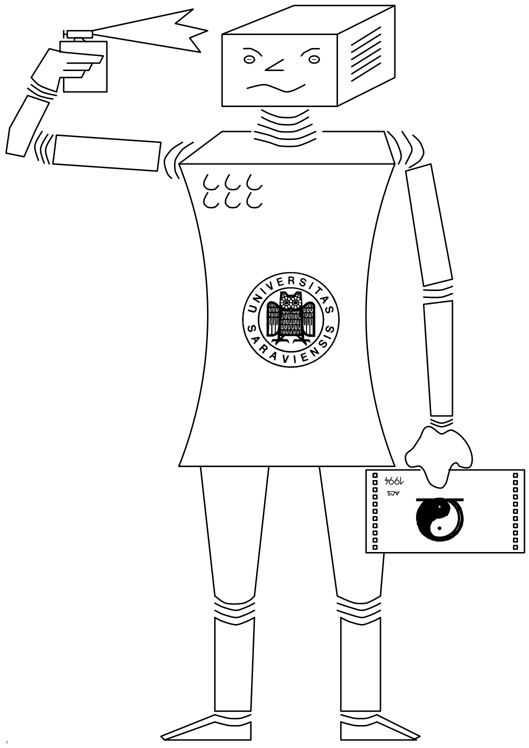
UNIVERSITÄT DES SAARLANDES
FACHBEREICH INFORMATIK
D-66041 SAARBRÜCKEN
GERMANY
WWW: http://js-sfbsun.cs.uni-sb.de/pub/www/



Planning Mathematical Proofs with Methods

Xiaorong Huang   Manfred Kerber
Jörn Richts   Arthur Sehn

Published as: In Journal of Information Processing and Cybernetics, EIK (formerly: Elektronische Informationsverarbeitung und Kybernetik), **30**:277–291, 1994.

J. Inform. Process. Cybernet. **EIK 30** (1994) 5/6, 277–291
(formerly: Elektron. Inform.verarb. Kybernet.)

Planning Mathematical Proofs with Methods[1]

By Xiaorong Huang, Manfred Kerber, Jörn Richts, and Arthur Sehn

Abstract: In this article we formally describe a declarative approach for encoding plan operators in proof planning, the so-called methods. The notion of method evolves from the much studied concept tactic and was first used by Bundy. While significant deductive power has been achieved with the planning approach towards automated deduction, the procedural character of the tactic part of methods, however, hinders mechanical modification. Although the strength of a proof planning system largely depends on powerful general procedures which solve a large class of problems, mechanical or even automated modification of methods is nevertheless necessary for at least two reasons. Firstly methods designed for a specific type of problem will never be general enough. For instance, it is very difficult to encode a general method which solves all problems a human mathematician might intuitively consider as a case of homomorphy. Secondly the cognitive ability of adapting existing methods to suit novel situations is a fundamental part of human mathematical competence. We believe it is extremely valuable to account computationally for this kind of reasoning.

The main part of this article is devoted to a declarative language for encoding methods, composed of a tactic and a specification. The major feature of our approach is that the tactic part of a method is split into a declarative and a procedural part in order to enable a tractable adaption of methods. The applicability of a method in a planning situation is formulated in the specification, essentially consisting of an object level formula schema and a meta-level formula of a declarative constraint language. After setting up our general framework, we mainly concentrate on this constraint language. Furthermore we illustrate how our methods can be used in a STRIPS-like planning framework. Finally we briefly illustrate the mechanical modification of declaratively encoded methods by so-called meta-methods.

1. Introduction

Mathematicians learn during their academic training not only facts like definitions or theorems, but also problem-solving *know-how* for proving mathematical theorems. An important part of this know-how can be described in terms of reasoning methods like the diagonalization procedure, the application of a definition, or the application of the homomorphy property. The main aim of this article is to formalize the concept of a *method* in order to reflect more closely the informal notion of a mathematical method.

[1]Revised version of a lecture given at the workshop "Tactical Theorem Proving" held at the 18th German Annual Conference on Artificial Intelligence, KI'94, Saarbrücken, September 21-22, 1994.

In particular, we try to capture some aspects of plausible reasoning by a proof planning process built on top of the methods as plan operators. The importance of such kind of reasoning in proof search has been pointed out by Pólya [16, p. vi].

Such a plan-based approach was first proposed by Bundy et al. in the OYSTER-CLAM-system [5], where the tactics are extended to methods. Their *methods* can be viewed as a unit consisting of a tactic and a specification. While the specification is declarative, the tactic itself is still procedural. This has, however, a severe drawback, namely, the mechanical adaption of methods to new situations is almost impossible, because that would require the transformation of programs – tactics are just programs – which is known to be a very hard problem in practice. However, the strength of human reasoning and problem solving depends to a great extent on the ability to adapt existing problem solving facilities to related, but not directly fitting situations. In order to allow such a mechanical modification of methods, we have proposed in [12] a separation of the tactic into a declaratively and a procedurally represented part. As shown there this separation not only leads to more natural methods, but practically enables the formulation of general meta-level mechanisms which adapt existing methods to suit novel situations. In this way, an automated modification needs only to be performed on the declarative part. While it is desirable to store most of the relevant information in this part, only the rest should be encoded in the procedural part.

One potential criticism is that we should instead construct more general methods which cover large classes of problems. Although general methods are definitely needed for effective proof planning systems, this by no means excludes the need of modification. It is very difficult, for instance, to come up with a single method covering all possible cases which a human mathematician would *intuitively* consider as an example of homomorphy. A well-known example is the rippling method developed in Bundy's group, which has been extended from *rippling-out* [3] to a method covering a wide range of related problems [4].

