Methods – The Basic Units for Planning and Verifying Proofs

Xiaorong Huang, Manfred Kerber, Michael Kohlhase*
Fachbereich Informatik, Universität des Saarlandes
Im Stadtwald, W-6600 Saarbrücken 11, Germany
{huang|kerber|kohlhase}@cs.uni-sb.de

Abstract

This paper concerns a knowledge structure called *method*, within a computational model for human oriented deduction. With human oriented theorem proving cast as an interleaving process of planning and verification, the body of all methods reflects the reasoning repertoire of a reasoning system. While we adopt the general structure of methods introduced by Alan Bundy, we make an essential advancement in that we strictly separate the declarative knowledge from the procedural knowledge. This is achieved by postulating some standard types of knowledge we have identified, such as inference rules, assertions, and proof schemata, together with corresponding knowledge interpreters. Our approach in effect changes the way deductive knowledge is encoded: A new compound declarative knowledge structure, the proof schema, takes the place of complicated procedures for modeling specific proof strategies. This change of paradigm not only leads to representations easier to understand, it also enables us modeling the even more important activity of formulating meta-methods, that is, operators that adapt existing methods to suit novel situations. In this paper, we first introduce briefly the general framework for describing methods. Then we turn to several types of knowledge with their interpreters. Finally, we briefly illustrate some meta-methods.

Keywords: Deduction, Planning and Verification, Methods, Declarative and Procedural Knowledge, Tactics.

*This work was supported by the Deutsche Forschungsgemeinschaft, SFB 314 (D3)

1 Introduction

There has been growing concern in the automated theorem proving community, that general purpose machine oriented procedures like resolution might have reached their limit in practice. Therefore the old discussion of the merits of a human-oriented vs. a machine oriented approach to automated theorem proving has been revived by researchers like Alan Bundy. In response to his request for a "science of reasoning" [3] a string of systems and theories have been proposed that aim at combining human-oriented deduction methods with sophisticated planners.

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, the elimination of a variable, the isolation of a function symbol, or a clearly described proof sketch for proving a theorem by diagonalization.

During his academic training a mathematician has to accumulate lots of methods, both domain-specific and general. This body of methods is the reasoning repertoire which together with the factual knowledge, to a great extent forms his technical knowledge. Another equally important knowledge source of a mathematician is his ability to adapt existing methods to suit a new situation. When he faces a new problem where no existing method fits, he will very often try to transform his reasoning repertoire in order to solve the new problem by analogy.

Much of this discussion has been anticipated by George Pólya in the context of analyzing mathematical reasoning (eg. "How to Solve It" [14]), where he gives hundreds of examples that mathematicians have to learn during their training. Some of these methods have been stated very explicitly, for instance, the method of two loci in geometry instructs us to describe the same point in two different ways in order to obtain some equations. Others are very general and are largely illustrated with the help of examples only. Allen Newell [12] discussed the relevance of Pólya's heuristics very intensively, although he did not achieve a formalization.

Among the concrete systems proposed so far the approach of Alan Bundy [2] is probably the most influencing and advanced. He views methods essentially as a triple consisting of a tactic, a precondition, and a postcondition. There the tactic is a piece of program code that can manipulate the actual proof in a controlled way. The precondition and the postcondition form a specification of the deductive ability of the tactic, formulating declaratively the applicability condition of the tactic and a description of the proof status after its application. This has been an essential progress compared with the mere tactic language of LCF [6] or the system Nuprl [4], because within this framework it is now possible to develop proof plans with the help of the declarative knowledge in the preconditions and postconditions. Following a one-sided approach relying on procedural knowledge only, the Oyster-Clam system developed by Bundy's group, still has however a severe drawback: the adaption of

methods to other problems 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.

To enable more natural manipulations on existing methods, we advance in this paper an extension of Bundy's notion of methods by separating the procedural and declarative knowledge in the tactic part of methods. We propose a notion of method, that consists of a five-tuple: precondition, postcondition, rating (these three tell when to apply a method), and the declarative content as well as the procedural content (these two slots contain the information that is stored in Bundy's tactic slot).

