

PDS — A Three-Dimensional Data Structure for Proof Plans

Lassaad Cheikhrouhou and Volker Sorge
Department of Computer Science (FB 14)
Saarland University
Postfach 151150, D-66041 Saarbrücken, Germany
{lassaad|sorge}@ags.uni-sb.de

Abstract

We present a new data structure that enables to store three-dimensional proof objects in a proof development environment. The aim is to handle calculus level proofs as well as abstract proof plans together with information of their correspondences in a single structure. This enables not only different means of the proof development environment (e.g., rule- and tactic-based theorem proving, or proof planning) to act directly on the same proof object but it also allows for easy presentation of proofs on different levels of abstraction. However, the three-dimensional structure requires adjustment of the regular techniques for addition and deletion of proof lines and backtracking of the proof planner.

1. Introduction

In some deduction systems, especially those for doing mathematics, proofs are explicitly kept [2, 7]. These *proof objects* generally consist of a planar, acyclic graph that stores derivations in a certain calculus, e.g., natural deduction calculus [8], together with some information on the proof history, i.e. information for backtracking. In tactic based theorem provers (c.f. [10]) proof tactics are either immediately executed, thereby introducing several calculus level proof steps into the proof object, or tactics are considered as macro steps which are equally stored within the proof steps without the possibility to view the calculus level subproof they abbreviate [2]. Furthermore, in proof planning systems (c.f. [5]) abstract proof methods are automatically combined by a planning component to proof plans which, when refined, result in a calculus level proof. However, both planning and projection onto calculus level is done in separate data structures such that the correspondence between single planning steps and the subproofs they contribute becomes blurred.

In the Ω MEGA system [3] we have developed a new

way of storing both proof objects and proof plans within a single data structure, called Proof Plan Data Structure (*PDS*). It enables to represent abstract tactics and methods as well as calculus level proof steps in the same proof object. Moreover, it is not just a planar graph, but has a three-dimensional structure that allows for representing direct correspondences between abstract proof steps and concrete calculus level subproofs. These correspondences can be successfully exploited when expanding abstract proof plans into machine checkable calculus level proofs. Furthermore, it aids the presentation of a proof in a proof development environment such as Ω MEGA, since it allows not only for displaying different levels of abstraction of a proof but also for freely shifting these levels with the help of expansion of proof tactics or methods and contraction of subproofs into abstract step. However, proof development in the *PDS* requires adjustment of the regular techniques for addition and deletion of proof lines and backtracking of the proof planner.

The paper is organized as follows: In Sec. 2 we elaborate our view on proofs and proof plans in the Ω MEGA system which will consequently lead to a proof object such as the *PDS*. We then give an example for an entry in the *PDS* in Sec. 3 which we will refer to when giving an overview on the single components and different operations on data structure in Sec. 4 and 5, respectively. We conclude and hint at possible future work in Sec. 6.

2. Characterization of the *PDS*

In Ω MEGA proofs can be constructed by automated or mixed-initiative planning, or by pure user interaction, starting from an initial problem consisting of a theorem together with asserted hypotheses. In particular, new pieces can be added to a proof by directly calling tactics, by inserting facts from a data base, or by calling some external reasoner, such as automated theorem provers or computer algebra systems. All these different kinds of reasoning methods are uniformly viewed as general *abstract inference steps* that

