Natural Language Proof Explanation

Armin Fiedler

FR Informatik, Universität des Saarlandes,
Postfach 15 11 50, D-66041 Saarbrücken, Germany
afiedler@cs.uni-sb.de

Abstract. State-of-the-art proof presentation systems suffer from several deficiencies. First, they simply present the proofs without motivating why the proof is done as it is done. Second, they neglect the issue of user modeling and thus forgo the ability to adapt the presentation to the specific user. Finally, they do not allow the user to interact with the system to ask questions about the proof.
As a first step to overcome these deficiencies, we developed a computational model of user-adaptive proof explanation, which is implemented in a generic, user-adaptive proof explanation system, called ***P.rex*** (for ***pr****oof* ***ex****plainer*). To do so, we used techniques from three different fields, namely from computational logic to represent proofs ensuring the correctness; from cognitive science to model the users mathematical knowledge and skills; and from natural language processing to plan the explanation of the proofs and to react to the user's interactions.

1 Introduction

Today, automated theorem provers are becoming increasingly important in practical industrial applications and increasingly useful in mathematical education. For many applications, it is important that a deduction system communicates its proofs reasonably well to the human user. To this end, proof presentation systems have been developed.

However, state-of-the-art proof presentation systems suffer from several deficiencies. First, they simply present the proofs, at best in a textbook-like format, without motivating why the proof is done as it is done. Second, they neglect the issue of user modeling and thus forgo the ability to adapt the presentation to the specific user, both with respect to the level of abstraction chosen for the presentation and with respect to steps that are trivial or easily inferable by the particular user and, therefore, should be omitted. Finally, they do not allow the user to interact with the system. He can neither inform the system that he has not understood some part of the proof, nor ask for a different explanation. Similarly, he cannot ask follow-up questions or questions about the background of the proof.

As a first step to overcome these deficiencies, we developed a computational model of user-adaptive proof explanation, which is implemented in a generic, user-adaptive proof explanation system, called ***P.rex*** (for ***pr****oof* ***ex****plainer*). The

D. Hutter, W. Stephan (Eds.): Mechanizing Mathematical Reasoning, LNAI 2605, pp. 342–363, 2005.

demands we make on the system are the following: The system should adapt to the user with respect to the level of abstraction at which the proof is presented. Moreover, the system should allow the user to intervene if he is not satisfied with an explanation and to ask follow-up or background questions. The metaphor we have in mind is a human mathematician who teaches a proof to a student or else explains a proof to a colleague.

In our approach, we combine techniques from three different fields, namely cognitive science, computational logic and natural language processing, as we shall discuss in the following.

Modern natural language generation systems take into account the intended audience's knowledge in the generation of explanations (see e.g. [7, 39, 42]). Most of them adapt to the addressee by choosing between different discourse strategies. Since proofs are inherently rich in inferences, the explanation of proofs must also consider which inferences the audience can make [44, 19, 20]. However, because of the constraints of the human memory, inferences are not chainable without costs. Explicit representation of the addressee's cognitive states proves to be useful in choosing the information to convey [43].

In cognitive science various theories of human cognition have been described by means of a *cognitive architecture*, that is, the fixed structure that realizes the cognitive apparatus. One of these theories is ACT-R [1], a cognitive architecture that separates declarative and procedural knowledge into a declarative memory and a production rule base, respectively. A goal stack allows ACT-R to fulfill a task by dividing it into subtasks, which can be fulfilled independently.

ACT-R combines the abilities for *user modeling* and *planning* in a uniform framework and is therefore particularly well suited as a basis for a user-adaptive dialog planner. Using ACT-R, we model a teacher who explains mathematical proofs to his student. In particular, we model the teacher's assumptions about the students cognitive states during the explanation (which establish the user model) and the teacher's knowledge of the mathematical theories and the way to explain the proof of a theorem in these theories (which is used to plan the explanation). The assumptions about the user's cognitive states are employed to choose the level of abstraction at which a proof is presented.

An important prerequisite that enables *P.rex* to choose among different levels of abstraction is the simultaneous representation of a proof at several levels of abstraction. This representation, which also serves as the interface between theorem provers and *P.rex*, is realized in TWEGA, an implementation of an extension of the *logical framework* LF [17] that corresponds to the *calculus of constructions* [9]. Both are very powerful typed lambda calculi that allow us to represent other logical calculi, and thus give us the possibility to represent the calculus of the theorem provers that are to be connected to *P.rex*. The input proof to be explained as well as relevant information from the mathematical theories that relate to the proof are encoded in TWEGA and then used by *P.rex*. In particular, using an expansion mechanism that defines for any derivation step a subproof at the lower level of abstraction, TWEGA allows us to represent a proof at several levels of abstraction simultaneously.

Successful communication via a natural language generation system presupposes that the content to be conveyed is appropriately structured to ensure a coherent semantic organization. For an explanation system, it is also important to accept user feedback and follow-up questions. To be able to clarify misunderstood explanations, the system needs to represent the different parts of the explanation as well as the relations between them.

An appropriate discourse theory that allows for these representations was formulated by Mann and Thompson [30]. This theory, called *Rhetorical Structure Theory (RST)*, states that the relations that hold between segments of normal English text can be represented by a finite set of relations. Based on RST, Hovy [21] described discourse as the nesting of discourse segments. According to his discourse theory, each segment essentially contains the communicative goal the speaker wants to fulfill with this segment and either one to several discourse segments with intersegment discourse relations or the semantic material to be communicated. Adapting Hovy's discourse segments, we define *discourse structure trees* as a representation of discourses for *P.rex*. The discourse structure trees are built by plan operators, which are defined in terms of ACT-R production rules.

This paper is organized as follows: First, in Section 2 we shall review relevant research in proof presentation and formulate requirements for proof explanation. Next, in Section 3, we shall give an overview of the architecture of *P.rex*. Section 4 is devoted to ACT-R, the theory of human cognition that serves as a basis for the dialog planner of *P.rex*. The dialog planner itself is the subject of Section 5. We shall define discourse structure trees to represent the discourse and plan operators to construct them. In particular, we shall show how the system uses special plan operators to adapt its explanations to the user. Moreover, we shall discuss the system's reaction to a user's interactions. This will be illustrated by some examples in Section 6.

2 Proof Presentation

While the field of automated theorem proving matured in the last four decades, it became more and more apparent that the systems had to output proofs that can be more easily understood by mathematicians. To provide the proofs in the internal, machine-oriented formalisms of the theorem provers is by far not sufficient. Thus, proof presentation systems had to be designed that presented proofs in natural language at an appropriate level of abstraction.

The problem of obtaining a natural language proof from a machine-found proof can be divided into two subproblems: First, the machine-found proof is transformed into a human-oriented calculus, which is much better suited for presentation. Second, the transformed proof is verbalized in natural language. In the following, we shall examine both stages in some more detail.

