Artificial Intelligence Review **14:** 153–179, 2000.
Stephen J. Hegner et al. (eds.), Intelligent Help Systems for UNIX.

The SINIX Consultant – Towards a Theoretical Treatment of Plan Recognition

MATTHIAS HECKING
Marienstrasse 20, D-56112 Lahnstein, Germany
(E-mail: matthias.hecking@bigfoot.com)

Abstract. We have realized the help system SINIX Consultant (SC) for SINIX[1] users. The system is capable of answering – in German – natural language questions concerning SINIX commands, objects, and concepts. But not only does this help system react to inquiries – additionally, the system is capable of activating itself. If the user employs a sequence of SINIX commands (a plan) in order to reach a specific goal, the help system proposes a sequence which reaches the same goal, but, with fewer commands. In this paper, a brief survey of the SINIX Consultant and the realized plan recognizer REPLIX is first given. Then, an initial attempt of a theoretical treatment of plan recognition is presented. This is done within the logical framework. We show how we can use an interval-based logic of time to describe actions, atomic plans, non-atomic plans, action execution, and simple plan recognition. We also show that the recognition of inserted sub-plans managed by REPLIX can be handled as well. Then, we present a problem which cannot be treated in the formalism. Thus, in this paper, we don't present a full developed theory but nevertheless, a step towards it.

1. Introduction

Within the SINIX Consultant (SC) project, we have realized a plan recognizer called REPLIX. This integrated component is able to scrutinize the commands the user typed in and can detect the goal persued by these commands. Beside the program code and various descriptions in natural language, we don't have an exact representation of the semantics in the field of plan recognition. In order to obtain this, an *appropriate mathematical apparatus* has to be employed.

We assume that *symbolic logic* is such an apparatus (some arguments for such an assumption can be found in [Genesereth and Nilsson 1987, Preface]).

Instead, in a first approach, we used attribute grammars (cf. Hecking and Harbusch 1987). These were dropped because logic presents a formal treatment of the meaning (extensional semantics).

Furthermore, we assume that, in general, symbolic logic provides a mathematical basis for AI theory. We do not claim that first-order predicate logic is expressive enough for all problems in AI, although it can be used quite

successfully (see Allen's theory of action and time; cf. Allen 1984). There are though two reasons to employ symbolic logic. First, logic is an old research topic with a lot of well established results (e.g. proof theory or model theory). We can use these results. Second, there are other logics (e.g. logic of knowledge and belief) which were invented to overcome the limitations of first-order predicate logic. These logics are already used to develop theories in AI (see *the deduction model of belief* from K. Konolige; cf. Konolige 1986).

Using these assumptions, we describe a step towards such a logical based theory of plan recognition. Thus, in this paper we do not present a fully developed theory.

This report is structured as follows:

1. In section 2, the help systems SC is described, especially the difference between its active and passive capabilities. The structure of SC, the interaction of its components, and the state of implementation follow.
2. In section 3, the REPLIX and the Advisor components are described in more detail.
3. In section 4 we try to identify those concepts in command-oriented plan recognition which should be modelled in a more theoretical manner.
4. In section 5, the first step towards a model is presented. The model is formulated within the logical framework. We describe actions, atomic and non-atomic plans, action execution, and simple plan recognition. We show that we can formulate the concept of *inserted plans* (which can be handled by REPLIX too) and we mention a problem which cannot be handled in this initial step.

2. Survey of the SINIX Consultant

At the University of Saarbrücken, the help system SINIX Consultant (SC) has been in the progress of development since summer 1985. One of the main goals of the project is to integrate *passive* and *active* capabilities. The demands on the system are:

- to answer natural language questions about concepts, commands, and objects of the operating system SINIX in German,
- to conduct a cooperative dialogue,
- to infer the goal out of closely examined commands in order to give voluntary advice on 'better' plans.

Next, we explain the differences between active and passive help systems.

