CADE-15

The 15th International Conference on Automated Deduction

July 5-10, 1998, Lindau, Germany

Workshop Proceedings

Using AI methods in Deduction

July 6

Jörg Denzinger, Michael Kohlhase, Bruce Spencer

WORKSHOP Web Site — `http://www.uni-kl.de/AG-AvenhausMadlener/cade-ws.html`

CADE-15 Web Site — `http://www.th-darmstadt.de/cade-15`

Table of Contents

Introduction

Although Automated Deduction has been one of the main subareas of Artificial Intelligence in the early days of this field, the two areas have drifted apart since those days. The idea of a (first-order) logic engine as general problem solver in AI was not able to prove its usefulness, so that other areas of AI, like planning or learning, quite successfully developed other (specialized) means to solve their problems. On the other side, the Automated Deduction community concentrated on extending the theoretical foundations of the field and on developing better base techniques.

In the last years, the growing need for applications of Automated Deduction has resulted in a concentration on deduction *systems*. And, due to the early experiences, deduction systems are not only inference engines anymore. Instead, potential users want to have systems that, in order to assist and help them, have to be able to take advice from the user or other systems, learn from previous proofs and proof attempts, plan their actions, present their results in a user friendly (and acceptable) form, and so on.

As a result, deduction systems must have components like knowledge bases with known proofs, proof tactics, proof plans, different control strategies, and learning and planning components to fill the knowledge bases and to make use of them. So, completeness and other theoretical issues are no longer the main concern of these systems. They focus instead on many techniques from other areas of AI and computer science in general, like user interfaces.

This concentration on systems is indicated by the CADE theorem prover competitions and many workshops in the last few years with subjects concerning systems, their assessment, their control, or their cooperation. Our CADE workshop on *Using AI methods in Deduction* is intended to focus on all the additional components and methods in deduction systems that stem from other areas of Artificial Intelligence. This may seem to be a rather broad range of topics, but in order to face the challenges provided by potential users of deduction systems all these methods have to be integrated into a system (or a team of systems) and they have to support each other.

This can also be seen in the papers collected in these proceedings. The papers of Cheikhrouhou and Siekmann, Melis, and Cheikhrouhou and Sorge deal with achieving a better control of a deduction system by planing proof attempts. This proof planning takes place at an abstracted level which offers a smaller search space. The abstraction is guided by general proof concepts employed by mathematicians, namely diagonalization and techniques for limit theorems. Although the three papers tackle quite different proof methods, nevertheless their results are achieved within one system by expanding its knowledge bases.

The paper of Fisher and Ireland speculates about the future developments of proof planning and combines it with concepts from cooperating multi-agent systems. Cooperation of deduction systems is also the topic of the paper of Fuchs. He addresses the problem of determining what information to communicate between systems employing different proof philosophies in order to achieve synergetical effects. The problem of selecting a good set of deduction systems that cooperate well with each other is tackled in the contribution of Wolf. He uses learning from previous proof runs for solving this problem. Learning from previous proof runs is also used by Schulz. He employs learning to improve the control of a deduction systems, which closes the cycle.

Besides our authors we want to thank CADE-15 (program committee and organizers) for the opportunity to cast a spotlight on the many AI methods used in deduction systems by having this workshop.

Kaiserslautern, Amsterdam, New Brunswick, June 1998

Jörg Denzinger, Michael Kohlhase, Bruce Spencer

Planning Diagonalization Proofs

Lassaad Cheikhrouhou Jörg Siekmann
Fachbereich Informatik, Universität des Saarlandes
{lassaad|siekmann}@cs.uni-sb.de

Abstract

Proof planning is an interesting knowledge-based approach for proof construction, since it allows to incorporate mathematical knowledge and mathematical proof techniques.

The diagonalization proof technique is a well-known method in theoretical computer science and in mathematics that originated with Cantor, who used it to show his seminal uncountability results. It is now widely used as a proof technique for unsolvability results and meta-mathematical arguments.

In this paper we give an account on how to systematically construct and plan diagonalization proofs: (i) by finding an indexing relation, (ii) by constructing a diagonal element, and (iii) by making the implicit contradiction of the diagonal element explicit. We suggest a declarative representation of the strategy and describe how it is realized in the proof planning environment of the ΩMEGA-System.

1 Introduction

In classical (automated) theorem proving the reasoning process is based *on search at the object level*, i.e. at the level of the (first order) logic representation of the mathematical objects under study. Searching for a proof is carried out by applying calculus inference rules to manipulate the initial problem situation, which at the beginning consists of the negated theorem to be proven and the given assertions (definitions, axioms, and possibly other lemmata and theorems), in order to find a final situation, for instance the empty clause $\Box$ in a resolution theorem prover. This guarantees that the theorem is a logical consequence of the given assertions. Tactical theorem proving applies tactics, i.e., a sequence of calculus inference rules, and tacticals are used to compose more complex tactics from basic ones. The essential reasoning remains however mainly at the object level.

Proof planning, as introduced by A. Bundy [Bun91], is based on the search for a sequence of tactics that constitutes a proof plan, which in turn can then be applied to construct an object level proof. The used operators are called methods, they are specifications of tactics represented in a meta-language. They state in this meta-language when a tactic can be applied and what its effects may be. Reasoning is therefore carried out at a meta level. Three main aspects make this approach interesting:

The first aspect of proof planning is that the search for a proof plan can be based on well known mathematical proof techniques such as induction, diagonalization, or the pigeon hole principle, provided the appropriate control knowledge can be found and represented. Such a proof technique characterizes a proof schema which is then instantiated to a sequence of planning steps (which in turn generate object level proofs).

The second aspect of proof planning is the abstraction from mere logical manipulation of formulae by calculus inference rules. For instance, the task of proving an induction conclusion in CLAM [BvHHS90] is tackled by reducing the syntactical difference to the induction hypothesis via the rippling strategy such that it can be used to close the proof path.

The third, not least important, aspect is the naturalness by which mathematical knowledge can be represented and used in the planning process (see [BCF+97]).

The point of proof planning is to analyze proof techniques in order to determine their typical proof steps and to find a suitable control to perform these steps within the proof planning process. The first author carried out an empirical study of many proofs by diagonalization and showed a systematic way how to construct such proofs in [Che96]. In this paper, we describe how this proof technique can be realized in the proof planning environment of the ΩMEGA-System [BCF+97]. In the next section, we introduce the main idea of diagonalization by a formal proof of the Cantor theorem. Thereafter, we give an overview of the proof planning environment in ΩMEGA. Before concluding, we discuss the main methods and show how these can be applied in the Cantor example.

2 Cantor Diagonalization

In order to see the main principle of a proof by diagonalization, consider the Cantor theorem. The theorem states that the power set of each set m has greater cardinality than the set itself, which is equivalent to the conjecture that there is no surjective function from a set into its power set: $\forall m_\bullet \neg \exists f_\bullet \mathrm{surj}(f, m, \mathrm{pset}(m))$. The diagonalization technique was first invented for its proof and it is therefore often called Cantor diagonalization [Kle43]. This theorem can be automatically proven by many (higher order) theorem provers, e.g., `TPS` [AINP90] was the first system that generated a computer proof. This theorem is therefore often used nowadays as a benchmark test for higher order theorem provers, e.g., `LEO` [Ben97] and `HOL` [GM93] generated alternative proofs. Hence we used this example for expository purposes.

To prove the above conjecture, we assume that there is a surjective function f_0 from some set m_0 into its power set $\mathrm{pset}(m_0)$ and deduce a contradiction by diagonalization. In a typical textbook such as [DSW94] a proof by diagonalization turns on the demonstration of two assertions:

1. A certain set E can be enumerated in a suitable fashion.
2. It is possible, with the help of the enumeration, to define an object d that is different from every object in the enumeration, i.e. $d \notin E$.

Below is the diagonalization part of the Cantor proof, where $\mathrm{pset}(m_0)$ is *the enumerated set*. This set can be enumerated with the help of *the indexing relation* f_0 and the set D is the object which is defined with the help of the enumeration. D is then different from every object $f_0(x)$ in the enumeration.

> *The set $D = \{x \in m_0 | x \notin f_0(x)\}$ belongs to $\mathrm{pset}(m_0)$, there is also an element y_0 of m_0 which is the index of D in m_0 ($D = f_0(y_0)$ with $y_0 \in m_0$). By the definition of D y_0 belongs to D iff y_0 is in m_0 and does not belong to $f_0(y_0)$. This is obviously a contradiction to $D = f_0(y_0)$.*

In order to formulate the characteristic proof steps of the above diagonalization argument, we consider the formal proof in Figure 2 of the Cantor theorem which was interactively

Surj-Def	$\forall f_{\iota\to(\iota\to o)} \centerdot \forall a_{\iota\to o} \centerdot \forall b_{(\iota\to o)\to o} \centerdot \mathrm{surj}(f,a,b) \leftrightarrow$ $\forall x_{\iota\to o} \centerdot x \in b \to \exists y_\iota \centerdot y \in a \wedge x = f(y)$
PSet-Def	$\forall a_{\iota\to o} \centerdot \forall x_{\iota\to o} \centerdot x \in \mathrm{pset}(a) \leftrightarrow x \subseteq a$
$\subseteq$-Def	$\forall a_{\iota\to o} \centerdot \forall b_{\iota\to o} \centerdot a \subseteq b \leftrightarrow \forall x_\iota \centerdot x \in a \to x \in b$
Cantor	$\forall m_{\iota\to o} \centerdot \neg\exists f_{\iota\to(\iota\to o)} \centerdot \mathrm{surj}(f, m, \mathrm{pset}(m))$

Figure 1: A formulation of the '**Cantor**' theorem

constructed by the ΩMEGA-System using the problem description in Figure 1. This proof was interactively constructed at the level of a natural deduction (ND) calculus, i.e. it was generated by the application of suitable ND rules [Gen35]. It was then abstracted to the so-called assertion level [Hua94], where assertions, in addition to ND rules, can be used as justifications.

The key steps in the diagonalization part of the proof in Figure 2 are:

- the property, that the function $\lambda z \centerdot z \in m_0 \wedge \neg z \in f_0(z)$ belongs to the power set, is stated in line 9,
- the application of the definition of surjectivity (**Surj-Def**) in line 10 to prove the existence of an index for the function $\lambda z \centerdot z \in m_0 \wedge \neg z \in f_0(z)$, which is assumed to be y_0, is stated in line 11,
- applying the function $\lambda z \centerdot z \in m_0 \wedge \neg z \in f_0(z)$ to the index y_0 is done in line 14 to obtain an implicit contradiction in line 16,
- the contradiction is made explicit by a case analysis in lines 17 to 25.

The analysis of the above key proof steps suggests a systematic way, how to construct a diagonalization proof:

The central point of diagonalization is the construction of the diagonal element, which is an element of the set to be enumerated, that is different from every object in the enumeration. In Figure 2 the diagonal element is represented by a lambda expression that has the indexing function f_0 as a sub-term (see line 9). It is therefore convenient to search for the indexing function first before trying to construct the diagonal element.

In the Cantor proof, the function f_0 binds not only the diagonal element but also each element of the enumerated set $\mathrm{pset}(m_0)$ to an element (its index) in m_0. This property follows from the surjectivity of the function f_0 from m_0 into $\mathrm{pset}(m_0)$ and is represented by the formula:

$$\forall x_{\iota\to o} \centerdot x \in \mathrm{pset}(m_0) \to \exists y_\iota \centerdot y \in m_0 \wedge x = f_0(y)$$

The indexing property provides important information for the specification of the diagonal element: its type (a functional type corresponding to the element type of $\mathrm{pset}(m_0)$), and its domain type (same type as the element type of m_0).

In addition to these type constraints, the diagonal element must be different from each element of the enumerated set $\mathrm{pset}(m_0)$, i.e. from each $f_0(z)$. In the Cantor proof this is achieved by enforcing that for each z the diagonal element differs from the element $f_0(z)$ in some property. We call this property *the diagonal property* which is a proposition that depends on the term $f_0(z)(z)$ which we call *the diagonal term*. In the Cantor proof, the diagonal property is represented by the conjecture $z \in f_0(z)$ which is a syntactic sugar of the diagonal term itself. The diagonal element inverts this diagonal property (occurrence of $\neg z \in f_0(z)$ in the lambda expression representing the diagonal element in line 9).

1.	1	$\vdash \exists f_{\blacksquare} \mathrm{surj}(f, m_0, \mathrm{pset}(m_0))$	`(Hyp)`
2.	2	$\vdash \mathrm{surj}(f_0, m_0, \mathrm{pset}(m_0))$	`(Hyp)`
3.	3	$\vdash x_1 \in \lambda z_{\blacksquare}[z \in m_0 \wedge \neg[z \in f_0(z)]]$	`(Hyp)`
4.	3	$\vdash [x_1 \in m_0 \wedge \neg[x_1 \in f_0(x_1)]]$	`(LambdaE 3)`
5.	3	$\vdash x_1 \in m_0$	`(AndEL 4)`
6.		$\vdash [x_1 \in \lambda z_{\blacksquare}[z \in m_0 \wedge \neg[z \in f_0(z)]] \to x_1 \in m_0]$	`(ImpI 5)`
7.		$\vdash \forall x_{\blacksquare} x \in [\lambda z_{\blacksquare}[z \in m_0 \wedge \neg[z \in f_0(z)]] \to x \in m_0]$	`(ForallI 6)`
8.		$\vdash \lambda z_{\blacksquare}[z \in m_0 \wedge \neg[z \in f_0(z)]] \subseteq m_0$	`(⊆-Def 7)`
9.		$\vdash \lambda z_{\blacksquare}[z \in m_0 \wedge \neg[z \in f_0(z)]] \in \mathrm{pset}(m_0)$	`(Pset-Def 8)`
		Proof of 16	
10.	2	$\vdash \exists y_{\blacksquare}[y \in m_0 \wedge \lambda z_{\blacksquare}[z \in m_0 \wedge \neg[z \in f_0(z)]] = f_0(y)]$	`(Surj-Def 2 9)`
11.	11	$\vdash [y_0 \in m_0 \wedge \lambda z_{\blacksquare}[z \in m_0 \wedge \neg[z \in f_0(z)]] = f_0(y_0)]$	`(Hyp)`
12.	11	$\vdash \lambda z_{\blacksquare}[z \in m_0 \wedge \neg[z \in f_0(z)]] = f_0(y_0)$	`(AndER 11)`
13.		$\vdash y_0 \in f_0(y_0) = y_0 \in f_0(y_0)$	`(=-Refl)`
14.	11	$\vdash y_0 \in \lambda z_{\blacksquare}[z \in m_0 \wedge \neg[z \in f_0(z)]] = y_0 \in f_0(y_0)$	`(=-Subst 12 13)`
15.	11	$\vdash [y_0 \in \lambda z_{\blacksquare}[z \in m_0 \wedge \neg[z \in f_0(z)]] \leftrightarrow y_0 \in f_0(y_0)]$	`(=-Equiv 14)`
16.	11	$\vdash [[y_0 \in m_0 \wedge \neg[y_0 \in f_0(y_0)]] \leftrightarrow y_0 \in f_0(y_0)]$	`(LambdaE 15)`
		Case 1	
17.	17	$\vdash y_0 \in f_0(y_0)$	`(Hyp)`
18.	11,17	$\vdash \neg[y_0 \in f_0(y_0)]$	`( 16 17)`
19.	1,2,11,17	$\vdash \bot$	`(NotE 18 17)`
		Case 2	
20.	20	$\vdash \neg[y_0 \in f_0(y_0)]$	`(Hyp)`
21.	11	$\vdash y_0 \in m_0$	`(AndEL 11)`
22.	1,2,11,20	$\vdash y_0 \in f_0(y_0)$	`( 16 21 20)`
23.	1,2,11,20	$\vdash \bot$	`(NotE 20 22)`
24.		$\vdash [y_0 \in f_0(y_0) \vee \neg[y_0 \in f_0(y_0)]]$	`(TND)`
25.	1,2,11	$\vdash \bot$	`(OrE 24 19 23)`
		End of Case Analysis	
26.	1,2	$\vdash \bot$	`(ExistsE 10 25)`
27.	1	$\vdash \bot$	`(ExistsE 1 26)`
28.		$\vdash \neg[\exists f_{\blacksquare} \mathrm{surj}(f, m_0, \mathrm{pset}(m_0))]$	`(NotI 27)`
29.		$\vdash \forall m_{\blacksquare} \neg[\exists f_{\blacksquare} \mathrm{surj}(f, m, \mathrm{pset}(m))]$	`(ForallI 28)`

Figure 2: A formal proof of the '**Cantor**' theorem

In order to get a contradiction, the diagonal element is constructed in such a way, that it belongs to the enumerated set $\mathrm{pset}(m_0)$ (occurrence of $z \in m_0$ in the lambda expression representing the diagonal element in line 9). Consequently, the diagonal element has an index y_0 and the diagonal property for this element of m_0 ($y_0 \in f_0(y_0)$) is contradicted according to the construction principle of the diagonal element.

In [Che96], the first author studied other proofs by diagonalization which are similar but different in some details from the above Cantor proof in Figure 2. Exploiting these differences a Diagonalization Proof Strategy was suggested that can be applied to prove all the examples in [Che96]. In the rest of this paper, we describe how this proof strategy can be implemented in the proof planning framework of ΩMEGA to construct diagonalization proofs.

3 Proof Planning in ΩMEGA

The central data structure in ΩMEGA is the so-called *proof plan data structure* ($\mathcal{PDS}$) which represents a partial proof of the theorem to be shown at several levels of abstraction. The levels correspond to different abstractions of the formal ND (natural deduction) proof. A $\mathcal{PDS}$ level is a proof tree with the theorem as its root and the hypotheses as its leaves. The child nodes in such a proof tree are either the premises for the justification of their *closed* parent node or the *supports* for their *open* parent node. The supports of an *open* node consist of a heuristically determined subset of the hypotheses of this node and of their deduced consequences, which should be used to close this open node.

A justification of a closed node denotes that the conjecture of this node follows logically by the stated *inference method* from the conjectures of the premises. "*Inference method*" is the common designation of base calculus inference rules (ND rules) and coherent compositions of these rules. A *non-base inference method* can be a tactic, a proof schema that represents a proof idea (technique) with still some gaps, or a call of an external reasoner its returned result can be transformed into a ND proof. Non-base inference methods abstract many individual inference steps by grouping them into one proof step at the next higher level of abstraction.

An initial $\mathcal{PDS}$ consists of an open node, the theorem, and its supports, the hypotheses. A formal ND proof of this problem is constructed by the application of inference methods to open nodes and the expansion of non-base inference methods. This construction process can be done interactively and/or with the help of a planner which uses the inference methods as plan operators. Plan operators, designated as *methods*, are actually specifications that represent applicability condition and application effects of inference methods. In ΩMEGA, we use a declarative representation as much as possible for methods, such that meta-methods can be employed to reformulate methods in analogy-driven theorem proving [Mel98]. A method is represented as a frame-like structure (see Figure 3), the main slots are described below:

- The *proof schema* represents the proof part that should be inserted into a $\mathcal{PDS}$ when expanding the method. It consists of *method nodes* that have to be matched to ND lines, and of *method variables* which are place holder for lists of ND lines.
- The *conclusions* comprise annotated method nodes that correspond to the ND lines closed by the method. The annotations $\oplus$ and $\ominus$ specify whether the associated ND line is added as a new support in the $\mathcal{PDS}$ or it is reduced to new open ND lines.
- The *premises* consist of possibly annotated method nodes that correspond to the ND lines, from which the conclusions logically follow. The non-annotated premises and the premises annotated by $\ominus$ must match some supports of the goal, the method contributes to its proof. The associated ND line to a premise annotated by $\oplus$ is added as an open node to the $\mathcal{PDS}$ and can be regarded as a subgoal of the method. This new open node inherits the supports of the considered goal excluding the ND lines which are associated to the premises annotated by $\ominus$.
- The *sidecondition* specifies additional restrictions for the application of the method.

Other optional slots that enable to prescribe additional control and computation information are explained whenever they occur in the definition of the presented methods.

We distinguish *reliable* and *non-reliable* methods. A method is non-reliable, when its expansion can lead to new open nodes. For instance, a method that calls a resolution theorem prover, e.g., `OTTER` [McC90], is non-reliable, because the considered problem may not be solved by this prover within the given resource limit. The expansion of a reliable method on the other hand, e.g., a specification of a tactic application, leads to a proof at a more detailed level. This distinction is important, because, during the search for a proof, only open nodes and nodes justified by non-reliable methods are taken into consideration by the planner.

While constructing a $\mathcal{PDS}$ we focus on the current tasks by keeping references, *goals* to the open ND lines and *pseudo-goals* to the ND lines closed by non-reliable methods in an agenda, which is a set of partially ordered tasks. The conjectures in the $\mathcal{PDS}$ can contain partially specified terms, i.e. terms with meta-variables. These meta-variables are place holder for

Method : Diag	
Conclusions	$(\ominus 3)$
Premises	$1\ (\oplus Ps)$
Sidecondition	$[match(formula(2), \Phi)\ \&\ (P \leftarrow\ 1)]\ \vert$ $[assert(\Phi, formula(2), Ps)\ \&\ (P \leftarrow\ 2)\ \&\ (J \leftarrow\ ass\_just(1, Ps))]$
Orderings	$(Ps < 3)$
Proof Schema	1. $\vdash$ Φ $()$ 2. $\vdash$ $\forall x_{\alpha\to\beta} \centerdot E(x) \to \exists y_\alpha \centerdot N(y) \wedge x = F(y)$ (J) 3. $\vdash$ $\perp$ (Diag-by P)

Figure 3: The Diag Method

object terms that are partially specified by some constraints that become more and more refined as the search for a proof plan continues.

4 Planning Diagonalization Proofs

We shall now present the main methods for the construction of proof plans for diagonalization and describe how these methods can be applied.

4.1 The Diagonalization Methods

The diagonalization proof technique leads to a proof by contradiction and reduces the job to two main tasks: first, search for an indexing property, and then construct a diagonal element. We represent this strategy by two methods: `Diag`, and `Diag-by`. All of the following is a slight simplification for expository purposes, the full set of methods and more details can be found in [Che97].

4.1.1 The `Diag` Method

The `Diag` method in Figure 3 is used to prove a contradiction, the goal of this method is therefore $\perp$. The application of this method is restricted by its sidecondition to goals with at least one support node that can be used to prove an indexing relation. The method `Diag` can be applied, when there is a support node of the goal that either matches the formula schema $\forall x_{\alpha\to\beta} \centerdot E(x) \to \exists y_\alpha \centerdot N(y) \wedge x = F(y)$ (the formula of line 2 in the proof schema of `Diag`, where E, N, and F are meta-variables and α, and β are type variables.), or can *assert* this formula schema. A formula asserts another formula (schema), when the latter matches a sub-formula of the former. The remaining sub-formulae correspond to the premises Ps in the justification of the asserted formula (schema) and are considered as subgoals of `Diag`.

For instance, for the Cantor proof, the agenda consists of the goal $\perp$ after applying the tactic sequence `ForallI`, `NotI`, and `ExistsE`. The method `Diag` can be applied to this goal, because the surjective definition **Surj-Def** is a support node of the current goal that asserts the indexing property using the premise $\mathrm{surj}(F, N, E)$.

The method `Diag` is non-reliable, since its proof schema contains among others a node justified by the non-reliable method `Diag-by`, which we describe in the next paragraph. Therefore, the node justified by `Diag` is included as pseudo-goal on the agenda. In case premise 1 of `Diag` asserts the indexing property, as in the Cantor example, we obtain additional subgoals which correspond to the premises of the assertion application. These new goals on the agenda must be worked off before the newly inserted pseudo-goal, i.e., before expanding the method `Diag` (compare the orderings-slot in Figure 3).

4.1.2 The `Diag-by` Method

The `Diag-by` method (see Figure 4) is not used by the planner to close goals, it is only used in the context of the method `Diag`. `Diag-by` can be applied to justify a goal $\bot$ using an indexing relation (see line 1 in Figure 4). The expansion of this method turns into the construction of the diagonal element D.

<table>
<tr><th colspan="2">Method : <code>Diag-by</code></th></tr>
<tr><td>Conclusions</td><td>9</td></tr>
<tr><td>Premises</td><td>1 (⊕2) (⊕7)</td></tr>
<tr><td>Constraint</td><td>$\mathrm{newconst}(i,\alpha)$ & $\mathrm{occurs}(i, D(i))$ & $\mathrm{differs}(D, \lambda x.x)$ &
$[\{\mathrm{equals}(\beta, o) \gg \mathrm{bind}(IP, \lambda x, y, z.x(y) \leftrightarrow \neg z(y))\} \mid$
$\{\mathrm{differs}(\beta, o) \gg$
$[\mathrm{match}(D, \lambda x.\mathrm{if}(U(F(x), x), G_1(x), G_2(x)))$ & $\mathrm{differs}(G_1, G_2) \gg$
$\mathrm{bind}(IP, \lambda x, y, z.(Q(z(y)) \to \neg U(z,y)) \wedge Q(G_1(y)) \wedge (R(z(y)) \to U(z,y)) \wedge R(G_2(y)))] \mid$
$\mathrm{bind}(IP, \lambda x, y, z.x(y) \neq z(y)) \mid$
$\mathrm{bind}(IP, \lambda x, y, z.U(x(y)) \leftrightarrow \neg U(z(y)))\}]$</td></tr>
<tr><td>Proof Schema</td><td>
<table>
<tr><td>1.</td><td></td><td>$\vdash$</td><td>$\forall x_{\alpha \to \beta}.E(x) \to \exists y_{\alpha}.N(y) \wedge x = F(y)$</td><td><code>()</code></td></tr>
<tr><td>2.</td><td></td><td>$\vdash$</td><td>$E(D)$</td><td><code>(Open)</code></td></tr>
<tr><td>3.</td><td></td><td>$\vdash$</td><td>$\exists y.N(y) \wedge D = F(y)$</td><td><code>(Ass 1 2)</code></td></tr>
<tr><td>4.</td><td>4</td><td>$\vdash$</td><td>$N(i) \wedge D = F(i)$</td><td><code>(Hyp)</code></td></tr>
<tr><td>5.</td><td>4</td><td>$\vdash$</td><td>$N(i)$</td><td><code>(AndEL 4)</code></td></tr>
<tr><td>6.</td><td>4</td><td>$\vdash$</td><td>$D = F(i)$</td><td><code>(AndER 4)</code></td></tr>
<tr><td>7.</td><td>4</td><td>$\vdash$</td><td>$IP(D, i, F(i))$</td><td><code>(Open)</code></td></tr>
<tr><td>8.</td><td>4</td><td>$\vdash$</td><td>$\bot$</td><td>(<code>MEC</code>(D) 7 6)</td></tr>
<tr><td>9.</td><td></td><td>$\vdash$</td><td>$\bot$</td><td>(<code>ExistsE</code>(i) 3 8)</td></tr>
</table>
</td></tr>
</table>

Figure 4: The `Diag-by` Method

The diagonal element D is a function that belongs to the enumerated set E and inverts some property wrt. the diagonal term $F(i)(i)$. The first property can be stated by closing the subgoal corresponding to the open line 2 in Figure 4. The second property can be fulfilled by a proposition which depends on β, the type of $F(i)(i)$, and on the instantiation of D. This is the reason why we represent the inverting property (line 7 in the proof schema of 4) as a formula schema with the constrained meta-variable IP.

