A Set Theory Prover Within *Theorema*

Wolfgang.Windsteiger@RISC.Uni-Linz.ac.at
RISC Institute
Hagenberg, Austria.

Abstract. In this paper, we present the *Theorema* Set Theory Prover. This prover is designed for proving statements involving notions from set theory using natural deduction inference rules for set theory. Moreover, it applies the PCS paradigm (Proving-Computing-Solving) for generating natural proofs that has already been used in other provers in the *Theorema* system, notably the prover for automated proofs in elementary analysis. We show some applications of this prover in a case study on equivalence relations and partitions, which also nicely shows the interplay between proving, computing , and solving during an exploration of some mathematical theory.

1 Introduction

In this paper we report on the implementation of a Set Theory prover within the *Theorema* system. Particular emphasis has been put on the integration of set theory proving into the PCS heuristic for proving in predicate logic that has been invented by B. Buchberger, see [1], [2] or [3], and that has already been successfully applied to proofs in the area of elementary calculus using *Theorema* in [4]. It is shown how prove ("P"), compute ("C"), and solve ("S") phases contribute to natural style proofs of theorems involving notions of set theory. Opposed to an immediate approach of reducing set constructions to predicate logic "by definition" we provide special inference rules for all kinds of formulae involving sets in order to produce "elegant proofs". For instance, the proof problem of showing $A \subseteq B$ reduces by definition of "$\subseteq$" to showing $\forall_x x \in A \Rightarrow x \in B$, which we could then pass to the predicate logic prover. Instead, a human mathematician would probably just *assume* $x_0 \in A$ and *show* $x_0 \in B$ for some arbitrary x_0. This is the type of reasoning employed in the "P" phase by the *Theorema Set Theory PCS Prover*.

This work has been supported by the "Spezialforschungsbereich for Numerical and Symbolic Scientific Computing" (SFB F013) at the University of Linz, the "PROVE Project" supported by the regional government of Upper Austria, and the european union "CALCULEMUS Project" (HPRN-CT-2000-00102).

In addition, we try to incorporate the capabilities of the semantics of the *Theorema* language in order to deal with finitary set constructions during the compute (simplify) phase. The semantics of set operations available in *Theorema* computations not only allows computations with finite sets (such as {1, 2, 3}) but also with finite versions of the set quantifier (such as $\{x \in A \mid P_x\}$ where A is a finite set and P_x is some formula depending on x). We present an approach to *combining proving and computing* by doing semantic based computations during the "C" phase of proving in order to simplify proof situations.

Other constructions in set theory, however, directly lead to "solve problems". For example, proving $8 \in \{n^2 - 1 \mid n \in \mathbb{N}\}$ amounts to solving the equation $8 = n^2 - 1$ (over the integers). We present an approach to *combining proving and solving* by reducing typical "solve patterns" to "easier prove problems" using known solving techniques available in Mathematica.

It will be demonstrated in numerous examples, that the approach of viewing each proof as a sequence of *alternating proof phases* ("PCS") gives quite promising results in many applications. In this version of the paper, we give some examples of fully automatically generated proofs by *Theorema*, for more examples and a more detailed description of the prover, see [5].

2 The P-C-S Paradigm for *Theorema* Provers

Proving is seen (mainly) as a sequence of alternating phases of

- Proving
- Computing
- Solving

It is a *heuristic method* that aims on *automated proofs in natural style* trying to mimick what is actually done in proofs by humans. In the following sections, we will try to give some more details on the notions "Proving", "Computing", and "Solving", in particular, their interaction in the development of a generally applicable prove strategy.

2.1 Proving

Problem Specification (Proving):

Given: a collection of formulae A (the assumptions),
a formula G (the goal, the conclusion).

Find: P, such that:
if G is a consequence of A then P should contain a convincing argument (logically correct, sound, understandable), *why* G necessarily follows from A.

We call the P to be found in a proving problem a *proof of G (from A).*

Example

Goal: $\{1\} \subseteq \mathbb{N}$. *Assumptions*: Definition of $\subseteq$, $\mathbb{N}$, $\{\ldots\}$, and 1.

Proof: Take $a \in \{1\}$, which forces $a = 1$, hence $a \in \mathbb{N}$.

The Role of Proving Within PCS

The proving phase in a PCS-cycle is the phase where standard inference techniques from logic are applied in order to reduce the goal or infer new knowledge from known assumptions. Typically, these rules eliminate quantifiers, split up formulae, or even split up an entire proof inte several branches.

2.2 Computing

Problem Specification (Computing, Simplifying):

Given: a collection of formulae A (the assumptions),
a term T or a formula F.

