JOHANNES KEPLER
UNIVERSITÄT LINZ | JKU

TNF

Technisch-Naturwissenschaftliche
Fakultät

An automated induction prover for finite sets implemented in the Theorema system

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplomingenieur

im Masterstudium

Computermathematik

Eingereicht von:
Dietmar Kerbl

Angefertigt am:
Research Institute for Symbolic Computation (RISC)

Beurteilung:
A.Univ.-Prof. Dr. Tudor Jebelean

Betreuung:
DI Dr. Wolfgang Windsteiger

Linz, Oktober 2010

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Linz, 2010

Dietmar Kerbl

Abstract

In this thesis we introduce an extension of the *Theorema* set theory prover. We implement a prover in *Theorema* that is able to automatically prove correctness properties about recursive algorithms dealing with finite sets. This prover contains inference rules for setting up an induction over the cardinality of finite sets and rules dealing with the *an arbitrary element of a set* language construct in *Theorema.*

We discuss various inference rules and present five case studies demonstrating the new prover and its interaction with the existing *Theorema* system, in particular the set theory prover.

Zusammenfassung

In dieser Arbeit stellen wir eine Erweiterung des Beweisers für Mengenlehre in *Theorema* vor. Wir implementieren einen Beweiser in *Theorema,* der Korrektheitsaussagen über rekursive Algorithmen, die mit endlichen Mengen rechnen, automatisch beweist. Der Beweiser beinhaltet die Beweisregeln für eine Induktion über die Anzahl der Elemente einer endlichen Menge und Beweisregeln für das *Theorema* Sprachkonstrukt *ein beliebiges Element einer Menge*.

Wir diskutieren einige der Beweisregeln und präsentieren fünf Fallbeispiele, die den neuen Beweiser und seine Interaktion mit dem existierenden *Theorema* System, insbesondere dem Beweiser für Mengenlehre, demonstrieren.

Acknowledgment

I deeply want to thank DI Dr. Wolfgang Windsteiger for all his advice and help during the development of this thesis. I especially want to thank him for helping me with the internals of *Mathematica* and *Theorema.*

A.Univ.-Prof. Dr. Tudor Jebelean piqued my interest in logic and automatic reasoning with his interesting and excellent lectures, thanks for that.

Thanks also to the whole *Theorema* group for the valuable discussions during the seminar and other lectures.

Beyond the university I would like to thank my family and friends for their valuable support over the years. In particular I want to thank my parents for their patience and understanding.

Table of contents

Chapter 1: Introduction

1.1 Motivation

In order to do computation in set theory with finite sets, we often use recursion. A class of objects or methods exhibit recursive behavior when they can be defined by the two following properties:

1. A simple base case.
2. A set of rules which reduce all other cases towards the base case.

Example: One's children are one's descendants (base case).

The children of one's descendants are also one's descendants (recursion step).

Many mathematical axioms are based upon recursive rules. For example, the formal definition of the natural numbers in set theory:

0 is a natural number, and each natural number has a successor, which is also a natural number. By this base case and the recursive rule, one can generate the set of all natural numbers. Finite sets do exhibit recursive behavior. For example, every subset of a finite set is again finite, e.g. if A is a finite set, $B \subseteq A$ and particularly $C = A \setminus \{a\}$ with $a \in A$, are again finite. We want to use this paradigm in algorithms for computation with finite sets.

Example:

Given: A finite set A

Compute: All the two element subsets of A

We provide the following algorithm: (For more details on this algorithm, see Section 3.2)

In the base case we start with the empty set. It has no two element subsets. In the recursion we first pick out one arbitrary element of our given set A. This is done by the *Theorema* language construct æ. If the set A is given, æ[A] gives us one arbitrary element in A.

We then compute the union of :

1. all 2-element subsets containing æ[*A*]
2. the 2-element subsets of $A \setminus \{æ[A]\}$

```
Definition["S2", any[A],
  S2[{}] = {}                                                    "base"
  A ≠ {} ⇒ (S2[A] =                                              "rec" ]
        where[a = æ[A], {{a, b} | b ∈ A \ {a}} ⋃ S2[A \ {a}]])
```

The *Theorema* system gives us the possibility to define such an algorithm (see Section 2.1). We are able to use this definition for computation and later in particular for proving.

Computation in *Theorema* can be performed with the *Compute* command.

```
Compute[expression, using → ⟨Definition["..."]⟩, built–in → ⟨Built–in["..."], Built–in["..."]⟩
```

To do computation in *Theorema,* we give as a first parameter of the *Compute* command the *expression* we want to compute. The parameter *using→K* defines the knowledge *K* we want to use, e.g. the definitions of functions occurring in *expression.* With the parameter *built–in→B* one can use some *Theorema* built-ins. Those built–ins contain pre defined knowledge about mathematical objects that are available in *Theorema*, for example natural numbers, sets, logical connectives and others.

In this case we use our algorithm and beyond that for example the built–in for sets. We use these built–in in order to be able to compute the set difference _ \ _, the union _ ⋃ _ and the set quantifier {_ | _}.

Examples of computations:

```
Compute[S2[{1, 2, 3}], using → ⟨Definition["S2"]⟩,
  built–in → ⟨Built–in["Sets"], Built–in["Numbers"], Built–in["Connectives"], Built–in["Quantifiers"]⟩]
{{1, 2}, {1, 3}, {2, 3}}

Compute[S2[{1, 2, 3, 4, 5, 6}], using → ⟨Definition["S2"]⟩,
  built–in → ⟨Built–in["Sets"], Built–in["Numbers"], Built–in["Connectives"], Built–in["Quantifiers"]⟩]
{{1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}, {2, 3}, {2, 4},
 {2, 5}, {2, 6}, {3, 4}, {3, 5}, {3, 6}, {4, 5}, {4, 6}, {5, 6}}
```

```
Compute[S2[{1}], using → ⟨Definition["S2"]⟩,
  built–in → ⟨Built–in["Sets"], Built–in["Numbers"], Built–in["Connectives"], Built–in["Quantifiers"]⟩]
{}

Compute[S2[{}], using → ⟨Definition["S2"]⟩,
  built–in → ⟨Built–in["Sets"], Built–in["Numbers"], Built–in["Connectives"], Built–in["Quantifiers"]⟩]
{}
```

This algorithm and the examples above give rise to the following questions:

1.) Does this algorithm actually compute all the two element subsets of a given finite set, e.g. is our algorithm partially correct?

2.) How many such two element subsets are there for a given finite set with n elements?

ad 1.) Given a finite set A we would typically define the set of all two element subsets of A *S2Def*$[A]$ as follows:

$$\underset{\text{is–finite}[A]}{\forall} \text{S2Def}[A] = \{B \in \mathcal{P}[A] \mid (|B| = 2)\}$$

So what we actually want to prove is the following:

$$\underset{\text{is–finite}[A]}{\forall} \text{S2}[A] = \{B \in \mathcal{P}[A] \mid (|B| = 2)\}$$

We want to prove a formula, that is universally quantified over all possible finite sets A.

ad 2.) The examples above give rise to the conjecture, that our algorithm delivers exactly $\binom{|A|}{2} = \frac{|A|\cdot(|A|-1)}{2}$ such two element subsets. This cardinality of the set of all two element subsets is of course well known from combinatorics. But what we actually want to prove is the following:

$$\underset{\text{is–finite}[A]}{\forall} |\text{S2}[A]| = \frac{|A|\cdot(|A|-1)}{2}$$

This is again a formula universally quantified over all possible finite sets.

Note

The approach in this thesis will be the following: We start with a given recursive algorithm and then try to

automatically prove various properties about it, e.g. correctness properties. Properties that speak about recursively defined objects are often proven by induction. We provide a special kind of induction, namely induction over the cardinality of finite sets to deal with such problems. For details see Section 2.3.

There is also the possibility to go somewhat the other way round, namely *algorithm synthesis.* One method of synthesis corresponds to the analysis of failing proofs and is called the *lazy thinking paradigm.* For a given predicate logic specification of the problem in terms of a set of operations (functions and predicates), the method produces an algorithm that solves the problem together with a correctness proof for the algorithm. In the ideal case, the only information that has to be provided by the user consists of the formal problem specification, an algorithm scheme and a complete knowledge base for the operations that occur in the problem specification, for details see [Buch03], [Buch04].

1.2 Goals and structure

In this thesis we want to generate automated proofs for the questions presented in the previous section. Given a recursive algorithm acting on finite sets, we want to automatically prove:

- Partial correctness properties
- Cardinality properties

Note: Such formulae are typically universally quantified over all possible finite sets, i.e. they have the form $\forall_{\text{is–finite}[A]} \phi_A$.

Our goal is to implement a new special prover in *Theorema,* which is able to automatically prove the described properties. We call this new prover *set induction prover*, SIP. Our strategy to prove such properties will be to prove them by induction over the cardinality of the set A. For details see Section 2.3.

The new special prover SIP should be able to set up the induction over the cardinality of finite sets and provide appropriate inference rules for the language construct æ. In Chapter 2 we introduce the new special prover SIP and discuss it in detail.

Chapter 2: The set induction prover

In this section we want to present the set induction prover, the new special prover for *Theorema*, in detail. We discuss the *Theorema* system shortly, our prover design, the implemented induction principle, the language construct æ and the inference rules.

2.1 The Theorema system

We only want to give the most important information about the system. For detailed explanations and a general outline on the system we refer to [Tma97], [Tma00a], [Tma00b], [Tma00c], [Tma06].

The *Theorema* system is a system for proving, solving, and simplifying mathematical formulae. It is programmed in the *Mathematica* language and also heavily uses the front-end of *Mathematica. Theorema* supports the entire process of mathematical exploration within one coherent logic and software system.

Every action in *Theorema* is performed in the following way:

Action[*entity*, using → *K*, by → *R*, *options*]

Action means the desired action, namely Compute, Prove or Solve. *Entity* means the mathematical entity to which the action should apply, e.g. an expression in the case of computing or a theorem in the case of proving. *K* defines the knowledge we want to use when we perform our action. *R* is the reasoner we want to use. *Options* are possible options to be given to the reasoner in order to influence its behavior.

Theorema is being developed by the TH∃OREM∀ Working Group at the RISC institute under the direction of Bruno Buchberger.

2.2 Prover design

We implement a new special prover in *Theorema,* called SIP, set induction prover.

- It contains the inference rules for induction over the cardinality of a finite set.
- It contains the inference rules for the language construct æ.
- It contains some other inference rules (see Section 2.5.4).

The *Theorema* system consists of special provers and user provers.

If one works with *Theorema* and calls a prover, what one means is a user prover. It is actually a bunch of components working together, e.g. several special provers. In *Fig.1* below one can see this interplay in more detail:

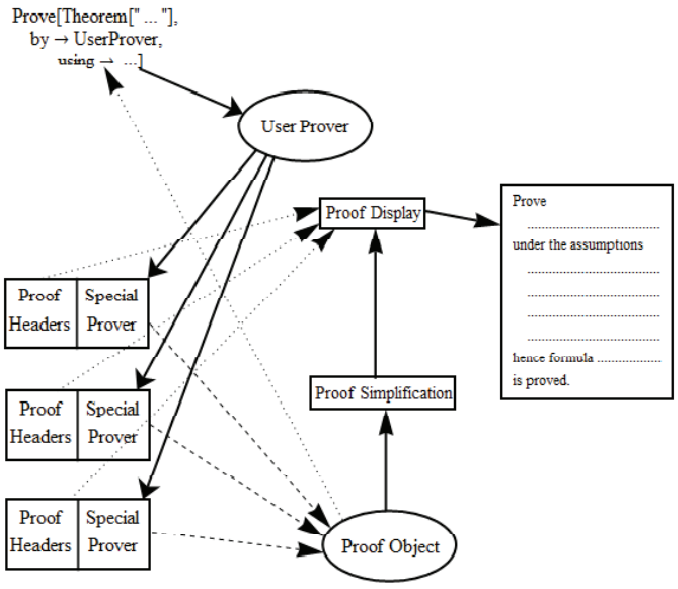

Fig .1 : Interplay between prover components

User prover

When a proof call is performed with the *Theorema* function Prove, a user prover is called. The user prover defines which special provers are used and how they are working together.

Special prover

The component, that includes the inference rules. Those inference rules produce a new proof situation from the current proof situation. One inference rule at a time is applied. Those special provers together generate the proof object. This proof object optionally runs through the proof simplification, in order to simplify the proof object (e.g. delete unsuccessful branches of the proof tree). Together with the proof headers a nice readable proof text is generated, the proof display.

Proof headers

The proof headers generate the nice readable proof text. They are independent and strictly separated from the special provers.

We integrate the new special prover SIP into the existing user prover SetTheoryPCSProver. See [WW01]. As its name tells, the SetTheoryPCSProver user prover deals with set theory. The prover applies a Proving-Computing-Solving strategy.

- Standard inference techniques from predicate logic (proving).
- Computing, rewriting, using the knowledge base (simplifying).
- Computer algebra methods for solving, e.g. existentially quantified formulae.

2.3 Induction over the cardinality of finite sets

For classical mathematical induction we use the so called *Axiom of induction*, which states for all formulae ϕ_n (with free variable n):

$$\phi_{0\leftarrow n} \bigwedge \left(\forall_{n\in\mathbb{N}} \; (\phi_n \Rightarrow \phi_{n+1\leftarrow n}) \right) \Rightarrow \forall_{n\in\mathbb{N}} \; \phi_n$$

where $\phi_{t\leftarrow v}$ is standard substitution form predicate logic, i.e. every free occurrence of the variable v in ϕ is substituted by t.

So if we want to prove: $\forall_{n\in\mathbb{N}} \; \phi_n$

We prove :

1. Induction base

Prove: $\phi_{0\leftarrow n}$

2. Induction step

Assume: $n_0 \in \mathbb{N} \wedge \phi_{n_0\leftarrow n}$ where n_0 is a new constant

Prove: $\phi_{n_0+1\leftarrow n}$

If we want to prove properties about recursive algorithms acting on finite sets, we want to prove formulae that are universally quantified over all finite sets, i.e.

$$\forall_{\text{is–finite}[A]} \phi_A ,$$

where ϕ_A denotes a formula with free variable A and the predicate *is–finite* means

$$\forall_A (\text{is–finite}[A] \Leftrightarrow \exists_{n\in\mathbb{N}} (|A| = n)) .$$

So what we want to prove is

$$\forall_A \left(\exists_{n\in\mathbb{N}} (|A| = n) \right) \Rightarrow \phi_A \quad \Leftrightarrow \quad \text{We change the implication into a disjunction.}$$

$$\forall_A \neg\left(\exists_{n\in\mathbb{N}} (|A| = n) \right) \vee \phi_A \quad \Leftrightarrow \quad \text{We simplify the negation.}$$

$$\forall_{A\,n\in\mathbb{N}} \neg(|A| = n) \vee \phi_A \quad \Leftrightarrow \quad \text{We change the disjunction into an implication.}$$

$$\forall_{A\,n\in\mathbb{N}} (|A| = n) \Rightarrow \phi_A \quad \Leftrightarrow \quad \text{We simplify.}$$

$$\forall_{n\in\mathbb{N}} \forall_{A,|A|=n} \phi_A$$

For this setting we now derive a new inference rule based on classical induction on $n \in \mathbb{N}$.

1. Induction base

Prove: $\forall_{A,|A|=0} \phi_A$

2. Induction step

Assume: $n_0 \in \mathbb{N} \bigwedge \forall_{A,|A|=n_0} \phi_A$

where n_0 is a new constant

Prove: $\underset{\substack{A \\ |A|=n_0+1}}{\forall} \phi_A$

We are now able to simplify the proof situation:

In our induction base we get from $|A| = 0$, that $A = \{\}$. Therefore we have to prove $\phi_{\{\}\leftarrow A}$. In the induction step we take A_0 arbitrary but fixed.

1. Induction base (new)

   Prove: $\phi_{\{\}\leftarrow A}$

2. Induction step (new)

   Assume: $n_0 \in \mathbb{N} \bigwedge \underset{\substack{A \\ |A|=n_0}}{\forall} \phi_A \bigwedge |A_0| = n_0 + 1$

   where n_0 and A_0 are a new constants

   Prove: $\phi_{A_0\leftarrow A}$

2.4 The language construct an arbitrary element of

The *Theorema* language provides several special language constructs. One of those is the so called *arbitrary element* or *an element* language construct.

If we speak about set theory in this thesis, we mean *Zermelo-Fraenkel* set theory *with* the axiom of choice.

The axiom of choice in *Zermelo-Fraenkel* set theory states the following:

$$\underset{A\neq\{\}}{\forall} \underset{x}{\exists} x \in A$$

Therefore we are allowed to introduce a so called *Skolem function æ* for which we know:

$$\underset{A\neq\{\}}{\forall} æ[A] \in A$$

Note: æ[A] stands for *an arbitrary element of A* or *an element of A.*

Example: Given: Set {5, 8, -3, 10}

Compute: New set with one arbitrary element taken out of our set.

```
Compute[{5, 8, −3, 10} \ {æ[{5, 8, −3, 10}]}, built–in → ⟨Built–in["Sets"], Built–in["Numbers"]⟩]
{5, 8, 10}
```

2.5 Inference rules

We discuss several inference rules that are implemented in the new special prover SIP.

- Rules for induction over the cardinality of finite sets.
- Rules conning the æ language construct.
- Other important rules.

2.5.1 Terminology

We write $K \vdash G$ for a proof situation where K denotes the knowledge-base and G the goal. What is actually meant is: "We have to prove the goal G from the knowledge K". Usually, G is a single formulae and K is a collection of assumptions. The transformation of $K \vdash G$ into a new proof situation $K' \vdash G'$ is called inference rule $\mathcal{I}$ and will be denoted by:

$$\mathcal{I}: \quad \frac{\mathbf{KB'} \vdash G'}{\mathbf{KB} \vdash G}$$

How does such an inference rule look like: The well known modus ponens rule is written in the following form.

$$\textbf{Modus Ponens}: \quad \frac{\mathbf{g, f, f \Rightarrow g, \Lambda} \vdash G}{\mathbf{f, f \Rightarrow g, \Lambda} \vdash G}$$

If we have the formulae f and $f \Rightarrow g$ in our knowledge base, we can also add g to our knowledge. Note that $\mathbf{\Lambda}$ denotes other formulae in the knowledge base that are not considered in the particular inference rule.