The intention of our work can be compared to Ireland's approach of proof critics [13]. Although they considerably enhance the flexibility, no new methods are created. Difficulties arise when new problems exceeding the power of existing methods are encountered. The work of Giunchiglia and Traverso [9], where tactics are represented in a logical meta-language, has a similar motivation as our work, namely to represent tactics in a declarative way. In their approach the whole tactic is represented on a logical meta-level, what enables a full declarative representation. In our approach only parts of the tactic are represented declaratively, what should result in easier transformations.

2. Logical Foundations: Calculus and Tactics

Since the technical mathematical language of a typical textbook is essentially a higher-order logic, in our work we adopt the *classical* higher-order logic based on Church's simple type theory as introduced in [6, 2].

The proof format in our approach is based on the natural deduction (ND) calculus first proposed by Gentzen in [8]. Concretely, we adopt a linearized version of ND proofs as introduced in [1]. In this formalism, an ND proof is a sequence of *proof lines*, each of them is of the form:

$$\mathit{Label} \qquad \triangle \quad \vdash \quad \mathit{Derived\text{-}Formula} \qquad\qquad (\mathit{Rule} \quad \mathit{premise\text{-}lines})$$

where *Rule* is restricted to a rule of inference in ND, which justifies the derivation of the *Derived-Formula* using the formulae in *premise-lines*. *Rule* and *premise-lines* together are called the justification of a line. $\triangle$ is a finite set of formulae, being hypotheses the derived formula depends on. Since a natural deduction proof can also be viewed as a proof tree, we will talk about proof trees as well.

Below are two typical rules in the ND-calculus:

$$\frac{\triangle \vdash F \vee G, \quad \triangle, F \vdash H, \quad \triangle, G \vdash H}{\triangle \vdash H} CASE, \qquad \frac{\triangle \vdash F, \quad \triangle \vdash F \rightarrow G}{\triangle \vdash G} \rightarrow E,$$

In order to uniformly represent proof plans, we extend the ND proof formalism by allowing the *Rule* slot to be replaced by the name of a method or simply the value "OPEN". Open lines are still to be justified in the planning process.

In our framework a *tactic* is a function that generates new proof lines and inserts them into the current proof. Following the declarative approach proposed in [12], this function is represented in two parts. One part is a set of proof line schemata, that is, proof lines with meta-variables. The other part contains a procedure. The whole tactic can then be seen as a function with parameters. An application of this function with concrete instances for the meta-variables generates new proof lines by applying the procedure to the proof line schemata.

Most commonly the procedure is just one standard interpreter, which basically instantiates proof line schemata by binding meta-variables. In other cases, the procedure can be a sophisticated theorem prover. Hence the range of possible tactics is very wide, reaching from the application of an ND rule to the call of an incorporated theorem prover. Since we allow arbitrary proof lines to be added to the current proof state, the correctness of the final proof is not ensured a priori, it must be checked by a verifier [11].

3. A Declarative Approach toward Methods

A central concept of knowledge-based reasoning in mathematics 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. Therefore methods can be quite general, such as finding proofs by a case analysis or complete induction, or the advice to expand definitions. On the other hand, domain specific methods are also very common, for instance a clearly described proof sketch for proving a theorem by diagonalization.

In our framework a method can be divided into a declarative and a procedural part. By discerning the declarative part of a method, it is now possible to formulate meta-methods adapting the declarative part of existing methods and thus come up with novel ones.

Concretely, we define a method as consisting of the following slots:

Declarations Formally the declarations are a set $D = \{x{:}\mathsf{s} \mid x \in \mathbf{V}^{\mathcal{M}}, \mathsf{s} \in \mathbf{SORT}^{\mathcal{M}}\}$, where $\mathbf{V}^{\mathcal{M}}$ is the set of meta-variables and $\mathbf{SORT}^{\mathcal{M}}$ is the set of sort symbols.

Tactic In our model a *tactic* is split into two parts: An ND proof schema with meta-variables, called the *declarative content*, and a piece of program, called the *procedural content*. This procedural part can be a standard interpreter which creates new proof lines by instantiating the declarative content and then inserting them into the current proof state. It can also be an arbitrary piece of procedural knowledge, for instance, an automated theorem prover.

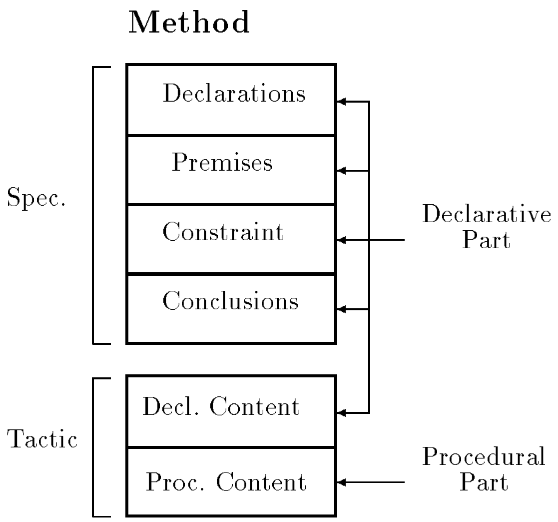
Formally a tactic is a pair $\langle T^{decl}, T^{proc}\rangle$, where T^{decl} is a finite list of schematic proof lines of the form $(\alpha_i) \quad H_i \vdash F_i \quad J_i$, or a meta-variable standing for such a list. Here the α_i are the labels of proof lines. The H_i are either a list of proof line labels standing for the formulae of these lines or a meta-variable representing such a list. The F_i are formula schemata, that is, formulae containing meta-variables. The J_i are justifications or the corresponding meta-variables. T^{proc} is a program, which generates new proof lines by interpreting T^{decl}.