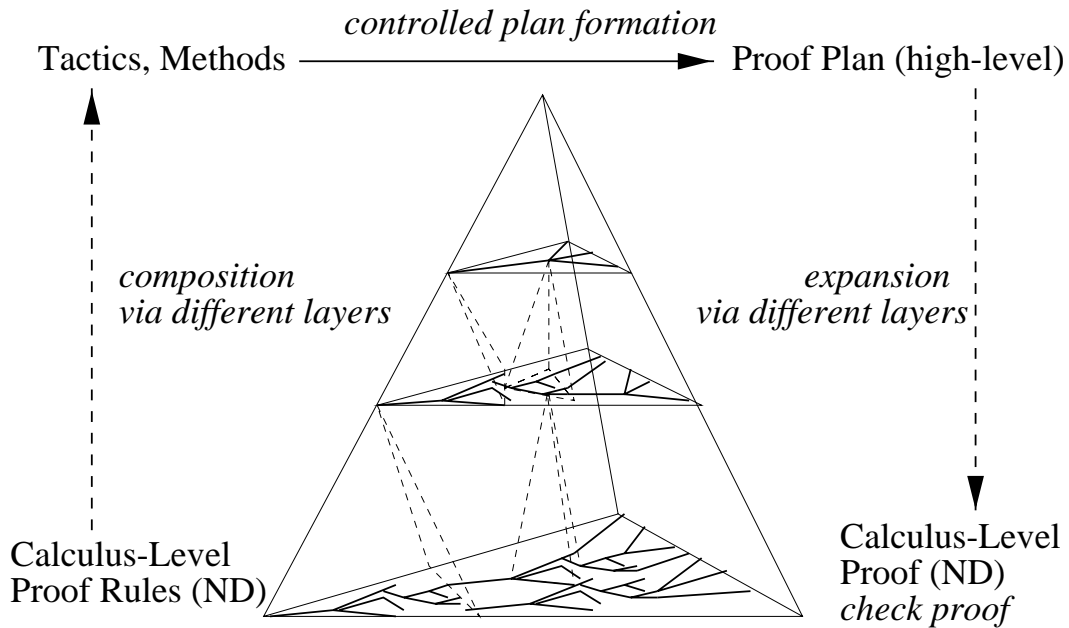


Figure 1. Proof Plan Data Structure.

can be formalized as

$$\frac{P_1, \dots, P_n}{C_0, \dots, C_m} S(T_1 \dots T_l), \quad (1)$$

where S is the name of the inference, e.g., the name of a tactic, C_0, \dots, C_m are *conclusions*, i.e. formulas that can be derived by applying S to the *premises* P_1, \dots, P_n with the help of the *additional parameters* $T_1 \dots T_l$. These additional parameters, for instance, can be some required terms or term positions. Note, that for an inference step at least one conclusion is mandatory, whereas premises and parameters are optional, only. This is also indicated by the choice of indices.

However, the Ω MEGA system accepts proofs that are machine-checkable in its own basic calculus, a natural deduction (ND) calculus [8] based on a typed higher order logic [6], only. Therefore, abstract inference steps are just valid if they can be expressed in terms of *primitive inference steps*, i.e. the rules of the ND calculus. Thus, we have to view any abstract inference step as an abbreviation of a subproof in ND calculus to which it can be expanded. Unlike in other tactical theorem proving or proof planning systems, the expansion of abstract inferences in Ω MEGA is generally not carried out immediately but postponed until the given theorem is fully justified from the hypotheses. Yet, as long as a proof step is still abstract it is considered as *planned*¹ —

¹We call abstract inference steps *planned* since we allow for specifying inferences that can sometimes be faulty. Hence, the expansion of such a step can fail, leaving a part of the proof still open. This feature permits us to

therefore the name proof plan data structure — and needs to be expanded in order for the proof to be fully checkable.

The expansion of an abstract inference step does not necessarily have to yield a subproof which contains primitive inferences, only. Instead, there might also be other abstract inference steps included in the subproof which, in turn, are expandable. This enables us to specify hierarchies for abstract inferences and abstraction levels within the proof that represent proof plans of different granularity. Moreover, the process of expansion is reversible, that is, once an abstract inference step has been expanded, the abstract step is not simply discarded, instead the subproof resulting from the expansion, can be contracted again to the single abstract step. Figure 1 depicts this view of proofs, albeit it is a schematic depiction of the reality, since there can be more than two levels of abstractions as well as the abstraction levels cannot be that easily distinguished.

The three-dimensional view of proofs leads to our implementation of the proof plan data structure (\mathcal{PDS}). The logical dependencies between single proof steps, corresponding to a horizontal level in the \mathcal{PDS} , are best modeled with a directed acyclic graph. This graph needs to be extended in order to also maintain the dependencies between abstract inference steps and the subproofs they abbreviate, which corresponds to the vertical links in Fig. 1. Besides the insertion and deletion of proof steps we need the expansion of abstract inferences as additional operation on the \mathcal{PDS} . Since

employ uncertain heuristics and external reasoners that are not necessarily always correct.

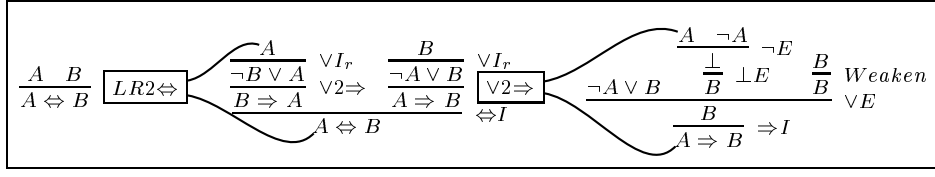


Figure 2. Expansion of the tactic $LR2\leftrightarrow$.

expansion generally leads to the insertion of new proof steps (on a less abstract level) we need to specially treat the insertion and deletion of those steps taking the vertical dependencies in the \mathcal{PDS} into account.