2.1 Proof Transformation

Already in the thirties, Gentzen devised a human-oriented calculus that aimed to reflect the way mathematicians reason, the *natural deduction (ND) calculus*

[16]. However, most automated theorem provers are based on machine-oriented formalisms, such as resolution or matings. As a consequence, proofs encoded in such formalisms are not suited for a direct verbalization, because their lines of reasoning are often unnatural and obscure. To remedy this problem, researchers developed algorithms to transform proofs in machine-oriented calculi into ND proofs [2, 27, 35, 40, 29].

Initially, the idea was that the transformation into an ND proof is sufficient for the subsequent verbalization. But the results of the transformations into the ND calculus turned out to be far from satisfactory. The reason is that the obtained ND proofs are very large and too involved in comparison to the original proof. Moreover, in the ND calculus, an inference step merely consists of the syntactic manipulation of a quantifier or a connective. Huang [23] realized that in human-written proofs, in contrast, an inference step is often described in terms of the application of a definition, axiom, lemma or theorem, which he collectively calls *assertions*. Based on this observation, he defined an abstraction of ND proofs, called the *assertion level*, where a proof step may be justified either by an ND inference rule or by the application of an assertion, and gave an algorithm to abstract an ND proof to an assertion level proof [23]. Based on Huang's ideas, Meier described an algorithm to transform refutation graphs (a data structure that represents resolution proofs) directly into assertion level proofs [33]. The assertion level proves to be much better suited for a subsequent verbalization of the proofs than a traditional calculus. However, the assertion level sometimes includes steps that include implicit applications of modus tollens, which prove to be difficult to comprehend for human beings. Therefore, Horacek introduced the *partial* assertion level, where these applications are made explicit [20].

Since traditional search-based automated theorem provers find only proofs for theorems that are usually considered easy by mathematicians, theorem provers based on human-oriented approaches such as planning have been developed [5, 3]. The key idea here is that proof techniques as used by mathematicians are encoded into plan operators, which are used by the proof planner to find a proof plan. Because a proof plan is an abstract representation of a proof, it provides a level that is better suited for presentation, such that proof transformation becomes obsolete for proof planners.

2.2 Proof Verbalization

Now, let us turn our attention to approaches to verbalize proofs. One of the earliest proof presentation systems was EXPOUND [8]. It translated the formal proofs directly into English. Even though sophisticated techniques were developed to plan the paragraphs and sentences that made up the written proof, the system verbalized every single step of the formal proof in a template driven way, such that even small proofs are too detailed and still not easy to follow.

Proofs were also used as test input for early versions of MUMBLE [32], a natural language generation (NLG) system that adopted more advanced generation techniques. However, its main concern was not proof presentation, but to show the feasibility of its two-staged architecture for the generation of natural language.

Whereas the previously mentioned systems focused on problems in natural language generation and used formal proofs only as well-defined input for the generation process, research in the field of automated theorem proving addressed the readability of proofs as well. The following systems were developed in the field of automated theorem proving with the aim to obtain human-readable proofs.

The χ-proof system [14] was one of the first theorem provers that was designed with a natural language output component. The system showed every step of the derivation by using predefined templates with English words, but left the formulae in their logical form. The same is true for the pseudo-natural language presentation components of Coq [10] and the proof system *Theorema* [4]. The latter additionally allows the user to hide or unhide proof parts that he considers too detailed or not detailed enough, respectively.

Natural Language Explainer [12] was devised as a back end for the natural deduction theorem prover THINKER. Employing several isolated strategies, it was the first system to acknowledge the need for higher levels of abstraction when explaining proofs. Another presentation system that uses templates with canned sentence chunks to verbalize proofs is ILF [11]. It has been connected to several automated theorem provers, whose output proofs are presented in natural language at the logic level of the machine-oriented formalism of the respective prover.

PROVERB [22, 24] can be seen as the first serious attempt to build a generic, comprehensive NLG system that produces adequate argumentative texts from machine-found proofs. It takes as input ND proofs, which are first abstracted to the assertion level before any subsequent processing starts. *PROVERB* consists of a text planner, which chooses the information to be conveyed, a sentence planner, which plans the internal structure of the sentences, and the linguistic realizer TAG-GEN [26], which produces the output text. The system uses presentation knowledge and linguistic knowledge to plan the proof texts, which are output in a textbook-like format.

Another recently developed NLG system that is used as a back end for a theorem prover is the presentation component of Nuprl [18]. The system consists of a pipeline of two components. It employs a content planner that selects the information to be included in the output text and decides how to refer to the information in the given context. The text plan is then passed on to the surface realizer FUF [13], which chooses the words and outputs the actual sentences.

To sum up, to remedy the problem that many theorem provers return proofs in their internal, machine-oriented formalisms, which are very hard to understand, more and more human-oriented interfaces for theorem provers have been developed. But these interfaces are mostly used in the theorem proving community, that is, by people who usually have an insight in the provers' formalisms. The systems present proofs mostly at a very low level of abstraction and none of the systems adapts its output to the user or can handle follow-up questions. Whereas this might be tolerable for the developers of the theorem provers, it is certainly not acceptable for mathematicians or mathematics students who want to work with theorem provers.

2.3 Requirements for the Explanation of Proofs

When designing a proof explanation system, we may study a mathematics teacher and examine how he explains proofs to his students.

In education, human teachers use *natural language* for the presentation and explanation. Mathematics teachers often experience that students are demotivated by an overload of formulae. Hence, it can be expected that a proof explanation system, in particular if used by novices, is much more accepted if it also communicates derivations and, to some extend, formulae in natural language.

Many concepts and ideas are much easier to understand when they are depicted graphically; the inclusion of graphs and diagrams in addition to natural language is standard routine in mathematics communication. The computer allows us to go beyond these traditional ways of presentation and to include parameterized animations as well, which, for example, display how a diagram changes when parameters are varied. That is, a *multi-modal* user interface would be desirable.

The major advantage of a teacher in comparison to a textbook is that the students can interact with the teacher during the lesson. For example, they can ask the teacher when they do not understand a derivation. These forms of *interaction* should be supported by an intelligent explanation system as well.

To communicate a proof, the teacher has to present individual proof steps choosing a degree of *explicitness*. Usually, he does not mention all proof steps explicitly by giving the premises, the conclusion and the inference method. Often, he only hints implicitly at some proof steps (e.g., by giving only the inference method) when the hint is assumed to suffice for the student to reconstruct the proof step. Other proof steps are completely omitted when they are obvious or easy to infer.

However, a presentation that consists of a mere enumeration of proof steps is often unnatural and tedious to follow. Therefore, teachers add many *explanatory comments*, which motivate a step or explain the structure of a subproof, for example, by stressing that a case analysis follows.