Find: a term $\tilde{T}$ or a formula $\tilde{F}$, such that:

(a) $T = \tilde{T}$ is a consequence of A and
$\tilde{T}$ is (in some sense) simpler than T.

or

(b) $F \Leftrightarrow \tilde{F}$ is a consequence of A and
$\tilde{F}$ is (in some sense) simpler than F.

We name the *process* of finding $\tilde{T}$ or $\tilde{F}$ *computation* (*simplification*) *w.r.t. A*, and we call $\tilde{T}$ and $\tilde{F}$ respectively the *result of the computation w.r.t. A*.

Example

$$\text{Compute}: \left\{ x \mid_{x=1,\ldots,7} \text{is-prime}[x] \right\} \cap \left\{ x \mid_{x=1,\ldots,10} \text{Mod}[x, 5] \leq 3 \right\}.$$

Result : {2, 3, 7} or ??

$$\text{Compute}: 3 \in \left\{ x \mid_{x=1,\ldots,7} \text{is-prime}[x] \right\} \cap \left\{ x \mid_{x=1,\ldots,10} \text{Mod}[x, 5] \leq 3 \right\}.$$

Result : $3 \in \{2, 3, 7\}$ or True or ?? (depending on the available knowledge A).

The Role of Computing Within PCS

The computing phase in a PCS-cycle is typically the phase where terms or formulae, either in the goal or in the knowledge base, are simplified using knowledge from the semantics of the *Theorema* language. The *Theorema* language semantics covers logical connectives and the algorithmic language constructs for quantifiers, sets, and tuples. For details on the *Theorema* language and its semantics, we refer to [6] and [7].

2.3 Solving

Problem Specification (Solving):

Given : a formula G of the form $\underset{\underline{x}}{\exists} P_{\underline{x}}$,
a collection of formulae A (the assumptions),
such that G is a consequence of A.

Find : a proof of G by giving S, such that
$S \subseteq \{s \mid P_s$ can be derived from A and P_s is free of $\underline{x}\}$ and $|S| \geq 1$.

We call the *process* of finding S *solving* G *w.r.t.* A, each element of S is called a *solution for* $\underline{x}$ *of* G *w.r.t.* A, and we call S a *set of solutions for* $\underline{x}$ *of* G *w.r.t.* A.

(In above problem specification, $\underline{x}$ stands for a sequence of one or more variables, P_s denotes the result of applying the substituition s to the formula P, and a term or formula is called *free* of $\underline{x}$ if it contains none of the variables in $\underline{x}$. We assume, by the way, that the reader is fimiliar with the notion of substitution, we will use it in a standard way.)

Example

$$\underset{x,y\in\mathbb{R}}{\exists}\ \underbrace{x^2 + y^2 = 1}_{P_{x,y}},$$

and the assumptions are "the basic properties of the domain of real numbers" and some knowledge about trigonometric functions "sin" and "cos". *One solution* for x and y would be

$$S_1 = \{x \to 0,\ y \to 1\}$$

since $P_{\{x\to 0, y\to 1\}}$ gives $1 = 1$, which is of course true. Of course, also

$$S_2 = \left\{\{x \to 0,\ y \to 1\},\ \{x \to 1,\ x \to 0\},\ \left\{x \to \frac{\sqrt{2}}{2},\ y \to \frac{\sqrt{2}}{2}\right\},\ \left\{x \to \frac{\sqrt{2}}{2},\ y \to -\frac{\sqrt{2}}{2}\right\}\right\}$$

is a set of solutions for x and y of G according to our specification, as can be easily checked by applying the four substitutions to $P_{x,y}$ and performing basic arithmetic calculations. Another type of solution, which also meets the specification, is

$$S_3 = \{\{x \to \cos(t),\ y \to \sin(t)\} \mid 0 \le t < 2\,\pi\}$$

since $P_{\{x\to\cos(t), y\to\sin(t)\}}$ gives $\cos(t)^2 + \sin(t)^2 = 1$, which holds for every $0 \le t < 2\,\pi$. This type of solution set is typically called a *parametrized solution*.

The Role of Solving Within PCS

Solving comes into play during a PCS-cycle as soon as we have an existential goal to be proven. A well-known technique for handling existential quantifiers in the proof goal is the introduction of *solve constants*, sometimes also called *meta-variables*, see [8], used also in M. Beeson's system MathXPert, see [9], and in the *Theorema* PCS prover for elementary analysis, see [4]. Finding an appropriate value for a solve constant usually amounts to solve a system of equations or inequalities, where it is, of course, problem-dependent what type of solving technique becomes necessary.