Since the entire discussion is embedded within a computational model of human deductive reasoning, section 2 first provides a brief sketch of the computational model. Section 3 is devoted to the main topic, the method structure. Apart from the general structure, we also propose three types of object level methods which quite naturally correspond to the knowledge structure of a human mathematician. As a third issue, we discuss meta-methods, which manipulate (object level) methods. A summary and a discussion in section 4 conclude the paper.

2 General Framework

Statically, we cast a reasoning being as a knowledge based system. We assume the existence of a planner and a verifier that manipulate the proof tree, which is the central datastructure that always reflects the current state of proof development is. A proof tree is an ordered tree where every node is a quadruple:

<Derived-Formula, Method-Name, List-of-Support-Nodes, Status>

The first slot consists of a formula in a fixed object logic. Since the logic is quite standard we do not discuss it here any further. The whole quadruple means that the formula is or might be derived, using the method (to be explained below) indicated, from the support nodes. The status slot has only three possible fillers: verified, unverified, or rejected. Support nodes must precede the node supported in the order defined by the ordered tree.

In our computational model we ascribe a reasoner's reasoning competence mainly to the existence of methods that reflect the reasoner's basic deductive repertoire. Methods essentially consist of a reasoning procedure, a piece of declarative knowledge, and a specification. We will elaborate on this concept in greater detail in section 3. To a remaining gap in the current proof tree, the planner usually consults the set of methods at his disposal. The chosen method normally proposes a subtree which can be integrated into the current proof tree. Since not all methods are sound or really fill the gap, a verification process must follow. In addition, the planner may also decide to generate new object level methods by applying meta-methods on existing object level methods.

Dynamically, we assume the entire process, from the analysis of a problem to the completion of a proof, to be an *interleaving process* of planning and verification.

This process is centered around the current proof tree, which accommodates concepts like *proof sketches*, *proof plans*, and *proofs*. For a more comprehensive discussion of the general framework of the computational model for human deductive reasoning, readers are referred to [9, 10].

3 Methods

The concept of method is central to the reasoning process, since methods are the basic units which are planned and carried out and body of methods constitutes the basic reasoning repertoire, that is constantly adapted and enriched, as experiences are collected. In the following subsection, we first provide a general definition of the structure of methods, and compare our definition with similar concepts already introduced in the literature. Then we turn to three types of specific methods identified thus far. In the last subsection, we illustrate how new methods can be constructed.

3.1 General Concepts and Classifications

In our computational model, we define every method as having the following slots:

- *Rating*: A function indicating whether the method is total or partial, and evaluating the appropriateness of applying this method.
- *Precondition*: Specifying preconditions of the problems a method is intended to solve.
- *Postcondition*: Specifying the effect the method will end up with.
- *Declarative content*: A piece of declarative knowledge. We currently only deal with three types of object level declarative knowledge: the natural deduction inference rules, the assertions (being facts either assumed of or proved previously), and proof schemata.
- *Procedural content*: Either a standard procedure *interpreting* the piece of declarative knowledge, or a special purpose inference procedure devised for a specific type of problems.

Viewed within a planning framework, the *precondition* and the *postcondition* slots together constitute the logical part of the *specification* of a method, which are both constraints on the partial proof tree. In other words, by these two conditions it is specified whether a method is applicable in a particular proof state or not. If several applicable methods are found the rating procedure should estimate how promising each one is. We are not going to elaborate on this concept, although for real planning tasks this rating may be crucial. For details we refer to [15].

