A Hybrid Model for Tutorial Dialogs

Helmut Horacek[1] and **Magdalena Wolska**[2]

[1]Fachrichtung Informatik [2]Fachrichtung Allgemeine Linguistik
Universität des Saarlandes, Postfach 15 11 50, D-66041 Saarbrücken, Germany
horacek@ags.uni-sb.de, magda@coli.uni-sb.de

Abstract

Until recently, rigid and sometimes cumbersome structures, which underly dialog patterns considered manageable for achieving a given task in a controlled manner, proved to be a serious weakness of interactive systems. Through the introduction of the information state as a representation to control the evolving state of a dialog, substantial improvements were obtained, with elaborations made for information-seeking and task-oriented dialogs. For handling tutorial dialogs, more rigid schemas are still in frequent use, due to the different requirements for this genre, which include more freedom on behalf of the human conversant due to limited pressure to understand a student's dialog contributions in full detail. In order to enable more flexible dialogs that also do justice to particularities of tutorial issues, we propose a mixed automaton- and information-state based model of dialogs. Capabilities of this model include elaborations to handle multiple task contributions in one turn, and abstractions from domain- and task-specific reasoning. A consequence of this design is the concentration on issues related to dialog proper, which increases the system's portability, a burning issue in the area of tutorial systems.

1 Introduction

Dialog patterns in human-computer interaction typically appear much more rigid than sequences in human conversation, and they may turn out even cumbersome in a number of situations. Through the introduction of the information state (IS) as a representation to control the evolving state of a dialog, this serious weakness is attacked with increasing success, which has been demonstrated in a number of elaborations. As opposed to information-seeking and task-oriented dialogs, more rigid schemas are still in frequent use for handling tutorial dialogs, due to the different requirements of this genre. By and large, tutorial dialogs are mostly driven by domain- and task-specific issues, such as providing hints in increasing degrees of detail, from an issue-related given repertoire. These schemas, however, somehow intermix dialog- and task-specific issues, which reduces modularity and therefore makes portability more difficult, a serious concern in the field of tutorial systems.

In order to enable participation in more flexible dialogs that also do justice to particularities of tutorial issues, we propose a mixed automaton- and IS-based model of dialogs. The underlying principles are implemented in the tool DiaWoZ that allows specifications of dialog structures, embedded in a system for carrying out Wizard-of-Oz (WoZ) experiments (Fiedler et al., 2004). In this paper, we present an instantiation of this model for dialog fragments from a corpus of simulated tutorial dialogs on proving mathematical theorems. Capabilities obtained include elaborations to handle multiple task contributions expressed in a single turn, and abstractions from domain- and task-specific reasoning, which are encapsulated in external system calls. A consequence of this design is the concentration on issues related to dialog proper, increasing portability.

This paper is organized as follows. First, we discuss differences between tutorial dialogs and task-oriented or information-seeking conversations. Then we describe the project scenario for our investigations. Next, we introduce the functionality of DiaWoZ. Then we present elaborations for dialogs in a tutorial session, in terms of information states. Finally, we present a mixed automaton- and IS-based model for an example dialog from our corpus.

2 Motivation

Information state representations increased the flexibility of dialog models. Elaborations have been made for typical situations in information-seeking and task-oriented dialogs, including grounding (Matheson et al., 2000), and negotiation (Larsson, 2002). Moreover, it has even been demonstrated that reasoning on cooperativity principles combined with pragmatic utterance interpretation enables a system to conduct natural dialogs (Sadek et al., 1997).

As opposed to that, tutorial systems mostly apply rigid schemata, driven by task and domain-specific given elaborations. Examples include Ms. Lindquist (Heffernan and Koedinger, 2000), where pseudo-natural language dialog is driven by well-elaborate tutorial strategies, Atlas-Andes (Freedman, 2000a), (Freedman, 2000b) which applies a plan-based mechanism uncovering hints in increasing degrees of detail, Autotutor (Person et al., 2000), where human-authored scripts encapsulate tutorial strategies and dialog interaction, and a geometry tutor (Popescu and Koedinger, 2000), which incorporates some specific strategies to address errors and flaws in the students use of terminology.

These examples give evidence for the divergence between the requirements in the contrasted categories of dialogs. The commonality among all these scenarios is the distribution of expertise: the system is ascribed full capabilities required for the issue at hand, while the user is ascribed at best limited knowledge. The difference among these genres lies in the way this divergence of expertise is treated. In information-seeking dialogs, all effort is concentrated on making the requested information available to the user. Similarly, all effort is invested in conveying necessary techniques and associated information to the apprentice. In contrast, the tutor's goal is to release as little information as possible, to enable students to uncover as much as possible about the solution to the given problem by themselves.