We describe, using pattern matching, an actual goal and/or a knowledge base. Then we do one step and produce from the current proof situation a new proof situation.

Note: Pattern matching makes possible some of the most succinct and elegant programs in the *Mathematica* programming language—immediately compressing large numbers of conditional cases into simple, readable and efficient pattern specifications.

The basic constructs for pattern matching, which we will use in our implementation, have the following meaning:

x_ *(one _ characters)* or Blank[] is a pattern object that can stand for any *Mathematica* expression.

x__ *(two _ characters)* or BlankSequence[] is a pattern object that can stand for any sequence of one or more *Mathematica* expressions.

x___ *(three _ characters)* or BlankNullSequence[] is a pattern object that can stand for any sequence of zero or more *Mathematica* expressions.

patt /; test is a pattern which matches only if the evaluation of test yields True.

An implementation of this rule in *Theorema* looks as follows :

```
PPND[ goal_,  •asml[pre___,  •lf[l_, f_, i_], between___, •lf[orl_, f_ ⇒ g_, ori_], post___],  af_]
    Module[{pf  =  •lf[NewLabel[¢ModusPonens, l, orl], g,  CombinedFormulaInfo[i, ori]]
        ProofStep[Prinfo[¢ModusPonens, {l, orl} ⟶ pf],
          Psit[goal, •asml[pf, pre, •lf[l, f,  i], between, post],  af]]];
```

What we have is:

Head In this case *PPND*, the name of the special prover.

Parameters 1. Goal:

In this case, the goal is not modified.

2. Knowledge Base:

We construct our knowledge base as a list of assumptions, called •*asml*.

We maybe have other formulae in our knowledge base before, in between and after those we are interested in (pre, between, post).

•*lf[l_, f_, i_]* means labeled formula and has as parameters the label of the formula, the pattern of our formula and a third parameter for additional information.

Program definition The actual program executed when the inference rule is applied. It is written in *Mathematica* programming language. In this case, we built up a new formula *pf*. It is a new labeled formula containing a new label (generated with the function *NewLabel*), the formula *g* and a combination of the additional information of the two used formulae.

Proof Step We then perform the proof step with the function *ProofStep*. We first have to give a proof info with the function *Prinfo* about the used and generated formulae. Those are used in order to generate a nice readable proof text.

And we have to describe our new proof situation: The function *PSit* states as parameters the new goal, the new list of assumptions and additional facts.

2.5.2 Inference rules for induction over the cardinality of finite sets

This section provides the rules for induction over the cardinality of a finite set. See Section 2.3 for details. We provide two inference rules. First one default case, where the induction base starts at $A = \{\}$.

$$\textbf{IndC:}\quad \frac{\mathbf{KB} \vdash \phi_{\{\} \leftarrow A} \qquad n_0 \in \mathbb{N} \bigwedge \forall_A (|A| = n_0 \Rightarrow \phi_A) \bigwedge A_0 \neq \{\} \bigwedge (|A_0| = n_0 + 1) \vdash \phi_{A_0 \leftarrow A}}{\mathbf{KB} \vdash \forall_{\text{is–finite}[A]} \phi_A}$$

(where n_0 and A_0 are new constants.)

This rule applies to formulae of the kind $\forall_{\text{is–finite}[A]} \phi_A$.

```
SIP[•lf[l1_, ∀ is–finite[A_] formula_, i1_], asmlist_, af_] :=

 Module[{IndBase, IndStepAssm, IndStepAssm1, IndStepGAssm, IndStepG, n, formulaBase,
    formulaStep, form = SimplifyQuantifier[formula], A0 = ArbitraryButFixed[•var[A]]},
  n = ArbitraryButFixed[•var[Tma`n]];
  formulaBase = SubstituteUnderQuantifier[form, {A → {}}];
  IndBase = •lf[NewLabel["IndC", l1, i1], formulaBase, i1];
  IndStepAssm1 = •lf[NewLabel["IndC", l1, i1], n ∈ ℕ, i1];
  IndStepAssm = •lf[NewLabel["IndC", l1, i1], ∀ A (((|A| = n)) ⇒ form), i1];

  IndStepGAssm = •lf[NewLabel["IndC", l1, i1], A0 ≠ {} ⋀ (|A0| = 1 + n), i1];
  formulaStep = SubstituteUnderQuantifier[form, {A → A0}];
  IndStepG = •lf[NewLabel["IndC", l1, i1], formulaStep, i1];
  ProofStep[
   Prinfo["IndC", {l1} → {IndBase, IndStepAssm1, IndStepAssm, IndStepGAssm, IndStepG}, A],

   Psit[IndBase, Map[SimplifyQuantifier, asmlist], af],
   Psit[IndStepG, Prepend[Prepend[Prepend[Map[SimplifyQuantifier, asmlist], IndStepAssm],
        IndStepAssm1], IndStepGAssm], af]]]
```

We set up the *induction base*:

We introduce the local variable *form*= SimplifyQuantifier[formula]. The function *SimplifyQuantifier* simplifies formulae that have quantifiers with conditions, e.g. $\underset{a\in A}{\forall}\ \phi$ is simplified to $\underset{a}{\forall}\ (a \in A) \Rightarrow \phi$.

In our induction base we have to prove the formula *formulaBase.* SubstituteUnderQuantifier[form, {A→{}}] means standard predicate logic substitution. Every occurrence of A is substituted by the empty set.

Induction step

We introduce the two induction step assumptions *IndStepAssm1* and *IndStepAssm.* They state $n_0 \in \mathbb{N}$ and $\underset{\substack{A\\|A|=n_0}}{\forall}$ form..

We introduce our goal formulae, *IndStepGAssm* and *IndStepG.* They state:

IndStepGAssm: $A_0 \neq \{\} \bigwedge |A_0| = 1 + n_0$. Using $A_0 \neq \{\}$ here is not standard procedure but done for technical reasons.

IndStepG: SubstituteUnderQuantifier[form, $\{A \to A_0\}$]

So we substitute A_0 for A in form in our *IndStepG*.

Proof step From our formula *l1* we generated the discussed formulae.

We now end up with two proof situations. One for the induction base and one for the induction step.

Here one can see the application of this rule. It just demonstrates the set up of our induction rule.

Note: In the following examples the formula ϕ always remains undefined. Therefore, the proofs cannot succeed and we just demonstrate the application of our inference rules.

Example: $\underset{\textbf{is–finite[A]}}{\forall} \boldsymbol{\phi_A}$

```
Proposition["IndTest", ∀_is–finite[A ] φ_A];
Prove[Proposition["IndTest"],
    by → SetTheoryPCSProver, ProverOptions → {TransformRanges → False}];
```

Prove:

(Proposition (IndTest)) $\underset{\text{is–finite}[A]}{\forall} \phi_A$,

with no assumptions.

We prove (Proposition (IndTest)) by Induction over the cardinality of the set A.

1. Induction base: We have to prove

(1) $\phi_{\{\}}$.

2. Induction step: We assume

(2) $n_0 \in \mathbb{N}$,

(3) $\underset{A}{\forall} ((|A| = n_0) \Rightarrow \phi_A)$,

and

(4) $A_0 \neq \{\} \bigwedge (|A_0| = 1 + n_0)$,

and prove

(5) ϕ_{A_0}

Anyway, there are some situations (see Section 3.1), where we want our induction base to start anywhere else, e.g. at $|A| = 1$. Therefore we provide a more general induction rule with a flexible induction base.

IndC :

$$\frac{\mathbf{KB} \vdash |A_0| = k \Rightarrow \phi_{A_0 \leftarrow A} \qquad n_0 \in \mathbb{N} \bigwedge \forall_A (|A| = n_0 \Rightarrow \phi_A) \bigwedge A_1 \neq \{\} \bigwedge (|A_1| = n_0 + 1) \vdash \phi_{A_1 \leftarrow A}}{\mathbf{KB} \vdash \forall_{\substack{\text{is–finite}[A] \\ |A| \geq k}} \phi_A}$$

This inference rule differs in pattern matching. It is applied if and only if the formula we have to prove has the shape: $\forall_{\substack{\text{is–finite}[A] \\ |A| \geq k}} \text{form}[A]$

It then starts the induction base not with the empty set, but with $|A| = k$.

```
SIP[•lf[l1_, ∀_{is–finite[A_] ∧ |A_|≥k_} formula_, i1_], asmlist_, af_] :=
  Module[{IndBase, IndStepAssm, IndStepAssm1, IndStepGAssm,
      IndStepG, n, formulaBase, formulaStep, form = SimplifyQuantifier[formula],
      A0 = ArbitraryButFixed[•var[A]], A1 = ArbitraryButFixed[•var[A]]},

    n = ArbitraryButFixed[•var[Tma`n]];
    IndBaseAssm = •lf[NewLabel["IndC", l1, i1], |A0| = k, i1];
    formulaBase = SubstituteUnderQuantifier[form, {A → A0}];
    IndBaseG = •lf[NewLabel["IndC", l1, i1], formulaBase, i1];
    IndStepAssm1 = •lf[NewLabel["IndC", l1, i1], n ∈ ℕ, i1];
    IndStepAssm = •lf[NewLabel["IndC", l1, i1], ∀_A ((|A| = n)) ⇒ form), i1];
    IndStepGAssm = •lf[NewLabel["IndC", l1, i1], A1 ≠ {} ∧ (|A1| = 1 + n), i1];
    formulaStep = SubstituteUnderQuantifier[form, {A → A1}];
    IndStepG = •lf[NewLabel["IndC", l1, i1], formulaStep, i1];
```

```
ProofStep[
    Prinfo["IndC", {l1} →
        {IndBaseAssm, IndBaseG, IndStepAssm1, IndStepAssm, IndStepGAssm, IndStepG}, A],

    Psit[IndBaseG, Prepend[Map[SimplifyQuantifier, asmlist], IndBaseAssm], af],
    Psit[IndStepG, Prepend[Prepend[Prepend[Map[SimplifyQuantifier, asmlist], IndStepAssm],
        IndStepAssm1], IndStepGAssm], af]
  ]]

NewLabel["IndC", l1_, i1_] := UniqueLabel[l1 <> ".IndC."];
```

Example : $\forall_{\substack{\text{is–finite}[A] \\ |A| \geq k}} \phi_A$

```
Proposition["IndTest", ∀_(is–finite[A] ∧ |A|≥k) ϕ_A];

Prove[Proposition["IndTest"],
    by → SetTheoryPCSProver, ProverOptions → {TransformRanges → False}];
```

Prove:

(Proposition (IndTest)) $\forall_{\substack{\text{is–finite}[A] \\ |A| \geq k}} \phi_A$,

with no assumptions.

We prove (Proposition (IndTest)) by Induction over the cardinality of the set A.

1. Induction base: We assume

(1) $|A_0| = k$,

and prove

(2) ϕ_{A_0}

2. Induction step: We assume

(3) $n_0 \in \mathbb{N}$,

(4) $\forall_A ((|A| = n_0) \Rightarrow \phi_A)$,

and

(5) $A_1 \neq \{\} \bigwedge (|A_1| = 1 + n_0)$,

and prove

(6) ϕ_{A_1}

2.5.3 Inference rules for æ language construct

Here we want to present the most important inference rules dealing with the æ language construct.

ArbitraryElementTr : $\dfrac{\mathbf{KB} \vdash \mathbb{T}}{\mathbf{KB} \vdash \mathbf{æ[A] \in A}}$

```
SIP[•lf[l_, æ[A_] ∈ A_, i_], asml_, af_] :=
  Block[{},
   FinalProofStep[Prinfo["ArbitraryElementTr", {l} –> {}], ¢Proved]]
```

The first inference rule talks about the trivial fact, that an arbitrary element of the set A is of course itself an element of A. (See also the definition of our Skolem function.)

So if we have to prove æ[A] ∈ A our rule leads us to a *FinalProofStep* and we are done with the proof.

Example: $\underset{A}{\forall}\ \mathbf{æ[}A\mathbf{] \in } A$

```
Proposition["RuleTest", ∀A æ[A] ∈ A]

Prove[Proposition["RuleTest"], by → SetTheoryPCSProver];
```

Prove:

(Proposition (RuleTest)) $\underset{A}{\forall}$ (æ[A] ∈ A),

with no assumptions.

For proving (Proposition (RuleTest)) we take all variables arbitrary but fixed and prove:

(1) æ[A_0] ∈ A_0.

The goal (1) trivially holds!

□

ArbitraryElementUn2 : $\dfrac{\mathbf{a = b,\ \Lambda\ \vdash\ G}}{\mathbf{A = \{b\},\ a = æ[A],\ \Lambda\ \vdash\ G}}$

```
SIP[goal_, •asml[pre___, f1 : •lf[l1_, A_ = {b_}, i1_],
      mid___, f2 : •lf[l2_, a_ = æ[A_], i2_], post___], af_] :=
  Block[{newF},
    newF = •lf[NewLabel["ArbitraryElementUn2", l1, l2], a = b, i1];
    ProofStep[Prinfo["ArbitraryElementUn2", {l1, l2} → newF],
      Psit[goal, •asml[newF, pre, mid, post], af]]
  ]
```

```
SIP[goal_, •asml[pre___, f1 : •lf[l2_, a_ = æ[A_], i2_],
      mid___, f2 : •lf[l1_, A_ = {b_}, i1_], post___], af_] :=
  Block[{newF},
    newF = •lf[NewLabel["ArbitraryElementUn2", l1, l2], a = b, i1];
    ProofStep[Prinfo["ArbitraryElementUn2", {l1, l2} → newF],
      Psit[goal, •asml[newF, pre, mid, post], af]]
  ]
```

If we have a singleton set $A = \{b\}$, then the arbitrary element $a = æ[A]$ has to be equal to b.

Example: $\underset{A,a,b}{\forall}\ (((A = \{b\}) \bigwedge (a = æ[A])) \Rightarrow \phi)$

```
Proposition["RuleTest", ∀_{A,a,b} (((A = {b}) ⋀ (a = æ[A])) ⇒ ϕ)]

Prove[Proposition["RuleTest"], by → SetTheoryPCSProver];
```

Prove:

(Proposition (RuleTest)) $\underset{a,A,b}{\forall}\ ((A = \{b\}) \bigwedge (a = æ[A]) \Rightarrow \phi)$,

with no assumptions.

We assume

(1) $(A_0 = \{b_0\}) \bigwedge (a_0 = æ[A_0])$,

and show

(2) ϕ.

From (1.1) and (1.2) we obtain by simplification:

(3) $a_0 = b_0$.

ArbitraryElementn1b : $\dfrac{|A\backslash\{æ[A]\}| = n, \Lambda \vdash G}{|A| = 1 + n, \Lambda \vdash G}$

```
NewKnowledgeFromOne[•lf[l1_, |A_| = 1 + n_, i1_]] :=
 Module[{newAssum},
    newAssum = •lf[NewLabel["ArbitraryElementn1b", l1], |A \ {æ[A]}| = n, i1];
    {{•asml[newAssum], l1}}
 ]
```

Here we have a rule, that is again frequently used in proofs by induction during the induction step.

If we know, that $|A| = 1 + n$, we can deduce, that $|A \setminus \{æ[A]\}| = n$. (This rule is used to then be able to use the induction hypothesis during the induction step.)

Note: Here we do not use the head *SIP*, but another *Theorema* function, *NewKnowledgeFromOne*. It is used to built up from one formula in the knowledge base another one. The important thing here is, that the formula $|A| = 1 + n$ is also kept in our knowledge base, not replaced by the other one.

Example: $\forall_{A,n} ((|A| = n + 1) \Rightarrow \phi)$

```
Proposition["RuleTest", ∀_{A,n} ((|A| = n + 1) ⇒ ϕ)]

Prove[Proposition["RuleTest"], by → SetTheoryPCSProver];
```

Prove:

(Proposition (RuleTest)) $\forall_{A,n} ((|A| = n + 1) \Rightarrow \phi)$,

with no assumptions.

We assume

(1) $|A_0| = n_0 + 1$,

and show

(2) ϕ.

From what we already know follows:

From (1) we can infer

(3) $|A_0 \setminus \{æ[A_0]\}| = n_0$.

ArbitraryElementC1 : $\dfrac{\mathbf{|T|=n,\ \Lambda \vdash G}}{\mathbf{|A| = n + 1,\ T = A\setminus\{æ[A]\},\ \Lambda \vdash G}}$

```
SIP[goal_, •asml[pre___, f1 : •lf[l1_, |A_| = n_ + 1, i1_],
      mid___, f2 : •lf[l2_, T_ = A_ \ {æ[A_]}, i2_], post___], af_] :=
  Block[{newF},
    newF = •lf[NewLabel["ArbitraryElementC1", l1, l2], |T| = n, i1];

 ProofStep[Prinfo["ArbitraryElementC1", {l1, l2} -> newF],
      Psit[goal, •asml[newF, pre, mid, post], af]] ]
```

```
SIP[goal_, •asml[pre___, f2 : •lf[l2_, T_ = A_ \ {æ[A_]}, i2_],
      mid___, f1 : •lf[l1_, |A_| = n_ + 1, i1_], post___], af_] :=

 Block[{newF},
 newF = •lf[NewLabel["ArbitraryElementC1", l1, l2], |T| = n, i1];

 ProofStep[Prinfo["ArbitraryElementC1", {l1, l2} -> newF],
      Psit[goal, •asml[newF, pre, mid, post], af]]]
```

This inference rule is a variation of our rule *ArbitraryElementn1b*. It looks similar to the other rule and was implemented for efficiency reasons. With this rule we save the prover some set theoretical proving steps. During our induction step, we for example know, that the set A has $n + 1$ elements. This knowledge together with the knowledge about a set T, which is built up from A taking out an arbitrary element of A, is used to deduce, that T must then have exactly n elements.

The proof of the correctness of this inference rule is easy.

We know that $|\{æ[A]\}| = 1$ for every non empty A. Since $|A| = 1 + n$ and $æ[A] \in A$, we know that

$T = A \setminus \{æ[A]\}$ has exactly n elements.