To obtain sensible instantiations of D, we use the restriction constraints: $\mathrm{occurs}(i, D(i))$, and $\mathrm{differs}(D, \lambda x.x)$ (compare the constraint-slot in Figure 4). A vague specification of D can be given by $D = \lambda x_{\alpha}.G(F(x), x)$, that requires the inverting property $U(G(F(i), i)) \leftrightarrow \neg U(F(i)(i))$. Alternative instantiations of these schemata, that were obtained from the examples in [Che96], can make this task easier, as they provide more control:

1. The diagonal term $F(i)(i)$ denotes a proposition: we consider the formula schema $\neg F(i)(i) \leftrightarrow D(i)$ as an inverting property.
2. Otherwise: two important possible instantiations of D are distinguished:
   2.1. $D(x)$ can be defined according to some condition $U(F(x), x)$: $D(x)$ equals $G_1(x)$, if $U(F(x), x)$ holds, and it is $G_2(x)$ otherwise. Thus, D must be instantiated by the schema $\lambda x_{\alpha}.if(U(F(x), x), G_1(x), G_2(x))$, where the constraint $\mathrm{differs}(G_1, G_2)$ must hold. The inversion of the term $F(i)(i)$ is obtained, if we can prove the

subgoals: $Q(F(i)(i)) \to \neg U(F(i), i)$, $Q(G_1(i))$, $R(F(i)(i)) \to U(F(i), i)$, and $R(G_2(i))$.

2.2. After proving the subgoal 2, D can be instantiated with $\lambda x_{\alpha}.C(F(x), x)$: the inverting property of D can be either the inequality $C(F(i), i) \neq F(i)(i)$ or the formula schema $U(C(F(i), i)) \leftrightarrow \neg U(F(i)(i))$.

If we succeed and obtain a complete $\mathcal{PDS}$ after applying the `Diag-by` method, we can expand this $\mathcal{PDS}$ to an ND proof, which can then be checked by the verifier.

The two methods `Diag` and `Diag-by` can lead to goals with meta-variables. Meta-variables are progressively instantiated while closing subgoals that are represented by formula schemata. This is done by middle out reasoning (MOR) [KBB93] with the help of methods which (heuristically) impose alternative instantiations for meta-variables and/or binding constraints.

4.2 Methods for Middle-Out-Reasoning

In this section we are concerned with subgoals represented by formula schemata. In general, *rigid subgoals*, i.e., subgoals of the form $P(t_1, .., t_n)$ where the function P is not a meta-variable, can be closed by assertion application or by simplification tactics.

For instance, after applying the `Diag` method in the Cantor theorem, we obtain the subgoal $\mathrm{surj}(F, N, E)$ (see the previous section). This subgoal can be closed by a method `MvarWeaken3` using the hypothesis $\mathrm{surj}(f_0, m_0, \mathrm{pset}(m_0))$ (see line 2 in Figure 2). Thus, the meta-variables F, N, and E are instantiated respectively to f_0, m_0, and $\mathrm{pset}(m_0)$. Generally, the method `MvarWeaken3` justifies a ternary atom with schematic arguments by a support node, whose formula unifies this atomic goal.

Simplification tactics may not be applied to *critical goals*, where a goal is critical, iff its splitting by a simplification tactic leads to a *flexible subgoal* and/or a flexible hypothesis. A flexible formula schema is an application of the form $P(t_1, .., t_n)$ where the function P is a meta-variable. A goal, whose splitting delivers such a formula schema, is said to be critical wrt. to the meta-variable P. For instance, the goal schema $p \to MV(c)$ is critical wrt. the meta-variable MV, it may not be simplified by applying the ND rule `ImpI`, because this would deliver the flexible subgoal $MV(c)$. A critical subgoal wrt. a meta-variable MV is suspended at first and we consider other goals until the meta-variable MV involved is instantiated.

Unfortunately, it is possible to obtain a lock situation where all goals on the agenda are either critical or flexible. To raise such a lock, we use a heuristic from [Ble77], where it is suggested that instantiations of set variables in higher order theorems can be used to obtain a first order theorem, which in turn can be proven by a (first order) automated theorem prover. In this procedure, higher order variables (HOV) that occur as heads of atoms are interpreted as sets. According to the position of the corresponding atom wrt. other subformulae in the theorem, a HOV is associated to the maximal possible set. If this HOV occurs several times as the head of an atom in the theorem, then it is instantiated with the intersection of the sets that result from each occurrence. For instance, if we had the goal $\forall x(D(x) \to m_0(x)) \wedge (\neg f_0(x)(x) \to D(x))$ then we would obtain the set $\{x : m_0(x)\}$ for the first occurrence and the set $\mathcal{U}$, which denotes the whole individual set, for the second occurrence. The intersection is clearly the set $\{x : m_0(x)\}$. We use the heuristic *BledsoeH* in the method `ImpBH1` (see Figure 5) to determine an instantiation for the meta-variable MV by considering the critical goals wrt. MV on the agenda starting with the goal of this method.

In the Cantor example, the expansion of the `Diag-by` method raises in two new goals

Method : `ImpBH1`	
Conclusions	$(\ominus 3)$
Premises	
Constraint	$\mathrm{bind}(MV, BledsoeH(MV, formula(3)))$
Proof Schema	1. 1 $\vdash$ $MV(arg)$ (Hyp) 2. 1 $\vdash$ ϕ (AndEL 1) 3. $\vdash$ $MV(arg) \to \phi$ (ImpI 2)

Figure 5: The `ImpBH1` Method

$\mathrm{pset}(m_0)(D)$ and $D(i) \leftrightarrow \neg f_0(i)(i)$. The former goal is reduced to the subgoal $D(a) \to m_0(a)$ after applying first the power set definition **PSet-Def**, then the subset definition **$\subseteq$-Def**, and finally the tactic `ForallI`. Since the resulting subgoal is critical wrt. D, we consider next the inverting property $D(i) \leftrightarrow \neg f_0(i)(i)$. This goal can be simplified by successfully applying the tactics `EquivI`, and `AndI` to the critical subgoals wrt. D $D(i) \to \neg f_0(i)(i)$, and $\neg f_0(i)(i) \to D(i)$. Consequently, we obtain a lock situation, where we have to apply one of the methods based on Bledsoe's heuristic. Here, we can close the first critical goal $D(a) \to m_0(a)$ by the method `ImpBH1` that binds D to the function $\lambda x_{\blacksquare} m_0(x) \wedge \neg f_0(x)(x)$ which can be interpreted as the set $\{x : m_0(x) \wedge \neg f_0(x)(x)\}$.

5 Conclusion and Future Work

The main advantage of proof planning for diagonalization proofs comes into play, when applying this technique to other, more difficult problems that are beyond the reach of search based classical systems. Just as the rational reconstruction of an induction theorem prover in the framework of proof planning (as in the work of the Edinburgh group [BSvH+93]) shows how to use this important proof technique in a more general setting, proof planning for diagonalization proofs can now be used for a great variety of theorems. Several textbooks on theoretical computer science and meta-mathematics were scanned and all theorems shown there by the diagonalization technique were isolated (see [Che96] for a report) in order to extract the general principles behind this construction. The rather general formulation of the above diagonalization method was motivated by the aim to prove all these theorems (semi-) automatically and we expect a system for diagonalization proofs to become just as essential as the well-known systems for induction proofs.

In this paper we defined the main methods for planning diagonalization proofs. The methods `Diag` and `Diag-by` correspond to a general proof schema for diagonalization proofs. The `Diag` method is applied to prove a contradiction, when an indexing property can be asserted. The expansion of the goal closed by `Diag` amounts to the application of the method `Diag-by` which in turn specifies the diagonal element according to the proven indexing property. The diagonal element is defined (constructed) while closing the subgoals of the method `Diag-by`. In this construction process we use methods for MOR to deal with schematic goals. Some of these methods are described in section 4.2, more material can be found in [Che97].

A first implementation of the represented methods succeeded to compute automatically a formal proof for the Cantor theorem mentioned in this paper. The planner in ΩMEGA is currently extended to treat schematic goals and to manage a constraint pool, after this the proofs for all the examples in [Che96] (including the Halting problem, the theorem that the set of total computable functions is not recursive enumerable, the theorem that total functions are not enumerable, etc.) should run without user interactions.

The Edinburgh group of Alan Bundy also used the framework of proof planning for diagonalization proofs. This work by Toby Walsh and Jeremy Gow was carried out independently using the CLAM system [Gow97].

References

[AINP90] P. B. Andrews, S. Issar, D. Nesmith, and F. Pfenning. The TPS theorem proving system. In M. E. Stickel, editor, *Proceedings of the 10th CADE*, pages 641–642, Kaiserslautern, Germany, 1990. Springer Verlag, Berlin, Germany, LNAI 449.

[BCF+97] 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 the 14th Conference on Automated Deduction*, LNAI, Townsville, Australia, 1997. Springer Verlag.

[Ben97] C. Benzmüller. A Calculus and a System architecture for Extensional Higher-Order Resolution. Report 97-198, Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh and Universität des Saarlandes, Juni 1997.

[Ble77] W. W. Bledsoe. A Maximal Method for Set Variables in Automatic Theorem Proving. Memo ATP-33, Math. Dept., Univ. of Texas, Februar 1977.

[BSvH+93] A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, **62**:185–253, 1993.

[Bun91] A. Bundy. A Science of Reasoning. In *Computational Logic: Essays in honor of Alan Robinson*. MIT Press, 1991. also presented at the 10th CADE 1990 as extended abstract.

[BvHHS90] A. Bundy, F. van Harmelen, Ch. Horn, and A. Smaill. The OYSTER-CLAM system. In M. E. Stickel, editor, *Proceedings of the 10th CADE*, pages 647–648, Kaiserslautern, Germany, 1990. Springer Verlag, Berlin, Germany, LNAI 449.

[Che96] L. Cheikhrouhou. The Mechanization of the Diagonalization Proof Strategy. SEKI Report SR-96-14, Fachbereich Informatik, Universität des Saarlandes, Im Stadtwald, Saarbrücken, Germany, 1996.

[Che97] L. Cheikhrouhou. Planning Diagonalization Proofs. SEKI Report SR-97-20, Fachbereich Informatik, Universität des Saarlandes, Im Stadtwald, Saarbrücken, Germany, 1997.

[DSW94] M. D. Davis, R. Sigal, and E. J. Weyuker. *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science*. Academic Press, second edition, 1994.

[Gen35] G. Gentzen. Untersuchungen über das logische Schließen I. *Mathematische Zeitschrift*, **39**:176–210, 1935.

[GM93] M. J. C. Gordon and T. F. Melham. *Introduction to HOL*. Cambridge University Press, Cambridge, United Kingdom, 1993.

[Gow97] J. Gow. The Diagonalization Method in Automatic Proof. Master thesis, Department of Artificial Intelligence, University of Edinburgh, Scotland, 1997.

[Hua94] X. Huang. Reconstructing Proofs at the Assertion Level. In A. Bundy, editor, *Proceedings of the 12th CADE*, pages 738–752, Nancy, France, 1994. Springer Verlag, Berlin, Germany, LNAI 814.

[KBB93] I. Kraan, D. Basin, and A. Bundy. Middle-Out Reasoning for Program Synthesis. In P. Szeredi, editor, *Proceedings of the 10-th International Conference on Logic Programming*. MIT Press, 1993.

[Kle43] S. C. Kleene. Recursive Predicates and Quantifiers. In M. Davis, editor, *The Undecidable: Basic Papers On Undecidable Propositions, Unsolvable Problems And Computable Functions*, pages 254–287. Raven Press, Hewlett, New York, 1965, 1943.

[McC90] W. McCune. Otter 2.0. In M. E. Stickel, editor, *Proceedings of the 10th CADE*, pages 663–664, Kaiserslautern, Germany, 1990. Springer Verlag, Berlin, Germany, LNAI 449.

[Mel98] E. Melis. The Heine-Borel challenge problem: In honor of Woody Bledsoe. *Journal of Automated Reasoning*, 21, 1998. forthcoming.

Planning Equivalence Proofs

Lassaad Cheikhrouhou and Volker Sorge
Fachbereich Informatik
Universität des Saarlandes
{lassaad|sorge}@cs.uni-sb.de

Abstract

Different axiomatizations of mathematical concepts prove to be useful in a mathematical knowledge base, since each axiomatization of a concept is more or less helpful for the task at hand. To keep the knowledge base consistent, the equivalence of distinct definitions for some concept must be formally proven.

Especially in algebra, where various axiomatizations of an algebraic structure often occur, the proofs of equivalent definitions are canonical in major simplification steps. Starting out with a proof for the equivalence of two distinct definitions for the group concept, we report in this paper on the first attempts of proof planning the equivalence of different definitions for algebraic structures in the ΩMEGA system. We present a proof method that implements the common simplification steps of such theorems and give some relevant methods for closing the subgoals of this method.

1 Introduction

For many proofs of mathematical theorems it is often desirable to choose an appropriate axiomatization for an occurring concept that meets the requirements of the specific problem formulation. To provide this opportunity within a theorem proving environment it is necessary to supply knowledge bases which can contain several parallel axiomatizations for the same mathematical concept. In order to ensure consistency of these kind of knowledge bases it is necessary to provide means for checking the equivalence of given definitions of the same entity. Those problems arise in various domains such as algebra and computation theory. Especially in algebra the proofs of equivalence for parallel definitions can be done in a very canonical way and consist of similar major steps.

In this paper we report on the first attempts of planning such equivalence proofs in the ΩMEGA mathematical assistance system [BCF+97]. We emphasize on developing a mechanism that automates the canonical proof scheme for algebraic structures. However, this canonical proof scheme does not correspond to a well-known mathematical proof technique as induction or diagonalization.

In particular we will discuss a systematic way how to construct proofs of equivalence of different axiomatizations of the same algebraic structure in a proof planning framework. We present a well-known example of such a proof from group theory as background for the extraction of the abstract general proof scheme for various algebraic structures. This general scheme is represented in a strategic method that splits the problem into subproblems which can be partially tackled by regular planning methods. We assume that some of these regular methods work analogously in different algebraic domains and some have to be domain specific. We discuss a few of those methods, both general and typical for group theory.

An important point in the proof of equivalent definitions is the setting of some concepts typical to the one definition, like the operators, some distinguished elements and functions, in termini of the concepts typical to the other definition. We sketch some first ideas how to construct those terms in an appropriate manner using middle-out-reasoning (MOR) [KBB93] methods. Especially with regard to the application of MOR the work reported here can be seen as test bed for finding a balanced and well-working mixture between proof planning, the use of automated reasoners and user interaction.

2 Example: Definitions of Group

To give an impression of the class of problems we are concerned with we give two examples of different yet equivalent definitions for a group. The first one is a standard axiomatization that can be found in most text books.

Definition 1 (Group-1): Let G be a non-empty set, then G together with binary operation '$\cdot$' is a group if the following properties hold:

$\mathcal{G}1$) For all $a, b \in G$ exists $c \in G$ with $a \cdot b = c$.

$\mathcal{G}2$) For all $a, b, c \in G$ holds $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.

$\mathcal{G}3$) There exists a $e \in G$ such that for all $a \in G$ holds $e \cdot a = a$ and $a \cdot e = a$.

$\mathcal{G}4$) There exists a unary function $^{-1}$ such that for all $a \in G$ holds $a \cdot a^{-1} = e$ and $a^{-1} \cdot a = e$.

Another possible definition of a group is obtained by expressing the given operation differently without the explicit use of the concept of associativity.

Definition 2 (Group-2): Let G be a non-empty set, then G together with binary operation '$/$' is a group if the following properties hold:

$\mathcal{H}1$) For all ordered pairs (a, b) where $a, b \in G$ exists a $c \in G$ with $a/b = c$.

$\mathcal{H}2$) For all $a, b \in G$ holds $a/a = b/b$.

$\mathcal{H}3$) For all $a, b \in G$ holds $a/(b/b) = a$.

$\mathcal{H}4$) For all $a, b, c \in G$ holds $(a/a)/(b/c) = c/b$.

$\mathcal{H}5$) For all $a, b, c \in G$ holds $(a/c)/(b/c) = a/b$.

In the remainder of this section we formalize definitions 1 and 2, and describe the formal proof that both definitions are equivalent[1]. We will use a higher order logic based on classical type theory [Chu40] as introduced in [And86], and define the set of base types as the set consisting of the type of truth values o and the type of individuals ι. We then let a set G be of type $\iota \rightarrow o$ and let each element of G be of type ι. Hence a binary operation on G is of type $(\iota, \iota) \rightarrow \iota$ and the property of an e_ι being element of the set G can be modeled as the univariate predicate $G(e)$. We formalize proofs in natural deduction calculus [Gen35] and denote them in the linearized style also introduced in [And86]. In order to preserve readability of the formulas we will, wherever possible, omit the type information and introduce an additional notational convention by writing variables in capital letters and constants in small letters.

We formalize the theorem that definitions 1 and 2 are equivalent straightforward as

[1]The complete informal proof is stated in [Hal59] together with a variety of other definitions for the concept of group.

$$\mathbf{T}.\quad \vdash \forall G_{\bullet}[\exists Op_1{}_{\bullet}\texttt{group-1}(G, Op_1) \Leftrightarrow \exists Op_2{}_{\bullet}\texttt{group-2}(G, Op_2)],$$

were `group-1` and `group-2` are abbreviations for the actual definitions of the groups. The definitions themselves are represented as λ-abstractions and associated to the respective predicate using a special definition operator '$=_{def}$'. In detail the definitions are formalized as

$$\begin{aligned}\mathbf{D}1.\quad \texttt{group-1} =_{def} \lambda G_{\bullet}\lambda Op_{(\iota,\iota)\to\iota}{}_{\bullet}&\texttt{not-empty}(G) \wedge \texttt{closed-under}(G, Op) \wedge \\ &\texttt{associative}(G, Op) \wedge \exists E_{\bullet}[G(E) \wedge \texttt{unit}(G, Op, E) \wedge \\ &\exists I_{\iota\to\iota}{}_{\bullet}[\texttt{inverse-func}(G, Op, E, I)]]\end{aligned}$$

$$\begin{aligned}\mathbf{D}2.\quad \texttt{group-2} =_{def} \lambda G_{\bullet}\lambda Op_{(\iota,\iota)\to\iota}{}_{\bullet}&\texttt{not-empty}(G) \wedge \texttt{closed-under}(G, Op) \wedge \\ &\texttt{quasi-unit}(G, Op) \wedge \texttt{quasi-right-unit}(G, Op) \\ &\texttt{reciprocal}(G, Op) \wedge \texttt{cancellation}(G, Op)\end{aligned}$$

Each axiom of the respective group definitions is yet again represented as predicates and not as compound propositions. This enables the recognition of similar concepts in different definitions on an abstract level. However, the predicates are once more defined as λ-abstraction and can be introduced in the proof when necessary. For instance, the concept of non-emptiness `not-empty` is defined as the λ-term $\lambda G_{\bullet}\exists X_{\bullet}G(X)$.

Considering theorem **T** we illustrate the typical major steps in proofs of equivalence of group definitions.

1. Split the equivalence
2. Expand the group-definitions into single subgoals
3. Prove the resulting subgoals

We examine the above points step by step and point out the main difficulties for the automation of the proof. Step 1 is simple as it only involves the elimination of the universally quantified variable G and the splitting of the equivalence. Assuming we instantiate g for the variable G we get the two new subgoals

$$\mathbf{S}1.\quad \exists Op_1{}_{\bullet}\texttt{group-1}(g, Op_1) \vdash \exists Op_2{}_{\bullet}\texttt{group-2}(g, Op_2),$$
$$\mathbf{S}2.\quad \exists Op_2{}_{\bullet}\texttt{group-2}(g, Op_2) \vdash \exists Op_1{}_{\bullet}\texttt{group-1}(g, Op_1),$$

to which the steps 2 and 3 will be applied in turn.

In step 2 the abstract predicates `group-1` and `group-2` are replaced by their respective definitions and the resulting formulas simplified by interwoven elimination of existentially quantified variables and splitting of conjunctions. This procedure can at first be applied to the antecedents of the two subgoals only. Here each of the existentially quantified variables can directly be instantiated with simple witness terms and any resulting conjunction can be split. When expanding the succedents, however, the elimination of existential quantifiers is less trivial as possible instantiations for variables generally depend on the witness terms introduced beforehand.

We first consider subgoal **S**1 and assume that, when expanding the left hand side of the derivation, the occurring variables Op_1, E, and I are instantiated by the constants $\cdot$, e, and $^{-1}$ respectively. The set **H** in Figure 1 displays all hypotheses obtained after splitting all conjunctions in the instantiated definition. In order to perform the similar expansion step on the right hand side of the subgoal the variable Op_2 has to be instantiated with a λ-term depending on both $\cdot$ and $^{-1}$, namely $\lambda XY_{\bullet}(X \cdot Y^{-1})$. This expansion step leads then to the open subgoals (1)–(6) displayed in Figure 1 which are then proved separately in step 3.

The first subgoal (1) can be directly closed by the first hypothesis, whereas the remaining subgoals are further expanded using the respective definitions. The available hypotheses are

$$
\begin{array}{rcll}
\mathbf{H} & \vdash & \texttt{not-empty}(g) & (1)\\
\mathbf{H} & \vdash & \texttt{closed-under}(g, \lambda XY.X \cdot Y^{-1}) & (2)\\
\mathbf{H} & \vdash & \texttt{quasi-unit}(g, \lambda XY.X \cdot Y^{-1}) & (3)\\
\mathbf{H} & \vdash & \texttt{quasi-right-unit}(g, \lambda XY.X \cdot Y^{-1}) & (4)\\
\mathbf{H} & \vdash & \texttt{reciprocal}(g, \lambda XY.X \cdot Y^{-1}) & (5)\\
\mathbf{H} & \vdash & \texttt{cancellation}(g, \lambda XY.X \cdot Y^{-1}) & (6)
\end{array}
$$

$$
\begin{aligned}
\mathbf{H} = \ & \{\texttt{not-empty}(g), \texttt{closed-under}(g, \cdot), \texttt{associative}(g, \cdot),\\
& g(e), \texttt{unit}(g, \cdot, e), \texttt{inverse-func}(g, \cdot, e, ^{-1})\}
\end{aligned}
$$

Figure 1: Subgoals for proof of example 2

expanded as well in order to close the resulted subgoals. Hereby, we employ besides basic simplification tactics, as the splitting of a conjunction subgoal to its conjuncts, tactics based on equality inferences. For instance, subgoal (3) in Figure 1 is expanded to the subgoal $[\forall A, B.(g(A) \wedge g(B)) \Rightarrow A \cdot A^{-1} = B \cdot B^{-1}]$ which is reduced by simplification tactics to the equality $[a \cdot a^{-1} = b \cdot b^{-1}]$. Both sides of this equality are reduced to the same constant e employing equalities that result from the expansion of hypothesis $\texttt{inverse-func}(g, \cdot, e, ^{-1})$ to the formula $[\forall X.g(X) \Rightarrow (g(X^{-1}) \wedge X^{-1} \cdot X = e \wedge X \cdot X^{-1} = e)]$.

For the proof of subgoal **S2** three different existentially quantified variables need to be instantiated with term depending on the operator Op_2. Assuming / is introduced as a witness term for Op_2, the necessary term to instantiate Op_1 is $\lambda XY.X/((Y/Y)/Y)$. The instantiated goal and the new hypothesis are expanded using the associated definitions and are subsequently split. We obtain among others the subgoal

$$\exists E.[g(E) \wedge \texttt{unit}(g, \lambda XY.X/((Y/Y)/Y), E) \wedge \exists I.\texttt{inverse-func}(g, \lambda XY.X/((Y/Y)/Y), E, I)]$$

Here appropriate terms must be given for the variables E and I. While the inverse function I corresponds to the λ-term $\lambda X.(X/X)/X$ which depends on / only, the unit element E depends additionally on some element of g. The fitting term can be derived using the definitions of $\texttt{not-empty}(g)$ and $\texttt{closed-under}(g, \cdot)$ and corresponds to (X'/X') where X' is an arbitrary free variable.

The rest of the proof for **S2** can then be accomplished similarly to the one of **S1** by expansion and application of both basic simplification tactics and tactics based on equality inferences.

3 A General Planning Strategy

In this section we generalize the equivalence proof presented above to a methodology that can be used in the proof planning framework of ΩMEGA [BCF+97] to construct similar proofs for equivalence of definitions of various algebraic structures. After giving an overview on the proof planning framework of ΩMEGA, we outline a strategic method that implements the first simplifying steps in such proofs and we describe some (domain) specific methods that should be used to solve the resulting subgoals.

3.1 Proof Planning in ΩMEGA

The ΩMEGA system is an interactive proof development environment where proofs can be constructed with the support of various automatic tools, such as a proof planner, automated theorem provers, etc. The central data structure in ΩMEGA is the so-called *proof plan data structure* ($\mathcal{PDS}$) which represents a partial proof of the theorem to be shown at several levels of abstraction. The levels correspond to different abstractions of the formal ND (natural deduction) proof. A $\mathcal{PDS}$ level is a proof tree with the theorem as its root and the hypotheses as its leaves. The child nodes in such a proof tree are either the premises for the justification of their *closed* parent node or the *supports* for their *open* parent node. The supports of an *open* node consist of a heuristically determined subset of the hypotheses of this node and of their deduced consequences, which should be used to close this open node.

A justification of a closed node denotes that the conjecture of this node follows logically by the stated *inference method* from the conjectures of the premises. "*Inference method*" is the common designation of basic calculus inference rules (ND rules) and coherent compositions of these rules. A *non-basic inference method* can be a tactic, a proof schema that represents a proof idea (technique) with still some gaps, or a call to an external reasoner whose returned result can be transformed into a ND proof. Non-basic inference methods abstract many individual inference steps by grouping them into one proof step at the next higher level of abstraction.

An initial $\mathcal{PDS}$ consists of an open node, the theorem, and its supports, the hypotheses. A formal ND proof of this problem is constructed by the application of inference methods to open nodes and the expansion of non-basic inference methods. This construction process can be done interactively and/or with the help of a planner which uses the inference methods as plan operators. Plan operators, designated as *methods*, are actually specifications that represent applicability condition and application effects of inference methods. In ΩMEGA, we use a declarative representation as much as possible for methods, such that meta-methods can be employed to reformulate methods in analogy-driven theorem proving [Mel98]. A method is represented as a frame-like structure, the meaning of the slots in Figure 2 will be explained in the next section where we describe the method `Equiv-Structure`.

3.2 The `Equiv-Structure` Method

We first present the main considerations leading to a strategic method that is applicable for a variety of algebraic structures by generalizing the equivalence statements for the definitions of groups in section 2 and thereby gaining a generic scheme for the theorems stating the equivalence between two different definitions. The general theorem can be formulated as $\forall \mathcal{S}. \exists \mathcal{O}_1. \mathrm{struct}_1(\mathcal{S}, \mathcal{O}_1) \Leftrightarrow \exists \mathcal{O}_2. \mathrm{struct}_2(\mathcal{S}, \mathcal{O}_2)$ where calligraphic letters stand for finitely many terms. The respective definitions of the considered algebraic structure are designated by struct_1 and struct_2. The involved sets are represented by $\mathcal{S}$; $\mathcal{O}_1$ and $\mathcal{O}_2$ correspond to the operations defined on these sets. As an instance of this general theorem scheme one can consider the theorem **T** from section 2. As examples of algebraic structures including more than one operator or more than one set of individuals one could think of definitions for *fields* or *modules*.