This divergence has consequences on specificities of dialogs: In task-oriented and information-seeking dialogs, grounding is of predominant importance, which increases the occurrence of clarification dialogs. Moreover, users should feel encouraged to make their utterances clear and well-structured. In tutorial dialogs, students are typically given more degrees of freedom to formulate their statements. While the systems may frequently not be able to understand these statements in full detail, they may still produce useful reactions that make progress towards the tutorial goal, such as hinting, and insisting on use of terminology.

3 Our Project Environment

Our work is part of the DIALOG project [1] (Benzmüller et al., 2003). Its goal is to (i) empirically investigate the use of flexible natural language dialog in tutoring mathematics, and (ii) develop an experimental prototype system gradually embodying the empirical findings. The system will conduct dialogs in written natural language to help a student understand and construct mathematical proofs. In contrast to most existing tutorial systems, we envision a modular design, making use of the powerful proof system ΩMEGA (Siekmann et al., 2002).

This design enables a detailed reasoning about the student's action and enables elaborate system responses. The scenario is illustrated in Fig. 1:

- *Learning Environment*: Students take an interactive course in the relevant subfield of mathematics within the web-based system ACTIVEMATH (Melis et al., 2001).
- *Mathematical Proof Assistant (MPA)*: Checks the appropriateness of user specified inference steps with respect to the problem-solving goal; based on ΩMEGA.
- *Proof Manager (PM)*: In the course of the tutoring session the user may explore alternative proofs. *PM* builds and maintains a representation of constructed proofs and communicates with the *MPA* to evaluate the appropriateness of the user's dialog contributions for the proof construction.
- *Dialog Manager*: The concern of this paper.
- *Knowledge Resources*: This includes *pedagogical knowledge* (teaching strategies), and *mathematical knowledge* (the domain expertirse).

Since there are virtually no dialogs about mathematical tutoring available, we have conducted a *WOz* experiment (Benzmüller et al., 2003b) with a simulated system in order to collect a corpus of this genre in the naive set theory domain. During the session, the subjects had to prove theorems about set complement and power set. The subjects were instructed to enter steps of a proof rather than a complete proof at once, in order to encourage dialog interaction with the system. The corpus showed that subjects made contributions in varying degrees of packaging, combining several proof steps, and even answers with further proof step specifications. We intend to handle such cases in our dialog model.

[1] The DIALOG project is part of and sponsored by the Collaborative Research Center on *Resource-Adaptive Cognitive Processes* (SFB 378) at Saarland University.

