Employing External Reasoners in Proof Planning

Erica Melis and Volker Sorge

Universität des Saarlandes, Fachbereich Informatik,
D-66041 Saarbrücken, Germany
{melis|sorge}@ags.uni-sb.de

Abstract

This paper describes the integration of computer algebra systems and constraint solvers into proof planners. It shows how efficient external reasoners can be employed in proof planning and how the shortcuts of the external reasoners can be expanded to verifiable natural deduction proofs in the proof planning framework. In particular, these shortcuts *simplify* and *guide* the formal proof. The paper illustrates the integration and cooperation of the external reasoners with an example from proof planning limit theorems.

1 Introduction

In many mathematical proofs, logical steps are combined with specialized reasoning such as computing integrals, solving polynomial equations, and solving inequalities. Usually, a purely logical proof of this specialized reasoning is possible in principle but far too long and inefficient and therefore practically not feasible. Such proofs may generate a search spaces that could easily prevent a system from finding a proof.

Hence, the integration of specialized external reasoners, for instance decision procedures and Computer Algebra Systems (CAS), into automated theorem provers has been a hot topic. Similarly, the integration of Constraint Solvers (CS) is desirable in order to restrict variable instantiations with the help of domain specific knowledge. Usually, these external systems have their own efficient data structures and algorithms. However, their integration into automated theorem proving systems can be very difficult and time consuming as, e.g., documented in [3].

This paper shows how these efficient external reasoners can be generally employed in proof planning and how the proof shortcuts produced by the

Preprint submitted to Elsevier Preprint

external reasoners *simplify* or *guide* the formal proof. In particular, the shortcut introduced by a CAS can be expanded to checkable a natural deduction (ND)-proof using the computations of the small self-tailored CAS, $\mu\mathcal{CAS}$, and for witnesses synthesized by external reasoners only the verification needs to be formally proved. These verification proofs are usually much shorter and *simpler* then a logical derivation of the witness. Moreover, while a constraint solver may detect inconsistencies and thereby *guided* the search in the overall proof, the verification is *guided* by $\mu\mathcal{CAS}$ which eliminates the search altogether.

After briefly introducing proof planning and a planning method that actually employs both CAS and CS, we address the integration issues for both types of external reasoners. In Sec. 4 we demonstrate the planning process for a concrete example from proof planning limit theorems [14], the LIM+ theorem: the limit of the sum of two functions equals the sum of their limits.

2 Proof Planning

As opposed to classical theorem proving that is based on exhaustive search at the calculus-level, proof planning employs abstract plan operators, called *methods*, that encapsulate (mathematical) proof techniques such as diagonalization and induction. Technically, Bundy [4] introduced these methods as (partial) specifications of tactics known from tactical theorem proving [8].

The basic approach to proof planning is essentially that of AI-planning [7]. In the ΩMEGA system [2], a planning state is a set of sequents that is divided into *goals* and *assumptions*. A proof planning problem is defined by an initial state specified by the proof assumptions and the goal g given by the theorem to be proved. Planning methods represent (inference) actions and specify preconditions of the action and its effects on the planning state. The planner searches for a solution, i.e., a sequence of actions that transforms the initial state into a state with no goals. Roughly, the planner searches backward for an instantiated method `M` whose application proves a goal g and introduces `M` into the plan. The subgoals needed for the application of `M` replace g in the planning state. The planner continues to search for methods applicable to a subgoal and terminates if no open goals are left or if no further method can be applied. A recursive expansion of the methods in a complete plan yields a calculus-level proof, e.g., a ND-proof that can be proof checked.