The first major steps for proving such a generic equivalence theorem are implemented in the strategic method `Equiv-Structure` in figure 2. The *proof schema* of `Equiv-Structure` represents the proof part that should be inserted into a $\mathcal{PDS}$ when expanding this method. Only the boundaries of the proof schema, i.e., the method *conclusions* and the method

Method : `Equiv-Structure`	
Conclusions	$(\ominus\ 16)$
Premises	$(\oplus\mathcal{G}_1)\ (\oplus\mathcal{G}_2)$
Computation	$\mathcal{S}' \leftarrow \mathrm{Var2Const}(\mathcal{S})$, $\mathcal{O}'_1 \leftarrow \mathrm{Var2Const}(\mathcal{O}_1),\ \underline{\mathcal{O}'_2} \leftarrow \mathrm{Var2Mvar}(\mathcal{O}_2)$, $\Psi_{\mathcal{S}',\mathcal{O}'_1} \leftarrow \mathrm{ApplyDefn}(\mathrm{struct}_1, <\mathcal{S}', \mathcal{O}'_1>),\ \mathcal{L}_1 \leftarrow \mathrm{AndExistsE}{*}(\Psi_{\mathcal{S}',\mathcal{O}'_1})$ $\Phi_{\mathcal{S}',\underline{\mathcal{O}'_2}} \leftarrow \mathrm{ApplyDefn}(\mathrm{struct}_2, <\mathcal{S}', \underline{\mathcal{O}'_2}>),\ <\mathcal{G}_1, \underline{\mathcal{M}}_1> \leftarrow \mathrm{AndExistsI}{*}(\Phi_{\mathcal{S}',\underline{\mathcal{O}'_2}})$ $\mathcal{O}'_2 \leftarrow \mathrm{Var2Const}(\mathcal{O}_2),\ \underline{\mathcal{O}'_1} \leftarrow \mathrm{Var2Mvar}(\mathcal{O}_1)$, $\Pi_{\mathcal{S}',\mathcal{O}'_2} \leftarrow \mathrm{ApplyDefn}(\mathrm{struct}_2, <\mathcal{S}', \mathcal{O}'_2>),\ \mathcal{L}_2 \leftarrow \mathrm{AndExistsE}{*}(\Pi_{\mathcal{S}',\mathcal{O}'_2})$ $\Xi_{\mathcal{S}',\underline{\mathcal{O}'_1}} \leftarrow \mathrm{ApplyDefn}(\mathrm{struct}_1, <\mathcal{S}', \underline{\mathcal{O}'_1}>),\ <\mathcal{G}_2, \underline{\mathcal{M}}_2> \leftarrow \mathrm{AndExistsI}{*}(\Xi_{\mathcal{S}',\underline{\mathcal{O}'_1}})$
Actions	$\ll \mathcal{G}_1 : +(\mathcal{L}_1), -(8,\ 10) \gg \ \ll \mathcal{G}_2 : +(\mathcal{L}_2), -(1,\ 3) \gg$
Proof Schema	1. $\quad\vdash \mathrm{struct}_2(\mathcal{S}', \mathcal{O}'_2)$ (`Hyp`) 2. 1 $\vdash \Pi_{\mathcal{S}',\mathcal{O}'_2}$ (`DefnE`(struct_2) 1) 3. $\quad\vdash \exists\mathcal{O}_2.\mathrm{struct}_2(\mathcal{S}', \mathcal{O}_2)$ (`Hyp`) 4. 1 $\vdash \Xi_{\mathcal{S}',\underline{\mathcal{O}'_1}}$ (`AndExistsI*`$(\underline{\mathcal{M}}_2)$ $\mathcal{G}_2$) 5. 1 $\vdash \mathrm{struct}_1(\mathcal{S}', \underline{\mathcal{O}'_1})$ (`DefnI`(struct_1) 4) 6. 1 $\vdash \exists\mathcal{O}_1.\mathrm{struct}_1(\mathcal{S}', \mathcal{O}_1)$ (`ExistsI*`$(\underline{\mathcal{O}'_1})$ 5) 7. 3 $\vdash \exists\mathcal{O}_1.\mathrm{struct}_1(\mathcal{S}', \mathcal{O}_1)$ (`ExistsE*`$(\mathcal{O}'_2)$ 3 6) 8. $\quad\vdash \mathrm{struct}_1(\mathcal{S}', \mathcal{O}'_1)$ (`Hyp`) 9. 8 $\vdash \Psi_{\mathcal{S}',\mathcal{O}'_1}$ (`DefnE`(struct_1) 8) 10. $\quad\vdash \exists\mathcal{O}_1.\mathrm{struct}_1(\mathcal{S}', \mathcal{O}_1)$ (`Hyp`) 11. 8 $\vdash \Phi_{\mathcal{S}',\underline{\mathcal{O}'_2}}$ (`AndExistsI*`$(\underline{\mathcal{M}}_1)$ $\mathcal{G}_1$) 12. 8 $\vdash \mathrm{struct}_2(\mathcal{S}', \underline{\mathcal{O}'_2})$ (`DefnI`(struct_2) 11) 13. 8 $\vdash \exists\mathcal{O}_2.\mathrm{struct}_2(\mathcal{S}', \mathcal{O}_2)$ (`ExistsI*`$(\underline{\mathcal{O}'_2})$ 12) 14. 10 $\vdash \exists\mathcal{O}_2.\mathrm{struct}_2(\mathcal{S}', \mathcal{O}_2)$ (`ExistsE*`$(\mathcal{O}'_1)$ 10 13) 15. $\quad\vdash \exists\mathcal{O}_1.\mathrm{struct}_1(\mathcal{S}', \mathcal{O}_1) \Leftrightarrow \exists\mathcal{O}_2.\mathrm{struct}_2(\mathcal{S}', \mathcal{O}_2)$ (`EquivSplit 14 7`) 16. $\quad\vdash \forall\mathcal{S}.\exists\mathcal{O}_1.\mathrm{struct}_1(\mathcal{S}, \mathcal{O}_1) \Leftrightarrow \exists\mathcal{O}_2.\mathrm{struct}_2(\mathcal{S}, \mathcal{O}_2)$ (`ForallI*`$(\mathcal{S}')$ 15)

Figure 2: The `Equiv-Structure` Method

premises are considered in the application of the associated method.

The *conclusions* of `Equiv-Structure` consist of an annotated method node that corresponds to the ND line closed by this method. The annotation $\ominus$ specifies that the associated ND line is reduced to new open ND lines. Thus, the `Equiv-Structure` method can be applied to goals whose formulas match the formula of line 16.

The *premises* of `Equiv-Structure` consist of two annotated method variables that represent list of ND lines, from which the conclusions logically follow. The associated ND lines to premises annotated by $\oplus$ are added as open nodes to the $\mathcal{PDS}$ and can be regarded as the subgoals of the method. In contrast to the conclusions of the `Equiv-Structure` method, we have to use method variables instead of method nodes to represent the premises, since the latter can not be specified declaratively. The slot *computation* comprises the function calls needed for determining the method subgoals.

Generally, the supports of each method subgoal correspond to those of the considered goal and to the new hypotheses inserted by this method. Moreover, the support list of a subgoal can be affected by the method's *actions* which specify which ND lines have to be eliminated from this list and which lines must be added.

When the conclusion line of the `Equiv-Structure` method (line 16) is successfully matched with an open ND line in the current $\mathcal{PDS}$, the method variables $\mathcal{S}, \mathcal{O}_1, \mathrm{struct}_1, \mathcal{O}_2$, and struct_2

are simultenously bound to appropriate object terms. Using these bindings, the computation instructions of `Equiv-Structure` determine the new ND lines to be associated to the method variables $\mathcal{G}_1$, $\mathcal{L}_1$, $\mathcal{G}_2$, and $\mathcal{L}_2$. Considering the example in section 2, $\mathcal{L}_1$ would be exactly the set $\mathbf{H}$ in Figure 1 and $\mathcal{G}_1$ consists of the subgoals (1)–(6) displayed in the same Figure with a meta-variable $\underline{Op}_2$ instead of the λ-term $\lambda XY_{\bullet}(X \cdot Y^{-1})$.

Without invoking the tactics in the proof schema of `Equiv-Structure`, the computation functions allow to determine the method subgoals depending on the given goal. In the sequel of this section, we outline these computation instructions and discuss how they are related to the tactics in the proof schema.

Split the equivalence: The function *Var2Const* computes for the variables bound to $\mathcal{S}$ a list of new constants $\mathcal{S}'$ to be employed as a parameter of the tactic `ForallI*` which justifies line 16 from line 15. The latter line is justified by the tactic `EquivSplit` which splits the rest of the proof schema into two symmetric proof trees eventually leading to the two lists of subgoals $\mathcal{G}_1$ and $\mathcal{G}_2$. Considering the theorem $\mathbf{T}$ in section 2, the introduced constant g for the universally quantified variable G is the sole element of $\mathcal{S}'$ and the subgoals $\mathbf{S}1$ and $\mathbf{S}2$ correspond to the lines 14 and 7 in the proof schema of `Equiv-Structure`.

As the computation instructions for both subgoal lists $\mathcal{G}_1$ and $\mathcal{G}_2$ are similar, we concentrate on the first one and describe how it is incrementally determined by the corresponding function calls in the *computation* slot.

Introduce witness terms: The tactic `ExistsE*` is simulated by the generation of new constants $\mathcal{O}_1'$ for the existential quantified variables in $\mathcal{O}_1$. This tactic proves line 14 using the hypothesis 10 introduced by `EquivSplit` and the line 13 as premises. The latter premise follows from line 12 by the tactic `ExistsI*` whose application is imitated by generating new meta-variables $\underline{\mathcal{O}_2'}$ [2] as place holder for the witness terms of the existential quantified variables in $\mathcal{O}_2$. Since these witness terms can not be determined at this point in the proof, we use meta-variables instead which have to be progressively instantiated while closing schematic subgoals that are represented by formulas containing meta-variables. In the example of section 2, $\mathcal{O}_1'$ would consist of the constant $\cdot$ introduced for the existentially variable Op_1, and the list $\underline{\mathcal{O}_2'}$ has to contain a meta-variable $\underline{Op}_2$ wich is finally instantiated to the λ-term $\lambda XY_{\bullet}(X \cdot Y^{-1})$.

We expect that the appropriate instantiations of meta-variables can be mostly done by the application of middle-out-reasoning (MOR) [KBB93] methods which heuristically reduce schematic goals and restrict the bindings of occurring meta-variables with some constraints. Thereby it should be possible to either directly determine the right instantiations or to use one of ΩMEGAs other reasoning components, such as an automated theorem prover or a constraint solver, to compute the desired term. However, it might be necessary that very tricky instantiations must be provided interactively by the user.

Expand the `Structure` definitions: The next calls of the function *ApplyDefn* substitute the predicates struct_1 and struct_2 with their respective properties. Considering the example in section 2, this function would return the β-normalform of the application of the λ-abstractions in the definitions $\mathbf{D}1$ and $\mathbf{D}2$ to the pairs $(g, \cdot)$, and $(g, \underline{Op}_2)$, respectively.

The resulting formulas $\Psi_{\mathcal{S}',\mathcal{O}_1'}$ and $\Phi_{\mathcal{S}',\underline{\mathcal{O}_2'}}$ are mainly conjunctions of predicates with some existentially quantified variables representing certain elements of the occurring sets (compare definitions $\mathbf{D}1$ and $\mathbf{D}2$, section 2). But as the positions and therefore the scopes of the

[2] Underlined letters denote (lists of) meta-variables.

existential quantifiers are a priori undetermined, a common representation of such formulas can only be approximately described. Let $P_1, \ldots, P_k$ be the properties of structure struct_1. Then we can view the formula $\Psi_{\mathcal{S}',\mathcal{O}'_1}$ as

$$\exists E_1 \ldots E_n . s'_1(E_{j1}) \wedge \ldots \wedge s'_m(E_{jm}) \wedge P_1^{\mathcal{O}'_1,\mathcal{E}} \wedge \ldots \wedge P_k^{\mathcal{O}'_1,\mathcal{E}} \tag{7}$$

where $s'_1, \ldots, s'_m$ are, not necessarily distinct, elements of the set list $\mathcal{S}'$ and $\{E_{j1}, \ldots, E_{jm}\}$ is a subset of $\{E_1, \ldots, E_n\}$. Furthermore the existential quantifications do not need to be at the beginning of the formula but can be intermixed with the conjunctions and the P_i can depend on the operations $\mathcal{O}'_1$ and some of the E_j which is indicated by superscript letters.

Although it is not possible to exactly determine (7), the simplification of this formula can be clearly defined as successive applications of $\wedge$- and $\exists$-eliminations. The latter ones are performed by using new distinct constants e'_j for the variables E_j. The function *AndExistsE** carries out this simplification and returns the list $\mathcal{L}_1$ of ND lines with the formulas $s'_1(e'_{j1}), \ldots, s'_m(e'_{jm}), P_1^{\mathcal{O}'_1,\mathcal{E}'}, \ldots,$ and $P_k^{\mathcal{O}'_1,\mathcal{E}'}$. For instance, this function introduces in the example of section 2 the constants e and $^{-1}$ for the existentially quantified variables E, and I in λ-abstraction defining `group-1` and returns the supports in set **H** of Figure 1.

The elements of $\mathcal{L}_1$ are added as supports for the open nodes in $\mathcal{G}_1$ which are determined by the call of function *AndExistsI** on $\Phi_{\mathcal{S}',\mathcal{O}'_2}$. This function imitates the effects of the tactic `AndExistsI*` by employing $\wedge$- and $\exists$-introduction rules to the given formula, producing a set of new open lines. Moreover the witness terms introduced during the $\exists$-introduction steps are again meta-variables, as we usually can not yet determine the required witness terms.

The open ND lines in $\mathcal{G}_2$ and their supports in $\mathcal{L}_2$ are determined in the same way by the specified function calls. The resulted open ND lines are then inserted as subgoals of the `Equiv-Structure` method. In addition to the actions-slot of this method, more control information is represented by control rules which prefer some specific methods while closing these subgoals. The evaluation of these control rules is restricted to the *scope* of the `Equiv-Structure` method, where a method *scope* corresponds to the subproof tree in the $\mathcal{PDS}$ its root is justified by this method.

3.3 Some Specific Methods

In many cases, the axioms of equivalent definitions of algebraic structures are somehow related to each other. Some axioms of the one definition can have similar, weaker, or stronger axioms of the other definition. Therefore, we employ the methods `same`, `weaken-axiom`, and `strengthen-axiom` as some specific methods to be used while proving the subgoals delivered by the strategic method `Equiv-Structure`:

- The method `same` closes a goal from a support line with the same formula.
- The goal of the method `weaken-axiom` is represented by the formula $\bigwedge_{i=0}^{n} P_i \Rightarrow \bigwedge_{j=0}^{m} C_j$. An open goal matching this formula can be closed by the method, when a support line of the form $\bigwedge_{i=0}^{k} P'_i \Rightarrow \bigwedge_{j=0}^{l} C'_j$ is available, for which $\{P'_0, \ldots, P'_k\}$ is a subset of $\{P_0, \ldots, P_n\}$ and the set of conclusions $\{C_0, \ldots, C_m\}$ is a subset of $\{C'_0, \ldots, C'_l\}$.
- The method `strengthen-axiom` can be used to reduce a goal that matches the formula $\bigwedge_{i=0}^{n} P_i \Rightarrow \bigwedge_{j=0}^{m} C_j$ provided a support line matching $\bigwedge_{i=0}^{k} P'_i \Rightarrow \bigwedge_{j=0}^{l} C'_j$ is given, where the set of premises $\{P'_0, \ldots, P'_k\}$ is a subset of $\{P_0, \ldots, P_n\}$ and the set $\{C'_0, \ldots, C'_l\}$ is a subset of $\{C_0, \ldots, C_m\}$. Each conjunct C_j that does not belong to the set of conclusions $\{C'_0, \ldots, C'_l\}$ is considered as a subgoal to be proven with the additional supports $P_0, \ldots, P_n, C'_0, \ldots, C'_{l-1}$, and C'_l.

As the strategic method `Equiv-Structure` can deliver schematic goals, we need specific methods for MOR which possibly introduce (binding) constraints for the occurring meta-variables. Each of the above methods can be generalized in order to be applicable in a flexible setting, where goals are represented with the help of meta-variables. However, the generalization must be carried out carefully to restrict the application of such methods for sensible situations. For instance, the method `same-MOR` is a generalization of `same` which closes a goal $\varphi(\underline{\mathcal{M}})$ using a hypothesis $\varphi(\mathcal{T})$ and binding the meta-variables in $\underline{\mathcal{M}}$ to the associated terms in $\mathcal{T}$. An additional application condition of this method demands that φ must be matched to a fully instantiated object term, e.g., a predicate symbol.

3.4 Some Domain Specific Methods

In the following we present two typical methods for group theory which are significant in solving the kind of subgoals that can be delivered by the strategic method `Equiv-Structure` applied in a context involving groups. These methods somehow imitate the problem solving behavior of mathematicians. Determining the unit (inverse) element in a group is usually achieved by first finding a left unit (inverse) element and a right unit (inverse) element, and then stating the equality of both elements. This is implemented by the method `unit2lr` (`inverse2lr`):

- The method `unit2lr` should be applied to subgoals resulted by expanding the concept `unit`, i.e., to formulas of the form $[g(\underline{E}) \wedge \forall X. g(X) \Rightarrow (\underline{Op}(\underline{E}, X) = X \wedge \underline{Op}(X, \underline{E}) = X)]$, where $\underline{E}$ and $\underline{Op}$ represent the unit element and the operator respectively and can be matched with schematic terms. The application of this method delivers two subgoals: one for the property of a left unit element $[g(\underline{E}) \wedge \forall X. g(X) \Rightarrow \underline{Op}(\underline{E}, X) = X]$, and a second one for the property of a right unit element $[g(\underline{E_r}) \wedge \forall X. g(X) \Rightarrow \underline{Op}(X, \underline{E_r}) = X]$. The equality of both unit elements $[\underline{E} = \underline{E_r}]$ is proven inside the tactical content of the method using the property of the left unit element.
- The goal of the method `inverse2lr` is a conjunction $[g(\underline{Y}) \wedge \underline{Op}(\underline{Y}, c) = \underline{E} \wedge \underline{Op}(c, \underline{Y}) = \underline{E}]$ stating that some element $\underline{Y}$ is the inverse of an element c, where the meta-variables $\underline{Y}$, $\underline{Op}$, and $\underline{E}$ can be matched with schematic terms. The application of this method deliver the subgoals $[g(\underline{Y})]$, $[\underline{Op}(\underline{Y}, c) = \underline{E}]$, $[\underline{Op}(c, \underline{Y_r}) = \underline{E}]$, and $[\underline{Y} = \underline{Y_r}]$, where $\underline{Y_r}$ stands for a right inverse element of the element c.

Furthermore, we speculate that some other domain specific methods can be defined which are relevant for proving equalities in the context of groups. More examples must be investigated to determine such kind of methods.

4 Conclusion and Future Work

Many mathematical concepts can be described by different axiomatizations which can prove useful for the task at hand. It is therefore appropriate to specify a concept by several equivalent definitions in a mathematical knowledge base. To keep the knowledge base consistent, the equivalence of distinct definitions for some concept must be formally proven. Especially in algebra, where various axiomatizations of an algebraic structure often occur, the proofs of equivalent definitions are canonical in major steps.

We have presented the first steps towards an automation of the planning process of equivalence proofs for different definitions of algebraic structures. The major proof steps are constructed with the strategic method `Equiv-Structure` that is generic enough to handle various algebraic entities. Some more specific methods are then employed to proof single axioms from the given ones. We use middle-out-reasoning to postpone the selection of crucial terms to some point in the proof where more information on the shape of the required terms is available. So far we have tested the approach for several equivalent definitions of groups and we want to test whether the approach is general enough by considering definitions for more complex algebraic structures such as modules.

In some related work [Kun92, McC93] equivalence proofs have been automatically constructed in order to show the existence of single axioms defining groups. But the operations on groups have been explicitly formalized as axioms for the proofs, that is single axioms have been formulated with respect to the functions (for example multiplication and inversion) of the original axiomatization. Moreover the postulation that a group is a non-empty set, secluded with respect to the binary operation has implicitly been assumed.

Our work attempts to assist proving equivalence between axiomatizations in a more general setting by combining general and domain specific knowledge. A proof planning framework like ΩMEGA's, where the object language is a sorted higher order logic with partial functions, allows for a more natural representation of mathematical knowledge and consequently more intuitive proof construction.

So far experiments with the higher order automatic theorem prover TPS [ABI+96] have shown that even simpler versions of the definitions presented in this paper are beyond the scope of current higher order theorem provers. Here proof planning has the advantage that domain specific knowledge can be used to guide the proof search. However, we believe that when carefully employed to prove certain subgoals TPS can not only provide subproofs but also useful instantiations for occurring meta-variables. Currently we are investigating the use of TPS during the middle-out-reasoning process and the application of computer algebra for closing equational subgoals.

References

[ABI+96] Peter B. Andrews, Matthew Bishop, Sunil Issar, Dan Nesmith, Frank Pfenning, and Hongwei Xi. TPS: A Theorem Proving System for Classical Type Theory. *Journal of Automated Reasoning*, 16(3):321–353, 1996.

[And86] P. B. Andrews. *An Introduction To Mathematical Logic and Type Theory: To Truth Through Proof*. Academic Press, San Diego, CA, USA, 1986.

[BCF+97] 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 the CADE–14*, LNAI, Townsville, Australia, 1997. Springer Verlag, Berlin, Germany.

[Chu40] A. Church. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[Gen35] G. Gentzen. Untersuchungen über das Logische Schließen I und II. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935.

[Hal59] M. Hall. *The Theory of Groups*. The Macmillan Company, New York, USA, 1959.

[HKRS94] X. Huang, M. Kerber, J. Richts, and A. Sehn. Planning Mathematical Proofs with Methods. *Journal of Information Processing and Cybernetics*, 30(5/6):277–291, 1994.

[KBB93] I. Kraan, D. Basin, and A. Bundy. Middle-Out Reasoning for Logic Program Synthesis. techreport MPI-I-93-214, Max-Planck-Institut, Saarbrücken, Germany, April 1993.

[Kun92] K. Kunen. Single Axioms for Groups. *Journal of Automated Reasoning*, 9(3):291–308, 1992.

[McC93] W. McCune. Single Axioms for Groups and Abelian Groups with Various Operations. *Journal of Automated Reasoning*, 10(1):1–13, 1993.

[Mel98] E. Melis. The Heine-Borel challenge problem: In honor of Woody Bledsoe. *Journal of Automated Reasoning*, 21, 1998. forthcoming.

Proof Planning for Limit Theorems

Erica Melis*
Universität des Saarlandes, FB Informatik
D-66041 Saarbrücken, Germany
melis@cs.uni-sb.de

Abstract

Proof planning is an alternative methodology to classical automated theorem proving. This paper extends proof planning such that it can cope with genuinely mathematical problems such as the limit theorems. The paper concentrates on an extended architecture of a proof planner, on search control, and on employing domain knowledge in a general proof planning framework. These ideas are implemented in the proof planner OMEGA that has succeeded in automatically planning, e.g., a proof for LIM* a theorem that was beyond the capabilities of other planners and theorem provers previously.

1 Introduction

While humans can cope with long and complex proofs and have strategies to avoid less promising proof paths, traditional automated theorem proving suffers from exhaustive search in super-exponential search spaces. As a potential solution to this problem, proof planning has been introduced by Alan Bundy in [6]. Proof planning tries to reduce search spaces by taking larger steps by using plan operators (methods) that are partial specifications of tactics [11], and by employing global and high-level search control as opposed to the local, low level search heuristics which are used for search control in automated theorem proving.

The Edinburgh group pioneered proof planning as a technique by building the proof planner *CIAM*[8] for inductive proofs that employs the rippling search heuristic for difference reduction. However, there are classes of theorems that are difficult or impossible to prove by state-of-the-art proof planners such as *CIAM* or OMEGA [2] before our extension. Reasons for this failure are that (i) for many problems, the available means of control are insufficient in these systems and (ii) the need to construct mathematical objects with certain properties is not supported. Such constructions occur, for example, in proving theorems about limits. Hence we extended proof planning to knowledge-based proof planning.

In the following, we present knowledge-based proof planning and illustrate it with proof planning limit theorems. In particular, we present

- an architecture for combining a general planning framework with domain knowledge;
- a way to extract mathematical knowledge for planning methods;

*This work was supported by the Deutsche Forschungsgemeinschaft, SFB 378

- means to reduce the search space in proof planning by domain knowledge.

By emphasizing the reduction of search through domain knowledge we aim at meeting Woody Bledsoe's challenge to "pay attention to mathematician's great deal of direction" rather than "cover the eyes with blinders and hunt through a cornfield for a diamond-shaped grain of corn" issued in 1986 [3].

1.1 The Limit Theorems

The class of limit theorems includes the well known theorem LIM+ from elementary calculus that states that the limit of the sum of two functions is the sum of their limits [1]. Other theorems in this class are, e.g., LIM- and LIM*, similar theorems about differences and products, ContIfDeriv that states that a function having a derivative at a point is continuous there, and theorems like $\lim_{x \to a} x^2 = a^2$. These theorems are formulated in the theory of the real numbers $\mathbb{R}$. Bledsoe [4] proposed several versions of LIM+ as a challenge problem for theorem proving, where LIM+ is.

$$\lim_{x \to a} f(x) = L_1 \wedge \lim_{x \to a} g(x) = L_2 \rightarrow \lim_{x \to a} (f(x) + g(x)) = L_1 + L_2, \tag{1}$$

which, after expanding the definition of $\lim_{x \to a}$, becomes

$$\begin{aligned}
&\forall \epsilon_1 \exists \delta_1 \forall x_1 (0 < \epsilon_1 \rightarrow 0 < \delta_1 \wedge (|x_1 - a| < \delta_1 \rightarrow |f(x_1) - L_1| < \epsilon_1)) \wedge \\
&\forall \epsilon_2 \exists \delta_2 \forall x_2 (0 < \epsilon_2 \rightarrow 0 < \delta_2 \wedge (|x_2 - a| < \delta_2 \rightarrow |f(x_2) - L_2| < \epsilon_2)) \\
&\rightarrow \forall \epsilon \exists \delta \forall x (0 < \epsilon \rightarrow 0 < \delta \wedge (|x - a| < \delta \rightarrow |(f(x) + g(x)) - (L_1 + L_2)| < \epsilon)) \ .
\end{aligned}$$

The typical way a mathematician goes about to prove such a theorem is to (incrementally) invent an instantiation of δ that depends on $\epsilon, \delta_1, \delta_2$.

In the remainder, $div, *, +, -, ||$ denote the division, multiplication, addition, subtraction, and the absolute value function in $\mathbb{R}$. We use σ for substitutions and abbreviate the result of applying σ to F by F_σ. Prolog notation is used for (Skolem)constants and variables.

2 Brief Review of Proof Planning

As opposed to classical theorem proving based on search *at the calculus level*, proof planning searches for operators, called methods, that encapsulate mathematical techniques. These methods represent actions that are more abstract than the inference steps in logical calculi and therefore the resulting proof plans are abstract representations of proofs. Furthermore, proof planning can be knowledge-based.

The basic approach to proof planning is essentially that of classical planning in artificial intelligence [9]. In OMEGA a planning state is a set of sequents that is divided into open *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. Methods represent actions and specify preconditions of the action and its effects to 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 a solution is reached. Planning and recursive expansion of the methods in a plan yields a calculus-level proof, e.g., a Natural Deduction (ND)-proof.

2.1 Methods in OMEGA

OMEGA's methods are structures whose format is introduced in [13]. The following description simplifies their actual implementation: Methods have the slots *premises* and *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 the method is designed to prove. Roughly, the annotations indicate the dynamic planning behavior of methods in the planning process, e.g., a $\ominus$ conclusion is deleted 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 instance, in `ComplexEstimate` from section 4.1 $\ominus$thm, $\oplus(1)$, $\oplus(2)$, $\oplus(3)$ means that the method removes the goal thm and introduces (1), (2), and (3) as new subgoals.

