The PCS Prover

in THEOREM

Bruno Buchberger
Research Institute for Symbolic Computation
Johannes Kepler University, Linz, Austria
Buchberger@RISC.Uni-Linz.ac.at

Copyright: Springer, Heidelberg.

Bibliographic Note: To appear in the Proc. of European Conference on Computer-Aided Systems Theory 2001 (EUROCAST 2001), Las Palmas, Gran Canarias, February 19-23, 2001, Lecture Notes in Computer Science, 2001. (Copyright: Springer, Heidelberg.)

Acknowledgement: This paper was written in the frame of the Project "Scientific Computing" sponsored by the Austrian National Science Foundation (FWF), project number SFB 1302. I would also like to thank M. Rosenkranz for helping me in the preparation of this manuscript.

1 Introduction

In this paper, we present the main ideas of a new, heuristic, proving method for predicate logic called the PCS method (Proving-Computing-Solving method). The method is particularly suited for proving theorems in theories whose main notions are defined by formulae with alternating quantifiers, i.e. formulae of the form ∀∃∀ ... A typical example of such a notion is the notion of limit:

$$\text{limit}[f, a] \Longleftrightarrow \underset{\epsilon>0}{\forall} \underset{N}{\exists} \underset{n\geq N}{\forall} |f[n] - a| < \epsilon$$

The main emphasis of the PCS method is naturalness, i.e. the method imitates human proof style and generates proofs that are easy to understand. Also, in the cases the method works, it normally finds the proof with very little search.

In contrast to the resolution method and other well-known methods for automated theorem proving in predicate logic, the PCS is not complete and will fail in many cases. However, we believe that, for the acceptance of theorem proving as a tool for practical theorem proving, it is important to come up with special proof methods that deliver natural proofs in short time for the many nontrivial but not too difficult theorems that occur in the usual exploration of mathematical theories. For this objective it seems that the PCS can make a useful contribution. In fact, proving the propositions about the elementary theory of analysis, e.g. propositions about the notion of limit, for general predicate logic theorem provers still is a hard problem and, thus, we believe that the PCS method is a decisive step forward.

Essentially, the PCS method, in a natural way, reduces proving to solving. In the case of analysis, proofs are reduced to solving constraints over the real numbers. Fortunately, by the work of Collins and others, see [Collins 1975] and [Caviness, Johnson 1998], there exist complete algorithms for solving the most general class of constraints over real numbers. Thus, in the cases we manage to reduce the proof of a theorem to constraint solving over the reals, the proof can be established. In fact, we will see that this reduction by the PCS method is "natural" and that the solution of the constraints is an "uninteresting" step whose details people do not want to see when they are exploring analysis because, at that stage, they already master the theory of real numbers and would like to concentrate on the exploration of the notions of analysis like limit, derivative, etc. Thus, it is methodologically appropriate to call constraint solvers as "black boxes" at this stage.

2 TH∃OREM∀

The *Theorema* system is a software system that aims at automating proving in a uniform logic and software frame for formal mathematics. It is programmed in Mathematica and, hence, is available on all platforms on which Mathematica is available.

However, this does not entail that, when doing proofs in *Theorema*, any of the implicit knowledge of Mathematica is used. All knowledge that is used in *Theorema* proofs can be stated explicity. However, we also have means to state explicitly that well-defined sections of Mathematica knowledge, i.e. algorithms for many mathematical functions, can be used in proofs. This gives maximum freedom for the user to "believe" in the correctness of Mathematica knowledge or not.

Theorema is a multi-method system, i.e. we do not attempt to generate proofs in all

areas of mathematics with just one general predicate logic proving method. In fact, we believe that having only one proof method for all of mathematics, although theoretically possible, is not practical. Thus, in *Theorema*, we provide a library of general and special provers together with general and special solvers and general and special simplifiers.

In *Theorema*, we emphasize the importance of readable proofs and nice output. Thus, we do not only generate abstract proof objects but we also provide post-processors that transform the abstract proof objects into proof text that can easily be read by humans.

The *Theorema* system is based on research of the author in the area of computer algebra, formal mathematics, and didactics since 1975 and is now a joint effort of the *Theorema* Working Group directed by the author since 1996, see *www.theorema.org*. More details about *Theorema* can be found in [Buchberger et al. 1997, 2000].