ΩMEGA's methods are structures whose format was introduced in [11]. They have the slots *premises, conclusions, application-conditions*, and *proof schema*. *Premises* are (annotated [1]) sequents that are used by a method to *logically derive* the *conclusions*, and *conclusions* are (annotated) sequents which

[1] with $\oplus$ and $\ominus$ for add- and delete-sequents, respectively, in STRIPS' notation.

method: `ComplexEstimate`(a, b, e_1, ϵ)	
premises	(0), $\oplus$(1),$\oplus$ (2), $\oplus$(3)
conclusions	$\ominus th$
appl.cond.	$\exists \sigma. GetSubst(a, b) = (\sigma)$ $\exists k, l. CAS\_split(a_\sigma, b) = (k, l)$
proof schema	(0).$\Delta \ \vdash \ \|a\| < e_1$ () (1).$\Delta \ \vdash \ \|k\| < \mathbf{M}$ (OPEN) (2).$\Delta \ \vdash \ \|a_\sigma\| < \epsilon/2 * \mathbf{M}$ (OPEN) (3).$\Delta \ \vdash \ \|l\| < \epsilon/2$ (OPEN) (4). $\ \vdash \ b = b$ (=Ref) (5). $\ \vdash \ b = k * a_\sigma + l$ (`CAS` (4)) th. $\Delta \ \vdash \ \|b\| < \epsilon$ (`fix`;(5)(0)(1)(2)(3))

Fig. 1. The `ComplexEstimate` Method

the method is designed to prove. Roughly, the annotations indicate the *dynamic planning behavior* of methods in the planning process. When a method is applied, a $\ominus$ conclusion is deleted from the planning state as a goal, a $\oplus$ premise is added as a new subgoal (backward planning), a $\ominus$ premise is deleted as an assumption, and a $\oplus$ conclusion is added as an assumption (forward planning). For more details see [11]. As an example we consider the method `ComplexEstimate` displayed in Fig. 1, a method for estimating the magnitude of the absolute value of a complex term.[2] For instance, when `ComplexEstimate` is applied, the goal $|b| < \epsilon$ is removed from the planning state and the goals $|k| < \mathbf{M}$, $|a_\sigma| < \epsilon/2 * \mathbf{M}$, and $|l| < \epsilon/2$ are added.

The *application-conditions (appl.cond.)* are formulated in a meta-language and restrict the applicability of a method, in particular the instantiations of its parameters. The method is applicable with an instantiation $\mathcal{I}$ of parameters, if for $\mathcal{I}$ the *application-condition* evaluates to *true*.

In case a fixed proof of *conclusion* from *premises* is known, *proof schema* is filled with a declarative schematic representation of this proof. The lines in the *proof schema* contain a label, a sequent, and a line-justification. A line-justification can be a name of a ND-rule, a tactic, the name of an external reasoner such as OTTER or CAS, a method, or OPEN. Additionally, the justification may contain supporting lines. For instance,

$$th. \quad \Delta \vdash |b| < \epsilon \qquad (\texttt{fix};(5),(0),(1),(2),(3)),$$

has the label th, the sequent $\Delta \vdash |b| < \epsilon$, and the justification (`fix`;(5),(0),(1),(2),(3)). The latter indicates that the sequent can be derived from the (support)lines (5),(0),(1),(2), and (3) by `fix` which is an abbreviation for a sub-

[2] In fact, the actual method is more general as the inequality symbol is an additional parameter. For the sake of clarity, however, we work with an instance of the method that employs $<$.

proof using ND-rules. The *proof schema* is used for the expansion of the method.

`ComplexEstimate` estimates the magnitude of the absolute value of a complex term such as $(f(x) - g(x)) - (L_1 - L_2)$. Among others, it is used in proof planning limit theorems which we present in section 4. The planner handles `ComplexEstimate` as follows.

