Representing and Reformulating Diagonalization Methods

Erica Melis

July 1994
CMU-CS-94-174

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

The author is on leave from University of Saarbrücken

Abstract

Finding an appropriate representation of planning operators is crucial for theorem provers that work with proof planning. We show a new representation of operators and demonstrate how diagonalization can be represented by operators. We explain how a diagonalization operator used in one proof-plan can be analogically transferred to an operator used in another proof-plan. Finally, we find an operator that is common to all the proof-plans and thus might be considered as the *Diagonal Method*.

This research was supported by the Max-Kade Foundation

Keywords: proof planning, analogy, knowledge representation

1 Introduction

As pointed out by Bundy [3] and Bledsoe [1], using proof-plans is often very helpful in automated deduction. In planning, operators are needed and therefore an appropriate representation of these operators is crucial for proof planning. The operators have the same function in proof planning as mathematical methods (in the following referred to as m-methods) have in human theorem proving. Since m-methods can be adapted to different proofs, it is also desirable to have mechanisms for adapting operators. To be employed by a human-oriented theorem prover, these operators should allow for representing logical proof methods, such as `Indirect Proof`, and mathematical methods, such as Cantor's `Diagonal` method.

In this paper we examine whether the presented representation actually covers mathematician's methods and how the methods can be adapted for other proof plans. We do this by analyzing the well-known Diagonal Method which is central and widely applicable in many mathematical proofs concerning computability and decidability, including Gödel's Incompleteness theorem for arithmetic, the Unsolvability of the halting problem, Rice's theorem, and the Second Recursion theorem (see [5]). Although this m-method seems to be clearly understood, not all proofs have an obvious common proof schema, and some proofs are difficult to generate in logical detail.

After defining our representation of operators by *methods*, we investigate several proofs in which mathematicians have used diagonalization. We sketch these proofs and show which methods belong to the respective proof-plans. We also discuss, how a method from the proof-plan for Cantor's theorem can be transferred to a method for a proof-plan of the halting problem and of Gödel's First incompleteness Theorem. Finally, a comparison of the methods yields a method that is common to all the proofs and which might be considered as *the Diagonal Method*.

2 Representation of Methods

First we give a brief definition of methods that allow for reformulation (For more details see [6]).

Sequents P, written as $(ass \vdash concl)$, are pairs of a set ass of formulas and a formula $concl$ in an object language that is extended by meta-variables for formulas, sets of formulas, and terms. As an abbreviation, we shall display formulas F instead of sequents $(\emptyset \vdash F)$.

Methods M are frame-like structures similar to Bundy's methods in [3]. Methods have the slots parameter, preconditions (pre(M)), postcondition (post(M)), constraints, proof schema and procedure. Preconditions is a set of sequents, postcondition is a sequent; both pre- and postconditions are needed in planning. Constraints are formulated in a meta-language and serve to restrict the search during planning and may, e.g., express restrictions of pre(M) and post(M). The proof schema is a declarative schematic representation of not necessarily correct proofs in the object logic, relying on the Natural Deduction (ND) calculus and on invoking automated theorem provers such as OTTER. That is, the justifications of lines in the proof schema may be Natural Deduction rules, the name of a theorem prover proving

the sequent in the line, or LEMMA[1]. If LEMMA occurs as a justification in line$_i$ of proof-schema(M), then the sequent of line$_i$ is an element of pre(M). Other justifications, such as Method$_i$ or several ND-rules, provide just an abbreviation for a sequence of lines. The program in the slot procedure executes the application of the proof schema by interpreting the proof schema. The structural template for methods is:

method: `name of the method`	
parameter	parameters which can be instantiated
preconditions	preconditions that have to be true for the method to be applicable
postcondition	postconditions that should be fulfilled after the method application, e.g., a derived sequent
constraints	meta-language constraints that may restrict preconditions, postconditions, and parameter
proof schema	a declarative proof schema
procedure	procedure that interprets the proof schema

By additionally representing the search direction we obtain operators, called forward-methods and backward-methods depending on whether they have the postcondition or the preconditions as inputs. Corresponding to the ND-rules, the simple method $\wedge$-deletion is typically employed in forward search, whereas the method $\wedge$-introduction is typically employed in backward search[2]. The operators are called verifiable if the corresponding method is verifiable. A method M is *verifiable* if it can be checked that for every instantiation of the meta-variables the method is correct, i.e., it yields a correct proof of post(M) when applied to pre(M) in case the constraints are satisfied.

The methods defined here differ from those in [3] mainly in that the tactic slot is replaced by a declarative proof schema *and* a procedure interpreting this schema. The intention behind this difference is to enable reformulations of methods.

3 The Diagonalization Methods

We want to check whether the definition of methods, as described above, is an appropriate representation for m-methods. To that end we investigate some proofs in which mathematicians have used diagonalization. Usually diagonalizations proceed by
supposing $a_1, a_2 \ldots$ is an enumeration of objects of a certain kind. Then an object a of the same kind is constructed, that is different from every a_n using the following principle: "Make a and a_n differ at n." The interpretation of *differ at n* depends on the kind of object involved.

In the following we examine proofs of

[1]For hierarchical planning we may admit method variables PLAN$_i$ as justifications.

[2]In case the planner employs backward-methods only, the proof-plan concept and the analogy procedure defined later become much simpler.