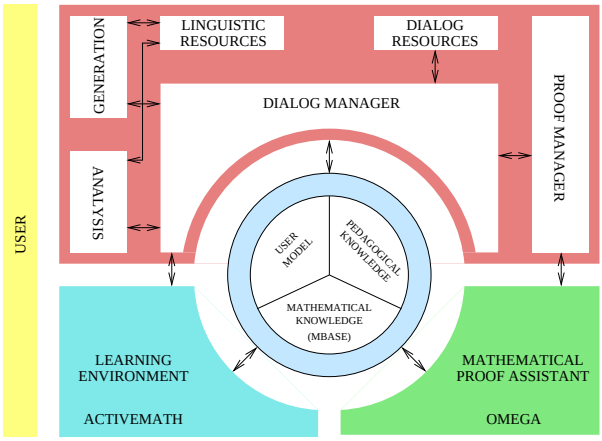

Figure 1: DIALOG project scenario

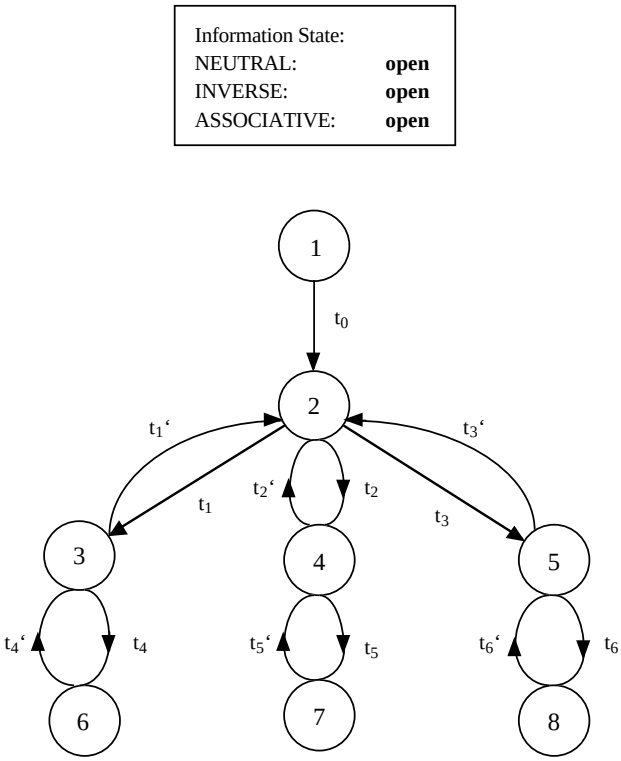

Figure 2: An example dialog specification

4 The Tool DiaWoZ

In order to meet the demands of modeling dialogs for tutorial systems, also in their development, we have designed and implemented DiaWoZ, a tool that enables setting up and executing of WOz experiments to collect dialog data. Its architecture is highly modular and allows for the progressive refinement of the experiments by both modeling increasingly sophisticated dialogs and successively replacing simulated components of the system by actual implementations, in particular, the dialog model itself.

The architecture of the entire tool is motivated and described in (Fiedler et al., 2004). In this section, we briefly introduce its dialog specification part, and we illustrate its functionality with a simple, constructed example.

4.1 The Dialog Specification

In DiaWoZ, a *dialog specification* is a finite state machine combined with an information state. The *finite state automaton* is defined by a set of states and a set of transitions between states. Furthermore, the dialog specification language allows one to define global variables, which are accessible from all states of the automaton, hence in the whole dialog. In addition, local variables can be defined for each state, whose scope comprises the corresponding subdialogs. The *information state* is conceived as the set of global and local variables accessible from the current state.

Going beyond other approaches, the transitions are associated with pre- and postconditions. The *preconditions*, whose fulfilment depends on processing in the node, are defined in terms of variables in the information state and restrict the set of applicable transitions for the current state dependent on the information state. The *postconditions* are effects that can both change the information state by setting its variables to different values and result in a function call, triggering an observable event such as an utterance. In particular, transitions can be parameterized in terms of the variables of the information state and the values to which they are changed in the transitions' effects.

As an example consider the following task from algebra: An algebraic structure $(S, \circ)$, where S is a set and $\circ$ an operator on S, should be classified. $(S, \circ)$ is a group if (i) there is a neutral element in S with respect to $\circ$, (ii) each element in S has an inverse element with respect to $\circ$, and (iii) $\circ$ is associative. In a tutorial dialog, the tutor must ensure, that the student addresses all three subtasks to conclude that a structure is a group. An appropriate dialog specification is given in Figure 2. The initial information state is displayed on top, while the finite-state automaton is shown below it. State 1 is the start state. In State 2, there are three transitions t_1, t_2, and t_3, which lead to parts of the automaton that represent subdialogs about the neutral element (States 3 and 6), the inverse elements (States 4 and 7), and associativity (States 5 and 8), respectively.

Hence, the dialog specifications in this simple model ensure that a started subtask is completed prior to addressing another subtask. However, the way how this subdialog is carried out is entirely left to the wizard. The information state consists of three global variables NEUTRAL, INVERSE, and ASSOCIATIVE capturing whether their corresponding values have been solved. The preconditions of the transitions are:

t_1: NEUTRAL = **open**
t_2: INVERSE = **open**
t_3: ASSOCIATIVE = **open**
The remaining transitions are always applicable.

These applicability conditions are defined in a rather free manner, to give the wizard maximum flexibility in deciding about the interpretation of the student's utterances. In addition, the wizard is free to apply further changes to the information state, if justified by that interpretation (see the overanswering in the example dialog in the next subsection). The effects of the transitions t'_1, t'_2, and t'_3 change the value of NEUTRAL, INVERSE, and ASSOCIATIVE, respectively, to **done**, indicating the successful completion of the associated subdialog. Moreover, each transition produces specifications for an utterance in the dialog. In an advanced system version with fairly elaborated analysis components, the interpretation of the student's utterances would be carried out directly and affect the information state.

4.2 An Example Dialog

To show how DiaWoZ works, let us come back to the example dialog specification given in Figure 2. It covers the following example dialog (where Z denotes the set of integers):

(U1) **Tutor:** To show that $(Z, +)$ is a group, we have to show that it has a neutral element, that each element in Z has an inverse, and that $+$ is associative in Z.
(U2) **Tutor:** What is the neutral element of Z with respect to $+$?
(U3) **Student:** 0 is the neutral element, and for each n in Z, $-n$ is the corresponding inverse.
(U4) **Tutor:** That leaves us to show associativity.

