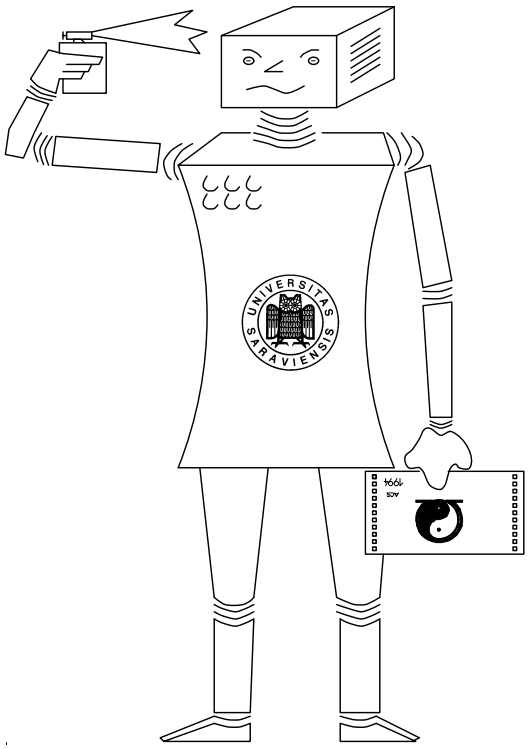
UNIVERSITÄT DES SAARLANDES
FACHBEREICH INFORMATIK
D-66041 SAARBRÜCKEN
GERMANY

WWW: http://js-sfbsun.cs.uni-sb.de/pub/www/



Adapting the Diagonalization Method by Reformulations

Xiaorong Huang Manfred Kerber
Lassaad Cheikhrouhou

Published as: In Alon Levy and Pandu Nayak, eds., *Proceedings of the Symposium on Abstraction, Reformulation, and Approximation, SARA-95*, Ville d'Esterel, Canada, 1995, forthcoming.

Adapting the Diagonalization Method by Reformulations

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

Abstract

Extending the plan-based paradigm for automated theorem proving, we developed in previous work a declarative approach towards representing methods in a proof planning framework to support their mechanical modification. This paper presents a detailed study of a class of particular methods, embodying variations of a mathematical technique called *diagonalization*. The purpose of this paper is mainly twofold. First we demonstrate that typical mathematical methods can be represented in our framework in a natural way. Second we illustrate our philosophy of proof planning: besides planning with a fixed repertoire of methods, meta-methods create new methods by modifying existing ones. With the help of three different diagonalization problems we present an example trace protocol of the evolution of methods: an initial method is extracted from a particular successful proof. This initial method is then reformulated for the subsequent problems, and more general methods can be obtained by abstracting existing methods. Finally 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.

1 Introduction

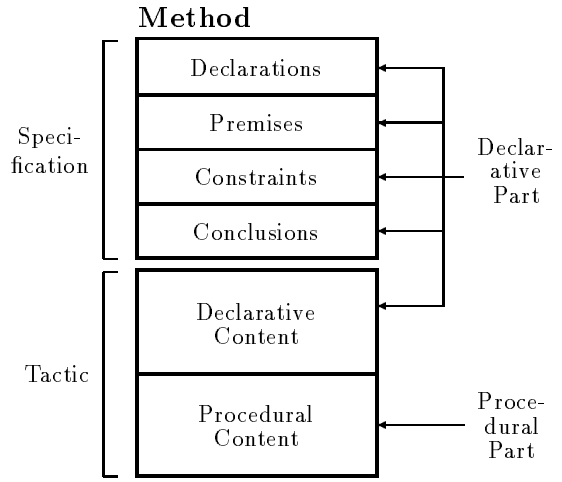
A central concept of knowledge based reasoning in mathematics and proof planning is that of a *method*. A method contains a piece of knowledge for solving or simplifying problems or transforming them into a form that is easier to solve. In this sense, methods can be quite general such as finding proofs by a case analysis or mathematical induction, or the advice to expand a specific definition.