3 Mathematical Texts in TH∃OREM∀

Before we go into the details of the PCS method, we present an example formal text in *Theorema* which will later be used for demonstrating the method. We start with the definition of the notion of limit:

Definition["limit:", any[f, a],

$$\text{limit}[f, a] \Longleftrightarrow \underset{\epsilon>0}{\forall_{\epsilon}} \underset{N}{\exists} \underset{n\geq N}{\forall_{n}} |f[n] - a| < \epsilon\Big]$$

The actual definition is the predicate logic formula

$$\text{limit}[f, a] \Longleftrightarrow \underset{\epsilon>0}{\forall_{\epsilon}} \underset{N}{\exists} \underset{n\geq N}{\forall_{n}} |f[n] - a| < \epsilon$$

that should be self-explanatory. The *Theorema* notation for formulae is close to the usual notation in mathematical textbooks. The only exception is that we use brackets instead of parentheses for function application, i.e. 'f[n]' is the term with function symbol 'f' and argument term 'n'. The use of brackets instead of parentheses is taken over from Mathematica because, in fact, parentheses are ambiguous: For example, 'f(n+m)' could be understood as both 'f[m+n]' and 'f.(m+n)'. The 'any[f,a]' declares 'f' and 'a' as free variables. All identifiers (and function and predicate symbols) that are neither declared as free variables nor bound by a quantifier are considered to be constants. Note that, in the above example formula, 'f' is a higher-order variable: It occurs at the position of a function symbol in the term 'f[n]'.

The keyword 'Definition' and the label "limit" have no logical meaning. The are only used for easy reference: As soon as the above definition is entered into an input cell of Mathematica (after having loaded the *Theorema* system on top of Mathematica) one can refer to the entire definition by just 'Definition["limit"]', for example when building up theories (see below) or when referring to the use of definitions in proofs.

Now let us formulate an easy proposition on the notion of limit in the notation of *Theorema*:

Proposition["limit of sum", any[f, a, g, b],
  (limit[f, a] ∧ limit[g, b]) ⇒ limit[f + g, a + b]]

We will show later how a proof of this proposition can be generated automatically by the PCS prover of *Theorema*. Before we attempt to do this we must, of course, provide some knowledge on the notions +, -, <, etc. occurring in the definition of the notion of limit. First, we need the definition of + on sequences:

Definition["+:", any[f, g, x],
  (f + g)[x] = f[x] + g[x]]

Also, we need a version of the "triangle inequality":

Lemma["|+|", any[x, y, a, b, δ, ϵ],
  (|(x + y) − (a + b)| < (δ + ϵ)) ⟸ (|x − a| < δ ∧ |y − b| < ϵ)]

Finally, we will need some knowledge on the maximum function:

Lemma["max", any[m, M1, M2],
  m ≥ max[M1, M2] ⇒ (m ≥ M1 ∧ m ≥ M2)]

In this paper, we do not discuss the interesting question of how one knows which knowledge is appropriate for proving a given theorem. In fact, playing with a system like *Theorema* gives a lot of insight into the mechanism of how to "explore theories" instead of just "proving isolated theorems", see some ideas on this question in [Buchberger 2000].

Now we can combine the individual formulae above in one knowledge base by the *Theorema* construct 'Theory'.

```
Theory["limit",
        Definition["limit:"]
        Definition["+:"]
        Lemma["|+|"]
        Lemma["max"]
]
```

In fact, the 'Theory' construct can be applied recursively, i.e. one can build up hierarchically structured theories in *Theorema* and refer to them by a single label.

4 The PCS Proving Method

4.1 An Overview on the PCS Method

The PCS proof method was established by the author in 2000 and aims at generating "natural" proofs. In fact, the PCS method basically is a formalization of a heuristic method the author has been teaching for many years in his "Thinking, Speaking, Writing" course as a practical proof technique for humans.

Roughly, the PCS method proceeds by iteratively going through the following three phases:

- the *P-phase* ("Proving" phase)
- the *C-phase* ("Computing" phase)
- the *S-phase* ("Solving" phase)

In the P-phase, a couple of predicate logic rules are applied in the "natural deduction" style in order to decompose the proof problem into a couple of more elementary proofs. In the C-phase, definitions (and other equalities and equivalences) and implications are used in a "rewrite" (symbolic computation) style in order to reduce proof goals and to expand knowledge bases. By the P-phase and the C-phase, one arrives at proof situations in which the goals have the form of existentially quantified formulae, i.e. one has to "find" terms that satisfy the conditions specified in the goals. In this moment, the proof can often be completed by calling algorithmic solvers for certain special theories, for example, the theory of real numbers. Hence, the PCS method brings together theorem proving with algebraic algorithms.

