is no surjective function from the natural numbers onto the interval $[0, 1]$. The problem can be formalized as follows:

Declarations	—			
Premises	Surj-Def			
Constraint	—			
Conclusions	16			
Declarative Content	1. 1	$\vdash$	$\exists f_{\bullet}\mathrm{surj}(f, M_0, P(M_0))$	(Hyp)
	2. 1,2	$\vdash$	$\mathrm{surj}(f_0, M_0, P(M_0))$	(Hyp)
	3. 1,2	$\vdash$	$\lambda z_{\bullet}[z \in M_0 \wedge \neg[z \in f_0(z)]] \in P(M_0)$	(PLAN)
	4. 1,2	$\vdash$	$\exists y_{0\bullet}[y_0 \in M_0 \wedge \lambda z_{\bullet}[z \in M_0 \wedge \neg[z \in f_0(z)]] = f_0(y_0)]$	(Surj-Def 2 3)
	5. 1,2,5	$\vdash$	$[y_0 \in M_0 \wedge \lambda z_{\bullet}[z \in M_0 \wedge \neg[z \in f_0(z)]] = f_0(y_0)]$	(Hyp)
	6. 1,2,5	$\vdash$	$[[y_0 \in M_0 \wedge \neg[y_0 \in f_0(y_0)]] \leftrightarrow y_0 \in f_0(y_0)]$	(PLAN 5)
			Case 1	
	7. 1,2,5,7	$\vdash$	$y_0 \in f_0(y_0)$	(Case 1)
	8. 1,2,5,7	$\vdash$	$\perp$	(PLAN 6 7)
			Case 2	
	9. 1,2,5,9	$\vdash$	$\neg[y_0 \in f_0(y_0)]$	(Case 2)
	10. 1,2,5,9	$\vdash$	$\perp$	(PLAN 6 9)
			End of Case 2	
	11.	$\vdash$	$[y_0 \in f_0(y_0) \vee \neg[y_0 \in f_0(y_0)]]$	(TND)
	12. 1,2,5	$\vdash$	$\perp$	(OrE 11 8 10)
			End of Case Analysis	
	13. 1,2	$\vdash$	$\perp$	(ExistsE 4 12)
	14. 1	$\vdash$	$\perp$	(ExistsE 1 13)
	15.	$\vdash$	$\neg[\exists f_{\bullet}\mathrm{surj}(f, M_0, P(M_0))]$	(NotI 14)
	16.	$\vdash$	$\forall M_{\bullet}\neg[\exists f_{\bullet}\mathrm{surj}(f, M, P(M))]$	(ForallI 15)
Procedural Content	`schema-interpreter`			

Figure 4: Method `Diag-1`

TND	$\forall x_{o\bullet} x \vee \neg x$
=-Ref	$\forall x_{\iota\bullet} x = x$
Surj-Def	$\forall f_{\iota\to(\iota\to\iota)\bullet}\forall a_{\iota\to o\bullet}\forall b_{(\iota\to\iota)\to o\bullet}\mathrm{surj}(f,a,b) \leftrightarrow$ $[\forall x_{\iota\bullet} x \in a \to f(x) \in b \wedge$ $(\forall x_{\iota\to\iota\bullet} x \in b \to (\exists y_{\iota\bullet}(y \in a \wedge x = f(y))))]$
Digits-0-1	$\mathrm{dig}(0) \wedge \mathrm{dig}(1)$
$0 \neq 1$	$0 \neq 1$
[0,1]-Def	$\forall h_{\iota\to\iota\bullet} h \in [0,1] \leftrightarrow (\forall n_{\iota\bullet} n \in \mathbb{N} \to \mathrm{dig}(h(n)))$
NatReal	$\neg\exists f_{\iota\to(\iota\to\iota)\bullet}\mathrm{surj}(f, \mathbb{N}, [0,1])$

In this formulation, the interval $[0,1]$ is defined as the set of all functions from the natural numbers into the digits, which corresponds to a decimal or binary representation of the real numbers depending on how many digits you use. (Indeed the proof is independent of the number of digits. We neglect here the problem of periods in the largest digit.)