2.1. *Passive help systems*

If the user seeks information about a concept or a command, the usual way in obtaining it is by either consulting the manual (in simple cases) or by asking a qualified colleague, i.e. posing questions in natural language. The user may need information about those concepts and commands which he rarely uses, or he may ask for advice pertaining to those concepts which are not part of the system. In these cases, the system may respond appropriately. The system delivers the information or simply states, that this is not a known concept (precondition: the help system covers every concept and command of the underlying system).

If a help system *reacts* only to inquiries, we define it as being *passive*. If the inquiries can be posed in natural language (other possibilities are menus, help keys etc.), the natural language interface (NLI) can exhibit a greater or lesser degree of sophistication. The NLI may by only able to handle simple, syntactically correct questions. More elaborate NLIs may process elliptical questions, syntactically incorrect inquires (robustness), or even manage dialogues.

There are several other passive help systems:

- the Unix Consultant (UC) (cf. Wilensky et al. 1984, 1986),
- the AQUA system (cf. Quilici et al. 1986).

2.2. *Active help systems*

Those concepts and commands which are completely unknown to the user also remain unknown in a passive help system. The user can be made aware of these unknown concepts only if the system is able to become *active* by itself. A precondition for a help system in becoming active is the ability to identify possible applications of these unknown concepts and commands. One attempt to realize this, is that the system must *infer the goal* of the user's actions and must find a more 'appropriate' way to reach the same goal.

For example, in order to infer the underlying goals of the user's non-verbal activities, we have a system which *scrutinizes the commands* the user types in. The ones used here are:

```
mv letter1 invoices[2]
mv letter2 invoices
```

The system infers the possible goal 'move more than one file to a directory'. In this example, a more 'appropriate' way is that of a sequence of actions through which the user can reach the same goal with fewer keystrokes (there are other possible metrics). After detecting a more efficient way to reach the goal, the system becomes active, and proposes to the user:

You can reach the same goal with:
mv letter1 letter2 invoices

In this example, the user is made aware of the fact that the 'mv' command can be used with multiple files. Completely new concepts can also be introduced in this manner.

The first active help system was WIZARD (cf. Finin 1983); another, the AKTIVIST (cf. Schwab 1984), is a help system for a screen-oriented editor.

2.3. *The structure of the system*

A short introduction into the system structure and its state of implementation follows. For a more detailed description of the implemented system cf. Hecking et al. 1988, Hecking 1987 and Kemke 1987.

The natural language input (cf. Figure 1) and the command input is passed on the *Filter*, which decides which type of input (NL or command) the user typed in.

After this decision, the *NL Input* is transfered to the morphological analyzer *MORPHIX* (cf. Finkler and Neumann 1986). The *deflected NL Input* is the input of the *Parser*. The parser works in a two-level, pattern oriented fashion and is based on ideas of various DYPAR versions (cf. Boggs et al. 1984). The semantic representation of natural language input is based on case frames. The case frame is handed to the *Question Evaluator* which determines the type of questions, accesses the *Domain Knowledge Base* in order to get the desired information, and updates the *user model SC-UM* in order to record the individual level of experience of the user. Depending on the user model, the system delivers a more, or less detailed explanation. The desired information is represented in an answer frame which is passed on to the *NL Generator*.

If the Filter detects that a command was given, the *Command Input* is passed on to the *Plan Recognizer REPLIX*. The plan recognizer detects accomplished plans and interactions (inserted sub-plans, overlapping plans, ignore and interrupt commands) between the plans. The output of the plan recognizer is passed on to the *Advisor*. First, the Advisor tries to find a better plan with which the user can reach the same goal by using fewer commands. Next, it asks the user model to deliver all unknown concepts and commands of this plan. In a third step, the information out of step one and two is represented in an *Advice Frame*, which is also passed on to the NL Generator.

The NL Generator constructs the natural language answer out of the Answer Frame and the Advice Frame. This answer is then presented to the user.

