The Adaptation of Proof Methods by Reformulation

Xiaorong Huang* Manfred Kerber† Lassaad Cheikhrouhou*

1 Introduction

A mathematician possesses a repertoire of problem solving methods that he/she has been acquiring and extending during his/her career. Confronted with a problem he/she tries to apply a known method either directly or after adaptation to the problem. The process of adaptation in addition to other learning processes extends his/her repertoire with new problem solving methods. In the mathematical assistant system Ω-MKRP [HKK+94] we want to imitate this human problem solving behavior. In order to achieve that we need a suitable framework for representing proof techniques and adapting them. We have chosen a declarative approach to capture proof techniques in form of tactics with some specifications. Following the proof planning terminology of Bundy [Bun88] we call these units *methods*. The adaptation is carried out by some procedures called *meta-methods*. The method base is extended either by inserting a new method resulted from the adaptation of an old one or by abstracting two similar methods to a more general one.

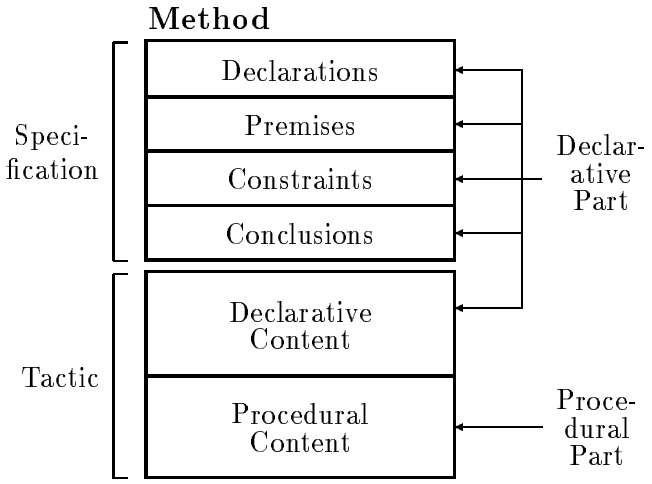
In this extended abstract we describe briefly, beside the declarative representation of methods, an adaptation meta-method and a meta-method for the generalization of similar methods. A more detailed primary specification of these meta-methods is given in [HKC95].

2 Representation of methods

For a tractable modification, methods are represented in a declarative formalism where each method consists of the following slots:

*Fachbereich Informatik, Universität des Saarlandes, D-66041 Saarbrücken, Germany {huang|lassaad}@cs.uni-sb.de

†School of Computer Science, The University of Birmingham, Birmingham, B15 2TT, England M.Kerber@cs.bham.ac.uk



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

Premises: Schemata of ND (natural deduction) lines (cp. [Gen35, And80]) which are used by the method to prove the conclusions. An ND line consists of a label, a sequent with a hypothesis list and a formula, and a justification,

Constraints: Additional restrictions on the premises and the conclusions, which can not be formulated in terms of ND line schemata,

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

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

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

Each symbol in the formula of an ND line schema is declared either as a typed meta-variable or as a constant. A meta-variable stands for any object-level term with the same type and a constant is interpreted as an object-level symbol. The constants, called *key symbols*, allow the representation of semantical informations in a method. (The key symbols of the methods

considered in this extended abstract are either logical constants or expression written in bold face.)

The declarative content, also referred to as the proof schema, can be constructed from an ND proof by stressing important proof informations and abstracting other proof details. The method `Diag1` in figure 1 is obtained from an ND proof of the Cantor theorem stating that there is no surjective function from a set M into its power set. The ND proof is first abstracted to the so-called assertion level [Hua94]. At this level the application of assertions, such as definitions or theorems, can be used as justifications, in addition to the application of ND rules. The key proof steps from the resulted assertion-level proof are captured in the method proof schema and the other intermediate steps are eliminated. The ND lines in this partial proof justified with 'PLAN' are actually not proved (closed) and therefore referred to as open. They specify the subgoals which result from the application of the method to prove an open goal matching the method conclusion (line 15). The proof of the Cantor theorem is an indirect proof: We assume the negation of the theorem as a hypothesis (line 1) and deduce a contradiction by diagonalization. The contradiction is implicitly given in the diagonal element represented by the lambda expression in line 3 and made explicit by a case analysis (from line 7 to line 12).