Premises and Conclusions Intuitively, the *premises* slot contains a list of proof line schemata which are used to prove the lines in the *conclusions* slot. In most cases both slots are subsets of the proof lines in the declarative content of the tactic. The proof lines can be marked with an additional sign "$\oplus$" or "$\ominus$". These signs only play a role in the planning process.

Constraint The more complex applicability condition of a method is formulated in the *constraint*. After describing the semantics of a method, we introduce the constraint language in detail.

The declarations, the premises, the constraint, and the conclusions form together the so-called specification of the method.

Declarations: A signature that declares meta-variables used,
Premises: Schemata of proof lines which are used by this method as assumptions,
Conclusions: Schemata of proof lines which this method is designed to prove,
Constraint: A formula in a constraint language to be described later. This is used to formulate further restrictions on the premises and the conclusions, which cannot be formulated in terms of proof line schemata,
Tactic: It is split into two components: the declarative content and the procedural content.



3. 1. Semantics of a Method

A method consists of two major parts: the specification and the tactic. Since the specification is used in the planning phase and the tactic in the execution phase of the plan operator, a method's semantics can also be divided into two parts: its semantics as a plan operator and as a tactic. The semantics as a plan operator is defined in the section 4., where STRIPS-like plan operators are constructed from the specification.

The semantics of a tactic is defined in terms of its execution: with the meta-variable bindings resulting from the planning process the procedural content can be applied to the instantiated declarative content constructing new proof lines.

3. 2. The Constraint Language $\mathcal{CL}$

As described previously, in the first place the applicability of a method is specified in terms of object level schemata for the premises and the conclusions. Furthermore, the constraint slot contains additional meta-level application conditions.

First we motivate our definition of the constraint language by discussing some criteria for an effective specification language in a proof planning environment. *Expressiveness:* Since we want to check the applicability of methods, in our language it must be possible to express all relevant properties of the objects in the current proof state. *Adaptability:* The specification language should support automatic modification of methods (by so-called meta-methods). *Tractability:* Since the specifications play the role of plan operators in a proof planning environment and the applicability of an operator must be effectively computable, the specification language must be decidable. *Structured Representation:* The specification language should not only allow the formulation of decidable application conditions of methods, but it should also be efficiently computable. Therefore the conditions should be structured in order to check the most important conditions first.

We also have another concepts deviating from the standard definition of logics, namely we need a binding mechanism. As we have mentioned, the free meta-variables of the method are bound by the planner via matching. But it is also possible to assign a value to remaining unbound ones. This can be necessary, for instance, when a new formula should be constructed by evaluating the constraint. Therefore we include a binding mechanism "$\leftarrow$", which is interpreted as a combination of a predicate and an assignment known from procedural languages. We describe it in definition 3.2.

The Syntax of $\mathcal{CL}$ Based on the discussion above, the syntax of our constraint language is a sorted first-order language with fixed function and predicate declarations. We use the general notions of order-sorted logics with a set of sort symbols $\mathbf{SORT}^{\mathcal{M}}$, subsort declarations $\mathbf{SD}^{\mathcal{M}}$, function declarations $\mathbf{FD}^{\mathcal{M}}$, predicate declarations $\mathbf{PD}^{\mathcal{M}}$, and terms $\mathbf{T}^{\mathcal{M}}_{\mathsf{s}}$ of the sort s as introduced in [17]. Our definition of a formula differs from the standard one, however, since we allow only quantification over finite lists of terms and because of the special binding predicate "$\leftarrow$".

Definition 3.1 The set of *well-formed formulae*, $\mathbf{WFF}^{\mathcal{M}}$, for a given signature $\Sigma^{\mathcal{M}}$ is inductively defined. The logic connectives and quantifiers are listed in groups (iii)–(v) in decreasing order of the binding priority.

(i) If $P \in \mathbf{P}^{\mathcal{M}}$, $P{<}\mathsf{s_1} \times \cdots \times \mathsf{s_n}$ is a sort declaration for the predicate symbol $P \in \Sigma^{\mathcal{M}}$, and $t_i \in \mathbf{T}^{\mathcal{M}}_{\mathsf{s_i}}$ for $1 \leq i \leq n$, then $P(t_1, \ldots, t_n) \in \mathbf{WFF}^{\mathcal{M}}$. Also $\top, \bot \in \mathbf{WFF}^{\mathcal{M}}$, $\top$ stands for truth and $\bot$ for falsehood.

(ii) If $x \in \mathbf{V}^{\mathcal{M}}$ with the sort s and $t \in \mathbf{T}^{\mathcal{M}}_{\mathsf{s}}$, then $(x \leftarrow t) \in \mathbf{WFF}^{\mathcal{M}}$ (binding predicate).

(iii) If $\Phi \in \mathbf{WFF}^{\mathcal{M}}$, then $(\sim \Phi) \in \mathbf{WFF}^{\mathcal{M}}$ (negation).
(iv) If $\Phi_1, \Phi_2 \in \mathbf{WFF}^{\mathcal{M}}$, then $(\Phi_1 \circ \Phi_2) \in \mathbf{WFF}^{\mathcal{M}}$, for $\circ \in \{\&, |\}$ (conjunction and disjunction).
(v) If $t \in \mathbf{T}^{\mathcal{M}}_{\mathsf{list(s)}}$, $x{:}\mathsf{s} \in \mathbf{V}^{\mathcal{M}}$ and $\Phi \in \mathbf{WFF}^{\mathcal{M}}$, then $(\bigwedge^t x{:}\mathsf{s}.\Phi) \in \mathbf{WFF}^{\mathcal{M}}$ and $(\bigvee^t x{:}\mathsf{s}.\Phi) \in \mathbf{WFF}^{\mathcal{M}}$ (finite universal and existential quantification).