3 Provers in *Theorema*

In general, *a Theorema Prover* is structured as shown in Figure 1:

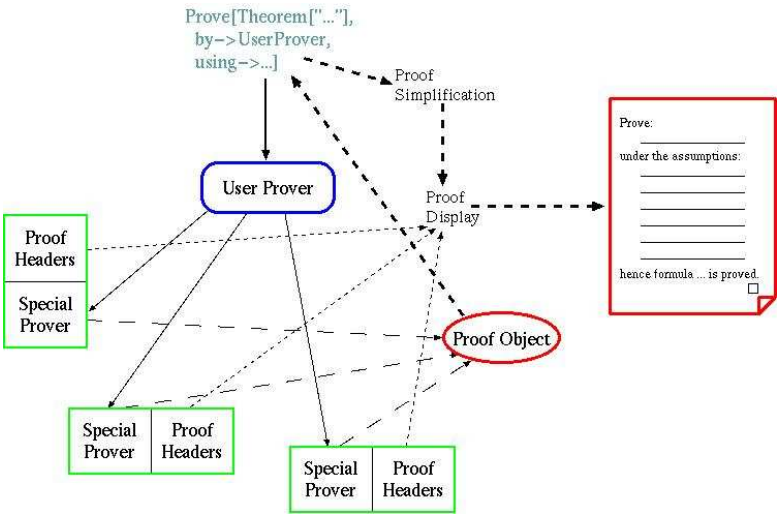

Figure 1. General structure of a *Theorema* prover.

4 The Set Theory PCS Prover

User Prover: The SetTheoryPCSProver combines:

Set theory specific special provers:

- **STP**: The **S**et **T**heory **P**rove module handles set theoretic expressions in the goal.
- **STKBR**: The **S**et **T**heory **K**nowledge**B**ase **R**ewriting module infers new knowledge from set theoretic expressions in the assumptions.
- **STC**: The **S**et **T**heory **C**ompute module performs (set theoretic) simplifications in the goal and in the assumptions.
- **STS**: The **S**et **T**heory **S**olve module tries to solve existential goals.
  In the context of set theory, such goals appear naturally e.g. in goals of the form $y \in \{T_x \mid \underset{x}{P_x}\}$, which immediately lead to $\underset{x}{\exists}\, y = T_x \wedge P_x$.

Other special provers involved from general PCS proving:

- Basic natural deduction.
- Recognition of terminal proof cases, see [5].
- Rewriting with respect to formulae in the knowledge base.

5 System Demonstration

5.1 General Outline of the Case Study

We investigate interactions of the following notions:

- Relations as sets of pairs.
- Equivalence relations and equivalence classes.
- The set of all equivalence classes in a set.
- Partitions of a set.
- The relation induced by a partition of a set.

5.2 Computing

The semantics of the *Theorema* language allows computations on certain expressions.

Definition["reflexivity", any[Rel, S],
reflexive$_S$[Rel] :⇔ $\forall_{x \in S}$ ⟨x, x⟩ ∈ Rel]

Definition["symmetry", any[Rel, S],
symmetric$_S$[Rel] :⇔ $\forall_{x,y \in S}$ (⟨x, y⟩ ∈ Rel ⇒ ⟨y, x⟩ ∈ Rel)]

Definition["Equivalence class", any[x, A, R],
class$_{A,R}$[x] := {a ∈ A | ⟨a, x⟩ ∈ R}]

Compute[reflexive$_{\{i \mid i=0,\ldots,10\}}$[{⟨i, j⟩ $_{i,j=0,\ldots,10}$ | (Mod[i, 5] = Mod[j, 5])}],
using → ⟨Definition["reflexivity"]⟩, built-in →
⟨**Built-in["Sets"][SetOf]**, Built-in["Quantifiers"], Built-in["Connectives"]⟩]

⟨0, 0⟩ ∈ {⟨0, 0⟩, ⟨0, 5⟩, ⟨0, 10⟩, ⟨1, 1⟩, ⟨1, 6⟩, ⟨2, 2⟩, ⟨2, 7⟩,
⟨3, 3⟩, ⟨3, 8⟩, ⟨4, 4⟩, ⟨4, 9⟩, ⟨5, 0⟩, ⟨5, 5⟩, ⟨5, 10⟩, ⟨6, 1⟩, ⟨6, 6⟩,
⟨7, 2⟩, ⟨7, 7⟩, ⟨8, 3⟩, ⟨8, 8⟩, ⟨9, 4⟩, ⟨9, 9⟩, ⟨10, 0⟩, ⟨10, 5⟩, ⟨10, 10⟩}
∧ ≪9≫ ∧
⟨10, 10⟩ ∈ {⟨0, 0⟩, ⟨0, 5⟩, ⟨0, 10⟩, ⟨1, 1⟩, ⟨1, 6⟩, ⟨2, 2⟩, ⟨2, 7⟩,
⟨3, 3⟩, ⟨3, 8⟩, ⟨4, 4⟩, ⟨4, 9⟩, ⟨5, 0⟩, ⟨5, 5⟩, ⟨5, 10⟩, ⟨6, 1⟩, ⟨6, 6⟩,
⟨7, 2⟩, ⟨7, 7⟩, ⟨8, 3⟩, ⟨8, 8⟩, ⟨9, 4⟩, ⟨9, 9⟩, ⟨10, 0⟩, ⟨10, 5⟩, ⟨10, 10⟩}

```
Compute[reflexive_{i | i=0,...,10}[{⟨i, j⟩ | i,j=0,...,10 (Mod[i, 5] = Mod[j, 5])}],
  using → ⟨Definition["reflexivity"]⟩,
  built-in → ⟨Built-in["Sets"], Built-in["Quantifiers"], Built-in["Connectives"]⟩]
```