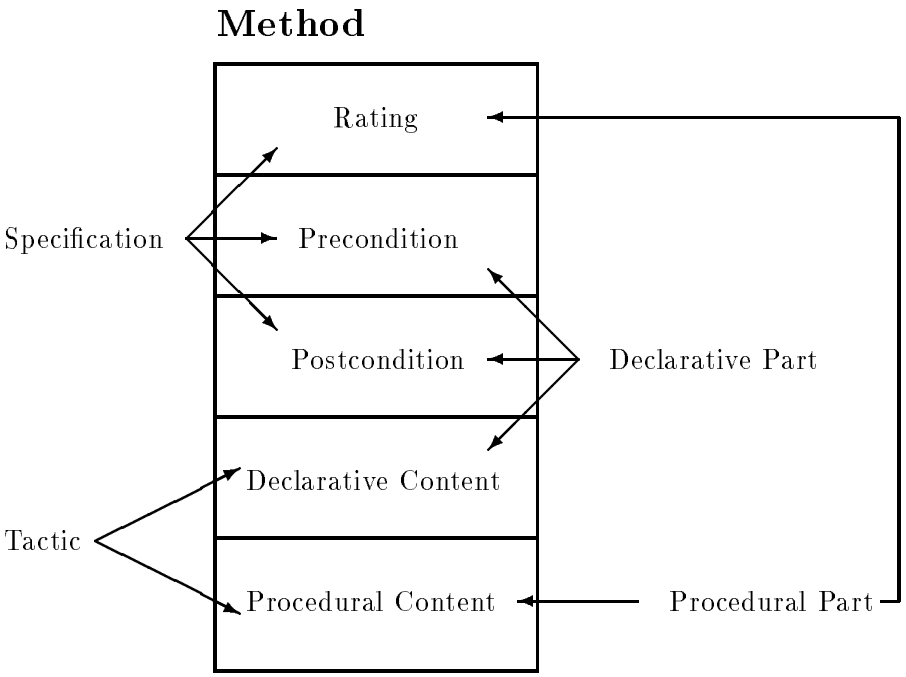

Figure 1: The Structure of Methods

We assume, that the planner exclusively consults the specification while planning a proof. The *declarative content* and the *procedural content* slots play the role of a so-called *tactic* of systems like Nuprl [4] or Bundy's framework for proof planning [3]. Concretely, the declarative content can be an arbitrary piece of declarative knowledge, and the procedural content a Lisp procedure of the following format:

```
interpreter(DK Prooftree &optional other-information)
```

In other words, it is an interpreter which takes as input a piece of declarative knowledge, a pointer to the current proof tree, and optionally other information, and produces a subproof tree that can be integrated into the current partial proof tree. From the logic point of view, the *precondition*, the *postcondition* and the *declarative content* slot together constitute the *logical part* of a method. These different partitions are illustrated in figure 1.

There was a long and heated debate in AI as to whether knowledge should be represented procedurally or declaratively. Arguments were put forward for both positions from psychological and computational perspectives. Advantages and drawbacks are discussed with respect to, among others, flexibility, computational efficiency, communicability. Resulting from this discussion (cf. [18]) it has been realized that both forms of knowledge are necessary to simulate intelligent behavior. Nevertheless most existing interactive proof development environments follow a one-sided approach relying on procedural knowledge only. Although we do not want to claim the psychological reality of our theory, we believe it is plausible that both aspects play an important role in human theorem proving.

Purely computationally, the generalizing the concept of tactic from a procedure (in Bundy's framework) to a pair containing both a procedure and a piece of declarative knowledge is also significant. By discerning the declarative part of tactics, it is now possible to formulate meta-methods adapting the declarative part of existing methods and thus come up with novel methods. If a tactic consisted of only procedural knowledge, we would in effect be confronted with the much more difficult problem of program synthesis, in order to achieve the above.