While specific mathematical techniques have been widely implemented as so-called tactics in deduction systems like LCF [Gordon *et al.*, 1979] or Nuprl [Constable *et al.*, 1986], and extended to methods for proof planning in the CLAM system [Bundy *et al.*, 1990; 1993], the important question of adapting methods to novel situations remains unsolved in these frameworks. However, such adaptions are very important, if we want to simulate the human problem solving behaviour. It is one of the main features contributing to the problem solving competence of mathematicians that they can extend their current problem solving repertoire by adapting existing methods to suit novel situations (see [Pólya, 1945] for mathematical reasoning and [VanLehn, 1989] for general problem solving). In previous work [Huang *et al.*, 1994a] we have developed a declarative approach for formulating methods, in order to mechanize some aspects of this procedure. The declarative representation of methods enables their mechanical adaption by so-called meta-methods. To achieve the same effect in a framework where tactics are pure procedures, we would be confronted with the much more difficult problem of adapting procedures. Actually the diagonalization problems have been used from the very beginning as a nontrivial example serving as a guideline to conceptualize our approach [Huang and Kerber, 1991].

The purpose of this paper is twofold. Firstly it is aimed to show that methods at different level of abstraction can be formulated in a fairly natural way in our approach. To this end we present three methods for solving diagonalization problems. While the first method is actually a fragment of a machine-generated proof of a particular problem, the last one represents an abstracted version of diagonalization, which applies to all three different problems. Secondly and most essentially, this paper demonstrates how the repertoire of methods evolves via the application of so-called meta-methods. To mechanize complex mathematical techniques like diagonalization with all its diversities, appropriate support for mechanical modifications is necessary since it is impossible to come up with one general method which solves all problems that a human mathematician may consider as a variation of diagonalization. The paper proceeds as follows: The next section illustrates our approach and exemplifies it by describing the first method for solving the theorem that the powerset of a set has greater cardinality than the set itself. In section 3 we stepwise modify this method to cope with the proofs of the theorem that the continuum has bigger cardinality than the natural numbers and of the halting problem. Besides generating new specific methods for new problems, meta-methods are defined to generalize existing methods.

2 A Declarative Representation for Methods

In our approach a method is a 6-tuple with the components:



Declarations: A signature that declares the meta-variables used in the method,
Premises: A list of proof lines used by this method to prove the conclusion,
Constraints: Additional applicability conditions,
Conclusions: A list of proof lines which are proved by this method,
Declarative Content: A piece of declarative knowledge interpreted by the procedural content,
Procedural Content: Either a standard procedure interpreting the declarative content, or a special purpose inference procedure.

In the following we illustrate how our first diagonalization method is extracted from a successful proof, constructed with the proof development environment Ω-MKRP [Huang *et al.*, 1994b]. The first diagonalization problem we examine is the *powerset* theorem stating that the cardinality of the powerset of a set is bigger then the cardinality of the set itself. Ω-MKRP uses a typed higher-order input language corresponding roughly to Church's λ-calculus [Church, 1940], and a natural deduction calculus as its main output proof formalism.

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 1 a proof of the powerset problem can be found. It was first interactively generated with Ω-MKRP on the level of a higher-order natural deduction calculus and subsequently abstracted onto the so-called assertion level [Huang, 1994], 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 1 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 following two lines, where the surjectivity definition is applied (lines 10 and 11), and
- line 16, which contains the contradiction in a concise form, the case analysis with the first lines and last lines in order to make the contradiction explicit, as well as the tertium non datur rule in order to bring the cases together (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 2.

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 fulfilled 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, 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 [Huang *et al.*, 1994a].

3 Modification of Methods

The standard practice of proof planning is to use a fixed repertoire of methods in order to produce a complete proof plan [Bundy *et al.*, 1993]. Difficulties arise when we encounter new problems exceeding the power of existing methods. To overcome this, a main feature contributing