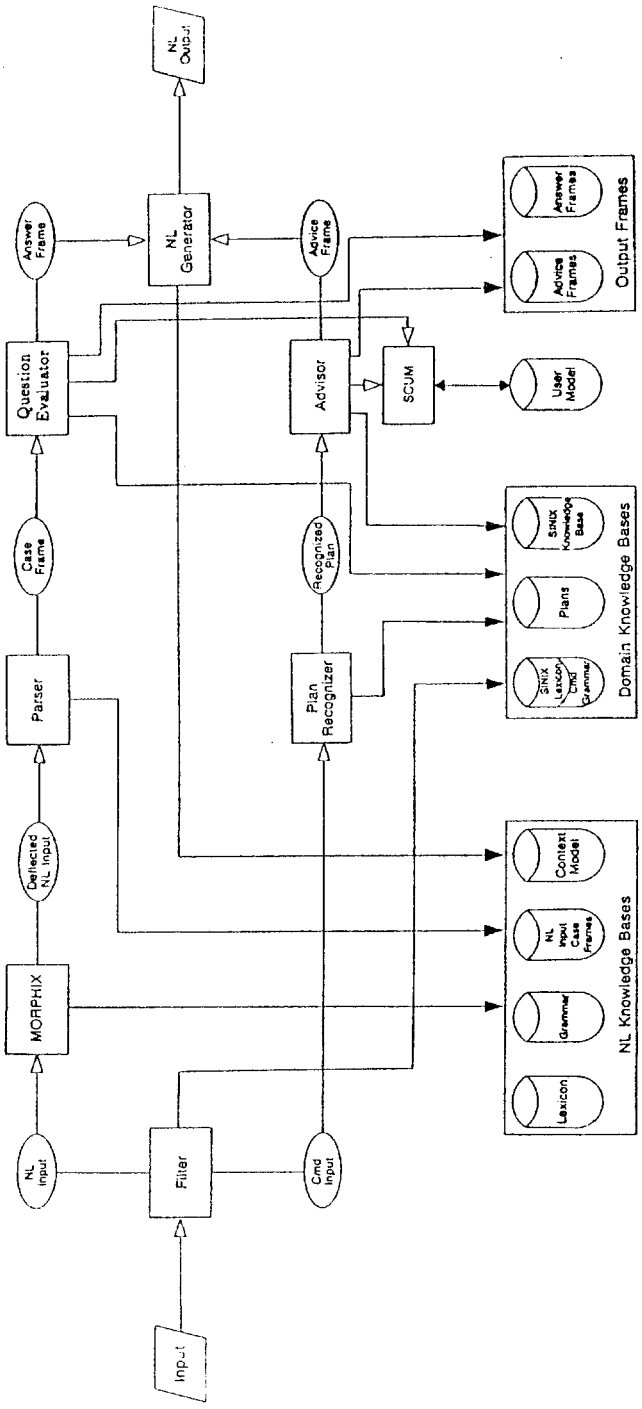
NL Output
NL Generator
Answer Frame
Advice Frame
Output Frames
Answer Frames
Advice Frames
Question Evaluator
Advisor
SCUM
User Model
Case Frame
Recognized Plan
Domain Knowledge Bases
SINIX Knowledge Base
Plans
SINIX Lexicon Cmd Grammar
Parser
Plan Recognizer
Deflected NL Input
NL Knowledge Bases
Context Model
NL Input Case Frames
Grammar
Lexicon
MORPHIX
NL Input
Filter
Cmd Input
Input

Figure 1.

2.4. *The state of implementation*

To date, a first version of the SC has been implemented. The modules were integrated into the system and successfully tested. The knowledge base describes 90 commands and 300 concepts pertaining to the following sections of SINIX:

- information about the system state,
- information about directory and file management,
- simple process communication.

Questions with regard to these commands and concepts can be posed. The plan recognizer works with a library of 40 suboptimal/optimal plan pairs. MORPHIX, the parser, the Question Evaluator, the user model, the plan recognizer REPLIX, and the Advisor have been realized. The global control structure works agenda-based. The lexicon contains about 2000 words.

A real natural language generator together with the possibility of conducting dialogues has not yet been implemented. The natural language answers are based on patterns assembled according to the user's expertise. A case-frame based parser is being developed.