Both decisions whether a proof step is only hinted at or omitted, and whether an explanatory comment is given are usually made *relative to the context*. For example, if a premise A of a proof step with conclusion B is used immediately after it was derived, the teacher only hints at it by saying: "Therefore, B holds." But if the premise A was derived a while ago he explicitly mentions it by saying: "Since A, we obtain B." A similar argument holds for explanatory comments. For example, if it is obvious that there is a case analysis it is not explicitly introduced, but the cases are presented right away.

The level of *abstraction* at which the proof is presented plays a major role. For example, the author of a textbook has an idea of his intended audience and adapts his presentation to that audience. Likewise, a teacher takes into account the abilities of his students when he decides at which level of abstraction he presents a derivation. To choose between different levels of abstraction, an intelligent proof explanation system needs a *user model*, which records the facts and inference methods the user knows.

Finally, a proof explanation system should also account for different *presentation strategies* with respect to the purpose of the session. For example, different strategies are needed when mere mathematical facts are to be conveyed in contrast to when mathematical skills are to be taught. In the former case, a proof can be presented in textbook style where the essential proof steps are shown. In the latter case, the presentation should be structured differently to convey also control knowledge, which explains *why* the various proof steps are taken as opposed to just show *that* they are taken [28]. These two strategies reflect the difference in the difficulty of checking the correctness of a proof versus finding a proof.

P.rex is the first attempt to build an interactive, user-adaptive proof explanation system. Except for multi-modality, we shall address all the requirements formulated in this paper. Although graphs and diagrams play an important role when included in a presentation, they occur only in certain mathematical theories and even in these theories, they do not occur very often. Therefore, we decided for the moment to neglect the generation of graphs and diagrams in our system.

3 The Architecture of *P.rex*

P.rex is a generic proof explanation system that can be connected to different theorem provers. Except for multi-modality, it fulfills all requirements given in Section 2.3. In this section, we shall give an overview of the architecture of *P.rex*, which is displayed in Figure 1.

As the interface between theorem provers and *P.rex*, we defined the formal language Twega, the implementation of an extension of the logical framework LF [17] that corresponds to the calculus of constructions [9]. Both are very powerful calculi from type theory that allow us to represent other logical calculi and, thus, to represent the proofs of most currently implemented theorem provers. Hence, Twega ensures that *P.rex* can be connected to these theorem provers. The type-theoretic foundations of Twega guarantee that only those proofs can be represented that are correct with respect to the calculus of the corresponding prover. The calculus of the prover, the input proof to be explained, as well as relevant information from the mathematical theories that relate to the proof are represented in Twega for further use by *P.rex*. An expansion mechanism that defines for any derivation step a subproof at the lower level of abstraction allows Twega to represent a proof at several levels of abstraction simultaneously.

The central component of *P.rex* is the *dialog planner*. It chooses the content and determines the order of the information to be conveyed, and organizes the pieces of information in a rhetorical structure, called *discourse structure*. The discourse structure also specifies the large-scale segmentation of the discourse into paragraphs.

The dialog planner is implemented in ACT-R, a production system that aims to model the human cognitive apparatus [1]. ACT-R separates declarative and procedural knowledge into a declarative memory and a production rule base,

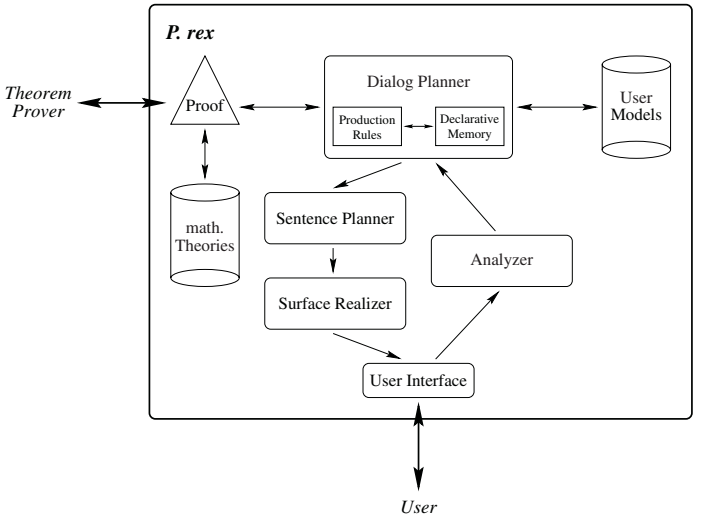

Fig. 1. The Architecture of *P.rex*.

respectively. A goal stack allows ACT-R to fulfill a task by dividing it into subtasks, which can be fulfilled independently. We shall review ACT-R briefly in Section 4.

The plan operators of the dialog planner organize the discourse structure. They are defined in terms of ACT-R productions. A discourse history is represented in the declarative memory by storing conveyed information. Moreover, presumed declarative and procedural knowledge of the user is encoded in the declarative memory and the production rule base, respectively. This establishes that the dialog planner is modeling the user.

In order to explain a particular proof, the dialog planner first assumes the user's cognitive state by updating its declarative and procedural memories. This is done by looking up the user's presumed knowledge in the user model, which was recorded during a previous session. An individual model for each user persists between the sessions. It is stored in the database of *user models*. Each user model contains assumptions about the knowledge of the user that is relevant to the proof explanation. In particular, it makes assumptions about which mathematical theories the user knows and which definitions, proofs, inference methods and mathematical facts he knows.

After updating the declarative and procedural memories, the dialog planner sets the global goal to show the proof. ACT-R tries to fulfill this goal by successively applying productions that decompose or fulfill goals. Thereby, the dialog planner not only produces a dialog plan, but also traces the user's cognitive states in the course of the explanation. This allows the system both to always choose an explanation adapted to the user, and to react to the user's interactions in a flexible way: The dialog planner interprets an interaction in terms of applications of productions. Then it plans an appropriate response. Section 5 is devoted to the dialog planner.

The dialog plan produced by the dialog planner is passed on to the *sentence planner*. We adapted and extended *PROVERB*'s micro-planner [15, 24] to use it in *P.rex* to plan the internal structure of the sentences. The sentence planner aggregates domain concepts when possible and maps them into a linguistic structure. The linguistic structure specifies the lexical items and referring expressions that realize the domain concepts as well as the small-scale segmentation, that is, the scope of the phrases and sentences.

The linguistic structure is then realized by the syntactic generator TAG-GEN [26], which ensures the correct morphology of the surface words. Note that dialog planner, sentence planner and surface realizer are organized in a pipeline.

The utterances that are produced by *P.rex* are presented to the user via a *user interface* that allows the user to enter remarks, requests and questions. An *analyzer* receives the user's interactions, analyzes them and passes them on to the dialog planner. Since natural language understanding is beyond the scope of this work, we use a simplistic analyzer that understands a small set of predefined interactions.