While in for instance the special purpose tactics of Bundy, the power of the method rests on the procedural part (the declarative content can be empty), the three types of object level methods to be introduced in the next subsection are supported by interpreters, which are standard and simple. Thus our framework is cast so general that it accommodates both a small set of general purpose procedures which operate by applying pieces of domain-specific declarative knowledge, and an open-end set of special purpose reasoning procedures, in which knowledge needed is already implicitly incorporated. In this paper, we are going to concentrate mainly on three types of general purpose methods at the object level, which are elaborated upon in detail in subsequent subsections. In the rest of this subsection, we discuss some general features along which methods may vary: A method is called

cognitively primitive, if it is planned and verified as a primitive unit, and its applications lead to the insertion of a single node in the proof tree.

cognitively compound, if its application results in a compound subtree containing nodes justified by subordinate methods it calls.

total, if the execution of its tactic part will certainly bring about the postconditions of the method, if the precondition is satisfied;

partial, if it is only likely, but not guaranteed, that the tactic part will bring about the postcondition even when the precondition is satisfied.

Note that for some methods the same tactic parts are identical and only the specifications, thus forming total and partial methods for one tactic. Since total methods are usually much more complicated than partial ones, a reasoner often tends to employ a partial method in the more global planning phase, leaving the precise checking to a more refining planning phase or even to the verification process. Indeed, besides the most primitive methods (see subsection 3.2.1), it is even difficult to devise feasible total methods. To keep the structure simple, we do not allow multiple specifications in one method in this first version of our theory. As a consequence, we must assume the existence of methods with identical tactic parts.

3.2 Three General Types of Object Level Methods

Within the framework set up so far, we are going to introduce three types of object level methods, each handled in a subsection below. Technically, each type of method

is primarily defined by coupling one standard interpreter with chunks of declarative knowledge of one type of object level knowledge. In the sequel, we refer to these methods as methods *applying* a piece of declarative knowledge. To each type of method, we also suggest some plausible pre- and postconditions. The first two types, the applications of natural deduction inference rules and the application of assertions, are cognitively primitive. We want to emphasis their naturalness, which in effect allows us to formulate proof schemata, the third type of object level knowledge, in a quite intuitive way.

3.2.1 The Methods Applying Rules of Inference

First we are introduce a procedure that *applies* rules of inference, a natural consequence of the natural logic hypothesis. Currently we assume an order sorted predicate logic of higher-order [10] as the working language, a language adequate for formalizing mathematics. However, the main content of this paper is independent to the choice of language. As the set of inference rules, we adopt the *natural deduction* system first proposed by Gerhard Gentzen [5, 1]. The following is a listing of several important inference rules similar to those presented in his calculus NK, with additional restrictions on sort structures. If a term t is of the sort S, we denote it as $t : S$. For a detailed definition, see [10].

$$\frac{\triangle, F \vdash G}{\triangle \vdash F \Rightarrow G} DEDuction, \qquad \frac{\triangle \vdash F \vee G;\ \triangle, F \vdash H;\ \triangle, G \vdash H}{\triangle \vdash H} CASE,$$

$$\frac{\triangle \vdash \exists x : S1 . F_x;\ \triangle, F_{a:S2} \vdash H;\ Subsort(S2, S1)}{\triangle \vdash H} CHOICE,$$

While the rules of inference included in the natural deduction system are considered as cognitively *elementary* and *innate*, a human reasoner may learn new, domain-specific rules during the reasoning activities, in which he is involved. For example, a rule about subset might be learned:

$$\frac{a \in S_1, S_1 \subseteq S_2}{a \in S_2}$$

where "a", "S_1" and "S_2" are metavariables of type "$Element$" or "Set". These new rules have the cognitive status *acquired* and *compound*. For more detailed discussions, the readers are referred to [7].

Now we turn to the notion of the applications of such rules of inference, and their role in the entire process of proof searching. We assume in our theory that the application of a rule of inference is carried out by a general purpose interpreter which mainly matches formula schemata in rules against formulas contained in support nodes. As a Lisp function, it has the format:

```
rule-interpreter(rule proof-tree & other-information)
```