The system is implemented on a Siemens APS 5815 (XEROX 1108–105) in INTERLISP-D and LOOPS. The operating system SINIX is simulated through the SINIX Emulator.

The next steps in the development of the SC are:

- the extension of the system in order to allow cooperative dialogues,
- the realization of a natural language generator,
- the realization of a plan generator.

3. Plan Recognition in SC

3.1. *The plan recognition component REPLIX*

In this section, the plan recognizer REPLIX[3] is described. For a more comprehensive treatment, the reader should refer to Dengler et al. 1987 or Hecking 1987.

The goal of this presentation is to illustrate the capabilities of the realized plan recognizer in order to provide justification for the theoretical treatment. Within the theoretical discussion, the capabilities of REPLIX should be modelled as to clarify any problems arising concerning semantics.

The plan recognition process is divided into two different phases:

1. the actual plan recognition (which is done by the REPLIX components; see Figure 1), i.e. the detection of the pursued plan and its associated goal and

2. the phase where the information about the detected plan is used to produce better plans (this is done by the Advisor; see Figure 1).

The component REPLIX was designed, implemented, and successfully integrated into the SC in order to be able to infer underlying goals of non-verbal user actions, i.e. the commands he types in. The main demands on the REPLIX component are:

- The possibility of adapting the plan recognizer to other operating systems. This is realized though an abstract syntax which differentiates between command words, flags, and objects. In addition, the wild-cards and special characters (e.g. the beginning sign of the flags) of the employed operating system can be specified.
- The realization of a special plan editor in order to facilitate the construction of the plans.
- The realization of collections of plans (plan packages) and interrupt commands for the focussing of the recognition process.
- The recognition of not only simply sequences of plans, but also inserted sub-plans and overlapping plans.
- The appropriate handling of commands which are not part of any plan (ignore commands).
- The realization of a graphical representation of the internal state of the plan recognizer.

The plans which should be recognized are formulated through a special syntax (unnecessary details are left out):

[(NameOfPlan (GoalString))
(CmdWord_1 Flags_1 Specification_1 ObjectList_1)
(CmdWord_2 Flags_2 Specification_2 ObjectList_2)
...
(CmdWord_n Flags_n Specification_n ObjectList_n)
(ListOfIgnoreCommands)
(ListOfInterruptCommands)]

For convenient reference, each plan is given a unique name. Any possible string represents the goal associated to the plan. Through this string, the adaption of the plan recognizer to the surrounding system (in the SC the Advisor) is realized. The *command word* (e.g. ‘mv’, ‘mkdir’), *flags* (e.g. ‘-la’, ‘-f’), a *specification*, used for complex flags, e.g. in order to determine the position of the flags after the command word, and a *list of objects* (e.g. the name of the directory used by the ‘mv’ command) can be specified with respect to each command word.

In the object list, three different types of parameters can be specified:

- the name of a *fixed object*, i.e. the command is successfully recognized only if the actual name is used,
- the name of a *schematic variable* (any name preceded by an ';'), i.e. the command can be used with any parameter. This construction is needed to allow for the possibility of different commands to use the same parameter.
- the name of a *set variable* (any name preceded by an '!') in order to specify the possibility of using a list of parameters.

The use of REPLIX is explained in the following example.

Example 1: The following plan containing four commands pursued the goal **delete a directory together with its content**:

1. cd ;dir
2. rm *
3. cd . .
4. rmdir ;dir

The first command is used to change the current directory into the specified directory. The corresponding name given to this command is stored in the schematic variable **;dir**. With the second command, all files in this directory are removed. The third command changes the current directory back to the parent directory ('. .' is the fixed parameter for the parent directory of the current directory). The last command deletes the empty directory.

In command 1 and 4, the same schematic variable is used. This mechanism allows the plan only to recognized if, in both commands, the same actual parameter is used. The schematic variables are means in order to realize *context-sensitivity within plans*.