3. Example

Before we describe the components of the \mathcal{PDS} and its operations in more detail, we illustrate the overall idea, in this section, by giving an example of an abstract proof tactic and its expansion. Although the example is relatively small due to the limited space, it sheds some light on the nature of the \mathcal{PDS} and suffices to explain most of its peculiarities. Thus, we will refer to this example throughout the remainder of this paper.

Figure 2 depicts the expansion of the relatively simple tactic $LR2\leftrightarrow$. It implements the inference step that if the two facts A and B can be derived in the same proof they have to have equivalent truth-value.

Since the tactic is only an abbreviation of a more complicated proof on the ND-calculus level, we can expand the abstract $LR2\leftrightarrow$ step to the next, less abstract, level. This expansion is also depicted in Fig. 2 in the second column. Here the subproof is a derivation of the two directions of the equivalence by exploiting the logical equivalence of $\neg A \vee B$ and $A \Rightarrow B$. Note, that the curves in Fig. 2 correspond to the vertical links in Fig. 1.

However, in the derivation of the second column only the steps justified by $\vee I_r$ are already ND proof steps, whereas the justifications $\vee 2\Rightarrow$ and $\Leftrightarrow I$ consist of abstract tactics themselves. Thus, these steps can, in turn, be expanded and this has been executed for the second application of $\vee 2\Rightarrow$ in the right column of Fig. 2. In this subproof all the justifications contained are now basic ND-rules and consequently cannot be expanded any further. Therefore, the right column of Fig. 2 actually corresponds to a part of the lowest layer of the \mathcal{PDS} as depicted in Fig. 1.

4. \mathcal{PDS} components

In this section, we describe the \mathcal{PDS} components and explain how they implement the characterization of Sec. 2.

Before elaborating the details, we need to introduce the actual format of proof nodes in the \mathcal{PDS} . This format is a linearized format for ND proofs as introduced, for instance, in [1]. Figure 3 depicts the first expansion step of the $LR2\leftrightarrow$ tactic from Fig. 2 in linearized form.

The components a proof node is composed of are: a unique *name* (e.g., L_2), a *sequent* ($\Delta \vdash A \leftrightarrow B$) and a *justification* ($LR2\leftrightarrow: L_1L_3$). The sequent itself consists of a formula and, preceding the \vdash symbol, a set of hypotheses the formula depends on. Another feature of a node is a so called *reason list* which stores information for backtracking purposes. The node reasons inform how the node was introduced into the \mathcal{PDS} and how it is related to other nodes. A reason indicates, for example, whether the node was created by some inference application or rather by expanding some abstract inference. Reasons are not displayed in Fig. 3, but are crucial for the operations on the \mathcal{PDS} presented in Sec. 5.

One observation we can make in the expansion of node L_2 in Fig. 3 is, that solely the justification of the node changes from one abstraction level to the other. And indeed, all other components, i.e. name, sequent, and reasons, remain constant on all levels of abstraction. Therefore, justifications implement both the horizontal and vertical links in the \mathcal{PDS} and motion between the abstraction levels is implemented by changing the justification of a node from more to less abstract or vice versa.

A justification consists of an inference (i.e. a ND rule, tactic, method, etc.), a set of premises (i.e. nodes the given node can be derived from with the inference), and possibly a list of additional parameters necessary for the application of the inference. The set of premise nodes functions as the horizontal links in the \mathcal{PDS} , since they determine the logical connections between the single nodes. As additional features a justification can have two pointers that function as the vertical links in the \mathcal{PDS} . There can be at most one pointer to a more abstract justification (*above link*) and at most one pointer to a less abstract justification (*below link*). These pointers might be empty in case the justification is on the topmost or lowest level of the \mathcal{PDS} , respectively. Changing the level of abstraction automatically introduces (or excludes) nodes from the proof as shown in Fig. 3: On the left-hand side, the horizontal links from node L_2 to L_1