3 Method adaptation

We consider the method `Diag1` to prove that there is no surjective function from the natural numbers into the real interval $[0,1]$ which can be formalized by the conjecture $\neg\exists g_{\iota\to(\iota\to\iota)}.\mathbf{surj}(g,\mathbb{N},I_{0,1})$. The attempt to match this theorem with the conclusion of `Diag1` fails because of two problems: the constant $I_{0,1}$ corresponds to a function application $P(M)$ on the side of the method conclusion and the function g, having a set of functions as range, should be matched to the function f whose range $P(M)$ is a set of sets. This mismatch can be resolved by adapting the method `Diag1` with the help of the meta-method `set2function` which intuitively makes it possible to use functions (the elements of $I_{0,1}$) instead of sets (the elements of $P(M)$): It maps the application $P(M)$ to a meta-variable P_M with the appropriate type, changes the type of f accordingly, and solves the resulting type inconsistencies within the method terms. Basically a type inconsistency consists of the occurrence of a (sub-)term whose type is different from the appropriate type in the term structure. Such a (sub-)term is prefixed with

a function which transforms it into an individual with the correct type at the corresponding position in the term structure:

- The problem imposed by the occurrence of a formula instead of a ι-term[1] is solved by introducing a function that builds up the truth values to different elements $\overline{A}$ and $\overline{B}$ of the ι-domain[2]. There is one possible function modulo the elements $\overline{A}$ and $\overline{B}$ which accomplishes this task, namely $\lambda x_o . \mathrm{if}(x, \overline{A}, \overline{B})$.
- A ι-term occurs at the position of a formula. The function to be introduced must map the ι-domain to the truth values and must be compatible with the inverse of the function considered to solve the first kind of type inconsistency. Three functions are possible:
  - the function $\lambda x_\iota . x = \overline{A}$ maps $\overline{A}$ to *true* and the rest of the ι-domain to *false*,
  - the function $\lambda x_\iota . \neg x = \overline{B}$ maps $\overline{B}$ to *false* and the rest of the ι-domain to *true*,
  - and the function $\lambda x_\iota . \overline{\Phi}(x)$ maps a subset $\overline{\Phi}$, including $\overline{A}$, to *true* and its complementary set, including $\overline{B}$, to *false*.

When solving the type inconsistencies we cannot foresee the actual functions to be used. These functions are then represented using constrained meta-variables, denoted by over-lined symbols. Such meta-variables are partially specified by the constraints and become fully instantiated during the rest of the proof planning process. The call of the meta-method `set2function` with the appropriate parameters constructs the method `Diag2` in figure 2 which can be used to prove the new theorem.

4 Method generalization

Since we prefer methods to be general in order to be applicable in many cases, it is possible to abstract two similar methods as `Diag1` and `Diag2` with the meta-method `abstract-methods`. This leads to the more general method `Diag12` in figure 3 which can be used to prove Cantor theorem and the non-denumerability of the real interval $[0, 1]$.

[1] A ι-term is a term with the type ι which is the type of all individuals different from the truth values. The type of truth values is o.

[2] The ι-domain consists of the individuals with type ι.

The meta-method `abstract-methods` checks whether the given methods are similar and constructs the most special generalization of them. Two methods are *similar* if their proof schemata have the same tree structure and the nodes are similar. We call two nodes *similar* if they have the same *key skeleton* which is the part of the term tree associated with the key sub-terms.

The *key sub-terms* are instantiated at the beginning of the abstraction process to the set of the method constants (key symbols) and later extended to other meta-variables. First we consider the conclusions, second the premises, and then the rest of the nodes. While testing the similarity of two nodes, by checking whether the key skeleton of a node occurs in the term structure of the other node, differences between the two term structures are bridged by inserting new so-called *generalizing meta-variables*, denoted by underlined symbols. The most special generalization is guaranteed by preserving as much as possible of the original term structures. For instance two different applications $h(a)$ and $k(b)$ which do not belong to the key skeleton are generalized to the application $\underline{F}(\underline{C})$. The generalizations carried out while considering two nodes are propagated throughout the proof schemata. The sub-terms in the resulting node are considered as key sub-terms in the rest of the abstraction process.