If the user types in the following sequence of commands:

1. cd letters
2. rm *
3. cd . .
4. rmdir letters

REPLIX recognizes plan no. 1 and delivers the following information to the Advisor:

(*4* (DeleteDir (delete a directory and its content))
((cd letters) (;dir (letters)))
((rm *) NIL)
((cd . .) NIL)
((rmdir letters) (;dir (letters))))

After the fourth command (*4* is an absolute numeration of the user input), the completion of the plan named **DeleteDir** was recognized and reported to the Advisor. The name of the plan, the goal, the commands, the

employed parameters (eg. letters), and the mapping between the schematic variable and the actual parameter (eg. (;dir (letters))) are also passed on to the Advisor.

REPLIX is able to detect the completion of two plans in sequence.

In order to insert another plan, a pursued plan is often interrupted after a command. The first plan is resumed after completing the inserted plan.

Now, suppose a second plan exists together with the goal **print two files**:

1. lpr ;file1
2. lpr ;file2

Suppose, the employed sequence of commands is as follows:

1. cd letters
2. lpr juliane
3. lpr sabine
4. rm *
5. cd . .
6. rmdir letters

then, plan no. 1 is interrupted. After the first command, plan no. 2 is inserted (command no. 2 and 3) and using command no. 4, plan no. 1 is resumed. REPLIX detects this insertion.

Besides the completion of the two plans, the insertion of **PrintFiles** in ***DeleteDir*** – after the initial command of **DeleteDir** – is reported as well.

The branch counter allows for the possibility of specifying the number of levels of insertion.

If the last commands of a plan are identical to the first commands of the following plan, REPLIX is able to recognize such overlapping.

Note that REPLIX is able to recognize each combination of insertions and overlappings interspersed with ignore commands.

As already mentioned, the plans which should be recognized are grouped together to form a plan package. A maximum of 20 different plans are contained within a plan package. The use of a specific plan package depends on which section of the operating system the user works in, e.g. the user can work in the section of file and directory management, or in the section of information pertaining to the state of the system.

3.2. *The advisor*

At the moment, we only use a part of the information that REPLIX delivers. The plan packages we use specify employed sub-optimal plans. If e.g. plan no. 1 was recognized, the described information is passed on to the Advisor. The Advisor tries to determine a more optimal plan in the plan library which will also reach the goal of **delete a directory together with its content**. If this attempt is successful, the Advisor determines those commands pertaining

to the optimal plan which, through the user model, are likely to be unknown to the user. The Advisor then formulates an answer which describes the unknown commands within the optimal plan as well as the optimal plan itself along with its appropriate inserted actual parameters. Then, the SC becomes active and presents this information to the user. In the end, the recognized plan, the newly learned commands of the optimal plan, and the optimal plan itself are all notified within the user model.

4. Towards a General Deductive Model of Plan Recognition

The experiences gained from designing and implementing the plan recognizer REPLIX build the foundation for the development of a theory of plan recognition. Beside various descriptions in natural language we do not have an exact presentation of the semantics in plan recognition. As mentioned in chapter 1, we use *symbolic logic* as the framework to reach such an exact description.

In the next chapter we first try to identify certain goals which should be reached through the formal model of plan recognition. Then, we give a survey of how plan recognition proceeds, which different levels of plan recognition processes and objects must be modelled.

4.1. *The goals of plan recognition*

If we try to realize a theory of plan recognition we are faced with the question of, apart from domain dependant goals, identifying the general goals of plan recognition.

The following questions at least must be answered:

- Which goals are persued by the user?
- Which sequences of actions (plans) are used to reach each goal?
- Which parameters are used in each action?
- Which temporal relationships exists between different goals?
- Can we order the set of goals, so that the most likely goal forms the top in a goal hierarchy?
- Which means can be used to resolve goal conflicts?

4.2. *How plan recognition proceeds*

Figure 2 shows all the necessary steps involved in plan/goal recognition for command-driven systems.