The method `Diag-1` is not directly applicable to this problem since the conclusions do not match each other. Here we want to show $\neg\exists f_{\iota\to(\iota\to\iota)\bullet}\mathrm{surj}(f, \mathbb{N}, [0,1])$, while method `Diag-1` proves $\forall M_{\bullet}\neg[\exists f_{\bullet}\mathrm{surj}(f, M, P(M))]$. Now we want to see how the proof fragment in method `Diag-1` can be transformed in order to cope with the second problem.

Reformulation of `Diag-1` to `Diag-1'`: First we have to apply the meta-method `Cut-Sub-method` which recognizes that the new theorem is similar to line 15 in method `Diag-1`, an

Declarations	—
Premises	Surj-Def,bool,value
Constraint	—
Conclusions	15
Declarative Content	1. 1 $\vdash \exists f_{\bullet}\mathrm{surj}(f, \mathbb{N}, [0,1])$ (Hyp) 2. 1,2 $\vdash \mathrm{surj}(f_0, \mathbb{N}, [0,1])$ (Hyp) 3. 1,2 $\vdash \lambda z_{\mathbb{N}}{}_{\bullet}\mathrm{value}(\neg[\mathrm{bool}(f_0(z)(z))]) \in [0,1]$ (PLAN) ——— 4. 1,2 $\vdash \exists y_{\mathbb{N}}{}_{\bullet}\lambda z_{\mathbb{N}}{}_{\bullet}\mathrm{value}(\neg[\mathrm{bool}(f_0(z)(z))]) = f_0(y)$ (Surj-Def 2 3) 5. 1,2,5 $\vdash \lambda z_{\mathbb{N}}{}_{\bullet}\mathrm{value}(\neg[\mathrm{bool}(f_0(z)(z))]) = f_0(y_0)$ (Hyp) 6. 1,2,5 $\vdash \neg\mathrm{bool}(f_0(y_0)(y_0)) \leftrightarrow \mathrm{bool}(f_0(y_0)(y_0))$ (PLAN 5) ——— Case 1 ——— 7. 1,2,5,7 $\vdash \mathrm{bool}(f_0(y_0)(y_0))$ (Case 1) 8. 1,2,5,7 $\vdash \bot$ (PLAN 6 7) ——— Case 2 ——— 9. 1,2,5,9 $\vdash \neg\mathrm{bool}(f_0(y_0)(y_0))$ (Case 2) 10. 1,2,5,9 $\vdash \bot$ (PLAN 6 9) ——— End of Case 2 ——— 11. $\vdash \mathrm{bool}(f_0(y_0)(y_0)) \vee \neg\mathrm{bool}(f_0(y_0)(y_0))$ (TND) 12. 1,2,5 $\vdash \bot$ (OrE 11 8 10) ——— End of Case Analysis ——— 13. 1,2 $\vdash \bot$ (ExistsE 4 12) 14. 1 $\vdash \bot$ (ExistsE 1 13) 15. $\vdash \neg[\exists f_{\bullet}\mathrm{surj}(f, \mathbb{N}, [0,1])]$ (NotI 14)
Procedural Content	`schema-interpreter`

Figure 5: Method `Diag-2`

intermediate conclusion of `Diag-1`. `Cut-Submethod` creates a new method `Diag-1'` from `Diag-1` by deleting the last line in the declarative content and by updating the conclusions slot: 16 becomes 15.

The meta-method `Cut-Submethod` looks in a breadth first search in the proof tree of an available method for a node that corresponds to an open line of the partial proof of the current problem (here the target theorem). Since this meta-method can be applied in many cases it is important to heuristically restrict its application, for instance by limiting the depth of the node in the proof tree of the method or by considering only lines without assumptions.

Reformulation of `Diag-1'` to `Diag-2`: This reformulation is carried out by a meta-method called `set2func`. Intuitively it transforms certain *sets* occurring in a method into *functions*. In this example, the set of sets (powerset) is transformed to a set of functions (the interval [0,1]). The first step carries out the parallel term mapping established by matching the two conclusion lines, specified below (the last two are the main reformulations, while the others are just type adjustments).