- If an open line in the planning state matches th and if an assumption matching (0) is available, `ComplexEstimate`'s parameters a, b, e_1, ϵ are instantiated by the matcher.
- Then *application-conditions* is evaluated, i.e., the procedure *GetSubst*(a, b) is invoked which returns a substitution σ or fails. σ maps some variables of a to variables of b, in order to equal sub-terms which represent non-arithmetic functions in both terms. In case *GetSubst* returns a substitution, the function *CAS_split* is applied to the terms a_σ, b and may return two terms k and l such that $b = k * a_\sigma + l$.[3] The computations are performed with the computer algebra system MAPLE using the polynomial division function `quo`.
- If the *application-conditions* evaluate to *true*, then the method is applicable and `ComplexEstimate` is introduced into the partial plan, the goal th is removed as planning goal, and the new goals (1), (2), and (3) are introduced instead.
- When the method is expanded later on, the *proof schema* is used. It contains a schematic proof of $\Delta \vdash |b| < \epsilon$ from $b = k * a_\sigma + l$, (0), (1), (2), and (3). The fixed subproof `fix` employs, among others, the triangle inequality $|X + Y| \leq |X| + |Y|$. The line-justification `CAS` indicates the application of a CAS that can justify the equation $b = k * a_\sigma + l$. During the expansion of `ComplexEstimate`, the tactic runs and returns a proof plan for its computation.

Each application of `ComplexEstimate` suggests the existence of a real number $\mathbf{M}$ whose value is restricted by the inequalities (1) and (2). Note that $\mathbf{M}$ is a meta-variable[4] that is used to propagate given value restrictions from universally quantified variables and constants to witnesses of existentially quantified variables.

3 An Integration of External Reasoners

The core of the system is a proof planner that manipulates the plan data structure (PDS) by successively applying methods. The PDS is the fundamental

[3] $/, *, |.|$ denote the division, multiplication, and absolute value functions, When σ denotes a substitution, then F_σ is the result of applying σ to F.

[4] Meta-variables are place holders for instantiations of existentially quantified variables.

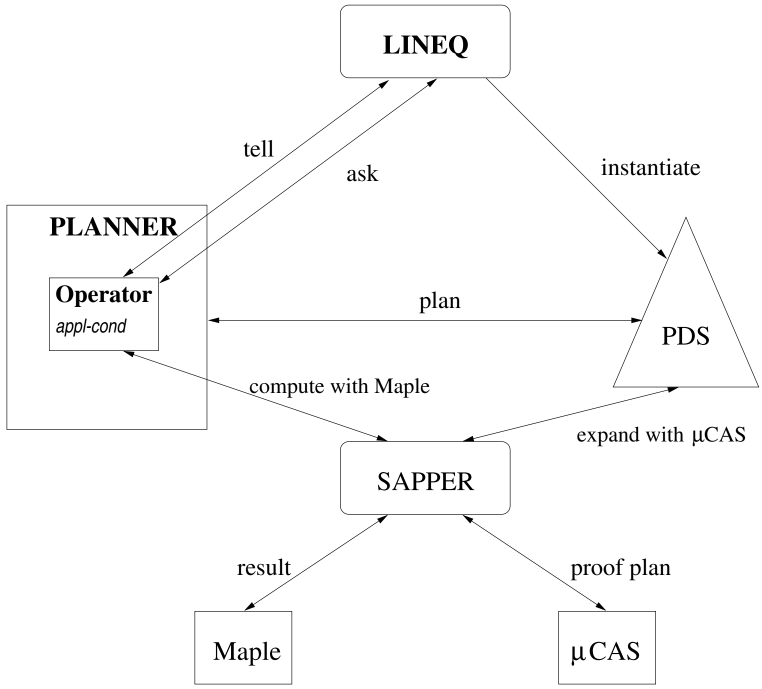

Fig. 2. Integration of Proof Planning with External Reasoners

data structure containing several hierarchical levels of a (partial) proof plan. Automated proof planning is supported by both types of external systems, constraint solvers and computer algebra systems, in several ways. The integration scheme for the systems is displayed in Fig. 2.

The lower half of Fig. 2 depicts the interface connecting the computer algebra system MAPLE and $\mu\mathcal{CAS}$. This interface — called SAPPER — can uniformly integrate arbitrary CAS in both the planning process and the interactive tactical theorem proving within the PDS itself. For an extensive description of SAPPER see [13].

The upper half of Fig. 2 shows how the constraint solver LINEQ is integrated. It is basically only connected to the proof planner itself which both initializes and employs it. Furthermore, LINEQ is able to import the reflection of its constraint store as a conjunction of simple constraints, the *answer constraint*, into the PDS when the proof planner has found some complete plan.