1.	1	$\vdash$	$\exists f_{\blacksquare}\mathrm{surj}(f, M_0, P(M_0))$	(Hyp)
2.	1,2	$\vdash$	$\mathrm{surj}(f_0, M_0, P(M_0))$	(Hyp)
3.	3	$\vdash$	$x \in \lambda z_{\blacksquare}[z \in M_0 \wedge \neg[z \in f_0(z)]]$	(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.		$\vdash$	$[x \in \lambda z_{\blacksquare}[z \in M_0 \wedge \neg[z \in f_0(z)]] \rightarrow x \in M_0]$	(ImpI 5 3)
7.		$\vdash$	$\forall x_{\blacksquare} x \in [\lambda z_{\blacksquare}[z \in M_0 \wedge \neg[z \in f_0(z)]] \rightarrow x \in M_0]$	(ForallI 6)
8.		$\vdash$	$\lambda z_{\blacksquare}[z \in M_0 \wedge \neg[z \in f_0(z)]] \subseteq M_0$	($\subseteq$-Def 7)
9.		$\vdash$	$\lambda z_{\blacksquare}[z \in M_0 \wedge \neg[z \in f_0(z)]] \in P(M_0)$	(PSet-Def 8)
10.	1,2	$\vdash$	$\exists y_{\blacksquare}[y \in M_0 \wedge \lambda z_{\blacksquare}[z \in M_0 \wedge \neg[z \in f_0(z)]] = f_0(y)]$	(Surj-Def 2 9)
11.	1,2,11	$\vdash$	$[y_0 \in M_0 \wedge \lambda z_{\blacksquare}[z \in M_0 \wedge \neg[z \in f_0(z)]] = f_0(y_0)]$	(Hyp)
12.	1,2,11	$\vdash$	$\lambda z_{\blacksquare}[z \in M_0 \wedge \neg[z \in f_0(z)]] = f_0(y_0)$	(AndE 11)
13.		$\vdash$	$y_0 \in f_0(y_0) = y_0 \in f_0(y_0)$	(=-Ref)
14.	1,2,11	$\vdash$	$y_0 \in \lambda z_{\blacksquare}[z \in M_0 \wedge \neg[z \in f_0(z)]] = y_0 \in f_0(y_0)$	(= 12 13)
15.	1,2,11	$\vdash$	$[y_0 \in \lambda z_{\blacksquare}[z \in M_0 \wedge \neg[z \in f_0(z)]] \leftrightarrow y_0 \in f_0(y_0)]$	(=-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	$\vdash$	$\bot$	(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	$\vdash$	$\bot$	(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	$\vdash$	$\bot$	(OrE 24 19 23)
			End of Case Analysis	
26.	1,2	$\vdash$	$\bot$	(ExistsE 10 25)
27.	1	$\vdash$	$\bot$	(ExistsE 1 26)
28.		$\vdash$	$\neg[\exists f_{\blacksquare}\mathrm{surj}(f, M_0, P(M_0))]$	(NotI 27)
29.		$\vdash$	$\forall M_{\blacksquare}\neg[\exists f_{\blacksquare}\mathrm{surj}(f, M, P(M))]$	(ForallI 28)

Figure 1: ND-Proof of the **Powerset** Example

to the problem solving competence of mathematicians becomes very important, 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 [Ireland, 1992]. 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 [Giunchiglia and Traverso, 1994] 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 procedural aspects like loops can be easily represented 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 in our earlier work, it is now not only possible to extract methods from proofs, but 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 adaption of methods to bridge syntactic gaps, for instance, arities of predicates [Huang *et al.*, 1994a]. In the following we examine the meta-methods `Cut-Submethod`, `Abstract`, and `set2func`.

3.1 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:

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

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

Figure 2: Method `Diag-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 dual 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 above is not directly applicable to this problem since the conclusions do not correspond to one another. Here we want to show $\neg\exists f_{\iota\to(\iota\to\iota)\blacksquare}\mathrm{surj}(f, \mathbb{N}, [0, 1])$, while method `Diag-1` proves $\forall M_{\blacksquare}\neg[\exists f_{\blacksquare}\mathrm{surj}(f, M, P(M))]$. 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`′: At first we have to apply the meta-method `Cut-Submethod` which recognizes that the new theorem is similar to line 15 in method `Diag-1`, an 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 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* occuring 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

- $f_{\iota\to(\iota\to o)} \mapsto f_{\iota\to(\iota\to\iota)}$
- $\mathrm{surj}_{(\iota\to(\iota\to o))\times(\iota\to o)\times((\iota\to o)\to o)\to o} \mapsto \mathrm{surj}_{(\iota\to(\iota\to\iota))\times(\iota\to o)\times((\iota\to\iota)\to o)\to o}$
- $M_{0\,\iota\to o} \mapsto \mathbb{N}_{\iota\to o}$
- $P(M_0)_{(\iota\to o)\to o} \mapsto [0, 1]_{(\iota\to\iota)\to o}$

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 adaption.

The procedure `term2formula` replaces a term t with $\mathrm{bool}(t)$. Semantically, the function symbol "bool" maps a ι-term to a truth value, 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 achieved by inserting $\mathrm{bool}(A) \wedge \neg\mathrm{bool}(B)$ as an extra axiom. For other elements "bool" is not specified.