Technically speaking, given a rule of inference of the form:

$$\frac{P_1, \ldots, P_n}{Q} \tag{1}$$

the rule interpreter allows both the derivation of Q' from $P'_1, \ldots, P'_n$, where Q' and $P'_1, \ldots, P'_n$ are the corresponding instances of Q and $P_1, \ldots, P_n$, and the derivation of $\neg P'_i$ from $P'_1, \ldots, P'_{i-1}, P'_{i+1}, \ldots P'_n$, and $\neg Q'$. Usually, the argument "other-information" points out a set of nodes in the proof tree serving as the support nodes. For rules where the instantiation cannot be determined by the matching alone (for example the Univ_Elim rule, the instatiation of the universal quantifier), additional information must be provided in the argument "other-information". Now for every rule of inference, we have a method which applies it, since the definition above fully specifies the ability of such methods, and yet is simple enough to be checked without undue efforts, it is plausible to assume that it may be instantiated for every particular rule of inference, and serve as specification in the corresponding methods.

3.2.2 Methods Applying Assertions

The second type of important object level knowledge is also encountered every day by mathematicians. It concerns objects such as axioms, definitions, lemmas and theorems, and even intermediate results achieved during proof search. They are, in our theory, collectively called *assertions*. Moreover, assertions are normally also interrelated in complex conceptual structures [11]. The notion of the *application* of an assertion, though normally not defined precisely, bears a central role both in proof searching and proof documentation. One prima facie evidence is that proofs found by mathematicians are almost exclusively presented in terms of the applications of some assertions.

Let us first illustrate this concept by examining a concrete example of the applications of assertions. Given an assertion defining the notion of subset:

$$\forall S_1, S_2 : Set_S_1 \subseteq S_2 \Leftrightarrow \forall x : Element_x \in S_1 \Rightarrow x \in S_2$$

We may derive

- $a \in S'_2$ from $a \in S'_1$ and $S'_1 \subseteq S'_2$;
- $S'_1 \not\subseteq S'_2$ from $a \in S'_1$ and $a \notin S'_2$;
- $\forall x : Element_x \in S'_1 \Rightarrow x \in S'_2$ from $S'_1 \subseteq S'_2$.

and so on; by *applying* this definition.

Although no introspection is possible to reveal the internal structure of the interpreter applying assertions, in [8], we have associated every application of an assertion to a proof segment justified by the natural deduction rules only, referred to as its

natural expansion. By studying the natural expansions in our preliminary empirical studies on naturally occurring mathematical proofs, we came up with a *characterization* of the input-output relation for the primitive procedure applying assertions. The characterization as well as the related definitions can be found in [8].

The reasoning ability of a method applying a certain assertion equals to that of a finite set of compound inference rules. However, it is not plausible to suggest that the planning decisions are made based on this information. As a means of assessment, it is apparently too complicated and time consuming. The kind of partial methods we believe to be a viable approximation is defined in the following pattern:
Suppose A is an arbitrary assertion, the following is one possible method applying A:

Method: `application-`A	
rating	rating-application-A
pre	exlineset $L_{\blacksquare}\forall l \in L_{\blacksquare}$instance-subformula-neg(formula(l), A)
post	exline $n_{\blacksquare}$ justification(n)=application-A $\wedge$ instance-subformula-neg(formula(n),A)
dec-cont	A
proc	`assertion-interpreter`

Here the predicate instance-subformula-neg(l, A) checks if l is either a subformula of A, an instance thereof, or, thirdly, a negation of the first two cases.

3.2.3 Methods Applying Proof Schemata

The third type of methods is tied to a more novel kind of knowledge structure called *proof schema*, and an interpreter *instantiating* them. These notions are introduced to account for the well-observed phenomenon, that people benefit from their successful and unsuccessful experiences. In other words, with the accumulation of experience, the reasoning ability of a reasoner also evolves. In our theory, this is simulated by the evolution of the collection of proof schemata at the disposal of a reasoner.

Intuitively, proof schemata are proofs or abstract proofs which provide solutions to reasoning problems. At the very beginning, a proof schema is usually a complete or partial proof found by a reasoning subject for a previous problem. A (partial) specification of the corresponding problem can serve as the pre- and postcondition of the method. Undergoing meta-level manipulations, proof schemata also provide solutions to novel problems. These manipulated proof schemata may contain meta-variables, which are instantiated by concrete formulas by the procedure applying the proof schemata. Technically, for now, it suffices to understand proof schemata as proof trees containing meta-level variables.