In the *world model*, the relevant aspects of our domain are represented. Because in most domains time plays a crucial role, the representation must

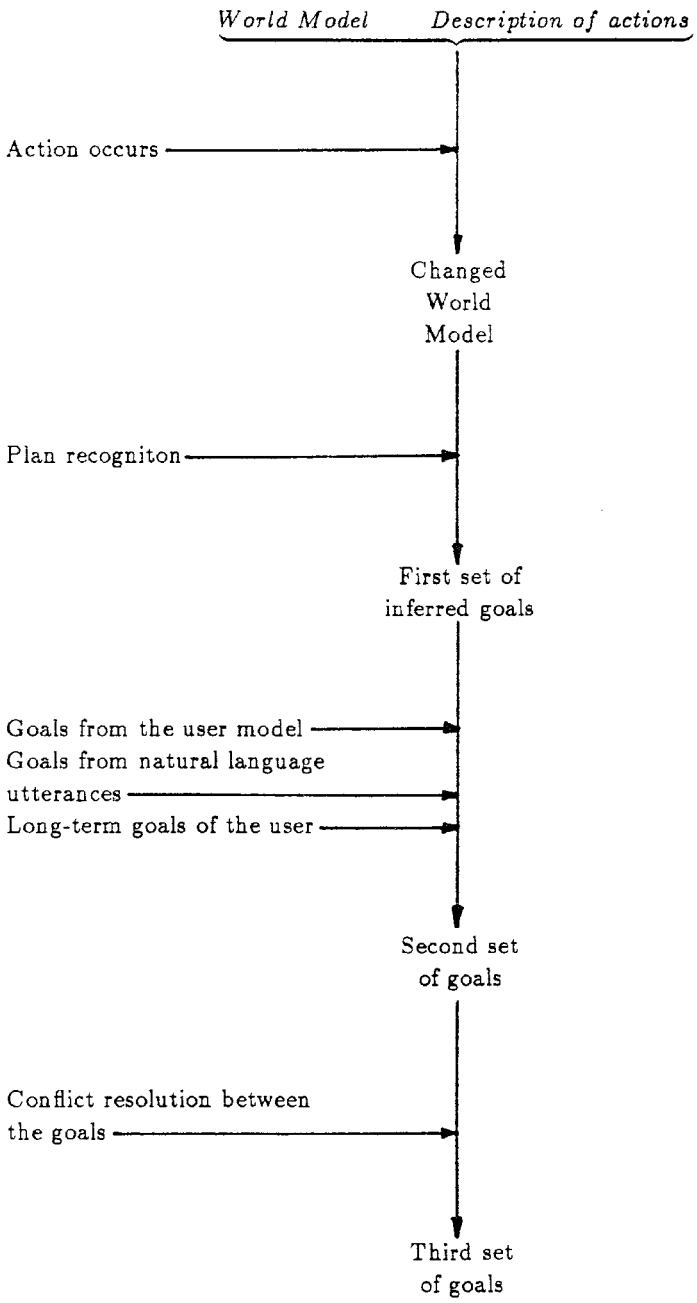
World Model
Description of actions
Action occurs
Changed World Model
Plan recogniton
First set of inferred goals
Goals from the user model
Goals from natural language utterances
Long-term goals of the user
Second set of goals
Conflict resolution between the goals
Third set of goals

Figure 2.

be chronologically oriented. If actions (e.g. commands) are used, the preconditions and the effects of those actions must be described. This is given in the *description of actions*. If an *action occurs*, the *world model is changed*. The commands used are the foundation for *plan recognition*. Therefore, we receive a *first set of inferred goals*. Since these goals are grounded solely on the commands used, we only get a rather restricted set of goals. Therefore, we must take into account other goals, e.g. goals *stored in the user model*, or *long-term goals of the user*, or if possible, the *goals which could be inferred from natural language utterances*. If we take these goals into consideration, we attain a *second set of goals*. Because there are competing goals *conflict resolution* is necessary. Thus, we reach a *third set of goals* in which one goal or a small set of goals are assumed to be the primary goal of the user.