A first implementation of the PCS method, within *Theorema*, was sketched and tested by the author in 1999 and was then worked out in detail in the PhD thesis [Vasaru 2000]. An implementation of the PCS method for the special case of set theory will be presented in the PhD thesis [Windsteiger 2001].

4.2 A Proof Generated by the PCS Prover

A proof of the above proposition can be found completely automatically, by entering the following *Theorema* call

```
Prove[Proposition["limit of sum"], using → Theory["limit"], by → PCS]
```

Below, we show the proof exactly as generated by the system. One can explain the essential ingredients of the method most easily by going through the individual steps of this example proof.

Prove:

(Proposition (limit of sum)) $\forall_{f,a,g,b} (\text{limit}[f, a] \wedge \text{limit}[g, b] \Rightarrow \text{limit}[f + g, a + b])$,

under the assumptions:

(Definition (limit:)) $\forall_{f,a} \left(\text{limit}[f, a] \Leftrightarrow \forall_{\epsilon > 0} \exists_{N} \forall_{n \geq N} (|f[n] - a| < \epsilon) \right)$,

(Definition (+:)) $\forall_{f,g,x} ((f + g)[x] = f[x] + g[x])$,

(Lemma (|+|)) $\forall_{x,y,a,b,\delta,\epsilon} (|(x + y) - (a + b)| < \delta + \epsilon \Leftarrow (|x - a| < \delta \wedge |y - b| < \epsilon))$,

(Lemma (max)) $\forall_{m,M1,M2} (m \geq \max[M1, M2] \Rightarrow m \geq M1 \wedge m \geq M2)$.

We assume

(1) $\text{limit}[f_0, a_0] \wedge \text{limit}[g_0, b_0]$,

and show

(2) $\text{limit}[f_0 + g_0, a_0 + b_0]$.

Formula (1.1), by (Definition (limit:)), implies:

(3) $\forall_{\epsilon > 0} \exists_{N} \forall_{n \geq N} (|f_0[n] - a_0| < \epsilon)$.

By (3), we can take an appropriate Skolem function such that

(4) $\forall_{\epsilon > 0} \forall_{n \geq N_0[\epsilon]} (|f_0[n] - a_0| < \epsilon)$.

Formula (1.2), by (Definition (limit:)), implies:

(5) $\underset{\epsilon>0}{\forall}\ \underset{N}{\exists}\ \underset{n\geq N}{\forall}\ (|g_0[n]-b_0|<\epsilon).$

By (5), we can take an appropriate Skolem function such that

(6) $\underset{\epsilon>0}{\forall}\ \underset{n\geq N_1[\epsilon]}{\forall}\ (|g_0[n]-b_0|<\epsilon).$

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

(7) $\underset{\epsilon>0}{\forall}\ \underset{N}{\exists}\ \underset{n\geq N}{\forall}\ (|(f_0+g_0)[n]-(a_0+b_0)|<\epsilon).$

We assume

(8) $\epsilon_0>0,$

and show

(9) $\underset{N}{\exists}\ \underset{n\geq N}{\forall}\ (|(f_0+g_0)[n]-(a_0+b_0)|<\epsilon_0).$

We have to find N_2^* such that

(10) $\underset{n}{\forall}\ (n\geq \mathrm{N}_2^* \Rightarrow |(f_0+g_0)[n]-(a_0+b_0)|<\epsilon_0).$

Formula (10), using (Definition (+:)), is implied by:

(11) $\underset{n}{\forall}\ (n\geq \mathrm{N}_2^* \Rightarrow |(f_0[n]+g_0[n])-(a_0+b_0)|<\epsilon_0).$

Formula (11), using (Lemma (|+|)), is implied by:

(12) $\underset{\delta+\epsilon=\epsilon_0}{\underset{\delta,\epsilon}{\exists}}\ \underset{n}{\forall}\ (n\geq \mathrm{N}_2^* \Rightarrow |f_0[n]-a_0|<\delta \wedge |g_0[n]-b_0|<\epsilon).$