The Semantics of $\mathcal{CL}$ The semantics of $\mathcal{CL}$ describes the applicability criterion of a method. For simplicity we only emphasize the part of the semantics deviating from the standard sorted logic. In our approach, we define the value of a formula Ψ under an interpretation function $\mathcal{I}$ and an assignment φ, denoted by $\Psi^{\mathcal{I},\varphi}$, as a truth-value assignment pair in $\{\mathsf{t}, \mathsf{f}\} \times \mathcal{F}_{\mathrm{p}}(\mathbf{V}^{\mathcal{M}}, \mathcal{D}_{\mathsf{s}})$, where $\mathcal{F}_{\mathrm{p}}(\mathbf{V}^{\mathcal{M}}, \mathcal{D}_{\mathsf{s}})$ is the set of all partial functions from $\mathbf{V}^{\mathcal{M}}$ to $\mathcal{D}_{\mathsf{s}}$. $\mathcal{D}_{\mathsf{s}}$ is the universe for the sort s (cp. [17]). The second component of such pairs is needed to keep track of the bindings.

The following definition specifies the semantics of the constraint language. Starting from an assignment φ given by the planner, all bindings are accumulated in the second component of the truth-value assignment pairs. A method is *applicable* if the interpretation of the constraint results in t in the first component and no variable specified in the declaration slot remains unbound, otherwise it is not applicable. In the following we denote the domain of the partial function φ by $\mathrm{Dom}(\varphi)$. $\mathrm{Var}(t)$ denotes the set of all variables of a term t.

Definition 3.2 Let φ be an assignment, which is a partial function mapping variables of sort s to elements of $\mathcal{D}_{\mathsf{s}}$. We define the *value* of a formula Ψ under the assignment φ and the interpretation function $\mathcal{I}$, denoted by $\Psi^{\mathcal{I},\varphi}$, recursively.

The value of the terms are interpreted as usual in the standard Tarskian model-theoretic semantics. We only explain the semantics of the binding predicate, the lazy evaluated connectives, and the universal quantifier:

(1) When Ψ has the form $(x \leftarrow t)$,

$$\Psi^{\mathcal{I},\varphi} = \begin{cases} [(x \doteq t)]^{\mathcal{I},\varphi} & \text{if } x \in \mathrm{Dom}(\varphi), \\ \left\langle \mathsf{t}, \varphi \cup \{x \mapsto t^{\mathcal{I},\varphi}\} \right\rangle & \text{if } x \notin \mathrm{Var}(t), x \notin \mathrm{Dom}(\varphi) \\ & \quad \text{and } \mathrm{Var}(t) \subseteq \mathrm{Dom}(\varphi) \\ \langle \mathsf{f}, \varphi \rangle & \text{otherwise.} \end{cases}$$

In the first case the assignment is interpreted as the equality predicate since the variable is already bound. A binding is given to a variable in the second case, which is the main goal of this construct. If the variable x recursively occurs in the term t, then the assignment cannot be properly performed.

The value of the logical connectives & and | are determined by lazy evaluation:

(2) When Ψ has the form $\Phi_1 \& \Phi_2$,

$$\Psi^{\mathcal{I},\varphi} = \begin{cases} \Phi_2^{\mathcal{I},\varphi'}, \text{ if } \Phi_1^{\mathcal{I},\varphi} = \langle \mathsf{t}, \varphi' \rangle, \\ \langle \mathsf{f}, \varphi' \rangle, \text{ if } \Phi_1^{\mathcal{I},\varphi} = \langle \mathsf{f}, \varphi' \rangle. \end{cases}$$

The interpretation of the disjunction is similar to that of a conjunction, but when the second part of a disjunction is interpreted, the bindings are restored to the bindings used before the first part was interpreted.
The quantifiers range only over terms representing finite lists.

(3) When Ψ has the form $\bigwedge^t x{:}\mathsf{s}.\Phi$,

$$\Psi^{\mathcal{I},\varphi} = \begin{cases} \langle \mathsf{t}, \varphi \rangle, \text{ if } & t^{\mathcal{I},\varphi} = \langle d_1, \ldots, d_n \rangle, \mathrm{Var}(t) \subseteq \mathrm{Dom}(\varphi), \\ & x \notin \mathrm{Var}(t), \{x \mapsto d_i\} \not\subseteq \varphi, 1 \leq i \leq n, \\ & \text{and for all } i \text{ there is a } \varphi' \text{ such that} \\ & \Phi^{\mathcal{I},\varphi \cup \{x \mapsto d_i\}} = \langle \mathsf{t}, \varphi' \rangle. \\ \langle \mathsf{f}, \varphi \rangle \text{ otherwise.} & \end{cases}$$

The fixed interpretation at the meta-level Our meta-language is intended to specify applicability criteria for methods. First we assume therefore a fixed model restricted to the domain of object level logical entities, namely terms, types, proof lines, proof line justifications, inference rules, subterm positions, and substitutions. Second the different categories of objects at the object level are reflected by different sorts at the meta-level to avoid unnecessary instantiations through sort restrictions, so we consider a fixed set of sorts, namely term, abstr, appl, const, var for terms and its subclasses, type for types, prln for proof lines, just for justifications of proof lines, ir for inference rules, pos for positions, and sub for substitutions. Moreover we consider finite lists of these objects, for instance, list(term). Third the functions and the predicates at the meta-level have a fixed interpretation: they are standard primitives for manipulating the objects at the object level. For instance, termtype(F) is a function that calculates the type of a term F and atom(F) tests whether a term F is an atomic formula.

3. 3. Decidability of the Constraint Language

A main criterion for the method language is its tractability, which strongly depends on the decidability of the constraint language.

