

Dialog-driven Adaptation of Explanations of Proofs

Armin Fiedler

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

Abstract

In order to generate high quality explanations in mathematical domains, the presentation must be adapted to the knowledge of the intended audience. Most proof presentation systems only communicate proofs on a fixed degree of abstraction independently of the addressee's knowledge. In this paper, we shall present the proof explanation system *Prex*. Based on assumptions about the addressee's knowledge, its dialog planner chooses a degree of abstraction for each proof step to be explained. In reaction to the user's interactions, which are allowed at any time, it enters clarification dialogs to revise its user model and to adapt the explanation.

1 Introduction

A person who explains to another person a logical line of reasoning adapts his explanations to the addressee's knowledge. A computer program designed to take over the explaining part should also adopt this principle.

Assorted systems take into account the intended audience's knowledge in the generation of explanations (see e.g., [Cawsey, 1990; Paris, 1991; Wahlster *et al.*, 1993; Horacek, 1997]). Most of them adapt to the addressee by choosing between different discourse strategies. Feedback from the user himself can be an important source for user modeling. [Moore and Swartout, 1991] presents 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. [Mooney *et al.*, 1991] emphasizes that the user must be able to interrupt the explanation system at any time.

Whereas a mathematician communicates a proof on a level of abstraction that is tailored to the audience and reacts to the audience's needs, most proof presentation systems such as *PROVERB* [Huang and Fiedler, 1997] verbalize proofs on a fixed degree of abstraction given by the initial representation of the proof without allowing for user interaction.

Drawing on results from cognitive science, we have been developing an *interactive proof explanation system*, called *Prex* (for **proof explainer**), which adapts its explanations to the user's assumed expertise. The system explains each proof

step of a proof on the most abstract level that the user is assumed to know. While the explanation is in progress, the user can interrupt *Prex* anytime, when he is not satisfied by the current explanation. Analyzing the user's interaction and entering a clarification dialog when needed, the system tries to identify the reason why the explanation was not satisfactory and re-plans a better adapted explanation, for example, by switching to another level of abstraction.

Hence, driven by the dialog, *Prex* adapts its explanations to the user in reaction to his interactions. However, in the current experimental stage, only a small set of interactions is allowed.

In this paper, we shall first introduce proofs as the domain objects that are subject to explanation in Section 2. Next, in Section 3, we shall give an overview of the architecture of *Prex*. Then, in Section 4, we shall describe the dialog planner in more detail. Section 5 is devoted to example dialogs.

2 The Domain

The objects of our domain that are to be explained are proofs of mathematical theorems. An example for a proof is given below. Each line consists of four elements (label, antecedent, succedent, and justification) and describes a node of the proof. The *label* is used as a reference for the node. The *antecedent* is a list of labels denoting the hypotheses under which the formula in the node, the *succedent*, holds. This relation between antecedent and succedent is denoted by $\vdash$.

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 $\cup$ (H_1)
H_2	H_2	$\vdash a \in V$	HYP
L_2	H_2	$\vdash a \in U \cup V$	Def $\cup$ (H_2)
L_3		$\vdash a \in U \cup V$	$\cup$ -Lemma(L_0) CASE(L_0, L_1, L_2)

We call $\Delta \vdash \varphi$ the *fact* in the node. The proof of the fact in the node is given by its *justification*. A justification consists of a rule and a list of labels, the *premises* of the node. J_i denotes an unspecified justification. HYP and Def $\cup$ stand for a hypothesis and the definition of $\cup$, respectively. L_3 has two justifications on different levels of abstraction: the less abstract justification with the inference rule CASE (i.e., the rule for case analyses) and the more abstract justification with the

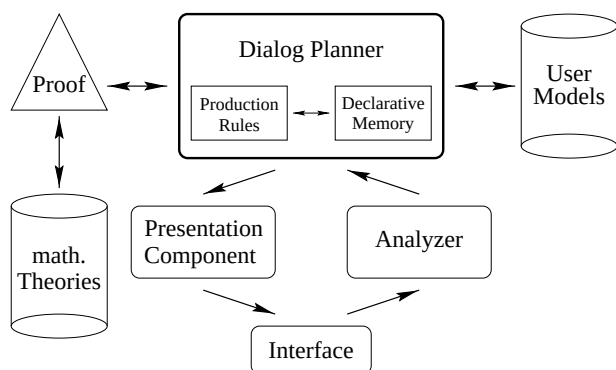


Figure 1: The Architecture of *P.rex*

inference rule $\cup$ -Lemma, which stands for an already proven lemma about a property of $\cup$. By agreement, if a node has more than one justification, these are sorted from most abstract to least abstract.

The proof is as follows: From $a \in U \vee a \in V$ we can conclude that $a \in U \cup V$ by the $\cup$ -Lemma. If we do not know the $\cup$ -Lemma, we can come to the conclusion by considering the case analysis with the cases that $a \in U$ or $a \in V$, respectively. In each case, we can derive that $a \in U \cup V$ by the definition of $\cup$.