3.1 Integrating CAS's

The integration for CAS via the SAPPER-interface is twofold: Firstly, arbitrary CAS's can be easily employed as blackbox term rewriting systems (similar to approaches of [1,9,6]) and thus SAPPER works as simple bridge between the planner and the CAS. Secondly, SAPPER also has the ability to use a CAS as a proof planner itself. That is, if the CAS can provide additional information on its computations, this information is recorded by SAPPER and translated into a sequence of tactics that can eventually verify the computation. So far there exists the small prototypical CAS, $\mu\mathcal{CAS}$, which basically is a collection of simple algorithms that include a plan generating mode (cf. [13]).

During the planning process, a CAS can be employed to compute a term (or a set of terms) that instantiates a method variable. The computation itself is invoked during the evaluation of the *application-conditions* of a method and the results are bound to the respective method variables. Any such term is 'synthesied' by the CAS and has to be fully verified by a proof checker later. For example, in our method `ComplexEstimate` from Sec. 2, when applying the *CAS_split* function on the terms a_σ and b, MAPLE is called and the polynomial division function `quo` decomposes b into $k * a_\sigma + l$. Since `quo` is essentially Euclid's algorithm, *CAS_split* has to translate the terms a_σ and b into an appropriate form, i.e., replace functions by constants, etc. The result of MAPLE's computation instantiates the method variables k and l.

For the manipulation of the PDS, a CAS can also be used as an efficient term rewriting system to simplify algebraic expressions. Thereby a new proof line is inserted that contains the simplified term and `CAS` as a justification. During the expansion of a method, the CAS can be applied in a similar way. An example for this use is the derivation of (5) from (4) in the `ComplexEstimate` method. The line's justification (`CAS` (4)) indicates that the term $k * a_\sigma + l$ on the right hand side of the equality can be simplified to b by the CAS.

In order to derive a complete calculus-level proof, it is necessary to expand all proof lines not containing an ND-rule as a justification. For this, $\mu\mathcal{CAS}$ can be called in a special plan generating mode in order to return some protocol information on the executed computations. For instance, for the verification of $b = k * a_\sigma + l$, $\mu\mathcal{CAS}$ would return a sequence of tactics indicating single computational steps that have been performed inside the computer algebra algorithm. The sequence of tactics we obtain from $\mu\mathcal{CAS}$'s computation describes another partial proof plan for the verification of the simplification of $b = k * a_\sigma + l$. This proof plan can be inserted into the PDS and further expanded recursively (without any additional calls to a CAS). $\mu\mathcal{CAS}$'s algorithm that produces the justification for MAPLE's computations is of course considerably simpler than `quo`, as it only has to perform multiplication and addition instead of polynomial division.

3.2 Integrating Constraint Solvers

Some search in proof planning can be avoided by incrementally evaluating the restrictions for witnesses of variables, by reducing the space of possible instantiations, and by delaying the instantiation of meta-variables with the help of domain-specific constraint solving. Constraint solvers can check consistency and propagate value restrictions of variables very efficiently because they provide efficient data structures and algorithms for specific domains, e.g., for finite domains [10], for sets [16], and for linear arithmetic in the set of real numbers $\mathcal{R}$ [12]. Therefore, constraint solvers are attached to a particular theory *Th* that is stored in a hierarchically organized mathematical knowledge base of the ΩMEGA system.

For each constraint solver CS that is employed in proof planning, the method `InitializeCS(`Th`)` with a parameter *Th* reduces the goal

$$\Delta \vdash thm$$

to the subgoals

$$\Delta, \mathcal{C} \vdash thm, \qquad \Delta, Th \vdash C,$$

where Th is the theory of CS. For variables of different sorts, different constraint solvers are responsible, e.g., for set variables or for real numbers variables. LINEQ's theory, for instance, is that of linear arithmetic in $\mathcal{R}$. $\mathcal{C}$ is a meta-variable that is introduced for an instantiation C to be computed later from the final constraint store of CS.