Theorem. *If all variables of the constraints are bound by matching the premises and conclusions against the proof lines occuring in the actual proof state, then the interpretation of the constraint terminates with* $\langle \mathsf{t}, \varphi \rangle$ *or* $\langle \mathsf{f}, \varphi \rangle$ *as value, where* φ *is a partial assignment function.*

Proof. The decidability of the constraint language can be derived from two facts, first from the fixed interpretation of function and predicate symbols, and second from the finite range of the quantifiers, which can be considered as finite conjunctions or disjunctions. □

3. 4. Homomorphy Example

We want to illustrate our method language by an example of a method for proving theorems in the field of algebra. Its proof strategy can be informally described as: *If f is a given function, P a defined predicate and the goal is to prove $P(f(c))$, then show*

$P(c)$ and use this to show $P(f(c))$. The idea is that f is a homomorphism for the property P and that f can be "rippled away" (compare [4]).

Suppose we want to prove the theorem that the converse relation of a binary relation ρ is symmetric (formally: $symmetric(converse(\rho))$). The method `hom1-1` can be applied by substituting *converse*, *symmetric*, and ρ for the meta-variables F, P, and C, respectively[2]. The method `hom1-1` proposes $symmetric(\rho)$ as a new line which can be used to prove $symmetric(converse(\rho))$ together with the definitions of *symmetric* and *converse*. The details of using this method in proof planning are discussed in the next section.

Method : `hom1-1`				
Declarations	$L_1, L_2, L_3, L_4, L_5, L_6$:prln X, Y:var $\Phi, \Psi, \Psi_1, \Psi_2$:term			J_1, J_2, J_3:just P, F, C:const
Premises	$L_1, L_2, \oplus L_3$			
Constraint	$\sim$ termoccs$(F,\Phi)\doteq\langle\rangle$ & termtype$(C)\doteq$typerange(termtype(F)) & $\Psi_1 \leftarrow$ termrploccs(X,Ψ,C) & $\Psi_2 \leftarrow$ termrploccs$(X,\Psi,F(C))$			
Conclusions	$\ominus L_6$			
Declarative Content	(L_1)	$\langle L_1\rangle$	$\vdash \forall Y. \Phi$	(J_1)
	(L_2)	$\langle L_1, L_2\rangle$	$\vdash \forall X. P(X) \leftrightarrow \Psi$	(J_2)
	(L_3)	$\langle L_1, L_2\rangle$	$\vdash P(C)$	(J_3)
	(L_4)	$\langle L_1, L_2\rangle$	$\vdash \Psi_1$	(`def-e` L_2, L_3)
	(L_5)	$\langle L_1, L_2\rangle$	$\vdash \Psi_2$	(OPENL_1, L_4)
	(L_6)	$\langle L_1, L_2\rangle$	$\vdash P(F(C))$	(`def-i` L_2, L_5)
Procedural Content	`schema − interpreter`			

The constraint of the method states the following: The first line says that F must occur in Φ, the second line means that certain types must be equal (otherwise the newly created formula $P(C)$ is not well-typed). The third line means that Ψ_1 is created by replacing all free occurences of X in Ψ by C, and the fourth line that Ψ_2 is obtained by replacing all free occurences of X in Ψ by $F(C)$. Actually Ψ_1 and Ψ_2 are just the expanded versions of $P(C)$ and $P(F(C))$, respectively. Note that L_1 and L_2 contain essential properties of F and P like their definitions.

4. Planning

In this section we specify a simple proof planning mechanism and the semantics of our methods from the planning perspective. After giving a motivation and showing an example, we formally define STRIPS plan operators from our methods and then show their use in a planning algorithm.

[2]In the following the capital letters denote meta-variables, while the object level elements are written in lowercase letters.

4. 1. Motivation

As a first attempt we adopt a STRIPS-like planning paradigm, where the plan operators correspond to the methods. 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* (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, they are marked by the label "(?)") and *support lines* (that can be used as premises to bridge it, they are marked by the label "(!)"). 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 are no more open lines in the planning state.

In order to demonstrate how the methods are used in the planning process, we show what the corresponding STRIPS plan operators would look like. We assume the STRIPS mechanism [7] as already known and only mention that a STRIPS plan operator consists of three slots: the *preconditions* slot, the *delete* slot, and the *add* slot. A STRIPS plan operator is applicable in a planning state if the propositions in the preconditions slot are present in the planning state. When a method is applied, the propositions in the delete slot are removed from the planning state and the propositions of the add slot are inserted.

4. 2. Defining STRIPS Plan Operators from Methods

Now we formally specify how the specification of a method corresponds to a STRIPS plan operator. The required slots of the specification are the premises, the constraint, and the conclusions. The premises and conclusions contain lines that are labeled with "$\oplus$" or "$\ominus$" or are unlabeled; the constraint contains an additional logical statement that can be evaluated to true or false. With this information we define a STRIPS plan operator that has three slots: the precondition list, the delete list, and the add list.

Method *name*	
Premises	$p_1, \ldots, p_{n_p}$
	$\ominus p_1^-, \ldots, \ominus p_{n_p^-}^-$
	$\oplus p_1^+, \ldots, \oplus p_{n_p^+}^+$
Constraint	Ψ
Conclusions	$c_1, \ldots, c_{n_c}$
	$\ominus c_1^-, \ldots, \ominus c_{n_c^-}^-$
	$\oplus c_1^+, \ldots, \oplus c_{n_c^+}^+$

$\Longrightarrow$

STRIPS-**Op** *name*	
Pre:	(!) $p_1, \ldots,$ (!) p_{n_p}
	(!) $p_1^-, \ldots,$ (!) $p_{n_p^-}^-$
	(?) $c_1, \ldots,$ (?) c_{n_c}
	(?) $c_1^-, \ldots,$ (?) $c_{n_c^-}^-$
	Ψ
Del:	(!) $p_1^-, \ldots,$ (!) $p_{n_p^-}^-$
	(?) $c_1^-, \ldots,$ (?) $c_{n_c^-}^-$
Add:	(?) $p_1^+, \ldots,$ (?) $p_{n_p^+}^+$
	(!) $c_1^+, \ldots,$ (!) $c_{n_c^+}^+$

The figure above shows the generation of the STRIPS plan operator from a method. It represents the most general case, although usually not all possible labelings are used in a single method. The unlabeled lines in a method go to the preconditions slot and the lines with a "$\oplus$" to the add slot. The lines labeled with "$\ominus$" are moved to the precondition slot and to the delete slot. In the preconditions and the delete slot the premises become support lines and the conclusions become open lines. In the add slot it is the other way round. The content of the constraint slot Ψ is inserted into the preconditions slot of a STRIPS operator. Note that Ψ is not a proof line that is present in the planning state, but a formula specifying an additional applicability condition.