	\vdots		L_1	$\{L_1\} \vdash A$	(Hyp)
L_1	$\{L_1\} \vdash A$	(Hyp)	L_4	$\Delta \vdash \neg B \vee A$	$(\forall I_r: L_1)$
L_3	$\Delta \vdash B$	(Open)	L_5	$\Delta \vdash B \Rightarrow A$	$(\forall 2 \Rightarrow: L_4)$
L_2	$\Delta \vdash A \Leftrightarrow B$	$(LR2 \Leftrightarrow: L_1 L_3)$	L_3	$\Delta \vdash B$	(Open)
	\vdots		L_6	$\Delta \vdash \neg A \vee B$	$(\forall I_r: L_3)$
	\vdots		L_7	$\Delta \vdash A \Rightarrow B$	$(\forall 2 \Rightarrow: L_6)$
	\vdots		L_2	$\Delta \vdash A \Leftrightarrow B$	$(\Leftrightarrow I: L_5 L_7)$

Figure 3. Expansion of the tactic $LR2 \Leftrightarrow$ in linearized form.

and L_3 are direct, whereas on the right-hand side, the link to L_1 is via L_5 and L_4 , the link to L_3 via L_7 and L_6 .

Exceptions to a justification as described are (Hyp) and (Open) justifications. The former denote lines that are genuine hypotheses for the original theorem and are not derived from any other line. Thus, they do not contain a list of premises and function as leaf nodes of a horizontal level in the \mathcal{PDS} . The latter, (Open) justifications, indicate a subgoal that is yet to be proven. Neither hypothesis nor open justifications contain horizontal or vertical links. However, in order to determine the applicability of inferences to a respective open proof line we also have a concept of *support lines*, i.e. those lines that are eligible to deduce the open line from, that is similar to the one introduced in [1]. Support lines are computed whenever an open node is created and can be updated after application of single inferences.

5. \mathcal{PDS} operations

Proofs in Ω MEGA are mainly constructed by interleaving inference application and inference expansion. Both operations, described below in Sec. 5.1 and Sec. 5.2, can insert new open nodes as subgoals. When such an open node cannot be closed, the inference which originally led to this node, together with other related proof construction steps, must be retracted. The removal of an inference step is explained in Sec. 5.3.

5.1. Inference application

The application of a single inference, always focuses on some open nodes in the \mathcal{PDS} , which we call the *focus goals*. The inference application should contribute to the proof of the focus goals by closing some of them and updating the supports of the rest. The focus goals which are closed directly are called the *primary goals* and those which remain open are referred to by the *secondary goals*. None of the focus goals is primary (secondary), when the inference is applied forwards (backwards). Inferences with more than one conclusion can be also applied sideways, where we have to deal simultaneously with primary and secondary goals.

Generally, some conclusions and premises of an inference are matched with focus goals and *focus supports* respectively, where the *focus supports* correspond to the common supports of the focus goals. The matched conclusions are the primary goals and the matched premises are called the *existent premises*. For instance, line L_2 in the left column of Fig. 3 matches the sole conclusion of the tactic $LR2 \Leftrightarrow$ and corresponds therefore to the primary goal. In this backward application, the left premise of $LR2 \Leftrightarrow$ is matched with the focus support L_1 , which is the sole existent premise, and a new open line L_3 is introduced for the right premise. Nodes like L_3 are called the *inference subgoals*, since they are created *open* for inference premises which are not matched with focus supports.

In principle, the inference subgoals inherit the focus supports. However, an inference application may exclude some of the inference premises as supports of the resulting inference subgoals. We call these premises the *delete premises*. In inferences, like the ND rule $\Rightarrow I$, new hypotheses are introduced locally as additional assumptions for some premises, we call these *extra hypotheses*. Subsequently, each inference subgoal that results from the application of an inference introducing extra hypotheses will have these as additional supports.

A specific operation in the forward application of an inference is the updating of the supports of the secondary goals by excluding the delete premises and adding the nodes which are created for the inference conclusions, called the *add conclusions*. Possibly arising inference subgoals are treated similar to the backward case.

We formalize the support changes for inference applications of arbitrary direction. Consider the inference \mathcal{S} , given in (1), with conclusions C_0, \dots, C_m . Let G_0, \dots, G_f be the focus goals, SG_1, \dots, SG_s inference subgoals, G'_1, \dots, G'_{f+a-m} secondary subgoals, S_1, \dots, S_d delete premises, and C'_1, \dots, C'_a the add conclusions. The supports of an inference subgoal SG_i is initialized according to (2), and those of a secondary goal G'_i is updated according to (3).