Example: $\underset{A,T,n}{\forall} (((|A| = n+1) \bigwedge (T = A \setminus \{æ[A]\})) \Rightarrow \phi)$

```
Proposition["RuleTest", ∀_{A,T,n} (((|A| = n + 1) ⋀ (T = A \ {æ[A]})) ⇒ ϕ)]

Prove[Proposition["RuleTest"], by → SetTheoryPCSProver];
```

Prove:

(Proposition (RuleTest)) $\underset{A,n,T}{\forall} ((|A| = n+1) \bigwedge (T = A \setminus \{æ[A]\}) \Rightarrow \phi)$,

with no assumptions.

We assume

(1) $(|A_0| = n_0 + 1) \bigwedge (T_0 = A_0 \setminus \{æ[A_0]\})$,

and show

(2) ϕ.

From (1.1) and (1.2) we obtain by simplification:

(3) $|T_0| = n_0$.

2.5.4 Other important inference rules

Here we want to present some other implemented inference rules which do not directly deal with the æ language construct. We added those inference rules during the work on our case studies (see Chapter 3). Those rules shall be implemented as universally applicable as possible. We intend them to be useful not only in those special case studies but also more generally.

Dis : $\dfrac{\mathbf{KB}, \Lambda \vdash G_{|A|+|B| \leftarrow |A \cup B|}}{\mathbf{C} \cap \mathbf{D} = \{\}, \Lambda \vdash \mathbf{G}}$

if ((EqRen[A, C] ⋀ EqRen[B, D]) ⋁ EqRen[A, D] ⋀ EqRen[B, C])

```
EqRen[A_, B_] :=
  Module[{boundA = BoundedVariables[A], boundB = BoundedVariables[B]},
    len1 = Length[boundA]; len2 = Length[boundB];
    If[len1 == len2
          Sub = Map[Thread[Rule[boundA, boundB[[#]]]] &, Permutations[Range[len1]]];
      MemberQ[Table[(A /. Sub[[i]]) === B, {i, 1, Length[Sub]}], True], False]]
```

```
SIP[•lf[l_, goal_, i_], •asml[pre___, •lf[l1_, c_ ⋂ d_ = {}, i_], post___], af_] /;
    ! FreeQ[goal, •[|_ ⋃ _|]] :=
  Module[{A, B, newgoal},
    newgoal = goal /. •[|A_ ⋃ B_|] /;
          ((EqRen[A, c] ⋀ EqRen[B, d]) ⋁ (EqRen[A, d] ⋀ EqRen[B, c])) :> •[|A| + |B|];
    If[newgoal === goal, Return[Null],
      newF = •lf[NewLabel["Dis", l, i], newgoal, i];
      ProofStep[Prinfo["Dis", {l, l1} → newF], Psit[newF, •asml[pre, post], af]]]]
```

This inference rule speaks about the cardinality of a disjoint union of two finite sets. If we want to prove for example $|A \bigcup B| = x$ and we already know $A \bigcap B = \{\}$, we may split up the union and prove $|A| + |B| = x$ instead. Anyway, maybe we don't know this disjointness for A and B actually, but for sets C and D which do just differ in their bound variables and therefore describe the same sets.

The function EqRen[A, C] does this test. It checks, weather A and C are identical except renaming of bound variables.

Example: We look at the sets: $A = \left\{\{a, \pi\} \underset{a}{\mid} a \in Z\right\}$

$$B = \left\{\{b, \pi\} \underset{b}{\mid} b \in Z\right\}$$

$$C = \left\{\{b\} \underset{b}{\mid} b \in Z\right\}$$

```
EqRen[•[{{a, π} | a ∈ Z}], •[{{b, π} | b ∈ Z}]]
```

True

```
EqRen[•[{{a, π} | a ∈ Z}], •[{{b} | b ∈ Z}]]
```

False

As we can see, the inference rule is only applied if in our pattern we have to prove something about a

cardinality of a union of two sets and the tests using the function *EqRen* return True.

Example: $\underset{A,B}{\forall}\ (((A \bigcap B = \{\})) \Rightarrow (|A \bigcup B|))$

```
Proposition["RuleTest", ∀_{A,B} (((A ⋂ B = {})) ⇒ (|A ⋃ B|))]

Prove[Proposition["RuleTest"], by → SetTheoryPCSProver];
```

Prove:

(Proposition (RuleTest)) $\underset{A,B}{\forall}\ ((A \bigcap B = \{\}) \Rightarrow |A \bigcup B|)$,

with no assumptions.

We assume

(1) $A_0 \bigcap B_0 = \{\}$,

and show

(2) $|A_0 \bigcup B_0|$.

In order to prove (2), due to (1), it is sufficient to prove

(3) $|A_0| + |B_0|$.

Card2 : $\dfrac{\mathbf{A = \{x_1,\ ...,\ x_k,\ y_{k+1},\ ...,\ y_n\},\ \{x_1,\ ...,\ x_k\} \bigcap \{y_{k+1},\ ...,\ y_n\} = \{\},\ \Lambda\ \vdash\ G}}{\mathbf{|A| = n,\ x_1 \in A,\ ...,\ x_k \in A,\ \Lambda\ \vdash\ G}}$ **(optional)**

```
SIP[goal_, a : •asml[pre___, f1 : •lf[l1_, |A_| = k_, i1_], post___], af_] /; $TmaSIPCardinv :=
 Block[{newF1, newF2, y = •var[NewIdentifierNotIn[Tma`y]], li},
    li = Cases[Table[a[[i]][[2]], {i, Length[a]}], •[_ ∈ A]];
    If[li === {}, Return[Null],
      If[Length[li] < k,
        set1 = Table[li[[i]][[1]], {i, Length[li]}];
        set2 = Table[•var[NewIdentifierNotIn[Tma`z]], {k – Length[li]}];
        set = Flatten[Append[set1, set2]];
        newF1 = •lf[NewLabel["Card2", l1], A = set, i1];
        newF2 = •lf[NewLabel["Card2", l1], set1 ⋂ set2 = {}, i1];
```

```
ProofStep[Prinfo["Card2", {l1} -> {newF1, newF2}, set2],
          Psit[goal, •asml[newF1, newF2 , pre, post], af]];

      set = Table[li[[i]][[1]], {i, k}];
      newF1 = •lf[NewLabel["Card2", l1], A = set , i1];
      ProofStep[Prinfo["Card2", {l1} -> {newF1}, set2],
        Psit[goal, •asml[newF1 , pre, post], af]]]]
```

Here we have a rule, that invents a new set if we know the cardinality of the set and some elements being in it. This rule is an optional rule. It is only used, if in our prover options *SIPCardinv* is set to True. The default value of this parameter *SIPCardinv* is False.

Example: $\underset{A}{\forall}\,((|A| = 5) \wedge x_1 \in A \wedge x_2 \in A \wedge x_3 \in A) \Rightarrow \phi$

```
Proposition["RuleTest", any[A],
  ((|A| = 5) ⋀ x1 ∈ A ⋀ x2 ∈ A ⋀ x3 ∈ A) ⇒ ϕ]

Prove[Proposition["RuleTest"], by → SetTheoryPCSProver,
    ProverOptions → {SimplifyFormula → True, SIPCardinv → True}] ;
```

Prove:

(Proposition (RuleTest)) $\underset{A}{\forall}\,((|A| = 5) \wedge x_1 \in A \wedge x_2 \in A \wedge x_3 \in A \Rightarrow \phi)$,

with no assumptions.

We assume

(1) $(|A_0| = 5) \wedge x_1 \in A_0 \wedge x_2 \in A_0 \wedge x_3 \in A_0$,

and show

(2) ϕ.

From (1.1), (1.2), (1.3) and (1.4) we obtain:

(3) $A_0 = \{x_1, x_2, x_3, z1, z2\}$,

and

(4) $\{x_1, x_2, x_3\} \cap \{z1, z2\} = \{\}$.

WhereG : $$\frac{\mathbf{KB} \vdash \mathbf{Seq}_{b \leftarrow a}}{\mathbf{KB} \vdash \mathbf{where[a = b, Seq]}}$$

Where : $$\frac{\mathbf{Seq}_{\mathbf{b \leftarrow a}}, \Lambda \vdash \mathbf{G}}{\mathbf{where[a = b, Seq]}, \Lambda \vdash \mathbf{G}}$$

```
SIP[•lf[l1_, goal_, i1_], asmlist_, af_] /; ! FreeQ[goal, •[where[_ = _, __]]] :=
  Module[{newF1},
    newgoal = goal /. •[where[a_ = b_, Seq__]] :> •[Seq /. a → b];
    If[newgoal === goal, Return[Null],
      newF1 = •lf[NewLabel["WhereG", l1, i1], newgoal, i1];
      ProofStep[Prinfo["WhereG", {l1} → {newF1}], Psit[newF1, asmlist, af]]]]
```

```
SIP[goal_, •asml[pre___, •lf[l1_, where[a_ = b_, Seq__], i1_], post___], af_] :=
  Block[{newF1, subs = Substitute[Seq, {a → b}]},
    newF1 = •lf[NewLabel["Where", l1, i1], subs, i1];
    ProofStep[Prinfo["Where", {l1} → {newF1}], Psit[goal, •asml[newF1, pre, post], af]]]
```

Here we have two rules, which take care of the *where* language construct. The *where* language construct is an abbreviation.

For example $\text{where}[a = æ[A],\ \{\{a,\ b\} \mid b \in A \setminus \{a\}\} \bigcup \text{S2}[A \setminus \{a\}]]$

is the abbreviation for

$$\{\{æ[A],\ b\} \mid b \in A \setminus \{æ[A]\}\} \bigcup \text{S2}[A \setminus \{æ[A]\}].$$

And this is exactly what this inference does. Both in the knowledge base and the goal if *where* somewhere occurs, the first parameter, the equality, is substituted. We replace left hand side of our equality by the right hand side in the second parameter.

Example: $\underset{A,n}{\forall}\ (\phi \Rightarrow (\text{where}[a = æ[A], |A \setminus \{a\}| = n]))$

$$\text{Proposition}\left["\text{RuleTest}",\ \underset{\text{A,n}}{\forall}\ (\phi \Rightarrow (\text{where}[\text{a} = æ[\text{A}], |\text{A} \setminus \{\text{a}\}| = \text{n}]))\right]$$

```
Prove[Proposition["RuleTest"], by → SetTheoryPCSProver];
```

Prove:

(Proposition (RuleTest)) $\underset{A,n}{\forall}\ (\phi \Rightarrow |A \setminus \{a\}| = n\ |_{a \leftarrow æ[A]})$,

with no assumptions.

By simplification we obtain from (Proposition (RuleTest)) the new Goal:

(1) $\underset{A,n}{\forall}\ (\phi \Rightarrow (|A \setminus \{æ[A]\}| = n))$.

In Chapter 3 we discuss five case studies to demonstrate our new special prover SIP and its interaction with the existing *Theorema* system.

Chapter 3: Case studies

In this section we want to present five case studies. We use our new special prover SIP together with the *SetTheoryPCSProver* already available in *Theorema* to automatically prove correctness properties about recursive algorithms and about user made definitions.

Note: Timings given in the following sections correspond to experiments carried out on a Windows PC (2.0 GHz) with 2.0 GB RAM memory. *Mathematica* version 7.0 was used.

3.1 Minimal element of a given finite set

As a first example, we talk about a recursive algorithm *MinEl* that gives the minimal element of a finite set, i.e. an element which is less or equal to all elements of the set.

Given: A set A

With: is–finite[A], $|A| \geq 1$

Compute: $m \in A$

Such that: $\forall_{a \in A} (m \leq a)$

A recursive algorithm to solve this problem works as follows:

> If A contains only one element, this element has to be our minimum.
>
> Otherwise we extract one arbitrary element a out of A and compute the standard two element minimum of a and the minimal element of the set $A \setminus \{a\}$.

```
Definition["MinEl", any[A, x],
  MinEl[{x}] = x                                        "base"
  MinEl[A] = where[a = æ[A], Min[a, MinEl[A \ {a}]]]    "rec"  ]
```

The correctness theorem for the algorithm *MinEl* with respect to the above specification is:

$$\text{Theorem}\Big[\text{"MinimalElement1"},\ \forall_{\substack{\text{is–finite[A]}\\ |A|\geq 1}}\ \forall_{a\in A}\ (\text{MinEl[A]} \leq a)\Big]$$

We prove the theorem *"MinimalElement"* by induction over the cardinality of A, starting our induction base with $|A| = 1$. The proof works automatically and needs less than 10 seconds to finish.

3.1.1 Theorema Proof

```
Prove[Theorem["MinimalElement1"], by → SetTheoryPCSProver,
    transformBy → ProofSimplifier, TransformerOptions → {branches → Proved, steps → Useful},
    built–in → {Built–in["Sets"], Built–in["Numbers"]},
    using → {Definition["MinEl"]}, SearchDepth → 70, ProverOptions →
      {UseCyclicRules → True, TransformRanges → False, SimplifyFormula → True}] // Timing
{6.312, - ProofObject -}
```

Note : The proofs we show in this chapter are built fully automatic. We provide a copy of the output as it is returned by the *Theorema* system.

Prove:

(Proposition (MinimalElement1)) $\forall_{\substack{\text{is–finite}[A]\\ |A|\geq 1}}\ \forall_{a\in A}\ (\text{MinEl}[A] \leq a)$,

under the assumptions:

(Definition (MinEl): base) $\forall_{x}\ (\text{MinEl}[\{x\}] = x)$,

(Definition (MinEl): rec) $\forall_{A}\ (\text{MinEl}[A] = \text{Min}[\text{MinEl}[A \setminus \{a\}], a]\ |_{a\leftarrow \text{æ}[A]})$.

We prove (Proposition (MinimalElement1)) by induction over the cardinality of the set A.

1. Induction base: We assume

(1) $|A_0| = 1$,

and prove

(2) $\forall_{a}\ (a \in A_0 \Rightarrow \text{MinEl}[A_0] \leq a)$

We assume

(7) $a_0 \in A_0$,

and show

(8) $\mathrm{MinEl}[A_0] \leq a_0$.

From (1) and (7) we obtain by simplification:

(9) $A_0 = \{a_0\}$.

Formula (8), using (9), is implied by:

$\mathrm{MinEl}[\{a_0\}] \leq a_0$,

which, using (Definition (MinEl): base), is implied by:

(10) $a_0 \leq a_0$.

Using built–in simplification rules and the additional assumption(s) (9) we simplify (10) to

(11) True.

Formula (11) is true because it is the constant True.

2. Induction step: We assume

(3) $n_0 \in \mathbb{N}$,

(4) $\forall_A \left((|A| = n_0) \Rightarrow \forall_a (a \in A \Rightarrow \mathrm{MinEl}[A] \leq a) \right)$,

and

(5) $A_1 \neq \{\} \bigwedge (|A_1| = 1 + n_0)$,

and prove

(6) $\forall_a (a \in A_1 \Rightarrow \mathrm{MinEl}[A_1] \leq a)$

From what we already know follows:

From (5.1) we know that we can choose an appropriate value such that

(12) $A1_0 \in A_1$.

From (5.2) we can infer

(13) $|A_1 \setminus \{\text{æ}[A_1]\}| = n_0$.

We assume

(14) $a_1 \in A_1$,

and show

(15) $\mathrm{MinEl}[A_1] \leq a_1$.

Formula (15), using (Definition (MinEl): rec), is implied by:

(16) $\mathrm{Min}[\mathrm{MinEl}[A_1 \setminus \{a\}], a] \mid_{a \leftarrow æ[A_1]} \leq a_1$.

By simplification we obtain from (16) the new goal:

(17) $\mathrm{Min}[\mathrm{MinEl}[A_1 \setminus \{æ[A_1]\}], æ[A_1]] \leq a_1$.

In order to prove (17), it is sufficient to prove

(18) $\mathrm{MinEl}[A_1 \setminus \{æ[A_1]\}] \leq a_1 \bigvee æ[A_1] \leq a_1$

We prove (18) by proving the first alternative negating the other(s).

We assume

(20) $æ[A_1] \not\leq a_1$.

We now show

(19) $\mathrm{MinEl}[A_1 \setminus \{æ[A_1]\}] \leq a_1$.

By simplification of (20) we may add to the knowledge base:

(21) $æ[A_1] > a_1$.

Formula (19), using (4), is implied by:

$a_1 \in A_1 \setminus \{æ[A_1]\} \bigwedge (|A_1 \setminus \{æ[A_1]\}| = n_0)$,

which, using (13), is implied by:

(22) $a_1 \in A_1 \setminus \{æ[A_1]\} \bigwedge (n_0 = n_0)$.

Using built–in simplification rules and the additional assumption(s) (21), (13), (5.1), and (5.2) we simplify (22) to

(23) $a_1 \in A_1 \setminus \{æ[A_1]\}$.

We have to prove (23), thus, we first show:

(24) $a_1 \in A_1$:

Formula (24) is true because it is identical to (14).

For proving (23) it still remains to show

(25) $a_1 \notin \{æ[A_1]\}$:

In order to prove (25), it is sufficient to prove

(26) $a_1 \neq æ[A_1]$

The goal (26) trivially holds because of (21).

□

3.1.2 Notes

Note: See Section 2.5 for detailed explanations of the inference rules.

We first set up our generalized induction base in formula (1). We plug in our algorithm definition (base case) into the proof goal (8) and end up with the trivial goal (10).

The induction step is set up with the formulae (2), (3), (4) and (5). Formula (13) is built from (5) with our inference rule *ArbitraryElementn1b*. In (16) we plug in our algorithm definition (rec) and do some simplifications to end up with the proof goal (18). We proof the first part of the or–clause and negate the second one and therefore get (19) and (20).

We then plug in (4), our induction hypothesis, and (13) into (19), simplify again and end up with (23). The proof of (23) is done with standard set theory.

3.2 The cardinality of all 2-element subsets of a given finite set

We now want to look at an algorithm that computes all the two element subsets of a given finite set *A*. We prove a cardinality property about it.