4 ACT-R: A Cognitive Architecture

In cognitive science several approaches are used to describe the functionality of the cognitive apparatus, for example, production systems, mental models or distributed neural representations. Production systems that model human cognition are called *cognitive architectures*. In this section we describe the cognitive architecture ACT-R [1], which is well suited for user adaptive explanation generation because of its conflict resolution mechanism. Further examples for cognitive architectures are SOAR [38] and EPIC [34].

ACT-R has two types of knowledge bases, or *memories*, to store permanent knowledge in: declarative and procedural representations of knowledge are explicitly separated into the declarative memory and the procedural production rule base, but are intimately connected.

Procedural knowledge is represented in production rules (or simply: *productions*) whose conditions and actions are defined in terms of declarative structures. A production can only apply if its conditions are satisfied by the knowledge currently available in the declarative memory. An item in the declarative memory is annotated with an activation that influences its retrieval. The application of a production modifies the declarative memory, or it results in an observable event. The set of applicable productions is called the *conflict set*. A *conflict resolution* heuristic derived from a rational analysis of human cognition determines which production in the conflict set will eventually be applied.

In order to allow for a goal-oriented behavior of the system, ACT-R manages goals in a goal stack. The current goal is that on the top of the stack. Only productions that match the current goal are applicable.

4.1 Declarative Knowledge

Declarative knowledge is represented in terms of *chunks* in the declarative memory. Below is an example for a chunk encoding the fact that $F \subseteq G$, where

`subset-fact` is a concept and `F` and `G` are contextual chunks associated to `factFsubsetG`.

```
factFsubsetG
  isa  subset-fact
  set1 F
  set2 G
```

Chunks are annotated with continuous activations that influence their retrieval. The activation A_i of a chunk C_i consists of its base-level activation B_i and the weighted activations of contextual chunks. In B_i, which is defined such that it decreases logarithmically when C_i is not used, ACT-R models the forgetting of declarative knowledge. Note that the definition of the activation establishes a spreading activation to adjacent chunks, but not further; multi-link-spread is not supported.

The constraint on the capacity of the human working memory is approached by defining a retrieval threshold τ, where only those chunks C_i can be matched whose activation A_i is higher than τ. Chunks with an activation less than τ are considered as forgotten.

New declarative knowledge is acquired when a new chunk is stored in the declarative memory, as is always the case when a goal is popped from the goal stack. The application of a production may also cause a new chunk to be stored if required by the production's action part.

4.2 Procedural Knowledge

The operational knowledge of ACT-R is formalized in terms of *productions*. Productions generally consist of a condition part and an action part, and can be applied if the condition part is fulfilled. In ACT-R both parts are defined in terms of chunk patterns. The condition is fulfilled if its first chunk pattern matches the current goal and the remaining chunk patterns match chunks in the declarative memory. An example for a production is

IF the current goal is to show that $x \in S_2$ and it is known that $x \in S_1$ and $S_1 \subseteq S_2$
THEN conclude that $x \in S_2$ by the definition of $\subseteq$

Similar to the base-level activation of chunks, the strength of a production is defined such that it decreases logarithmically when the production is not used. The time spent to match a production with a chunk depends on the activation of the chunk[1]. It is defined such that it is negative exponential to the sum of the activation of the chunk and the strength of the production. Hence, the higher the activation of the chunk and the strength of the production, the faster the production matches the chunk. Since the activation must be greater than the retrieval threshold τ, τ constrains the time maximally available to match a production with a chunk.

[1] In this context, *time* does not mean the CPU time needed to calculate the match, but the time a human would need for the match according to the cognitive model.

The conflict resolution heuristic starts from assumptions on the probability P that the application of the current production leads to the goal and on the costs C of achieving that goal by this means. Moreover G is the time maximally available to fulfill the goal. The net utility E of the application of a production is defined as

$$E = PG - C. \tag{1}$$

We do not go into detail on how P, G and C are calculated. For the purposes of this paper, it is sufficient to note that G only depends on the goal, but not on the production.

To sum up, in ACT-R the choice of a production to apply is as follows:

1. The conflict set is determined by testing the match of the productions with the current goal.
2. The production p with the highest utility is chosen.
3. The actual instantiation of p is determined via the activations of the corresponding chunks. If no instantiation is possible (because of τ), p is removed from the conflict set and the algorithm resumes in step 2, otherwise the instantiation of p is applied.

ACT-R also provides a learning mechanism, called *production compilation*, which allows for the learning of new productions.

5 Dialog Planning

In the community of NLG, there is a broad consensus that it is appropriate to generate natural language in three major steps [41, 6]. First, a *text planner* determines what to say, that is, content and order of the information to be conveyed. Then, a *sentence planner* determines how to say it, that is, it plans the scope and the internal structure of the sentences. Finally, a *linguistic realizer* produces the surface text. In this classification, the dialog planner is a text planner for managing dialogs.

The dialog planner of *P.rex* plans the dialog by building a representation of the structure of the discourse that includes speech acts as well as relations among them. The discourse structure is represented in the declarative memory. The plan operators are defined as productions.

5.1 Discourse Structure

Drawing on *Rhetorical Structure Theory (RST)* [30], Hovy argues in favor of a single tree to represent a discourse [21]. He considers a discourse as a structured collection of clauses, which are grouped into segments by their semantic relatedness. The discourse structure is expressed by the nesting of segments within each other according to specific relationships (i.e., RST relations). According to his discourse theory, each segment essentially contains the communicative goal the speaker wants to fulfill with this segment and either one to several discourse segments with intersegment discourse relations or the semantic material to be communicated.

Similarly to Hovy's approach, we describe a discourse by a *discourse structure tree*, where each node corresponds to a segment of the discourse. The speech acts, which correspond to minimal discourse segments, are represented in the leaves. We achieve a linearization of the speech acts by traversing the discourse structure tree depth-first from left to right. Explicit representation of the discourse purpose allows for presentations using different styles. Moreover, discourse structure trees also account for restricted types of dialogs as well, namely certain types of interruptions and clarification dialogs. This is a necessary prerequisite for the system to represent user interactions and to appropriately react to them.

5.2 Speech Acts

Speech acts are the primitive actions planned by the dialog planner. They represent frozen rhetorical relations between exchangeable semantic entities. The semantic entities are represented as arguments to the rhetorical relation in the speech act. Each speech act can always be realized by a single sentence. We use speech acts in *P.rex* not only to represent utterances that are produced by the system, but also to represent utterances from the user in the discourse.

We distinguish two major classes of speech acts. First, *mathematical communicative acts (MCAs)* are employed to convey mathematical concepts or derivations. MCAs suffice for those parts of the discourse, where the initiative is taken by the system. Second, *interpersonal communicative acts (ICAs)* serve the dialog, where both the system and the user alternately take over the active role.