All the conclusion nodes in an inference application, the primary goals as well as the add conclusions, are associated the same justification which consist of the same inference,

$$\text{supps}(SG_{i \in \{1, \dots, s\}}) := \left(\bigcap_{j=0}^f \text{supps}(G_j) \setminus \{S_1, \dots, S_d\} \right) \cup \text{extra_hyps}(SG_i) \quad (2)$$

$$\text{supps}(G'_{i \in \{1, \dots, f+a-m\}}) := (\text{supps}(G'_i) \setminus \{S_1, \dots, S_d\}) \cup \{C'_1, \dots, C'_a\} \quad (3)$$

parameters, and premises. However, these justifications are distinct objects since they are generally associated different justifications in the inference expansion. Each node involved in the inference application, i.e. the conclusion nodes, the premise nodes, the extra hypotheses, and the secondary goals, gets a new reason associated containing detailed *control information* of the application step (i.e. information on all effects of inference application on the \mathcal{PDS}). This control information is crucial for both expanding the inference (see Sec. 5.2) and retracting it (see Sec. 5.3).

5.2. Inference expansion

For the expansion of an abstract inference it is important to consider whether the inference has more than one conclusion. In this case the expansion of a given node imposes to expand all other conclusions simultaneously. These nodes are obtained from the reason that represents how the given node was justified by the abstract inference.

Instead of the sole proof step which justifies the conclusion nodes, a more detailed proof plan is inserted by the inference expansion on the next, less abstract, level of the \mathcal{PDS} . This *expansion proof plan* possibly includes new nodes, called *expansion nodes*. Each of the conclusion nodes is associated a new justification which is connected to the original, now expanded, justification by vertical links. That is, the below link of the expanded justification points to the new justification, whose above link, in turn, refers to the old, more abstract, justification. The above and below links of the expansion nodes are initially empty.

For backtracking purposes we keep a *direct* dependency of the expansion nodes to the *original* inference application whose associated proof plan includes these nodes: If a node is expanded that was closed by an inference application — it then corresponds either to a primary goal or to an add conclusion (see Sec. 5.1) — the control information for this inference application is passed to all the expansion nodes as a reason. Likewise, when expanding one of these expansion nodes the same control information is passed on to any originating expansion node. Thus, the information on which original inference step an expansion node depends on is propagated by subsequent expansion steps.

In our example of the expansion of the tactic $LR2 \Leftrightarrow$ in Fig. 3 the below link of the justification ($LR2 \Leftrightarrow: L_1 L_3$) of L_2 points to ($\Leftrightarrow I: L_5 L_7$) and its above link vice versa. The reasons of the expansion nodes L_4, \dots, L_7 correspond

to the reason of L_2 which represents the application of the tactic $LR2 \Leftrightarrow$. The same reason would be associated to the expansion nodes of the tactic $\forall 2 \Rightarrow$ in the current justification of the node L_7 .

One way of implementing hierarchical proof planning is to hide some subgoals until a planning method is expanded. The expansion proof plan of such a method comprises consequently open nodes, which we call the *expansion subgoals*, whose supports have to be initialized. Since we suppose that the premises of the expanded justification suffice to construct a complete proof plan of the expanded node, these premises are used as supports to the expansion subgoals. In addition to the premises of the expanded justification, the supports of each expansion subgoal can contain its own extra hypotheses specified in the method.

5.3. Undoing inference application

The removal of an inference application includes, in addition to retracting its elementary effects, the recursive removal of other related proof steps in order to maintain a consistent state of the \mathcal{PDS} . For proof planning purposes, control information needs to be updated to prevent the re-consideration of the same inference in the same application context.

Let R be the reason that contains the control information for the application of inference I_R that is to be removed. The undoing operation, as described below, is based on this control information.

- Suppose the \mathcal{PDS} contains a proof plan for I_R after several expansions. This proof plan must be removed first by deleting those intermediate nodes, between the conclusion nodes and the premise nodes, which are associated the reason R . Such a node can be involved in other inference applications, when it corresponds to an expansion subgoal, i.e. it is introduced open by some expansion. In this case we recursively remove these inference applications as well.
- The retraction of the elementary effects of I_R is carried out by:
 - deleting the inference subgoals and the add conclusions, which includes the recursive undoing of other inference applications these nodes are involved in,

- removing R from the reason lists of the remaining, involved, nodes, i.e. the focus goals and the existent premises,
 - reopening the primary goals, and
 - updating the supports of the secondary goals and of their subgoals which were created after I_R was introduced into the \mathcal{PDS} , by replacing the add conclusions with the delete premises.
- I_R , including its parameters (and possibly other information which define the application context), has to be stored as a failed step for the focus goals.