Given: A set *A*
With: is–finite[*A*]
Compute: S2
Such that: S2 = {B ∈ 𝒫[A] | |B| = 2}

Our recursive algorithm works as follows:

If *A* is the empty set, there are of course no 2-element subsets.
Otherwise we do the following:
We take one arbitrary element *a* out of *A*. We form the union of two sets, namely:

(1) The set of all two element subsets of *A* with the arbitrary element *a* in it, e.g. the set of all {*a*, *b*}, where *b* is an element of *A* \ {*a*}.
(2) The set of all 2-element subsets of *A* \ {*a*}.

Together we computed all 2-element subsets of *A*, those containing *a* in (1) and those that do not contain *a* in (2).

```
Definition["S2", any[A],
  S2[{}] = {}                                                          "base"
  A ≠ {} ⇒ (S2[A] = where[a = æ[A], {{a, b} | b ∈ A \ {a}} ⋃ S2[A \ {a}]])  "rec"  ]
```

The *standard definition S2Def* of a all two element subsets of a given finite set looks as follows:

```
Definition["S2Def", any[A],
  S2Def[A] = {B ∈ 𝒫[A] | (|B| = 2)}]
```

We now are also able to use our definition *S2*, to really compute such two element subsets. For this we use our definition and some *Theorema* Built–in's as a knowledge base for computation. Note that computation with the *standard definition* does not work really efficiently because one first computes the whole powerset and then has to filter all the two element subsets. Look at the following examples:

```
Compute[S2[{}], using → ⟨Definition["S2"]⟩,
  built–in → ⟨Built–in["Sets"], Built–in["Numbers"], Built–in["Connectives"],
      Built–in["AllOperators"], Built–in["Quantifiers"]⟩]
```

```
{}
```

```
Compute[S2[{1, 2, 3, 4, 5, 6}], using → ⟨Definition["S2"]⟩,
  built–in → ⟨Built–in["Sets"], Built–in["Numbers"],
      Built–in["Connectives"], Built–in["AllOperators"], Built–in["Quantifiers"]⟩]
```

```
{{1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}, {2, 3}, {2, 4},
 {2, 5}, {2, 6}, {3, 4}, {3, 5}, {3, 6}, {4, 5}, {4, 6}, {5, 6}}
```

```
Compute[S2Def[{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13}], using → ⟨Definition["S2Def"]⟩,
      built–in → ⟨Built–in["Sets"], Built–in["Numbers"], Built–in["Connectives"],
          Built–in["Quantifiers"], Built–in["Number Domains"]⟩]; // Timing
```

```
{16.625, Null}
```

```
Compute[S2[{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13}],
      using → ⟨Definition["S2"]⟩, built–in → ⟨Built–in["Sets"], Built–in["Numbers"],
          Built–in["Connectives"], Built–in["Quantifiers"]⟩]; // Timing
```

```
{0.172, Null}
```

Computation with the *standard definition* takes in our example of a 13 element set nearly 17 seconds whereas with our algorithmic definition *S2* we succeed within less than one second.

The correctness of this algorithm will be discussed in detail in Section 3.4. In this section we discuss the cardinality of the set generated by the algorithm *S2*, namely:

$$\text{Theorem}\left["2eSS", \underset{\text{is–finite[A]}}{\forall} \left(|S2[A]| = \frac{1}{2} * (|A| * (|A| - 1))\right)\right]$$

To prove this we need some additional knowledge.

3.2.1 Lemmata for Induction

1. The two sets $\left\{\{æ[A], b\} \mid_{b} b \in A \setminus \{æ[A]\}\right\}$ and $S2[A \setminus \{æ[A]\}]$ are disjoint.

$$\text{Lemma}\left["\text{disjoint sets}", \text{any}[A, n], \text{with}[|A| = 1 + n], \left\{\{æ[A], b\} \mid_{b} b \in A \setminus \{æ[A]\}\right\} \cap S2[A \setminus \{æ[A]\}] = \{\}\right]$$

This is easy to see, because we know, that the first set contains all 2-element subsets of A with æ[A], whereas the second set contains the 2-element subsets of $A \setminus \{æ[A]\}$, e.g. all the 2-element subsets without æ[A].

2. The set $\left\{\{æ[A], b\} \mid_{b} b \in A \setminus \{æ[A]\}\right\}$ contains exactly as many elements as the set $A \setminus \{æ[A]\}$.

$$\text{Lemma}\left["\text{card prop}", \text{any}[A], \left|\left\{\{æ[A], b\} \mid_{b} b \in A \setminus \{æ[A]\}\right\}\right| = |A \setminus \{æ[A]\}|\right]$$

3.2.2 Theorema Proof

Using those lemmata, we are now prepared to prove our desired cardinality theorem. The proof takes about 10 seconds to finish.

```
Prove[Theorem["2eSS"], by → SetTheoryPCSProver,
    using → {Definition["S2"], Lemma["disjoint sets"], Lemma["card prop"]},
    transformBy → ProofSimplifier, TransformerOptions → {branches → Proved, steps → Useful},
    built–in → {Built–in["Numbers"]}, SearchDepth → 100, ProverOptions →
      {SimplifyFormula → True, UseCyclicRules → True, TransformRanges → False}] // Timing
{9.922, - ProofObject -}
```

Prove:

(Theorem (2eSS)) $\underset{\text{is–finite}[A]}{\forall} \left(|\text{S2}[A]| = \frac{1}{2} * (|A| * (|A| - 1))\right)$,

under the assumptions:

(Definition (S2): base) $\text{S2}[\{\}] = \{\}$,

(Definition (S2): rec) $\underset{A}{\forall}\left(A \neq \{\} \Rightarrow \left(\text{S2}[A] = \left\{\{a, b\} \mid_{b \in A \setminus \{a\}}\right\} \cup \text{S2}[A \setminus \{a\}]\,|_{a \leftarrow æ[A]}\right)\right)$,

(Lemma (disjoint sets)) $\underset{\substack{A,n \\ |A|=1+n}}{\forall}\left(\left\{\{æ[A], b\} \mid_{b} b \in A \setminus \{æ[A]\}\right\} \cap (\text{S2}[A \setminus \{æ[A]\}]) = \{\}\right)$,

(Lemma (card prop)) $\underset{A}{\forall}\left(\left|\left\{\{æ[A], b\} \mid_{b} b \in A \setminus \{æ[A]\}\right\}\right| = |A \setminus \{æ[A]\}|\right)$.

We prove (Theorem (2eSS)) by induction over the cardinality of the set A.

1. Induction base: We have to prove

(1) $|S2[\{\}]| = \frac{1}{2} * (|\{\}| * (|\{\}| - 1))$.

Using built–in simplification rules and the additional assumption(s) (Definition (S2): base) we simplify (1) to

(7) $|S2[\{\}]| = 0$.

Formula (7), using (Definition (S2): base), is implied by:

(8) $|\{\}| = 0$.

Using built–in simplification rules and the additional assumption(s) (Definition (S2): base) we simplify (8) to

(9) True.

Formula (9) is true because it is the constant True.

2. Induction step: We assume

(2) $n_0 \in \mathbb{N}$,

(3) $\underset{A}{\forall} \left((|A| = n_0) \Rightarrow \left(|S2[A]| = \frac{1}{2} * (|A| * (|A| - 1))\right)\right)$,

and

(4) $A_0 \neq \{\} \bigwedge (|A_0| = 1 + n_0)$,

and prove

(5) $|S2[A_0]| = \frac{1}{2} * (|A_0| * (|A_0| - 1))$

Using built–in simplification rules we can simplify the knowledge base:

Formula (3) simplifies to

(10) $\underset{A}{\forall} \left((|A| = n_0) \Rightarrow \left(|S2[A]| = \frac{1}{2} * (-1 + |A|) * |A|\right)\right)$.

From what we already know follows:

From (Definition (S2): base) we can infer

(11) $\underset{x1}{\forall} (x1 \notin S2[\{\}])$.

Using built–in simplification rules and the additional assumption(s) (Definition (S2): base) we simplify (5) to

(12) $|S2[A_0]| = \frac{1}{2} * (-1 + |A_0|) * |A_0|$.

From what we already know follows:

From (4.1) we know that we can choose an appropriate value such that

(13) $A1_0 \in A_0$.

From (4.2) we can infer

(14) $|A_0 \setminus \{æ[A_0]\}| = n_0$.

Formula (14), by (10), implies:

(15) $|S2[A_0 \setminus \{æ[A_0]\}]| = \frac{1}{2} * (-1 + |A_0 \setminus \{æ[A_0]\}|) * |A_0 \setminus \{æ[A_0]\}|$.

Formula (15), by (14), implies:

(16) $|S2[A_0 \setminus \{æ[A_0]\}]| = \frac{1}{2} * (-1 + n_0) * n_0$.

Formula (4.1), by (Definition (S2): rec), implies:

(17) $S2[A_0] = \left\{\{a, b\} \mid_b b \in A_0 \setminus \{a\}\right\} \bigcup S2[A_0 \setminus \{a\}] \,|_{a \leftarrow æ[A_0]}$.

Formula (4.2), by (Lemma (disjoint sets)), implies:

(18) $\left\{\{æ[A_0], b\} \mid_b b \in A_0 \setminus \{æ[A_0]\}\right\} \bigcap (S2[A_0 \setminus \{æ[A_0]\}]) = \{\}$.

Formula (12), using (17), is implied by:

$$\left|\left\{\{a, b\} \mid_b b \in A_0 \setminus \{a\}\right\} \bigcup S2[A_0 \setminus \{a\}] \,|_{a \leftarrow æ[A_0]}\right| = \frac{1}{2} * (-1 + |A_0|) * |A_0|,$$

which, using (4.2), is implied by:

(21) $$\left|\left\{\{a, b\} \mid_b b \in A_0 \setminus \{a\}\right\} \bigcup S2[A_0 \setminus \{a\}] \,|_{a \leftarrow æ[A_0]}\right| = \frac{1}{2} * (-1 + (1 + n_0)) * (1 + n_0).$$

By simplification we obtain from (21) the new goal:

(22) $$\left|\left\{\{æ[A_0], b\} \mid_b b \in A_0 \setminus \{æ[A_0]\}\right\} \bigcup S2[A_0 \setminus \{æ[A_0]\}]\right| = \frac{1}{2} * (-1 + (1 + n_0)) * (1 + n_0)\ .$$

In order to prove (22), due to (18), it is sufficient to prove

(23) $$\left|\left\{\{æ[A_0], b\} \mid_b b \in A_0 \setminus \{æ[A_0]\}\right\}\right| + |S2[A_0 \setminus \{æ[A_0]\}]| = \frac{1}{2} * (-1 + (1 + n_0)) * (1 + n_0)\ .$$

Using built–in simplification rules and the additional assumption(s) (17), (16), (14), (4.1), (4.2), and (Definition (S2): base) we simplify (23) to

(24) $$|S2[A_0 \setminus \{æ[A_0]\}]| + \left|\left\{\{æ[A_0], b\} \mid_b b \in A_0 \setminus \{æ[A_0]\}\right\}\right| = \frac{1}{2} * n_0 * (1 + n_0).$$

Formula (24), using (16), is implied by:

$$\frac{1}{2} * (-1 + n_0) * n_0 + \left| \left\{ \{æ[A_0], b\} \mid b \in A_0 \setminus \{æ[A_0]\} \right\} \right| = \frac{1}{2} * n_0 * (1 + n_0),$$

which, using (Lemma (card prop)), is implied by:

$$\frac{1}{2} * (-1 + n_0) * n_0 + |A_0 \setminus \{æ[A_0]\}| = \frac{1}{2} * n_0 * (1 + n_0),$$

which, using (14), is implied by:

(25) $\frac{1}{2} * (-1 + n_0) * n_0 + n_0 = \frac{1}{2} * n_0 * (1 + n_0)$.

Using built–in simplification rules and the additional assumption(s) (17), (16), (14), (4.1), (4.2), and (Definition (S2): base) we simplify (25) to

(26) True.

Formula (26) is true because it is the constant True.

□

3.2.3 Notes

We first set up our induction base in formula (1). We do a simplification step and then plug in our algorithm definition (base) into (7) and end up with the trivial goal (8).

The induction step is set up with the formulae (2), (3), (4) and (5). Formula (14) is built from (4) with our inference rule *ArbitraryElementn1b*. The proof goal (5) is simplified to (12).

Together with (14) we use our induction hypothesis (10) and get (15) and (16). We then use (4) in order to plug in our algorithm definition (rec) and get (17). We also instantiate our disjoint sets lemma in (18).

We then focus on our proof goal and built up the new goal (21). We eliminate the *where* language construct and end up with (22).

We now use (18) and our inference rule *Dis* to split the cardinality of the union into two parts and get (23) and after another simplification step (24). Together with (16) and the second lemma (card prop) we get (25), which can be proven by simplification.

3.3 The cardinality of all k-element subsets of a given finite set

We now want to look at an algorithm, that computes all the *k*-element subsets of a given finite set *A*. We again want to prove a cardinality property about it.

Given : A set *A*, a natural number *k*

With: is–finite[*A*]

Compute : kSub

Such that: kSub = $\{B \in \mathcal{P}[A] \mid |B| = k\}$

Our recursive algorithm works as follows :

If $k = 0$, there can only be one subset of *A*, namely the empty set.

If $k > 0$, we do the following:

If *A* is the empty set, there are of course no *k*-element subsets.

Otherwise we do the following : We take one arbitrary element *a* out of *A*.

We form the union of two sets, namely :

(1) The set of all k–element subsets of *A* with the arbitrary element *a* in it, e.g. the set $\{a\} \cup b$,

where *b* itself is the set of all *k*-1-element subsets of $A \setminus \{a\}$.

(2) The set of all *k*-element subsets of $A \setminus \{a\}$.

Together we computed all *k*-element subsets of *A*, those containing *a* in (1) and those that do not contain *a* in (2).

```
Definition["kSub", any[A, k],
  kSub[A, 0] = {{}}                                                    "base1.a"
  k > 0 ⇒ (kSub[{}, k] = {})                                           "base1.b"
  k > 0 ⋀ k ∈ ℕ ⋀ A ≠ {} ⇒                                             "rec"
    (kSub[A, k] = where[c = æ[A], {{c} ⋃ b | b ∈ kSub[A \ {c}, k − 1]} ⋃ (kSub[A \ {c}, k])])
]
```

Again we are able to use our definition for computation, see the examples below.

```
Compute[kSub[{1, 2, 3, 4, 5}, 4],
  using → ⟨Built–in["Sets"], Built–in["Numbers"], Built–in["Number Domains"],
      Built–in["Connectives"], Built–in["Quantifiers"], Built–in["AllOperators"], Definition["kSub"]⟩]
```

```
{{1, 2, 3, 4}, {1, 2, 3, 5}, {1, 2, 4, 5}, {1, 3, 4, 5}, {2, 3, 4, 5}}
```

```
Compute[kSub[{1, 2, 3, 4}, 0],
  using → ⟨Built–in["Sets"], Built–in["Numbers"], Built–in["Number Domains"],
      Built–in["Connectives"], Built–in["Quantifiers"], Built–in["AllOperators"], Definition["kSub"]⟩]
```

```
{{}}
```

```
Compute[kSub[{}, 2], using → ⟨Built–in["Sets"], Built–in["Numbers"], Built–in["Number Domains"],
      Built–in["Connectives"], Built–in["Quantifiers"], Built–in["AllOperators"], Definition["kSub"]⟩]
```

```
{}
```

We discuss the cardinality of the set generated by the algorithm *kSub*, namely:

```
Theorem["keSS", ∀k∈ℕ ∀is–finite[A] (|kSub[A, k]| = Binomial[|A|, k])]
```

Again we need some additional knowledge in order to prove this theorem.

3.3.1 Lemmata for induction

The first lemma is a simple property of the binomial coefficient. Namely for all $k \in \mathbb{N}$ and $k > 0$ we know, that $\binom{0}{k} = 0$.

```
Lemma["Binomial", any[k], with[k ∈ ℕ ⋀ k > 0], Binomial[0, k] = 0]
```

The second lemma is very similar to the one about disjointness in the two element case. Again we know, that the sets $\{\{æ[A]\} \bigcup b \mid b \in \text{kSub}[A \setminus \{æ[A]\},\ k-1]\}$ and $\text{kSub}[A \setminus \{æ[A]\},\ k]$ must be disjoint. The argument is the same as in the two element case. The first set gives all those subsets containing a certain arbitrary a, whereas the second set gives all the subsets that do not contain a.

```
Lemma["Disjoint sets", any[A, k], with[k > 0 ⋀ k ∈ ℕ ⋀ A ≠ {}],
  {{æ[A]} ⋃ b | b ∈ kSub[A \ {æ[A]}, k – 1]} ⋂ (kSub[A \ {æ[A]}, k]) = {}]
```

We also again need the cardinality property below.

```
Lemma["Cardinality prop", any[A, k], with[k > 0 ⋀ k ∈ ℕ ⋀ A ≠ {}],
  |{{æ[A]} ⋃ b | b ∈ kSub[A \ {æ[A]}, k – 1]}| = |kSub[A \ {æ[A]}, k – 1]|]
```

3.3.2 Theorema Proof

Using those lemmata, we are prepared to prove our desired cardinality theorem. The proof works on induction, both on k and on the cardinality of A.

Note: We implemented also one new inference rule for classical mathematical induction on k. This induction runs from $k-1$ to k during the induction step. This was done in order to be better compatible with our recursive definitions.

The structure of the proof looks as follows:

We want to prove the following: $$\forall_{k\in\mathbb{N}} \; \forall_{\text{is–finite[A]}} \; |\text{kSub[A, k]}| = \binom{|\text{A}|}{\text{k}} \tag{1}$$

We prove by induction over k.

Induction base

Prove: $$\forall_{\text{is–finite[A]}} \; |\text{kSub[A, 0]}| = \binom{|\text{A}|}{0} \tag{2}$$

We prove (2) by induction over the cardinality of A.