The following method `hom1` is a very simple example of a method applying a proof schema. It represents the following proof strategy: 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 very idea is that f is a homomorphism for the property P and that f can be "rippled out" (compare [2]).

Method: `hom1`		
rating	`rating-hom1`	
pre	(exline 1) $\wedge$ (exline 2)	
post	(exline 5)	
dec-cont	1. 1; $\vdash$ $\forall x_{\blacksquare}\text{Formula}_f$	(J1)
	2. 2; $\vdash$ $\forall x_{\blacksquare}P(x) \Leftrightarrow \text{Formula}'$	(J2)
	3. 3; $\vdash$ $P(c)$	(PLAN)
	4. 4; $\vdash$ Eq_Elim_r(Univ_Elim((2), $f(c)$), (5))	(PLAN 2)
	5. 2,4; $\vdash$ $P(f(c))$	(PLAN 2 4)
proc	`schema-interpreter`	

The specification of this method means that `hom1` can be applied if the lines 1 and 2 exist in the partial proof under construction and line 5 is an open goal. In this case, the `schema-interpreter` will insert line 3 and 4 into the partial proof, as well as adapt the justification of line 5.

The formulas in line 1 and 2 are properties of the function f and the predicate P (e.g. their definitions). Both f and P are meta-variables standing for (function/predicate) constants of the object logic. As opposed to formulas in other lines, which are given as formula schemata, the formula in line 4 must be constructed by the `schema-interpreter` applying the natural deduction rules for eliminating universal quantifier and equivalence.

For example, to prove that the converse relation of a binary relation ρ is symmetric (formally: symmetric(converse(ρ))), the method `hom1` can be applied by substituting converse, symmetric, and ρ for the meta-variables f, P, and c, respectively. The resulting proof fragment is listed below:

1. 1;	$\vdash$ $\forall\sigma_{\blacksquare}\forall x, y_{\blacksquare}\langle x, y\rangle \in \text{converse}(\sigma) \Leftrightarrow \langle y, x\rangle \in \sigma$	(J1)
2. 2;	$\vdash$ $\forall\sigma_{\blacksquare}\text{symmetric}(\sigma) \Leftrightarrow \forall x, y_{\blacksquare}\langle x, y\rangle \in \sigma \Rightarrow \langle y, x\rangle \in \sigma$	(J2)
3. 3;	$\vdash$ symmetric(ρ)	(PLAN)
4. 4;	$\vdash$ $\forall x, y_{\blacksquare}\langle x, y\rangle \in converse(\rho) \Rightarrow \langle y, x\rangle \in converse(\rho)$	(PLAN 2)
5. 2,4;	$\vdash$ symmetric(converse(ρ))	(PLAN 2 4)

3.3 Mechanisms Constructing Methods

Our theory is also devised to account for the evolution of the reasoner's basic reasoning repertoire. This is achieved by the existence of *meta-methods* and automatic learning procedures. Notice, while the methods cause changes in the current partial proof, meta-methods enrich the knowledge base by adding new methods. Meta-methods are usually invoked by the intention to solve a specific problem, and their applications require concentration and efforts, as opposed to those more perceptual procedures, like remembering a proof or a rule, running in a more uncontrolled way.

3.3.1 The Combination of Methods

In this subsection, we briefly explain the automatic part of meta-level activities. These activities are very similar to those called *learning during problem solving* described in cognitive models in general, and in frameworks for problem solving in particular, for a comprehensive survey, readers are referred to [16].

The most simple form of learning is similar to the process called *compounding* identified in problem solving process. There, the compounding process puts two or more existing operators into a sequence. A mechanism called *chunking* is proposed in [13], which combines compounding with the *tuning* process adapting the heuristic knowledge associated with the operators. In our framework, chunking can be viewed as compounding instantiated methods, adding information about the instantiation into pre- and postcondition.