In proof planning, the constraint solving component serves three main purposes: First, it is used during the process of proof planning to determine whether a certain method can be legally applied (search restriction). This is implemented by the *application-conditions* of a method that check consistency or entailment of a particular constraint with the constraint store. Secondly, when no open goals are left in proof planning, a reflection of the final constraint store (i.e., the set of all computed constraints) can be assembled to an answer constraint that is used to instantiate the meta-variable $\mathcal{C}$ by the formula C. C is a formula about the value restrictions of the meta-variables representing implicitly existentially quantified variables. From C, every line that was justified by adding a constraint to the constraint store follows. Hence, in a ND-proof that is produced by expanding the proof plan, every goal that was removed by telling it to the constraint solver can be justified. Thirdly, based on the answer constraint the constraint solver can search for instantiations of the meta-variables and return witnesses that are introduced into the PDS.

For instance, for planning proofs of limit theorems our most important method connecting the planner with LINEQ is $\mathtt{Solve}_{<}\mathtt{b}$ (cf. Fig. 3). Its purpose is to remove goals with an inequality between simple terms by adding this constraint to the constraint store as long as it is consistent with the store and not yet included. It can be legally applied if the respective function, *tell* or *ask* returns *true*. The function *tell* checks constraints c for consistency with the current constraint store and propagates c in case of consistency, whereas *ask* checks for entailment of a constraint from the constraint store. The evaluation of the *application-conditions* decides whether to access the constraint solver by *tell* or by *ask*. An expansion of this `Solve` method by the `solv` tactic yields a *simple* proof plan (essentially consisting of `AndElimination`s for inferring

method: $\texttt{Solve}_{<}\texttt{b}(a,b)$	
premises	L1
conclusions	$\ominus$L2
appl.cond.	**if** var-in$(a<b)$ **then** $tell(a<b)$ **else** $ask(a<b)$
proof schema	L1.$\Delta \vdash \mathcal{C}$ () L2.$\Delta \vdash (a<b)$ ($\texttt{solv}(\mathcal{C})$)

Fig. 3. The $\texttt{Solve}_{<}\texttt{b}$ Method

$(a<b)$ from the answer constraint formula C that eventually instantiates $\mathcal{C}$.

However, the goal $\Delta, Th \vdash C$, introduced earlier, needs to be justified by a subproof as well. This subproof can be derived by some additional trace output of LinEq during the constraint solving process. After the final constraint store is derived, unnecessary information is removed (i.e., information on variables that do not occur in the answer constraint) and the rest is translated into a sequence of tactics that eventually lead to a ND proof for C.

4 Example

In this section we examine more closely the cooperation of Maple and LinEq in the proof of an example from the class of limit theorems [14]. Especially, we observe the application of the `ComplexEstimate` method and how the shortcuts introduced into the proof by its application are expanded to ND-level proofs. The actual example is LIM+, that informally states that the limit of the sum of two functions is the sum of their limits. LIM+ can be formalized by:

$$\lim_{x\to a} f(x) = l_1 \wedge \lim_{x\to a} g(x) = l_2 \to \lim_{x\to a}(f(x)+g(x)) = l_1 + l_2.$$

In planning LIM+, the assumption $\Delta \vdash |f(X_1) - l_1| < E_1$ is available at some point. Then the goal $\Delta \vdash |f(x)+g(x)-(l_1+l_2)| < \epsilon$ can be removed by applying the method `ComplexEstimate`. After matching the premise and conclusion lines the method variables a and b are instantiated by $(g(X_2)-l_2)$ and $(f(x)+g(x)-(l_1+l_2))$ respectively. When the *application-conditions* of `ComplexEstimate` are evaluated, *GetSubst* returns $[x/X_2]$ and *CAS_split* uses Maple to compute the instantiations 1 for k and $(g(X_2)-l_2))$ for l. Then the goal $\Delta \vdash |f(x)+g(x)-(l_1+l_2)| < \epsilon$ is replaced by the new goals

(i) $\Delta \vdash |1| < \mathbf{M}$,

(ii) $\Delta \vdash |f(X_1) - l_1| < \epsilon/2*\mathbf{M}$,

(iii) $\Delta \vdash |g(x) - l_2| < \epsilon/2$.