Let us now examine the dialog in detail. Starting in State 1, there is only one transition that can be picked, namely t_0. It leads to State 2, which produces specifications for utterance (U1). In State 2, all three transitions t_1, t_2, and t_3 can be picked, because their preconditions are fulfilled. Let us assume that t_1 is chosen, which leads to State 3 and provides specifications for the tutor's utterance (U2). Now, the student enters utterance (U3). Note that the student not only answers the tutor's question, but also gives the solution for the second subtask about the inverse elements. The next action depends on the interpretation of the student's utterance. For a wrong answer, transition t_4 would be pursued, while with the answer as given, transition t'_1 is the appropriate one. The utterance interpretation, which is done by the wizard, includes setting the variables NEUTRAL and INVERSE of the information state to **done**, by the effect of t'_1, which brings us back to State 2. Due to the over-answering, the tutor should not choose the subtask about the inverse elements in the next dialog turn. Hence, only transition t_3 is applicable, which provides specifications for the production of utterance (U4).

5 The Dialog Model

In the information-seeking dialog genre, the emphasis in dialog model design is on allowing to guide the dialog in such way that the dialog participants provide/obtain all the supplementary information needed to retrieve the required data. Dialog modeling techniques used here include finite-state automata (McTear, 1999). More complex IS-based dialog models have been used in task-oriented dialogs in which the focal point is on negotiating participants' problem solving goals (Matheson et al., 2000). In tutorial dialogs, especially in the context of socratic tutoring, the emphasis is placed on accounting for flexible adaptation in following the student's reasoning, while hinting at a correct solution when student's line of reasoning becomes inconsistent. As opposed to information-seeking or task-oriented dialogs, tutorial dialogs are characterized by an inherent bias in the knowledge states between the dialog participants: the student and the tutor. Moreover, student contributions to the discourse tend to be terminologically flawed, imprecise at the domain-level, erroneous in the formal parts of presentation, and may contain domain-specific misconceptions. A typical dialog scheme involves an attempt at solution followed by tutor's evaluation of the attempt, a hint at a correct solution, or a request for elaboration on an aspect of the contribution that was not clear.

In this section, we present an IS-inspired dialog model for tutorial dialogs that comprises a record representing the state of information at the given stage of the dialog, whose purpose is to guide dialog execution by the dialog automaton (cf. Section 6).

$$\begin{bmatrix} local : \begin{bmatrix} state\_id : \begin{bmatrix} var_1 : val_1 \\ \vdots \\ var_n : val_n \end{bmatrix} \end{bmatrix} \\ global : \begin{bmatrix} Given : Set(Prop) \\ LatestTurn : List(DM) \\ Proof - State : int \\ QUD : Stack(qud) \end{bmatrix} \end{bmatrix}$$

Figure 3: Excerpt of the IS record

5.1 Knowledge sources

The information incorporated into the Information State comes from the following sources:

Input understanding component The input understanding module stores a *discourse representation*, including a representation of the semantics of utterances in every turn. The discourse representation consists of a tree structure in which alternative readings of ambiguous statements, rhetorical relations, and dialog functions between adjacent segments are represented.

Proof Manager The Proof Manager provides information on the *state of the proof* at a given point in dialog and an *evaluation* of each of the alternative interpretations of a given proof contribution in the context of the proof constructed so far. The proof representation maintained by the proof manager is co-indexed with the discourse representation via labeling: utterances (or spans thereof) in the discourse model have direct counterparts in the constructed proof structure. This allows us to notice changes to the proof strategy indicated by the dialog.

Pedagogical resources The pedagogical resources used during dialog execution include the *tutorial strategy* and the *student model* dynamically updated in the course of the session. Additionally, the tutoring component updates information on *domain knowledge* attributed to the student.

At each dialog state, status information of the relevant IS components is computed and the IS is updated using information returned by the above knowledge sources. Below, we present the contents of the Information State.

5.2 Information State components

The IS record includes *global* and *local* variables that reflect the assumed knowledge states of the dialog participants, abstracting from details of the domain, that is, interior parts of proof statements are only represented when they play a role as discourse referents. An excerpt of the IS record is presented in Figure 3. Traditionally, it is divided into a *global* and *local* variables sub-records. In our model, values of *global* variables are accessible from all automaton states, while *local* variables are node-specific. In the context of a tutoring system, the crucial variable guiding the dialog at the meta-level is the status of the problem solution. In case of the theorem proving domain, this is the *proof state* at a given time (represented as an `integer` value). Every turn is represented in the IS in the Dialog Move `buffer`. Once the student starts constructing a proof (proof-state : in progress), what is *under discussion* is the correctness state of the intermediate contributions (QUD `stack`). *Local* variables only become relevant in specific points in the dialog, for instance, in clarification sub-dialogs (initiated at node N_3; see Figure 4). A local variable associated with the node N_3 will store the "issue" to be clarified. Below, we present more details on some of the IS components.