Methods in our theory may be combined similarly, yet in a little bit more complicated way. We are not going to go into details here, interested readers are referred to [10].

3.3.2 Meta-Methods

As already mentioned, our theory is also devised to account for evolution of the reasoner's basic reasoning repertoire. In addition to those procedures merely remembering useful information, this is achieved mainly through the existence of meta-methods manipulating proof schemata. When a reasoner is confronted with a novel yet similar problem, proof schemata evolving from previously successfully found proofs are modified to cope with the new problem. This is also the advice Pólya gives in his survey to problem solving [14].

As opposed to methods, meta-methods are thought to be very general and problem independent. As a consequence of this we do not think that meta-meta-methods and a whole hierarchy of meta-levels are necessary.

Currently we have identified two groups of meta-methods. Guided by heuristic knowledge of different kinds, they will

- *generalize* existing methods built upon a proof schema, or,
- *reformulate* existing methods built upon a proof schema, to suit new problems.

The second kind of meta-method consists of a concrete mapping stated in the declarative content and an interpreter for mappings, which applies a mapping in a controlled way to the logical content of the method to be reformulated. In particular there are strict constraints on mappings to be applied on proof schemata that prohibit the formation of syntactically ill-formed formulas. For details see [10]. A discussion on reformulations can also be found in [17]. For space restrictions, we are only going to illustrate our approach to meta-methods with a *generalization* example.

In section 3.2.3 we have introduced the method `hom1`, which simplifies a problem by generating an intermediate goal, where a *unary* function symbol is eliminated. Suppose we are facing the novel problem of proving that the union of symmetric relations is itself a symmetric relation. What we need is a variant of `hom1`, which is able to handle a *binary* function symbol (i.e. "union") in a similar way.

In the following, we illustrate how to use the meta-method `add-argument` to obtain a binary version `hom2` from the unary version `hom1`.

Meta-Method: `add-argument`	
rating	meta-add-argument-rating
pre	exmethod `M`. subterm($f(x)$,post(`M`))
post	goal=post(proc-add-argument(α,`M`))
dec-cont	$\alpha = \{f(x) \mapsto g(x,y)\}$
proc	`proc-add-argument`

This meta-method is supposed to add an argument to a key function f used in a method, this modified function is called g. Note that the precondition states that there is indeed such a function in `M`. In order to ensure that f is important to `M`, it is required that f is a part of the postcondition of `M`. Based on the mapping given as the declarative content, the procedure `add-argument` modifies the proof schema in `M` by primarily carrying out the following three actions:

- replace all occurrences of terms $f(x)$ by $g(x,y)$ and modify the corresponding quantifications,
- replace all occurrences of terms $f(c)$ by $g(c,d)$ (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 below, line 4 is copied from 3).

As a crucial advantage of separating the procedural and the declarative knowledge in methods, the procedural content of `M` can be taken over for the new method.

If we apply `add-argument` to `hom1`, we obtain the new method `hom2`.

Method: `hom2`	
rating	`rating-hom1`
pre	(exline 1) (exline 2)
post	(exline 6)
dec-cont	1. 1; $\vdash$ $\forall x, y_{\blacksquare}\text{Formula}_g$ (J1) 2. 2; $\vdash$ $\forall x_{\blacksquare} P(x) \Leftrightarrow \text{Formula}'$ (J2) 3. 3; $\vdash$ $P(c)$ (PLAN) 4. 4; $\vdash$ $P(d)$ (PLAN) 5. 5; $\vdash$ Eq_Elim_r(Univ_Elim((2), $g(c,d)$), (5)) (PLAN 2) 6. 2,5; $\vdash$ $P(g(c,d))$ (PLAN 2 5)
proc	`schema-interpreter`

It is generally the case as in this example, the information of meta-methods is largely encoded as procedures. We believe, however, that this is not a real drawback, since meta-methods are devised in a domain independent way, and therefore no meta-meta-methods are needed.