Induction base

Prove: $$|\text{kSub}[\{\}, 0]| = \binom{|\{\}|}{0}$$

Induction step

Assume: $$\forall_{A} \left((|A| = n_0) \Rightarrow \left(|\text{kSub}[A, 0]| = \binom{|A|}{0} \right) \right)$$

Prove: $$A_0 \neq \{\} \bigwedge (|A_0| = 1 + n_0) \Rightarrow \left(|\text{kSub}[A_0, 0]| = \binom{|A_0|}{0} \right)$$

Induction step

Assume: $$\forall_{\text{is–finite}[A]} \left(|\text{kSub}[A, k_0 - 1]| = \binom{|A|}{k_0 - 1} \right)$$

Prove: $$\forall_{\text{is–finite}[A]} \left(|\text{kSub}[A, k_0]| = \binom{|A|}{k_0} \right) \tag{3}$$

We prove (3) by induction over the cardinality of A.

Induction base

Prove: $|\text{kSub}[\{\}, k_0]| = \binom{|\{\}|}{k_0}$

Induction step

Assume: $\forall_A \left((|A| = n_1) \Rightarrow \left(|\text{kSub}[A, k_0]| = \binom{|A|}{k_0} \right) \right)$

Prove: $A_0 \neq \{\} \bigwedge (|A_1| = 1 + n_1) \Rightarrow |\text{kSub}[A_1, k_0]| = \binom{|A_1|}{k_0}$

We use some special options for this proof, namely :

GRWTarget → {"goal", "kb"}

For rewriting, we want to mainly focus on our proof goal

AllowIntroduceQuantifiers → True

We allow the prover to introduce new quantifiers during the proof. In fact, this is not really the reason for using this option. In our case, we want the prover to be able to instantiate formulas, which are quantified with more than one variable, step by step.

The full proof of our theorem takes about 5 minutes to finish.

```
Prove[Theorem["keSS"], by → SetTheoryPCSProver, using →
    {Definition["kSub"], Lemma["Binomial"], Lemma["Disjoint sets"], Lemma["Cardinality prop"]},
  built–in → {Built–in["Numbers"]}, SearchDepth → 120, transformBy → ProofSimplifier,
  TransformerOptions → {branches → Proved, steps → Useful},
  ProverOptions → {GRWTarget → {"goal", "kb"}, SimplifyFormula → True, UseCyclicRules → True,
    AllowIntroduceQuantifiers → True, TransformRanges → False}] // Timing
```

```
{303.75, - ProofObject -}
```

Prove:

(Theorem (Double Induction)) $\forall_{k \in \mathbb{N}} \; \forall_{\text{is–finite}[A]} \left(|\text{kSub}[A, k]| = \binom{|A|}{k} \right)$,

under the assumptions:

(Definition (kSub): base1.a) $\forall_A (\text{kSub}[A, 0] = \{\{\}\})$,

(Definition (kSub): base1.b) $\forall_{k} (k > 0 \Rightarrow (\mathrm{kSub}[\{\}, k] = \{\}))$,

(Definition (kSub): rec)

$$\forall_{A,k} \left(k > 0 \wedge k \in \mathbb{N} \wedge A \neq \{\} \Rightarrow \left(\mathrm{kSub}[A, k] = \left\{ \{c\} \cup b \;\middle|\; b \in \mathrm{kSub}[A \setminus \{c\}, k-1] \right\} \cup \mathrm{kSub}[A \setminus \{c\}, k] \,|_{c \leftarrow \mathrm{æ}[A]} \right) \right),$$

(Lemma (Binomial)) $\forall_{\substack{k \\ k \in \mathbb{N} \wedge k > 0}} \left(\binom{0}{k} = 0 \right)$,

(Lemma (Disjoint sets)) $\forall_{\substack{A,k \\ k>0 \wedge k \in \mathbb{N} \wedge A \neq \{\}}} \left(\left\{ \{\mathrm{æ}[A]\} \cup b \;\middle|\; b \in \mathrm{kSub}[A \setminus \{\mathrm{æ}[A]\}, k-1] \right\} \cap (\mathrm{kSub}[A \setminus \{\mathrm{æ}[A]\}, k]) = \{\} \right)$,

(Lemma (Cardinality prop)) $\forall_{\substack{A,k \\ k>0 \wedge k \in \mathbb{N} \wedge A \neq \{\}}} \left(\left| \left\{ \{\mathrm{æ}[A]\} \cup b \;\middle|\; b \in \mathrm{kSub}[A \setminus \{\mathrm{æ}[A]\}, k-1] \right\} \right| = |\mathrm{kSub}[A \setminus \{\mathrm{æ}[A]\}, k-1]| \right)$.

We prove (Theorem (Double Induction)) by induction:

1. Induction base: We have to prove

(1) $\forall_{\mathrm{is\text{–}finite}[A]} \left(|\mathrm{kSub}[A, 0]| = \binom{|A|}{0} \right)$.

We prove (1) by induction over the cardinality of the set A.

1. Induction base: We have to prove

(5) $|\mathrm{kSub}[\{\}, 0]| = \binom{|\{\}|}{0}$.

Using built–in simplification rules we simplify (5) to

(11) $|\mathrm{kSub}[\{\}, 0]| = 1$.

Formula (11), using (Definition (kSub): base1.a), is implied by:

(12) $|\{\{\}\}| = 1$.

Using built–in simplification rules we simplify (12) to

(13) True.

Formula (13) is true because it is the constant True.

2. Induction step: We assume

(6) $n_0 \in \mathbb{N}$,

(7) $\underset{A}{\forall}\left((|A| = n_0) \Rightarrow \left(|\text{kSub}[A, 0]| = \begin{pmatrix} |A| \\ 0 \end{pmatrix}\right)\right)$,

and

(8) $A_0 \neq \{\} \bigwedge (|A_0| = 1 + n_0)$,

and prove

(9) $|\text{kSub}[A_0, 0]| = \begin{pmatrix} |A_0| \\ 0 \end{pmatrix}$

Using built–in simplification rules we simplify (9) to

(16) $|\text{kSub}[A_0, 0]| = 1$.

From what we already know follows:

From (8.1) we know that we can choose an appropriate value such that

(17) $A1_0 \in A_0$.

From (8.2) we can infer

(18) $|A_0 \setminus \{\text{æ}[A_0]\}| = n_0$.

Formula (16), using (Definition (kSub): base1.a), is implied by:

(19) $|\{\{\}\}| = 1$.

Using built–in simplification rules and the additional assumption(s) (18), (8.1), and (8.2) we simplify (19) to

(20) True.

Formula (20) is true because it is the constant True.

2. Induction step: We assume

(2) $k_0 \in \mathbb{N} \bigwedge k_0 > 0$,

and

(3) $\underset{\text{is–finite}[A]}{\forall}\left(|\text{kSub}[A, k_0 - 1]| = \begin{pmatrix} |A| \\ k_0 - 1 \end{pmatrix}\right)$,

and prove

(4) $\underset{\text{is–finite}[A]}{\forall}\left(|\text{kSub}[A, k_0]| = \begin{pmatrix} |A| \\ k_0 \end{pmatrix}\right)$

From (3) we obtain

(21) $\forall_A \left(|\mathrm{kSub}[A, k_0 - 1]| = \binom{|A|}{k_0 - 1} \right)$.

We prove (4) by induction over the cardinality of the set A.

1. Induction base: We have to prove

(22) $|\mathrm{kSub}[\{\}, k_0]| = \binom{|\{\}|}{k_0}$.

Using built–in simplification rules we simplify (22) to

(29) $|\mathrm{kSub}[\{\}, k_0]| = \binom{0}{k_0}$.

Formula (2.1), by (Lemma (Binomial)), implies:

(33) $k_0 > 0 \Rightarrow \left(\binom{0}{k_0} = 0 \right)$.

From (2.2) and (33) we obtain by modus ponens

(34) $\binom{0}{k_0} = 0$.

Formula (29), using (34), is implied by:

(35) $|\mathrm{kSub}[\{\}, k_0]| = 0$.

Formula (2.2), by (Definition (kSub): base1.b), implies:

(42) $\mathrm{kSub}[\{\}, k_0] = \{\}$.

Formula (35), using (42), is implied by:

(44) $|\{\}| = 0$.

Using built–in simplification rules and the additional assumption(s) (42), (34), and (2.2) we simplify (44) to

(45) True.

Formula (45) is true because it is the constant True.

2. Induction step: We assume

(23) $n_1 \in \mathbb{N}$,

(24) $\forall_A \left((|A| = n_1) \Rightarrow \left(|\mathrm{kSub}[A, k_0]| = \binom{|A|}{k_0} \right) \right)$,

and

(25) $A_1 \neq \{\} \bigwedge (|A_1| = 1 + n_1)$,

and prove

(26) $|\mathrm{kSub}[A_1, k_0]| = \binom{|A_1|}{k_0}$

Using built–in simplification rules we can simplify the knowledge base:

Formula (21) simplifies to

(46) $\underset{A}{\forall}\left(|\mathrm{kSub}[A, -1 + k_0]| = \binom{|A|}{-1 + k_0}\right)$.

Formula (Lemma (Cardinality prop)) simplifies to

(47)

$$\underset{A,k}{\forall}\left(k > 0 \bigwedge k \in \mathbb{N} \bigwedge A \neq \{\} \Rightarrow \left(\left|\left\{\{æ[A]\} \bigcup b \underset{b}{\mid} b \in \mathrm{kSub}[A \setminus \{æ[A]\}, k - 1]\right\}\right| = |\mathrm{kSub}[A \setminus \{æ[A]\}, -1 + k]|\right)\right).$$

From what we already know follows:

From (25.1) we know that we can choose an appropriate value such that

(48) $A2_0 \in A_1$.

From (25.2) we can infer

(49) $|A_1 \setminus \{æ[A_1]\}| = n_1$.

Formula (26), using (25.2), is implied by:

(50) $|\mathrm{kSub}[A_1, k_0]| = \binom{1 + n_1}{k_0}$.

Formula (49), by (24), implies:

(51) $|\mathrm{kSub}[A_1 \setminus \{æ[A_1]\}, k_0]| = \binom{|A_1 \setminus \{æ[A_1]\}|}{k_0}$.

Formula (51), by (49), implies:

(52) $|\mathrm{kSub}[A_1 \setminus \{æ[A_1]\}, k_0]| = \binom{n_1}{k_0}$.

Formula (2.1), by (Definition (kSub): rec), implies:

(63)

$$\underset{A}{\forall}\left(k_0 > 0 \Rightarrow \left(A \neq \{\} \Rightarrow \left(\mathrm{kSub}[A, k_0] = \left\{\{c\} \bigcup b \underset{b}{\mid} b \in \mathrm{kSub}[A \setminus \{c\}, k_0 - 1]\right\} \bigcup \mathrm{kSub}[A \setminus \{c\}, k_0]\ |_{c \leftarrow æ[A]}\right)\right)\right).$$

Formula (63) is simplified to:

(64)

$$k_0 > 0 \Rightarrow \underset{A}{\forall}\left(A \neq \{\} \Rightarrow \left(\mathrm{kSub}[A, k_0] = \left\{\{c\} \cup b \underset{b}{\mid} b \in \mathrm{kSub}[A \setminus \{c\}, k_0 - 1]\right\} \cup \mathrm{kSub}[A \setminus \{c\}, k_0] \mid_{c \leftarrow æ[A]}\right)\right).$$

From (2.2) and (64) we obtain by modus ponens

(65) $$\underset{A}{\forall}\left(A \neq \{\} \Rightarrow \left(\mathrm{kSub}[A, k_0] = \left\{\{c\} \cup b \underset{b}{\mid} b \in \mathrm{kSub}[A \setminus \{c\}, k_0 - 1]\right\} \cup \mathrm{kSub}[A \setminus \{c\}, k_0] \mid_{c \leftarrow æ[A]}\right)\right).$$

Formula (25.1), by (65), implies:

(66) $$\mathrm{kSub}[A_1, k_0] = \left\{\{c\} \cup b \underset{b}{\mid} b \in \mathrm{kSub}[A_1 \setminus \{c\}, k_0 - 1]\right\} \cup \mathrm{kSub}[A_1 \setminus \{c\}, k_0] \mid_{c \leftarrow æ[A_1]}.$$

Formula (50), using (66), is implied by:

(67) $$\left|\left\{\{c\} \cup b \underset{b}{\mid} b \in \mathrm{kSub}[A_1 \setminus \{c\}, k_0 - 1]\right\} \cup \mathrm{kSub}[A_1 \setminus \{c\}, k_0] \mid_{c \leftarrow æ[A_1]}\right| = \binom{1 + n_1}{k_0}.$$

By simplification we obtain from (67) the new goal:

(68) $$\left|\left\{\{æ[A_1]\} \cup b \underset{b}{\mid} b \in \mathrm{kSub}[A_1 \setminus \{æ[A_1]\}, k_0 - 1]\right\} \cup \mathrm{kSub}[A_1 \setminus \{æ[A_1]\}, k_0]\right| = \binom{1 + n_1}{k_0}.$$

Formula (2.1), by (Lemma (Binomial)), implies:

(69) $$k_0 > 0 \Rightarrow \left(\binom{0}{k_0} = 0\right).$$

From (2.2) and (69) we obtain by modus ponens

(70) $$\binom{0}{k_0} = 0.$$

Formula (2.1), by (Lemma (Disjoint sets)), implies:

(71)

$$\underset{A}{\forall}\left(k_0 > 0 \Rightarrow \left(A \neq \{\} \Rightarrow \left(\left\{\{æ[A]\} \cup b \underset{b}{\mid} b \in \mathrm{kSub}[A \setminus \{æ[A]\}, k_0 - 1]\right\} \cap (\mathrm{kSub}[A \setminus \{æ[A]\}, k_0]) = \{\}\right)\right)\right).$$

Formula (71) is simplified to:

(72)

$$k_0 > 0 \Rightarrow \underset{A}{\forall}\left(A \neq \{\} \Rightarrow \left(\left\{\{æ[A]\} \bigcup b \underset{b}{\mid} b \in \text{kSub}[A \setminus \{æ[A]\}, k_0 - 1]\right\} \bigcap (\text{kSub}[A \setminus \{æ[A]\}, k_0]) = \{\}\right)\right).$$

From (2.2) and (72) we obtain by modus ponens

(73) $$\underset{A}{\forall}\left(A \neq \{\} \Rightarrow \left(\left\{\{æ[A]\} \bigcup b \underset{b}{\mid} b \in \text{kSub}[A \setminus \{æ[A]\}, k_0 - 1]\right\} \bigcap (\text{kSub}[A \setminus \{æ[A]\}, k_0]) = \{\}\right)\right).$$

Formula (25.1), by (73), implies:

(74) $$\left\{\{æ[A_1]\} \bigcup b \underset{b}{\mid} b \in \text{kSub}[A_1 \setminus \{æ[A_1]\}, k_0 - 1]\right\} \bigcap (\text{kSub}[A_1 \setminus \{æ[A_1]\}, k_0]) = \{\}.$$

In order to prove (68), due to (74), it is sufficient to prove

(75) $$\left|\left\{\{æ[A_1]\} \bigcup b \underset{b}{\mid} b \in \text{kSub}[A_1 \setminus \{æ[A_1]\}, k_0 - 1]\right\}\right| + |\text{kSub}[A_1 \setminus \{æ[A_1]\}, k_0]| = \binom{1 + n_1}{k_0}.$$

Using built–in simplification rules and the additional assumption(s) (70), (66), (52), (49), (25.1), (25.2), and (2.2) we simplify (75) to

(76) $$|\text{kSub}[A_1 \setminus \{æ[A_1]\}, k_0]| + \left|\left\{\{æ[A_1]\} \bigcup b \underset{b}{\mid} b \in \text{kSub}[A_1 \setminus \{æ[A_1]\}, k_0 - 1]\right\}\right| = \binom{1 + n_1}{k_0}.$$

Formula (76), using (52), is implied by:

(77) $$\binom{n_1}{k_0} + \left|\left\{\{æ[A_1]\} \bigcup b \underset{b}{\mid} b \in \text{kSub}[A_1 \setminus \{æ[A_1]\}, k_0 - 1]\right\}\right| = \binom{1 + n_1}{k_0}.$$

Formula (2.1), by (47), implies:

(78)

$$\underset{A}{\forall}\left(k_0 > 0 \Rightarrow \left(A \neq \{\} \Rightarrow \left(\left|\left\{\{æ[A]\} \bigcup b \underset{b}{\mid} b \in \text{kSub}[A \setminus \{æ[A]\}, k_0 - 1]\right\}\right| = |\text{kSub}[A \setminus \{æ[A]\}, -1 + k_0]|\right)\right)\right).$$

Formula (78) is simplified to:

(79)

$$k_0 > 0 \Rightarrow \underset{A}{\forall}\left(A \neq \{\} \Rightarrow \left(\left|\left\{\{æ[A]\} \bigcup b \underset{b}{\mid} b \in \text{kSub}[A \setminus \{æ[A]\}, k_0 - 1]\right\}\right| = |\text{kSub}[A \setminus \{æ[A]\}, -1 + k_0]|\right)\right).$$

From (2.2) and (79) we obtain by modus ponens

(80) $$\underset{A}{\forall}\left(A \neq \{\} \Rightarrow \left(\left|\left\{\{æ[A]\} \bigcup b \underset{b}{\mid} b \in \text{kSub}[A \setminus \{æ[A]\}, k_0 - 1]\right\}\right| = |\text{kSub}[A \setminus \{æ[A]\}, -1 + k_0]|\right)\right).$$

Formula (25.1), by (80), implies:

$$\left|\left\{\{æ[A_1]\} \cup b \mid_b b \in \text{kSub}[A_1 \setminus \{æ[A_1]\}, k_0 - 1]\right\}\right| = |\text{kSub}[A_1 \setminus \{æ[A_1]\}, -1 + k_0]|,$$

which, by (46), implies:

$$\left|\left\{\{æ[A_1]\} \cup b \mid_b b \in \text{kSub}[A_1 \setminus \{æ[A_1]\}, k_0 - 1]\right\}\right| = \binom{|A_1 \setminus \{æ[A_1]\}|}{-1 + k_0},$$

which, by (49), implies:

(81) $$\left|\left\{\{æ[A_1]\} \cup b \mid_b b \in \text{kSub}[A_1 \setminus \{æ[A_1]\}, k_0 - 1]\right\}\right| = \binom{n_1}{-1 + k_0}.$$

Formula (77), using (81), is implied by:

(82) $$\binom{n_1}{k_0} + \binom{n_1}{-1 + k_0} = \binom{1 + n_1}{k_0}.$$

Using built–in simplification rules and the additional assumption(s) (81), (70), (66), (52), (49), (25.1), (25.2), and (2.2) we simplify (82) to

(83) True.