Proof-State Proof state is the status of proof development at the given stage of the dialog, represented as an `integer` value. At any time in the dialog the proof is in one of four states:

undefined: an initial state before the task is communicated to the student ($Proof - State = -1$),

communicated: the system offered a problem to the student, but the student has not yet started solving it ($Proof - State = 0$),

in progress: the student has contributed at least one attempt at the solution, but, according to the domain reasoner, the proof is not completed ($Proof - State = 1$), and finally

completed: the domain reasoner confirms that the proof is completed ($Proof - State = 2$).

Latest Turn Latest Turn is a pointer to the last turn in the discourse history maintained by Discourse Manager as well as the corresponding node in the proof structure maintained by the Proof Manager.[2]

Given "Given" is a `set` of propositions (domain facts) that are assumed to be known to both dialog participants. In particular, in case of tutorial dialogs, this includes information presented to the student

[2]In Figure 3, we make it explicit that a turn may comprise more than one utterance and so more than one Dialog Move (DM); represented here as a `list`. Each DM is addressed and processed by the model individually.

in the lesson material provided before the tutoring session. In the course of dialog, the set is gradually extended by completed proof steps.

QUD Question Under Discussion is a `stack` of currently pending moves: proof contributions that have not been evaluated by the tutor yet, tutor's (clarification) questions, and student's answers to tutor's questions. Hence, the information available through this stack is weaker than an agenda. In our environment, the role of the task agenda is performed by an interplay between the proof manager, which determines the subtasks required to solve the overall problem, and the pedagogical resources, which keep track of contributions to each of these subtasks. The dialog automaton is essentially driven by the current QUD and its attributes. For our purposes, we distinguish the following moves as relevant QUDs:

proof-statement (made by the user): an assert act that contains a *domain contribution* that is an attempt at a (part of) proof.

A proof-statement is characterized by its *correctness state* attribute delivered by the Proof Manager whose values may be: *correct* (the proof-statement is correct and brings the proof forward), *partially correct* (parts of the proof-statement correspond to a contextually relevant correct proof-step), *irrelevant* (the propositional content of the proof-statement is true, but the step does not bring the proof forward), *alternative correct* (the proof-statement has been recognized as a correct step, however, in a proof that has not been identified as the one that the student is trying to pursue), *wrong* (the propositional content of the proof-statement is false).

proof-statement evaluation (made by the system): evaluation of the proof contribution by the tutor in the context of the proof being constructed.

A given proof contribution is evaluated in one of the following categories: *accept*: the Proof Manager accepts the correct proof-step, *reject*: rejects a wrong step, or *pending*: evaluation is pending due to the proof-step being partially correct, irrelevant, or correct in an alternative proof (then a clarification dialog is issued first).

question: yes/no question, alternative question, or wh-question issued by the system[3]

answer: an assert act that is a relevant answer to a *question*.

unknown: none of the above,

address unknown: a move addressing the utterance classified as *unknown*.

[3]In the preliminary version of the system, we make a simplification in that only the system is allowed to ask questions. Questions on the part of the student are classified as *unknown*.

6 An Example

In this section, we illustrate the DiaWoz model for one of the dialogs from our corpus (see Figure 6). This dialog demonstrates some communication patterns that can be observed in tutorial dialogs with advanced students, including a change of strategy and multiple statements in one utterance. After the first correct and confirmed statement (2), the student specifies a proof step (4) that belongs to a solution path different from that initiated with the previous proof statement (2). Consequently, the system starts a clarification dialog to establish agreement about the solution path (5). Subsequently, the student not only provides the clarification required, but he also specifies a further proof step in the same utterance (6). The system response addresses both parts (7). Again, the next student dialog contribution specifies two proof steps in a single turn (8), which is answered in a similar fashion (9). The next user statement completes the proof (10), which is acknowledged and followed by a summary presentation (11).

In order to handle the driving forces underlying dialogs of this kind, the burden is split between the information state and an automaton which encapsulates more structurally oriented properties of the dialog. Specifically, the nodes of the automaton implicitly specify whose turn it is and what the status of the proof specification is. The latter may be the normal elaboration mode, comprising proof statements on behalf of the student and a system reaction, or some subdialog about one of the issues involved. In the example dialog, one such subdialog occurs, which is a meta-dialog about the proof state and associated clarifications. The links in the automaton express possible reactions alternatives, some of which are specific contributions, while there always needs to be a 'catch-all' response, in case a student dialog contribution cannot be interpreted in the context.