1. **Cantor's Theorem**, which states that for any set M the cardinality of M is smaller than the cardinality of the powerset $\mathcal{P}M$ of M,
   $cardM < card\mathcal{P}M$ (see [2])

2. **Uncountability of the set of real numbers** (actually the interval [0,1]), which means that $card\ \mathbb{N} < card\ [0,1]$ (see [4])

3. **Unsolvability of the Halting Problem** for Turing machines, which means there is no algorithm (no t-computable function) to determine whether an arbitrary Turing machine in an arbitrary configuration with a finite length of nonblank tape symbols will eventually halt (see [7]). That is, no t-computable function c[3] exists with

$$c(t, \text{conf}_y) = \begin{cases} 1 & : \quad t \text{ does not halt with } \text{conf}_y \\ 0 & : \quad t \text{ halts with } \text{conf}_y \end{cases}$$

4. **Gödel's Incompleteness theorem for S**, which states that there is a sentence in the language of arithmetic S with addition and multiplication that is not provable in S neither is its negation(see [10]).

3.1 The Proof Sketches

Throughout the paper $(M \mapsto N)$ denotes the set of functions from M to N.

3.1.1 $cardM < card\mathcal{P}M$

1. **Unfold definition**
   The theorem $cardM < card\mathcal{P}M$ is rerepresented by applying the definition of the partial order of cardinals to the theorem:
   There is a one-to-one correspondence from all elements of M to a subset of $\mathcal{P}M$, but no one-to-one correspondence from all elements of $\mathcal{P}M$ to a subset of M. Since the proof of the first part is trivial, the task can be reduced to prove
   $\neg\exists f \forall x \exists y (f \in (M \mapsto \mathcal{P}M) \wedge (x \in \mathcal{P}M \rightarrow y \in M \wedge f(y) = x))$.

2. **Indirect proof**
   Derives $\neg\exists f \forall x \exists y (f \in (M \mapsto \mathcal{P}M) \wedge (x \in \mathcal{P}M \rightarrow y \in M \wedge f(y) = x))$ via a contradiction. This means, the indirect assumption[4] is
   $\forall x \exists y (x \in \mathcal{P}M \rightarrow y \in M \wedge F(y) = x)$, for a function $F \in (M \mapsto \mathcal{P}M)$. .

3. **Method D1** yields the contradiction. In more detail, D1 includes the derivations of

   (a) the existence of a function $G(x)$ with $G(x) = non(F(x)(x))$ with

$$non(x) = \begin{cases} t & : \quad x \neq t \\ f & : \quad x = t \end{cases}$$

[3] c would be the characteristic function of the halting problem.

[4] We call the negations of the theorem to be proved by an indirect proof the *indirect assumption*.

Using the assumption p_0: $\forall g(g \in \mathcal{P}M \to \forall x(x \in M \to g(x) \in \{t,f\}))$, the indirect assumption can be formulated for a function $F \in (M \mapsto M \mapsto \{t,f\})$. **Preconditions** for step (a) are the indirect assumption and the comprehension axioms[5] for *non* and compound functions.

(b) $\forall x(x \in M \to (F(x)(x) = t \to G(x) = f) \wedge (F(x)(x) \neq t \to G(x) = t))$.
Preconditions are (a) and the definitions of *non* and G.

(c) $G(x) \in \mathcal{P}M$. **Preconditions** for this step are
p_2: $\forall g \forall x(x \in M \to g(x) \in \{t,f\}) \to g \in \mathcal{P}M)$ and $p_4 : t, f \in \{t,f\}$.

(d) $F(x_0)(x_0) = G(x_0)$. **Preconditions** are the indirect assumption and $G \in \mathcal{P}M$.

(e) $\bot$ is proved in two steps from $F(x_0)(x_0) = t \vee \neg F(x_0)(x_0) = t$:
derive $\bot$ from $F(x_0)(x_0) = t$ and
derive $\bot$ from $F(x_0)(x_0) \neq t$. **Preconditions** are (c), (b), the definition of F, $f \neq t, \forall x(x = x)$.

The assumptions p_0 and p_2 are given by a representation lemma that states that each subset of M can uniquely be represented as a function from $(M \mapsto \{t,f\})$. Figure 1 shows the proof structure of the proof of Cantor's theorem as well as of the proof of the uncountability of $\mathbb{R}$.

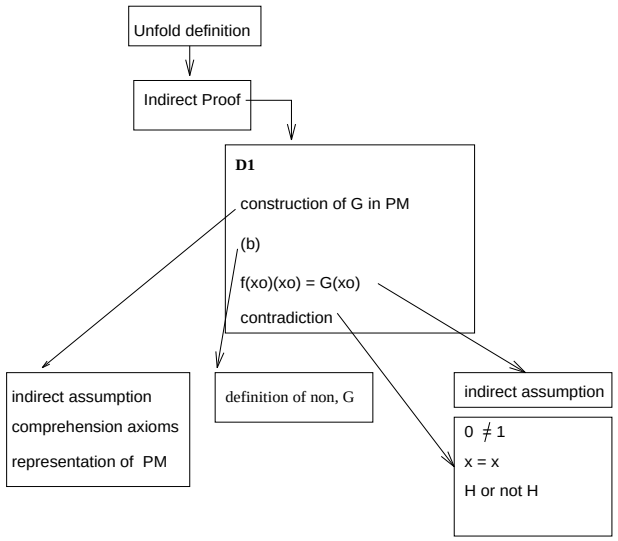

Figure 1: Proof Structure of Cantor's Theorem

3.1.2 $card\,\mathbb{N} < card[0,1]$

1. **Unfold definition**
   The theorem $card\,\mathbb{N} < card[0,1]$ is rerepresented by applying the definition of the

[5] This idea is due to X.Huang and M.Kerber who also gave an ND-proof

partial order of cardinals to the theorem:
There is a one-to-one correspondence from all elements of $\mathbb{N}$ to a subset of $[0,1]$, but no one-to-one correspondence from all elements of $[0,1]$ to a subset of $\mathbb{N}$. Since the proof of the first part is trivial, the task can be reduced to prove
$\neg\exists f\forall x\exists y(f \in (\mathbb{N} \mapsto [0,1]) \wedge (x \in [0,1] \rightarrow y \in \mathbb{N} \wedge f(y) = x))$.