The planning algorithm can be abstractly described as follows:

While there are still open lines in the planning state

1. Find all possibilities of an application of the methods
   (a) Select a method M.
   (b) Find all possibilities of matching the lines in M's precondition slot with lines in the planning state.
   (c) Evaluate the *constraint* of M with the bindings established in 1(b).
2. Select the "best" method $\widetilde{M}$ (this is the choice point for the backtracking mechanism and the point where some heuristic control can take place)
3. Apply the plan operator $\widetilde{M}$ to the planning state
   (a) Insert the lines in the add slot of $\widetilde{M}$ into the planning state.
   (b) Remove the lines in the delete slot of $\widetilde{M}$ from the planning state.

Note that backward planning is not possible since the terminal state is defined by the absence of open lines. During the matching of the lines in the preconditions slot and the evaluation of the constraint all meta-variables should have been bound to object level terms. Therefore the new lines of step 3.(a) can be constructed by simply instantiating the meta-variables.

Once a complete proof plan is found, all methods (i.e. their tactics) in the proof plan are successively executed in order to construct a calculus level proof. The verification phase, which follows the application of the methods, may result in a recursive call to the planner or in backtracking. While a recursive call refines a plan and models hierarchical planning, the backtracking rejects the plan and calls the proof planner in order to find an alternative one.

4. 3. Homomorphy Example

Having illustrated the basic framework, let us examine an example, related to the `hom1-1` method, shown in section 3. 4. Note that line L_5 is an open line that does not occur in the specification and therefore does not enter the planning state. This leads to an abstraction in the planning process (i.e. there is less information in the planning state) and results in a hierarchical proof planning: since line L_5 is not considered by the planner, after completing the plan it will be inserted into the proof tree as an open line by executing the tactic of `hom1-1`. This will result in a recursive call of the planner after the following verification phase.

Planning State 1

(!)	1.	1;	$\vdash$	$\forall\sigma\bullet\forall x,y\bullet\langle x,y\rangle \in \mathit{converse}(\sigma) \leftrightarrow \langle y,x\rangle \in \sigma$	(!)
(!)	2.	2;	$\vdash$	$\forall\sigma\bullet \mathit{symmetric}(\sigma) \leftrightarrow \forall x,y\bullet\langle x,y\rangle \in \sigma \rightarrow \langle y,x\rangle \in \sigma$	(!)
(?)	9.	2,1;	$\vdash$	$\mathit{symmetric}(\mathit{converse}(\rho))$	(?)

`hom1-1` 1 2 9

Planning State 2

(!)	1.	1;	$\vdash$	$\forall\sigma\bullet\forall x,y\bullet\langle x,y\rangle \in \mathit{converse}(\sigma) \leftrightarrow \langle y,x\rangle \in \sigma$	(!)
(!)	2.	2;	$\vdash$	$\forall\sigma\bullet \mathit{symmetric}(\sigma) \leftrightarrow \forall x,y\bullet\langle x,y\rangle \in \sigma \rightarrow \langle y,x\rangle \in \sigma$	(!)
(?)	8.	2,1;	$\vdash$	$\mathit{symmetric}(\rho)$	(?)

The figure above shows the transition of the planning state. When the plan is completed (by proving the remaining open line with some additional information about ρ) the proof resulting from the application of the tactic `hom1-1` looks as follows:

1.	1;	$\vdash$	$\forall\sigma\bullet\forall x,y\bullet\langle x,y\rangle \in \mathit{converse}(\sigma) \leftrightarrow \langle y,x\rangle \in \sigma$	(J1)
2.	2;	$\vdash$	$\forall\sigma\bullet \mathit{symmetric}(\sigma) \leftrightarrow \forall x,y\bullet\langle x,y\rangle \in \sigma \rightarrow \langle y,x\rangle \in \sigma$	(J2)
3.	1,2;	$\vdash$	$\mathit{symmetric}(\rho)$	(J3)
4.	1,2;	$\vdash$	$\forall x,y\bullet\langle x,y\rangle \in \rho \rightarrow \langle y,x\rangle \in \rho$	(`def-e` 3 2)
5.	1,2;	$\vdash$	$\forall x,y\bullet\langle x,y\rangle \in \mathit{converse}(\rho) \rightarrow \langle y,x\rangle \in \mathit{converse}(\rho)$	(OPEN 1 4)
6.	1,2;	$\vdash$	$\mathit{symmetric}(\mathit{converse}(\rho))$	(`def-i` 2 5)

Let us have a look at the justifications in this proof fragment. Justifications J1 and J2 are found via matching when applying the plan operator of `hom1-1`, J3 will be instantiated by the further proof planning process. The justifications of lines 4 and 6 stand for the subproofs generated by the application of the tactics of these methods, whereas the justification of line 5 defines a new gap with support lines 1 and 4.