When the method `ComplexEstimate` in the proof plan for LIM+ is expanded, the instantiated proof schema of the method is inserted, yielding among others the line with the formula $(f(x)+g(x)-(l_1+l_2)) = ((1*(g(x)-l_2))+(f(x)-l_1))$ justified by the application of a CAS. In order to gain a pure ND-level proof this line needs to be further expanded. However, since during the application of `ComplexEstimate` the values for l and k were computed by MAPLE, we do not have any additional information for an expansion. To justify the computation in more detail we use an algorithm within our $\mu\mathcal{CAS}$ system in plan generation mode that produces a trace output giving more detailed information on single computational steps. Instead of simulating the complex algorithm for polynomial division within $\mu\mathcal{CAS}$, we simply use an algorithm that simplifies the term on the right-hand side of the equation. Thus, $\mu\mathcal{CAS}$ verifies the result of MAPLE's computation with the help of much simpler algorithm by yielding a proof plan. The proof plan consists of a sequence of tactics indicating single computational steps of the algorithm. Most of these tactics are roughly equivalent to field axioms of the real numbers. Within the PDS, the single step can be expanded to a plan with higher granularity. Some of the newly introduced proof steps are:

$$
\begin{array}{ll}
((1*(g(x)-l_1))+(f(x)-l_2)) & \\
(((1*g(x))-(1*l_1))+(f(x)-l_2)) & \text{(Left Distributivity)} \\
((g(x)-(1*l_1))+(f(x)-l_2)) & \text{(Mult-1-Left)} \\
((g(x)-l_1)+(f(x)-l_2)) & \text{(Mult-1-Left)} \\
\quad\vdots & \\
(f(x)+g(x)-(l_1+l_2)) &
\end{array}
$$

While planning the proof of the LIM+ theorem, the `Solve` methods tell the following constraints to the constraint solver in a row: $0 < D, 1 < \mathbf{M}, E_1 < \epsilon/2 * \mathbf{M}, x = X_1, E_2 < \epsilon/2, x = X_2, D < \delta_2, D < \delta_1$, where $D, E_1, E_2, \delta_1, \delta_2, \epsilon$ stem from the planning problem and $\mathbf{M}$ is the auxiliary meta-variable introduced by `ComplexEstimate`. From the constraints $1 < \mathbf{M}$ and $E_1 < \epsilon/2*\mathbf{M}$ the new upper bound $\epsilon/2$ of E_1 is propagated. This leads to a final constraint store that looks as follows

$$
\begin{array}{ccccl}
0 & < & E_2 & \leq & \epsilon/2; \\
0 & < & D & \leq & \delta_2(E_2), \delta_1(E_1); \\
0 & < & E_1 & \leq & \epsilon/(2*\mathbf{M}), \epsilon/2; \\
1 & < & \mathbf{M} & < & \epsilon/(2*E_1) \\
-\infty & \multicolumn{3}{c}{< X_1 = x = X_2 <} & +\infty
\end{array}
$$

The constraint solver reflects these accumulated value restrictions to assemble the answer constraint $C(D, E_1, E_2, X_1, X_2) : E_1 \leq \epsilon/2 \wedge E_2 \leq \epsilon/2 \wedge D \leq \delta_1 \wedge D \leq \delta_2$ that reads "Let $E_1, E_2 \leq \epsilon/2$ and $D \leq \delta_1, \delta_2$." as known from proofs in mathematics textbooks. The reflection removes redundancies and replaces as many bounds as possible by numeric bounds. When no goals are left in planning, then $\mathcal{C}$ is instantiated by $C(D, E_1, E_2, X_1, X_2)$ all over the proof plan.