2. **Indirect proof**
Derives $\neg\exists f\forall x\exists y(f \in (\mathbb{N} \mapsto [0,1]) \wedge (x \in [0,1] \rightarrow y \in \mathbb{N} \wedge f(y) = x))$ via a contradiction. The indirect assumption is $\forall x\exists y(x \in [0,1] \rightarrow y \in \mathbb{N} \wedge F(y) = x)$ for a function $F \in (\mathbb{N} \mapsto [0,1])$.

3. **Method D2** yields a contradiction. In more detail, D2 includes the derivations of

   (a) the existence of a function $G(x)$ with: $G(x) = nonF(x)(x)$ for

   $$non(x) = \begin{cases} 0 & : \quad x \neq 0 \\ 1 & : \quad x = 0 \end{cases}$$

   Using the assumption p_0= $\forall g(g \in [0,1] \rightarrow \forall x(x \in \mathbb{N} \rightarrow g(x) \in \{0 \ldots 9\}))$, the indirect assumption can be formulated for a function $F \in (\mathbb{N} \mapsto \mathbb{N} \mapsto \{0 \ldots 9\})$.
   Preconditions for step (a) are the indirect assumption and the comprehension axioms for *non* and compound functions.

   (b) $\forall x(x \in \mathbb{N} \rightarrow (F(x)(x) = 0 \rightarrow G(x) = 1) \wedge (F(x)(x) \neq 0 \rightarrow G(x) = 0))$.
   Preconditions for this step are (a) and the definitions of *non* and G.

   (c) $G(x) \in [0,1]$. **Preconditions** for this step are the assumptions
   p_2: $\forall g\forall x(x \in \mathbb{N} \rightarrow g(x) \in \{0 \ldots 9\}) \rightarrow g \in [0,1])$ and $p_4 : 1, 0 \in \{0 \ldots 9\}$.

   (d) $F(x_0)(x_0) = G(x_0)$. **Preconditions** are the indirect assumption, and $G \in [0,1]$.

   (e) $\perp$ is proved in two steps from $F(x_0)(x_0) = 0 \vee \neg F(x_0)(x_0) = 0$:
   derive $\perp$ from $F(x_0)(x_0) = 0$ and
   derive $\perp$ from $F(x_0)(x_0) \neq 0$. **Preconditions** are (c), (b), the definition of $F, 1 \neq 0, \forall x(x = x)$.

The assumptions p_0 and p_2 are given by a representation lemma that states that each real number in [0,1] can uniquely be represented as a function from $(\mathbb{N} \mapsto \{0 \ldots 9\})$. Figure 1 also shows the proof structure of the $\mathbb{R}$-proof for $\mathcal{P}M$ is replaced by [0,1] and t, f by 0,1.

3.1.3 Unsolvability of the Halting Problem

We follow the proof of the halting problem in [2]. Prior to conducting the proof, a Gödel enumeration of the configurations is assumed and the fact that Turing machines are t-computable functions. "t halts on $config_y$" is equivalent (modulo the presumed theory) to "$t(y)$ is defined" and "t does not halt on $config_y$" is equivalent to "$t(y)$ is undefined" for $y \in \mathbb{N}$, and thus the halting problem can be rerepresented as
There is no t-computable function c such that for $y \in \mathbb{N}$. $t \in T$ (Turing machines)

$$c(t(y)) = \begin{cases} 0 & : \quad t(y) \text{ defined} \\ 1 & : \quad t(y) \text{ undefined} \end{cases}$$

That is, the theorem is then $\neg\exists c(c \in T \wedge \forall t, n(t \in T \wedge n \in \mathbb{N} \rightarrow (c(t(n)) \in \{0,1\})))$, where $c \in T$ means c is t-computable.

1. **Indirect proof**
   derives the theorem via contradiction, starting with the indirect assumption $c \in T \wedge \forall t, n(t \in T \wedge n \in \mathbb{N} \rightarrow (c(t(n)) \in \{0,1\}))$.

2. **Method D3** infers a contradiction from the indirect assumptions ($c \in T$) and p_0: $\forall t, n(t \in T \wedge n \in \mathbb{N} \rightarrow (c(t(n)) \in \{0,1\}))$. Assuming a Gödel enumeration F of Turing machines, for each Turing machine t there is an $m \in \mathbb{N}$, and for each $m \in \mathbb{N}$ there is a $t \in T$, such that $F(m) = t$ and hence, $F(m)(n) = t(n)$ which is used throughout the proof.

   In more detail, D3 includes the derivations of

   (a) the existence of $G(x)$, such that $c(G(x)) = non(c(F(x)(x)))$, i.e.,

   $$G(x) = \begin{cases} 0 & : \quad c(F(x)(x)) = 1 \\ \text{undefined} & : \quad c(F(x)(x)) = 0 \end{cases}$$

   Preconditions are the definitions of c, non and F.