4 Conclusion

In this paper, we have proposed a knowledge structure called method. With this notion, we want to suggest a change of paradigm concerning the encoding of deductive knowledge. Instead of encoding proof strategies as complicated procedures that are difficult to understand and to adapt, our approach uses a compound declarative knowledge structure called proof schema. The naturalness of the methods applying proof schemata is due in large part to the naturalness of the types of primitive methods, namely the applications of inference rules and the applications of assertions. The appropriateness of assuming these three types of methods is supported by examining the proofs in mathematical text books. We want also to indicate that our method structure can accommodate more procedural knowledge as well, this is however not the subject of concern here.

Our declarative approach is not only cognitively more adequate, it is also computationally more feasible. Meta-methods have been devised to adapt existing methods applying proof schemata to suit new situations. As opposed to methods, meta-methods are normally not domain-specific but of a very general nature. There is no need therefore for meta-meta-methods. In [10], an example can be found how the proof of a diagonalization problem is modified to solve three other similar problems.

Methods and meta-methods can be used in two ways: as powerful deductive operators in an interactive proof development environment, or as the basic reasoning repertoire of an automated proof planner. For the latter purpose, much more experience must still be gathered concerning the formulation of specifications of methods. We are also working on a general mechanism to accommodate more powerful meta-methods.

Acknowledgement

We would like to thank Jörg Denzinger, Erica Melis, and Inger Sonntag for many fruitful discussions about proof plans, which inspired and clarified many of the ideas presented here. In addition, thanks are due to Erica Melis and Dan Nesmith for their comments on a draft of this paper.

References

[1] P. B. Andrews. Transforming Matings into Natural Deduction Proofs. LNCS 87, Springer, 1980.

[2] A. Bundy. The Use of Explicit Plans to Guide Inductive Proofs. In *CADE-9*, Springer, 1988.

[3] A. Bundy. A Science of Reasoning: Extended Abstract. In *CADE-10*, Springer, 1990.

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

[5] G. Gentzen. Untersuchungen über das logische Schließen I. *Math. Zeitschrift*, 39, 1935.

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

[7] X. Huang. An Extensible Natural Calculus for Argument Presentation. Technical Report SEKI SR-91-3, Univ. Kaiserslautern, 1991.

[8] X. Huang. Applications of assertions as elementary tactics in proof planning. In V. Sgurev and B. du Boulay, editors, *AIMSA-92*. Elsevier Science Publishers, 1992.

[9] X. Huang. An explanatory framework for human theorem proving. In H. J. Ohlbach, editor, *GWAI-92*, LNAI, Springer, 1992.

[10] X. Huang, M. Kerber, and M. Kohlhase. Theorem proving as a planning and verification process. Technical Report to appear as SEKI Report, Univ. des Saarlandes, 1992.

[11] M. Kerber. On the representation of mathematical knowledge in frames and its consistency. In *WOCFAI-91*, 1991.

[12] A. Newell. The heuristic of George Polya and its relation to artificial intelligence. Technical Report CMU-CS-81-133, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, USA, 1981.

[13] A. Newell and P. Rosenbloom. Mechanism of skill acquisition and the law of practice. In J. R. Anderson, editor, *Cognitive Skills and Their Acquisition.* Hillsdale, 1981.

[14] G. Pólya. *How to Solve it.* Princeton Univ. Press, 1945.

[15] I. Sonntag and J. Denzinger. Extending automatic theorem proving by planning. Personal communication, Fachbereich Informatik, Univ. Kaiserslautern, 1992.

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

[17] F. J. T. William M. Farmer, Joshua D. Guttman. Little theories. In D. Kapur, editor, *CADE-11.* LNAI 607, Springer, 1992.

[18] T. Winograd. Frame representations and the declarative/procedural controversy. In D. G. Bobrow and A. M. Collins, editors, *Representation and Understanding: Studies in Cognitive Science.* Academic Press, 1975.