Generalizing meta-variables could abstract important proof information. In order to prevent the loss of such information we plan to make use of their instantiations in the generalized methods.

5 Conclusion

The declarative formalism for method representation is an adequate basis to capture mathematical proof techniques in a natural way as it is exemplified by the method `Diag1`. The formalism enables a tractable adaptation of such proof techniques with the help of meta-methods. Each meta-method is a procedure which creates a new method by carrying out some defined modifications of the given methods. Therefore, meta-methods like `set2function` and `abstract-methods` do not depend on the methods to be adapted,

The use of meta-methods in the process of proof planning raises the question when to switch from planning to meta-planning and which modifications should be carried out for the adaptation of some inapplicable method. In general this is achieved by a user, but the use of heuristic control knowledge would be interesting for more automation.

References

[And80] 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.

[Bun88] 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, May1988. Springer Verlag, Berlin, Germany, LNCS 310.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen I. *Mathematische Zeitschrift*, **39**:176–210, 1935.

[HKC95] Xiaorong Huang, Manfred Kerber, and Lassaad Cheikhrouhou. Adaptation of declaratively represented methods in proof planning. SEKI Report SR-95-12, Fachbereich Informatik, Universität des Saarlandes, Im Stadtwald, Saarbrücken, Germany, 1995.

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

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

Declarations	—
Premises	Surj-Def
Constraint	—
Conclusions	15
Declarative Content	1. 1 $\vdash$ $\exists f_{\blacksquare}\mathbf{surj}(f, M, P(M))$ (Hyp) 2. 1,2 $\vdash$ $\mathbf{surj}(f_0, M, P(M))$ (Hyp) 3. 1,2 $\vdash$ $P(M)(\lambda z_{\blacksquare}[M(z) \wedge \neg[f_0(z)(z)]])$ (PLAN) 4. 1,2 $\vdash$ $\exists y_{\blacksquare}[M(y) \wedge \lambda z_{\blacksquare}[M(z) \wedge \neg[f_0(z)(z)]] = f_0(y)]$ (Surj-Def 2 3) 5. 1,2,5 $\vdash$ $[M(y_0) \wedge \lambda z_{\blacksquare}[M(z) \wedge \neg[f_0(z)(z)]] = f_0(y_0)]$ (Hyp) 6. 1,2,5 $\vdash$ $[M(y_0) \wedge \neg[f_0(y_0)(y_0)]] \leftrightarrow f_0(y_0)(y_0)$ (PLAN 5) ——— Case 1 ——— 7. 1,2,5,7 $\vdash$ $f_0(y_0)(y_0)$ (Case 1) 8. 1,2,5,7 $\vdash$ $\bot$ (PLAN 6 7) ——— Case 2 ——— 9. 1,2,5,9 $\vdash$ $\neg[f_0(y_0)(y_0)]$ (Case 2) 10. 1,2,5,9 $\vdash$ $\bot$ (PLAN 6 9) ——— End of Case 2 ——— 11. $\vdash$ $[f_0(y_0)(y_0) \vee \neg[f_0(y_0)(y_0)]]$ (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. $\vdash$ $\neg[\exists f_{\blacksquare}\mathbf{surj}(f, M, P(M))]$ (NotI 14)
Procedural Content	`schema-interpreter`