Declarations	—
Premises	Surj-Def, bool, value
Constraint	—
Conclusions	15
Declarative Content	1. 1 $\vdash$ $\exists f_{\blacksquare}\mathrm{surj}(f, \mathbb{N}, [0,1])$ (Hyp) 2. 1,2 $\vdash$ $\mathrm{surj}(f_0, \mathbb{N}, [0,1])$ (Hyp) 3. 1,2 $\vdash$ $\lambda z_{\mathbb{N}\blacksquare}\mathrm{value}(\neg[\mathrm{bool}(f_0(z)(z))]) \in [0,1]$ (PLAN) 4. 1,2 $\vdash$ $\exists y_{\mathbb{N}\blacksquare}\lambda z_{\mathbb{N}\blacksquare}\mathrm{value}(\neg[\mathrm{bool}(f_0(z)(z))]) = f_0(y)$ (Surj-Def 2 3) 5. 1,2,5 $\vdash$ $\lambda z_{\mathbb{N}\blacksquare}\mathrm{value}(\neg[\mathrm{bool}(f_0(z)(z))]) = f_0(y_0)$ (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. $\vdash$ $\neg[\exists f_{\blacksquare}\mathrm{surj}(f, \mathbb{N}, [0,1])]$ (NotI 14)
Procedural Content	`schema-interpreter`

Figure 3: Method `Diag-2`

The procedure `formula2term` replaces a formula φ with value(φ). The function symbol "value" maps true to the element A and false to the element B. This can be specified by the formula

$$\forall x_{o\blacksquare}(x \leftrightarrow \mathrm{value}(x) = A) \wedge (\neg x \leftrightarrow \mathrm{value}(x) = B)$$

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 $\lambda z_{\blacksquare}[\mathbb{N}(z) \wedge \neg[\mathrm{bool}(f_0(z)(z))] \in [0,1]$ is transformed to $\lambda z_{\mathbb{N}\blacksquare}(\neg[\mathrm{bool}(f_0(z)(z))]) \in [0,1]$.

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 $\lambda x_{\blacksquare}\mathrm{value}(\neg(\mathrm{bool}(x)))$, 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 $f_0(y_0)(y_0)$ with the procedure `term2formula` in the lines 3, 4, 5, 6, 7, 9 and 11 we get:

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

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

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

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

Finally we get the method `Diag-2` in figure 3. 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 [Basin and Walsh, 1993]).

By the generation of the most special generalization of $\neg[\exists f_{\blacksquare}\mathrm{surj}(f, M_0, P(M_0))]$ and $\neg[\exists f_{\blacksquare}\mathrm{surj}(f, \mathbb{N}, [0,1])]$, we

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

Figure 4: Method `Diag-1-2`

get the generalized conclusion $\neg[\exists f_{\blacksquare}\text{surj}(f, \overline{X}, \overline{Y})]$. Note that meta-variables are overlined. In the same manner $f_0(y_0)(y_0)$ 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 $\text{value}_{\text{o}}\neg$ 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 4.

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

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

3.2 The Second Generalization

While the first two problems are closely related, we want to sketch out how our approach can be applied to a fairly distinct example, 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, $\text{defined}(t(x))$ 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 [Melis, 1994].

We formalize the problem in the following way:

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

In this formalization we use the following sorts: $\mathbb{N}$ denotes the set of natural numbers. U is the union of $\mathbb{N}$ and $\{u\}$. The symbol u represents the non-terminating function. 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 $\lambda z_{\mathbb{N}\blacksquare}\text{if}(f(e(z),z) = 0, x, y)$ is computable too.

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

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 $\vdash$ $\exists \overline{f}_{\blacksquare} \overline{F}$ (Hyp) 2. 1,2 $\vdash$ $\overline{F'}$ (Hyp) 3. 1,2 $\vdash$ $\lambda z_{\overline{X}\blacksquare} \overline{\text{non}}(\overline{Z}(\overline{App}(\overline{h}(z), z))) \in \overline{Y}$ (PLAN 2) 4. 1,2 $\vdash$ $\exists y_{\overline{X}\blacksquare} \overline{h}(y) = \lambda z_{\overline{X}\blacksquare} \overline{\text{non}}(\overline{Z}(\overline{App}(\overline{h}(z), z)))$ ($\overline{J}$ 2 3) 5. 1,2,5 $\vdash$ $\overline{h}(y_0) = \lambda z_{\overline{X}\blacksquare} \overline{\text{non}}(\overline{Z}(\overline{App}(\overline{h}(z), z)))$ (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. $\vdash$ $\overline{Z}(\overline{App}(\overline{h}(y_0), y_0)) \vee \neg \overline{Z}(\overline{App}(\overline{h}(y_0), y_0))$ (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. $\vdash$ $\neg[\exists \overline{f}_{\blacksquare} \overline{F}]$ (NotI 14)
Procedural Content	`schema-interpreter`

Figure 5: Method `Diag-1-2-3`

problem then in a method, this mode, however, requires more user-guidance. In order to perform this abstraction generalizations like the following two are applied.