Formula (83) is true because it is the constant True.

□

3.3.3 Notes

The interesting part of this proof starts with the formulae (23), (24), (25) and (26). This is the branch of the proof, where we are in the induction step, both on k and on $|A|$.

We built up (46), the induction hypothesis (for k). In (47) we instantiate our lemma *card prop*. (49) is built again with our inference rule *ArbitraryElementn1b* from (25). We simplify our goal to (50) and then get the induction hypothesis (now for |A|), (52).

We then instantiate the algorithm definition rec and simplify to (66).

We plug our definition into the goal (50) and get after simplification (68). Together with our lemma *disjoint sets* in (71), (72), (73) and (74) we are able to split the cardinality of the union in (74) with our inference rule *dis* and get finally (76).

With the knowledge in (78), (79) and (80), the instantiated and combined lemmata and induction hypotheses, we end up with (82). This property if the binomial coefficient is proven by simplification.

3.3.4 Corollary

With the help of the just proved theorem and the famous binomial theorem we are able to prove the following statement:

$$\forall_{\text{is–finite}[A]} \quad |\mathcal{P}[A]| = 2^{|A|}$$

Proof: We use the binomial theorem $(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k\, b^{n-k}$ for $a = b = 1$ and get:

$$2^n = \sum_{k=0}^{n} \binom{n}{k}$$

Since we know, that the number of all k–element subset of an n–element set is exactly $\binom{n}{k}$ and we know, that there can not be subsets with more than n elements, we can conclude, that the number of all subsets of an n–element set is exactly 2^n. E.g. we are done.

3.4 The correctness of algorithm S2

In this section we discuss the correctness of the algorithm *S2*. This algorithm computes all the two element subsets of a given finite set. A detailed explanation of this algorithm is given in Section 3.2.

```
Theorem["CorrS2", ∀_is–finite[A] (S2[A] = {B ∈ 𝒫[A] | (|B| = 2)})]
```

To structure the proof and make it work automatically, we introduce two lemmata. We also prove them with the help of the SetTheoryPCSProver in *Theorema*.

3.4.1 Lemmata for induction

The first lemma speaks about the equality of two sets. For all nonempty A, the sets

$\{\{æ[A],\ b\}\ |\ b\ \in\ A \setminus \{æ[A]\}\}$ and $\{B \in \mathcal{P}[A]\ |\ æ[A]\ \in\ B\ \bigwedge\ (|B|\ =\ 2)\}$ are equal.

```
Lemma["1", any[A], with[A ≠ {}],
  {{æ[A], b} | b ∈ A \ {æ[A]}} = {B ∈ 𝒫[A] | æ[A] ∈ B ⋀ (|B| = 2)}]

Prove[Lemma["1"], by → SetTheoryPCSProver, transformBy → ProofSimplifier,
    TransformerOptions → {branches → Proved, steps → Useful},
    built–in → {Built–in["Numbers"]}, SearchDepth → 70,
    ProverOptions → {DisableInferenceRule → {"KBComposeIntersection", "KBInferNonEmpty"},
        ChooseFromFiniteSet → True, SimplifyFormula → True, SIPCardinv → True}] // Timing

{25.563, - ProofObject -}
```

Prove:

(Lemma (1)) $\forall_A \left(A \neq \{\} \Rightarrow \left(\left\{ \{æ[A], b\} \mid_b b \in A \setminus \{æ[A]\} \right\} = \left\{ B \mid_B (æ[A] \in B \bigwedge (|B| = 2)) \bigwedge B \in \mathcal{P}[A] \right\} \right) \right)$,

with no assumptions.

We assume

(1) $A_0 \neq \{\}$,

and show

(2) $\left\{ \{æ[A_0], b\} \mid_b b \in A_0 \setminus \{æ[A_0]\} \right\} = \left\{ B \mid_B (æ[A_0] \in B \bigwedge (|B| = 2)) \bigwedge B \in \mathcal{P}[A_0] \right\}$.

We show (2) by mutual inclusion:

$\subseteq$: We assume

(4) $b1_0 \in \left\{ \{æ[A_0], b\} \mid_{b} b \in A_0 \setminus \{æ[A_0]\} \right\}$

and show:

(5) $(æ[A_0] \in b1_0 \bigwedge (|b1_0| = 2)) \bigwedge b1_0 \in \mathcal{P}[A_0]$.

From what we already know follows:

From (4) we know by definition of $\left\{ T_x \mid_{x} P \right\}$ that we can choose an appropriate value such that

(6) $b2_0 \in A_0 \setminus \{æ[A_0]\}$,

(7) $b1_0 = \{æ[A_0], b2_0\}$.

From what we already know follows:

From (6) we can infer

(8) $b2_0 \in A_0$,

(9) $b2_0 \notin \{æ[A_0]\}$.

From what we already know follows:

From (9) we can infer

(10) $b2_0 \neq æ[A_0]$.

Using built–in simplification rules and the additional assumption(s) (10), (9), (7), and (1) we simplify (5) to

(11) $æ[A_0] \in b1_0 \bigwedge (|b1_0| = 2) \bigwedge b1_0 \in \mathcal{P}[A_0]$.

We prove the individual conjunctive parts of (11):

Proof of (11.1) $æ[A_0] \in b1_0$:

Formula (11.1), using (7), is implied by:

(12) $æ[A_0] \in \{æ[A_0], b2_0\}$.

Using built–in simplification rules and the additional assumption(s) (10), (9), (7), and (1) we simplify (12) to

(13) True.

Formula (13) is true because it is the constant True.

Proof of (11.2) $|b1_0| = 2$:

Formula (11.2), using (7), is implied by:

(14) $|\{æ[A_0], b2_0\}| = 2$.

In order to prove (14), it is sufficient to prove

(15) $æ[A_0] \neq b2_0$.

Using built–in simplification rules and the additional assumption(s) (10), (9), (7), and (1) we simplify (15) to

(16) $æ[A_0] \neq b2_0$.

We prove (16) by contradiction.

We assume

(17) $æ[A_0] = b2_0$,

and show a contradiction.

Formula (10), by (17), implies:

(18) $b2_0 \neq b2_0$.

Formula (a contradiction) holds because the assumption (18) contains a contradiction.

Proof of (11.3) $b1_0 \in \mathcal{P}[A_0]$:

For proving (11.3) we choose

(19) $b11_0 \in b1_0$,

and show:

(20) $b11_0 \in A_0$.

Formula (19), by (7), implies:

(21) $b11_0 \in \{æ[A_0], b2_0\}$.

From what we already know follows:

From (21) we can infer

(22) $(b11_0 = æ[A_0]) \bigvee (b11_0 = b2_0)$.

We prove (20) by case distinction using (22).

Case (22.1) $b11_0 = æ[A_0]$:

From what we already know follows:

From (22.1) we can infer

(23) $b11_0 \in A_0$.

Formula (20) is true because it is identical to (23).

Case (22.2) $b11_0 = b2_0$:

Formula (20), using (22.2), is implied by:

(26) $b2_0 \in A_0$.

Formula (26) is true because it is identical to (8).

$\supseteq$: Now we assume

(5) $(æ[A_0] \in b1_0 \bigwedge (|b1_0| = 2)) \bigwedge b1_0 \in \mathcal{P}[A_0]$

and show:

(4) $b1_0 \in \left\{\{æ[A_0], b\} \underset{b}{\mid} b \in A_0 \setminus \{æ[A_0]\}\right\}$.

Using built–in simplification rules we can simplify the knowledge base:

Formula (5) simplifies to

(27) $æ[A_0] \in b1_0 \bigwedge (|b1_0| = 2) \bigwedge b1_0 \in \mathcal{P}[A_0]$.

In order to prove (4) we have to show

(28) $\underset{b}{\exists} (b \in A_0 \setminus \{æ[A_0]\} \bigwedge (b1_0 = \{æ[A_0], b\}))$.

From (27.1) and (27.2) we obtain:

(29) $b1_0 = \{æ[A_0], z1\}$,

and

(30) $\{æ[A_0]\} \bigcap \{z1\} = \{\}$.

Using built–in simplification rules we can simplify the knowledge base:

Formula (29) simplifies to

(31) $b1_0 = \{æ[A_0], z1\}$.

Formula (30) simplifies to

(32) $\{æ[A_0]\} \bigcap \{z1\} = \{\}$.

From what we already know follows:

From (32) we can infer

(33) $\underset{z11}{\forall} (z11 \notin \{æ[A_0]\} \bigcap \{z1\})$.

From (27.3) we can infer

(34) $b1_0 \subseteq A_0$.

Now, let $b := z1$. Thus, for proving (28) it is sufficient to prove:

(37) $z1 \in A_0 \setminus \{æ[A_0]\} \bigwedge (b1_0 = \{æ[A_0], z1\})$.

Using built–in simplification rules and the additional assumption(s) (31), (32), and (1) we simplify (37) to

(38) $z1 \in A_0 \setminus \{æ[A_0]\}$.

We have to prove (38), thus, we first show:

(39) $z1 \in A_0$:

Formula (34), by (31), implies:

(41) $\{æ[A_0], z1\} \subseteq A_0$.

From what we already know follows:

From (41) we can infer

(42) $\bigwedge_{z12} (z12 \in \{æ[A_0], z1\} \Rightarrow z12 \in A_0)$.

Formula (39), using (42), is implied by:

(46) $z1 \in \{æ[A_0], z1\}$.

Using built–in simplification rules and the additional assumption(s) (31), (32), and (1) we simplify (46) to

(47) True.

Formula (47) is true because it is the constant True.

For proving (38) it still remains to show

(40) $z1 \notin \{æ[A_0]\}$:

In order to prove (40), it is sufficient to prove

(48) $z1 \neq æ[A_0]$

We prove (48) by contradiction.

We assume

(49) $z1 = æ[A_0]$,

and show a contradiction.

Formula (32), by (49), implies:

(57) $\{æ[A_0]\} \bigcap \{æ[A_0]\} = \{\}$.

Using built–in simplification rules we can simplify the knowledge base:

Formula (57) simplifies to

(58) False.

Formula (a contradiction) is true because the assumption (58) is false.

□

Notes

This proof is done by standard set theory proving. One of our implemented inference rules is applied in (29) and (30), our rule *card2*.

The second lemma again speaks about two sets being equal for all nonempty A, namely

$\{B \in \mathcal{P}[A \setminus \{æ[A]\}] \mid (|B| = 2)\}$ and $\{B \in \mathcal{P}[A] \mid æ[A] \notin B \bigwedge (|B| = 2)\}$.

```
Lemma["2", any[A], with[A ≠ {}],
  {B ∈ 𝒫[A \ {æ[A]}] | (|B| = 2)} = {B ∈ 𝒫[A] | æ[A] ∉ B ⋀ (|B| = 2)}]

Prove[Lemma["2"], by → SetTheoryPCSProver, transformBy → ProofSimplifier,
    TransformerOptions → {branches → Proved, steps → Useful},
    built–in → {Built–in["Numbers"]}, SearchDepth → 70,
    ProverOptions → {SimplifyFormula → True}]  // Timing
{7.984, - ProofObject -}
```

Prove:

(Lemma (2))

$$\forall_A \left(A \neq \{\} \Rightarrow \left(\left\{ B \mid_B B \in \mathcal{P}[A \setminus \{æ[A]\}] \bigwedge (|B| = 2) \right\} = \left\{ B \mid_B (æ[A] \notin B \bigwedge (|B| = 2)) \bigwedge B \in \mathcal{P}[A] \right\} \right) \right),$$

with no assumptions.

We assume

(1) $A_0 \neq \{\}$,

and show

(2) $\left\{ B \mid_B B \in \mathcal{P}[A_0 \setminus \{æ[A_0]\}] \bigwedge (|B| = 2) \right\} = \left\{ B \mid_B (æ[A_0] \notin B \bigwedge (|B| = 2)) \bigwedge B \in \mathcal{P}[A_0] \right\}$.

We show (2) by mutual inclusion:

$\subseteq$: We assume

(4) $B1_0 \in \mathcal{P}[A_0 \setminus \{æ[A_0]\}] \bigwedge (|B1_0| = 2)$

and show:

(5) $(æ[A_0] \notin B1_0 \bigwedge (|B1_0| = 2)) \bigwedge B1_0 \in \mathcal{P}[A_0]$.

Using built–in simplification rules and the additional assumption(s) (1) we simplify (5) to

(6) $æ[A_0] \notin B1_0 \bigwedge (|B1_0| = 2) \bigwedge B1_0 \in \mathcal{P}[A_0]$.

From what we already know follows:

From (4.1) we can infer

(7) $B1_0 \subseteq A_0 \setminus \{æ[A_0]\}$.

From what we already know follows:

From (7) we can infer

(8) $\underset{B11}{\forall} (B11 \in B1_0 \Rightarrow B11 \in A_0 \setminus \{æ[A_0]\})$.

Using built–in simplification rules and the additional assumption(s) (4.2)and (1) we simplify (6) to

(9) $æ[A_0] \notin B1_0 \bigwedge B1_0 \in \mathcal{P}[A_0]$.

We prove the individual conjunctive parts of (9):

Proof of (9.1) $æ[A_0] \notin B1_0$:

We prove (9.1) by contradiction.

We assume

(10) $æ[A_0] \in B1_0$,

and show a contradiction.

Formula (10), by (8), implies:

(11) $æ[A_0] \in A_0 \setminus \{æ[A_0]\}$.

From what we already know follows:

From (11) we can infer

(12) $æ[A_0] \in A_0$,

(13) $æ[A_0] \notin \{æ[A_0]\}$.

Using built–in simplification rules we can simplify the knowledge base:

Formula (13) simplifies to

(14) False.

Formula (a contradiction) is true because the assumption (14) is false.

Proof of (9.2) $B1_0 \in \mathcal{P}[A_0]$:

For proving (9.2) we choose

(15) $B12_0 \in B1_0$,

and show:

(16) $B12_0 \in A_0$.

Formula (15), by (8), implies:

(17) $B12_0 \in A_0 \setminus \{æ[A_0]\}$.

From what we already know follows:

From (17) we can infer

(18) $B12_0 \in A_0$,

(19) $B12_0 \notin \{æ[A_0]\}$.

Formula (16) is true because it is identical to (18).

$\supseteq$: Now we assume

(5) $(æ[A_0] \notin B1_0 \bigwedge (|B1_0| = 2)) \bigwedge B1_0 \in \mathcal{P}[A_0]$

and show:

(4) $B1_0 \in \mathcal{P}[A_0 \setminus \{æ[A_0]\}] \bigwedge (|B1_0| = 2)$.

Using built–in simplification rules we can simplify the knowledge base:

Formula (5) simplifies to

(20) $æ[A_0] \notin B1_0 \bigwedge (|B1_0| = 2) \bigwedge B1_0 \in \mathcal{P}[A_0]$.

From what we already know follows:

From (20.3) we can infer

(21) $B1_0 \subseteq A_0$.

From what we already know follows:

From (21) we can infer

(22) $\underset{B13}{\forall} (B13 \in B1_0 \Rightarrow B13 \in A_0)$.

Using built–in simplification rules and the additional assumption(s) (20.1), (20.2), and (1) we simplify (4) to

(23) $B1_0 \in \mathcal{P}[A_0 \setminus \{æ[A_0]\}]$.

For proving (23) we choose

(24) $B14_0 \in B1_0$,

and show:

(25) $B14_0 \in A_0 \setminus \{æ[A_0]\}$.

We have to prove (25), thus, we first show:

(26) $B14_0 \in A_0$:

Formula (24), by (22), implies:

(28) $B14_0 \in A_0$.

Formula (26) is true because it is identical to (28).

For proving (25) it still remains to show

(27) $B14_0 \notin \{æ[A_0]\}$:

In order to prove (27), it is sufficient to prove

(29) $B14_0 \neq æ[A_0]$

We prove (29) by contradiction.

We assume

(30) $B14_0 = æ[A_0]$,

and show a contradiction.

Formula (24), by (30), implies:

(33) $æ[A_0] \in B1_0$.

Formula (a contradiction) is proved because (33) and (20.1) are contradictory.

□

Notes

This proof is done by standard set theory proving.

3.4.2 Theorema Proof

So we are now able to prove our correctness theorem. It takes less than 3 minutes to finish.

```
Prove[Theorem["CorrS2"], by → SetTheoryPCSProver,
    transformBy → ProofSimplifier, TransformerOptions → {branches → Proved, steps → Useful},
    using → {Definition["S2"], Lemma["1"], Lemma["2"]}, SearchDepth → 120,
    ProverOptions → {EarlyRewriting → True, RWSetOperators → True, GRWTarget → {"kb", "goal"},
        UseCyclicRules → True, TransformRanges → False, SimplifyFormula → True}] // Timing
{145.547, - ProofObject -}
```

Prove:

(Theorem (CorrS2)) $\underset{\text{is–finite}[A]}{\forall}\left(\text{S2}[A] = \left\{B \;\middle|\; |B| = 2\right\}_{B\in\mathcal{P}[A]}\right)$,

under the assumptions:

(Definition (S2): base) $\text{S2}[\{\}] = \{\}$,

(Definition (S2): rec) $\underset{A}{\forall}\left(A \neq \{\} \Rightarrow \left(\text{S2}[A] = \left\{\{a, b\} \;\middle|\; b\in A\setminus\{a\}\right\} \cup \text{S2}[A\setminus\{a\}]\,|_{a\leftarrow æ[A]}\right)\right)$,

(Lemma (1)) $\underset{\substack{A\\A\neq\{\}}}{\forall}\left(\left\{\{æ[A], b\} \;\middle|\; b\in A\setminus\{æ[A]\}\right\} = \left\{B \;\middle|\; B\in\mathcal{P}[A]\ æ[A]\in B \wedge (|B| = 2)\right\}\right)$,

(Lemma (2)) $\underset{\substack{A\\A\neq\{\}}}{\forall}\left(\left\{B \;\middle|\; B\in\mathcal{P}[A\setminus\{æ[A]\}]\ |B| = 2\right\} = \left\{B \;\middle|\; B\in\mathcal{P}[A]\ æ[A]\notin B \wedge (|B| = 2)\right\}\right)$.