The fragment of states in the dialog automaton which cover the example session in Figure 6 is depicted in Figure 4. These states comprise an initial and an end state (N_0 and N_6, resp.), two states where user utterances are handled (N_1 and N_4), and three states where the system reaction is specified (N_2, N_3 and N_5). These states (on the left side in the Figure) represent provision of proof statements and the associated assessment. The remaining three states (on

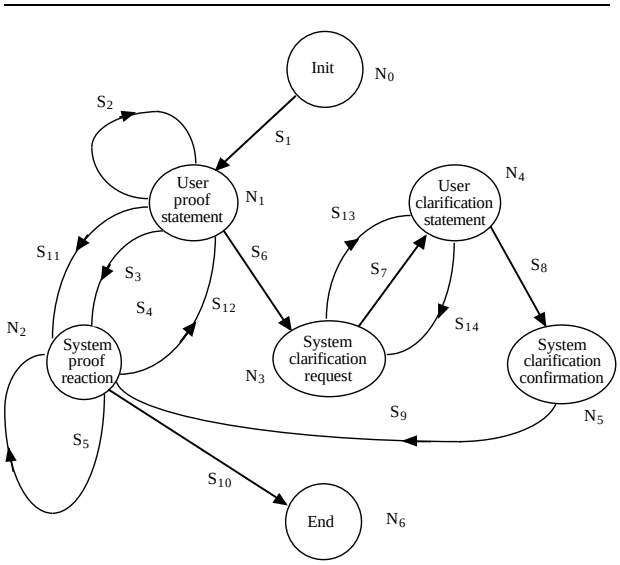

Figure 4: Automaton part of the dialog model for the example

$$\left[\begin{array}{l} local: \left[n_3 : \left[\begin{array}{l} query\_alternative\_proof : true \\ . \\ . \end{array} \right] \right] \\ global: \left[\begin{array}{l} Given : \{\ldots, F2\} \\ LatestTurn : \\ List(assert - meta(change), \\ domain - contribution \\ \qquad (split(powerset) \wedge \mathrm{F6})) \\ Proof - State : in - progress \\ QUD : Stack(evaluate(F6), \\ \qquad evaluate(F4), \ldots)) \end{array} \right] \end{array}\right]$$

Figure 5: Excerpt of the IS after student turn 6

the right side in the Figure) deal with the clarification dialog, where the second system reaction related node (N_5) handles specifics for the proof manager, prior to resuming with the standard reaction.

Interpretation of student utterances and contextually-based evaluation by the proof manager in case of a proof statement is done along the links S_2, S_3, S_6, S_8, S_{11}, and S_{14}. The interpretation is abstracted into dialog-relevant parts by the proof manager, which also updates the information state accordingly. When a move is accessed by the preconditions associated with a node, it is always the first move in the discourse representation that has not yet been accessed by the dialog processing model. Moreover, output generation is called on links originating from nodes which are associated with system turns, except S_5 and S_9, where reactions to multiple issues are collected. After system reactions of type confirm, the associated issue is popped from the QUD stack, and it is then moved to the *Given* set.

The remaining pre- and postconditions of the transitions associated with these nodes are illustrated in Figures 7 and 8, for states addressing user statements and system reactions, respectively. In addition to the conditions listed in this Figure, some common actions are carried out: (1) a pointer to the interpretation of the next user statement is pushed on the latest turn list in each user-related node, (2) generator specifications are cleared along each transition from a user- to a system-related node, and (3) the generator is called at each transition from a system- to a user-related node.

The two looping transitions, S_2 and S_5, serve the purpose of collecting multiple user statements expressed in one turn, resp. reacting on them according to their order. However, such a sequence of specifications must be coherent, which in our context is interpreted as being correct proof statements. After the clarification subdialog, a special transition, S_8, handles feedback to the proof manager, by means of an external system call. Note the resulting difference of interpretating utterance (4) in the example dialog, if the next response would have been "yes" instead of utterance (6). In this case, utterance (4) would have been assessed as "not relevant" for the solution path pursued, leading to another system reaction. In addition to the transition used for the dialog example, the illustrations also contain catch-all interpretations for user contributions, S_{11} and S_{13}, and the corresponding system responses S_{12} and S_{14}.

Figure 5 presents an excerpt of the Information State after the student utters turn 6. A solution attempt has been contributed by the student, hence the *Proof-status* variable has value *in-progress*. Because the proof manager identified the previous statement in turn 4 (formula F4) as valid in an alternative proof, the tutor issued a clarification request to verify this hypothesis (this is symbolically indicated by the local variable *query-alternative-proof*, associated with the clarification node, N_3, of the dialog automaton, taking a boolean value *true*). The formula in turn 6 (F6) is put on top of the QUD stack, with F4 still pending evaluation. Neither F4 nor F6 have been added to the *Given* set because their correctness status is still unresolved.

First proof step

(1) Tutor:

Bitte zeigen Sie: $A \cap B \in P((A \cup B) \cap (B \cup C))$!