   (b) $\forall x(x \in \mathbb{N} \rightarrow (c(F(x)(x)) = 0 \rightarrow c(Gx) = 1) \wedge (c(F(x)(x)) \neq 0 \rightarrow c(Gx) = 0))$. **Preconditions** are (a) and the definition of non in p_8.

   (c) $(G \in T)$. **Preconditions** are Church's thesis and the definitions of c and F.

   (d) $F(x_0)(x_0) = G(x_0)$ for a constant x_0. **Preconditions** are $G \in T$ and the enumerability of Turing machines.

   (e) $\perp$ is proved in two steps:
   derive $\perp$ from $c(F(x_0)(x_0)) = 0$ and
   derive $\perp$ from $c(F(x_0)(x_0)) \neq 0$. **Preconditions** are (d), (c), $0 \neq 1, \forall x(x = x)$, $(c(F(x_0)(x_0)) = 0 \vee \neg c(F(x_0)(x_0)) = 0)$, and the definition of c.

Figure 2 shows a proof structure of the proof of the halting problem.

3.1.4 Gödel's Theorem

The following mathematical proof is taken from [11].
A Gödel numbering f of the expressions in S with one variable is assumed.
Lemma: The predicate R(xy) which states that the proof with Gödel number y proves the sentence which is the instantiation ϕ_x(x) of an one-variable-expression ϕ_x with Gödel number x by the number x is numeralwise expressible by $R(xy)$ in the arithmetic S.

Consider the formula $\forall y \neg R(xy)$. It has a Gödel number p, and thus $\forall y \neg R(xy) = \phi_p(x)$. Now consider the formula $\phi_p(p)$, i.e., $\forall y \neg R(py)$ which contains no variable free.

Theorem: If the number-theoretic formal system S is consistent, then not $\vdash_S \phi_p(p)$; and if the system is ω-consistent, then not $\vdash_S \neg\phi_p(p)$.

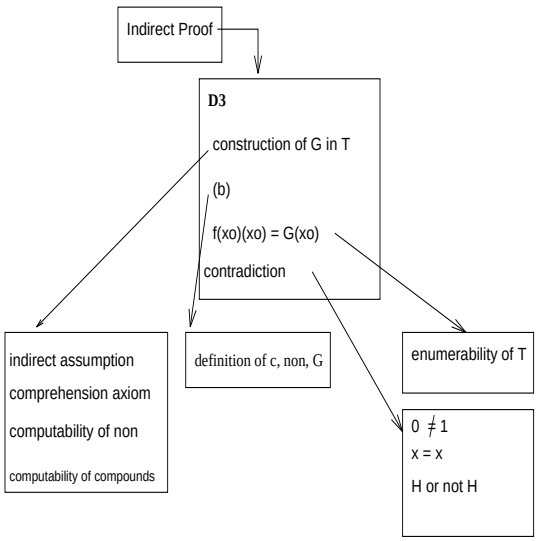

Figure 2: Proof Structure of the Halting Problem

For the function w defined for sentences H by
w(H) = provable_in_S(H), i.e., w is a function from object language sentences to meta-formulas. By the lemma above, we have $w(\phi_{\mathrm{x}}(\mathrm{x})) = w(f(x)(x)) = \exists y R(xy)$, i.e., $w(f(x)(x)) \in$ set_of_formulas. A logical reconstruction of the proof is:

1. **Indirect proof**
   Derives $\neg\forall g(g \in set_of_formulas \rightarrow (w(g) \vee w(\neg g)))$ via contradiction. A weakened[6] indirect assumption is $\forall g \forall x(g \in E \rightarrow x \in \mathbb{N} \rightarrow (w(g(x)) \vee w(\neg g(x))))$, where E is the set of all expressions in S with one free variable.

2. **Method D-Gödel** yields the contradiction. In more detail, D-Gödel includes the derivations of

   (a) the existence of G with $G(x) = neg(w(f(x)(x)))$, which is substantiated by the definitions of formulas and of w, f and by the definition $neg(F) = \neg F$.

   (b) $G \in E$. **Preconditions** are the definition of E and the lemma.

   (c) $f(x_0)(x_0) = G(x_0)$ for a constant x_0. **Preconditions** are (b) and the enumeration of the expressions of S by f.

   (d) $\bot$. This is shown by deriving a contradiction from $w(G(x_0)) \vee w(negG(x_0))$[7]. **Preconditions** are $\forall x((w(x) \rightarrow w(neg(neg(x)))) \wedge (negw(x) \rightarrow w(neg(x))))$,

[6] Weakened because it is only for sentences $g(x)$ with $g \in E$.

[7] Proof sketch: Assume $wG(x_0)$, then $\vdash G(x_0)$, then exists a proof p_k of $G(x_0)$, then $\vdash R(x_0 k)$, then $\vdash \neg G(x_0)$ because of p_8, then $w\neg G(x_0)$, then $\bot$ because of consistency of S. Hence, $\neg wG(x_0)$, then not exists a proof of $G(x_0)$, then $R(x_0 0), R(x_0 1) \ldots$ are false because of p_8, then $\vdash \neg R(x_0 0) \ldots$ because of the definition of R, then $\nvdash \neg\forall y \neg R(x_0 y)$ because of ω-consistency of S, then $\nvdash \neg G x_0$ because of the definition of G, then $\neg w\neg G(x_0)$ which yields the contradiction with $w\neg G(x_0)$.

the second conjunct of which is the indirect assumption, and the definitions of w, R, G, the consistency of S, and the ω-consistency of S.