The *application-conditions* are formulated in a meta-language and restrict the applicability of a method and the instantiations of its parameters. The method is applicable with an instantiation $\mathcal{I}$ of parameters, if for $\mathcal{I}$ *application condition* evaluates to true.[2]

In case a proof of *conclusion* from *premises* is known, *proof schema* is filled with a declarative schematic representation of the proof. This *proof schema* can be used for the expansion of the method to a lower level plan. The lines in *proof schema* contain a label, a sequent, and a line-justification. Since a proof line can be justified by a calculus rule, by a method, by a tactic, or by invoking an automated theorem prover such as OTTER [14] the line-justification can be a name of an ND-rule, the name of a method, tactic, or a prover, a meta-variable, or OPEN in case the sequent is to be planned for.

3 Extensions of Proof Planners

We extended proof planning in the following way. First, we add a control unit to the planner architecture and control-rules to the domain knowledge. The control unit interprets control-rules in order to restrict the choices of methods and of goals before the planner decides about the applicability of methods, see § 4.2.

The control works on the basis of information about the constraint state, resources, the planning state, and the planning history. Secondly, we add a constraint solver to the domain knowledge, see §4.3 for a very brief explanation. Thereby we can take advantage of a lot of work accomplished by the CLP community. The constraint solver can be accessed via methods and can transform the constraint state.

In OMEGA, domain knowledge is stored in a hierarchically organized mathematical theory knowledge base, where theories have parents they inherit from. For instance, the theory *ordered-field* inherits, among others, from the most basic theory *base*. Previously, axioms and definitions belonged to a theories only. Now the domain knowledge (methods, control-rules, special constraint solvers) belongs to a theory in the theory knowledge base.

[1]annotations $\oplus$ and $\ominus$ for add- and delete-sequents, respectively

[2]formulae $(X \leftarrow A)$ evaluate to true.

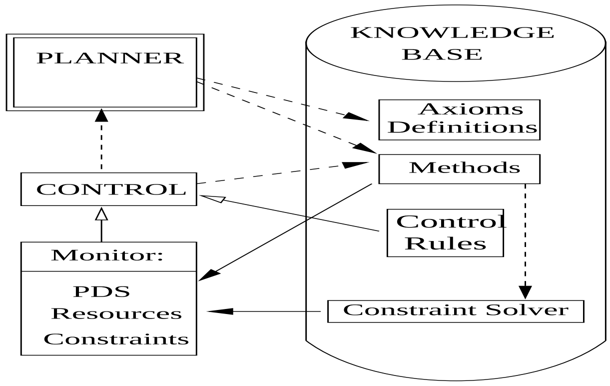


Figure 1: Extended architecture for proof planning.

4 Planning Proofs of Limit Theorems

The limit proofs can be considered prototypical for certain proofs that need to construct witnesses for existentially quantified variables. In this section we present the domain knowledge needed to plan the proofs for limit theorems. Here some methods that belong to the *calculus* domain theory or to its parent theories are presented. The entire set of methods is described in [15].

4.1 Some Methods for Planning Limit Theorems

`ComplexEstimate` is the central method in planning limit theorems. Foremost, the application of the other methods prepares the application of `ComplexEstimate` or prove inequalities. Depending on the particular problem, `ComplexEstimate` has to be applied a different numbers of times. For instance, in planning LIM+ it is applied once and for LIM* it is applied three times.

Note that `ComplexEstimate` is the only method that is used exclusively for planning proofs for limit theorems. It is a rational reconstruction of Bledsoe's limit heuristic in [5]. Compared with Bledsoe‘s program, however, the rational reconstruction in a proof planning framework has several advantages: It relies on a general-purpose problem solver; it provides high-level, representations of proofs that can be expanded to checkable ND-proofs; it employs declarative control knowledge that is modularly organized.

`ComplexEstimate` purpose is to prove an inequality $|B| < E$ by reducing it to inequalities (1), (2), and (3) that contain a new auxiliary variable. For instance it reduces the goal $|f_1(x) + f_2(x) - (l_1 + l_2)| < \epsilon$ to the subgoals

1. $\Delta \vdash |1| < M$
2. $|f(X_1) - l_1| < E_1 \vdash |f(x) - l_1| < div(\epsilon, 2 \cdot M)$
3. $\Delta \vdash |g(x) - l_2| < div(\epsilon, 2)$

method: $\texttt{ComplexEstimate}_<(a, b, e_1, \epsilon)$			
premises	(0), $\oplus$(1),$\oplus$ (2), $\oplus$(3)		
conclusions	$\ominus th$		
appl.cond	$\exists\sigma.\texttt{extract}(a, b) = ((k, l, \sigma)$		
proof schema	(0).Δ	$\vdash \lvert a\rvert < e_1$	()
	(1).Δ	$\vdash \lvert k\rvert < M$	(OPEN)
	(2).Δ	$\vdash \lvert a_\sigma\rvert < \epsilon/2 * M$	(OPEN)
	(3).Δ	$\vdash \lvert l\rvert < \epsilon/2$	(OPEN)
	L0.	$\vdash b = b$	(=Ref)
	L1.	$\vdash b = k * a_\sigma + l$	(CAS L0)
	th. Δ	$\vdash \lvert b\rvert < \epsilon$	(fix;L1,(1)(2)(3))

Each application of `ComplexEstimate` introduces a new auxiliary variable M upon which δ finally depends. M restricts the range of the object δ and its range is restricted by (1), (2), and (3) in turn. M is used to propagate range restrictions from the known constants and variables to δ.

application condition requires that the procedure `extract` returns terms k, l, and a substitution σ. $\texttt{extract}(a, b)$ works as an oracle that tries to compute terms k, l, and σ such that b can be represented as a linear combination of a, i.e., $b = k * a_\sigma + l$ holds. `extract` returns $\perp$ if it did not succeed in finding such k, l, and σ. In this case, the method is not applicable.

proof schema contains a schematic proof of $|B| < E$ from the assumption $|A| < E_1$ and from the premises $B = k * A_\sigma + l$, (1), (2), and (3). The line-justification CAS names a computer algebra tactic integrated into OMEGA that can justify the equation $B = k * A_\sigma + l$. During the expansion of `ComplexEstimate`, CAS runs and returns a proof plan for its computation. The line-justification in th, "fix", is an abbreviation for a fixed subproof using, e.g., the triangle inequality. The planner handles `ComplexEstimate` as follows.

- If a goal 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. That is, the procedure $\texttt{extract}(A, B)$ is invoked. For instance, if A is instantiated by $g(X_2) - l_2$ and B is instantiated by $f(x) + g(x) - l_1 + l_2$, then $\texttt{extract}(a, b)$ returns the list $(1, (g(X_2) - l_2), [X_2/x])$.
- 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, 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 methods $\texttt{Solve}_<\texttt{b and Solve}_<\texttt{f}$ (b for backward and f for forward) handle goals and assumptions, respectively, that are linear inequalities. These methods call the function `tell` that provides an interface between the methods and the constraint solver LINEQ. `tell` takes an inequality $(a < b)$ that is told to the constraint solver and returns *true* if the current constraint store is consistent with $(a < b)$. For the lack of space we describe $\texttt{Solve}_<\texttt{b}$ only.

method: $\texttt{Solve}_{<}\texttt{b}(a,b)$	
premises	L1
conclusions	$\ominus$L2
appl.cond	$tell(a<b)$
proof schema	L1. $\Delta \quad \vdash \mathcal{C} \quad ()$ L2. $\Delta \quad \vdash (a<b) \quad (\texttt{solv}(\mathcal{C}))$

If $\texttt{Solve}_{<}\texttt{b}$ can be applied to a goal $(a < b)$, then the goal is removed from the planning state. Since $\texttt{Solve}_{<}\texttt{b}$ has no preconditions, it produces no subgoals. The applicability is checked as follows. `tell` is invoked and tells its argument $(a < b)$ to the constraint solver. If $(a < b)$ is consistent with the current constraint store, `tell` returns *true* and the operator is applicable.

In *proof schema*, the line-justification solv names a tactic that can recompute $(a < b)$ from the constraint store. During the expansion of $\texttt{Solve}_{<}\texttt{b}$, this tactic runs and provides a proof plan for its computation.

4.2 Mathematical Control Knowledge

In planning, several decisions have to be made. There are choices between alternatives for methods, goals to work on, and for instantiations of variables. Experience in AI-planning [16] indicates the superiority of a separate and explicit representation of control knowledge. In particular, control-rules are well suited for the user's comprehension, for experiments with different rules, for modifications, and for learning control knowledge. This modular and declarative representation of control-rules, can be considered a small expert system.

Our control-rules contain decidable meta-predicates that inspect the current planning state, the planning history, resources, constraint state, and the current partial proof plan. Meta-predicates used in the rules return all satisfying binding alternatives in case an argument is not instantiated; otherwise they return a truth value. Examples for meta-predicates are `current-goal`(x) and `last-operator`(x) which yield instantiations for x, if x is a variable. Otherwise they return the truth value, e.g., of `current-goal`(a).

Currently, we distinguish the classes `method` and `sequent` of control-rules that correspond to the different decisions of the planner. In order to reduce, e.g., the number of alternative methods, the planner presents a list of alternatives to the control-rule interpreter. The interpreter returns a (reduced) list of alternatives via the interface. Only then the specification of methods is matched with goals and assumptions in the planning state.

In the following, we present some of the control-rules that produced a satisfying search behavior in planning limit theorems. Most of the rules are designed in order to capture the following global mathematical control story:

1. Linear inequalities can be proved by direct estimation or by a decomposition that introduces auxiliary variables. 2. For such a decomposition proof, a (part of a) proof assumption has to be used.

The mathematical knowledge can be translated into the following verbally expressed control knowledge that refers to proof planning methods.

1. Open lines with linear inequalities can be removed by $\texttt{Solve}_{<}\texttt{b}$, `SOLVE*`, or by `ComplexEstimate`.

2. The latter requires some preparation by UNWRAPHYP. UNWRAPHYP extracts a subformula s from an assumption in order to afterwards employ s by ComplexEstimate.

This control knowledge can be captured in control-rules such as

```
(control-rule attack-inequality
        (if (and (goal-matches ("goal" (less  "x" "y")))
                 (most-similar-subterm-in ("goal"
                                           "ass"
                                           "pos"))))
         (then
             (prefer ((Solve<b "goal")
                      (SOLVE* "goal")
                      (UNWRAPHyp () ("ass") ("goal"))))))
```

The intention of attack-inequality is an attempt to apply Solve$_<$b or alternatively SOLVE* if a goal is of the form $(x < y)$, and if these two fail to be applicable, to move on to the preparations for ComplexEstimate by UNWRAPHYP.

The meta-predicate most-similar-subterm-in returns for each instantiation of "goal", most-similar-subterm-in returns instantiations of "ass" and "pos". It computes the formula s at position pos in an assumption ass such that s is most similar to the goal. The similarity is measured by function symbol occurrences and characterizes the suitability of an antecedent for ComplexEstimate to succeed. Inside the planning of UNWRAPHYP, Focus colors a certain subformula s of an assumption. The purpose of this focus is to concentrate the operator applications on extracting s.

Afterwards SOLVE* or alternatively ComplexEstimate – are tried to be applied. (This control-rule is omitted). SOLVE* is tried first because it produces simple or no subgoals.

```
(control-rule use-unwrapped
     (if (last-method UNWRAPHYP "ass""pos""goal"))
     (then
        (select((SOLVE* "goal")
                (ComplexEstimate "ass""goal")))))
```

4.3 Combining Constraint Solving with Proof Planning

In many proofs, mathematical objects with certain properties need to be constructed, for example, the δ in proofs of theorems about limits. These constructions can be supported by constraint solving (rather than by unification only) that we integrated into proof planning.

Our methods at the interface between the proof planning and constraint-handling component are Solve$_<$b, Solve=b, and Solve*$_<$b. Their purpose is to remove an equational or an inequality goal by adding it to the constraint store. Each of these methods is applicable only if passing the inequality to the constraint solver results in a consistent constraint store and in this case, the proved inequality can be derived from the final constraint store. The function tell accesses the constraint solver LINEQ and return $\perp$ if LINEQ reaches an inconsistent constraint store and *true* otherwise.

The constraint solver serves two main purposes: Firstly, it is used during the process of proof planning to determine whether a `Solve` method can be legally applied. Secondly, after the completion of the proof plan, the constraint store is condensed into an answer assertion about the values of variables that is included into a final proof and that justifies inequalities and equalities that follow from the final constraint store. See see [15] for details.

4.4 Experiments and Results

We illustrated the approach by presenting knowledge that is needed in order to plan all limit theorems. We succeeded in planning these theorems among them theorems that were beyond the problem solving horizon of other systems, e.g. LIM*. Other theorems we proved are, e.g., ContinuousComp that states that the composition of two continous functions is continuous, and ContinuousIfDeriv that states that a function having a derivative at a point is continuous there. The use of control-rules was essential for this success. The constraint solving component helped restricting the instantiations of existentially quantified variables. The results show that our approach is useful.

A comparison of the search spaces of proof planning and classical automated theorem proving, epitomized by OTTER [14], shows a qualitative progress: For a simpler version of LIM+ and with a particular strategy and parameters, OTTER generates a search space of 3650 clauses and keeps 1418 clauses.[3] However, this strategy does not work for LIM* or other limit theorems. In auto-mode OTTER generates up to 437706898 clauses and does not succeed in proving LIM+. In comparison, the search space of our proof planning for LIM+ is 350 nodes.[4] The control-rules are necessary to find a plan for LIM+. However, with the same methods and control-rules OMEGA's planner succeeds to prove all the limit theorems. For instance, the search space of planning LIM* has 470 nodes.

How can the work presented be assessed in terms of of the criteria for proof plans from Bundy's Science of Reasoning [7]?
Expectancy: The methods and control work for planning the limit theorems. *Generality*: All methods can be used for a larger class of proof plans. The `Solve` methods can even be generalized beyond the particular constraint solver for linear inequalities, e.g., in set theory another constraint solver has to be integrated. No fixed structure of limit plans can be established but common control-rules. *Prescriptiveness*: The expansions of methods does not produce search and the plans are not over-detailed. *Simplicity*: From the planning point of view, the specifications is simple. The knowledge engineering process for designing the methods was not simple though. *Efficiency*: The described proof planning avoids search and is efficient. *Parsimony*: We need several control-rules for the collection of all limit theorems. These are, however, not specific for one theorem.

The knowledge-based proof planning has several advantages over the procedurally encoded use of specific knowledge in special-purpose theorem provers, e.g., for set theory [12]: it employs a general-purpose problem solver, the plans are expandable to checkable calculus-level proofs, the control knowledge is modular and can therefore be subject to modification, and last but not least, the resulting plans can be communicated to the user.

A bottleneck of the knowledge-based proof planning is the knowledge acquisition. In mathematics the acquisition of methods and control knowledge is more difficult than in typical

[3] Specific input file of Bill McCune.

[4] This figure applies for a plan that includes the elimination of quantifiers. It can still be optimized.

AI-planning domains, e.g., the transportation domain or blocks world. Often the mathematical knowledge about methods and control in textbooks etc. is implicit only and it is difficult to make it explicit. This is one reason why mathematics is hard for many people.

5 Conclusion

We extended proof planning to cope with genuinely mathematical problems In particular, we extended the proof planner's architecture and the domain knowledge used for proof planning. The reduction of search was our main concern and this made it possible to succeed with the planning at all.

The fact that we have succeeded in automatically planning a proof of LIM* with our general purpose proof planner shows that these techniques can be successfully used for highly non-trivial mathematical theorems. While LIM+ represents the limit of what current theorem provers and planners could handle, LIM* is beyond the capabilities of other current proof planners and theorem provers.[5]

The idea behind our 'limit' enterprise, which we have chosen as a test-bed for ideas, has been to make proof planning a more widely applicable technique in automated theorem proving and to foster even more the application of AI-techniques within the theorem proving paradigm. The paper has shown some directions to follow in proof planning such as extending the domain knowledge by modularly represented control-rules and by constraint solving and encoding existing heuristics into planning operators.

Related Work
Proof planning in $CL^{A}M$ is most closely related to our work but there are many differences: For instance, as opposed to our hiding of axioms, $CL^{A}M$ with colored rippling [17] proved LIM+ by explicitly using a number of proof assumptions such as $|x+y| < w \Leftarrow |x| + |y| < w$ and $q \rightarrow p_1 \wedge p_2 \Leftarrow (q \rightarrow p_1) \wedge (q \rightarrow p_2)$, etc. As opposed to our domain-dependent control, $CL^{A}M$ uses the general difference reduction heuristic rippling. As far as we know, LIM* has not be proved by colored rippling.

Gerberding and Noltemeier [10] use control-rules for inductive proofs, however, without referring to their origins in AI-planning.

References

[1] R.G. Bartle and D.R. Sherbert. *Introduction to Real Analysis.* Wiley Inc., 2nd ed edition, 1992.

[2] C. Benzmueller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, A. Meier, E. Melis, W. Schaarschmidt, J. Siekmann, and V. Sorge. OMEGA: Towards a mathematical assistant. In W. McCune, editor, *Proceedings 14th International Conference on Automated Deduction (CADE-14)*, pages 252–255, Townsville, 1997. Springer.

[5] OTTER was able to prove only the easiest of Bledsoe's formulations of LIM+, and that required a specialized strategy. Personal communication with Bill McCune.

[3] W.W. Bledsoe. The use of analogy in automatic proof discovery. Tech.Rep. AI-158-86, Microelectronics and Computer Technology Corporation, Austin, TX, 1986.

[4] W.W. Bledsoe. Challenge problems in elementary analysis. *Journal of Automated Reasoning*, 6:341–359, 1990.

[5] W.W. Bledsoe, R.S. Boyer, and W.H. Henneman. Computer proofs of limit theorems. *Artificial Intelligence*, 3(1):27–60, 1972.

[6] A. Bundy. The use of explicit plans to guide inductive proofs. In E. Lusk and R. Overbeek, editors, *Proc. 9th International Conference on Automated Deduction (CADE-9)*, volume 310 of *Lecture Notes in Computer Science*, pages 111–120, Argonne, 1988. Springer.

[7] A. Bundy. A science of reasoning. In J-L. Lassez and G. Plotkin, editors, *Computational logic: Essays in Honor of Alan Robinson*, pages 178–198. MIT Press, 1991.

[8] A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324, 1991.

[9] 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.

[10] S. Gerberding and A. Noltemeier. Incremental proof-planning by meta-rules. In *Proceedings of the 10th Florida International AI Conference (FLAIRS-97)*, Dayton Beach, FL, 1997.

[11] M. Gordon, R. Milner, and C.P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation.* Lecture Notes in Computer Science 78. Springer, Berlin, 1979.

[12] L.M. Hines. Str+ve$\subseteq$: The stri+ve-based subset prover. In *Proceedings of the Tenth Conference on Automated Deduction (CADE)*, 1990.

[13] X. Huang, M. Kerber, M. Kohlhase, and J. Richts. Methods - the basic units for planning and verifying proofs. In *Proceedings of Jahrestagung für Künstliche Intelligenz KI-94*, Saarbrücken, 1994. Springer.

[14] W.W. McCune. Otter 2.0 users guide. Technical Report ANL-90/9, Argonne National Laboratory, Maths and CS Division, Argonne, Illinois, 1990.

[15] E. Melis. Progress in proof planning: Planning limit theorems automatically. Technical Report SR-97-08, Fachbereich Informatik, Universität des Saarlandes, October 1997. available from http://www.ags.uni-sb.de/publications/deduktion/seki/SR-97/index.html.

[16] S. Minton. Explanation-based learning: A problem solving perspective. *Artificial Intelligence*, 40:63–118, 1989.

[17] Y. Tetsuya, A. Bundy, I. Green, T. Walsh, and D. Basin. Coloured rippling: An extension of a theorem proving heuristic. In A.G. Cohn, editor, *Proceedings of 11th European Conference on Artificial Intelligence (ECAI-94)*, pages 85–89. John Wiley, 1994.

Towards a Science of Reasoning through Societies of Proof Planning Agents

Michael Fisher
Dept. of Computing & Mathematics
Manchester Metropolitan University
Manchester M1 5GD, U.K.
EMAIL: `M.Fisher@doc.mmu.ac.uk`

Andrew Ireland
Dept. of Computing & Electrical Engineering
Heriot-Watt University
Edinburgh, EH14 4AS, U.K.
EMAIL: `A.Ireland@hw.ac.uk`

Abstract

The value of having a proof plan comes from the structure it imposes upon the search space. We will argue that the benefits of this structure have yet to be fully exploited, in particular the opportunities it provides for parallelising search. As well as having a positive effective, the structure of a proof plan can also have a negative impact on the search for proofs. Currently the hierarchical structure of a proof plan is fixed during its design. This fixed structure significantly limits the flexibility of a proof plan in dealing with new applications and proof strategies. Here we propose an agent-based approach to proof planning which we believe will enable us to harness the potential parallelism while also increasing the flexibility of proof plans.

1 Introduction

Within his *Science of Reasoning* manifesto [5] Alan Bundy describes a methodology which complements formal logic by promoting a high-level understanding of reasoning. *Proof planning* [4] is a computational framework which aims to realize this manifesto within the context of mathematical reasoning. Here we present a proposal which further develops the capabilities of the current framework. Our proposal builds upon an *agent-based* view of proof planning and extends the framework in two distinct dimensions. Firstly, we will argue that our approach will give access to the opportunities for parallelism which are latent within the current framework. Secondly, we believe that the adoption of an agent-based approach will facilitate greater flexibility in how proof plans adapt to new applications and proof strategies.

2 Background: proof planning

A *proof plan* encapsulates meta-level properties which define a family of object-level proofs. Such meta-level properties provide search control information which is used within the activity of planning a proof. The success of planning a proof gives rise to a LCF style *tactic* [21] which must be executed in order to verify the soundness of the high-level proof planning steps. In terms of meta-level guidance, a proof plan has two complementary parts, a *method* and a *critic*. Both parts contain declarative *preconditions* and procedural *effects*. The preconditions of a method express the applicability of a corresponding object-level tactic with respect to a given goal. The effects of a method simulate the outcome of executing the tactic at the object-level and thus provide the link between atomic steps within the proof planning process. A critic [23] expresses patchable exceptions to the method. Unlike methods, a critic's preconditions have access to the partial instantiation of the proof plan during the planning process. The effects of a critic typically lead to the discovery of properties required in order to complete the proof or a modification to the existing proof plan instantiation. Proof planning is implemented within the CLAM[1] proof planner [6].

2.1 Successful Applications

Proof planning is most effective for applications where the:

- Object-level search space is very large, *i.e.* super-exponential or worse;
- Meta-level search space is well-structured.

Proof by mathematical induction is an example of where the technique has been shown to be very successful [8]. To expand upon this further, there exist three key choice points within the search space for a proof by mathematical induction, *i.e.*

- Induction rule synthesis: there exists an infinite number of induction rules so their dynamic creation is an important issue;
- Generalisation: paradoxically, it is sometimes necessary to generalise a conjecture in order for an inductive proof to succeed;
- Lemma discovery: trying to anticipate all the lemmata required in order to prove an inductive conjecture is not practical. The dynamic creation of missing lemmata is therefore desirable.

Each represents an infinite branching point within the search space. While the first is the subject of current research[2] significant success has been achieved with regard to the second and third [24, 25]. This success can be attributed in part to the structure which *rippling* [1, 8], a subcomponent of the overall proof plan for induction, imposes on the search space. Other notable successful applications within mathematical reasoning include finding closed form sums for series [35] and loop invariant discovery [26].

2.2 Opportunities for Improvements

There are still opportunities, however, for improving upon the current implementation of proof planning. The work on lemma discovery and generalisation mentioned above is search intensive. The structuring which the proof plan brings to the problem offers real opportunities for parallelising the search for such eureka steps. So far these opportunities have not been explored.

[1] The public version of CLAM does not contain proof critics. The critics version is experimental [24, 25, 26].

[2] Jeremy Gow is investigating the use of proof planning for the dynamic synthesis of induction rules as part of his PhD studies within the Artificial Intelligence department at Edinburgh University.

The way in which proof plans are currently structured can also have a negative impact on the search for proofs. That is, the overall structure or *architecture* of a proof plan is crafted by hand and remains fixed during the proof planning process. Having a fixed architecture means that certain areas of the search space will never be considered. The critics mechanism provides a degree of flexibility. It does not allow, however, for the re-configuration of the essential architecture of a proof plan. To illustrate, the subcomponents of the proof plan for induction [8] form the basis for a proof plan for loop invariant discovery [26]. While the sharing of proof methods and critics was relatively automatic the process of constructing the new overall proof plan was undertaken by hand. Our aim is to automate this process.

3 Motivations: why an agent-based approach?

An agent-based framework is only one of a number of possible approaches to achieving parallel search within proof planning. Agent-based frameworks, however, have been increasingly used to attack complex problem spaces through competitive and cooperative search [12, 22, 33]. We believe that competitive and cooperative search has a role to play within proof planning, in particular where planning involves the synthesis of unknowns, *e.g.* the synthesis of missing lemmata and induction rules. By viewing a proof as being distributed amongst *autonomous agents* [15], we may be able to combine the benefits of parallel search with novel agent-based search strategies.

Our desire for a more flexible approach to the architectural aspects of proof plans, as highlighted in §2.2, requires the sophistication which an agent-based approach provides. Our long term aim is to incorporate, within a variety of communicating autonomous agents, not only effective control and organisational strategies, but also AI techniques such as learning and analogy [11, 32]. This will lead to the possibility of developing *agent societies* [9] capable of supporting proof planning. Such societies would evolve, adapt and organise themselves in order to attack the particular problem under consideration. Similar approaches are being developed in areas such as distributed problem-solving [18] and we expect our structures to benefit from this. A key element of more flexible approaches is the move away from fixed communication links and agents towards the concept of groups of agents utilising broadcast communication, and providing open, dynamic and self organising systems.

4 Proof Planning Agents

The current implementation of proof planning [6], like many theorem provers, utilises a centralised control mechanism for managing the deductive process. By adopting an agent-based approach we take a *distributed* view of the deduction process where several parts of the search space may be being expanded at the same time. Furthermore, the agents will be able to exchange empirical data on the effectiveness of heuristics and organisational structures between themselves directly.

We see our agents as encapsulating both the method and critic components of the existing proof planning framework. However, unlike the current implementation [6], we associate *postconditions* to both methods[3] and critics. The significance of this will be explained in §4.2. In the simplest case agents will be organised into hierarchies. Collectively the agents

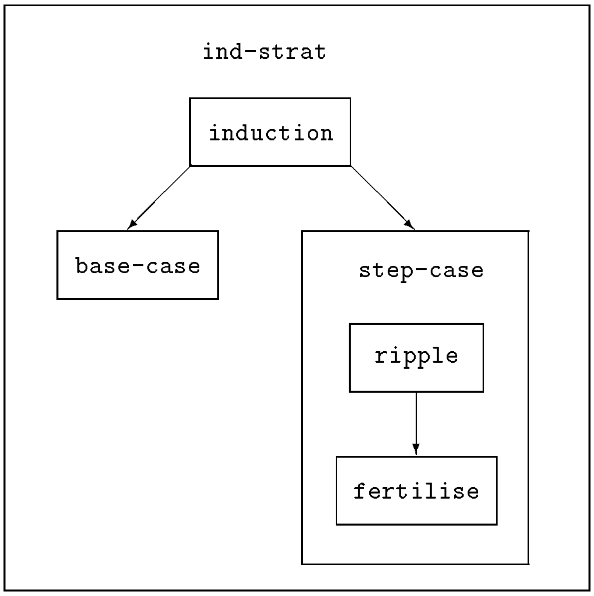
[3]Postconditions, rather than procedural effects, were part of the original proof planning proposal [4].

define a family of object-level proofs. The instantiation of a proof plan corresponds to a group of agents, each of which holds part of the tactic corresponding to an object-level proof. As well as patching failed proof attempts, the critic component of an agent will also contribute to the management of the parallelisation and re-configuration aspects of our proposal.