Mathematical Communicative Acts. *Mathematical communicative acts (MCAs)* are speech acts that convey mathematical concepts or derivations. Our class of MCAs was originally derived from *PROVERB*'s PCAs [22], but has been substantially reorganized and extended. We distinguish two classes of MCAs:

- *Derivational MCAs* convey steps of the derivation, which are logically necessary. Failing to produce a derivational MCA makes the presentation logically incorrect. The following is an example for a derivational MCA given with a possible verbalization:

  `(Derive :Reasons (`$a \in F \vee a \in G$`)  :Conclusion` $a \in F \cup G$
  `:Method` $\cup$-Lemma`)`
  "Since $a \in F$ or $a \in G$, $a \in F \cup G$ by the $\cup$-Lemma."

- Explanatory MCAs comment on the steps of a derivation or give information about the structure of a derivation. This information is logically unnecessary, that is, omission leaves the derivation logically correct. However, inclusion of explanatory MCAs makes it much easier for the addressee to understand the derivations, since these comments keep him oriented. For example, an MCA of type `Case` introduces a case in a case analysis:

  `(Case :Number 1 :Hypothesis` $a \in F$`)`
  "Case 1: Let $a \in F$."

Interpersonal Communicative Acts. MCAs, which only convey information to the dialog partner without prompting any interaction, suffice to present mathematical facts and derivations in a monolog. To allow for dialogs we also need *interpersonal communicative acts (ICAs)*, which are employed for mixed-initiative, interpersonal communication. In our taxonomization we distinguish four classes of ICAs: questions, requests, acknowledgments and notifications. Note that the user never enters speech acts directly into the system. Instead, the user's utterances are interpreted by the analyzer and mapped into the corresponding speech acts.

ICAs are especially important to allow for clarification dialogs. If the system failed to successfully communicate a derivation, it starts a clarification dialog to detect the reason for the failure. Then, it can re-plan the previously failed part of the presentation and double-check that the user understood the derivation. We shall come back to this issue in Section 5.4.

5.3 Plan Operators

Operational knowledge concerning the presentation is encoded as productions. Each production either fulfills the current goal directly or splits it into subgoals. Let us assume that the following nodes are in the current proof:

Label	*Antecedent*	*Succedent*	*Justification*
P_1	Δ_1	$\vdash \varphi_1$	J_1
		$\vdots$	
P_n	Δ_n	$\vdash \varphi_n$	J_n
C	Γ	$\vdash \psi$	$R(P_1, \ldots, P_n)$

An example for a production is:

(P1) IF the current goal is to show $\Gamma \vdash \psi$
and R is the most abstract known rule justifying the current goal
and $\Delta_1 \vdash \varphi_1, \ldots, \Delta_n \vdash \varphi_n$ are known
THEN produce MCA
`(Derive :Reasons (`$\varphi_1, \ldots, \varphi_n$`) :Conclusion` ψ `:Method` R`)`
insert it in the discourse structure tree
and pop the current goal

By producing the MCA the current goal is fulfilled and can be popped from the goal stack. An example for a production decomposing the current goal into several subgoals is:

(P2) IF the current goal is to show $\Gamma \vdash \psi$
and R is the most abstract known rule justifying the current goal
and $\Phi = \{\varphi_i | \Delta_i \vdash \varphi_i \text{ is unknown for } 1 \leq i \leq n\} \neq \emptyset$
THEN for each $\varphi_i \in \Phi$, push the goal to show $\Delta_i \vdash \varphi_i$

Note that the conditions of (P1) and (P2) only differ in the knowledge of the premises φ_i for rule R. (P2) introduces the subgoals to prove the unknown premises in Φ. As soon as those are derived, (P1) can apply and derive the conclusion. Hence, (P1) and (P2) in principle suffice to plan the presentation

of a proof starting from the conclusion and traversing the proof tree towards its hypotheses. However, certain proof situations call for a special treatment. Assume that the following nodes are in the current proof:

Label	*Antecedent*	*Succedent*	*Justification*
P_0	Γ	$\vdash \varphi_1 \vee \varphi_2$	J_0
H_1	H_1	$\vdash \varphi_1$	HYP
P_1	Γ, H_1	$\vdash \psi$	J_1
H_2	H_2	$\vdash \varphi_2$	HYP
P_2	Γ, H_2	$\vdash \psi$	J_2
C	Γ	$\vdash \psi$	CASE(P_0, P_1, P_2)

A specific production managing such a case analysis is the following:

(P3) IF the current goal is to show $\Gamma \vdash \psi$
and CASE is the most abstract known rule justifying the current goal
and $\Gamma \vdash \varphi_1 \vee \varphi_2$ is known
and $\Gamma, H_1 \vdash \psi$ and $\Gamma, H_2 \vdash \psi$ are unknown
THEN push the goals to show $\Gamma, H_1 \vdash \psi$ and $\Gamma, H_2 \vdash \psi$
and produce MCA
`(Case-Analysis :Goal` ψ `:Cases (`φ_1, φ_2`))`
and insert it in the discourse structure tree

This production introduces new subgoals and motivates them by producing the MCA.

Since more specific rules treat common communicative standards used in mathematical presentations, they are assigned lower costs, that is, $C_{(\mathrm{P3})} < C_{(\mathrm{P2})}$ (cf. Equation 1 in Section 4.2).

Moreover, it is supposed that each user knows all natural deduction (ND) rules, since ND rules are the least abstract possible logical rules in proofs. Hence, for each production p that is defined such that its goal is justified by an ND rule in the proof, the probability P_p that the application of p leads to the goal to explain that proof step equals one. Therefore, since CASE is such an ND rule, $P_{(\mathrm{P3})} = 1$.

Note that the productions ensure that only those inference rules are selected for the explanation that are known to the user.

5.4 User Interaction

The ability for user interaction is an important feature of explanation systems. Moore and Swartout presented a context-sensitive explanation facility for expert systems that, on the one hand, allows the user to ask follow-up questions and, on the other hand, actively seeks feedback from the user to determine whether the explanations are satisfactory [37]. Mooney and colleagues emphasized that the user must be able to interrupt the explanation system at any time [36].

In *P.rex*, the user can interact with the system at any time. When the system is idle – for example, after starting it or after completion of an explanation – it waits for the user to tell it the next task. During an explanation, *P.rex* checks

after each production cycle if the user wishes to interrupt the current explanation. Each interaction is analyzed by the analyzer and passed on to the dialog planner as a speech act, which is included in the current discourse structure tree.

We allow for three types of user interaction in *P.rex*: A *command* tells the system to fulfill a certain task, such as explaining a proof. An *interruption* interrupts the system to inform it that an explanation is not satisfactory or that the user wants to insert a different task. In clarification dialogs, finally, the user is prompted to give *answers* to questions that *P.rex* asks when it cannot identify a unique task to fulfill. In this paper, we shall concentrate on interruptions.

Interruptions. The user can interrupt *P.rex* anytime to enter a new command or to complain about the current explanation. The following speech acts are examples for messages that can be used to interrupt the system:

`(too-detailed :Conclusion` C`)`
: The explanation of the step leading to C is too detailed, that is, the step should be explained at a more abstract level.