True

```
Compute[class_{i | i=0,...,10},{⟨i,j⟩ | i,j=0,...,10 (Mod[i,5]=Mod[j,5])}[4],
  using → ⟨Definition["Equivalence class"]⟩,
  built-in → ⟨Built-in["Sets"], Built-in["Quantifiers"]⟩]
```

{4, 9}

```
Compute[class_{i | i=0,...,10},{⟨i,j⟩ | i,j=0,...,10 (Mod[i,5]=Mod[j,5])}[4] = {9, 4},
  using → ⟨Definition["Equivalence class"]⟩,
  built-in → ⟨Built-in["Sets"], Built-in["Quantifiers"]⟩]
```

True

5.3 Computing ⊆ Proving

Goal: Make use of knowledge provided by the *Theorema* language semantics during proving!

```
Proposition["R reflexive on finite set",
  reflexive_{i | i=0,...,10}[{⟨i, j⟩ | i,j=0,...,10 (Mod[i, 5] = Mod[j, 5])}]]
```

```
Prove[Proposition["R reflexive on finite set"],
  using → Definition["reflexivity"]]
```

Prove:

(Proposition (R refl)) $\text{reflexive}_{\{i \mid i=0,\ldots,10\}}\left[\left\{\langle i, j\rangle \mid \substack{i=0,\ldots,10 \\ j=0,\ldots,10} \text{Mod}[i, 5] = \text{Mod}[j, 5]\right\}\right]$,

under the assumption:

(Definition (reflexivity)) $\forall_{Rel,S}\left(\text{reflexive}_S[Rel] :\Leftrightarrow \forall_{x\in S}(\langle x, x\rangle \in Rel)\right)$.

Using available computation rules we simplify (Proposition (R refl)):

(1) $\text{reflexive}_{\{0,1,2,3,4,5,6,7,8,9,10\}}[\{\langle 0, 0\rangle, \langle 0, 5\rangle, \langle 0, 10\rangle, \ll 19 \gg, \langle 10, 0\rangle, \langle 10, 5\rangle, \langle 10, 10\rangle\}]$.

Formula (1), using (Definition (reflexivity)), is implied by:

(2) $\forall_{x\in\{0,1,2,3,4,5,6,7,8,9,10\}}(\langle x, x\rangle \in \{\langle 0, 0\rangle, \langle 0, 5\rangle, \langle 0, 10\rangle, \ll 19 \gg, \langle 10, 0\rangle, \langle 10, 5\rangle, \langle 10, 10\rangle\})$

.

Using available computation rules we simplify (2):

(3) True.

Formula (3) is true because it is the constant True.

```
Proposition["R symmetric on finite set",
  symmetric_{i | i=0,...,10}[{⟨i, j⟩ | i,j=0,...,10 (Mod[i, 5] = Mod[j, 5])}]]
```

```
Prove[Proposition["R symmetric on finite set"],
  using → Definition["symmetry"],
  built-in → ⟨Built-in["Quantifiers"], Built-in["Tuples"]⟩]
```

Prove:

(Proposition (R symm)) $\text{symmetric}_{\{i \mid i=0,\ldots,10\}}\left[\left\{\langle i, j\rangle \mid \substack{i=0,\ldots,10 \\ j=0,\ldots,10} \text{Mod}[i, 5] = \text{Mod}[j, 5]\right\}\right]$,

under the assumption:

(Definition (symmetry)) $\forall_{Rel,S}\left(\text{symmetric}_S[Rel] :\Leftrightarrow \forall_{\substack{x\in S \\ y\in S}} (\langle x, y\rangle \in Rel \Rightarrow \langle y, x\rangle \in Rel)\right)$.

Using available computation rules we simplify (Proposition (R symm)):

(1) $\text{symmetric}_{\{0,1,2,3,4,5,6,7,8,9,10\}}[\{\langle 0, 0\rangle, \langle 0, 5\rangle, \langle 0, 10\rangle, \ll 19 \gg, \langle 10, 0\rangle, \langle 10, 5\rangle, \langle 10, 10\rangle\}]$

.