We prove (Theorem (CorrS2)) by induction over the cardinality of the set A.

1. Induction base: We have to prove

(1) $\text{S2}[\{\}] = \left\{B \;\middle|\; B\in\mathcal{P}[\{\}] \wedge (|B| = 2)\right\}_{B}$.

Using built–in simplification rules and the additional assumption(s) (Definition (S2): base) we simplify (1) to

(7) $\text{S2}[\{\}] = \left\{B \;\middle|\; B\in\mathcal{P}[\{\}] \wedge (|B| = 2)\right\}_{B}$.

Formula (7), using (Definition (S2): base), is implied by:

(8) $\{\} = \left\{B \;\middle|\; B\in\mathcal{P}[\{\}] \wedge (|B| = 2)\right\}_{B}$.

We have to prove (8), hence, we have to show:

(9) $\neg\,(B_0 \in \mathcal{P}[\{\}] \bigwedge (|B_0| = 2))$.

Using built–in simplification rules and the additional assumption(s) (Definition (S2): base) we simplify (9) to

(10) $\neg\,(B_0 \in \{\{\}\} \bigwedge (|B_0| = 2))$.

We prove (10) by contradiction.

We assume

(11) $B_0 \in \{\{\}\} \bigwedge (|B_0| = 2)$,

and show a contradiction.

From what we already know follows:

From (11.1) we can infer

(12) $B_0 = \{\}$.

Formula (11.2), by (12), implies:

(16) $|\{\}| = 2$.

Using built–in simplification rules we can simplify the knowledge base:

Formula (16) simplifies to

(17) False.

Formula (a contradiction) is true because the assumption (17) is false.

2. Induction step: We assume

(2) $n_0 \in \mathbb{N}$,

(3) $\underset{A}{\forall}\left((|A| = n_0) \Rightarrow \left(\mathrm{S2}[A] = \left\{B \underset{B}{\mid} B \in \mathcal{P}[A] \bigwedge (|B| = 2)\right\}\right)\right)$,

and

(4) $A_0 \neq \{\} \bigwedge (|A_0| = 1 + n_0)$,

and prove

(5) $\mathrm{S2}[A_0] = \left\{B \underset{B}{\mid} B \in \mathcal{P}[A_0] \bigwedge (|B| = 2)\right\}$

From what we already know follows:

From (4.1) we know that we can choose an appropriate value such that

(19) $A1_0 \in A_0$.

From (4.2) we can infer

(20) $|A_0 \setminus \{æ[A_0]\}| = n_0$.

Formula (20), by (3), implies:

(21) $S2[A_0 \setminus \{æ[A_0]\}] = \left\{B \mid_B B \in \mathcal{P}[A_0 \setminus \{æ[A_0]\}] \wedge (|B| = 2)\right\}$.

Formula (4.1), by (Definition (S2): rec), implies:

(22) $S2[A_0] = \left\{\{a, b\} \mid_b b \in A_0 \setminus \{a\}\right\} \cup S2[A_0 \setminus \{a\}] \,|_{a \leftarrow æ[A_0]}$.

Formula (4.1), by (Lemma (1)), implies:

(23) $\left\{\{æ[A_0], b\} \mid_b b \in A_0 \setminus \{æ[A_0]\}\right\} = \left\{B \mid_B B \in \mathcal{P}[A_0] \wedge æ[A_0] \in B \wedge (|B| = 2)\right\}$.

Formula (4.1), by (Lemma (2)), implies:

(24) $\left\{B \mid_B B \in \mathcal{P}[A_0 \setminus \{æ[A_0]\}] \wedge (|B| = 2)\right\} = \left\{B \mid_B B \in \mathcal{P}[A_0] \wedge (|B| = 2) \wedge æ[A_0] \notin B\right\}$.

Formula (21), by (24), implies:

(30) $S2[A_0 \setminus \{æ[A_0]\}] = \left\{B \mid_B B \in \mathcal{P}[A_0] \wedge (|B| = 2) \wedge æ[A_0] \notin B\right\}$.

Formula (5), using (22), is implied by:

(31) $\left\{\{a, b\} \mid_b b \in A_0 \setminus \{a\}\right\} \cup S2[A_0 \setminus \{a\}] \,|_{a \leftarrow æ[A_0]} = \left\{B \mid_B B \in \mathcal{P}[A_0] \wedge (|B| = 2)\right\}$.

By simplification we obtain from (31) the new goal:

(32) $\left\{\{æ[A_0], b\} \mid_b b \in A_0 \setminus \{æ[A_0]\}\right\} \cup S2[A_0 \setminus \{æ[A_0]\}] = \left\{B \mid_B B \in \mathcal{P}[A_0] \wedge (|B| = 2)\right\}$.

Formula (32), using (30), is implied by:

$$\left\{\{æ[A_0], b\} \mid_b b \in A_0 \setminus \{æ[A_0]\}\right\} \cup \left\{B2 \mid_{B2} B2 \in \mathcal{P}[A_0] \wedge (|B2| = 2) \wedge æ[A_0] \notin B2\right\} =,$$
$$\left\{B \mid_B B \in \mathcal{P}[A_0] \wedge (|B| = 2)\right\}$$

which, using (23), is implied by:

(33)

$$\left\{B3 \underset{B3}{\mid} B3 \in \mathcal{P}[A_0] \wedge æ[A_0] \in B3 \wedge (|B3| = 2)\right\} \cup \left\{B2 \underset{B2}{\mid} B2 \in \mathcal{P}[A_0] \wedge (|B2| = 2) \wedge æ[A_0] \notin B2\right\} =.$$
$$\left\{B \underset{B}{\mid} B \in \mathcal{P}[A_0] \wedge (|B| = 2)\right\}$$

We show (33) by mutual inclusion:

$\subseteq$: We assume

(34)

$$B31_0 \in \left\{B3 \underset{B3}{\mid} B3 \in \mathcal{P}[A_0] \wedge æ[A_0] \in B3 \wedge (|B3| = 2)\right\} \cup \left\{B2 \underset{B2}{\mid} B2 \in \mathcal{P}[A_0] \wedge (|B2| = 2) \wedge æ[A_0] \notin B2\right\}$$

and show:

(35) $B31_0 \in \mathcal{P}[A_0] \wedge (|B31_0| = 2)$.

From what we already know follows:

From (34) we can infer

(36) $B31_0 \in \left\{B3 \underset{B3}{\mid} B3 \in \mathcal{P}[A_0] \wedge æ[A_0] \in B3 \wedge (|B3| = 2)\right\} \bigvee .$
$B31_0 \in \left\{B2 \underset{B2}{\mid} B2 \in \mathcal{P}[A_0] \wedge (|B2| = 2) \wedge æ[A_0] \notin B2\right\}$

We prove the individual conjunctive parts of (35):

Proof of (35.1) $B31_0 \in \mathcal{P}[A_0]$:

We prove (35.1) by case distinction using (36).

Case (36.1) $B31_0 \in \left\{B3 \underset{B3}{\mid} B3 \in \mathcal{P}[A_0] \wedge æ[A_0] \in B3 \wedge (|B3| = 2)\right\}$:

From what we already know follows:

From (36.1) we can infer

(37) $B31_0 \in \mathcal{P}[A_0] \wedge æ[A_0] \in B31_0 \wedge (|B31_0| = 2)$.

Formula (35.1) is true because it is identical to (37.1).

Case (36.2) $B31_0 \in \left\{B2 \underset{B2}{\mid} B2 \in \mathcal{P}[A_0] \wedge (|B2| = 2) \wedge æ[A_0] \notin B2\right\}$:

From what we already know follows:

From (36.2) we can infer

(38) $B31_0 \in \mathcal{P}[A_0] \wedge (|B31_0| = 2) \wedge æ[A_0] \notin B31_0$.

Formula (35.1) is true because it is identical to (38.1).

Proof of (35.2) $|B31_0| = 2$:

We prove (35.2) by case distinction using (36).

Case (36.1) $B31_0 \in \left\{B3 \underset{B3}{\mid} B3 \in \mathcal{P}[A_0] \wedge æ[A_0] \in B3 \wedge (|B3| = 2)\right\}$:

From what we already know follows:

From (36.1) we can infer

(39) $B31_0 \in \mathcal{P}[A_0] \wedge æ[A_0] \in B31_0 \wedge (|B31_0| = 2)$.

Formula (35.2) is true because it is identical to (39.3).

Case (36.2) $B31_0 \in \left\{B2 \underset{B2}{\mid} B2 \in \mathcal{P}[A_0] \wedge (|B2| = 2) \wedge æ[A_0] \notin B2\right\}$:

From what we already know follows:

From (36.2) we can infer

(40) $B31_0 \in \mathcal{P}[A_0] \wedge (|B31_0| = 2) \wedge æ[A_0] \notin B31_0$.

Formula (35.2) is true because it is identical to (40.2).

$\supseteq$: Now we assume

(35) $B31_0 \in \mathcal{P}[A_0] \wedge (|B31_0| = 2)$

and show:

(34)

$$B31_0 \in \left\{B3 \underset{B3}{\mid} B3 \in \mathcal{P}[A_0] \wedge æ[A_0] \in B3 \wedge (|B3| = 2)\right\} \cup \left\{B2 \underset{B2}{\mid} B2 \in \mathcal{P}[A_0] \wedge (|B2| = 2) \wedge æ[A_0] \notin B2\right\}.$$

In order to prove (34) we may assume

(45) $B31_0 \notin \left\{ B2 \underset{B2}{\mid} B2 \in \mathcal{P}[A_0] \bigwedge (|B2| = 2) \bigwedge æ[A_0] \notin B2 \right\}$.

and show:

(44) $B31_0 \in \left\{ B3 \underset{B3}{\mid} B3 \in \mathcal{P}[A_0] \bigwedge æ[A_0] \in B3 \bigwedge (|B3| = 2) \right\}$.

(Note, that in all other cases the formula (34) trivially holds!)

From what we already know follows:

From (45) we can infer

(46) $\neg\, (B31_0 \in \mathcal{P}[A_0] \bigwedge (|B31_0| = 2) \bigwedge æ[A_0] \notin B31_0)$.

Formula (46) is simplified to

(47) $B31_0 \notin \mathcal{P}[A_0] \bigvee |B31_0| \neq 2 \bigvee \neg\, (æ[A_0] \notin B31_0)$.

Using built–in simplification rules we can simplify the knowledge base:

Formula (47) simplifies to

(48) $B31_0 \notin \mathcal{P}[A_0] \bigvee |B31_0| \neq 2 \bigvee æ[A_0] \in B31_0$.

From (35.1) and (48) we obtain by resolution

(49) $|B31_0| \neq 2 \bigvee æ[A_0] \in B31_0$.

From (35.2) and (49) we obtain by resolution

(50) $æ[A_0] \in B31_0$.

In order to prove (44) we have to show:

(52) $B31_0 \in \mathcal{P}[A_0] \bigwedge æ[A_0] \in B31_0 \bigwedge (|B31_0| = 2)$.

Using built–in simplification rules and the additional assumption(s) (45), (35.2), (30), (24), (23), (22), (20), (4.1), (4.2), and (Definition (S2): base) we simplify (52) to

(53) $B31_0 \in \mathcal{P}[A_0] \bigwedge æ[A_0] \in B31_0$.

We prove the individual conjunctive parts of (53):

Proof of (53.1) $B31_0 \in \mathcal{P}[A_0]$:

Formula (53.1) is true because it is identical to (35.1).

Proof of (53.2) $æ[A_0] \in B31_0$:

Formula (53.2) is true because it is identical to (50).

□

3.4.3 Notes

Our induction base (1) is proven by contradiction. We first plug in our base definitions and end up with formula (7) and finally (8). The proof of (8) works with standard set theory proving.

In our induction step we prove (5). We again built up (20) with our inference rule *ArbitraryElementn1b.* We then instantiate the knowledge of our lemmata and definitions in (21) - (24) and (30).

We have to prove (31) and after simplification and together with the knowledge from above we get (33). Formula (33) is now proven by mutual inclusion with standard set theory proving.

3.5 An algorithm to compute the powerset

We develop an algorithm to compute the powerset of a given finite set and prove its correctness.

Given : A set A

With: is–finite[A]

Compute : P

Such that: $P = \mathcal{P}[A]$

We provide the following recursive algorithm:

1. The empty set has only itself as a subset
2. If $A \neq \{\}$, we take one arbitrary element a out of A and compute the union of:
   a.) Those subsets of A not containing a, namely the powerset of $A \setminus \{a\}$.
   b.) All the subsets of A which do contain a, namely $\left\{x \cup \{a\} \mid x \in Ps[A \setminus \{a\}]\right\}$.

```
Definition["PowersetAlg", any[A],
  Ps[{}] = {{}}                                                    "base"
  A ≠ {} ⇒ (Ps[A] = where[a = æ[A], Ps[A \ {a}] ∪ {x ∪ {a} | x∈Ps[A\{a}]}])   "rec" ]

Compute[Ps[{1, 2, 3, 4}], using → Definition["PowersetAlg"],
  built–in → ⟨Built–in["Sets"], Built–in["Numbers"],
      Built–in["Connectives"], Built–in["AllOperators"], Built–in["Quantifiers"]⟩]

{{}, {1}, {2}, {3}, {4}, {1, 2}, {1, 3}, {1, 4}, {2, 3},
  {2, 4}, {3, 4}, {1, 2, 3}, {1, 2, 4}, {1, 3, 4}, {2, 3, 4}, {1, 2, 3, 4}}
```

What we want to prove is the following:

```
Theorem["PowersetCorr", ∀ (Ps[A] = 𝒫[A])]
                        is–finite[A]
```

To structure the proof and make it work automatically, we introduce two lemmata. We also prove them with the help of the SetTheoryPCS prover in *Theorema*.

3.5.1 Lemmata for induction

The first lemma speaks about the equality of two sets namely $\mathcal{P}[A \setminus \{æ[A]\}]$ and $\{\{B \in \mathcal{P}[A] \mid æ[A] \notin B\}\}$.

```
Lemma["1a", any[A], with[A ≠ {}],
  𝒫[A \ {æ[A]}] = {B ∈ 𝒫[A] | æ[A] ∉ B}]

Prove[Lemma["1a"], by → SetTheoryPCSProver, transformBy → ProofSimplifier,
    TransformerOptions → {branches → Proved, steps → Useful},
    built–in → {Built–in["Numbers"]}, SearchDepth → 70,
    ProverOptions → {SimplifyFormula → True}]  // Timing
{8.016, - ProofObject -}
```

Prove:

(Lemma (1a)) $\forall_A \left(A \neq \{\} \Rightarrow \left(\mathcal{P}[A \setminus \{æ[A]\}] = \left\{ B \mid_B B \in \mathcal{P}[A] \wedge æ[A] \notin B \right\} \right) \right)$,

with no assumptions.

We assume

(1) $A_0 \neq \{\}$,

and show

(2) $\mathcal{P}[A_0 \setminus \{æ[A_0]\}] = \left\{ B \mid_B B \in \mathcal{P}[A_0] \wedge æ[A_0] \notin B \right\}$.

We show (2) by mutual inclusion:

The proof of this lemma is actually realtively easy and only uses basic set theory. We do not show here the full proof.

We skip the "⊆" part and do only show the "⊇" part of the mutual inclusion proof:

⊇: Now we assume

(5) $B1_0 \in \mathcal{P}[A_0] \wedge æ[A_0] \notin B1_0$

and show:

(4) $B1_0 \in \mathcal{P}[A_0 \setminus \{æ[A_0]\}]$.

For proving (4) we choose

(18) $B13_0 \in B1_0$,

and show:

(19) $B13_0 \in A_0 \setminus \{æ[A_0]\}$.

We have to prove (19), thus, we first show:

(20) $B13_0 \in A_0$:

From what we already know follows:

From (5.1) we can infer

(22) $B1_0 \subseteq A_0$.

From what we already know follows:

From (22) we can infer

(23) $\underset{B14}{\forall} (B14 \in B1_0 \Rightarrow B14 \in A_0)$.

Formula (18), by (23), implies:

(24) $B13_0 \in A_0$.

Formula (20) is true because it is identical to (24).

For proving (19) it still remains to show

(21) $B13_0 \notin \{æ[A_0]\}$:

In order to prove (21), it is sufficient to prove

(25) $B13_0 \neq æ[A_0]$

We prove (25) by contradiction.

We assume

(28) $B13_0 = æ[A_0]$,

and show a contradiction.

Formula (18), by (28), implies:

(31) $æ[A_0] \in B1_0$.

Formula (a contradiction) is proved because (31) and (5.2) are contradictory.

□

The second lemma again speaks about two sets being equal for all nonempty A, namely

{x ⋃ {æ[A]} | x ∈ {B ∈ 𝒫[A] | æ[A] ∉ B}} and {$B \in \mathcal{P}[A]$ | æ[A] ∈ B}

```
Lemma["2a", any[A, B], with[A ≠ {}],
  {x ⋃ {æ[A]} | x ∈ {B ∈ 𝒫[A] | æ[A] ∉ B}} = {B ∈ 𝒫[A] | æ[A] ∈ B}]

Prove[Lemma["2a"], by → SetTheoryPCSProver, transformBy → ProofSimplifier,
    TransformerOptions → {branches → Proved, steps → Useful},
    built–in → {Built–in["Numbers"]}, SearchDepth → 100,
    ProverOptions → {SimplifyFormula → True, SIPSetinv → True}]  // Timing
```

```
{44.313, - ProofObject -}
```

Prove:

(Lemma (2a))

$$\forall_A \left(A \neq \{\} \Rightarrow \left(\left\{ x \cup \{æ[A]\} \mid_x x \in \left\{ B \mid_B B \in \mathcal{P}[A] \wedge æ[A] \notin B \right\} \right\} = \left\{ B \mid_B B \in \mathcal{P}[A] \wedge æ[A] \in B \right\} \right) \right),$$

with no assumptions.