4.3. *What should be modelled?*

From different steps in the process of plan recognition and from experiences with REPLIX, we come up with the following objects and processes of plan recognition which must be modelled if we want to realize the formal model:

- time dependent *properties* of the domain,
- the *time structure*,
- there are *commands*,
- the command has various *parameters* and *flags*,
- there are *preconditions* for using a command,
- the command has *effects* on the world model,
- the *application of a command*,
- the *recognition* of a used command (simple action recognition),
- there are *goals*,
- a goal can be reached with *different plans*,
- how commands and a goal are put together to form an *atomic plan*,
- the *recognition of the goal* of an atomic plan,
- how commands, sub-goals, and a goal are put together to form an *non-atomic plan*,
- the *recognition of the goal* of an non-atomic plan,
- which *temporal relationships* are possible between the application of commands and used sub-goals, the recognition of these commands and sub-goals, and the recognized goal of an atomic or non-atomic plan.

4.4. *Interval-based logic of time*

Since we are mainly interested in modelling inserted sub-plans, a time logic must be used in our deductive model. A very common approach in AI is the time logic of J. Allen (cf. Allen 1984, Allen 1983). In his general theory of

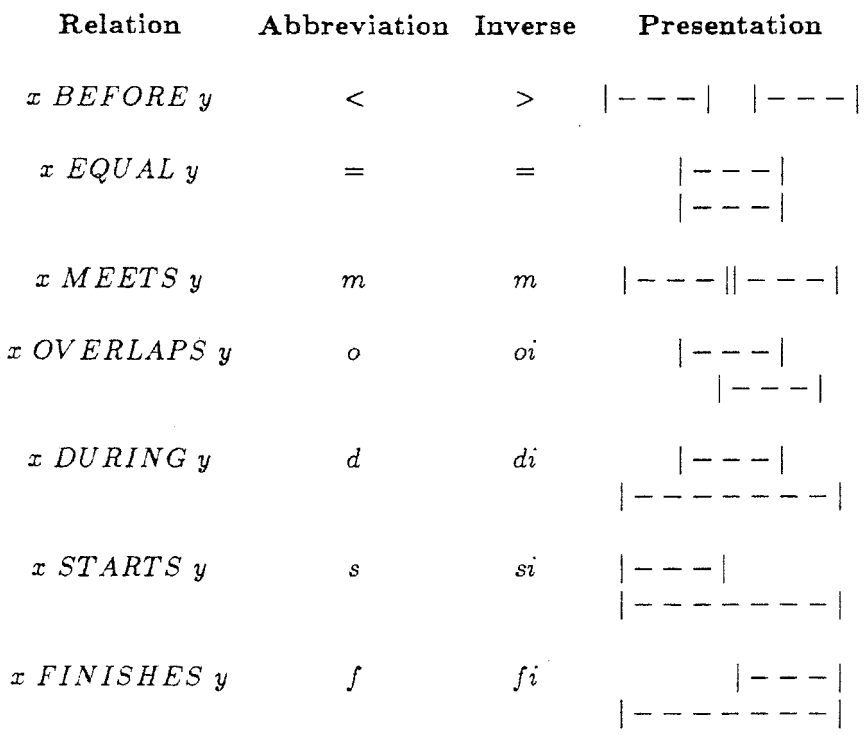

Figure 3.

action and time, an interval-based time logic is used. Actually, it is not a time logic because there are no specific temporal operators, like e.g. in the theory of A. Prior (cf. Prior 1957). Still, we adhere to this definition.

In this interval-based logic, 13 relations exists between two intervals, e.g. interval T1 can be **BEFORE** interval T2 or T1 can be **DURING** T2. The relations and their abbreviations are depicted in Figure 3. If we use these relations in our model, we change freely between the abbreviations and the regular names. We use ($T1 < = m\ T2$) to express (T1 < T2) ∧ (T1 = T2) ∧ (T1 *m* T2).

5. The Deductive Model

In this chapter, we try to *partially formalize* the world model of our domain, the actions, the plans, and the plan recognition which is performed by REPLIX (see section 3.1). Especially the problem of inserted sub-plans is modelled. Our main goal is to identify the limits of our deductive model. Starting with the interval-based time logic, we describe how properties in

our domain are represented in the world model. Then we describe several commands, atomic plans, and a non-atomic plan.