Formula (1), using (Definition (symmetry)), is implied by:

(2)

$$\forall_{\substack{x\in\{0,1,2,3,4,5,6,7,8,9,10\} \\ y\in\{0,1,2,3,4,5,6,7,8,9,10\}}} (\langle x, y\rangle \in \{\langle 0, 0\rangle, \langle 0, 5\rangle, \langle 0, 10\rangle, \ll 19 \gg, \langle 10, 0\rangle, \langle 10, 5\rangle, \langle 10, 10\rangle\} \Rightarrow .$$
$$\langle y, x\rangle \in \{\langle 0, 0\rangle, \langle 0, 5\rangle, \langle 0, 10\rangle, \ll 19 \gg, \langle 10, 0\rangle, \langle 10, 5\rangle, \langle 10, 10\rangle\})$$

Using available computation rules we simplify (2):

(3) True.

Formula (3) is true because it is the constant True.

5.4 Computing ⊂ Proving

The *Theorema* language does not provide semantics for infinite ranges in quantifiers, thus, when switching to expressions of that form we cannot anymore reduce proving to simple (finite) computations.

```
Compute[reflexive_ℕ[{⟨i, j⟩ | i,j∈ℕ (Mod[i, 5] = Mod[j, 5])}],
  using → ⟨Definition["reflexivity"]⟩]
```

$$\forall_{x\in\mathbb{N}}\left(\langle x, x\rangle \in \left\{\langle i, j\rangle \mid \substack{i\in\mathbb{N} \\ j\in\mathbb{N}} \text{Mod}[i, 5] = \text{Mod}[j, 5]\right\}\right)$$

```
Compute[class_{ℕ,{⟨i,j⟩ | i,j∈ℕ (Mod[i,5]=Mod[j,5])}}[4] = {4 + 5 k | k∈ℕ},
  using → ⟨Definition["Equivalence class"]⟩]
{a | a∈ℕ ⟨a, 4⟩ ∈ {⟨i, j⟩ | i∈ℕ, j∈ℕ Mod[i, 5] = Mod[j, 5]}} = {4 + 5 ∗ k | k∈ℕ}
```

5.5 Solving ⊆ Proving

Goal: Make use of available algebraic solution techniques during proving!

```
Proposition["R reflexive on ℕ",
  reflexive_ℕ[{⟨i, j⟩ | i,j∈ℕ (Mod[i, 5] = Mod[j, 5])}]]
```

```
Prove[Proposition["R reflexive on ℕ"],
  using → Definition["reflexivity"], built-in → Built-in["Tuples"]]
```

Prove:

(Proposition (R refl)) $\text{reflexive}_{\mathbb{N}}\left[\{\langle i, j\rangle \mid_{i,j} (i \in \mathbb{N} \wedge (\text{Mod}[i, 5] = \text{Mod}[j, 5])) \wedge j \in \mathbb{N}\}\right]$,

under the assumption:

(Definition (reflexivity)) $\forall_{Rel,S} \left(\text{reflexive}_S[Rel] :\Leftrightarrow \forall_x (x \in S \Rightarrow \langle x, x\rangle \in Rel)\right)$.

Formula (Proposition (R refl)), using (Definition (reflexivity)), is implied by:

(1) $\forall_x \left(x \in \mathbb{N} \Rightarrow \langle x, x\rangle \in \{\langle i, j\rangle \mid_{i,j} i \in \mathbb{N} \wedge j \in \mathbb{N} \wedge (\text{Mod}[i, 5] = \text{Mod}[j, 5])\}\right)$.

We assume

(2) $x_0 \in \mathbb{N}$,

and show

(3) $\langle x_0, x_0\rangle \in \{\langle i, j\rangle \mid_{i,j} i \in \mathbb{N} \wedge j \in \mathbb{N} \wedge (\text{Mod}[i, 5] = \text{Mod}[j, 5])\}$.

In order to prove (3) we have to show

(4) $\exists_{i,j} ((i \in \mathbb{N} \wedge j \in \mathbb{N} \wedge (\text{Mod}[i, 5] = \text{Mod}[j, 5])) \wedge (\langle x_0, x_0\rangle = \langle i, j\rangle))$.

Since $i := x_0$ and $j := x_0$ solves the equational part of (4) it suffices to show

(5) $x_0 \in \mathbb{N} \wedge x_0 \in \mathbb{N} \wedge (\text{Mod}[x_0, 5] = \text{Mod}[x_0, 5])$.

We prove the individual conjunctive parts of (5):

Proof of (5.1) $x_0 \in \mathbb{N}$:

Formula (5.1) is true because it is identical to (2).

Proof of (5.2) $x_0 \in \mathbb{N}$:

Formula (5.2) is true because it is identical to (2).