- $f_{\iota\to(\iota\to o)} \mapsto f_{\iota\to(\iota\to\iota)}$
- $\mathrm{surj}_{(\iota\to(\iota\to o))\times(\iota\to o)\times((\iota\to o)\to o)\to o} \mapsto \mathrm{surj}_{(\iota\to(\iota\to\iota))\times(\iota\to o)\times((\iota\to\iota)\to o)\to o}$
- $M_{0_{\iota\to o}} \mapsto \mathbb{N}_{\iota\to o}$

- $P(M_0)_{(\iota\to o)\to o} \mapsto [0,1]_{(\iota\to\iota)\to o}$

Furthermore the constant f_0 (which instantiates the variable f) must be transformed in the same way. These mappings, however, may introduce type inconsistencies which can be eliminated by introducing new function symbols adapting the types. In our example we use the procedures `term2formula` and `formula2term` which introduce the function symbols "bool" and "value" for this adaptation.

The procedure `term2formula` replaces a term t with $\text{bool}(t)$. Semantically, the function symbol "bool" maps some specific ι-terms to some corresponding truth values, namely the meta-variable A (of type ι, later instantiated to 1) to true and the meta-variable B (also of type ι and later instantiated to 0) to false. This is accompanied by inserting the extra axiom $\text{bool}(A) \wedge \neg\text{bool}(B)$. For other elements "bool" is not specified.

The procedure `formula2term` replaces a formula φ with $\text{value}(\varphi)$. The function symbol "value" maps true to the element A and false to the element B. This can be specified by the formula $\forall x_{o}\text{.}(x \leftrightarrow \text{value}(x) = A) \wedge (\neg x \leftrightarrow \text{value}(x) = B)$.

The specification of bool and value are not represented in the method `Diag-2` directly, but for its applicability they have to be available in the background theory just as the definition of the surjectivity.

Furthermore we use a procedure `predicate2sort`. It creates new sort symbols from predicate symbols and can be considered as the inverse of the relativization. In our example the expression

$$\lambda z\text{.}[\mathbb{N}(z) \wedge \neg[\text{bool}(f_0(z)(z))] \in [0,1]$$

is transformed to

$$\lambda z_{\mathbb{N}}\text{.}(\neg[\text{bool}(f_0(z)(z))]) \in [0,1].$$

Now, the meta-method `set2func` is basically a saturation algorithm iterating the three procedures above through the proof tree and the term tree in the proof nodes in a bottom-up way. Refinements have to be incorporated for handling indeterminism. For instance, the first two procedures are always both applicable in the case of an equality with disagreeing term types. We are also investigating an alternative approach, where only `formula2term` is used. Although this alternative eliminates the indeterminism mentioned above, it has the drawback that the function $\lambda x\text{.}\text{value}(\neg(\text{bool}(x)))$, fixpoint-free on the digits, cannot be automatically synthesized (lines 3, 4, and 5 in `Diag-2`).

Concretely, by eliminating the type inconsistencies related to $f_0(z)(z)$ and $f_0(y_0)(y_0)$ (they are of type ι but in order to be used as proposition they have to be of type o) with the procedure `term2formula` in the lines 3, 4, 5, 6, 7, 9 and 11 we get:

3. ... $\vdash$ $\lambda z\text{.}[\mathbb{N}(z) \wedge \neg[\text{bool}(f_0(z)(z))]] \in [0,1]$
4. ... $\vdash$ $\exists y\text{.}[\mathbb{N}(y) \wedge \lambda z\text{.}[\mathbb{N}(z) \wedge \neg[\text{bool}(f_0(z)(z))]] = f_0(y)]$
5. ... $\vdash$ $\mathbb{N}(y_0) \wedge \lambda z\text{.}[\mathbb{N}(z) \wedge \neg[\text{bool}(f_0(z)(z))]] = f_0(y_0)$
6. ... $\vdash$ $[\mathbb{N}(y_0) \wedge \neg\text{bool}(f_0(y_0)(y_0))] \leftrightarrow \text{bool}(f_0(y_0)(y_0))$
7. ... $\vdash$ $\text{bool}(f_0(y_0)(y_0))$
9. ... $\vdash$ $\neg\text{bool}(f_0(y_0)(y_0))$
11. ... $\vdash$ $\text{bool}(f_0(y_0)(y_0)) \vee \neg\text{bool}(f_0(y_0)(y_0))$

After handling the type inconsistencies related to $f_0(y)$ and $f_0(y_0)$ with the procedure `predicate2sort` and `formula2term`, the lines 4 and 5 above become:

4. ... $\vdash$ $\exists y_{\mathbb{N}}\text{.}\lambda z_{\mathbb{N}}\text{.}\text{value}(\neg[\text{bool}(f_0(z)(z))]) = f_0(y)$
5. ... $\vdash$ $\lambda z_{\mathbb{N}}\text{.}\text{value}(\neg[\text{bool}(f_0(z)(z))]) = f_0(y_0)$

Declarations	$\overline{X}, \overline{Y}, \overline{Z}, \overline{\text{non}}$
Premises	Surj-Def
Constraint	—
Conclusions	15
Declarative Content	1. 1 $\vdash \exists f_\bullet \text{surj}(f, \overline{X}, \overline{Y})$ (Hyp) 2. 1,2 $\vdash \text{surj}(f_0, \overline{X}, \overline{Y})$ (Hyp) 3. 1,2 $\vdash \lambda z_{\overline{X}} \bullet \overline{\text{non}}(\overline{Z}(f_0(z)(z))) \in \overline{Y}$ (PLAN) ——— 4. 1,2 $\vdash \exists y_{\overline{X}} \bullet \lambda z_{\overline{X}} \bullet \overline{\text{non}}(\overline{Z}(f_0(z)(z))) = f_0(y)$ (Surj-Def 2 3) 5. 1,2,5 $\vdash \lambda z_{\overline{X}} \bullet \overline{\text{non}}(\overline{Z}(f_0(z)(z))) = f_0(y_0)$ (Hyp) 6. 1,2,5 $\vdash \neg\overline{Z}(f_0(y_0)(y_0)) \leftrightarrow \overline{Z}(f_0(y_0)(y_0))$ (PLAN 5) ——— Case 1 ——— 7. 1,2,5,7 $\vdash \overline{Z}(f_0(y_0)(y_0))$ (Case 1) 8. 1,2,5,7 $\vdash \perp$ (PLAN 6 7) ——— Case 2 ——— 9. 1,2,5,9 $\vdash \neg\overline{Z}(f_0(y_0)(y_0))$ (Case 2) 10. 1,2,5,9 $\vdash \perp$ (PLAN 6 9) ——— End of Case 2 ——— 11. $\vdash \overline{Z}(f_0(y_0)(y_0)) \vee \neg\overline{Z}(f_0(y_0)(y_0))$ (TND) 12. 1,2,5 $\vdash \perp$ (OrE 11 8 10) ——— End of Case Analysis ——— 13. 1,2 $\vdash \perp$ (ExistsE 4 12) 14. 1 $\vdash \perp$ (ExistsE 1 13) 15. $\vdash \neg[\exists f_\bullet \text{surj}(f, \overline{X}, \overline{Y})]$ (NotI 14)
Procedural Content	`schema-interpreter`

Figure 6: Method `Diag-1-2`

The type inconsistency related to $[0, 1]$ in line 3 can be eliminated by `predicate2sort` and `formula2term` again:

3. ... $\vdash \lambda z_{\mathbb{N}} \bullet \text{value}(\neg[\text{bool}(f_0(z)(z))]) \in [0, 1]$

Finally we get the method `Diag-2` in figure 5. When creating it from `Diag-1`$'$ by the meta-method `set2func` the meta-variables A and B are instantiated to 1 and 0.

To avoid the explosion of the method base and to abstract away from concrete proofs, meta-methods are also incorporated to extract more general methods from existing ones. In this example, we want to construct a method `Diag-1-2`, which covers the first two cases. This can be done by the meta-method `Abstract` that essentially abstracts the disagreeing terms in `Diag-1`$'$ and `Diag-2` to meta-variables (see also [BW93]).

By the generation of the most specific generalization of $\neg[\exists f_\bullet \text{surj}(f, M_0, P(M_0))]$ and $\neg[\exists f_\bullet \text{surj}(f, \mathbb{N}, [0, 1])]$, we get the generalized conclusion $\neg[\exists f_\bullet \text{surj}(f, \overline{X}, \overline{Y})]$. Note that meta-variables are overlined. In the same way $f_0(y_0)(y_0)$ in `Diag-1`$'$ and $\text{bool}(f_0(y_0)(y_0))$ in `Diag-2` are generalized to $\overline{Z}(f_0(y_0)(y_0))$. Finally "$\neg$" in `Diag-1` and "value$\circ\neg$" in `Diag-2` are abstracted to a new meta-variable $\overline{\text{non}}$. With these generalizations, we get the method `Diag-1-2`, shown in figure 6.