3 The Architecture of *P.rex*

P.rex is a generic explanation system that can be connected to different theorem provers. An overview of its architecture is provided in Figure 1.

A formal language for specifying proofs and mathematical theories is the interface by which theorem provers can be connected to *P.rex*. **Mathematical theories** are organized in a hierarchical knowledge base. Each theory in it may contain axioms, definitions, theorems along with proofs, as well as proof methods, and control rules how to apply proof methods.

A **proof** of a theorem can be represented hierarchically such that it makes explicit the various levels of abstraction by providing several justifications for a single proof node, where each justification belongs to a different level of abstraction.

The central component of the system is the **dialog planner**. It is implemented in ACT-R [Anderson and Lebiere, 1998], a goal-directed production system that aims to model human cognition. 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.

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.

The plan operators of the dialog planner are defined in terms of productions and the discourse structure is represented in the declarative memory. Since presumed declarative and procedural knowledge of the user is encoded in the declarative memory and the production rule base, respectively, the dialog planner is modeling the user.

In order to explain a particular proof, the dialog planner first assumes the user's supposed cognitive state by updating its declarative and procedural memories, which were recorded during a previous session. An individual model for each user persists between the sessions. The individual user models are stored in the database of **user models**. Each user model contains assumptions on the knowledge of the user that are relevant to proof explanation. In particular, it makes assumptions on which mathematical theories the user knows, which definitions, proofs, proof methods and mathematical facts he knows, and which productions he has already learned.

After updating the declarative and procedural memories, the dialog planner sets the global goal to show the conclusion of the proof's theorem. 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 (cf. Section 4.1), 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 (cf. Section 4.3), and to react to the user's interactions in a flexible way: The dialog planner analyzes the interaction in terms of applications of productions. Then it plans an appropriate response (cf. Section 4.4).

The dialog plan produced by the dialog planner is passed on to the **presentation component**. Currently, we use a derivative of *PROVERB*'s micro-planner [Huang and Fiedler, 1997] to plan the scope and internal structure of the sentences, which are then realized by the syntactic generator TAG-GEN [Kilger and Finkler, 1995].

The uttered sentences are finally displayed on the **interface**, which also allows the user to enter remarks, requests and questions. An **analyzer** receives the user's interactions and passes them on to the dialog planner. In the current experimental stage, we use a simplistic analyzer that understands a small set of predefined interactions.

4 Discourse Planning

In the community of NLG, there is a broad consensus that it is appropriate to generate natural language in three major steps [Reiter, 1994; Cahill et al., 1999]. First, a *macro-planner* (*text planner*) determines what to say, that is, content and order of the information to be conveyed. Then, a *micro-planner* (*sentence planner*) determines how to say it, that is, it plans the scope and the internal structure of the sentences. Finally, a *realizer* (*surface generator*) produces the surface text. In this classification, the dialog planner is a macro-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.

4.1 Discourse Structure

Drawing on *Rhetorical Structure Theory (RST)* [Mann and Thompson, 1987], Hovy argues in favor of a single tree to represent a discourse [Hovy, 1993]. 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). 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.

4.2 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 [Huang, 1994], 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 4.4.

4.3 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	$\vdash \psi$	$R(P_1, \dots, P_n)$

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  $\Delta_1 \vdash \varphi_1, \dots, \Delta_n \vdash \varphi_n$  are known
THEN      produce MCA
           (Derive :Reasons ( $\varphi_1, \dots, \varphi_n$ )
            :Conclusion  $\psi$  :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  $\Phi = \{\varphi_i \mid \Delta_i \vdash \varphi_i \text{ is unknown for } 1 \leq i \leq n\} \neq \emptyset$ 
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 productions such as (P3) treat common communicative standards used in mathematical presentations, they are preferred to more general ones. Note that the productions ensure that only those inference rules are selected for the explanation that are known to the user.

4.4 User Interaction

The ability for user interaction is an important feature of explanation systems. [Moore and Swartout, 1991] presents 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. [Mooney *et al.*, 1991] emphasizes that the user must be able to interrupt the explanation system at any time.

In *Prex*, 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, *Prex* 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 *Prex*: 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 *Prex* asks when it cannot identify a unique task to fulfill. In this paper, we shall concentrate on interruptions.

Interruptions

The user can interrupt *Prex* 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 5.

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.

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.

5 Example Dialogs

Let us examine more closely how *Prex* plans the discourse with the help of two example dialogs. In both dialogs, *Prex*