Agents will have two styles of communication. Firstly, within an established proof plan architecture, agents will use a *contract net* [34] like protocol for managing distributed tasks amongst a group of agents. Here, an agent can split its task into sub-tasks, can advertise for other known agents to undertake sub-tasks, and can then analyse and combine solutions it receives. Secondly, in order to support greater architectural flexibility we will also make provision for a more anonymous style of communication (see §4.2).

4.1 An Example of Parallel Proof Planning

We now consider how the proof planning agents outlined above can achieve effective parallelism. For the moment we consider the fixed architecture of an existing proof plan for mathematical induction as presented in figure 1. To illustrate the idea of agents collectively



The overall proof plan for mathematical induction, i.e. `ind-strat`, *consists of induction selection which is controlled by the* `induction` *method. The resulting subgoals are tackled by the* `base-case` *and* `step-case` *methods. Only one of each is shown here. Within the* `step-case` *the method* `ripple` *controls the rewriting of the induction conclusion so that the* `fertilise` *method can apply the induction hypotheses.*

Figure 1: A proof plan for mathematical induction

generating a successful proof attempt we consider the synthesis of a partitioning function based upon Dijkstra's Dutch national flag problem [13] which was discussed in the original proof critics proposal [23]. The example demonstrates how critics may be used to suggest a case analysis by recognising complementary failed partial proofs. A synthesis task may be expressed by the schema:

$$\forall inputs.\ \exists output.\ spec(inputs, output) \tag{1}$$

If we use a constructive logic [30] to prove a theorem of this form then the associated proof will yield a function, *prog*, which satisfies the logical relation $spec(inputs, prog(inputs))$. Proving synthesis theorems introduces the problem of calculating the value of existential witnesses. To be true to the synthesis paradigm we must incrementally construct the witnessing term hand–in–hand with the proof. Middle-out reasoning [7] provides one approach to achieving this objective. We begin by replacing *output* in (1) by the meta-term $F(inputs)$, our initial speculation for the existential witness, where F denotes a second-order meta-variable. This gives a new goal of the form:

$$\forall inputs.\ spec(inputs, F(inputs))$$

Agents will work on such a goal cooperatively and/or competitively in order to incrementally synthesize a definition of F.

Returning to our application, informally we require a program to partition a list of red and blue objects into a pair of lists containing the component colours. A program to achieve this task may be specified formally by the theorem:

$$\begin{array}{l} \forall l : list(objects).\ \exists x : list(objects) \times list(objects). \\ \quad perm(l, fst(x) <> snd(x)) \wedge reds(fst(x)) \wedge blues(snd(x)) \end{array}$$

Note that $<>$ denotes list concatenation while *fst* and *snd* are the projection functions for pairs. The predicate $perm(p, q)$ is true when p and q are lists such that p is a permutation of q while the predicates $reds(p)$ and $blues(p)$ hold when p is a list of *red* and *blue* objects, respectively. In addition we assume the following closure property for elements of type *objects*:

$$\forall x : objects.\ red(x) \vee blue(x) \tag{2}$$

Introducing a meta-term $F(l)$ for x the initial goal is reduced to:

$$perm(l, fst(F(l)) <> snd(F(l))) \wedge reds(fst(F(l))) \wedge blues(snd(F(l)))$$

Our agent for `ind-strat` will initiate an `induction` agent which will select structural induction on the list l. This has the effect of partially instantiating F to the primitive recursion schema which is the dual of structural induction on lists:

$$\begin{array}{l} F(l) \equiv \textbf{if}\ l = nil\ \textbf{then}\ G \\ \qquad\qquad\qquad\qquad \textbf{else let}\ u :: v = l\ \textbf{in}\ H(u, v, F(v)) \end{array}$$

The `base-case` agent instantiates G to be $\langle nil, nil \rangle$. We focus on the planning for the step-case where the induction hypothesis takes the form:

$$\begin{aligned} perm(v, fst(F(v)) <> snd(F(v))) \quad & \wedge \\ reds(fst(F(v))) \quad & \wedge \\ blues(snd(F(v))) & \end{aligned}$$

while the induction conclusion takes the form:

$$\begin{aligned} perm(u :: v, fst(F(u :: v)) <> snd(F(u :: v))) \quad & \wedge \\ reds(fst(F(u :: v))) \quad & \wedge \\ blues(snd(F(u :: v))) & \end{aligned} \tag{3}$$

Note that for reasons of brevity we only present the proposed effects of the proof agents on the goal structure. That is, we do not show the meta-level annotations used by the ripple heuristic [8] in guiding the proof search. The original more detailed presentation [23] contains the meta-level component.

Focusing on (3), the `step-case` agent initiates `ripple` and `fertilise` agents. The `ripple` agent refines the induction conclusion to give:

$$\begin{aligned} perm(u :: v, J(u, v, fst(F(v))) <> K(u, v, snd(F(v)))) \quad & \wedge \\ reds(J(u, v, fst(F(v)))) \quad & \wedge \\ blues(K(u, v, snd(F(v)))) & \end{aligned} \tag{4}$$

where H is instantiated to $\lambda x.\lambda y.\lambda z.\langle J(x, y, fst(z)), K(x, y, snd(z))\rangle$ as a side effect of the goal refinement. At this point in the proof the partial definition of the program takes the form:

$$\begin{aligned} F(l) \equiv \textbf{if } & l = nil \textbf{ then } \langle nil, nil\rangle \\ & \textbf{else let } u :: v = l \textbf{ in } \langle J(u, v, fst(F(v))), K(u, v, snd(F(v)))\rangle \end{aligned}$$

An OR-choice point now arises for the `ripple` agent. Both of the following rewrite rules are classified as being applicable according to the rippling heuristic:

$$perm(W :: X, W :: Y <> Z) \;\Rightarrow\; perm(X, Y <> Z) \tag{5}$$

$$perm(W :: X, Y <> W :: Z) \;\Rightarrow\; perm(X, Y <> Z) \tag{6}$$

However, only one may be applied to (4). In [23] both alternative ripple proofs were explored giving rise to two partial proofs. The `step-case` critic suggested a case split based upon the complementary nature of the partial proofs.

Here, we advocate that the `ripple` agent would communicate the OR-choice to its parent agent, *i.e.* the instance of the `step-case` agent. The critic component of the `step-case` agent would then make a decision as to whether or not parallel planning should be undertaken. Assuming that a parallelisation is selected then two `ripple` agents would be initiated, *i.e.* `ripple-1` and `ripple-2`, one for each alternative ripple.

Ripple-1 agent

Assuming `ripple-1` agent deals with the application of (5) then J is instantiated to $\lambda x.\lambda y.\lambda z.x :: z$ and the resulting ripple proof will enable the `fertilise` agent to instantiate K to be

$\lambda x.\lambda y.\lambda z.z$ leaving the subgoal $red(u)$. The definition of the program at this point takes the form:

$$
\begin{aligned}
F(l) \equiv \textbf{if } l = nil \textbf{ then } & \langle nil, nil\rangle \\
& \textbf{else let } u :: v = l \textbf{ in } \langle u :: fst(F(v)), snd(F(v))\rangle
\end{aligned}
$$

Note that this program will only produce correct results when given a list of red objects.

Ripple-2 agent

Assuming `ripple-2` agent deals with the application of (6) then the associated proof attempt again leads to a partial proof. This time the unproven residue is $blue(u)$ and the resulting program definition is:

$$
\begin{aligned}
F(l) \equiv \textbf{if } l = nil \textbf{ then } & \langle nil, nil\rangle \\
& \textbf{else let } u :: v = l \textbf{ in } \langle fst(F(v)), u :: snd(F(v))\rangle
\end{aligned}
$$

Note that this program will only produce correct results when given a list of blue objects.

Step-case agent

Both the `ripple-1` and `ripple-2` agents have achieved partial success. The idea is that the two agents communicate their partial success to the `step-case` agent. Taken together, the unproven subgoals suggest a case split based upon (2). The patched proof attempt gives rise to the following complete definition for the program:

$$
\begin{aligned}
F(l) \equiv \textbf{if } l = nil \textbf{ then } & \langle nil, nil\rangle \\
& \textbf{else} \quad \textbf{let } u :: v = l \textbf{ in} \\
& \qquad\qquad \textbf{case } colour(u) \textbf{ of} \\
& \qquad\qquad\qquad red : \langle u :: fst(F(v)), snd(F(v))\rangle \\
& \qquad\qquad\qquad blue : \langle fst(F(v)), u :: snd(F(v))\rangle
\end{aligned}
$$

where *colour* is the decision procedure associated with (2). This kind of retrospective discovery of case splits appears to be a common feature of synthesis proofs, for instance, it arises in the synthesis of insertion sort among other list manipulation functions.

4.2 Increasing the Flexibility of Proof Plans

The synthesis example presented in §4.1 exhibits a fairly standard form of distributed activity, taking place within a well defined organisational structure where agents know each other's identity. As mentioned earlier we wish agents to be able to dynamically re-configure their hierarchical relationships as and when the need arises. To achieve this we envisage building upon a *broadcast message passing* style of communication. When the current proof plan architecture fails then the analysis of the available methods may lead to a new and successful architecture. We see this analysis being achieved through the broadcasting of messages which contain unsolved goals as well as pre- and postconditions for methods and critics. These messages will form the basis of a process of meta-level inference in which agents will reason about their capabilities with respect to the current goal. This mode of communication will also allow agents, which encapsulate new proof strategies, to be introduced dynamically into an existing group of agents.

Broadcast mode will also be used to share empirical data on the success or failure of particular heuristics or organisational relationships. We believe that this empirical component will complement the formal meta-level reasoning component and will further enhance the flexibility and robustness of our proposed proof planning framework.

The richness of our extended framework will combine both cooperation and competition, together with agent grouping and the dynamic creation of agents [16]. Although grounded in the area of theorem proving, we propose to utilise results from both distributed AI and multi-agent systems research [9, 19]. We believe that from this framework will emerge simple *societies of proof planning agents* [15, 17]. The social activity of these agents might consist of:

- Evaluating the association of proof plans with proofs, a central component of Alan Bundy's Science of Reasoning methodology [5];
- Learning new proof plans from the existing methods and critics;
- Identifying new patterns of reasoning and potentially new methods and critics;
- Investigating analogies [32] and the re-use of proofs.

5 Conclusion

The programme of work outlined in this proposal will bring together research on novel approaches to distributed theorem proving [3, 15, 31], the development of societies of agents [9, 10, 19, 20], the modelling and evolution of cooperation [2, 14, 27, 29], and the use of AI techniques in theorem proving [11, 12, 28]. We see two key developments arising from our proposal:

- Firstly, it will harness the potential for parallel search which is latent within the current proof planning framework;
- Secondly, agent-based techniques will enable us to introduce a degree of flexibility which has been absent from the current proof planning framework.

Agent-based techniques will potentially play an ever increasing role in facilitating the exchange and analysis of scientific results across the Internet. Consequently we see our proposal for societies of proof planning agents as not only extending the role of computation within the emerging discipline of a science of reasoning [5], but also as accelerating its development through the Internet.

Acknowledgements

We are indebted to Alan Bundy, Julian Richardson, Alan Smaill, Jamie Stark and three anonymous referees[4] for feedback on an earlier version of this paper. We also thank Carole Douglas who played

[4]CADE-15: Workshop on *Using AI methods in Deduction.*

an invaluable role during the final preparation of the camera-ready copy. The ideas presented in this paper were supported by EPSRC grant GR/L11724.

References

[1] David Basin and Toby Walsh. A calculus for and termination of rippling. *Journal of Automated Reasoning*, 16(1–2):147–180, 1996.

[2] C. Beyssade, P. Enjalbert, and C. Lefèvre. Cooperating logical agents: Model, applications. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI 1037)*, pages 299–314. Springer-Verlag: Heidelberg, Germany, 1996.

[3] M. Bonacina and J. Hsiang. The Clause-Diffusion Methodology for Distributed Deduction. *Fundamenta Informaticae*, 24:177–207, 1995.

[4] A. Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *9th Conference on Automated Deduction*, pages 111–120. Springer-Verlag, 1988.

[5] A. Bundy. A science of reasoning. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 178–198. MIT Press, 1991.

[6] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam system. In M. E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 647–648. Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449. Also available from Edinburgh as DAI Research Paper 507.

[7] A. Bundy, A. Smaill, and J. Hesketh. Turning eureka steps into calculations in automatic program synthesis. In S.L.H. Clarke, editor, *Proceedings of UK IT 90*, pages 221–6, 1990. Also available from Edinburgh as DAI Research Paper 448.

[8] A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, 62:185–253, 1993.

[9] R. Conte and C. Castelfranchi. *Cognitive and Social Action*. UCL Press, London, 1995.

[10] J. Denzinger. Knowledge-based distributed search using teamwork. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 81–88, San Francisco, USA, June 1995.

[11] J. Denzinger, M. Fuchs, and M. Fuchs. High Performance ATP Methods by Combining Several AI Methods. In *Proceedings of IJCAI-97*, 1997.

[12] J. Denzinger and M. Kronenburg. Planning for Distributed Theorem-Proving: The Teamwork Approach. SEKI-Report SR-94-09, University of Kaiserslautern, Germany, 1994.

[13] E. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[14] E. H. Durfee, V. R. Lesser, and D. D. Corkill. Coherent cooperation among communicating problem solvers. *IEEE Transactions on Computers*, 36, 1987.

[15] M. Fisher. An Open Approach to Concurrent Theorem-Proving. In J. Geller, H. Kitano, and C. Suttner, editors, *Parallel Processing for Artificial Intelligence*, volume 3. Elsevier/North-Holland, 1997.

[16] M. Fisher and A. Kellett. Programming Dynamic Multi-Agent Systems. In *Proceedings of the UK Intelligent Agents Workshop*, Oxford, 1997. SGES Publications.

[17] M. Fisher and M. Wooldridge. A Logical Approach to the Representation of Societies of Agents. In N. Gilbert and R. Conte, editors, *Artificial Societies*. UCL Press, 1995.

[18] M. Fisher and M. Wooldridge. Distributed Problem Solving as Concurrent Theorem-Proving. In *Multi-Agent Rationality*. LNAI vol. 1237, Springer-Verlag, 1997.

[19] N. Gilbert and R. Conte, editors. *Artificial Societies.* UCL Press, 1995.

[20] N. S. Glance and B. A. Huberman. Organizational fluidity and sustainable cooperation. In C. Castelfranchi and J.-P. Müller, editors, *From Reaction to Cognition — Fifth European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW-93 (LNAI Volume 957)*, pages 89–108. Springer-Verlag: Heidelberg, Germany, 1995.

[21] M.J.C. Gordon, R. Milner, and C.P. Wadsworth. *Edinburgh LCF.* Volume 78 of *Lecture Notes in Computer Science*, Springer-Verlag LNCS 78, 1979.

[22] T. Hogg and C. Williams. Solving the Really Hard Problems with Cooperative Search. In *AAAI-93.* MIT Press, 1993.

[23] A. Ireland. The Use of Planning Critics in Mechanizing Inductive Proofs. In A. Voronkov, editor, *International Conference on Logic Programming and Automated Reasoning – LPAR 92, St. Petersburg*, Lecture Notes in Artificial Intelligence No. 624, pages 178–189. Springer-Verlag, 1992. Also available from Edinburgh as DAI Research Paper 592.

[24] A. Ireland and A. Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16(1):79–111, 1996.

[25] A. Ireland and A. Bundy. Extensions to a Generalization Critic for Inductive Proof. In M.A. McRobbie and J.K. Slaney, editors, *13th Conference on Automated Deduction*, pages 47–61. Springer-Verlag, 1996. Springer Lecture Notes in Artificial Intelligence No. 1104.

[26] A. Ireland and J. Stark. On the automatic discovery of loop invariants. In *Proceedings of the Fourth NASA Langley Formal Methods Workshop.* NASA Conference Publication 3356, 1997. Also available as Research Memo RM/97/1 from Dept of Computing and Electrical Engineering, Heriot-Watt University.

[27] A. Ito and H. Yano. The emergence of cooperation in a society of autonomous agents. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, San Francisco, CA, June 1995.

[28] M. Kerber, M. Kolhase, and V. Sorge. Integrating Computer Algebra with Proof Planning. In *Proceedings of DISCO-96.* Springer-Verlag: Heidelberg, Germany, 1996.

[29] A. Lux and D. Steiner. Understanding cooperation: an agent's perspective. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 261–268, San Francisco, CA, June 1995.

[30] P. Martin-Löf. Constructive mathematics and computer programming. In *6th International Congress for Logic, Methodology and Philosophy of Science*, pages 153–175, Hanover, August 1979. Published by North Holland, Amsterdam. 1982.

[31] D. MacIntosh, S. Conry, and R. Meyer. Distributed Automated Reasoning: Issues in Coordination, Cooperation, and Performance. *IEEE Transactions on Systems, Man and Cybernetics*, 21(6):1307–1316, November/December 1991.

[32] E. Melis and J. Whittle. Analogy as a Control Strategy in Theorem-Proving. In *Proceedings of Tenth Florida Artificial Intelligence Research Symposium (FLAIRS)*, May 1997.

[33] J. Schumann. SiCoTHEO – simple competitive parallel theorem-provers based on SETHEO. In J. Geller, H. Kitano, and C. Suttner, editors, *Parallel Processing for Artificial Intelligence*, volume 3. Elsevier/North-Holland, 1997.

[34] R. G. Smith. *A Framework for Distributed Problem Solving.* UMI Research Press, 1980.

[35] T. Walsh, A. Nunes, and Alan Bundy. The use of proof plans to sum series. In D. Kapur, editor, *11th Conference on Automated Deduction*, pages 325–339. Springer Verlag, 1992. Lecture Notes in Computer Science No. 607. Also available from Edinburgh as DAI Research Paper 563.

Processing Top-Down generated Subgoal Clauses in a Bottom-Up Prover

Dirk Fuchs
FB Informatik, Universität Kaiserslautern
67663 Kaiserslautern
Germany
E-mail: dfuchs@informatik.uni-kl.de

Abstract

Top-down and bottom-up theorem proving approaches have each specific advantages and disadvantages. Bottom-up provers profit from strong redundancy control and suffer from the lack of goal-orientation, whereas top-down provers are goal-oriented but have weak calculi when their proof lengths are considered. In order to integrate both approaches our method is to achieve cooperation between a top-down and a bottom-up prover: the top-down prover generates subgoal clauses, then they are processed by a bottom-up prover. We sketch theoretic aspects of this methodology and we introduce techniques for a relevancy-based filtering of generated subgoal clauses. Experiments with a model elimination and a superposition-based prover reveal the high potential of our approach.

1 Introduction

Automated deduction is—at its lowest level—a search problem that spans huge search spaces. In the past, many different calculi have hence been developed in order to cope with problems from the area of automated theorem proving. Essentially, for first-order theorem proving two main different paradigms for calculi are in use: *Top-down calculi* like *model elimination* (ME, see [Lov68]) attempt to recursively break down and transform a goal into subgoals that can finally be proven immediately with the axioms or with assumptions made during the proof. *Bottom-up calculi* like *superposition* (see [BG94]) go the other way by producing consequences from the initial clauses until the empty clause is derived.

When comparing results of various provers it is obvious that provers based on different paradigms often have quite a different behavior. There are problems where bottom-up theorem provers perform considerably well, but top-down provers perform poorly, and vice versa. The main reason for this is that many bottom-up provers suffer from the lack of goal-orientation of their search, but profit from their strong redundancy control mechanisms. In contrast, top-down provers profit from their goal-orientation and suffer from insufficient redundancy control thus entailing long proofs for many problems. Therefore, a topic that has come into focus of research is the integration of both approaches. In particular, co-operation between theorem provers based on top-down and bottom-up principles (see, e.g., [Sut92, Sch94, Fuc98c]) appears to be a promising way because by exchanging information each approach can profit from the other. Note that it is also possible to modify calculi or provers which work according to one paradigm so as to introduce aspects of the other paradigm into it. This, however, requires a lot of implementational effort to modify the provers, whereas our approach does not require changes of the provers but only changes of their input (see section 3). As a consequence, we can employ arbitrary state-of-the-art provers and are very flexible.

Information well-suited for top-down provers are lemmas deduced by bottom-up theorem provers. These lemmas are added to the input of a top-down prover and can help to shorten the proof length by immediately solving subgoals. Since the arising problem to filter relevant lemmas has already been discussed (see, e.g., [AS92, AL97, Fuc98c, Fuc98b]) we are not going to deal with this aspect here.

Instead, we want to consider top-down/bottom-up integration by transferring information from a top-down prover to a bottom-up prover. By transferring top-down generated *subgoal clauses* we are able to introduce a goal-oriented component into a bottom-up prover which enables it to solve proof problems considerably faster (see section 5). However, techniques for *filtering relevant subgoal clauses* must be developed because an unbounded transfer of subgoal clauses is not sensible. This is analogous the phenomenon that occurs when use of lemmas is unbounded. Note that the method proposed here is a possible solution to the problem to extract information suitable for cooperation from top-down theorem provers. This problem remained unsolved in [DF98] where a general framework for cooperation of different kinds of theorem provers was introduced.

In order to examine our kind of top-down/bottom-up integration we start by giving a brief overview about superposition and model elimination theorem proving in section 2. After that, in section 3 we introduce subgoal clauses and in section 4 we point out two variants of a relevancy-based filtering of subgoal clauses. An experimental study conducted with the theorem provers SETHEO [MIL+97] and SPASS [WGR96] reveals the potential of our techniques. Finally, a discussion and outlook at possible future work conclude the paper.

2 Automated Theorem Proving with Model Elimination and Superposition

The general problem in first-order theorem proving is to show the inconsistency of a given set $\mathcal{C}$ of clauses. Clauses are sets of literals. As already discussed, theorem provers utilize either top-down or bottom-up calculi to accomplish this task. In the following, we want to introduce the calculi that we employ for our theoretic and experimental study.

Typically, a bottom-up calculus contains several inference rules which can be applied to a set of clauses that constitute the search state. The superposition calculus (e.g., [BG94]) contains the inference rules superposition, equality resolution, and equality factoring.

Usually, a bottom-up theorem prover maintains a set $\mathcal{F}^P$ of so-called *potential* or *passive clauses* from which it selects and removes one clause C at a time. This clause is put into the set $\mathcal{F}^A$ of *activated clauses*. Activated clauses are, unlike potential clauses, allowed to produce new clauses via the application of some inference rules. The inferred new clauses are put into $\mathcal{F}^P$. Initially, $\mathcal{F}^A = \emptyset$ and $\mathcal{F}^P = \mathcal{C}$. The indeterministic selection or *activation step* is realized by heuristic means. To this end, a heuristic $\mathcal{H}$ associates a natural number $\varpi_C \in \mathbb{N}$ with each $C \in \mathcal{F}^P$, and the $C \in \mathcal{F}^P$ with the smallest weight ϖ_C is selected.

A typic top-down calculus is model elimination, essentially a restricted version of the *connection tableau calculus* (*CTC*) [LMG94]. This calculus works on *connected tableaux* for the initial clause set $\mathcal{C}$. A tableau for a set of clauses $\mathcal{C}$ is a tree whose non-root nodes are marked with literals and where it holds: if the immediate successor nodes $n_1, \ldots, n_m$ of a node n are marked with literals $l_1, \ldots, l_m$, then the clause $l_1 \vee \ldots \vee l_m$ (*tableau clause*) is an instance of a clause from $\mathcal{C}$. A tableau is called connected if it holds: if a non-leaf node n is marked with literal l then there is an immediate successor leaf-node n' marked with literal l' such that $l = \sim l'$. (Note that $\sim l = \neg l$, if l is positive, $\sim l = \tilde{l}$, if $l = \neg \tilde{l}$.)

The connection tableau calculus contains inference rules for transforming tableaux into others. The rules are *start*, *extension*, and *reduction*. The start rule can only be applied to the trivial tableau consisting of the unmarked root. It selects a clause from $\mathcal{C}$ and attaches its literals to the unmarked root. The step to select a clause from $\mathcal{C}$ and to attach its literals to an *open* leaf node (the branch leading to this node does not contain complementary literals) is also called *expansion*. Note that the start rule can be restricted to so-called *start relevant* clauses without causing incompleteness. Start relevancy of a clause is defined as follows. If $\mathcal{C}$ is an unsatisfiable set of clauses we call $S \in \mathcal{C}$ start relevant if there is a satisfiable subset $\mathcal{C}' \subset \mathcal{C}$ such that $\mathcal{C}' \cup \{S\}$ is unsatisfiable. Since the set of negative clauses contains at least one start relevant clause, in the following we also consider a restricted calculus which employs only negative clauses for the start expansion (CTC_{neg}). Tableau reduction closes a branch by unifying the literal which is the mark of an open leaf node (also called *subgoal*) with the complement of a literal on the same branch and applying the substitution to the whole tableau. Extension is performed by expanding a subgoal s and immediately performing a reduction step involving s and one of the newly introduced subgoals. We write $T \vdash T'$ if the tableau T' can be derived from T by the application of one inference rule. A *search tree* $\mathcal{T}^{\mathcal{C}}$ defined by a set of clauses $\mathcal{C}$ is given by a tree, whose root is labeled with the trivial tableau. Every inner node in $\mathcal{T}^{\mathcal{C}}$ labeled with tableau S has as immediate successors the maximal set of nodes $\{v_1, \ldots, v_n\}$, where v_i is labeled with S_i and $S \vdash S_i$, $1 \leq i \leq n$.

A theorem prover based on the connection tableau calculus tries to solve a proof problem $\mathcal{C}$ by deriving from the trivial tableau a closed tableau, i.e. a tableau with no open branch. A common way to perform derivations is to employ iterative deepening procedures ([Kor85]): in this approach iteratively larger segments of $\mathcal{T}^{\mathcal{C}}$ are explored in a depth-first manner. These segments are defined by a so-called *bound* (which poses structural restrictions on the tableaux which are allowed in the current segment) and a fixed natural number, a so-called *resource*. Iterative deepening is performed by starting with a basic resource value $n \in \mathbb{N}$ and iteratively increasing n until a closed tableau is found in the current segment. A prominent example for a completeness bound is the *inference bound* (see [Sti88]) which limits the number of inferences needed to infer a certain tableau. Note that the connection tableau calculus has no specific rules for handling equality. Hence, the common equality axioms must be added to the initial clause set when equality is involved in a problem.

3 Subgoal Clauses for Top-Down/Bottom-Up Integration

Because of the fact that connection tableau-based provers have a search state which contains deductions (tableaux) instead of clauses, it is at first sight not obvious how to extract valid clauses from such a search state which are well-suited for a superposition-based prover. A common method in order to extract valid clauses is to employ lemma mechanisms of ME provers: Assume that a literal s is a label of the root node of a closed sub-tableau T^s. Let $l_1, \ldots, l_n$ be the literals that are used in reduction steps and that are outside of T^s. Then, the clause $\sim s \vee \sim l_1 \vee \ldots \vee \sim l_n$ may be derived as a new lemma (since it is a logical consequence of the tableau clauses in T^s). Then, such a lemma could be transferred to a bottom-up prover.

As appealing as this idea sounds, it has some grave restrictions: Usually, such lemmas are—due to instantiations needed to close other branches before—not as general as they could be. Hence, they often cannot be used in inferences, especially not in *contracting inferences* (subsumption, rewriting) which are very important for bottom-up provers. Moreover, since these clauses are generated during the proof run in a rather unsystematic way they do not really introduce much goal-orientation and hence do not make use of the advantages of the search scheme typical for ME.