In order to obtain a version of the method which is better comparable to D1, D2, and D3, we additionally introduce a characteristic function c for sentences H of S by

$$c(H) = \begin{cases} 0 & : \vdash_S H \\ 1 & : \vdash_S \neg H \end{cases}$$

Summarized the new version **D4** consists of:

(a) Show the existence of $G(x)$ with $G(x) = neg(w(f(x)(x)))$. **Preconditions** are the definition of formulas, the definitions of neg, f, comprehension axiom, and the lemma p_0: $\forall x(f(x) \in E \rightarrow (x \in \mathbb{N} \rightarrow w(f(x)(x)) \in set_of_formulas))$.

(b) $G \in E$ which follows from (a) and the lemma.

(c) $f(x_0)(x_0) = G(x_0)$ for a constant x_0. **Preconditions** are (b) and the enumeration of E by f.

(d) $\forall x(c(w(f(x)(x)))) = 0 \rightarrow c(G(x)) = 1) \wedge (c(w(f(x)(x))) \neq 0 \rightarrow c(G(x)) = 0))$ which can be proved using (a) and the sequent p_8: $\forall x((c(x) = 0 \rightarrow c(neg(x)) = 1) \wedge (c(x) \neq 0 \rightarrow c(neg(x)) = 0))$ derived from the definitions of c and neg..

(e) $\perp$ is proved starting with $c(w(f(x_0)(x_0))) = 0 \vee c(w(f(x_0)(x_0))) \neq 0$. Relying on (d), $c(w(f(x_0)(x_0))) = 0$ leads to a contradiction, using the ω-consistency of S[8], (c), and $0 \neq 1$ (which encodes the consistency of S), and $c(w(f(x_0)(x_0))) \neq 0$ leads to a contradiction as well, using (c), the definition of R, and $x = x$.

Figure 3 shows the proof structure of a proof of Gödel's theorem.

3.2 The Methods

One of the methods in all four proof plans is `Indirect Proof`:

method: `Indirect Proof`		
parameter	F: formula, Δ: set of formulae	
preconditions	$\Delta, \neg F \vdash \perp$	
postcondition	$\Delta \vdash F$	
constraints		
proof schema	1. $;\neg F \quad \vdash \neg F$	(HYP)
	2. $\Delta;\neg F \quad \vdash \perp$	(LEMMA)
	3. $\Delta; \quad \vdash \neg\neg F$	($\neg$ I;2)
	4. $\Delta; \quad \vdash F$	($\neg\neg D$;3)
procedure	schema-interpreter	

[8]The ω-consistency states that If $\neg R(0y), \neg R(1y), \ldots$ then $\not\forall \neg\forall x \neg R(xy)$, which allows to deduce $cG(x_0) = 1 \rightarrow cwG(x_0) = 1$.

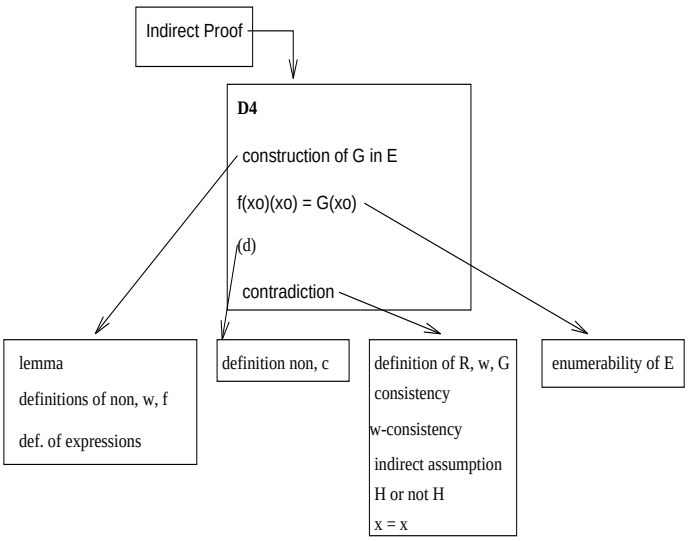

Figure 3: Proof Structure of Gödel's Theorem

A method that jointly represents D1 and D2 is D12, where p_1 is the indirect assumption referred to in D1 and D2, equ denotes the application of an equality axiom, and Method_i is an abbreviation for a certain sequence of lines.

method: D12	
parameter	F: function, $M1, M2$: structures, U: set, 0, 1: elements
preconditions	$p_0 : \forall g \forall x(g \in M2 \rightarrow (x \in M1 \rightarrow g(x) \in U))$ $p_1 : \forall x \exists y(x \in M2 \rightarrow y \in M1 \wedge F(y) = x)$, $p_2 : \forall g \forall x((x \in M1 \rightarrow g(x) \in U) \rightarrow g \in M2)$, $p_3 : U \subseteq domain(non)$, $p_4 : 0 \in U \wedge 1 \in U$, $p_5 : \forall x(x = x)$, $p_6 : 1 \neq 0$, $p_7 : \forall x(F(x)(x) = 0 \vee \neg F(x)(x) \neq 0)$, p_8: $\forall x((x \neq 0 \rightarrow non(x) = 0) \wedge (x = 0 \rightarrow non(x) = 1))$
postcondition	$\perp$
constraints	
proof schema	1. ; $\vdash \exists g \forall x(x \in M1 \rightarrow g(x) = non(F(x)(x)))$ (comprehension,p_0,p_3) 2. ; $\vdash \forall x(x \in M1 \rightarrow G(x) = non(F(x)(x)))$ ($\exists$D 1) 3. ; $\vdash \forall x(x \in M1 \rightarrow (F(x)(x) = 0 \rightarrow G(x) = 1) \wedge (F(x)(x) \neq 0 \rightarrow G(x) = 0))$ (Method_1,p_8 2) 4. ; $\vdash \forall z(z \in M1 \rightarrow G(z) \in U)$ (OTTER p_2 p_4 p_7 3) 5. ; $\vdash G \in M2$ ($\forall$D,$\leftrightarrow$D,p_2 4) 6. ; $\vdash \exists x(x \in M1 \wedge F(x) = G)$ ($\forall$D,$\rightarrow$D,,p_1 5) 7. ; $\vdash x_0 \in M1 \wedge F(x_0)(x_0) = G(x_0)$ (OTTER 6) 8. ; $\vdash \perp$ (OTTER p_5 p_6 p_7 3 7)
procedure	schema-interpreter