We assume

(1) $A_0 \neq \{\}$,

and show

(2) $\left\{ x \cup \{æ[A_0]\} \mid_x x \in \left\{ B \mid_B B \in \mathcal{P}[A_0] \wedge æ[A_0] \notin B \right\} \right\} = \left\{ B \mid_B B \in \mathcal{P}[A_0] \wedge æ[A_0] \in B \right\}$.

We show (2) by mutual inclusion:

Because the proof of this lemma is very similar to the one in the 2-element subset case, we do not show here the complete proof.

We again skip the "⊆" part and do only show the "⊇" part of the mutual inclusion proof:

⊇: Now we assume

(5) $x1_0 \in \mathcal{P}[A_0] \wedge æ[A_0] \in x1_0$

and show:

(4) $x1_0 \in \left\{x \bigcup \{æ[A_0]\} \mid_x x \in \left\{B \mid_B B \in \mathcal{P}[A_0] \bigwedge æ[A_0] \notin B\right\}\right\}$.

In order to prove (4) we have to show

(54) $\exists_x \left(x \in \left\{B \mid_B B \in \mathcal{P}[A_0] \bigwedge æ[A_0] \notin B\right\} \bigwedge (x1_0 = x \bigcup \{æ[A_0]\})\right)$.

In order to prove (54) we have to show:

(55) $\exists_x ((x \in \mathcal{P}[A_0] \bigwedge æ[A_0] \notin x) \bigwedge (x1_0 = x \bigcup \{æ[A_0]\}))$.

Using built–in simplification rules and the additional assumption(s) (1) we simplify (55) to

(56) $\exists_x (x \in \mathcal{P}[A_0] \bigwedge æ[A_0] \notin x \bigwedge (x1_0 = x \bigcup \{æ[A_0]\}))$.

From what we already know follows:

From (5.1) we can infer

(57) $x1_0 \subseteq A_0$.

From (57) and (5.2) we obtain the new formulae:

(58) $x1_0 = z2 \bigcup \{æ[A_0]\}$,

(59) $æ[A_0] \notin z2$,

(60) $z2 \subseteq A_0$.

Now, let $x := z2$. Thus, for proving (56) it is sufficient to prove:

(62) $z2 \in \mathcal{P}[A_0] \bigwedge æ[A_0] \notin z2 \bigwedge (x1_0 = z2 \bigcup \{æ[A_0]\})$.

Using built–in simplification rules and the additional assumption(s) (58), (59), and (1) we simplify (62) to

(64) $z2 \in \mathcal{P}[A_0]$.

For proving (64) we choose

(65) $z22_0 \in z2$,

and show:

(66) $z22_0 \in A_0$.

From (60) and (65) we obtain the new formulae:

(67) $z2 = z3 \bigcup \{z22_0\}$,

(68) $z22_0 \notin z3$,

(69) $z3 \subseteq A_0$.

Formula (58), by (67), implies:

(71) $x1_0 = (z3 \bigcup \{z22_0\}) \bigcup \{æ[A_0]\}$.

Using built–in simplification rules we can simplify the knowledge base:

Formula (71) simplifies to

(72) $x1_0 = z3 \bigcup \{z22_0\} \bigcup \{æ[A_0]\}$.

Formula (5.1), by (72), implies:

(77) $z3 \bigcup \{z22_0\} \bigcup \{æ[A_0]\} \in \mathcal{P}[A_0]$.

From what we already know follows:

From (77) we can infer

(78) $z3 \bigcup \{z22_0\} \bigcup \{æ[A_0]\} \subseteq A_0$.

From what we already know follows:

From (78) we can infer

(79) $\underset{z32}{\forall} (z32 \in z3 \bigcup \{z22_0\} \bigcup \{æ[A_0]\} \Rightarrow z32 \in A_0)$.

Formula (66), using (79), is implied by:

(80) $z22_0 \in z3 \bigcup \{z22_0\} \bigcup \{æ[A_0]\}$.

In order to prove (80) we may assume

(82) $z22_0 \notin \{z22_0\}$,

(83) $z22_0 \notin \{æ[A_0]\}$.

and show:

(81) $z22_0 \in z3$.

(Note, that in all other cases the formula (80) trivially holds!)

Using built–in simplification rules we can simplify the knowledge base:

Formula (82) simplifies to

(84) False.

From what we already know follows:

From (83) we can infer

(85) $z22_0 \neq æ[A_0]$.

Formula (81) is true because the assumption (84) is false.

□

We are now able to start our correctness proof.

3.5.2 Theorema proof

```
Prove[Theorem["PowersetCorr"], by → SetTheoryPCSProver,
    transformBy → ProofSimplifier, TransformerOptions → {branches → Proved, steps → Useful},
    using → {Definition["PowersetAlg"], Lemma["2a"], Lemma["1a"]}, SearchDepth → 120,
    ProverOptions → {EarlyRewriting → True, RWSetOperators → True, GRWTarget → {"kb", "goal"},
        UseCyclicRules → True, TransformRanges → False, SimplifyFormula → True}] // Timing
{109.204, - ProofObject -}
```

Prove:

(Proposition (PowersetCorr)) $\underset{\text{is–finite}[A]}{\forall} (\text{Ps}[A] = \mathcal{P}[A])$,

under the assumptions:

(Definition (PowersetAlg): base) $\text{Ps}[\{\}] = \{\{\}\}$,

(Definition (PowersetAlg): rec) $\underset{A}{\forall} \left(A \neq \{\} \Rightarrow \left(\text{Ps}[A] = \text{Ps}[A \setminus \{a\}] \cup \left\{ x \cup \{a\} \;\middle|\; x \in \text{Ps}[A \setminus \{a\}] \right\} \big|_{a \leftarrow \text{æ}[A]} \right) \right)$,

(Lemma (2a)) $\underset{\substack{A \\ A \neq \{\}}}{\forall} \left(\left\{ x \cup \{\text{æ}[A]\} \;\middle|\; x \in \left\{ B \;\middle|\; B \in \mathcal{P}[A], \text{æ}[A] \notin B \right\} \right\} = \left\{ B \;\middle|\; B \in \mathcal{P}[A], \text{æ}[A] \in B \right\} \right)$,

(Lemma (1a)) $\underset{\substack{A \\ A \neq \{\}}}{\forall} \left(\mathcal{P}[A \setminus \{\text{æ}[A]\}] = \left\{ B \;\middle|\; B \in \mathcal{P}[A], \text{æ}[A] \notin B \right\} \right)$.

We prove (Proposition (PowersetCorr)) by induction over the cardinality of the set A.

1. Induction base: We have to prove

(1) $\text{Ps}[\{\}] = \mathcal{P}[\{\}]$.

Using built–in simplification rules and the additional assumption(s) (Definition (PowersetAlg): base) we simplify (1) to

(6) $\text{Ps}[\{\}] = \{\{\}\}$.

Formula (6) is true because it is identical to (Definition (PowersetAlg): base).

2. Induction step: We assume

(2) $n_0 \in \mathbb{N}$,

(3) $\underset{A}{\forall}\ ((|A| = n_0) \Rightarrow (\mathrm{Ps}[A] = \mathcal{P}[A]))$,

and

(4) $A_0 \neq \{\} \bigwedge (|A_0| = 1 + n_0)$,

and prove

(5) $\mathrm{Ps}[A_0] = \mathcal{P}[A_0]$

From what we already know follows:

From (4.1) we know that we can choose an appropriate value such that

(7) $A1_0 \in A_0$.

From (4.2) we can infer

(8) $|A_0 \setminus \{æ[A_0]\}| = n_0$.

Formula (8), by (3), implies:

(9) $\mathrm{Ps}[A_0 \setminus \{æ[A_0]\}] = \mathcal{P}[A_0 \setminus \{æ[A_0]\}]$.

Formula (4.1), by (Definition (PowersetAlg): rec), implies:

(10) $\mathrm{Ps}[A_0] = \mathrm{Ps}[A_0 \setminus \{a\}] \bigcup \left\{ x \bigcup \{a\} \underset{x}{\mid} x \in \mathrm{Ps}[A_0 \setminus \{a\}] \right\} |_{a \leftarrow æ[A_0]}$.

Formula (4.1), by (Lemma (2a)), implies:

(11) $\left\{ x \bigcup \{æ[A_0]\} \underset{x}{\mid} x \in \left\{ B \underset{B}{\mid} B \in \mathcal{P}[A_0] \bigwedge æ[A_0] \notin B \right\} \right\} = \left\{ B \underset{B}{\mid} B \in \mathcal{P}[A_0] \bigwedge æ[A_0] \in B \right\}$.

Formula (4.1), by (Lemma (1a)), implies:

(12) $\mathcal{P}[A_0 \setminus \{æ[A_0]\}] = \left\{ B \underset{B}{\mid} B \in \mathcal{P}[A_0] \bigwedge æ[A_0] \notin B \right\}$.

Formula (9), by (12), implies:

(13) $\mathrm{Ps}[A_0 \setminus \{æ[A_0]\}] = \left\{ B \underset{B}{\mid} B \in \mathcal{P}[A_0] \bigwedge æ[A_0] \notin B \right\}$.

Formula (5), using (10), is implied by:

(14) $\mathrm{Ps}[A_0 \setminus \{a\}] \bigcup \left\{ x \bigcup \{a\} \underset{x}{\mid} x \in \mathrm{Ps}[A_0 \setminus \{a\}] \right\} |_{a \leftarrow æ[A_0]} = \mathcal{P}[A_0]$.

By simplification we obtain from (14) the new goal:

(15) $\mathrm{Ps}[A_0 \setminus \{æ[A_0]\}] \cup \left\{ x \cup \{æ[A_0]\} \mid_x x \in \mathrm{Ps}[A_0 \setminus \{æ[A_0]\}] \right\} = \mathcal{P}[A_0]$.

Formula (15), using (13), is implied by:

$$\left\{ B \mid_B B \in \mathcal{P}[A_0] \wedge æ[A_0] \notin B \right\} \cup \left\{ x \cup \{æ[A_0]\} \mid_x x \in \left\{ B \mid_B B \in \mathcal{P}[A_0] \wedge æ[A_0] \notin B \right\} \right\} = \mathcal{P}[A_0],$$

which, using (11), is implied by:

(16) $\left\{ B \mid_B B \in \mathcal{P}[A_0] \wedge æ[A_0] \notin B \right\} \cup \left\{ B \mid_B B \in \mathcal{P}[A_0] \wedge æ[A_0] \in B \right\} = \mathcal{P}[A_0]$.

We show (16) by mutual inclusion:

$\subseteq$: We assume

(17) $B1_0 \in \left\{ B \mid_B B \in \mathcal{P}[A_0] \wedge æ[A_0] \notin B \right\} \cup \left\{ B \mid_B B \in \mathcal{P}[A_0] \wedge æ[A_0] \in B \right\}$

and show:

(18) $B1_0 \in \mathcal{P}[A_0]$.

From what we already know follows:

From (17) we can infer

(19) $B1_0 \in \left\{ B \mid_B B \in \mathcal{P}[A_0] \wedge æ[A_0] \notin B \right\} \vee B1_0 \in \left\{ B \mid_B B \in \mathcal{P}[A_0] \wedge æ[A_0] \in B \right\}$.

We prove (18) by case distinction using (19).

Case (19.1) $B1_0 \in \left\{ B \mid_B B \in \mathcal{P}[A_0] \wedge æ[A_0] \notin B \right\}$:

From what we already know follows:

From (19.1) we can infer

(20) $B1_0 \in \mathcal{P}[A_0] \wedge æ[A_0] \notin B1_0$.

Formula (18) is true because it is identical to (20.1).

Case (19.2) $B1_0 \in \left\{ B \mid_B B \in \mathcal{P}[A_0] \wedge æ[A_0] \in B \right\}$:

From what we already know follows:

From (19.2) we can infer

(21) $B1_0 \in \mathcal{P}[A_0] \wedge æ[A_0] \in B1_0$.

Formula (18) is true because it is identical to (21.1).

⊇: Now we assume

(18) $B1_0 \in \mathcal{P}[A_0]$

and show:

(17) $B1_0 \in \left\{B \underset{B}{\mid} B \in \mathcal{P}[A_0] \bigwedge æ[A_0] \notin B\right\} \bigcup \left\{B \underset{B}{\mid} B \in \mathcal{P}[A_0] \bigwedge æ[A_0] \in B\right\}$.

In order to prove (17) we may assume

(25) $B1_0 \notin \left\{B \underset{B}{\mid} B \in \mathcal{P}[A_0] \bigwedge æ[A_0] \in B\right\}$.

and show:

(24) $B1_0 \in \left\{B \underset{B}{\mid} B \in \mathcal{P}[A_0] \bigwedge æ[A_0] \notin B\right\}$.

(Note, that in all other cases the formula (17) trivially holds!)

From what we already know follows:

From (25) we can infer

(26) $\neg\, (B1_0 \in \mathcal{P}[A_0] \bigwedge æ[A_0] \in B1_0)$.

Using built–in simplification rules we can simplify the knowledge base:

Formula (26) simplifies to

(27) $æ[A_0] \notin B1_0 \bigvee B1_0 \notin \mathcal{P}[A_0]$.

From (18) and (27) we obtain by resolution

(28) $æ[A_0] \notin B1_0$.

In order to prove (24) we have to show:

(29) $B1_0 \in \mathcal{P}[A_0] \bigwedge æ[A_0] \notin B1_0$.

Using built–in simplification rules and the additional assumption(s) (28), (25), (13), (12), (11), (10), (8), (4.1), (4.2), and (Definition (PowersetAlg): base) we simplify (29) to

(30) $B1_0 \in \mathcal{P}[A_0]$.

Formula (30) is true because it is identical to (18).

□

3.5.3 Notes

Our induction base (1) is proven by plugging in the base definition of our algorithm and one simplification step.

In our induction step we prove (5). We again built up (8) with our inference rule *ArbitraryElementn1b*. In (9) we use our induction hypothesis (3). We then built up the formulae (10) - (13), the knowledge of our lemmata and our algorithm definition.

We end up with the proof goal (14) and get after simplification and plugging in our knowledge formula (16). Formula (16) is again proved by mutual inclusion and standard set theory proving.

Chapter 4 : Conclusion

In this thesis we discussed a new special prover, namely the set induction prover SIP. It is an extension of the SetTheoryPCSProver in *Theorema*. In future versions of the *Theorema* system the set induction prover will be fully integrated into the system, namely into the SetTheoryPCSProver.

Our focus was on automatically proving correctness and cardinality properties about given recursive algorithms in *Theorema*. More precisely we talked about algorithms which compute with finite sets.

We discussed the implementation of our SIP prover and five different case studies. We provide a timing table:

Example	**Timing (in seconds)**
Minimal element of *a* given finite set	6.31
The cardinality of all 2 –element subsets of *a* given finite set	9.92
The cardinality of all k–element subsets of *a* given finite set	303.75
The correctness of algorithm S2 (Lemma 1)	25.56
The correctness of algorithm S2 (Lemma 2)	7.98
The correctness of algorithm S2	145.55
An algorithm to compute the powerset (Lemma 1 *a*)	8.02
An algorithm to compute the powerset (Lemma 2 *a*)	44.31
An algorithm to compute the powerset	109.20

Our prover is powerful enough to succeed the proofs in all our examples within the outermost left branch of the proof tree. Anyway, the depths in those branches of the proof trees are bigger in our last three examples.

Future work

- Full integration of the set induction prover SIP into the existing SetTheoryPCSProver in future versions of the *Theorema* system .
- Refine the strategy of those inference rules that expand our knowledge base, i.e. follow the same strategy as in our rule *ArbitraryElementn1b* with the head *NewKnowledgeFromOne*.

References

[Buch03] B. Buchberger.
Algorithm Invention and Verification by Lazy Thinking
Johannes Kepler University Linz. Spezialforschungsbereich F013. Technical report no. 2003-29

[Buch04] B. Buchberger.
Towards the Automated Synthesis of a Groebner Bases Algorithm.
RACSAM - Revista de la Real Academia de Ciencias (Review of the Spanish Royal Academy of Science), Serie A:Mathematicas 98(1), pp. 65-75. 2004.
ISSN 1578 7303.

[Mayrh09] G. Mayrhofer.
Symbolic Computation Prover with Induction.
Johannes Kepler University. Diploma Thesis. September 2009.

[Tma97] B.Buchberger, T.Jebelean, F.Kriftner, M.Marin, E.Tomuta, and D.Vasaru.
A Survey of the Theorema Project.
Technical report no.97-15 in RISC Report Series, University of Linz, Austria, March 1997.

[Tma00a] B.Buchberger, D.Vasaru, and T.Jebelean.
The Theorema System: Current Status and the Proving-Solving-Computing Cycle.
Technical Report 00-37, RISC Report Series, University of Linz, Austria, May 2000.

[Tma00b] B.Buchberger, C.Dupre, T.Jebelean, B.Konev, F.Kriftner, T.Kutsia, K.Nakagawa, F.Piroi, D.Vasaru and W.Windsteiger.

The Theorema System: Proving, Solving, and Computing for the Working Mathematician.
Technical Report 00-38, RISC Report Series,University of Linz, Austria, August 2000.

[Tma00c] B.Buchberger, C.Dupre, T.Jebelean, B.Konev, F.Kriftner, T.Kutsia, K.Nakagawa, F.Piroi, D.Vasaru and W.Windsteiger.
The Natural Style Provers of Theorema: A Survey of Strategies for Different Mathematical Domains.
Technical Report 00-39, RISC Report Series, University of Linz, Austria, June 2000.

[Tma06] B. Buchberger, A. Craciun, T. Jebelean, L. Kovacs, T. Kutsia, K. Nakagawa, F. Piroi, N. Popov, J. Robu, M. Rosenkranz, W. Windsteiger.
Theorema: Towards Computer-Aided Mathematical Theory Exploration.
Journal of Applied Logic 4(4), pp. 470-504. 2006. ISSN 1570-8683.

[Wind03] W.Windsteiger.
A Set Theory Prover in Theorema: Implementation and Practical Applications.
RISC Institute. PhD Thesis.

[Wolf03] S.Wolfram.
The Mathematica Book
Wolfram Media Inc., Champaign, Illinois (USA), 2003

[Wolf10] Wolfram Mathemtica Documentation center
http://reference.wolfram.com/mathematica/guide/Mathematica.html, June 2010