The concept of subgoal clauses, however, allows for the generation of clauses derived by inferences involving a proof goal: A *subgoal clause* S_T regarding a tableau T is the clause $S_T \equiv l_1 \vee \ldots \vee l_m$, where the literals l_i are the (open) subgoals of the tableau T. The *subgoal clause set* $\mathcal{S}^{\mathcal{B},n,\mathcal{C}}$ w.r.t. a bound $\mathcal{B}$, a resource n, and a clause set $\mathcal{C}$, is defined by $\mathcal{S}^{\mathcal{B},n,\mathcal{C}} = \cup S_T$, T is a tableau which is a label of a node of the initial segment of the search tree for $\mathcal{C}$ that is defined by bound $\mathcal{B}$ and resource n. Note that subgoal clauses are valid clauses, i.e. logical consequences of the initial clause set. In order to increase the goal-direction of our method it is wise to only consider subgoal clauses which are derived from "real" proof goals, i.e. which are derived from start relevant clauses. E.g., it might be sensible to restrict the set of subgoal clauses to descendants of the set of negative clauses, i.e. to consider only such subgoal clauses where the start expansion was performed with a negative clause. We call the set of these subgoal clauses $\mathcal{S}^{\mathcal{B},n,\mathcal{C}}_{neg}$.

Example 3.1 *Let* $\mathcal{C} = \{\neg g, \neg p_1 \vee \ldots \vee \neg p_n \vee g, \neg q_1 \vee \ldots \vee \neg q_m \vee g\}$. *Then,* $\neg p_1 \vee \ldots \vee \neg p_n$ *is the subgoal clause* S_T *belonging to the tableau obtained when extending the goal* $\neg g$ *with the clause* $\neg p_1 \vee \ldots \vee \neg p_n \vee g$*. If we employ* $\mathcal{B} =$ *inference bound* (Inf) *and resource* $k = 1$*, then* $\mathcal{S}^{\mathcal{B},k,\mathcal{C}} = \mathcal{S}^{\mathcal{B},k,\mathcal{C}}_{neg} = \{\neg p_1 \vee \ldots \vee \neg p_n, \neg q_1 \vee \ldots \vee \neg q_m\}$.

A subgoal clause S_T represents a transformation of an original goal clause (which is the start clause of the tableau T) into a new subgoal clause realized by the deduction which led to the tableau T. The set $\mathcal{S}^{Inf,k,\mathcal{C}}$ is the set of all possible goal transformations into subgoal clauses within k inferences if we consider *all input clauses* to be goal clauses, the set $\mathcal{S}^{Inf,k,\mathcal{C}}_{neg}$ is the set of all possible goal transformations into subgoal clauses within k inferences if we consider *only the negative clauses* to be goal clauses.

Now, in order to couple an ME and a superposition prover, we generate with the inference bound and a fixed resource $k > 0$ either the set $\mathcal{S}^{Inf,k,\mathcal{C}}$ or the set $\mathcal{S}^{Inf,k,\mathcal{C}}_{neg}$, depending on the fact whether CTC or CTC_{neg} is used. A superposition-based prover obtains then $\mathcal{C} \cup \mathcal{S}^{Inf,k,\mathcal{C}}$ or $\mathcal{C} \cup \mathcal{S}^{Inf,k,\mathcal{C}}_{neg}$ as input.

4 Relevancy-Based Selection of Subgoal Clauses

4.1 Relevancy criteria for subgoal clauses

As pointed out in [Fuc98a], in general the integration of subgoal clauses into the search state of a superposition-based prover need not lead to a reduction of the lengths of its proofs or a reduction of its proof search. Nevertheless, in practice it might be often the case that a re-structuring of the search caused by using subgoal clauses allows for finding proofs faster. The integration of subgoal clauses into the search state of a superposition-based prover promises a strong gain of efficiency in the following cases:

Firstly, it is important that some of the subgoal clauses can be proven quite easily, especially more easily than the original goal(s). In order to estimate this, it is necessary to judge whether they can probably be solved with the help of clauses of the initial clause set. E.g., measuring similarity between a goal and other clauses with the techniques developed in [DF94] may be well-suited.

Secondly, a solution of a newly introduced subgoal clause should not always entail a solution of an original goal within few steps of the prover. If this were the case then the integration of new subgoal clauses would not promise much gain. Criteria in order to estimate this are: on the one hand, the transformation of an original goal clause into a subgoal clause by a ME prover should have been performed by using many inferences, i.e. k should be quite high. Then, it is possible that a solution of a new subgoal clause does not entail a solution of an original goal within few steps because the probability is rather high that a bottom-up prover cannot—due to its heuristic search—reconstruct quickly the inferences needed to infer the original goal. On the other hand, if there is a subgoal clause S_T and some of the tableau clauses of the tableau T have a high heuristic weight regarding the heuristic of the superposition-based prover, a high gain of efficiency may occur if the prover can prove S_T. This is because inferences needed to infer the original goal may not be performed by the prover.

4.2 Relevancy-based selection

Already when using small resources k the sets $\mathcal{S}^{Inf,k,\mathcal{C}}$ or $\mathcal{S}_{neg}^{Inf,k,\mathcal{C}}$ can become quite large. Thus, it is not sensible to integrate all subgoal clauses from $\mathcal{S}^{Inf,k,\mathcal{C}}$ ($\mathcal{S}_{neg}^{Inf,k,\mathcal{C}}$) into the search state of a superposition-based prover: Integrating too many clauses usually does not entail a favorable rearrangement of the search because the heuristic "gets lost" in the huge number of clauses which can be derived from many subgoal clauses. Hence, it is reasonable to develop techniques for filtering subgoal clauses that entail a large gain of efficiency for a superposition prover if they can be proven, that is we are interested in filtering *relevant* subgoal clauses. Therefore, our approach is as follows: At first, we generate a *set of subgoal clause candidates* and then we select some subgoal clauses from this set. The chosen subgoal clauses are added to the search state of the bottom-up prover. We shall first introduce two techniques for generating a set of subgoal clause candidates. After that, we shall deal with the selection of relevant subgoal clauses.

In order to generate a set of interesting subgoal clauses it is important that we employ a large resource for generating subgoal clauses. As we have already discussed, subgoal clauses that are generated with a small number of inferences do not promise much gain because a bottom-up prover may easily reconstruct the inferences needed to infer them. Thus, if the prover is able to prove the subgoal clause it can also prove the original goal clause with few inferences and we do not gain much efficiency. However, it is not possible to generate all subgoal clauses $\mathcal{S}^{Inf,k,\mathcal{C}}$ ($\mathcal{S}_{neg}^{Inf,k,\mathcal{C}}$) for a sufficiently large resource k as subgoal clause candidates because their large number entails too high costs for the generation and additional selection. Hence, we are only able to choose as a set of subgoal clause candidates a subset of $\mathcal{S}^{Inf,k,\mathcal{C}}$ ($\mathcal{S}_{neg}^{Inf,k,\mathcal{C}}$), k sufficiently large (see section 5).

Our first variant, an *inference-based* method, starts by generating subgoal clauses from $\mathcal{S}^{Inf,k,\mathcal{C}}$ ($\mathcal{S}_{neg}^{Inf,k,\mathcal{C}}$) for a rather large resource k and stops when N_{sg} subgoal clause candidates are generated. The advantage of this method is that it is very easy and can efficiently be implemented: Tableaux are enumerated with a fixed strategy for selecting subgoals for inferences (usually left-most/depth-first) and for each tableau its subgoal clause is stored. The main disadvantage of this method is that due to the fixed strategy and the limit of the

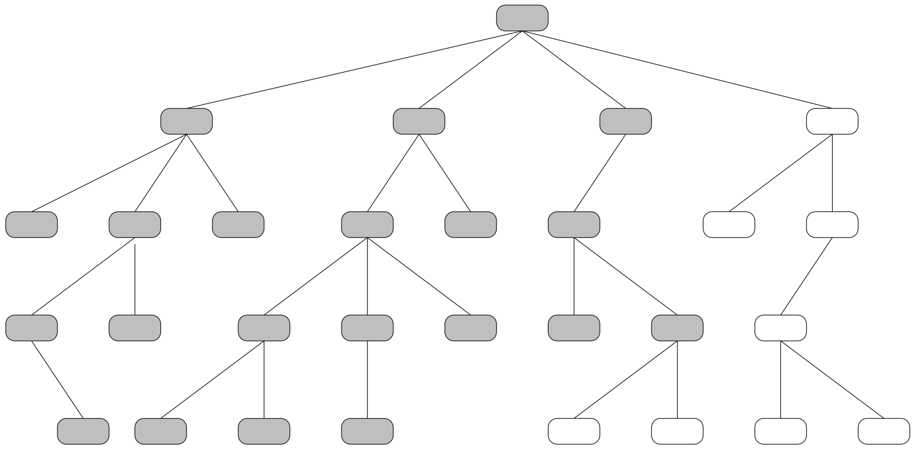


Figure 1: Inference-based generation of a set of subgoal clause candidates

number of subgoal clauses, we only obtain subgoal clauses which are inferred from goal clauses by expanding certain of their subgoals with a high number of inferences, other subgoals only with a small number of inferences. (See also Figure 1: Ovals are tableaux in a finite segment of the search tree $\mathcal{T}$, the lines represent the $\vdash$ relation. Grey ovals represent enumerated tableaux, i.e. their subgoal clauses are stored, white ovals represent tableaux which are not enumerated within N_{sg} inferences.) Thus, the method is somewhat unintelligent because no information about the quality of the transformation of an original goal clause into a subgoal clause is used. Certain transformations are favored only due to the uninformed subgoal selection strategy.

Our second variant, an *adaptive* method, tries to overcome the disadvantages of the first as follows. Instead of allowing for more inferences when generating subgoal clauses due to an uninformed subgoal selection strategy, we want to allow for more inferences at certain *interesting positions* of the search tree $\mathcal{T}^{\mathcal{C}}$ for a given set of clauses $\mathcal{C}$.

In detail, our approach is as follows: At first, we generate all subgoal clauses $\mathcal{S}^{Inf,k_1,\mathcal{C}}$ ($\mathcal{S}_{neg}^{Inf,k_1,\mathcal{C}}$) with a resource k_1 which is smaller compared to the first variant. Then, a fixed number N_{ref} of subgoal clauses is chosen which promise the highest gain of efficiency regarding the criteria from section 3. More exactly, we choose subgoal clauses which are maximal w.r.t. function ψ:

$$\begin{aligned}\psi(S_T) = \alpha_1 \cdot I(S_T) + \alpha_2 \cdot \max\{\varpi_C : C \text{ is a tableau clause in } T\} \\ + \alpha_3 \cdot \max\{sim(S_T, C) : C \in \mathcal{C}, |C| = 1\}\end{aligned}$$

The higher the number of inferences $I(S_T)$ which are needed to infer S_T the higher $\psi(S_T)$ should be. Hence, α_1 should be positive. Also a rating $\alpha_2 > 0$ is sensible. If there are tableau clauses C in T which have a high heuristic weight ϖ_C regarding the heuristic $\mathcal{H}$ of the superposition-based prover we can perhaps gain a lot of efficiency. The function *sim* measures whether literals from S_T could probably be solved with unit clauses from $\mathcal{C}$. We utilized the function *occnest* [DF94] for accomplishing this task. We set $\alpha_1 > \alpha_2 > \alpha_3 \geq 0$ due to the increasing vagueness of the criteria.

Now, let $M^{N_{ref}} \subseteq \mathcal{S}^{Inf,k_1,\mathcal{C}}$ ($M^{N_{ref}} \subseteq \mathcal{S}_{neg}^{Inf,k_1,\mathcal{C}}$) be the set of chosen subgoal clauses. Then, we generate again with a resource k_2 subgoal clauses but employ as start clauses for the tableau (subgoal clause) enumeration the clauses from $M^{N_{ref}}$. We call the set of subgoal clauses generated with this method $\mathcal{S}^{Inf,k_2,\mathcal{C},M^{N_{ref}}}$. (Consider also Figure 2: The dotted line

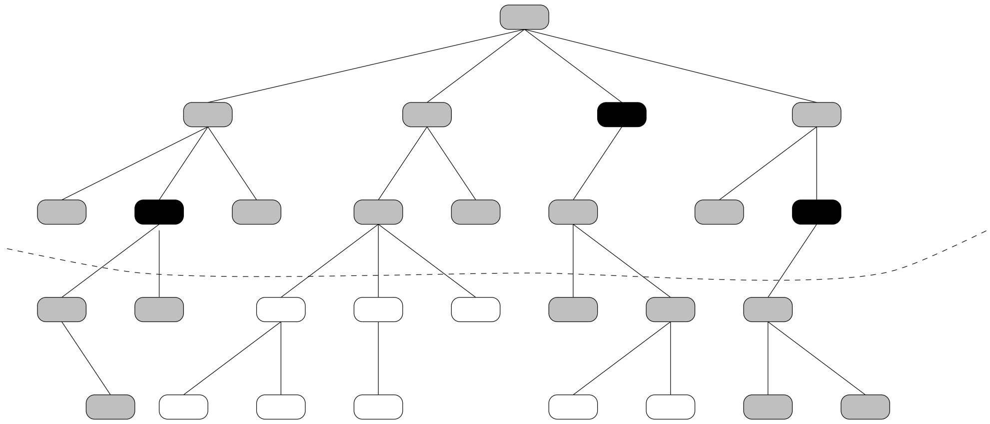


Figure 2: Adaptive generation of a set of subgoal clause candidates

shows which subgoal clauses are generated with resource k_1. Then some of them are selected (black ovals) and used as starting points for the generation of new subgoal clauses (grey ovals) with resource k_2.) The resource k_2 should again be not too high in order to allow for a fast enumeration of the subgoal clauses. The set of subgoal clause candidates is then given by $\mathcal{S}^{Inf,k_1,\mathcal{C}} \cup \mathcal{S}^{Inf,k_2,\mathcal{C},M^{N_{ref}}}$ or $\mathcal{S}_{neg}^{Inf,k_1,\mathcal{C}} \cup \mathcal{S}^{Inf,k_2,\mathcal{C},M^{N_{ref}}}$. Thus, subgoal clause candidates are on the one hand all subgoal clauses derived with a certain number of inferences which may lead to a reduction of the proof length. On the other hand, we have some subgoal clause candidates derived with a higher number of inferences, at most k_1+k_2. These subgoal clauses promise a high gain of efficiency because they are derived from subgoal clauses selected with function ψ.

For selecting subgoal clauses from the set of subgoal clause candidates we employed function φ, defined by $\varphi(S_T) = \psi(S_T) - \theta(S_T)$ and selected clauses with the highest weight. θ simply counts a weighted sum of the number of variables in S_T and two times the number of function or predicate symbols in S_T. Hence, "general" subgoal clauses are preferred because they can more often be used in inferences of a bottom-up prover.

5 Experimental Results

In order to conduct an experimental evaluation of our cooperation concept we coupled two renowned provers: the ME prover SETHEO and the superposition prover SPASS. Each prover runs on an own processor and obtains the initial clause set $\mathcal{C}$ as input. Because of the fact that when tackling simple problems it is unnecessary to let the provers cooperate, each prover tries to solve the problem independently with a timeout of 4 seconds. If no prover could solve the problem, the top-down prover generates subgoal clauses with one of the two variants. Note that SETHEO uses CTC_{neg}. Further, note that in our context the subgoal clause generation does not require changes in the top-down prover but can be performed with built-ins of the PROLOG-style input language of SETHEO. Then, these subgoal clauses are filtered, transferred to the bottom-up prover, and integrated into its search state. Finally, the provers proceed to tackle the problem in parallel.

Hence, we can efficiently solve simple problems. Moreover, cooperation can be performed for harder problems after the first timeout. Note that during this cooperation phase it is also possible to add some clauses that the bottom-up prover has generated to the axiomatization

Table 1: Integration of top-down/bottom-up approaches by cooperative provers

problem	SPASS	SETH.	inf.	adapt.	problem	SPASS	SETH.	inf.	adapt.
LDA004-1	–	–	–	–	CAT001-1	–	–	–	–
LDA005-1	–	–	–	–	CAT001-3	134s	32s	6s	6s
LDA005-2	279s	–	265s	8s	CAT001-4	33s	11s	5s	5s
LDA006-1	–	–	–	–	CAT002-2	–	–	–	–
LDA006-2	276s	–	304s	10s	CAT003-1	–	–	–	–
LDA007-1	16s	366s	19s	21s	CAT004-3	–	23s	–	9s
LDA007-2	–	50s	7s	7s	CAT008-1	91s	126s	6s	6s
LDA008-1	–	–	–	–	CAT009-1	–	–	10s	10s
LDA008-2	–	–	–	–	CAT009-3	–	–	–	29s
LDA009-1	–	–	–	–	CAT009-4	53s	–	47s	50s
LDA009-2	–	–	–	24s	CAT010-1	–	–	11s	9s
LDA010-1	–	–	–	9s	CAT011-3	17s	–	12s	12s
LDA010-2	–	–	–	26s	CAT014-3	18s	–	11s	11s
LDA011-1	54s	–	58s	9s	CAT018-3	–	–	–	74s
LDA011-2	21s	–	35s	7s	CAT019-4	–	–	–	–

of the ME prover. Thus, we can achieve cooperation by exchanging lemmas and subgoal clauses without one concept disturbing the other.

We experimented in the light of problems from the well-known problem library TPTP [SSY94]. In order to obtain a reliable collection of data, we employed two domains of TPTP as our test set, the domains CAT (category theory) and LDA (LD-algebras). The CAT domain consists of 58 problems, the LDA domain of 22. From these domains we extracted 22 and 15 non-trivial problems, respectively, i.e. problems none of the provers can solve within 4 seconds. Note that the problems in both domains contain equality. The subgoal clause candidates were generated in the following way: For variant 1 we employed the resource $k = 10$ which performed best in the experiments. The use of higher resources did not entail better results. We limited the set of subgoal clauses by $N_{sg} = 500$. For variant 2 we employed as resources $k_1 = k_2 = 9$. These resources allowed for the efficient generation of all subgoal clauses within the initial segments of the search tree. Usually at most 500 subgoal clauses were generated, i.e. about the same number as when employing variant 1. As start clauses for an adaptive refinement we selected $N_{ref} = 5$ clauses.

Finally, 100 clauses were selected from the set of candidates and transmitted to SPASS. Table 1 presents results of our experiments (we omitted all problems which could not be solved by any of the alternatives from table 1).

Column 1 of each part of the table displays the name of the problem. Columns 2 and 3 present the runtimes of SPASS and SETHEO (on a SPARCstation 20) when working alone. SPASS was used with its standard setting. SETHEO employed the weighted-depth bound ([MIL+97]) which performed best in the considered domains. Columns 4 and 5 display the runtimes of SPASS when it obtains subgoal clauses from SETHEO which are generated regarding variants 1 and 2, respectively. Note that the runtimes include the 4 seconds before the cooperation, the selection of subgoal clauses, and the transmission to SPASS. The entry "–" means that the problem could not be solved within 1000 seconds.

The results reveal the high potential of our approach to significantly improve on single provers. However, when considering the results of variant 1, they also show that a naive and uninformed generation of subgoal clauses usually does not entail much gain. In the LDA domain, we can solve 9 of 15 hard problems by using variant 2 for subgoal generation, whereas SPASS can only solve 5, SETHEO only 2. A simple competitive prover which employs SPASS and SETHEO in parallel would also only be able to solve 6 problems. Hence, cooperation is really important in order to increase the success rate. We can also in almost all cases significantly decrease the runtimes. E.g. problems `LDA005-2` and `LDA006-2` which require a runtime of more than 4 minutes when using SPASS can be proven in a few seconds. In the CAT domain the results are similar. By employing subgoal clauses generated regarding variant 2 we can solve 11 of 22 hard problems. SPASS and SETHEO are only able to solve 6 and 4, respectively. In the most cases the runtimes could be decreased.

6 Discussion and Future Work

Integration of top-down and bottom-up provers by employing cooperation is very promising in the field of automated deduction. Due to certain strengths and weaknesses of provers following different paradigms, techniques that try to combine the strengths by cooperation can allow for an improvement of the deductive system. Our approach of combining top-down and bottom-up provers by processing top-down generated subgoal clauses in a bottom-up prover achieves this combination by introducing goal-orientation into a bottom-up prover thus combining strong redundancy control mechanisms and goal-directed search.

So far, related approaches mainly aimed at supporting a top-down prover by a bottom-up based lemma component. Results presented, e.g., in [Sch94] or [Fuc98b], reveal that also these approaches are well-suited. However, in some domains, especially if equality is involved, superposition-based provers clearly outperform ME provers. Thus, in such domains it may be more sensible to develop techniques in order to support the more powerful bottom-up prover than the weaker top-down prover. Transmitting information from a top-down to a bottom-up prover was so far—to our knowledge—only discussed in [Sut92]. However, there lemmas generated by a ME prover were transferred to resolution-based provers and the results were not satisfactory.

Future work should deal with an extension of the empiric investigation. It would be interesting to detect in which kinds of domains (horn/non-horn, equality/no equality) the approach is especially well-suited. Moreover, it might be interesting to develop more complex methods for generating subgoal clause candidates. The results from section 5 suggest that an even more intelligent generation and selection of subgoal clauses leads to further improvements.

References

[AL97] O.L. Astrachan and D.W. Loveland. The use of Lemmas in the Model Elimination Procedure. *Journal of Automated Reasoning*, 19(1):117–141, 1997.

[AS92] O.L. Astrachan and M.E. Stickel. Caching and Lemmaizing in Model Elimination Theorem Provers. In *Proceedings of CADE-11*, pages 224–238, Saratoga Springs, USA, 1992. Springer LNAI 607.

[BG94] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[DF94] J. Denzinger and M. Fuchs. Goal oriented equational theorem proving. In *Proc. 18th KI-94*, pages 343–354, Saarbrücken, 1994. LNAI 861.

[DF98] J. Denzinger and D. Fuchs. Enhancing conventional search systems with multi-agent techniques: a case study. In *Proc. Int. Conf. on Multi Agent Systems (IC-MAS) 98*, Paris, France, 1998.

[Fuc98a] D. Fuchs. Cooperation between Top-Down and Bottom-Up Theorem Provers by Subgoal Clause Transfer. Technical Report SR-98-01, University of Kaiserslautern, Kaiserslautern, 1998.

[Fuc98b] M. Fuchs. Relevancy-Based Lemma Selection for Model Elimination using Lazy Tableaux Enumeration. In *Proc. ECAI-98*, pages 346–350, Brighton, UK, 1998. John Wiley and Sons Ltd.

[Fuc98c] M. Fuchs. Similarity-Based Lemma Generation for Model Elimination. In *Proc. CADE-15*, Lindau, Germany, 1998.

[Kor85] Richard E. Korf. Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. *AI*, 27:97 – 109, 1985. Elsevier Publishers B.V. (North-Holland).

[LMG94] R. Letz, K. Mayr, and C. Goller. Controlled Integration of the Cut Rule into Connection Tableau Calculi. *Journal of Automated Reasoning*, 13:297–337, 1994.

[Lov68] D.W. Loveland. Mechanical Theorem-Proving by Model Elimination. *Journal of the ACM*, 15(2), 1968.

[MIL+97] M. Moser, O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, and K. Mayr. The Model Elimination Provers SETHEO and E-SETHEO. *Journal of Automated Reasoning*, 18(2), 1997.

[Sch94] J. Schumann. Delta - a bottom-up preprocessor for top-down theorem provers. system abstract. In *Proceedings of CADE-12*. Springer, 1994.

[SSY94] G. Sutcliffe, C.B. Suttner, and T. Yemenis. The TPTP Problem Library. In *CADE-12*, pages 252–266, Nancy, 1994. LNAI 814.

[Sti88] M.E. Stickel. A prolog technology theorem prover: Implementation by an extended prolog compiler. *Journal of Automated Reasoning*, 4:353–380, 1988.

[Sut92] G. Sutcliffe. A heterogeneous parallel deduction system. In *Proc. FGCS'92 Workshop W3*, 1992.

[WGR96] C. Weidenbach, B. Gaede, and G. Rock. Spass & Flotter Version 0.42. In *Proc. CADE-13*, pages 141–145, New Brunswick, 1996. LNAI 1104.

Pre-Run Prover Strategy Optimization*

Andreas Wolf
Institut für Informatik der Technischen Universität München
wolfa@informatik.tu-muenchen.de

Abstract

Automated theorem provers use search strategies. Unfortunately, no strategy is uniformly successful on all problems. This motivates us to spend the available resources in terms of processors and time on different strategies. In this paper, we develop the basic concept of the complementarity of strategy sets. The problems of the initial strategy selection are discussed in detail. The paper also contains a short description of an implementation of a strategy parallel theorem prover (p-SETHEO) and an experimental evaluation of the schedule selection algorithm.

1 Introduction

A search problem is typically solved by applying an *uniform* search procedure. In automated deduction, different search strategies may have strongly different behaviors on a given problem. Unfortunately, in general, it cannot be decided in advance which strategy is the best for a given problem. This motivates us to use different strategies in a competitive manner, e.g. by parallelizing or time-slicing them. In order to be successful with such an approach, the strategies must satisfy the following two informally given conditions. First, the growth of the number of problems solved by a strategy should decrease with increasing time. Only assuming this condition it is useful to split the available time among different strategies. Second, the strategies should be successful on "significantly" different sets of problems. This ensures that the number of problems solved by a combination of strategies is not much less than the sum of the problems solved by each single strategy. These two vaguely formulated conditions will be reformulated exactly later. If both conditions are satisfied, then a competitive use of different strategies can be more successful than the best single strategy. A convincing illustration of this fact is the unexpected success of the theorem prover Gandalf[1] at the CADE-14 theorem prover competition [2].

In this paper, we investigate one problem of strategy parallelism in automated deduction. Section 2 begins formulating some basic notions. In Section 3, strategy parallelism is compared to other similar approaches. Section 4 explains in detail the strategy selection problem. Section 5 contains an algorithm which solves this problem. To study the possibilities of strategy parallelism in practice, we have implemented the experimental p-SETHEO framework. This implementation is explained in Section 6. Section 7 contains an experimental evaluation

*This work is supported by the Deutsche Forschungsgemeinschaft within the Sonderforschungsbereich 342, subproject A5 (PARIS).

[1] Gandalf time slices different strategies.

of the strategy selection algorithm. We conclude with summarizing the main results and list some future research perspectives in Section 8.

2 Basic Concepts

The selection of more than one search strategy in combination with techniques to partition the available resources (time and processors) with respect to a certain task defines a parallelization method which we call *strategy parallelism*. (Possibly distributed) competitive agents traverse the same search space, but in a different order. Whenever an agent finds a solution, all other agents are stopped. The intent of this method is to practically avoid repeated consideration of the same parts of the search space.

In order to simplify the presentation of our concepts, we assume the underlying (parallel) hardware to be fixed and having a uniform performance. Then a strategy can be defined as a mapping $s : F \longrightarrow \mathbb{N} \cup \{\infty\}$ where F is the finite set of all formulae or *problems*. For any problem f, the value $s(f)$ is the time[2] the strategy s needs to solve the problem on the given hardware.

Definition 1 *Let* $\{s_1, \ldots, s_n\}$ *be a set of strategies. A* schedule *is a set* S *of ordered pairs* $\langle s_i, t_i\rangle$ *(*$1 \leq i \leq n$*) with the* t_i *being non-negative natural numbers. The ordered pairs* $\langle s_i, t_i\rangle$ *are called* schedule components*. Let additionally* p_i *(*$1 \leq i \leq n$*) be non-negative integers. Then the ordered triples* $\langle s_i, t_i, p_i\rangle$ *(*$1 \leq i \leq n$*) are called* processor including schedule components*. The set* $S = \{\langle s_i, t_i, p_i\rangle : 1 \leq i \leq n\}$ *is called a* processor including schedule.