Some Definitions

In the following, *goals* and *assumptions* are intended to be sequents. A *proof-plan* is a forest. Its trees consist of sequent nodes and nodes for verifiable operators, where the successor of a sequent node $\mathtt{g}$ is a method node M and the successors of a method node are sequent nodes $\mathtt{g}_1, \ldots, \mathtt{g}_n = \overline{\mathtt{g}}$, such that the following "link condition" is satisfied: $\sigma(\text{post(M)}) = \mathtt{g}$ and $\sigma(\text{pre(M)}) = \overline{\mathtt{g}}$ for a substitution σ (using the obvious extension of σ to sequents and sequences of sequents). A *sketch* differs from a proof-plan in that it may contain methods which are not verifiable. In a proof-plan/sketch $\mathcal{P}$ a node N is termed *dependent* on a node N', if N is an ancestor of N' or N' is an ancestor of N in $\mathcal{P}$.

A proof-plan may contain forward-trees for which the sequents are assumptions and the operators are forward-methods. It may contain a backward-tree with a goal root node, backward-methods as operators, and goals as sequent nodes or it contains a tree which combines the backward-tree with forward-trees.

The planning starts from a root goal and from assumption leaves $(\emptyset \vdash F_i)$, where F_i are proof-assumptions, or $F_i \in KB$, where KB is the knowledge base of axioms, definitions, and lemmas. The planning proceeds by inserting methods and sequents satisfying the link

condition, always aiming at closing the gap between leaf goals and assumptions. Leaf goals that are not equal to an assumption are called *open goals*. As soon as a goal $\mathtt{g}_i$ equals an assumption, the two nodes collapse and thus the backward- and a forward-tree are combined. Then $\mathtt{g}_i$ is no longer an open goal but *satisfied*. The planning terminates if there are no open goals anymore.

3.3 Transferring Methods Analogically

We shall utilize the analogy-driven proof-plan construction for analogically transferring the method D12 of the Cantor and uncountability proof-plan to a method `Transferred D3` for the proof-plan of the halting problem and to a method `Transferred D4`. By employing the analogy-driven proof-plan construction we are able to choose the application of a reformulation according to the reformulations needed for matching goals and assumptions.

The general idea of analogy-driven proof-plan construction, detailed in [8], is to use a source proof-plan as a guide for constructing an analogous target proof-plan. The construction employs the structure and the methods belonging to the source plan or somewhat reformulated and restructured methods, for the target proof-plan.

The analogy-driven proof-plan construction includes two different kinds of mapping proof-plans/sketches, namely reformulation and restructuring as characterized below. *Reformulation* denotes a sequence of reformulations which are triggered by a source sequent. Reformulation aims at matching a source goal with a target goal or as many preconditions of a source method with target assumptions respectively.

Reformulations ρ map a proof-plan/sketch to another proof-plan/sketch while preserving the proof-plan/sketch structure. A set of admissible reformulations is stored in a data base. A reformulation ρ consists of a mapping of methods M to methods (ρ applied to M) and mappings of sequents P to sequents (ρ applied to P). If ρ is applied to a sequent $\mathtt{g}$ of a plan $\mathcal{P}$, then ρ has to be applied to the nodes in the $\mathcal{P}$ that are dependent on $\mathtt{g}$ as well. (Imagine, e.g., that a symbol is replaced by another one in M, then it has to be replaced in the same way in all methods and sequents dependent on M.) Thus a reformulation in analogy-driven proof-plan construction is not only dependent on ρ but also on the node in $\mathcal{P}$ that *triggers* the reformulation.

Restructuring aims at providing submethods of the source plan the postcondition (or preconditions) of which eventually matches an open target goal (or assumptions) or at splitting a method that cannot be verified in order to find a verifiable submethod. By finding a verifiable submethod the amount of work left for the base-level planning is diminished.

Restructuring maps a proof-plan/sketch to another proof-plan/sketch by replacing a sub-proof-plan $\mathcal{P}$ with one method by a proof-plan with several methods while preserving the root and leaves of $\mathcal{P}$. We refer to restructurings that are executed by restructuring meta-methods. Some of these meta-methods are `Deduction-Theorem-Splitting`, `Conjunctive-Decomposition`, and `Apply-Axiom-Splitting`. For a more detailed motivation, description and examples see [9].

Table 1 shows the top-level procedure of the analogy-driven proof-plan construction. The actual analogy procedure is embedded into the planning process. Starting with a given source proof-plan, target assumptions, and a target goal, the output of the procedure is a

proof-plan for the target goal.