We have to find δ_0^*, ϵ_1^* and N_2^* such that

(13) $(\delta_0^*+\epsilon_1^*=\epsilon_0)\bigwedge \underset{n}{\forall}\ (n\geq \mathrm{N}_2^* \Rightarrow |f_0[n]-a_0|<\delta_0^* \wedge |g_0[n]-b_0|<\epsilon_1^*).$

Formula (13), using (6), is implied by:

$(\delta_0^*+\epsilon_1^*=\epsilon_0)\bigwedge \underset{n}{\forall}\ (n\geq \mathrm{N}_2^* \Rightarrow \epsilon_1^*>0 \wedge n\geq N_1[\epsilon_1^*] \wedge |f_0[n]-a_0|<\delta_0^*),$

which, using (4), is implied by:

$(\delta_0^*+\epsilon_1^*=\epsilon_0)\bigwedge \underset{n}{\forall}\ (n\geq \mathrm{N}_2^* \Rightarrow \delta_0^*>0 \wedge \epsilon_1^*>0 \wedge n\geq N_0[\delta_0^*] \wedge n\geq N_1[\epsilon_1^*]),$

which, using (Lemma (max)), is implied by:

(14) $(\delta_0^*+\epsilon_1^*=\epsilon_0)\bigwedge \underset{n}{\forall}\ (n\geq \mathrm{N}_2^* \Rightarrow \delta_0^*>0 \wedge \epsilon_1^*>0 \wedge n\geq \max[N_0[\delta_0^*],\ N_1[\epsilon_1^*]]).$

Formula (14) is implied by

(15) $(\delta_0^*+\epsilon_1^*=\epsilon_0)\bigwedge \delta_0^*>0\bigwedge \epsilon_1^*>0\bigwedge \underset{n}{\forall}\ (n\geq \mathrm{N}_2^* \Rightarrow n\geq \max[N_0[\delta_0^*],\ N_1[\epsilon_1^*]]).$

Partially solving it, formula (15) is implied by

(16) $(\delta_0^*+\epsilon_1^*=\epsilon_0)\wedge \delta_0^*>0\wedge \epsilon_1^*>0\wedge (\mathrm{N}_2^*=\max[N_0[\delta_0^*],\ N_1[\epsilon_1^*]]).$

Now,

$(\delta_0^* + \epsilon_1^* = \epsilon_0) \wedge \delta_0^* > 0 \wedge \epsilon_1^* > 0$

can be solved for δ_0^* and ϵ_1^* by a call to Collins cad-method yielding the solution

$0 < \delta_0^* < \epsilon_0$,

$\epsilon_1^* \leftarrow \epsilon_0 + -1 * \delta_0^*$.

Let us take

$N_2^* \leftarrow \max[N_0[\delta_0^*], N_1[\epsilon_0 + -1 * \delta_0^*]]$.

Formula (16) is solved. Hence, we are done.

4.3 The Essential Ideas of the PCS Method

Taking the above PCS-generated proof as an example, we now describe the essential steps of the PCS method in more detail:

The proof starts by echoing the proposition to be proved. In our example, this is the proposition with label (Proposition (limit of sum)). Then we echo the formulae in the initial knowledge base. In our example, these are the formulae with labels (Definition (limit:)), (Definition (+:)), (Lemma (|+|)), and (Lemma (max)).

P-phase: Now we start with a phase in which the "natural deduction" rules of predicate logic, except the ones for equalities, equivalences, and implications, are applied to the proof goal and the knowledge. By doing this, the given proof situation is reduced to one or more other, simpler, proof situations. In the above example, the P-phase produces formulae (1) and (2) by applying the "arbitrary but fixed" rule and the deduction rule of predicate logic.

C-phase: Now we try to use "rewrite knowledge" (equivalences, equalities, and implications in the knowledge base) in the "rewrite" style, i.e. we replace, in the goal formula and in formulae of the knowledge base, appropriate instances of the left-hand sides of the rewrite knowledge by the corresponding instances of the right-hand sides. Note that, by doing so, goals are reduced to other goals that *imply* the given goals whereas formulae in the knowledge base are expanded to other formuale in the knowledge base that are *implied* by the given knowledge. In our example, formualae (3) and (5) are generated from (1) by C-phase steps using (Definition (limit:)) as rewrite knowledge.