Proof of (5.3) $\text{Mod}[x_0, 5] = \text{Mod}[x_0, 5]$:

It can be easily verified that formula (5.3) always holds.

5.6 Computing ⋃ Solving ⊆ Proving

Due to space limitations, we *only* give *one example* of the interplay between proving, computing, and solving. For more examples we refer to the technical report [10], for a full exploration of equivalence relations, partitions, etc. with a proof also for the auxiliary proposition, see [5].

```
Proposition["induced by partition is transitive", any[P], with[is-partition[P]],
  is-transitive[induced-relation[P]]]
```

One auxiliary property is needed in the proof:

```
Proposition["⋂ =", any[P], with[is-partition[P]],
  ∀_{X,Y∈P} (X ⋂ Y ≠ ∅ ⇒ (X = Y))]
```

```
Prove[Proposition["induced by partition is transitive"],
  using →
    ⟨Definition["transitivity"], Definition["induced relation"], Proposition["⋂ ="]⟩]
```

Prove:

(Proposition (ind is trans)) $\forall_{P} (\text{is-partition}[P] \Rightarrow \text{is-transitive}[\text{induced-relation}[P]])$,

under the assumptions:

(Definition (trans)) $\forall_{R} \left(\text{is-transitive}[R] :\Leftrightarrow \forall_{x,y,z} (\langle x, y\rangle \in R \wedge \langle y, z\rangle \in R \Rightarrow \langle x, z\rangle \in R)\right)$,

(Definition (ind relation)) $\forall_{S} \left(\text{induced-relation}[S] :\Leftrightarrow \left\{\langle x, y\rangle \;\middle|\; \exists_{M\in S} (x \in M \wedge y \in M)\right\}\right)$,

(Proposition (⋂ =)) $\forall_{P} \left(\text{is-partition}[P] \Rightarrow \forall_{X,Y} (X \in P \wedge Y \in P \Rightarrow ((X \cap Y \neq \{\}) \Rightarrow (X = Y)))\right)$.

We assume

(1) $\text{is-partition}[P_0]$,

and show

(2) $\text{is-transitive}[\text{induced-relation}[P_0]]$.

Formula (2), using (Definition (trans)), is implied by:

(3) $\forall_{x,y,z} (\langle x, y\rangle \in \text{induced-relation}[P_0] \wedge \langle y, z\rangle \in \text{induced-relation}[P_0] \Rightarrow \langle x, z\rangle \in \text{induced-relation}[P_0])$.

We assume

(4) $\langle x_0, y_0\rangle \in \text{induced-relation}[P_0] \wedge \langle y_0, z_0\rangle \in \text{induced-relation}[P_0]$,

and show

(5) $\langle x_0, z_0\rangle \in \text{induced-relation}[P_0]$.

From what we already know follows:

From (4.1) we can infer

(6) $\text{induced-relation}[P_0] \neq \{\}$.

Formula (5), using (Definition (ind relation)), is implied by:

(8) $\langle x_0, z_0\rangle \in \left\{\langle x, y\rangle \;\middle|\; \underset{x,y}{\exists}\, \underset{M\in P_0}{\exists}\, (x \in M \wedge y \in M)\right\}.$

In order to prove (8) we have to show

(9) $\underset{x,y}{\exists}\left(\underset{M}{\exists}\, (M \in P_0 \wedge (x \in M \wedge y \in M)) \bigwedge (\langle x_0, z_0\rangle = \langle x, y\rangle)\right).$

Since $x := x_0$ and $y := z_0$ solves the equational part of (9) it suffices to show

(10) $\underset{M}{\exists}\, (M \in P_0 \wedge (x_0 \in M \wedge z_0 \in M)).$

Formula (6), by (Definition (ind relation)), implies:

(11) $\left\{\langle x, y\rangle \;\middle|\; \underset{x,y}{\exists}\, \underset{M\in P_0}{\exists}\, (x \in M \wedge y \in M)\right\} \neq \{\}.$

We did not find any ground formula to match a part of (10).

Formula (4.1), by (Definition (ind relation)), implies:

(12) $\langle x_0, y_0\rangle \in \left\{\langle x, y\rangle \;\middle|\; \underset{x,y}{\exists}\, \underset{M\in P_0}{\exists}\, (x \in M \wedge y \in M)\right\}.$

From what we already know follows:

From (12) we know by definition of $\left\{T_{\underset{x}{}} \mid P\right\}$ that we can choose an appropriate value such that

(13) $\underset{M}{\exists}\, (M \in P_0 \wedge (x1_0 \in M \wedge x2_0 \in M)),$

(14) $\langle x_0, y_0\rangle = \langle x1_0, x2_0\rangle.$

Using available computation rules we can simplify the knowledge base:

Formula (13) simplifies to

(15) $\underset{M}{\exists}\, (M \in P_0 \wedge x1_0 \in M \wedge x2_0 \in M),$

Formula (14) simplifies to

(16) $(x_0 = x1_0) \wedge (y_0 = x2_0),$

By (15) we can take appropriate values such that:

(17) $M_0 \in P_0 \wedge x1_0 \in M_0 \wedge x2_0 \in M_0.$

From what we already know follows:

From (17.1) we can infer

(18) $P_0 \neq \{\}.$

From (17.2) we can infer

(19) $M_0 \neq \{\}.$

Because parts of the knowledge base match a part of (10), we try to find an instance of (10).