The following specifics are to note in Table 1: Restructure_and_A_or_base-level_plan_for_post(M)_and_B. is an abbreviation for

- if restructuring M is possible, then
  - restructure M
  - update source plan
  - go to A
- go to B.

θ is a threshold usually set to 2. A goal `g` is considered *not establishable*, if neither `g` not a reformulation of `g` holds for the target. This judgement may be provided by the user and may reduce the search notably. Testing the verifiability is combined with producing additional preconditions of an operator if necessary. We employ Bledsoe's PC prover for this task. A procedure based on backward planning only is considerably simpler.

3.4 Transferring D12 to D3

We consider a proof-plan only containing the forward-method of D12. A new method with postcondition $(G \in M2)$ has to be found by base-level planning because a precondition similar to that of p_2 in D12 is not establishable for D3. Thus $(G \in M2)$ becomes an additional precondition in the reformulated D12 and the justification of its line becomes LEMMA.

After instantiating the parameter $M1, M2, U$ the assumption p_0 of the proof-plan of D12 can be reformulated to the assumption of D3 $p_0 = \forall t, n(t \in T \wedge n \in \mathbb{N} \rightarrow c(t(n)) \in \{0,1\})$ by replacing each term $term$ of type $f(x)$ with $f \in M2$ and $x \in M1$ by a term $c(term)$. This reformulation is a constrained `Term Mapping`. It has to be applied to all the nodes of the source proof-plan which are dependent on p_0.

Since the line $(1. \vdash \exists g \forall x(x \in M1 \rightarrow c(g(x)) = non(c(F(x)(x))))$ compr.) cannot be verified, the current reformulated method is restructured, such that one of the two resulting methods contains the former lines 1 and 2. Since this method cannot be verified and not restructured it has to be replaced by another method with the same postcondition $\forall x(x \in M1 \rightarrow c(g(x)) = non(c(F(x)(x))))$ which is tried to find by base-level planning. Note that establishing this subgoal requires to creatively find the very construction of a Turing machine G satisfying $c(G(x)) = non(c(F(x)(x)))$. This construction yields the $G(x)$ described in the sketch of the proof. The analogy-driven proof-plan construction has produced a proof-plan that contains the method `Transferred D3` with the new preconditions p_9 and p_{10} shown below.

method: `Transferred D3`	
parameter	F, G, c, non: (partial) function, $M1, M2$: structure, U: set, 0,1: elements
preconditions	$p_0 : \forall g \forall x(g \in M2 \rightarrow (x \in M1 \rightarrow c(g(x)) \in U))$ $p_1 : \forall x \exists y(x \in M2 \rightarrow y \in M1 \wedge F(y) = x)$, $p_5 : \forall x(x = x)$ $p_6 : 1 \neq 0$ $p_7 : c(F(x)(x)) = 0 \vee \neg c(F(x)(x)) = 0$, $p_8 : \forall x((x \neq 0 \rightarrow non(x) = 0) \wedge (x = 0 \rightarrow non(x) = 1))$ $p_9 : \forall x(x \in M1 \rightarrow cG(x) = non(c(F(x)(x)))$ $p_{10} : G \in M2$
postcondition	$\bot$
constraints	
proof schema	1. ; $\vdash$ $\forall x(x \in M1 \rightarrow cG(x) = non(c(F(x)(x)))$ (LEMMA) 2. ; $\vdash$ $G \in M2$ (LEMMA) 3. ; $\vdash$ $\exists x(x \in M1 \wedge F(x) = G)$ ($\wedge$ D,$\rightarrow$D,p_1 2) 4. ; $\vdash$ $x_0 \in M1 \wedge c(F(x_0)(x_0)) = c(G(x_0))$ (OTTER 3) 5. ; $\vdash$ $\forall x(x \in M1 \rightarrow (c(F(x)(x)) = 0 \rightarrow c(G(x)) = 1) \wedge (c(F(x)(x)) \neq 0 \rightarrow c(G(x)) = 0))$ ($Method_1$,p_8 1) 6. ; $\vdash$ $\bot$ (OTTER p_5 p_6 p_7 4 5)
procedure	schema-interpreter

3.5 Transferring D12 to D4

The construction of D4 proceeds similarly to the previous transfer relying only on actual proof assumptions such as the definition of c rather than on derived sequents such as $p_8 = \forall x((c(x) \neq 0 \rightarrow c(neg(x)) = 0) \wedge (c(x) = 0 \rightarrow c(neg(x)) = 1))$. This transfer produces, however, a method `Transferred D4` different from the expected D4.

Again, a submethod providing $(G \in M2)$ as a postcondition has to be found by base-level planning because an assumption similar to p_2 in D12 cannot be established.

The assumption p_0 of D4 is $\forall g \forall x(g \in E \rightarrow x \in \mathbb{N} \rightarrow c(g(x)) \in \{0,1\})$ which is given by the definition of c and it suggests a reformulation from p_0 of D12 by a `Term Mapping` $(term \Rightarrow c(term))$ for terms of type $f(x)$ for $f \in M2$ and $x \in M1$. As in the reformulation to `Transferred D3` this reformulation yields a new submethod consisting of line 1 and line 2, for line 1 that cannot be verified or structured and, hence, its postcondition (1) $\emptyset \vdash \forall x(x \in M1 \rightarrow c(G(x)) = non(c(F(x)(x))))$ becomes an open subgoal which has to be satisfied by base-level planning. Note that $G(x) = neg(w(F(x)(x)))$ satisfies (1) and $G \in M2$ which can be shown by a method employing the indirect assumption, the consistency, and the ω-consistency of S. Eventually the proof-plan construction yields the method `Transferred`

`D4` that is identical to `Transferred D3`. Basically this method could be used in all four proof plans. Then, however, a subplan with three operators replaces the original single operator.