Please show: $A \cap B \in P((A \cup B) \cap (B \cup C))$*!*

(2) Student:

Zerlegen der Potenzmenge: $P((A \cup C) \cap (B \cup C)) = P((A \cup C) \cap P(B \cup C))$

Partitioning the power set: $P((A \cup C) \cap (B \cup C)) = P((A \cup C) \cap P(B \cup C))$

(3) Tutor:

Das ist richtig!

That is correct!

A subdialog

(4) Student:

Anwenden der Distributivität: $P((A \cup C) \cap (B \cup C)) = P(C \cup (A \cap B))$

Application of distributivity: $P((A \cup C) \cap (B \cup C)) = P(C \cup (A \cap B))$

(5) Tutor:

Ist das noch derselbe Lösungsweg wie in der vorigen Antwort?

Is this still the same solution path as indicated in the previous answer?

(6) Student:

Nein, ich habe mich umentschieden: Ich zerlege jetzt die Potenzmenge:$P(C \cup (A \cap B)) \supseteq P(C) \cup P(A \cap B))$

No, I have changed my mind: Now I will partition the power set: $P(C \cup (A \cap B)) \supseteq P(C) \cup P(A \cap B))$

(7) Tutor:

Dann ist dieser wie auch der vorige Schritt richtig!

Then this step is correct, and the previous one too

Multiple proof statements

(8) Student:

Ich schätze die Vereinigung der Teilmenge ab: $P(C \cup (A \cap B)) \supseteq P(A \cap B) \supseteq \{A \cap B\}$

I estimate the union of the subset: $P(C \cup (A \cap B)) \supseteq P(A \cap B) \supseteq \{A \cap B\}$

(9) Tutor:

Diese beiden Schritte sind richtig!

These two steps are correct!

(10) Student:

$A \cap B \in \{A \cap B\}$

$A \cap B \in \{A \cap B\}$

(11) Tutor:

Das ist auch richtig und vollendet den Beweis. Ich wiederhole noch einmal:. . . <proof summary>

This is also correct and completes the proof. I repeat once again: . . .<proof summary>

Figure 6: An example dialog from the corpus (the predicate P stands for power set)

7 Conclusion

In this paper, we have proposed a hybrid dialog model combining a finite state automaton with information state representations. The model mediates between the rigid automata-based models mostly used in tutorial systems and information state based models used for task-oriented and information-seeking dialogs. While this model still exhibits a good deal of the flexibility information state based approaches have to offer, the structural commitments encapsulated in the automaton part makes control and maintainability of the model easier. Specifically, development of the dialog model is supported by the incorporation in a WoZ tool.

Here, we have presented by-hand elaborations for handling multiple moves within one dialog contribution, and making external calls to domain knowledge sources, which are required for meeting properties of tutorial dialogs. In the future, we intend to extend this model by increasing the variety of the interplay with external knowledge sources, for example, making dialog continuations in part dependent on assessments of tutorial strategies, e.g., by interpretating student utterances containing minor mistakes "cooperatively" by correcting these mistakes, or by insisting on precision in such cases.

References

C. Benzmüller, A. Fiedler, M. Gabsdil, H. Horacek, I. Kruijff-Korbayová, M. Pinkal, J. Siekmann, D. Tsovaltzi, B. Vo, and M. Wolska. Tutorial dialogs on mathematical proofs. In *Proceedings of the IJCAI Workshop on Knowledge Representation and Automated Reasoning for E-Learning Systems*, pages 12–22, Acapulco, 2003.

C. Benzmüller, A. Fiedler, M. Gabsdil, H. Horacek, I. Kruijff-Korbayová, M. Pinkal, J. Siekmann, D. Tsovaltzi, B. Vo, and M. Wolska. A Wizard-of-Oz experiment for tutorial dialogues in mathematics. In V. Aleven, U. Hoppe, J. Kay, R. Mizoguchi, H. Pain, F. Verdejo, and K. Yacef, editors, *AIED2003 Supplementary Proceedings*, volume VIII: Advanced Technologies for Mathematics Education, pages 471–481, Sydney, 2003.

Reva Freedman. Plan-Based Dialogue Management in a Physics Tutor. In *Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP 2000)*. Seattle, WA. 2000.

Reva Freedman. Using a Reactive Planner as the Basis for a Dialogue Agent. In *Proceedings of the Thirteenth Florida Artificial Intelligence Research Symposium (FLAIRS 2000)*. Orlando, FL. 2000.

A. Fiedler, M. Gabsdil and H. Horacek. A Tool for Supporting Progressive Refinement of Wizard-of-Oz Experiments in Natural Language. In *Intelligent Tutoring Systems - 7th International Conference (ITS 2004)*, number 3220 in LNCS, pp. 325-335, Springer, 2004.