Now, let $M := M_0$. Thus, for proving (10) it is sufficient to prove:

(21) $M_0 \in P_0 \wedge (x_0 \in M_0 \wedge z_0 \in M_0).$

We prove the individual conjunctive parts of (21):

Proof of (21.1) $M_0 \in P_0$:

Formula (21.1) is true because it is identical to (17.1).

Proof of (21.2) $x_0 \in M_0 \wedge z_0 \in M_0$:

We prove the individual conjunctive parts of (21.2):

Proof of (21.2.1) $x_0 \in M_0$:

Formula (21.2.1), using (16.1), is implied by:

(22) $x1_0 \in M_0.$

Formula (22) is true because it is identical to (17.2).

Proof of (21.2.2) $z_0 \in M_0$:

Formula (4.2), by (16.2), implies:

$\langle x2_0, z_0 \rangle \in \text{induced-relation}[P_0]$,

which, by (Definition (ind relation)), implies:

(23) $\langle x2_0, z_0 \rangle \in \left\{ \langle x, y \rangle \;\middle|\; \underset{x,y}{} \underset{M \in P_0}{\exists} (x \in M \wedge y \in M) \right\}$.

From what we already know follows:

From (23) we know by definition of $\left\{ T_x \mid \underset{x}{P} \right\}$ that we can choose an appropriate value such that

(24) $\underset{M}{\exists} (M \in P_0 \wedge (x3_0 \in M \wedge x4_0 \in M))$,

(25) $\langle x2_0, z_0 \rangle = \langle x3_0, x4_0 \rangle$.

Using available computation rules we can simplify the knowledge base:

Formula (24) simplifies to

(26) $\underset{M}{\exists} (M \in P_0 \wedge x3_0 \in M \wedge x4_0 \in M)$,

Formula (25) simplifies to

(27) $(x2_0 = x3_0) \wedge (z_0 = x4_0)$,

By (26) we can take appropriate values such that:

(28) $M_1 \in P_0 \wedge x3_0 \in M_1 \wedge x4_0 \in M_1$.

From what we already know follows:

From (28.2) we can infer

(30) $M_1 \neq \{\}$.

Formula (21.2.2), using (27.2), is implied by:

(32) $x4_0 \in M_0$.

Formula (17.3), by (27.1), implies:

(33) $x3_0 \in M_0$.

From what we already know follows:

From (28.2) together with (33) we know

(35) $x3_0 \in M_1 \bigcap M_0$.

From what we already know follows:

From (35) we can infer

(36) $M_1 \bigcap M_0 \neq \{\}$.

Formula (36), by (Proposition ($\bigcap$ =)), implies:

(37) $\underset{P}{\forall}$ (is-partition$[P] \wedge M_0 \in P \wedge M_1 \in P \Rightarrow (M_1 = M_0)$).

Formula (17.1), by (37), implies:

(71) is-partition$[P_0] \Rightarrow (M_1 \in P_0 \Rightarrow (M_1 = M_0))$.

From (1) and (71) we obtain by modus ponens

(72) $M_1 \in P_0 \Rightarrow (M_1 = M_0)$.

From (28.1) and (72) we obtain by modus ponens

(73) $M_1 = M_0$.

Formula (32) is true because of (28.3) and (73).

Comments on the Method

The proof starts with a general "P" (prove) phase, which structures the proof according to the structure of the goal and the knowledge base. Natural deduction inference rules are used to eliminate universal quantifier(s) and to decompose compound formulae in the goal and in the knowledge base. As soon as one is left with atomic formulae (e.g. formula (2) and (5)) the "P" phase is terminated and a "C" (compute, simplify) phase is initiated, which uses particular formulae in the knowledge base in order to rewrite the goal or other formulae in the knowledge base. In this phase, definitions for function and predicate symbols are applied to goal and knowledge. After some cycles of "P" and "C" phases we arrive at formula (8), which then initiates a special "P" phase for handling formulae whose outermost symbol is a set operator. In this example, the goal is of the form $t \in \left\{ T_x \mid \underset{x}{c} \right\}$, which, "by definition", introduces an existential quantifier. Note however, that the definition of this variant of the set constructor is applied as an inference rule in the special set theory "P" phase, and it need not be given as a definition in the knowledge base.