5. Extending the Reasoning Repertoire by Meta-Methods

It is one of the main features contributing to the problem solving competence of mathematicians that they can extend their current problem solving methods by adapting them to suit new situations (see [15, 14] for mathematical reasoning and [18] for general problem solving).

By discerning the declarative part of tactics, it is now feasible in our approach to formulate meta-methods adapting the declarative part of existing methods and thus come up with novel ones. In a framework where tactics consist only of procedural knowledge, we would in effect be confronted with the much more difficult problem of adapting procedures in order to achieve the above. A more detailed discussion can be found in [12].

We define a *meta-method* as consisting of a *body* and a *rating*. The body is a procedure which takes as input a method, and possibly further parameters from the planner (in particular the current state of proof planning) and generates a new method with the same procedural part. The rating is a procedure which takes as input a method, the current state of proof planning and the proof history. It estimates the contribution of the application of the meta-methods to the solution of the current problem.

We illustrate this definition with the method `hom1-1` introduced in section 3. 4. The method `hom1-1` simplifies a problem by generating an intermediate goal, where a *unary* function symbol is eliminated. Suppose that we are facing a new but similar problem of proving that the intersection of symmetric relations is itself a symmetric relation. What we need is a variant of `hom1-1`, which is able to handle a *binary* function symbol (*intersection* in this case) in a similar way.

In the following, we illustrate how to use a meta-method called `add-argument` to obtain a binary version `hom1-2` from the unary version `hom1-1`. `hom1-1` is suited to situations with a unary predicate constant P and a unary function constant F, while `hom1-2` can handle situations with a unary predicate constant P and a binary function constant F'. The meta-method `add-argument` takes as input a method `M` and a unary function or predicate constant F.

The meta-method `add-argument` is supposed to add an argument to a key constant symbol F which is a unary predicate or function used in a method, the modified function or predicate constant is called F'. Essentially `add-argument` creates a method `M'` by carrying out the following modification on the declarative part of `M`: Replace all occurences of terms $F(x)$ and $F(C)$ by $F'(x,y)$ and $F'(C,D)$, respectively and augment the corresponding quantifications. (D has to be a new meta-variable standing for a constant). If C occurs in a proof line, but not in a term $F(C)$, a copy of this line will be inserted into the proof schema, replacing C by D (in the example `hom1-2`, line 4 is copied from 3). Such a copy must be accompanied by a corresponding augmentation to the specification of the method. The procedural part of `M` is taken over for the new method. The method `hom1-2` below can be obtained by applying `add-argument` with the arguments `hom1-1` and F.

Method : `hom1-2`			
Declarations	$L_1, L_2, L_3, L_4, L_5, L_6, L_7, L_8$:prln X, Y:var $\Phi, \Psi, \Psi_1, \Psi_2, \Psi'_1, \Psi'_2$:term		J_1, J_2, J_3, J_4:just P, G, C, D:const
Premises	$L_1, L_2, \oplus L_3, \oplus L_4$		
Constraint	$\sim$ termoccs$(F', \Phi) \doteq \langle\rangle$ & termtype$(C) \doteq$ typerange(termtype(F')) & termtype$(D) \doteq$ typerange(termtype(F')) & $\Psi_1 \leftarrow$ termrploccs(X, Ψ, C) & $\Psi'_1 \leftarrow$ termrploccs(X, Ψ, D) & $\Psi_2 \leftarrow$ termrploccs$(X, \Psi, F'(C,D))$		
Conclusions	$\ominus L_6$		
Declarative Content	(L_1)	$\langle L_1 \rangle \quad \vdash \forall Y_\blacksquare \Phi$	(J_1)
	(L_2)	$\langle L_1, L_2 \rangle \vdash \forall X_\blacksquare P(X) \leftrightarrow \Psi$	(J_2)
	(L_3)	$\langle L_1, L_2 \rangle \vdash P(C)$	(J_3)
	(L_4)	$\langle L_1, L_2 \rangle \vdash P(D)$	(J_4)
	(L_5)	$\langle L_1, L_2 \rangle \vdash \Psi_1$	(`def-e` L_2, L_3)
	(L_6)	$\langle L_1, L_2 \rangle \vdash \Psi'_1$	(`def-e` L_2, L_4)
	(L_7)	$\langle L_1, L_2 \rangle \vdash \Psi_2$	(OPEN L_1, L_5, L_6)
	(L_8)	$\langle L_1, L_2 \rangle \vdash P(F'(C,D))$	(`def-i` L_2, L_7, L_8)
Procedural Content	`schema − interpreter`		

Note that the `hom1-2` method is indeed useful to solve the intended problem of showing that the intersection of two symmetric relations is symmetric too. From the initial problem the method `hom1-2` produces the following partial proof:

1.	1;	$\vdash$	$\forall \rho, \sigma_\blacksquare \forall x, y_\blacksquare \langle x, y\rangle \in \mathit{intersection}(\rho,\sigma) \leftrightarrow \langle x, y\rangle \in \rho \wedge \langle x, y\rangle \in \sigma$	(J1)
2.	2;	$\vdash$	$\forall \sigma_\blacksquare \mathit{symmetric}(\sigma) \leftrightarrow \forall x, y_\blacksquare \langle x, y\rangle \in \sigma \rightarrow \langle y, x\rangle \in \sigma$	(J2)
3.	1,2;	$\vdash$	$\mathit{symmetric}(\rho)$	(J3)
4.	1,2;	$\vdash$	$\mathit{symmetric}(\sigma)$	(J4)
5.	1,2;	$\vdash$	$\forall x, y_\blacksquare \langle x, y\rangle \in \rho \rightarrow \langle y, x\rangle \in \rho$	(`def-e` 2 3)
6.	1,2;	$\vdash$	$\forall x, y_\blacksquare \langle x, y\rangle \in \sigma \rightarrow \langle y, x\rangle \in \sigma$	(`def-e` 2 4)
7.	1,2;	$\vdash$	$\forall x, y_\blacksquare \langle x, y\rangle \in \mathit{intersection}(\rho,\sigma) \rightarrow \langle y, x\rangle \in \mathit{intersection}(\rho,\sigma)$	(OPEN 1 5 6)
8.	1,2;	$\vdash$	$\mathit{symmetric}(\mathit{intersection}(\rho,\sigma))$	(`def-i` 2 7)

Analogously a method `hom2-1` (for handling a unary function symbol but a binary predicate symbol) can be obtained by applying `add-argument` with the arguments `hom1-1` and P.