Note that some of the t_i in a schedule may be equal to zero. The schedule concept intends to describe an allocation of time and processor resources to each strategy in a certain considered strategy set. In the processor including schedule components the p_i denote processor ID numbers. In the following the notions of schedules and schedule elements are used for the processor including variants, too, if that can not cause confusion.

When trying to formalize the notion of complementarity of strategies, a certain difficulty is posed by problems that are *trivial*, i.e., problems that can be solved by almost any reasonable strategy in a short time. Typically, in automated deduction, there are always a number of such problems. To facilitate the exclusion of trivial problems, we define the solution set of a strategy with respect to a *trivial time* t_{min}.

Definition 2 *Given a set of problems* F*, a non-negative integer* t_{min}*, and a schedule component* $\langle s, t\rangle$*, the* solution set *sol of* $\langle s, t\rangle$ *with respect to* F *and* t_{min} *is* $sol_{F,t_{min}}(s, t) := \{f \in F : t_{min} < s(f) \leq t\}$.

Definition 3 *Given a set of problems* F*, a non-negative integer* t_{min}*, and a schedule* $S = \{\langle s_1, t_1\rangle, \ldots, \langle s_n, t_n\rangle\}$*,* S *is said to be* complementary with respect to F and t_{min} *if the solution sets of all schedule components in* S *are pairwise disjoint, i.e., if*

$$\bigcup_{i=1}^{n} \bigcup_{j=1}^{i-1} sol_{F,t_{min}}(s_i, t_i) \cap sol_{F,t_{min}}(s_j, t_j) = \emptyset.$$

Even if we exclude trivial problems from solution sets, in practice, a complementary schedule will be an exceptional case. Therefore we need a more quantitative notion of complementarity.

[2]Time values are given in natural numbers, reflecting seconds. In practice, this granularity turns out to be sufficient. Furthermore, finer time units will increase the time the schedule selection algorithm needs.

Definition 4 *Given a set of problems F, a non-negative integer t_{min}, and two schedule components $\langle s_1, t_1\rangle$ and $\langle s_2, t_2\rangle$. The* overlap value *of $\langle s_1, t_1\rangle$ and $\langle s_2, t_2\rangle$ is*
$ovl(\langle s_1, t_1\rangle, \langle s_2, t_2\rangle) = \dfrac{\mid sol_{F,t_{min}}(s_1, t_1) \cap sol_{F,t_{min}}(s_2, t_2) \mid}{\mid sol_{F,t_{min}}(s_1, t_1) \cup sol_{F,t_{min}}(s_2, t_2) \mid}$.

Definition 5 *Given a set of problems F, a non-negative real number t_{min}, and a schedule $S = \{\langle s_1, t_1\rangle, \ldots, \langle s_n, t_n\rangle\}$, the* overlap value *of S is defined as the arithmetic mean of the overlap values of all pairs of distinct schedule components:*
$ovl(S) := \frac{2}{n(n-1)} \sum_{i=1}^{n} \left(\sum_{j=1}^{i-1} ovl(\langle s_i, t_i\rangle, \langle s_j, t_j\rangle) \right)$.

Alternatively, one could use the geometric mean or the arithmetic mean of the squared overlap values. Note, however, that it is not reasonable to take the quotient of the cardinalities of $\bigcup\limits_{i=1}^{n} \bigcup\limits_{j=1}^{i-1} sol_{F,t_{min}}(s_i, t_i) \cap sol_{F,t_{min}}(s_j, t_j)$ and $\bigcup_{i=1}^{n} sol_{F,t_{min}}(s_i, t_i)$. The inadequacy of this measure can be seen from the following example. Consider two schedules with the same solution sets and in which every pair of different strategies overlaps on just one problem, in the first schedule always on the same problem and in the second schedule on a different problem for each pair. These schedules would have different overlap values. Intuitively, however, both schedules should have the same overlap value. This is guaranteed when using the concept in definition 5. The definitions given above can be canonically extended to processor including schedules.

Now we are able to state the two requirements for the success of strategy parallelism, informally given in the introduction, in a more precise form. At first, we discuss the sub-linearity. Given a problem set F and a strategy s, for each $\Delta \in \mathbb{N}$, let $g_\Delta : \mathbb{N} \longrightarrow \mathbb{N}$ be the function defined by $g_\Delta(n) := \mid sol_{F,n\Delta}(s, (n+1)\Delta) \mid$. Then we demand that there exists a $\Delta \in \mathbb{R}$ such that for all $n \in \mathbb{N} : g_\Delta(n) > g_\Delta(n+1)$.

Concerning complementarity, it is clear that one should strive for schedules with a low overlap value. However, normally, one attempts to develop strategies that are relatively successful as *single* strategies. Against this background, it is natural to include preferably such strategies in a schedule that are relatively successful when used separately. But the schedule composed this way may have a very high overlap value so that no real gain will be achieved with strategy parallelism. Consequently, it may be worthwhile not to develop single strategies separately, but entire sets of strategies to obtain a low overlap value. Although each of the elements of such a schedule may be inferior to any element of another schedule, their composition may be superior.

3 Related Work

The paradigm of strategy parallelism differs significantly from a *partitioning* of the search space [15] which is done, for instance, in PARTHEO [13]. Partitioning *guarantees* that no part of the search space is explored more than once. However, partitioning has the main disadvantage that completeness can be guaranteed only if *all agents* are reliable. In contrast, strategy parallelism retains completeness as long as *one agent* is reliable.

A combination of different strategies is used, e.g., within the teamwork concept [4] of DISCOUNT [5]. There a combination of several completion strategies is used for unit equality problems. These strategies periodically exchange intermediate results and work together evaluating these intermediate results and determining the further search strategies. Strategy selection with respect to features of problem to be solved was investigated here, too [7]. In this article methods of machine learning have been used.

In the clause diffusion concept of AQUARIUS [1], a resolution based prover with cooperating agents works on splitted databases. The cooperation of the distributed agents is necessary to guarantee completeness of the prover system.

Applying the nagging concept [14], dependent subtasks will be sent by a master process to the naggers, which try to solve them and report on their success. The results will be integrated into the main proof attempt.

In contrast to the other methods mentioned above, a strategy parallel prover takes into account all the available resources, as time and processors. The strategies are selected with respect to the prover results so far using learning techniques. This is an improvement with respect to *competition* [15]. Furthermore, strategy parallelism was developed with regard to technical requirements like scalability and completeness preservation even with not responding sub-provers.

4 Strategy Selection by Optimization

To make use of strategy parallelism a method to determine an optimal selection of strategies from a fixed set of given strategies is needed. A non specialist user of a theorem proving system could adapt the system to his specific domain of problems by selecting a representative *training* set of problems and computing an optimal set of competitive strategies S for the training set. If the training set is representative for the domain, then the computed strategies will also perform well on the whole domain. Concrete applications we envisage are interactive proof systems like ILF [3], Isabelle [11] or KIV [12], which use or intend to use automated theorem provers as subroutines.

This *strategy allocation problem* can informally be described as follows: given a set of (training) problems, a set of admissible strategies, a time limit, and a number of available processors, we want to determine an optimal distribution of resources to each strategy, i.e., a combination of strategies which can solve a maximal number of problems from the training set. In order to determine the worst-case complexity of this problem, we formulate it more precisely.

GIVEN (1) a set $F = \{f_1, \ldots, f_n\}$ of formulae (problems), (2) a set $S = \{s_1, \ldots, s_m\}$ of strategies, and (3) nonnegative integers t (time resource), p (processors), and k (the limit for the decision variant of the problem).

FIND a schedule $\{\langle s_1, t_1, p_1\rangle, \ldots, \langle s_m, t_m, p_m\rangle\}$ with $1 \leq p_i \leq p$ (strategy s_i will be scheduled for time t_i on processor p_i) such that (1) $\sum\limits_{\{i:p_i=j\}} t_i \leq t$ for all $j = 1, \ldots, p$, and

(2) $\mid \bigcup\limits_{i=1}^{m} \{f : s_i(f) \leq t_i\} \mid$ is maximal (optimization variant) or $\geq k$ (decision variant).

Obviously, the decision variant of the problem[3] is in NP: a given schedule can be verified in polynomial time.

Unfortunately, the decision variant of the problem for a single processor is *strongly NP-*

[3] Guess a schedule $\{\langle s_1, t_1, p_1\rangle, \ldots, \langle s_m, t_m, p_m\rangle\}$ such that the number of solved problems is greater than or equal to a given number k.

complete. Note that a problem is strongly NP-complete if it is NP-complete even if the numbers occurring in the input are written in unary notation. For example, the *knapsack problem* [8] which is obviously related with our problem is not strongly NP-complete unless P=NP. For the knapsack problem, there exist algorithms with *pseudo-polynomial* complexity, i.e., algorithms which are polynomial for a unary representation of the input numbers.

Clearly, for the optimization problem for the single processor case any formula can be ignored that is solved by every strategy in time $\lfloor \frac{t}{m} \rfloor$, since at least one of the strategies will be scheduled for time $\geq \lfloor \frac{t}{m} \rfloor$. Therefore it is natural to define the trivial time t_{min} as $\frac{t}{m}$. If a (one-processor) schedule $\{\langle s_1, t\rangle, \ldots, \langle s_m, t\rangle\}$ is complementary in the sense defined above, then our problem reduces to the knapsack problem. This motivates to strive for sets of strategies with low overlap values, for a given time limit.

For the general single processor case containing overlapping strategies no pseudo-polynomial algorithms are known. The strong NP-completeness of the problem can be shown by providing a polynomial reduction of the strongly NP-complete *minimum cover problem* [8] to our problem. Given a set $\hat{F}$, a set $\hat{S} = \{\hat{s}_1, \ldots, \hat{s}_m\}$ of subsets of $\hat{F}$, and an integer $\hat{t}$, find a subset $\hat{S}'$ of $\hat{S}$ with cardinality $\hat{t}$ such that $\bigcup \hat{S}' = \hat{F}$. We can reformulate this problem by restricting the single processor case of our strategy allocation problem as follows. Let $F = \hat{F} = \{f_1, \ldots, f_n\}$; define s_i by setting $s_i(f_j) := 1$ if $f_j \in \hat{s}_i$, and otherwise $s_i(f_j) := \infty$ $(1 \leq i \leq m, 1 \leq j \leq n)$. Furthermore, let $t := \hat{t}$ and $k := \mid F \mid$. Obviously, a tuple $\langle t_1, 1\rangle, \ldots, \langle t_m, 1\rangle$, solves the strategy allocation problem if and only if $\hat{S}' := \{\hat{s}_i : t_i = 1\}$ solves the minimum cover problem.

The full strategy allocation problem includes an additional placement of strategies on different processors as a subproblem, the *multiprocessor scheduling problem* [8]. The latter is also known to be strongly NP-complete, independently from the complementarity of the involved strategies. Therefore, in practice, one would give up finding optimal solutions for the full problem.

5 An Algorithm for the Determination of Strategy Schedules

We have seen that the determination of an optimal schedule is a strongly NP-complete problem, even if we use only one processor. Therefore it is not possible to solve the problem exactly in acceptable time for real sets of test problems and real sets of strategies. To reduce the complexity we introduce here an algorithm which combines randomized selected initial schedules and a modified gradient method of restricted dimension.

Given a set of strategies $S = \{s_1, \ldots, s_n\}$, a set of formulae $F = \{f_1, \ldots, f_m\}$, a time limit t, and a number of processors p. To perform the algorithm, the solution times $s_i(f_j)$ (which may be ∞) for all strategies $s_i \in S$ and all problems $f_i \in F$ have to be known.

(1) Assign arbitrary sets T_i with $\emptyset \neq T_i \subseteq S$ to each processor i $(1 \leq i \leq p)$.

(2) Assign arbitrarily selected initial time values t_i to each strategy s_i such that $\sum_{\{i: s_i \in T_j\}} t_i = t$ *for all* $j = 1, \ldots, p$.

(3) Treat the subsets T_i separately. The algorithm continues with one T_i.

(4) Determine the number L_0 of problems from F which can be solved by the so far generated schedule $SCHEDULE$. Let $STEP$ have the value 1.

(5) Determine the schedule with the most solved problems which can be generated from $SCHEDULE$ by decreasing the time value assigned to one strategy and increasing the time value of another strategy by $STEP$. If there is no such schedule, increase $STEP$ by 1 and repeat 5 as long as the increase/decrease by $STEP$ generates at least one correct defined schedule. Otherwise return the actually considered best schedule and exit. If there is a non-empty set Z of schedules with a maximal number of solutions higher than L_0, let $SCHEDULE$ be an arbitrary element of Z. Go to 4.

Obviously, in general, the algorithm will not determine the globally best schedule. But in some sense, the obtained schedules represent locally best ones. As it is usual in the field of algorithms with randomized elements, repeated execution increases the quality of the result. A Perl-like notation of step 5 of the algorithm considering the case of one processor follows next. The consideration of processor including schedules can be done easily in a similar manner.

```
@VECTOR = get_random_vector($STRAEGY_INDICES,$TIME);
$L_0 = get_number_of_solutions(@VECTOR);
$SUCCESS = 1;
while ($SUCCESS == 1) { @VECTOR = gradient(@VECTOR); }
return @VECTOR;

sub gradient {
  $L_1 = $L_0; @NEW_VECTOR = @VECTOR; $STEP = 0;
  $READY = 0; $SUCCESS = 0;
  while ($READY == 0 && $L_1 == $L_0) {
    $READY = 1; $STEP++;
    foreach $I (random_order($STRATEGY_INDICES)) {
      foreach $J (random_order($STRATEGY_INDICES)) {
        @TMP_VECTOR = @VECTOR; $TMP_VECTOR[$I] -= $STEP;
        if ($TMP_VECTOR[$I] >= 0) {
          $READY = 0; $TMP_VECTOR[$J] += $STEP;
          $L_2 = get_number_of_solutions(@TMP_VECTOR);
          if ($L_2 > $L_1) {
            $SUCCESS = 1; $L_1 = $L_2; @NEW_VECTOR = @TMP_VECTOR;
}}}}}}
```

One can see that the algorithm has the worst case complexity $O(t * m * n^3)$.

6 Implementation: p-SETHEO

To investigate the potential of strategy parallelism in practice, we have evaluated the method on different strategies of the sequential model elimination prover SETHEO [10]. We have implemented p-SETHEO, an environment for the strategy parallel execution of theorem provers. p-SETHEO can be configured very easily by an ASCII file containing information about the usable hosts, the maximum load allowed (the degree of time-slicing on a processor), and the strategy allocation for the competing strategies. Currently, all contained theorem provers are variants of SETHEO obtained by modifying the parameter settings. Because of the generic

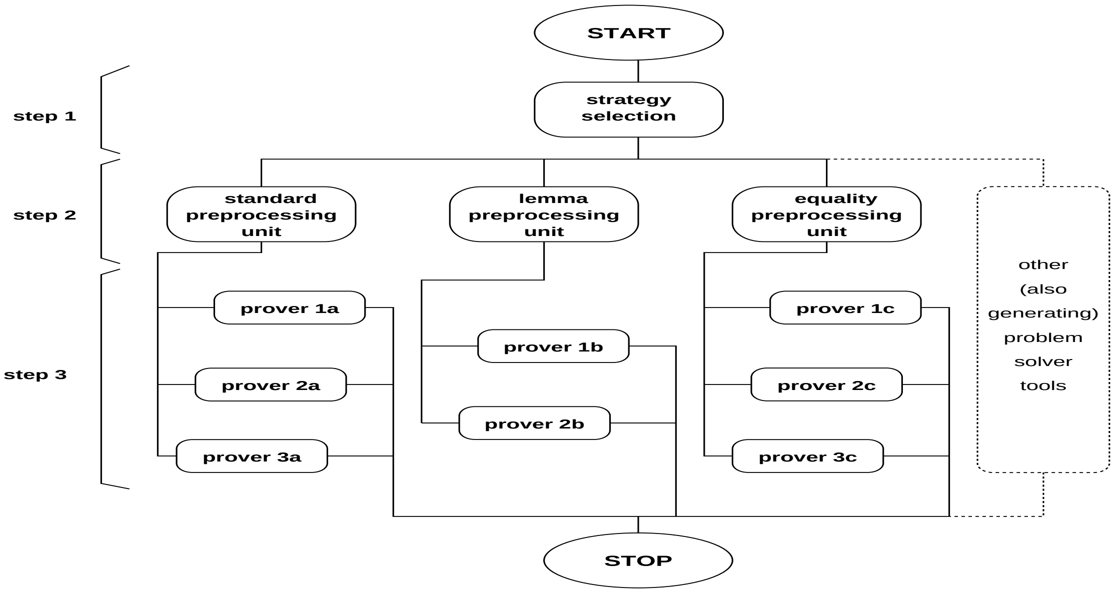


Figure 1: Parallelization model of p-SETHEO.

layout of the p-SETHEO controlling mechanism, new strategies and even new theorem provers can be integrated very easily. The implementation uses PERL, PVM [9], C, and shell tools in approximately 1000 lines of code (without SETHEO). p-SETHEO works as follows.

(1) According to the features of the problem to be solved and the usable resources in terms of processors, processor load, and time, a processor including schedule is selected.

(2) The preprocessing steps needed for all selected strategies are performed (reordering, equality treatment, etc.) using the available parallel resources (usable processors, maximum load per host).

(3) Subsequently, all prover strategies are started using the available parallel resources. The first prover that finds a proof stops all other processes.

Figure 1 shows the parallel flow of processes for a p-SETHEO configuration example.

7 Experiments

To evaluate the quality of the initial schedule selection algorithm, we tested it using p-SETHEO on a set X of 224 non trivial TPTP problems considering 8 strategies. These strategies stem from earlier experiments with single strategies and so they do not take into account the complementarity of strategies. The algorithm was used to determine optimal schedules for an arbitrarily selected test-set T of 40 problems (out of the 224 problems), for 200 seconds, and one processor. The application of the algorithm was repeated 100 times. For the resulting schedules, also the number of solved problems from X was determined. The results of this experiment are shown in the Figures 2 and 3.

It can be seen that a schedule with a large number of solved problems T performs probably well in X, too. Even the schedules which do not reach the maximum of solutions in T perform in most cases quite well on X, at least better than the best single strategy.

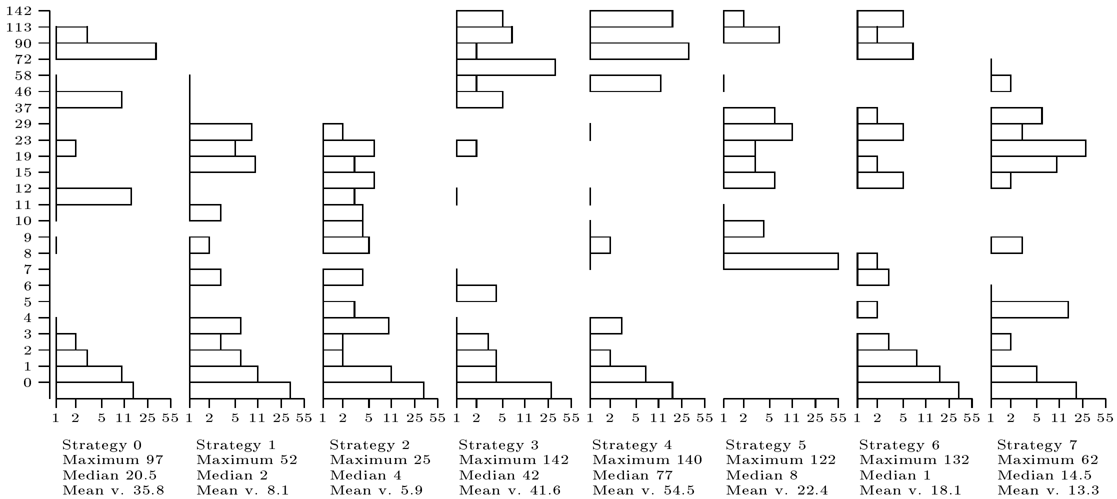


Figure 2: Statistics on times assigned by schedules to the different strategies. The diagrams show, how often certain time values (vertical) occur for the considered 8 strategies. Below each column the maximal and minimal time values as well as the median and the arithmetic mean are given.

The experiment described above was performed with many other sizes of test sets and many other time limits, too. From all these thousands of experiments we know that the probability to get a schedule with a maximal set of solved problems (in T) is 0.36. Assuming a binomial distribution 5 repetitions of the experiment increase that probability to 0.89. In Figure 4 the numbers of (totally) solved problems for *any* of the schedules[4] which solve a maximal number of problems on T is visualized for all times up to 1000 seconds. We see, that even for most other time values than that the schedule was optimized for, all these test set optimal schedules clearly outperform the best single strategy.

8 Conclusions and Future Work

In this paper, we considered the paradigm of strategy parallelism. We have developed fundamental notions like the overlap value of strategy sets. The worst-case complexity of the strategy allocation problem has been discussed and compared to other optimization problems. We have given an efficient method for the determination of a nearly optimal strategy schedule. Furthermore, the strategy parallel theorem prover p-SETHEO has been introduced which even on a single processor clearly outperforms the best single strategy of SETHEO. Our experiments demonstrate that one can significantly increase the performance of theorem provers even with a relatively simple strategy allocation algorithm and a non-optimized set of admissible strategies. In a certain subset of the TPTP the best single strategy solves 1233 problems in 274 seconds. p-SETHEO reaches this number of solved problems already in only 11 seconds using 5 processors. On single problems speed-up values up to 2000 have been observed, typical values range from 10 to 100.

While, in theorem proving, often the system developer or advanced user can tune (the pa-

[4]The resources have been scaled proportionally to fit the time limit.

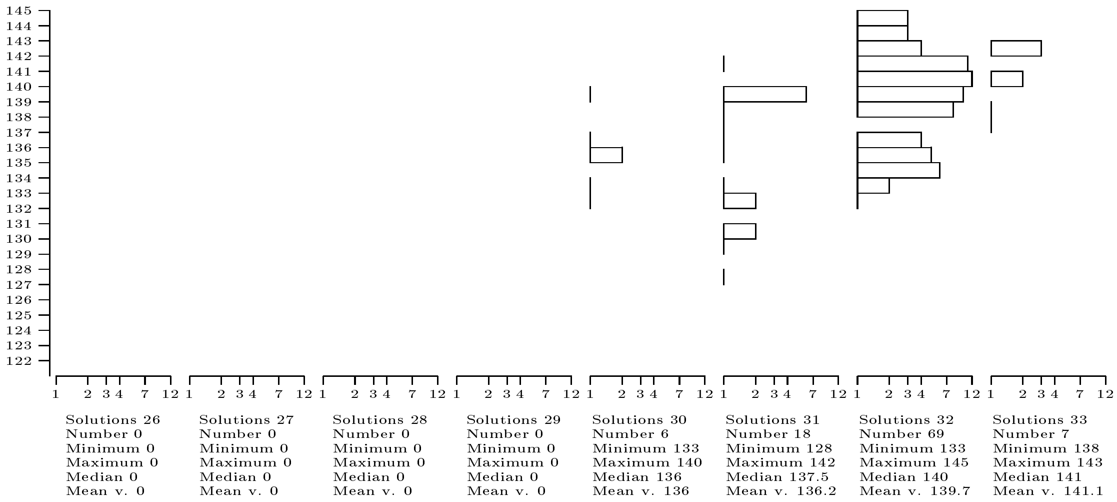


Figure 3: Comparison of the numbers of solved problems on T and on X. The diagrams compare, how often certain solution numbers (vertical) for X occur for the solution numbers in T. The subscripts of the columns contain the maximal and minimal numbers as well as the median and the arithmetic mean. The best single strategy solves 121 problems from X.

rameters of) his system to a given problem by using his experience, this is not possible if the theorem prover is integrated into a larger proof environment like ILF [3]. In this case the configuration *must* be done automatically, and strategy parallelism may be applied to solve this problem. p-SETHEO is used via ILF at other locations, and in a software reuse application at TU Braunschweig [6]. A further advantage of p-SETHEO and strategy parallelism, in general, is that such a system can easily be upgraded to use always the latest versions of the integrated provers.

Often a strategy is particularly successful for a certain class of problems. For example, unit equality problems need different treatment than problems without equality. If such features can be identified, the selection of strategies can be made more specific and hence more successful. Features that turned out to be particularly distinctive are the following: does a formula contain the theory of equality, does it contain non-Horn clauses, what is the size of the formula, what is the size of the signature? Normally, such information is available for a given theorem proving search paradigm and can then be applied by dividing the set of formulae into different classes. Partitioning the set of test problems into several classes may increase the quality of the determined schedules, but at least it increases the practicability of the schedule determination algorithm.

As further interesting research topics we envisage the following. The success of the selected strategies depends on the given training set. How do we obtain a training set which is representative for the considered domain of problems? Furthermore, the number of sensible strategies which are successful and have a low overlap value seems to be bounded. This restricts the scalability of strategy parallelism to large platforms. Can this problem be overcome, i.e., can we find a systematic method for producing as many successful and low overlapping strategies as we want? Probably such a method needs randomized elements.

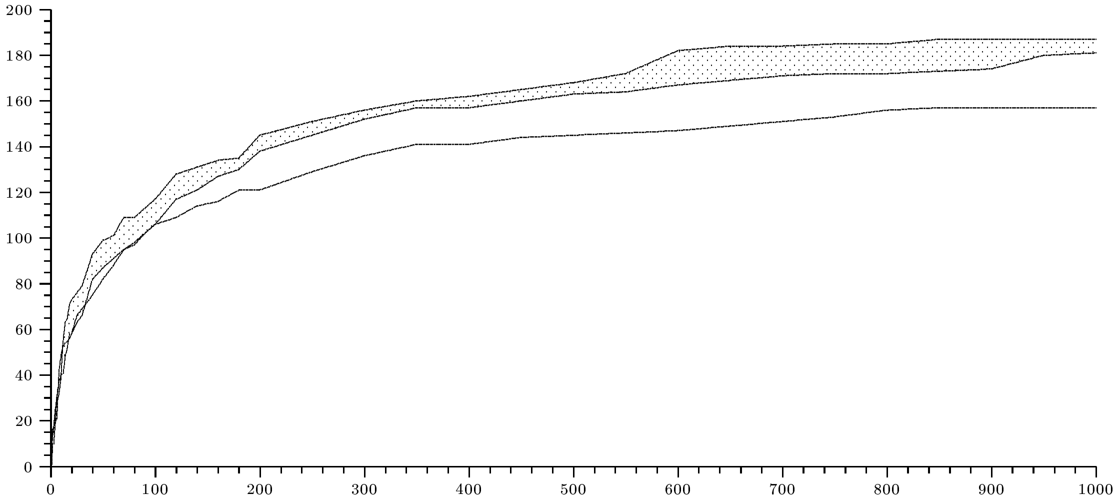


Figure 4: The horizontal scale gives the cumulated run-time of all scheduled strategies. The vertical scale denotes the total number of the solved problems (in the full set). The dotted area shows the performance range of the test set optimal schedules. The single curve denotes the performance of the best one of the considered single strategies.

References

[1] M. P. Bonacina, J. Hsiang. Distributed Deduction by Clause Diffusion: The Aquarius Prover. *DISCO-93*, vol. 722 of *LNCS*, pp. 272–287. Springer, 1993.

[2] CASC-14 Homepage. http://www.cs.jcu.edu.au/~tptp/casc-14/, 1997.

[3] B. I. Dahn, J. Gehne, T. Honigmann, A. Wolf. Integration of Automated and Interactive Theorem Proving in ILF. *CADE-14*, vol. 1249 of *LNAI*, pp. 57–60. Springer, 1997.

[4] J. Denzinger. Knowledge-Based Distributed Search Using Teamwork. *ICMAS-95*, pp. 81–88. AAAI-Press, 1995.