P-phase with Skolemization: Now we may be back in a P-phase, i.e. a phase in which natural deduction steps can be applied. In this phase, we apply, in addition to the usual natural deduction rules of predicate logic, Skolemization, i.e. for formulae of the form $\forall_x \exists_y F[x, y]$ in the knowledge base we introduce new function constants ("Skolem" function constants) and assert $\forall_x F[x, f[x]]$. This step is crucial for having the possibility in the later S-phase to construct solving terms for existentially quantified formulae in an explicit way. In our example, formulae (4) and (6) are derived from formulae (3) and (5), respectively, by Skolemization.

C-phase: Now we may again be in a C-phase in which rewrite knowledge is applicable in rewrite style. In our example, (7) is now obtained from (2) by using again (Definition (limit:)) as a rewrite rule.

P-phase: Now again a P-phase brings us to the additional assumption (8) and the new goal (9).

S-phase: Now the goal is an existentially quantified formula and we must start "solving", i.e. finding an appropriate term that satisfies the condition stated in the goal formula. We start solving by, first, introducing a "find constant", i.e. a new constant whose value will be determined later as the proof proceeds. We use constants with asterisks for this purpose. Introducing these constants is important in order to be able to decompose goals further, i.e. to work inside the existentially quantified formula in a couple of alternating P- and C-phases. In our example, we introduce now N_2^* and obtain the new goal (10).

P- and C-phases: In our example, by using (Definition (+:)) as a rewrite rule, goal (10) can be reduced to goal (11).

C-phase with existential rewriting: Now we are at an important proof situation. Namely, the conclusion of goal (11)

$$\forall_n (n \geq \mathrm{N}_2^* \Rightarrow |(f_0[n] + g_0[n]) - (a_0 + b_0)| < \epsilon_0).$$

is very close to being an instance of the conclusion in (Lemma (|+|))

$$\forall_{x,y,a,b,\delta,\epsilon} (|(x + y) - (a + b)| < \delta + \epsilon \Leftarrow (|x - a| < \delta \wedge |y - b| < \epsilon))$$

so that a reduction of the goal by rewriting would be possible. However, 'ϵ_0' is a constant and, thus, we cannot find a substitution for 'δ' and 'ϵ' such that, by this substitution, '$\delta+\epsilon$' would be transformed into 'ϵ_0'. For handling this situation, we propose "existential rewriting": We reduce goal (11), by using (Lemma (|+|)), to goal (12):

$$\exists_{\substack{\delta,\epsilon \\ \delta+\epsilon=\epsilon_0}} \forall_n (n \geq \mathrm{N}_2^* \Rightarrow |f_0[n] - a_0| < \delta \wedge |g_0[n] - b_0| < \epsilon).$$

It is easy to prove that this generalized form of rewriting is correct. By existential rewriting, we are able to handle the above proof situation in a natural way on the expense of introducing existential quantifiers in the goal.

S-phase: Now we are again in an S-phase, which we handle by introducing extra find constants. In our case 'δ_0^*' and 'ϵ_1^*' are introduced as new find constants yielding the new goal (13).

C-phase: Goal (13) can now be reduced by a couple of rewrite steps, using the Skolemized formulae (6) and (4) and also (Lemma (max)), to formula (14).

P-phase: Now P-steps are possible that bring the formulae '$\delta_0^* > 0$' and '$\epsilon_1^* > 0$', which do not contain variable 'n', outside the scope of the $\forall_n$ quantifier.

S-phase: The resulting formula (15) has now the property that it is the conjunction of two independent solve problems, the first one asking to find 'δ_0^*' and 'ϵ_1^*' in dependence on 'ϵ_0' and the second one asking to find N_2^* in dependence on 'δ_0^*' and 'ϵ_1^*'. The second problem can be solved by simple predicate logic rules and yields

$$\mathrm{N}_2^* = \max[N_0[\delta_0^*], N_1[\epsilon_1^*]]$$

as a possible solution. The first problem is a problem that is a constraint solving problem over the real numbers and, hence, can be solved by a call to any complete real number constraint solver. We use Collins' algorithm for this purpose, which is available in the extended Mathematica library. We obtain a general answer back, namely

$$0 < \delta_0^* < \epsilon_0,$$
$$\epsilon_1^* \leftarrow \epsilon_0 - \delta_0^*.$$

This means that any δ_0^* satisfying $0 < \delta_0^* < \epsilon_0$ is a possible solution and that then ϵ_1^* must be chosen as $\epsilon_0 - \delta_0^*$. This concludes the proof.