3.6 The Diagonal Method

In a hierarchical planning setting method variables can be introduced as the justifications of the new LEMMA lines, thereby indicating that the respective sequents have to be shown by a recursive call of the planner. This call results in different subplans in our different cases. Thus we obtain a top level method `Diagonal` where PLAN2 and PLAN 3 together denote the unspecified proof of what is often called the "diagonalization lemma" in mathematical diagonalization proofs.

method: `Diagonal`	
parameter	F, G, c, non: (partial) function, $M1, M2$: structure, U: set, 0,1: element
preconditions	$p_0 : \forall g \forall x(g \in M2 \rightarrow (x \in M1 \rightarrow c(g(x)) \in U))$, $p_1 : \forall x \exists y(x \in M2 \rightarrow y \in M1 \wedge F(y) = x)$, $p_5 : \forall x(x = x)$ $p_6 : 0 \neq 1$, $p_7 : c(F(x)(x)) = 0 \vee \neg c(F(x)(x)) = 0$, p_8 $\forall x(x \neq 0 \rightarrow non(x) = 0) \wedge (x) = 0 \rightarrow non(x)) = 1))$
postcondition	$\perp$
constraints	
proof schema	1. ; $\vdash$ $\forall x(x \in M1 \rightarrow (c(G(x)) = non(c(F(xx)))))$ (PLAN2) 2. ; $\vdash$ $G \in M$ (PLAN3) 3. ; $\vdash$ $\exists x(x \in M1 \rightarrow F(x) = G)$ ($\rightarrow$D,$\forall$D p_1 2) 4. ; $\vdash$ $x_0 \in M1 \rightarrow c(F(x_0)(x_0)) = c(G(x_0))$ (OTTER 3) 5. ; $\vdash$ $\forall x(c(F(xx)) = 0 \rightarrow c(G(x)) = 1) \wedge (c(F(xx)) \neq 0 \rightarrow c(G(x)) = 0)$ (Method$_1$) 6. ; $\vdash$ $\perp$ (OTTER p_5 p_6 p_7 4 5)
procedure	schema-interpreter

References

[1] W. Bledsoe. A precondition prover for analogy. In *Biosystems*. 1994. in press.

[2] G.S. Boolos and R.C. Jeffrey. *Computability and Logic*. Cambidge University Press, Cambridge, third edition, 1989.

[3] A. Bundy. The use of explicit plans to guide inductive proofs. In E. Lusk and R. Overbeek, editors, *Proc. 9th International Conference on Automated Deduction (CADE)*,

volume 310 of *Lecture Notes in Computer Science*, pages 111–120, Argonne, 1988. Springer.

[4] G. Cantor. *Contributions to the Foundation of the Theory of Transfinite Numbers.* Dover (Reprint), New York, 1955.

[5] N. Cutland. *Computability, an Introduction to Recursive Function Theory.* Cambridge University Press, Cambridge, New York, 1980.

[6] X. Huang, M. Kerber, E. Melis, and J. Richts. Representing the diagonalization methods. In *Proc. of Workshop 13 at KI-94*, Saarbrücken, 1994.

[7] H.R. Lewis and C.H. Papadimitriou. *Elements of the Theory of Computation.* Prentice Hall, 1981.

[8] E. Melis. Change of representation in theorem proving by analogy. SEKI-Report SR-93-07, Universität des Saarlandes, Saarbrücken, 1993.

[9] E. Melis. Decomposition techniques and their applications. In *Proc. AIMSA'94*, Sofia, Bulgaria, 1994.

[10] E. Mendelson. *Introduction to Mathematical Logic.* Van Nostrand Company Inc., Princeton, 1964.

[11] S.C.Kleene. *Introduction to Metamethematics.* North Holland Publ Co., Amsterdam, 1952.

1. Terminate if there are no open goals.
2. For assumptions that are not establishable, base-level plan for the closest preceding goal or assumption found by restructuring.
3. If the source plan is exhausted, then base-level plan for the open goals.
4. Get next sequent P from the source plan. The sequent is either an assumption or a goal. If P is an assumption, then go to step 11.
5. If P is satisfied in target, then go to step 4.
6. If there is no reformulation ρ, such that ρP matches an open target goal or a current target assumption respectively, then
   - if restructuring possible, then
     - restructure
     - update source plan
     - go to step 4
   - else go to step 3
7. If there is such a reformulation ρ, then reformulate the source plan by ρ triggered by P. Link P to the source plan.
8. Select for the target the method M chosen in the source.
9. If M is verifiable, then
   - update open target goals
   - link M to the target
   - go to step 1
10. Restructure_and_8_or_base-level_plan_for_post(M)_and_4
11. Select for the target the f-method chosen in the source.
12. If there is no reformulation ρ' such that $|missing| < \theta$, for $missing$:= set of preconditions of ρ'M not matching a current target assumption, then
    - advance source plan.
    - restructure_and_4_or_base-level_plan_for_post(M)_and_4.
13. Reformulate the source plan by ρ' and triggered by the respective assumptions.
14. If M is verifiable, then
    - update target goals and assumptions.
    - link the matching target assumptions and M to the source plan.
    - go to step 1.

    .
15. Restructure_and_11_or_base-level_plan_for_post(M)_and_4.

Table 1: Outline of the analogy-driven proof-plan construction