The subgoal $\Delta, \mathcal{R} \vdash C(D, E_1, E_2, X_1, X_2)$ has to be proved when expanding the proof plan where $\mathcal{R}$ is the theory of linear arithmetic of the real numbers. For this proof, the constraint solver searches for witnesses of D, E_1, E_2, X_1, X_2 which are then introduced into the proof plan. This way, the formal proof of C becomes much simpler. For instance, it can easily be proved that $min(\delta_1, \delta_2) \leq \delta_1$. The constraint solver's search is currently implemented.

5 Conclusion

We described the integration of computer algebra systems and constraint solvers with a proof planner. On the one hand, the interface comprises functions that connect the external reasoners with the proof planner via methods. On the other hand, it provides functionality that enables the expansion of shortcuts introduced into the proof by the external systems in order to obtain a checkable ND-proof. This expansion guides the formal proof. The same applies for constraint solvers that restrict the search in proof planning. Moreover, since external systems can synthesize witnesses for meta-variables or instantiations of parameters whose formal verification is relatively simple compared to the synthesis, the shortcuts can also simplify the proof process. The guidance and simplification of proofs are main advantages of our integration of external reasoners that have not been exploited in previous uses of theory-reasoning (see, e.g., [5]).

In proof planning limit theorems with the ΩMEGA system, the use of the constraint solver LINEQ and the computer algebra systems MAPLE and μ $\mathcal{CAS}$ resulted in proof plans of theorems that could previously not be proved automatically by general-purpose provers.

References

[1] C. Ballarin, K. Homann, and J. Calmet. Theorems and Algorithms: An Interface between Isabelle and Maple. In A. H. M. Levelt, editor, *Proceedings of ISSAC'95*, pages 150–157, 1995. ACM Press.

[2] C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. ΩMega: Towards a Mathematical Assistant. In W. McCune, editor, *Proceedings of CADE-14*, 1997. Springer.

[3] R.S. Boyer and J S. Moore. Integrating Decision Procedures into Heuristic Theorem Provers: A Case Study of Linear Arithmetic. *Machine Intelligence (Logic and the Acquisition of Knowledge)*, 11:83–124, 1988.

[4] A. Bundy. The Use of Explicit Plans to Guide Inductive Proofs. In E. Lusk and R. Overbeek, editors, *Proceedings of CADE-9*, pages 111–120, 1988. Springer.

[5] H.-J. Bürckert. A resolution principle for constrained logics. *Artificial Intelligence*, 66(2):235–271, 1994.

[6] Edmund Clarke and Xudong Zhao. Analytica-A Theorem Prover in Mathematica. In D. Kapur, editor, *Proceedings of CADE–11*, pages 761–763, 1992. Springer.

[7] R.E. Fikes and N.J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2:189–208, 1971.

[8] M. Gordon, R. Milner, and C.P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation.* Springer, 1979.

[9] J. Harrison and L. Théry. Reasoning About the Reals: The Marriage of HOL and Maple. In A. Voronkov, editor, *Proceedings of LPAR'93*, pages 351–353, 1993. Springer.

[10] P. v. Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.

[11] X. Huang, M. Kerber, M. Kohlhase, and J. Richts. Methods - The Basic Units for Planning and Verifying Proofs. In *Proceedings of KI-94*, 1994. Springer.

[12] J. Jaffar, S. Michaylow, P. Stuckey, and R. Yap. The CLP(R) Language and System. *ACM Transactions on Programming Languages*, 14(3):339–395, 1992.

[13] M. Kerber, M. Kohlhase, and V. Sorge. Integrating Computer Algebra Into Proof Planning. *Journal of Automated Reasoning*, 21(3):327–355, 1998.

[14] E. Melis. The "Limit" Domain. In R. Simmons, M. Veloso, and S. Smith, editors, *Proceedings of AIPS-98*, 1998.

[15] D. Redfern. *The Maple Handbook: Maple V Release 5.* Springer, 1998.

[16] F. Stolzenburg. Membership Constraints and Complexity in Logic Programming with Sets. In F. Baader and U. Schulz, editors, *Frontiers in Combining Systems*, pages 285–302. Kluwer Academic, 1996.