- `split-symbol` replaces 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. 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 5.

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 $\lambda z_{\mathsf{IN}\blacksquare}\text{if}(\text{halt}(e(z), z) = 0, u, 0)$ (line 3). According to the Gödel enumeration theorem there is a natural number y_0 so that $e(y_0) = \lambda z_{\mathsf{IN}\blacksquare}\text{if}(\text{halt}(e(z), z) = 0, u, 0)$ (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 problems discussed in this paper. Nevertheless the 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 6.

It is worth comparing 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 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 $\lambda x_{\blacksquare} x$. Moreover, such information is not only crucial for a direct application. From this binding it is comparatively easier to come to the binding $\lambda x_{\blacksquare}\text{bool}(x)$ 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 6 along with generalized methods, so that it can be considered in the planning process.

4 Conclusion and Future Development

The problem solving competence of a mathematician relies heavily on his ability to adapt problem solving know-

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

Figure 6: Instantiations of Meta-variables in Diag-1-2-3

ledge 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 previous work we proposed a declarative extension to the proof planning approach developed by Bundy, in order to mechanize parts of this ability.

This paper is aimed to provide evidence that our goal can be achieved with our declarative approach. Besides demonstrating how a class of diagonalization methods can be represented in our framework, we present a detailed trace protocol of its 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.

Up to now, we have been concentrating on the meta-methods. 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. Another interesting question concerns how the bindings for meta-variables can be preserved during the abstraction process and be reused by the planner.

References

[Basin and Walsh, 1993] D. Basin and T. Walsh. Difference unification. In R. Bajcsy, editor, *Proceedings of the 13th IJCAI*, pages 116–122, 1993. Morgan Kaufmann.

[Bundy *et al.*, 1990] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The OYSTER-CLAM system. In M. E. Stickel, editor, *Proceedings of the 10th CADE*, pages 647–648, 1990. Springer Verlag, LNAI 449.

[Bundy *et al.*, 1993] A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, **62**:185–253, 1993.

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

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

[Giunchiglia and Traverso, 1994] F. Giunchiglia and P. Traverso. Program tactics and logic tactics. In F. Pfenning, editor, *Proceedings of LPAR*, pages 16–30, 1994. Springer Verlag, LNAI 822.

[Gordon *et al.*, 1979] M. Gordon, R. Milner, and C. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation.* LNCS 78. Springer Verlag, 1979.

[Huang and Kerber, 1991] X. Huang and M. Kerber. Theorem proving as an interleaving process of planning and verification. unpublished manuscript, 1991.

[Huang *et al.*, 1994a] X. Huang, M. Kerber, M. Kohlhase, and J. Richts. Adapting methods to novel tasks in proof planning. In B. Nebel and L. Dreschler-Fischer, editors, *KI-94: Advances in Artificial Intelligence*, pages 379–390, 1994. Springer Verlag, LNAI 861.

[Huang *et al.*, 1994b] X. Huang, M. Kerber, M. Kohlhase, E. Melis, D. Nesmith, J. Richts, and J. Siekmann. Ω-MKRP: A proof development environment. In A. Bundy, editor, *Proceedings of the 12th CADE*, pages 788–792, 1994. Springer Verlag, LNAI 814.

[Huang, 1994] X. Huang. Reconstructing proofs at the assertion level. In A. Bundy, editor, *Proceedings of the 12th CADE*, pages 738–752, 1994. Springer Verlag, LNAI 814.

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

[Melis, 1994] E. Melis. Representing and reformulating diagonalization methods. Technical Report CMU-CS-94-174, Carnegie Mellon University, Pittsburgh, USA, 1994.

[Pólya, 1945] G. Pólya. *How to Solve It.* Princeton University Press, 1945.

[VanLehn, 1989] K. VanLehn. Problem solving and cognitive skill acquisition. In M. I. Posner, editor, *Foundations of Cognitive Science*, chapter 14. MIT Press, 1989.