N. Heffernan and K. Koedinger. Intelligent tutoring systems are missing the tutor: Building a more strategic dialog-based tutor. In *Papers from the 2000 AAAI Fall Symposium on Building Dialogue Systems for Tutorial Applications*, pages 14–19, North Falmouth, MA, AAAI Press, 2000.

S. Larsson. Issue-Based Dialogue Management. PhD thesis, Gothemburg University, 2002.

Colin Matheson, Massimo Poesio, and David Traum. Modelling Grounding and Discourse Obligations Using Update Rules. Proceedings of the 1st Annual Meeting of the North American Association for Computational Linguistics (NAACL2000). 2000.

Michael McTear. 1999. Software to support research and development of spoken dialogue systems. Proceedings of Eurospeech 99, 339–342, Budapest, Hungary, 1999.

E. Melis, E. Andres, A. Franke, G. Goguadze, M. Kohlhase, P. Libbrecht, M. Pollet, and C. Ullrich. A generic and adaptive web-based learning environment. In *Artificial Intelligence and Education*, pages 385–407, 2001.

Natalie K. Person, Arthur C. Graesser, Derek Harter, Eric Mathews, and the Tutoring Research Group. Dialog move generation and conversation management in AutoTutor. In Carolyn Penstein Rosé and Reva Freedman, editors, *Building Dialog Systems for Tutorial Applications—Papers from the AAAI Fall Symposium*, pages 45–51, North Falmouth, MA, AAAI press, 2000.

Popescu, O. and Koedinger, K. (2000). Towards understanding geometry explanations. In *Building Dialogue Systems for Tutorial Applications, Papers from the 2000 AAAI Fall Symposium*, pages 80–86, Menlo Park, California. AAAI Press.

D. Sadek, P. Bretier and F. Panaget. ARTIMIS: Natural Dialogue Meets Rational Agency. In *15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1030–1035, Nagoya, Japan, 1997.

J. Siekmann, C. Benzmüller, V. Brezhnev, L. Cheikhrouhou, A. Fiedler, A. Franke, H. Horacek, M. Kohlhase, A. Meier, E. Melis, M. Moschner, I. Normann, M. Pollet, V. Sorge, C. Ullrich, C-P Wirth, and J. Zimmer. Proof development with ΩMEGA. In A. Voronkov, editor, *Automated Deduction — CADE-18*, number 2392 in LNAI, pages 144–149. Springer Verlag, 2002.

David Traum and Staffan Larsson. The Information State Approach to Dialogue Management. *Current and New Directions in Discourse and Dialogue*. Kluwer. 2003.

S_2:
Preconditions:
utterance-type(move) = proof-statement
correctness-state(move) = correct
buffer not empty
Postconditions:
proof-state = in-progress
push(QUD, move)

S_3:
Preconditions:
utterance-type(move) = proof-statement
correctness-state(move) not alternative-correct
buffer empty
Postconditions:
proof-state = in-progress
push(QUD, move)
invert(QUD)

S_6:
Preconditions:
utterance-type(move) = proof-statement
correctness-state(move) = alternative-correct
buffer empty
Postconditions:
same as (postconditions S_3)

S_8:
Preconditions:
utterance-type(move) = answer
utterance-content(move) = remain or change
Postconditions:
if utterance-content(move) = change
then correctness-state(move) = correct
call(proof-manager,
correctness-state(move) = correct)

S_{11}:
Preconditions:
not ((Preconditions S_2) or (Preconditions S_3))
Postconditions:
if utterance-type(move) = proof-statement
then push(QUD, move)

S_{14}:
Preconditions:
not (Preconditions S_8)
Postconditions:
none

Figure 7: Pre- and postconditions for user nodes

S_1:
Preconditions:
proof-state = undefined
Postconditions:
proof-state = communicated
QUD = [describe(proof-task)]

S_4:
Preconditions:
QUD = <single move>
Postconditions:
push(QUD, proof-statement-evaluation(move))

S_5:
Preconditions:
QUD not <single move>
correctness-state(move) = correct
Postconditions:
push(QUD, proof-statement-evaluation(move))

S_7:
Preconditions:
utterance-type(move) = proof-statement
correctness-state(move) = alternative-correct
Postconditions:
push(QUD,alternative-question)

S_9:
Preconditions:
none
Postconditions:
pop(proof-stack, move)
push(QUD, proof-statement-evaluation(move))

S_{10}:
Preconditions:
proof-state = complete
Postconditions:
pop(QUD, move)
push(QUD, proof-statement-evaluation(move))
push(QUD, assess(proof))

S_{12}:
Preconditions:
proof-stack empty
Postconditions:
pop(QUD, move)
push(QUD, proof-statement-evaluation(move))

S_{13}:
Preconditions:
move not answer
Postconditions:
push(QUD, address-unknown)

Figure 8: Pre- and postconditions for system nodes