The following instantiations can be used to obtain the first two methods.

Diag-1-2	first method	second method
$\overline{X}$	M_0	$\mathbb{N}$
$\overline{Y}$	$P(M_0)$	$[0,1]$
$\overline{Z}$	$\lambda x_\bullet x$	$\lambda x_\bullet \mathrm{bool}(x)$
$\overline{\mathrm{non}}$	$\neg$	$\lambda x_\bullet \mathrm{value}(\neg(x))$

4.2 The Second Generalization

While the first two problems are closely related, this example should illustrate how the previous method can be adapted to a fairly distinct problem, namely the Halting Problem. The theorem states that there is no binary computable function (there is no h with $T_2(h)$) which decides for unary computable functions (Turing machines), whether they halt or not (that is, defined$(t(x))$ iff $h(t,x) = 0$ for all t with $T_1(t)$ and for all x in $\mathbb{N}$). We are pointed to this third example by our colleague Melis, another formulation of the problem can be found in [Mel94].

We formalize the problem in the following way:

TND	$\forall x_{o\bullet} x \vee \neg x$
Ext	$\forall f_{\mathbb{N}\to U\bullet} \forall g_{\mathbb{N}\to U\bullet} \forall x_{\mathbb{N}\bullet} f = g \to f(x) = g(x)$
Gödel	$\forall t_{\mathbb{N}\to U\bullet} T_1(t) \to \exists n_{\mathbb{N}\bullet} e(n) = t$
if-Comp	$\forall f_{((\mathbb{N}\to U),\mathbb{N})\to D\bullet} T_2(f) \to$ $\forall x_{U\bullet} \forall y_{U\bullet} T_1(\lambda z_{\mathbb{N}\bullet} \mathrm{if}(f(e(z),z) = 0, x, y))$
if-Def	$\forall P_{o\bullet} \forall x_{U\bullet} \forall y_{U\bullet} P \to \mathrm{if}(P,x,y) = x \wedge$ $\neg P \to \mathrm{if}(P,x,y) = y$
defined	$\neg\mathrm{defined}(u) \wedge \mathrm{defined}(0)$
Halting	$\neg\exists h_{((\mathbb{N}\to U),\mathbb{N})\to D\bullet} T_2(h) \wedge \forall t_{\mathbb{N}\to U\bullet} T_1(t)$ $\to \forall x_{\mathbb{N}\bullet} \mathrm{defined}(t(x)) \leftrightarrow h(t,x) = 0$

In this formalization we use the following sorts: $\mathbb{N}$ denotes the set of natural numbers. The symbol u represents the non-terminating function. U is the union of $\mathbb{N}$ and $\{u\}$. D denotes the set $\{0,1\}$.

In order to prove the theorem we need the Gödel enumeration theorem that there is an enumeration function e so that for every unary computable function t there is a natural number n with $e(n)$ corresponding to t. In addition the application of e to any natural number is always a computable function. Furthermore, we use some obvious definitions and the lemma that for a total and computable function f the function $\lambda z_{\mathbb{N}\bullet} \mathrm{if}(f(e(z),z) = 0, x, y)$ is computable too, where "if(condition,then,else)" has the usual semantics.

In order to generate a method `Diag-1-2-3` from the method `Diag-1-2` for solving the Halting Problem, we employ the meta-method `Abstract` again, which has been used to produce method `Diag-1-2` from the methods `Diag-1'` and `Diag-2`. This meta-method can be applied to a single method and a problem specification as well. Since less information is given in a specification of a problem then in a method, this mode requires more user-guidance. In order to perform this abstraction generalizations like the following two are applied.

- `split-symbol` replaces different occurrences of one symbol (in this case f_0) by two different meta-variables (here $\overline{g}$ and $\overline{h}$). This is necessary in our example, since in `Diag-1-2` f_0 serves both as the function assumed in the theorem as well as the surjective function which is used to derive the membership of the diagonalization element.