`(too-difficult :Conclusion` C`)`
: The explanation of the step leading to C is too difficult, that is, the step should be explained in more detail.

The Reaction to `too-detailed`. When the user complains that the derivation of a conclusion C was too detailed, the dialog planner checks if there is a higher level of abstraction on which C can be shown. If so, the corresponding higher level inference rule is marked as known, so that it is available for future explanations. Then, the explanation of the derivation of C is re-planned. Otherwise, the dialog planner informs the user, that there is no higher level available for presentation. This reaction of the system is encoded in the following two productions:

(P4) IF the user message is
`(too-detailed :Conclusion` C`)`
and the inference rule R was used to justify C
and there is an inference rule R' justifying C that is more abstract than R
THEN mark R' as known by the user
and push the goal to show C

(P5) IF the user message is
`(too-detailed :Conclusion` C`)`
and the inference rule R was used to justify C
and there is no inference rule R' justifying C that is more abstract than R
THEN produce ICA
`(Most-abstract-available :Rule` R`)`
and insert it in the discourse structure tree

An example dialog where the user complained that the original explanation of a proof was too detailed shall be given in Example 1 in Section 6.

The Reaction to `too-difficult`. When the user complains that the derivation of a conclusion C was too difficult, the dialog planner enters a clarification dialog to find out which part of the explanation failed to remedy this failure. The control of the behavior of the dialog planner is displayed in Figure 2. Note that every arrow in the figure corresponds to a production such that we cannot give the productions here due to space restrictions.

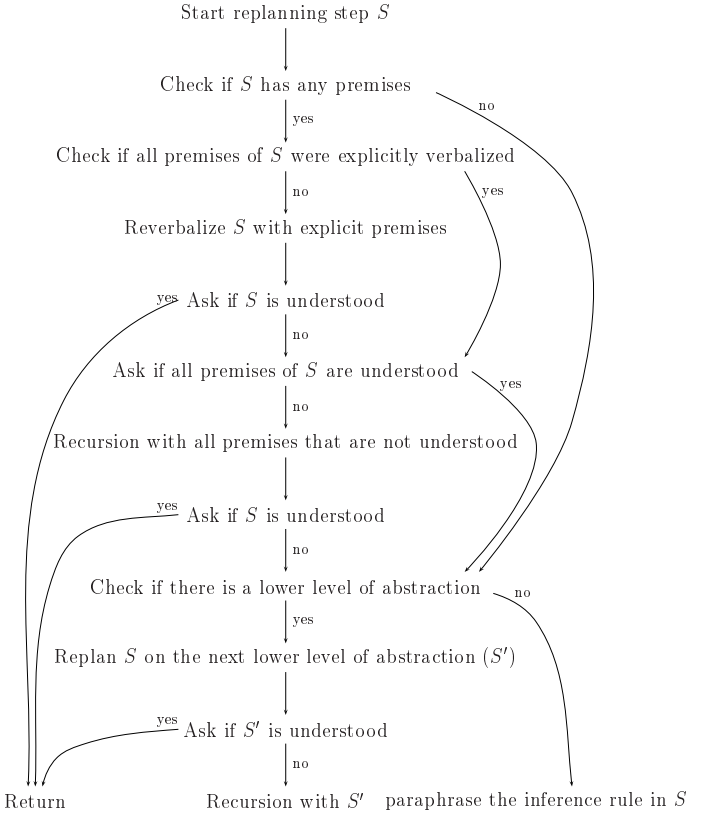

Fig. 2. The reaction of the dialog planner if a step S was too difficult.

To elucidate the diagram in Figure 2, an example dialog where the user complained that the original explanation of a proof was too difficult shall be given in Example 2 in the following section.

6 Example Dialogs

Let us examine more closely how *P.rex* plans the discourse with the help of two example dialogs. In both dialogs, *P.rex* explains the proof given in Figure 3. Note that this proof consists of two similarly proved parts with L_3 and L_7 as roots, respectively.

Label	*Antecedent*	*Succedent*	*Justification*
L_0		$\vdash a \in U \vee a \in V$	J_0
H_1	H_1	$\vdash a \in U$	HYP
L_1	H_1	$\vdash a \in U \cup V$	Def∪(H_1)
H_2	H_2	$\vdash a \in V$	HYP
L_2	H_2	$\vdash a \in U \cup V$	Def∪(H_2)
L_3		$\vdash a \in U \cup V$	∪-Lemma(L_0)
			CASE(L_0, L_1, L_2)
L_4		$\vdash a \in F \vee a \in G$	J_4
H_5	H_5	$\vdash a \in F$	HYP
L_5	H_5	$\vdash a \in F \cup G$	Def∪(H_5)
H_6	H_6	$\vdash a \in G$	HYP
L_6	H_6	$\vdash a \in F \cup G$	Def∪(H_6)
L_7		$\vdash a \in F \cup G$	∪-Lemma(L_4)
			CASE(L_4, L_5, L_6)

Fig. 3. A proof to be explained by *P.rex*.

Example 1. Let us consider the following situation:

- The current goal is to show the fact in L_3. The next goal on the stack is to show the fact in L_7.
- The rules HYP, CASE, and Def∪ are known, the rule ∪-Lemma is unknown.
- The facts in L_0 and L_4 are known, the facts in H_1, L_1, H_2, L_2, H_5, L_5, H_6, and L_6 are unknown.

Since CASE is the most abstract known rule justifying the current goal, both decomposing productions (P2) and (P3) are applicable. Recall that the conflict resolution mechanism chooses the production with the highest utility E (cf. Equation 1 in Section 4.2). Since $P_{(\mathrm{P3})} = 1$ and $P_p \leq 1$ for all productions p, $P_{(\mathrm{P3})} \geq P_{(\mathrm{P2})}$. Since the application of (P2) or (P3) would serve the same goal, $G_{(\mathrm{P3})} = G_{(\mathrm{P2})}$. Since (P3) is more specific than (P2), $C_{(\mathrm{P3})} < C_{(\mathrm{P2})}$. Thus

$$E_{(\mathrm{P3})} = P_{(\mathrm{P3})}G_{(\mathrm{P3})} - C_{(\mathrm{P3})} > P_{(\mathrm{P2})}G_{(\mathrm{P2})} - C_{(\mathrm{P2})} = E_{(\mathrm{P2})}$$

Therefore, the dialog planner chooses (P3) for the explanation, thus producing the MCA

$$\texttt{(Case-Analysis :Goal } a \in U \cup V \quad \texttt{:Cases } (a \in U, a \in V)\texttt{)}$$

which can be realized as "To prove $a \in U \cup V$ let us consider the cases that $a \in U$ and $a \in V$," and then explains both cases.