In addition to the information on the involved nodes and on their role in the application of I_R , we need temporal information wrt. other inference applications in the \mathcal{PDS} . This allows us to determine the subgoals of the secondary goals which were created after I_R was introduced in order to update their supports. Each secondary goal can be involved in inference applications which precede I_R and in other applications after I_R . Only the inference subgoals in the latter applications and their recursive inference subgoals inherit the supports of the secondary goals, which were modified by I_R .

For instance, the undoing of the application of $LR2 \Leftrightarrow$ in Fig. 3 would delete the expansion nodes L_4, \dots, L_7 and the subgoal node L_3 . Moreover, the node L_2 is reopened and the tactic $LR2 \Leftrightarrow$ is noted as a failed step which may not be applied again to the now open node L_2 .

The operation of undoing some inference application serves to implement the deletion of an arbitrary node in the \mathcal{PDS} . A given node can be deleted by undoing the original inference application which introduced this node and recursively all other inference applications that depend on the existence of this node. The necessary control information is given in the reason list of the deleted node. For instance, if a node is deleted that originated from an expansion, the original inference on whose expansion the node depends, must be removed, too. This is secured by the reason that has been propagated from the original inference as described in Sec. 5.2.

6. Conclusion and future work

We have presented a new data structure \mathcal{PDS} for storing proofs within a deduction system. Its main feature is a way to represent proofs in a three-dimensional way thereby enabling to store several levels of abstraction simultaneously. On the lowest level of the \mathcal{PDS} proofs are represented in the basic calculus of the deduction system. Higher levels contain abstract steps that abbreviate subproofs on the respective lower level. The \mathcal{PDS} provides facilities to introduce and remove proof steps of different granularity as well

as to expand abstract steps into lower-level sub-proofs and to maintain the appropriate correspondences.

Although we have presented the \mathcal{PDS} in the context of a higher order natural deduction calculus — the basis calculus of the Ω MEGA system — the data structure can obviously be used in any tactical theorem prover independent of the underlying calculus. In fact, since different calculi can be specified within Ω MEGA the \mathcal{PDS} is already used to represent proofs of other systems, namely TPS [2] and LEO [4], where the former implements a higher order natural deduction calculus (however slightly different of the one used in Ω MEGA) and the latter works with a higher order resolution calculus. Unfortunately, the advanced features of the \mathcal{PDS} , which necessitate to maintain many dependencies, make the implementation of both objects and algorithms of the data-structure slightly cumbersome.

So far one can introduce in the \mathcal{PDS} abstract inferences and subsequently expand them or abstract automatically from the ND-level to the *assertion level* [9]. However, it will be desirable to provide a mechanism that enables a user to manually abstract within an already constructed proof in a meaningful way. Thus future work has to include the construction of means to enable manual abstraction. Moreover, abstraction should not only be available on the level of justification but on the level of formulas as well. Thereby proofs can also be presented including simplified (abstracted) formulas.

References

- [1] P. Andrews. *An Introduction To Mathematical Logic and Type Theory: To Truth Through Proof*. Acad. Press, 1986.
- [2] P. Andrews, M. Bishop, S. Issar, D. Nesmith, F. Pfenning, and H. Xi. TPS: A Theorem Proving System for Classical Type Theory. *J. of Autom. Reasoning*, 16(3):321–353, 1996.
- [3] The Ω MEGA Group. Ω MEGA: Towards a Mathematical Assistant. In *Proceedings of CADE-14*, LNAI 1249. Springer, 1997.
- [4] C. Benzmüller and M. Kohlhase. LEO – a Higher Order Theorem Prover. In *Proceedings of CADE-15*, LNAI 1421. Springer, 1998.
- [5] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The OYS^TER-CLAM system. In *Proceedings of CADE-10*, LNCS 449. Springer, 1990.
- [6] A. Church. A Formulation of the Simple Theory of Types. *J. of Symbolic Logic*, 5:56–68, 1940.
- [7] R. Constable, et al. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice Hall, 1986.
- [8] G. Gentzen. Untersuchungen über das logische Schließen I. *Mathematische Zeitschrift*, 39:176–210, 1935.
- [9] X. Huang. Reconstructing Proofs at the Assertion Level. In *Proceedings of CADE-12*, LNAI 814. Springer, 1994.
- [10] L. Paulson. *Isabelle: a Generic Theorem Prover*, LNCS 828. Springer, 1994.