Figure 1: Method `Diag1`

Declarations	—
Premises	Surj-Def
Constraint	—
Conclusions	15
Declarative Content	1. 1 $\vdash$ $\exists f_{\blacksquare}\mathbf{surj}(f, M, P_M)$ (Hyp) 2. 1,2 $\vdash$ $\mathbf{surj}(f_0, M, P_M)$ (Hyp) 3. 1,2 $\vdash$ $P_M(\lambda z_{\blacksquare}\mathrm{if}([M(z) \wedge \neg\overline{P}[f_0(z)(z)]], \overline{A}, \overline{B}))$ (PLAN) 4. 1,2 $\vdash$ $\exists y_{\blacksquare}[M(y) \wedge \lambda z_{\blacksquare}\mathrm{if}([M(z) \wedge \neg\overline{P}[f_0(z)(z)]], \overline{A}, \overline{B}) = f_0(y)]$ (Surj-Def 2 3) 5. 1,2,5 $\vdash$ $[M(y_0) \wedge \lambda z_{\blacksquare}\mathrm{if}([M(z) \wedge \neg\overline{P}[f_0(z)(z)]], \overline{A}, \overline{B}) = f_0(y_0)]$ (Hyp) 6. 1,2,5 $\vdash$ $[M(y_0) \wedge \neg\overline{P}[f_0(y_0)(y_0)]] \leftrightarrow \overline{P}[f_0(y_0)(y_0)]$ (PLAN 5) Case 1 7. 1,2,5,7 $\vdash$ $\overline{P}[f_0(y_0)(y_0)]$ (Case 1) 8. 1,2,5,7 $\vdash$ $\bot$ (PLAN 6 7) Case 2 9. 1,2,5,9 $\vdash$ $\neg\overline{P}[f_0(y_0)(y_0)]$ (Case 2) 10. 1,2,5,9 $\vdash$ $\bot$ (PLAN 6 9) End of Case 2 11. $\vdash$ $\overline{P}[f_0(y_0)(y_0)] \vee \neg\overline{P}[f_0(y_0)(y_0)]$ (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. $\vdash$ $\neg[\exists f_{\blacksquare}\mathbf{surj}(f, M, P_M)]$ (NotI 14)
Procedural Content	`schema-interpreter`

Figure 2: Method `Diag2`

Declarations	—
Premises	Surj-Def
Constraint	—
Conclusions	15
Declarative Content	1. 1 $\vdash$ $\exists f_{\blacksquare}\mathbf{surj}(f, M, \underline{S})$ (Hyp) 2. 1,2 $\vdash$ $\mathbf{surj}(f_0, M, \underline{S})$ (Hyp) 3. 1,2 $\vdash$ $\underline{S}(\lambda z_{\blacksquare}\underline{F}[M(z) \wedge \neg\underline{G}[f_0(z)(z)]])$ (PLAN) 4. 1,2 $\vdash$ $\exists y_{\blacksquare}[M(y) \wedge \lambda z_{\blacksquare}\underline{F}[M(z) \wedge \neg\underline{G}[f_0(z)(z)]] = f_0(y)]$ (Surj-Def 2 3) 5. 1,2,5 $\vdash$ $[M(y_0) \wedge \lambda z_{\blacksquare}\underline{F}[M(z) \wedge \neg\underline{G}[f_0(z)(z)]] = f_0(y_0)]$ (Hyp) 6. 1,2,5 $\vdash$ $[M(y_0) \wedge \neg\underline{G}[f_0(y_0)(y_0)]] \leftrightarrow \underline{G}[f_0(y_0)(y_0)]$ (PLAN 5) ——— Case 1 ——— 7. 1,2,5,7 $\vdash$ $\underline{G}[f_0(y_0)(y_0)]$ (Case 1) 8. 1,2,5,7 $\vdash$ $\bot$ (PLAN 6 7) ——— Case 2 ——— 9. 1,2,5,9 $\vdash$ $\neg\underline{G}[f_0(y_0)(y_0)]$ (Case 2) 10. 1,2,5,9 $\vdash$ $\bot$ (PLAN 6 9) ——— End of Case 2 ——— 11. $\vdash$ $\underline{G}[f_0(y_0)(y_0)] \vee \neg\underline{G}[f_0(y_0)(y_0)]$ (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. $\vdash$ $\neg[\exists f_{\blacksquare}\mathbf{surj}(f, M, \underline{S})]$ (NotI 14)
Procedural Content	`schema-interpreter`

Figure 3: Method `Diag12`