In an interactive proof development environment like Ω-MKRP [11] the user has the opportunity to choose and apply a meta-method himself. To provide the user with heuristic support or even to automatize this meta-level planning, heuristics are necessary. For a discussion on heuristics and for a preliminary classification of meta-methods, readers are referred to [12]. Another elaborated meta-method `connective-to-quantifier` can be found in [10].

6. Conclusion

A good mathematician has to learn a remarkable repertoire of technical knowledge. On the one hand this includes factual knowledge, namely definitions, theorems, and proofs. On the other hand he has to learn problem solving know-how as well. This kind of knowledge consists of standard *methods* for manipulating proofs, like mathematical induction or diagonalization. Among other important activities of mathematical reasoning (like defining new concepts and adapting a definition so that new theorems can be proved) there is one very important feature, namely the ability to adapt existing problem solving facilities to new, not directly fitting situations.

In order to model such mechanical modification we have presented in this article a formal definition of a method language. In particular we have defined the notion of a method with the following components: declarations, premises, constraint, conclusions, declarative part, and procedural part. The main feature of our approach is the separation of procedural and declarative knowledge in the tactic part. In this way, all parts are declarative and subject to automatic modification, except the procedural content of the tactic. Another technical emphasis of this article is a declarative constraint language. With this language we can bind free variables and formulate applicability conditions not expressible in terms of proof line schemata. In order to compromise between competing requirements, namely expressivity, adaptability, and tractability; the constraint language is designed as a decidable variant of sorted first-order logic.

The dual semantical characteristics of a method correspond to its two different roles: a method is a tactic and a plan operator. The semantics of the tactic part specifies the effect of its execution, while the semantics of the plan operator specifies its behavior in a planning process. In order to model the latter we have presented how a method can be translated into a STRIPS plan operator, together with a preliminary version of a planning algorithm. In the last section we have shown how the modification of methods can be performed by so-called meta-methods, producing new methods applicable in new situations.

To summarize we have proposed a declarative extension to Bundy's proof planning framework in order to enable reformulations of methods. Much work remains to be done. We are currently extending our first implementation, the interpreter for the constraint language is almost finished. The efficiency of the whole approach depends largely on the implementation of a planning algorithm, which is more sophisticated than the naive STRIPS-like planner. We hope to judge the adequacy of our approach by accumulating experience with more examples.

References

[1] Peter B. Andrews. Transforming matings into natural deduction proofs. In Wolfgang Bibel and Robert Kowalski, editors, *Proceedings of the 5th CADE*, Les Arcs, France, 1980. Springer Verlag, Berlin, Germany, LNCS 87.

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

[3] Alan Bundy. The use of explicit plans to guide inductive proofs. In Ewing Lusk and Ross Overbeek, editors, *Proceedings of the 9th CADE*, pages 111–120, Argonne, Illinois, USA, May 1988. Springer Verlag, Berlin, Germany, LNCS 310.

[4] 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.

[5] 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.

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

[7] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, **2**:189–208, 1971.

[8] Gerhard Gentzen. Untersuchungen über das logische Schließen I & II. *Mathematische Zeitschrift*, **39**:176–210, 572–595, 1935.

[9] Fausto Giunchiglia and Paolo Traverso. Program tactics and logic tactics. In Frank Pfenning, editor, *Proceedings of the 5th International Conference on Logic Programming and Automated Reasoning*, pages 16–30, Kiev, Ukraine, 1994. Springer Verlag, Berlin, Germany, LNAI 822.

[10] Xiaorong Huang and Manfred Kerber. Reuse of proofs by meta-methods. IJCAI-Workshop on Formal Approaches to the Reuse of Plans, Proofs, and Programs, 1995. forthcoming.

[11] 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.

[12] 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.

[13] Andrew Ireland. The use of planning critics in mechanizing inductive proofs. In Andrei Voronkov, editor, *Proceedings of the 3rd International Conference on Logic Programming and Automated Reasoning*, pages 178–189, St. Petersburg, Russia, 1992. Springer Verlag, Berlin, Germany, LNAI 624.

[14] Allen Newell. The heuristic of George Polya and its relation to artificial intelligence. In Rudolf Groner, Marina Groner and Walter F. Bishoof, editors, Methods of Heuristics, Lawrence Erlbaum, Hillsdale, New Jersey, USA, pages 195–243.

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

[16] George Pólya. *Mathematics and Plausible Reasoning.* Princeton University Press, Princeton, New Jersey, USA, 1954. Two volumes, Vol.1: Induction and Analogy in Mathematics, Vol.2: Patterns of Plausible Inference.

[17] Manfred Schmidt-Schauß. *Computational Aspects of an Order-Sorted Logic with Term Declarations*, Springer Verlag, Berlin, Germany, LNAI 395, 1989.

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

(Received: October 21, 1994; revised version: April 28, 1995)

Authors' address:

X. Huang, M. Kerber, J. Richts, A. Sehn
Fachbereich Informatik
Universität des Saarlandes
Postfach 15 11 50
D-66041 Saarbrücken
Germany
e-mail: {huang|kerber|richts|acsehn}@cs.uni-sb.de