Suppose now that the user interrupts the system throwing in that the presentation is too detailed. Then, the analyzer passes the speech act `(too-detailed :Conclusion` $a \in U \cup V$`)` to the dialog planner. Since the inference rule ∪-Lemma, which is more abstract than CASE, also justifies the conclusion, production (P4) applies. Hence, the inference rule ∪-Lemma is marked as known and the goal to show the fact in L_3 is again pushed onto the goal stack. Then, (P1) is the only applicable production. Since ∪-Lemma is more abstract than CASE and both are known, it is chosen to instantiate (P1). Hence, the dialog planner produces the MCA

(Derive :Reasons ($a \in U \vee a \in V$) :Conclusion $a \in U \cup V$:Method ∪-Lemma)

which can be verbalized as "Since $a \in U$ or $a \in V$, $a \in U \cup V$ by the ∪-Lemma."

Since ∪-Lemma is now marked as known by the user, it can also be used for presentation in subsequent situations, for example, when L_7 is to be shown. The whole dialog takes place as follows:

***P.rex*:** In order to prove that $a \in U \cup V$ let us consider the following cases.
Case 1: Let $a \in U$. Then $a \in U \cup V$ by the definition of ∪.
Case 2: Let $a \in V$. That implies that $a \in U \cup V$ by the definition of ∪.
User: This derivation is too detailed.
***P.rex*:** Since $a \in U$ or $a \in V$, $a \in U \cup V$ by the ∪-Lemma. Since $a \in F$ or $a \in G$, $a \in F \cup G$ by the ∪-Lemma.

To elucidate the behavior of the dialog planner as depicted in Figure 2, let us examine the following example:

Example 2. We now consider the following situation:

- The current goal is to show the fact in L_3. The next goal on the stack is to show the fact in L_7.
- the rules HYP, CASE, Def∪, and ∪-Lemma are known,
- The facts in L_0 and L_4 are known, the facts in H_1, L_1, H_2, L_2, H_5, L_5, H_6, and L_6 are unknown.

This time, the only applicable production is (P1). Since ∪-Lemma is more abstract than CASE and both are known, it is chosen to instantiate (P1). Hence, the dialog planner produces the MCA

(Derive :Reasons ($a \in U \vee a \in V$) :Conclusion $a \in U \cup V$:Method ∪-Lemma)

which can be verbalized as "Since $a \in U$ or $a \in V$, $a \in U \cup V$ by the ∪-Lemma."

Suppose now that the user points to this utterance and interrupts *P.rex* throwing in that this step was too difficult. The analyzer translates the user's interaction to the speech act (too-difficult :Conclusion $a \in U \cup V$). Now, the dialog planner enters the clarification dialog as displayed in Figure 2. Since all premises were explicitly mentioned, the system does not reverbalize the step, but asks whether all premises are understood, what the user affirms. Hence, the system checks whether there is a lower level of abstraction, at which the step can be presented. Since this is the case, *P.rex* replans the explanation of the step by marking the inference rule ∪-Lemma as unknown and pushing the goal to show the fact in L_3 onto the goal stack. Now, (P1) is not applicable, since ∪-Lemma is unknown, but (P2) and (P3) are applicable. Hence, the system proceeds as in the beginning of Example 1 and verbalizes the case analysis. Since ∪-Lemma is unknown, it is not used in subsequent situations either. Therefore, L_7 is also explained by a case analysis. The whole dialog takes place as follows:

***P.rex*:** Since $a \in U$ or $a \in V$, $a \in U \cup V$ by the $\cup$-Lemma.
User: This step is too difficult.
***P.rex*:** Do you understand the premises?
User: Yes.
***P.rex*:** In order to prove that $a \in U \cup V$ let us consider the following cases.
Case 1: Let $a \in U$. That leads to $a \in U \cup V$ by the definition of $\cup$.
Case 2: Let $a \in V$. Then $a \in U \cup V$ by the definition of $\cup$.
Do you understand this step?
User: Yes.
***P.rex*:** In order to prove that $a \in F \cup G$ let us consider the following cases.
Case 1: Let $a \in F$. Therefore $a \in F \cup G$ by the definition of $\cup$.
Case 2: Let $a \in G$. Then $a \in F \cup G$ by the definition of $\cup$.

7 Conclusion

We reviewed research in proof presentation and presented the proof explanation system *P.rex*. Based on assumptions about the addressee's knowledge (e.g., which facts does he know, which definitions, lemmas, etc.), the dialog planner of *P.rex* chooses a degree of abstraction for each proof step to be explained. In reaction to the user's interactions, it enters clarification dialogs to revise its user model and to adapt the explanation. The architecture of the dialog planner can also be used to adapt content selection and explicitness reactively to the audience's needs. The rationale behind the architecture should prove to be useful for explanation systems in general.

However, in the current experimental stage, only a small set of user interactions is allowed. More elaborate interactions that call for more complex reactions are desirable. Therefore, empirical studies of teacher-student interactions in mathematics classes are necessary. Moreover, powerful natural language analysis and multi-modal generation is desirable.

P.rex is implemented in Allegro Common Lisp with CLOS and has been tested on dozens of input proofs. The implementation (currently Version 1.0) can be accessed via the *P.rex* homepage at `http://www.ags.uni-sb.de/~prex`.

Acknowledgments

Part of this project was funded by the Graduiertenkolleg Kognitionswissenschaft (doctoral program in cognitive Science) at Saarland University. I thank C. P. Wirth who read an earlier version of this work.

References

1. John R. Anderson and Christian Lebiere. *The Atomic Components of Thought.* Lawrence Erlbaum, 1998.
2. Peter B. Andrews. Transforming matings into natural deduction proofs. In *Proceedings of the 5th International Conference on Automated Deduction*, pages 281–292. Springer Verlag, 1980.