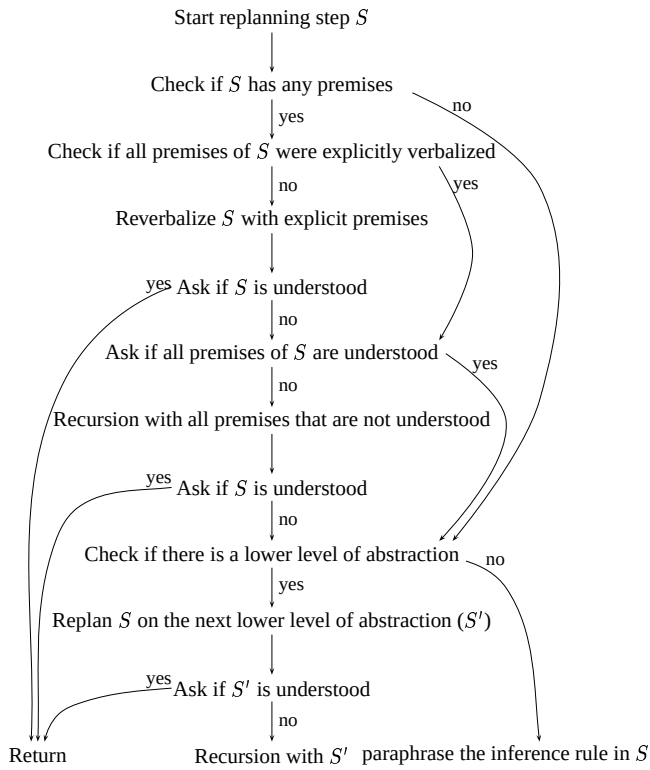


Figure 2: The reaction of the dialog planner if a step S was too difficult.

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.

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 DefU are known, the rule U-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 more specific productions are preferred to more general ones. Therefore, the dialog planner chooses (P3) for the explanation, thus producing the MCA

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

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 U-Lemma, which is more abstract than CASE, also justifies the conclusion, production (P4) applies. Hence, the inference rule U-Lemma is marked as known and the goal to show

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	$\vdash a \in U \cup V$	DefU(H_1)
H_2	H_2	$\vdash a \in V$	HYP
L_2	H_2	$\vdash a \in U \cup V$	DefU(H_2)
L_3		$\vdash a \in U \cup V$	U-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	$\vdash a \in F \cup G$	DefU(H_5)
H_6	H_6	$\vdash a \in G$	HYP
L_6	H_6	$\vdash a \in F \cup G$	DefU(H_6)
L_7		$\vdash a \in F \cup G$	U-Lemma(L_4)
			CASE(L_4, L_5, L_6)

Figure 3: A proof to be explained by *Prer*.

the fact in L_3 is again pushed onto the goal stack. Then, (P1) is the only applicable production. Since U-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 U-Lemma)

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

Since U-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:

Prer: 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 $\cup$.

Case 2: Let $a \in V$. That implies that $a \in U \cup V$ by the definition of $\cup$.

User: This derivation is too detailed.

Prer: Since $a \in U$ or $a \in V$, $a \in U \cup V$ by the U-Lemma. Since $a \in F$ or $a \in G$, $a \in F \cup G$ by the U-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, DefU, and U-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 U-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 U-Lemma)

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

Suppose now that the user points to this utterance and interrupts *P.r*ex 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 reverbitalize 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.r*ex replans the explanation of the step by marking the inference rule $\cup$ -Lemma as unknown and pushing the goal to show the fact in L_3 onto the goal stack. Now, (P1) is not applicable, since $\cup$ -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 $\cup$ -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 $\cup$.

□

6 Conclusion

In this paper, we presented the dialog planner of the proof explanation system *P.r*ex. Based on assumptions about the addressee’s knowledge (e.g., which facts does he know, which definitions, lemmas, etc.), the dialog planner 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.

References

- [Anderson and Lebiere, 1998] John R. Anderson and Christian Lebiere. *The Atomic Components of Thought*. Lawrence Erlbaum, 1998.
- [Cahill et al., 1999] 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.
- [Cawsey, 1990] 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.
- [Horacek, 1997] Helmut Horacek. A model for adapting explanations to the user’s likely inferences. *User Modeling and User-Adapted Interaction*, 7:1–55, 1997.
- [Hovy, 1993] Eduard H. Hovy. Automated discourse generation using discourse structure relations. *Artificial Intelligence*, 63:341–385, 1993.
- [Huang and Fiedler, 1997] 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.
- [Huang, 1994] Xiaorong Huang. *Human Oriented Proof Presentation: A Reconstructive Approach*. PhD thesis, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 1994.
- [Kilger and Finkler, 1995] Anne Kilger and Wolfgang Finkler. Incremental generation for real-time applications. Research Report RR-95-11, DFKI, Saarbrücken, Germany, July 1995.
- [Mann and Thompson, 1987] William C. Mann and Sandra A. Thompson. Rhetorical structure theory: A theory of text organization. ISI Reprint Series ISI/RS-87-190, University of Southern California/Information Science Institute, Marina del Rey, CA, 1987.
- [Mooney et al., 1991] 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.
- [Moore and Swartout, 1991] 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.
- [Paris, 1991] Cécile Paris. The role of the user’s domain knowledge in generation. *Computational Intelligence*, 7:71–93, 1991.
- [Reiter, 1994] Ehud Reiter. Has a consensus NL generation architecture appeared, and is it psycholinguistically plausible? In *Proceedings of the 7th International Workshop on Natural Language Generation*, pages 163–170, Kennebunkport, Maine, USA, 1994.
- [Wahlster et al., 1993] 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.