Declarations	$\overline{X}, \overline{Y}, \overline{Z}, \overline{App}, \overline{\text{non}}, \overline{f}, \overline{g}, \overline{h}, \overline{F}, \overline{F'}, \overline{J}$
Premises	—
Constraint	$\overline{F'} = \mathsf{applysubst}(\overline{f} \mapsto \overline{g}, \overline{F})$
Conclusions	15
Declarative Content	1. 1 $\vdash \exists \overline{f} \centerdot \overline{F}$ (Hyp) 2. 1,2 $\vdash \overline{F'}$ (Hyp) 3. 1,2 $\vdash \lambda z_{\overline{X}} \centerdot \overline{\text{non}}(\overline{Z}(\overline{App}(\overline{h}(z), z))) \in \overline{Y}$ (PLAN 2) ——— 4. 1,2 $\vdash \exists y_{\overline{X}} \centerdot \overline{h}(y) = \lambda z_{\overline{X}} \centerdot \overline{\text{non}}(\overline{Z}(\overline{App}(\overline{h}(z), z)))$ ($\overline{J}$ 2 3) 5. 1,2,5 $\vdash \overline{h}(y_0) = \lambda z_{\overline{X}} \centerdot \overline{\text{non}}(\overline{Z}(\overline{App}(\overline{h}(z), z)))$ (Hyp) 6. 1,2,5 $\vdash \neg \overline{Z}(\overline{App}(\overline{h}(y_0), y_0)) \leftrightarrow \overline{Z}(\overline{App}(\overline{h}(y_0), y_0))$ (PLAN 5) ——— Case 1 ——— 7. 1,2,5,7 $\vdash \overline{Z}(\overline{App}(\overline{h}(y_0), y_0))$ (Case 1) 8. 1,2,5,7 $\vdash \bot$ (PLAN 6 7) ——— Case 2 ——— 9. 1,2,5,9 $\vdash \neg \overline{Z}(\overline{App}(\overline{h}(y_0), y_0))$ (Case 2) 10. 1,2,5,9 $\vdash \bot$ (PLAN 6 9) ——— End of Case 2 ——— 11. $\vdash \overline{Z}(\overline{App}(\overline{h}(y_0), y_0)) \lor \neg \overline{Z}(\overline{App}(\overline{h}(y_0), y_0))$ (TND) 12. 1,2,5 $\vdash \bot$ (OrE 11 8 10) ——— End of Case Analysis ——— 13. 1,2 $\vdash \bot$ (ExistsE 4 12) 14. 1 $\vdash \bot$ (ExistsE 1 13) 15. $\vdash \neg[\exists \overline{f} \centerdot \overline{F}]$ (NotI 14)
Procedural Content	`schema-interpreter`

Figure 7: Method `Diag-1-2-3`

In the third problem, however, the second role is played by Gödel's enumeration function e.

- `generalize-application` maps an application $x(y)$ to an expression $\overline{App}(x, y)$.

We finally arrive at the method `Diag-1-2-3`, see figure 7.

Informally the proof of the Halting Problem contains the following key steps. Suppose that there is a function "halt", this corresponds to line 1 and 2 in the method above. We get a unary computable function $\lambda z_{\mathsf{IN}} \centerdot \text{if}(\text{halt}(e(z), z) = 0, u, 0)$ (line 3). According to the Gödel enumeration theorem there is a natural number y_0 so that $e(y_0) = \lambda z_{\mathsf{IN}} \centerdot \text{if}(\text{halt}(e(z), z) = 0, u, 0)$ (line 4 and 5). With the extensionality theorem we deduce that $e(y_0)(y_0) = \text{if}(\text{halt}(e(y_0), y_0) = 0, u, 0)$, what can be used to derive line 6, which leads to a contradiction.

Finally we have arrived at a general method covering all three diagonalization problems discussed in this section. Nevertheless some problem specific information has been lost during the abstraction. If a planner is confronted with a concrete problem, it has to instantiate the meta-variables of the method. The instantiations for the three examples are summarized in the table displayed in figure 8.