Note that the proof generated by the PCS prover, in addition to showing that the proposition is a consequence of the formulae in the knowledge base, yields interesting information on the convergence of the sum sequence *f*+*g*: The solving terms for δ_0^*, ϵ_1^*, and N_2^* that are constructed during the proof of the proposition tell us that, given $\epsilon_0 > 0$, one can find an index N_2^* such that, from N_2^* on, the elements of the sequence f+g stay closer than ϵ_0 to a+b by the following procedure:

Choose an arbitrary δ_0^* such that $0 < \delta_0^* < \epsilon_0$.
Then compute $\epsilon_1^* := \epsilon_0 - \delta_0^*$.
Finally compute $N_2^* := \max[N_0[\delta_0^*], N_1[\epsilon_1^*]]$.

Here N_0 is a procedure by which, given an arbitrary $\epsilon > 0$, one can find an index from which on f stays closer to a than ϵ, and, similarly, N_1 gives an index bound for g. Thus the solving terms constructed in the proof can be viewed as a procedure for the index bound of $f + g$ with index bounds for f and g as "black-box" subprocedures. In other words, the PCS prover is not only a prover but also a procedure synthesizer. In case one has algorithmic procedures N_0 and N_1 for finding the index bounds for f and g, the procedure synthesizer synthesizes an algorithm for computing the index bound for $f + g$. Thus, the PCS prover does not only generate proofs but also provides interesting constructive information on the notions involved in the proposition.

5 Conclusion

The PCS prover combines, in a natural way, proving by a restricted set of inference rules, simplifying, and solving. In fact, also other general and special automated provers combine restricted proving, simplifying and solving. For example, proving geometrical theorems by the Gröbner bases method, essentially is also a reduction, by certain proving and simplifying steps, of deciding the truth of formulae to deciding the solvability of certain related sets of algebraic equations. Also, the famous resolution method for general predicate logic proving, is essentially a reduction of proving, by simplifiying, to solving certain sets of standard predicate logic formulae, namely clauses.

In a future version of *Theorema*, the flexible interplay between proving, solving, and simplifying will be our main design feature so that *Theorema* will appear as a library of built-in provers, solvers, and simplifyers from which the user can build provers, solvers, and simplifiers for the particular given application in an easy, flexible and general way.

References

[Buchberger 2000] B. Buchberger. Theory Exploration with *Theorema.* In: *Proc. of the 2nd International Workshop on Symbolic and Numeric Algorithms for Scientific Computing* (SYNASC 2000), Dept. of Computer Science, Univ. of West Timisoara, Romania, Oct. 4-6, 2000, pp. 1-16.

[Buchberger et al. 1997] B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, D. Vasaru. An Overview on the Theorema project. In: *Proc. of the International Symposium on Symbolic and Algebraic Computation* (ISSAC 97), Maui, Hawaii, July 21-23, 1997, W.Kuechlin (ed.), ACM Press 1997, pp. 384-391.

[Buchberger et al. 2000] B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, W. Windsteiger. The Theorema Project: A Progress Report. In: *Proc. of the 8th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning* (Calculemus 2000), St. Andrews, Scotland, August 6-7, M. Kerber and M. Kohlhase (eds.), available from: Fachbereich Informatik, Universität des Saarlandes, pp. 100-115.

[Collins 1975] G. E. Collins. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In: *Second GI Conference on Automata Theory and Formal Languages*, LNCS 33, pages 134-183, Springer Verlag, Berlin, 1975. Reprinted in [Caviness, Johnson 1998], pp. 85-121.

[Caviness, Johnson 1998] B. F. Caviness, J. R. Johnson (eds.). *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Texts and Monographs in Symbolic Computation of the Research Institute for Symbolic Computation (B. Buchberger, G.E. Collins, eds.), Springer, Wien-New York, 431 pages.

[Vasaru 2000] D. Vasaru-Dupre. *Automated Theorem Proving by Integrating Proving, Solving and Computing*, PhD Thesis, April 2000, Research Institute for Symbolic Compuationat (RISC), Johannes Kepler University Linz, A-4232 Castle of Hagenberg, Austria.

[Windsteiger 2001] W. Windsteiger. *A Set Theory Prover in Theorema: Implementation and Practical Applications*, PhD Thesis, May 2001, Research Institute for Symbolic Compuationat (RISC), Johannes Kepler University Linz, A-4232 Castle of Hagenberg, Austria.