3. 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 McCune [31], pages 252–255.
4. Bruno Buchberger. Natural language proofs in nested cells representation. In J. Siekmann, F. Pfenning, and X. Huang, editors, *Proceedings of the First International Workshop on Proof Transformation and Presentation*, pages 15–16, Schloss Dagstuhl, Germany, 1997.
5. A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam System. In Mark Stickel, editor, *Proceedings of the 10th Conference on Automated Deduction*, number 449 in LNCS, pages 647–648, Kaiserslautern, Germany, 1990. Springer Verlag.
6. Lynne Cahill, Christy Doran, Roger Evans, Chris Mellish, Daniel Paiva, Mike Reape, Donia Scott, and Neil Tipper. In search of a reference architecture for NLG systems. In *Proceedings of the 7th European Workshop on Natural Language Generation*, pages 77–85, Toulouse, France, 1999.
7. Alison Cawsey. Generating explanatory discourse. In Robert Dale, Chris Mellish, and Michael Zock, editors, *Current Research in Natural Language Generation*, number 4 in Cognitive Science Series, pages 75–101. Academic Press, San Diego, CA, 1990.
8. Daniel Chester. The translation of formal proofs into English. *AI*, 7:178–216, 1976.
9. Thierry Coquand and Gérard Huet. The Calculus of Constructions. *Information and Computation*, 76(2/3):95–120, 1988.
10. Yann Coscoy, Gilles Kahn, and Laurent Théry. Extracting text from proofs. In Mariangiola Dezani-Ciancaglini and Gordon Plotkin, editors, *Typed Lambda Calculi and Applications*, number 902 in LNCS, pages 109–123. Springer Verlag, 1995.
11. B. I. Dahn, J. Gehne, Th. Honigmann, and A. Wolf. Integration of automated and interactive theorem proving in ILF. In McCune [31], pages 57–60.
12. Andrew Edgar and Francis Jeffry Pelletier. Natural language explanation of natural deduction proofs. In *Proceedings of the 1st Conference of the Pacific Association for Computational Linguistics*, Vancouver, Canada, 1993. Centre for Systems Science, Simon Fraser University.
13. Michael Elhadad and Jacques Robin. Controlling content realization with functional unification grammars. In Robert Dale, Eduard Hovy, Dietmar Rösner, and Oliviero Stock, editors, *Aspects of Automated Natural Language Generation*, number 587 in LNAI, pages 89–104. Springer Verlag, 1992.
14. Amy Felty and Dale Miller. Proof explanation and revision. Technical Report MC-CIS-88-17, University of Pennsylvania, Philadelphia, PA, 1988.
15. Armin Fiedler. Mikroplanungstechniken zur Präsentation mathematischer Beweise. Master's thesis, Computer Science Department, Universität des Saarlandes, Saarbrücken, Germany, 1996.
16. Gerhard Gentzen. Untersuchungen über das logische Schließen I & II. *Mathematische Zeitschrift*, 39:176–210, 572–595, 1935.
17. Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.
18. Amanda M. Holland-Minkley, Regina Barzilay, and Robert L. Constable. Verbalization of high-level formal proofs. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99) and Eleventh Innovative Application of Artificial Intelligence Conference (IAAI-99)*, pages 277–284. AAAI Press, 1999.

19. Helmut Horacek. A model for adapting explanations to the user's likely inferences. *User Modeling and User-Adapted Interaction*, 7:1–55, 1997.
20. Helmut Horacek. Presenting proofs in a human-oriented way. In Harald Ganzinger, editor, *Proceedings of the 16th Conference on Automated Deduction*, number 1632 in LNAI, pages 142–156. Springer Verlag, 1999.
21. Eduard H. Hovy. Automated discourse generation using discourse structure relations. *Artificial Intelligence*, 63:341–385, 1993.
22. Xiaorong Huang. *Human Oriented Proof Presentation: A Reconstructive Approach*. PhD thesis, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 1994.
23. Xiaorong Huang. Reconstructing proofs at the assertion level. In Alan Bundy, editor, *Proceedings of the 12th Conference on Automated Deduction*, number 814 in LNAI, pages 738–752, Nancy, France, 1994. Springer Verlag.
24. Xiaorong Huang and Armin Fiedler. Proof verbalization as an application of NLG. In Martha E. Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 965–970, Nagoya, Japan, 1997. Morgan Kaufmann.
25. INLG. *Proceedings of the 7th International Workshop on Natural Language Generation*, Kennebunkport, ME, 1994.
26. Anne Kilger and Wolfgang Finkler. Incremental generation for real–time applications. Research Report RR-95-11, DFKI, Saarbrücken, Germany, July 1995.
27. Peter Kursawe. Transformation eines Resolutionsbeweises: Der erste Schritt auf dem Weg zum natürlichen Schließen. Master's thesis, Fakultät für Informatik, Universität Karlsruhe, Karlsruhe, Germany, 1982.
28. Uri Leron. Structuring mathematical proofs. *The American Mathematical Monthly*, 90:174–185, 1983.
29. Christoph Lingenfelder. *Transformation and Structuring of Computer Generated Proofs*. PhD thesis, Universität Kaiserslautern, Kaiserslautern, Germany, 1990.
30. William C. Mann and Sandra A. Thompson. Rhetorical structure theory: A theory of text organization. ISI Reprint Series ISI/RS-87-190, Univerisity of Southern California, Information Science Institute, Marina del Rey, CA, 1987.
31. William McCune, editor. *Proceedings of the 14th Conference on Automated Deduction*, number 1249 in LNAI, Townsville, Australia, 1997. Springer Verlag.
32. David D. McDonald. Natural language generation as a computational problem. In Michael Brady and Robert C. Berwick, editors, *Computational Models of Discourse*. The M. I. T. Press, Cambridge, Massachusetts/London, 1984.
33. Andreas Meier. System description: TRAMP: Transformation of machine-found proofs into ND-proofs at the assertion level. In David McAllester, editor, *Automated Deduction – CADE-17*, number 1831 in LNAI, pages 460–464. Springer Verlag, 2000.
34. D. E. Meyer and D. E. Kieras. EPIC: A computational theory of executive cognitive processes and multiple-task performance: Part 1. Basic mechanisms. *Psychological Review*, 104:3–65, 1997.
35. Dale Miller. Expansion tree proofs and their conversion to natural deduction proofs. In R. E. Shostak, editor, *Proceedings of the 7th International Conference on Automated Deduction*, number 170 in LNCS, pages 375–303. Springer Verlag, 1984.
36. David J. Mooney, Sandra Carberry, and Kathleen McCoy. Capturing high-level structure of naturally occurring, extended explanations using bottom-up strategies. *Computational Intelligence*, 7:334–356, 1991.

37. Johanna D. Moore and William R. Swartout. A reactive approach to explanation: Taking the user's feedback into account. In Cécile L. Paris, William R. Swartout, and William C. Mann, editors, *Natural Language Generation in Artificial Intelligence*, pages 3–48, Boston, MA, USA, 1991. Kluwer.
38. A. Newell. *Unified Theories of Cognition.* Havard University Press, Cambridge, MA, 1990.
39. Cécile Paris. The role of the user's domain knowledge in generation. *Computational Intelligence*, 7:71–93, 1991.
40. F. Pfenning. *Proof Transformations in Higher-Order Logic.* PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, 1987.
41. Ehud Reiter. Has a consensus NL generation architecture appeared, and is it psycholinguistically plausible? In INLG [25], pages 163–170.
42. Wolfgang Wahlster, Elisabeth André, Wolfgang Finkler, Hans-Jürgen Profitlich, and Thomas Rist. Plan-based integration of natural language and graphics generation. *Artificial Intelligence*, 63:387–427, 1993.
43. Marilyn A. Walker and Owen Rambow. The role of cognitive modeling in achieving communicative intentions. In INLG [25], pages 171–180.
44. Ingrid Zukerman and R. McConachy. Generating concise discourse that addresses a user's inferences. In Ruzena Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1202–1207, Chambery, France, 1993. Morgan Kaufmann, San Mateo, CA.