It is worth comparing the two distinct modes of meta-planning allowed in our approach. First, existing methods can be generalized to more general methods, which in turn are used

`Diag 1-2-3`	first method	second method	third method
$\overline{X}$	M_0	$\mathbb{N}$	$\mathbb{N}$
$\overline{Y}$	$P(M_0)$	$[0,1]$	T_1
$\overline{Z}$	$\lambda x_\bullet x$	$\lambda x_\bullet \mathrm{bool}(x)$	$\lambda x_\bullet x = 0$
$\overline{App}$	$\lambda x_\bullet \lambda y_\bullet x(y)$	$\lambda x_\bullet \lambda y_\bullet x(y)$	$\lambda x_\bullet \lambda y_\bullet \mathrm{halt}(x, y)$
$\overline{\mathrm{non}}$	$\neg$	$\lambda x_\bullet \mathrm{value}(\neg(x))$	$\lambda x_\bullet \mathrm{if}(x, u, 0)$
$\overline{f}$	f	f	h
$\overline{g}$	f_0	f_0	halt
$\overline{h}$	f_0	f_0	e
$\overline{F}$	$\mathrm{surj}(f, M_0, P(M_0))$	$\mathrm{surj}(f, \mathbb{N}, [0,1])$	$T_2(h) \wedge \forall t_{\mathbb{N}\to U \bullet} T_1(t) \ldots$
$\overline{F'}$	$\mathrm{surj}(f_0, M_0, P(M_0))$	$\mathrm{surj}(f_0, \mathbb{N}, [0,1])$	$T_2(\mathrm{halt}) \wedge \forall t_{\mathbb{N}\to U \bullet} T_1(t) \ldots$
$\overline{J}$	Surj-Def	Surj-Def	Gödel

Figure 8: Instantiations of Meta-variables in `Diag-1-2-3`

as starting points for solving further problems. For instance, the method `Diag-1-2-3` may be obtained and instantiated to solve concrete problems. Second it also supports the adaptation of concrete methods like `Diag-1` by meta-methods. The first procedure has the advantage that in the long run the method base will be populated by very general methods. There is a significant disadvantage as well, however, that important information may get lost. For instance, to solve the first problem with the method `Diag-1-2-3`, you must know (or retrieve the information) that the variable $\overline{Z}$ is bound to $\lambda x_\bullet x$. Moreover, such information is not only crucial for a direct application. From this binding it is comparatively easier to come to the binding $\lambda x_\bullet \mathrm{bool}(x)$ in the second example by adapting the concrete method `Diag-1`. Therefore, one point of future investigation is to store binding informations such as the table in figure 8 along with generalized methods, so that it can be reused in the planning process.

5 Conclusion and Future Development

The problem solving competence of a mathematician relies heavily on his/her ability to adapt problem solving knowledge to new situations where existing methods are not directly applicable. Up to now this has not received enough attention in the field of automated theorem proving. In order to mechanize parts of this ability, we propose a declarative extension to the proof planning approach developed by Bundy.

This technical report is aimed to provide evidence that our goal can be achieved with our declarative approach. To do this, we have presented how methods making use of conjunctions can be transformed to those making use of universal quantifications and how a class of diagonalization methods can be represented in our framework. In the case of the diagonalization method, we have presented a detailed trace protocol of the evolution: first an initial method is extracted from a concrete proof. This initial method is then adapted for the subsequent problems, and more general methods can be obtained by abstracting existing methods. Most interestingly we come up with a fairly abstract method capable of dealing with all the three problems, since it captures the very key idea of diagonalization.

The modifications used are, of course, sensitive to the representation of the problems. If the formulations are more different then the chosen ones, the reformulation efforts will increase. Note that, however, the meta-methods used do not rely on any particular properties of diagonalization and most of them have been employed already in other contexts.

Meta-methods can be incorporated into a planning algorithm. To do this, first it must be decided when to interrupt the planning with methods, in order to create a new method with meta-methods. Second for the current proof situation an adequate pair of a method and a meta-method have to be chosen from the knowledge base. We believe that there can hardly be any general answer to this problem and we have to rely on heuristics. In an interactive proof development environment like Ω-MKRP [HKK+94] the user might want to make this choice himself. Therefore our main emphasis lies in the task of providing the user with heuristic support for this choice. Even more challenging would be an automation, of course. A trivial answer would be to apply all existing meta-methods on all existing methods and then choose heuristically the best. Such a procedure can be fairly expensive with a large knowledge base. The first heuristics for choosing a method to adapt we will investigate are listed below:

- Organize methods in a hierarchy of mathematical theories and prefer methods that belong to the same theory as the current problem or whose theory is close to that of the problem in the hierarchy.
- Use general conflict solving strategies like those of OPS5, for instance, favor the methods and meta-methods with the most specific specification.