Formula (9) is an existential goal, which in many situations describes a proof problem that can be handled by *solving* the quantified formula (or parts of it) with respect to the quantified variables. This is done now in an "S" (solve) phase. It of course depends on the structure of the formula, which solving technique is appropriate, but all available powerful black-box solving techniques (Gauss method, Gröbner bases method, Collins' algorithm) are natural candidates to be applied in this step. In the example, one part of the formula is a tuple equation, which, after transforming it into a *system of individual equations*, can be solved by the standard Mathematica "Solve" command. We then continue to proof the remaining part of formula (9). Formula (10) is still an existential goal, but this time, due to the structure of the quantified formula, no solve technique is available, and we therefore suspend the "S" phase.

The steps from formula (12) until (17) are again cycles of "P" and "C", where (13) and (14) are derived by using definitions built into the prover, and, by application of *Theorema* language semantics of logical connectives and tuples, simplified to (15) and (16), repectively. Having new knowledge in the knowledge base, we now re-enter the "S" phase and we solve the existential goal (10) by basic predicate logic solving, i.e. we try to instanciate (10) guided by available formulae in the knowledge base. We now leave the "S" phase and continue in a "P"-"C" cycles. By a special inference technique for set theory, we infer formula (35): if we know $x \in A$ and also $x \in B$ then we can infer $x \in A \bigcap B$! Formula (35) is now the key to make use of the auxiliary property about partitions given in the knowledge base and the proof quickly succeeds.

6 Conclusion

Two main theorems in the area of equivalence relations, induced relations, and partitions

Theorem["induced by factor set of R is R itself", any[R], with[is-equivalence[R]], induced-relation[factor-set[R]] = R]

Theorem["factor set of induced by P is P itself", any[P], with[is-partition[P]], factor-set[induced-relation[P]] = P]

together with all auxiliary propositions and lemmata starting from the definitions have been proved completely automatized using the Set Theory PCS Prover. After this case study we think that structuring proving into alternating cycles of P-C-S

- is a feasible approach for structuring automated theorem provers,
- gives naturally structured proofs, AND MOST IMPORTANTLY
- it opens the door for incorporating powerful known solving and computing techniques from the field of Computer-Algebra (Gröbner bases, Collins' algorithm, Gauss method, Lazy Narrowing, etc.) into automated theorem provers.

References

1. B. Buchberger. Theorema: A Proving System Based on Mathematica. International Symposium on Symbolic and Numeric Scientific Computing, RISC, Hagenberg, Austria, 1999.
2. B. Buchberger. Theory Exploration Versus Theorem Proving. In A. Armando and T. Jebelean, editor, *Electronic Notes in Theoretical Computer Science*, volume 23–3 of pages 67-69, Elsevier, 1999. CALCULEMUS Workshop, University of Trento, Trento, Italy.
3. B. Buchberger and C. Duprè and T. Jebelean and K. Kriftner and K. Nakagawa and D. Vasaru and W. Windsteiger. The Theorema Project: A Progress Report. In M. Kerber and M. Kohlhase, editor, *Proceedings of the 8th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, St. Andrews, Scotland, August 6–7* pages 100-115, 2000.
4. D. Vasaru–Duprè. *Automated Theorem Proving by Integrating Proving, Solving and Computing*. RISC Institute, May 2000. RISC report 00–19.
5. Wolfgang Windsteiger. *A Set Theory Prover in Theorema: Implementation and Practical Applications.* RISC Institute, May 2001.
6. B. Buchberger. Mathematics: An Introduction to Mathematics Integrating the Pure and Algorithmic Aspect. Volume I: A Logical Basis for Mathematics. Lecture notes for the mathematics course in the first and second semester at the Fachhochschule for Software Engineering in Hagenberg, Austria, 1996.
7. F. Kriftner. Theorema: The Language. In B. Buchberger and T. Jebelean, editor, *Proceedings of the Second International Theorema Workshop*, pages 39-54, 1998. RISC report 98–10.
8. E. Melis. The Limit Domain. In *4th International Conference on Artificial Intelligence Proof Planning*, American Association for Artificial Intelligence. 1998.
9. M. Beeson. Design Principles of Mathpert: Software to Support Education in Algebra and Calculus. In N. Kajler, editor, *Human Interfaces to Symbolic Computation*, Springer–Verlag, Berlin/Heidelberg/New York, 1996.
10. W. Windsteiger. A Set Theory Prover in Theorema. RISC Series No. 01–07,RISC, February 2001. Talk given at the CAL'01 workshop, Las Palmas, Gran Canaria, February 20, 2001.