[5] J. Denzinger, M. Kronenburg, S. Schulz. DISCOUNT. A Distributed and Learning Equational Prover. *JAR*, 18(2):189–198, 1997.

[6] B. Fischer, J. Schumann. SETHEO Goes Software Engineering: Application of ATP to Software Reuse. *CADE-14*, vol. 1249 of *LNAI*, pp. 65–68. Springer, 1997.

[7] Matthias Fuchs. Automatic Selection of Search-Guiding Heuristics. *FLAIRS-10*, pp. 1–5. 1997.

[8] M. R. Garey, D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[9] A. Geist et al. *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*. 1994.

[10] M. Moser et al. SETHEO and E-SETHEO. The CADE-13 Systems. *JAR*, 18(2):237–246, 1997.

[11] L. C. Paulson. *Isabelle: A Generic Theorem Prover*, vol. 828 of *LNCS*. Springer, 1994.

[12] W. Reif. The KIV System: Systematic Construction of Verified Software. *CADE-11*, vol. 607 of *LNAI*, pp. 753–757. Springer, 1992.

[13] J. Schumann, R. Letz. PARTHEO: a High Performance Parallel Theorem Prover. *CADE-10*, vol. 449 of *LNAI*, pp. 44–56. Springer, 1990.

[14] D. B. Sturgill, A. M. Segre. A Novel Asynchronous Parallelism Scheme for First-Order Logic. *CADE-12*, vol. 814 of *LNAI*, pp. 484–498. Springer, 1994.

[15] C. Suttner, J. Schumann. Parallel Automated Theorem Proving. *PPAI*, pp. 209–257. Elsevier, 1994.

Term Space Mapping for DISCOUNT

Stephan Schulz

Institut für Informatik
Technische Universität München
Arcisstrasse 20, D-80290 München, Germany
schulz@informatik.tu-muenchen.de

June 16, 1998

Abstract

We present a learning inference control heuristic implemented for the equational theorem prover DISCOUNT. Information about necessary and superfluous facts generated during successful proof searches (*positive* and *negative* examples) is compiled into a *term space map*, a recursive structure describing aspects of the topology of term sets. This map is then used to compute evaluations for facts in new proof searches and thus to guide the proof search. We present results of the learning system on a set of TPTP examples and demonstrate its usefulness.

1 Introduction

The last years have seen an increase in the use of formal methods for research and development in various areas. Formal methods are even beginning to make inroads into industrial use. In particular, theorem provers like DISCOUNT [DKS97] and SETHEO [LSBB92] are being used for the verification of protocols [Sch97] and the retrieval of software components [FS97] and mathematical theorems [DW97] from libraries. They are also being integrated into interactive systems for the support of mathematicians [DGHW97].

A major requirement for the application of theorem provers is the performance of proof systems in terms of proofs found within a practical time limit. While new calculi and refined implementations can improve the performance (as shown by the success of SPASS [WGR96] and Waldmeister [HBF96]), the proper use of search control knowledge offers at least the same potential for improvements.

Search control knowledge in most current theorem provers is represented by various, often parameterized heuristics for the selection of certain inference steps. Optimization of the theorem prover for a given domain consists in the selection of an existing heuristic (with suitable parameters), or even in the manual coding of a new heuristic based on the experience of a user with the domain. Both tasks are tedious, and expensive in terms of time and manpower[1]. Our aim is to adapt a theorem prover to a domain by capturing information

[1] Attempts to automate these tasks at least partially are described in [Fuc95b, Fuc97].

about good search decisions from recordings of successful proof searches in *term space maps* and to use this information to create good search strategies for new problems.

In this paper we first give a short overview of DISCOUNT, the theorem prover used for our implementation. We then describe the general set-up of the learning theorem prover and the selection of training examples for the acquisition of search control knowledge. Section 3.3 describes learning with and application of term space maps. Experimental results on the TPTP are presented next, and we conclude the paper with a short evaluation and some remarks on future work.

2 The DISCOUNT System

DISCOUNT is a purely equational theorem prover, based on *unfailing completion* [HR87, BDP89]. It uses ordering-restricted rewriting and unit-paramodulation (*critical pair building*) as its main inferences. DISCOUNT works on a set of equations (derived from the axioms) and an equational goal (the hypothesis). The set of equations is split into two subsets, one consisting of *processed equations* (between which direct consequences have already been calculated) and unprocessed equations. The basic algorithm repeatedly selects an unprocessed equation, rewrites it to normal form w.r.t. the already processed equations, and uses it to build critical pairs (new unprocessed equations). The equation is then used for interreduction (backward subsumption) and added to the set of processed equations. The process stops when either the goal has been proven or all equations have been processed[2].

The selection of the next equation to process is the most crucial decision for the efficiency of the proof procedure. It is controlled by one of multiple heuristics. All of DISCOUNT's heuristics are realized as *weight functions*, i.e. functions that assign a numerical weight to an equation. Equations with a low weight (a *good evaluation*) are processed before equations with a higher weight.

Two of these heuristics (which are used for comparison later on) are *AddWeight*, which uses (weighted) symbol counting, and *Occnest* [DF94], a goal-oriented heuristic which prefers equations structurally similar to the goal. *AddWeight* is one of the strongest general purpose heuristics implemented in DISCOUNT. *Occnest* is not quite as successful in finding proofs, but performs better in certain domains and usually is much faster if it succeeds.

One of the strengths of the DISCOUNT system is its ability to record proof searches in the PCL format [DS94b, DS96b]. A PCL protocol contains all inferences performed by the prover during a proof search, as well as the facts generated by these inferences. A variety of programs use these protocols for proof transformation, analysis, and presentation. PCL protocols also serve as the basis for a number of learning approaches, including the one presented in this paper. The basic setup of the DISCOUNT environment (for the sequential case) is shown in figure 1.

DISCOUNT was implemented in 1992 as a prototypical system for the evaluation of the teamwork method [Den93, Den95]. Its inference engine was fully adequate for this task, however, in raw throughput, it cannot compete with current provers as SPASS or Waldmeister. We found, however, that it gave us an excellent framework for adding and evaluating a variety of AI methods. Due to these improvements, DISCOUNT remains a viable and competitive

[2]For this work we ignore the ability of DISCOUNT to distribute deduction processes with team work, and we also do not make use of the ability to deal with non-ground goals.

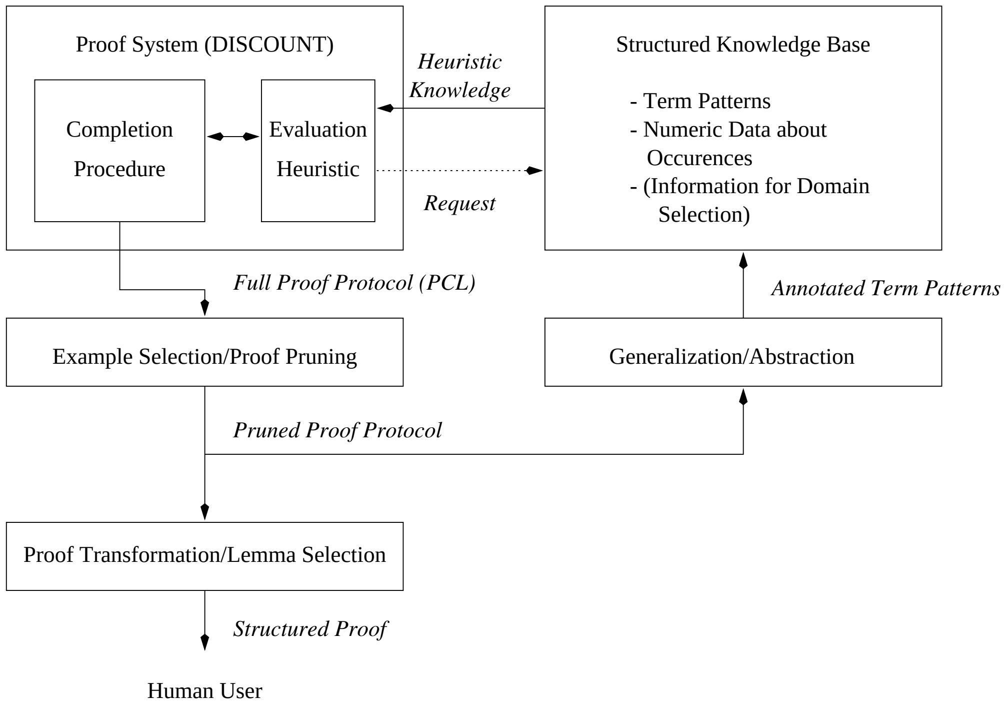


Figure 1: The DISCOUNT System

theorem prover despite its (by today's standards) rather weak inference engine.

3 Acquisition of Search Control Knowledge

Learning search control knowledge for theorem provers is based on the hypothesis that experience from previous proof searches is useful in guiding new proof searches. Given this hypothesis, the basic questions are which parts of a proof search should be used in learning, what kind of knowledge should be learned, and what learning algorithm should be employed.

Our approach to learning for DISCOUNT tries to extract search control knowledge from listings of inference steps, in particular from the facts (equations) generated during this proof. We try to learn *heuristic evaluation functions* for equations, using a new learning algorithm, *term space mapping*, for the detection of certain features of classes of terms. Our basic assumption is that the equations occurring in a successful proof search contain enough information to describe the proof adequately for reproduction, and that information about the exact structure of the proof is less important. This assumption is supported by the success of *learning by pattern memorization* [DS96a] particularly in reproducing proofs. We believe that the most important reason for this effect is that much of the relevant structure of the proof is given implicitly by the calculus (inferences can only be performed after all the necessary preconditions are fulfilled).

Our learning algorithm is structured into three phases (again, compare figure 1). First, a proof protocol of a successful proof search is analyzed, and positive and negative examples are selected from the equations generated in a proof search. In the second phase, the selected equations are generalized and stored in a knowledge base, annotated with some data about their role in the proof. Finally, the generalized equations from multiple proofs are compiled into a term space map, which is used in the main theorem prover to evaluate facts during the search for further proofs.

Our approach is a continuation of the work on *term evaluation trees* as presented in [DS96a]. Related work has been presented in [Fuc95b, Fuc95a, Fuc96] (where heuristical evaluation functions for DISCOUNT are computed based on facts from a single analogous proof) and [SE90, Gol94, Gol97], where evaluation functions for states of the model elimination theorem prover SETHEO are learned using a variety of neural network approaches. First experiments for learning fact evaluation functions for DISCOUNT with the powerful *folding architecture networks* [GK96, KG96] have been presented in [SKG97].

3.1 Example Selection

During an average proof search, a very large number of facts is generated, most of which do not contribute to the final proof. In typical non-trivial examples, between several 10000 and several 100000 inference steps are performed, only around 100-200 of which are usually needed for the proof (compare [DS94a] for a detailed discussion and [DS94b, DS96b] for some examples). This makes it impractical to use all these facts as training examples. However, it also turns out to be unnecessary. If we consider any given proof, it is obvious that the subset of facts used in the proof already describe a strategy for reproducing this proof ("select only inferences creating one of the facts used in the proof"). Similarly, the set of all unused direct descendants of used facts also describes such a strategy ("avoid all inferences creating an unused fact").

We therefore use only a subset of equations *close* to the actual proof. The complete protocol of all inferences in a proof search is viewed as a *directed graph*, were the nodes are labeled by the equations generated during the proof and edges go from facts used in the premise of an inference to the new equation generated by it. This graph is first pruned by removing all equations immediately proven redundant[3]. In the remaining graph, equations contributing to the proof (those on a *proof path* from the axioms to the final inference proving the goal) are considered positive examples, while facts up to a certain number of edges removed from any proof path are used as negative examples. All other facts are discarded.

3.2 Annotated Term Patterns

In order to abstract from the particular signature used for a given example without resolving to a potentially expensive signature match, the equations are stored as generalized *representative patterns*. A representative pattern for a term t is computed by normalizing the variables in the term and substituting its function symbols in the way that the ith original function symbol of arity j occurring in t is replaced by the new symbol f_{ji}[4]. As an example, $\texttt{pat}(f(x,a,a)) = f_{31}(x_1, f_{01}, f_{01})$. A representative pattern for an equation is computed by first orienting the equation according to some ordering stable with respect to the pattern

[3]While these facts are not useful in the proof, they also do not lead to a significant increase in search cost. Hence, we try to select more important negative examples.

[4]This definition specializes the one given in [DS96a], ensuring that each new symbol is only used with one arity even in independently generated patterns.

transformation, and then treating it as a single term with top symbol '='. For details consult [DS96b]. It is important to note that representative patterns of both terms and equations are terms over a new signature with function symbols $\{f_{01} \ldots f_{0n}, f_{11} \ldots f_{1n}, f_{m1} \ldots f_{mn}\}$ for suitably large numbers of n and m. Thus, all operations on terms (including learning algorithms) can be directly transferred to patterns.

To store additional properties of facts, we *annotate* term patterns. Annotations of term patterns carry e.g. information about the number of positive/negative occurrences of terms corresponding to a given pattern in a proof, or the distance of a fact from a proof path. Of course multiple equations can be mapped to the same pattern. If this happens, the corresponding annotations are accumulated. For most annotations, e.g. number of applications, this accumulation involves a simple addition. Other annotations, like the names of the proofs (kept for administrative purposes), are stored as a list.

We represent a proof by a set of annotated patterns, corresponding to the facts selected as examples. These sets are stored in a knowledge base. Currently, we use a flat, unstructured knowledge base for term space mapping. Annotated patterns from multiple proof are collected in a single set, and annotations from different proofs are freely combined. However, work on the preselection of subsets of patterns, based on similarity measures on problem specifications, is currently carried out (see section 5).

3.3 Term Space Maps

Our learning approach tries to detect whether a certain function symbol in a certain position in a term increases or decreases the likelihood of a term to belong to a given class (or, in our case, to contribute to a new proof). This class of features also includes information like "terms above a certain size are bad" or "terms who reach their maximal depth in the first argument are good". To uncover this kind of knowledge, we compile the annotated term patterns from multiple proofs to construct a *term space map* (TSM).

A TSM is a recursive structure describing the distribution of certain properties among term nodes in a given set of terms (and thus describes some topological properties of the term set). It generalizes the concept of a *term evaluation tree* [DS96a]. More exactly, a term space map $T = \{A_1, \ldots, A_n\}$ is a set of *term space alternatives* (TSAs) such that any given function symbol occurs as the first element in at most one TSA (which represents the subset of terms with this top symbol). A TSA is a tuple $(f, (T_1, \ldots, T_n), a)$, where f is a function symbol of arity n, the T_i are TSMs corresponding to the principal argument positions of f, and a is an annotation accumulating the annotations of the terms in the term set corresponding to this TSA. figure 2 shows the graphical representation of a very simple example which uses natural numbers as annotations and accumulates them by addition.

Even fairly large sets of example terms or equations can be compiled into the corresponding TSM efficiently[5].

A compiled term space map records information about the occurrence of term nodes in a set of terms (and about their annotations). It can be used to evaluate terms. This is done by selecting the subset of TSM nodes corresponding to nodes in the term, and compute an evaluation based on the the annotations stored at these nodes. More exactly, a term can be

[5]The knowledge base used in our experiments (Section 4) contains about 5000 term patterns, resulting from more than 40000 equations. It is read from the disk and compiled into the TSM in about 1.5 seconds.

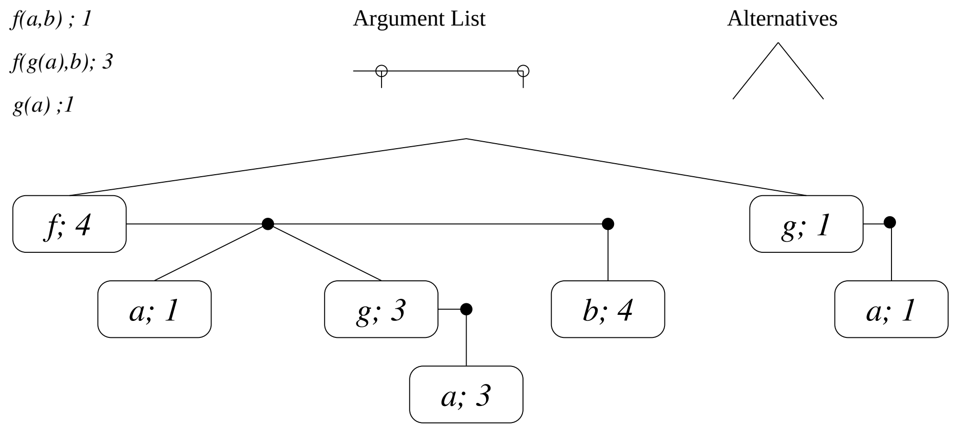


Figure 2: Example TSM for $\{(f(a,b);1),(f(g(a),b);3);(g(a);1)\}$

mapped onto a TSM by first selecting the appropriate TSA (if it exists) and then mapping the arguments of the terms onto the corresponding sub-TSMs. This operation selects a number of TSM nodes, which can then be used to evaluate the term.

As an example, consider the TSM in figure 2 and use the sum of the annotations of nodes corresponding to a term as the evaluation of the term. Then the term $f(g(a),a)$ is evaluated as follows: $f(g(a),a)$ is mapped onto the left TSA, with the top node $f;4$. The first argument, $g(a)$ is mapped onto the second TSA of the first argument TSM, i.e. the node labeled $g;3$. Similarly, the subterm a of $g(a)$ is mapped onto the node $a;3$. The second argument of the whole term, a, does not correspond to any TSA and remains unmapped (for a more realistic evaluation function, we would assign a default weight here). The resulting evaluation of the whole term is $4+3+3=10$.

Our learning algorithm compiles the patterns of example equations into a TSM, using the number of positive and negative occurrences of an equation in a proof search as annotations. During the proof search, new equations generated by the prover are transformed into patterns and mapped onto the TSM. An evaluation for the equation is computed by assigning a bonus to term nodes mapped to TSM nodes occurring primarily in positive examples and a penalty to nodes occurring primarily in negative examples, with a default evaluation for term nodes not represented in the TSM. More exactly, at each TSM node corresponding to a term node in the equation to be evaluated, we compute the distribution of positive and negative annotations among the alternatives, and modify the base weight of the corresponding term node accordingly. The resulting weights for individual term nodes are summed up into an evaluation of the equation. This sum is used as a selection criterion for the next equation to process (compare section 2).

4 Experimental Results

Table 1 shows the performance of the two standard heuristics described above and the TSM-based heuristic in terms of problems solved in a given (per-problem) time limit. We used the set of all unit-equality problems with universally quantified goals from version 2.0.0 of the TPTP problem library [SSY94], i.e. all problems suitable for using ordinary unfailing completion. This set consist of 303 problems, 247 for which there are known proofs, 51 which should be provable, but for which no proof by any ATP system has been reported, and 5 of entirely open status. Tests were performed on a SUN SPARCstation Ultra 2/200 running Solaris 2.5.1. All times given are cpu-times.

Time limit (sec.)	5	10	20	30	50	100	200	300
AddWeight	143	162	177	179	184	190	194	197
Occnest	155	155	168	171	174	183	187	188
TSM	161	173	186	189	199	204	207	207

Table 1: Comparative performance of different heuristics

The knowledge base contained proofs for 203 problems of the TPTP library. This set contains all problems that we were able to proof with DISCOUNT's other strategies. As the table shows, the learning heuristic performs better than both conventional ones at all time limits. Despite the fact that we used run time limits up to 1000 seconds and a large number of different strategies (including distributed teams) to find the proofs used in training, the single TSM-based strategy was able to proof all training problems in at most 160 seconds, with an average of only 6 seconds per problem. Moreover, the strategy was able to solve 4 problems previously unprovable by DISCOUNT despite all our efforts. It also solved 3 problems (LCL109-2, RNG009-5, and RNG009-7), for which no previous proofs had been recorded in the TPTP.

5 Conclusion

Our results once more confirm that patterns of equations contain enough information to adequately describe the decisions made during the search for an equational proof, and that information about good search strategies can be extracted from them. Moreover, they show that term space maps compiled from these equations still maintain enough of this information to allow a prover to efficiently reproduce the proofs and even allow a generalization to new cases.

Our future work will concentrate on a couple of topics. First, we will work on improving TSM-related learning algorithms by aiming at stronger expressive power. Currently, TSM-based learning algorithms only allow us to express some rather simple concepts, namely distribution of certain function symbols and simple substructures in the training set. By using more general *index functions* on terms to discriminate between term space alternatives, we can express more complex concepts, and can extend term space mapping to include pattern memorization as a special case. Finally, the introduction of *recurrent term space maps* should further increase the expressive power of the learning algorithm by generalizing to term nodes beyond the scope of the training set.

With the current approach, we found cases of over-training, i.e. degraded performance after adding additional examples, for most learning algorithms based on term space maps and term evaluation trees. We will address this problem by selecting suitable training examples (proof protocols) for a given proof problem, using both feature-based and TSM-based similarity measures. This will also support the use of more specialized heuristics for particular domains.

Another current focus of our work is the implementation of a more efficient inference engine capable of handling full clausal logic. This will allow us to evaluate our learning techniques for the more general case, and will hopefully also lead to an even more competitive prover.

References

[BDP89] L. Bachmair, N. Derschowitz, and D.A. Plaisted. Completion Without Failure. In *Coll. on the Resolution of Equations in Algebraic Structures, Austin, 1987*. Academic Press, 1989.

[Den93] J. Denzinger. *Teamwork: Eine Methode zum Entwurf verteilter, wissensbasierter Theorembeweiser*. PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, 1993. (German Language).

[Den95] J. Denzinger. Knowledge-Based Distributed Search using Teamwork. In *Proc. of the ICMAS-95, San Francisco*, pages 81–88. AAAI Press, 1995.

[DF94] J. Denzinger and M. Fuchs. Goal Oriented Equational Theorem Proving Using Team Work. In *Proc. of the KI94, Saarbrücken*, number 861 in LNAI, pages 343–354. Springer, 1994.

[DGHW97] B.I. Dahn, J. Gehne, T. Honigmann, and A. Wolf. Integration of Automated and Interactive Theorem Proving in ILF. In W. McCune, editor, *Proc. of the 14th CADE, Townsville*, number 1249 in LNAI, pages 57–60. Springer, 1997.

[DKS97] J. Denzinger, M. Kronenburg, and S. Schulz. DISCOUNT: A Distributed and Learning Equational Prover. *Journal of Automated Reasoning*, 18(2):189–198, 1997. Special Issue on the CADE 13 ATP System Competition.

[DS94a] J. Denzinger and S. Schulz. Analysis and Representation of Equational Proofs Generated by a Distributed Completion Based Proof System. Seki-Report SR-94-05, Universität Kaiserslautern, 1994.

[DS94b] J. Denzinger and S. Schulz. Recording, Analyzing and Presenting Distributed Deduction Processes. In H. Hong, editor, *Proc. 1st PASCO, Hagenberg/Linz*, volume 5 of *Lecture Notes Series in Computing*, pages 114–123, Singapore, 1994. World Scientific Publishing.

[DS96a] J. Denzinger and S. Schulz. Learning Domain Knowledge to Improve Theorem Proving. In M.A. McRobbie and J.K. Slaney, editors, *Proc. of the 13th CADE, New Brunswick*, number 1104 in LNAI, pages 62–76. Springer, 1996.

[DS96b] J. Denzinger and S. Schulz. Recording and Analysing Knowledge-Based Distributed Deduction Processes. *Journal of Symbolic Computation*, 21:523–541, 1996.

[DW97] B.I. Dahn and C. Wernhard. First Order Proof Problems Extracted from an Article in the MIZAR Mathematical Library. In *Proceedings of the 1st FTP, Linz*, pages 58–62. RISC Linz, Austria, 1997.

[FS97] B. Fischer and J. Schumann. SETHEO Goes Software-Engineering: Application of ATP to Software Reuse. In W. McCune, editor, *Proc. of the 14th CADE, Townsville*, number 1249 in LNAI, pages 65–68. Springer, 1997.

[Fuc95a] M. Fuchs. Experiments in the Heuristic Use of Past Proof Experience. Seki-Report SR-95-10, Universität Kaiserslautern, 1995.

[Fuc95b] M. Fuchs. Learning Proof Heuristics by Adapting Parameters. In *Proc. of the 12th International Workshop on Machine Learning*, pages 235–243, San Mateo, CA, 1995. Morgan Kaufmann.

[Fuc96] M. Fuchs. Experiments in the Heuristic Use of Past Proof Experience. In M.A. McRobbie and J.K. Slaney, editors, *Proc. of the 13th CADE, New Brunswick*, number 1104 in LNAI, pages 523–537. Springer, 1996.

[Fuc97] M. Fuchs. Automatic Selection of Search-Guiding Heuristics. In D.D. Dankel II, editor, *Proc. of the 10th FLAIRS, Daytona Beach*, pages 1–5. Florida AI Research Society, 1997.

[GK96] C. Goller and A. Küchler. Learning Task-Dependent Distributed Representations by Backpropagation Through Structure. In *Proc. of the ICNN-96*, volume 1, pages 347–352. IEEE, 1996.

[Gol94] C. Goller. A Connectionist Control Component for the Theorem Prover SETHEO. In *Proc. of the ECAI'94 Workshop W14: Combining Symbolic and Connectionist Processing*, pages 99–93. ECAI in cooperation with AAAI and IJCAI, 1994.

[Gol97] C. Goller. *A Connectionist Approach for Learning Search-Control Heuristics for Automated Deduction Systems*. PhD thesis, Fakultät für Informatik, Technische Universität München, 1997. (to be published).

[HBF96] T. Hillenbrand, A. Buch, and R. Fettig. On Gaining Efficiency in Completion-Based Theorem Proving. In H. Ganzinger, editor, *Proc. of the 7th RTA, New Brunswick*, number 1103 in LNCS, pages 432–435. Springer, 1996.

[HR87] J. Hsiang and M. Rusinowitch. On Word Problems in Equational Theories. In *Proc. of the 14th ICALP, Karlsruhe*, number 267 in LNCS, pages 54–71. Springer, 1987.

[KG96] A. Küchler and C. Goller. Inductive Learning in Symbolic Domains Using Structure-Driven Recurrent Neural Networks. In G. Görz and S. Hölldobler, editors, *Proc. of the 20th German Annual Conference on Artificial Intelligence, Dresden (KI-96)*, number 1137 in LNAI, pages 183–198. Springer, 1996.

[LSBB92] R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: A High-Performance Theorem Prover. *Journal of Automated Reasoning*, 1(8):183–212, 1992.

[Sch97] J. Schumann. Automatic Verification of Cryptographic Protocols with SETHEO. In W. McCune, editor, *Proc. of the 14th CADE, Townsville*, number 1249 in LNAI, pages 87–11. Springer, 1997.

[SE90] C.B. Suttner and W. Ertel. Automatic Acquisition of Search Guiding Heuristics. In M.E. Stickel, editor, *Proc. of the 10th CADE, Kaiserslautern*, number 449 in LNAI, pages 470–484. Springer, 1990.

[SKG97] S. Schulz, A. Küchler, and C. Goller. Some Experiments on the Applicability of Folding Architecture Networks to Guide Theorem Proving. In D.D. Dankel II, editor, *Proc. of the 10th FLAIRS Conference, Daytona Beach*, pages 377–381. Florida AI Research Society, 1997.

[SSY94] G. Sutcliffe, C.B. Suttner, and T. Yemenis. The TPTP Problem Library. In *Proc. of the 12th CADE, Nancy*, number 814 in LNAI, pages 252–266. Springer, 1994.

[WGR96] C. Weidenbach, B. Gaede, and G. Rock. SPASS & FLOTTER Version 0.42. In M.A. McRobbie and J.K. Slaney, editors, *Proc. of the 13th CADE, New Brunswick*, number 1104 in LNAI, pages 141–145. Springer, 1996.