Of course only successful methods generated in a short-term memory are integrated into the permanent base of methods. Another way to reduce the cost of the operation would be to create only the specification of the methods to be generated, select one for application and create the tactic part by need.

Admittedly, the heuristic control concerning the choice of methods and meta-methods with a specific instantiation, as well as the interleaving of planning and meta-level planning, remain as open problems.

References

[BSvH+93] Alan Bundy, Andrew Stevens, Frank van Harmelen, Andrew Ireland, and Alan Smaill. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, **62**:185–253, 1993.

[BvHHS90] 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.

[BW93] David Basin and Toby Walsh. Difference unification. In Ruzena Bajcsy, editor, *Proceedings of the 13th IJCAI*, pages 116–122, Chambéry, France, 1993. Morgan Kaufmann, San Mateo, California, USA.

[Chu40] Alonzo Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, **5**:56–68, 1940.

[Con86] Robert L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1986.

[GMW79] Michael Gordon, Robin Milner, and Christopher Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation.* LNCS 78. Springer Verlag, Berlin, Germany, 1979.

[GT94] Fausto Giunchiglia and Paolo Traverso. Program tactics and logic tactics. In Frank Pfenning, editor, *Proceedings of LPAR*, pages 16–30, Kiev, Ukraine, 1994. Springer Verlag, Berlin, Germany, LNAI 822.

[HK91] Xiaorong Huang and Manfred Kerber. Theorem proving as an interleaving process of planning and verification. unpublished manuscript, 1991.

[HKK+94] Xiaorong Huang, Manfred Kerber, Michael Kohlhase, Erica Melis, Dan Nesmith, Jörn Richts, and Jörg Siekmann. Ω-MKRP: A proof development environment. In Alan Bundy, editor, *Proceedings of the 12th CADE*, pages 788–792, Nancy, 1994. Springer Verlag, Berlin, Germany, LNAI 814.

[HKKR94] Xiaorong Huang, Manfred Kerber, Michael Kohlhase, and Jörn Richts. Adapting methods to novel tasks in proof planning. In Bernhard Nebel and Leonie Dreschler-Fischer, editors, *KI-94: Advances in Artificial Intelligence – Proceedings of KI-94, 18th German Annual Conference on Artificial Intelligence*, pages 379–390, Saarbrücken, Germany, 1994. Springer Verlag, Berlin, Germany, LNAI 861.

[HKRS94] Xiaorong Huang, Manfred Kerber, Jörn Richts, and Arthur Sehn. Planning mathematical proofs with methods. *Journal of Information Processing and Cybernetics, EIK*, **30**:277–291, 1994.

[Hua94] Xiaorong Huang. Reconstructing proofs at the assertion level. In Alan Bundy, editor, *Proceedings of the 12th CADE*, pages 738–752, Nancy, France, 1994. Springer Verlag, Berlin, Germany, LNAI 814.

[Ire92] Andrew Ireland. The use of planning critics in mechanizing inductive proofs. In A. Voronkov, editor, *Proceedings of LPAR*, pages 178–189, St. Petersburg, Russia, 1992. Springer Verlag, Berlin, Germany, LNAI 624.

[Ker89] Manfred Kerber. Some aspects of analogy in mathematical reasoning. In Klaus P. Jantke, editor, *Analogical and Inductive Inference; International Workshop AII '89, Reinhardsbrunn Castle, GDR*, pages 231–242. Springer Verlag, Berlin, Germany, LNAI 397, October 1989.

[KP96] Manfred Kerber and Axel Präcklein. Using tactics to reformulate formulae for resolution theorem proving. Annals of Mathematics and Artificial Intelligence, forthcoming, 1996.

[Mel94] Erica Melis. Representing and reformulating diagonalization methods. Technical Report CMU-CS-94-174, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, 1994.

[Pól45] George Pólya. *How to Solve It.* Princeton University Press, Princeton, New Jersey, USA, also as Penguin Book, 1990, London, United Kingdom, 1945.

[Van89] Kurt VanLehn. Problem solving and cognitive skill acquisition. In Michael I. Posner, editor, *Foundations of Cognitive Science*, chapter 14. MIT Press, Cambridge, Massachusetts, 1989.

[WOLB84] Larry Wos, Ross Overbeek, Ewing Lusk, and Jim Boyle. *Automated Reasoning – Introduction and Applications.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1984.