Starting with a given time structure, we show how the use of the commands change the world model and how plans are used for plan recognition.

Finally, we mention what can not be modelled and compare our approach to another formal theory of plan recognition.

5.1. *The model of the world*

Properties of the modelled plan recognition domain are time-dependent. They are valid in an interval. The fact that a property **p** holds in an interval **t** is expressed through the predicate **HOLDS (p,t)**. For example, with

HOLDS (CurrentDir = /usr/matt/, T1)

the fact is expressed that the current directory is **/usr/matt/** during the interval **T1**. Properties hold in each subinterval of the specified interval as well.

With

HOLDS (LoggedIn (Matthias, Yes), T1)

the fact is expressed that the user *Matthias* is logged in during **T1**.

5.2. *Description of actions*

Commands, parameters, the fact that a command was used, and the recognition of used commands are all represented as terms.

5.2.1. *The 'cd' command*

With

$$
\begin{aligned}
&\forall x, y, z, t_1, \exists t_2 \\
&HOLDS(CurrentDir = x, t_1) \wedge \\
&HOLDS(WasUsed(cd(y)), t_1) \wedge \\
&HOLDS(Parameter(cd(y), z), t_1) \wedge \\
&MEETS(t_1, t_2) \\
&\Rightarrow \\
&HOLDS(CurrentDir = x\ o\ z, t_2) \wedge \\
&HOLDS(Occur(cd(y), t_1), t_2) \wedge \\
&HOLDS(Occur(Parameter(cd(y), z), t_1), t_2)
\end{aligned}
\tag{1}
$$

the command **cd** is formalized. In the second line, the precondition "there must be a **CurrentDir**" is specified. With **WasUsed**, the fact that a command of type **cd(y)** was used is described. If a specific **cd** command was used, the variable **y** must be instantiated with a unique identifier. In the fourth line, the parameter **z** which was used by the **cd** command is specified. If all these formulae are valid in the world model, then the theorems on the right side of the implication can be deduced.

With the predicate **MEETS**, the fact is specified that there are no intervals allowed between t_1 and t_2. Given this possibility, then another command execution could destroy the properties which are valid after the execution of the **cd** command (the **CurrentDir** might have be changed).

In the seventh, line the fact that the **cd** command has changed the world model is stated. The new current directory is the concatenation of **x** and **z**.

HOLDS(Occur(cd(y), t_1**),** t_2**)** specifies that the employment of the **cd** command (which was used in interval t_1) was recognized (simple action recognition) in interval t_2. With the last formula the use of the parameter of the **cd** command is recognized.

5.2.2. *The 'lpr' command*

$$\begin{aligned} &\forall x, y, z, t_1, \exists t_2 \\ &HOLDS(CurrentDir = x, t_1) \wedge \\ &HOLDS(IsFileIn(z, x), t_1) \wedge \\ &HOLDS(WasUsed(lpr(y)), t_1) \wedge \\ &HOLDS(Parameter(lpr(y), z), t_1) \wedge \\ &MEETS(t_1, t_2) \\ &\Rightarrow \\ &HOLDS(Occur(lpr(y), t_1), t_2) \wedge \\ &HOLDS(Occur(Parameter(lpr(y), z), t_1), t_2) \end{aligned} \tag{2}$$

With **IsFileIn (z, x)**, it is guaranteed that the **lpr** command is only used if the specified parameter **z** (the file which should be printed) defines an existent file in the current directory.

5.2.3. *The 'rm *' command*

$$\begin{aligned} &\forall x, t_1, \exists t_2 \\ &HOLDS(WasUsed(rm(x)), t_1) \wedge \\ &HOLDS(Parameter(rm(x), *), t_1) \wedge \\ &MEETS(t_1, t_2) \end{aligned} \tag{3}$$

$$\Rightarrow$$
$$HOLDS(Occur(rm(x), t_1), t_2) \wedge$$
$$HOLDS(Occur(Parameter(rm(x), *), t_1), t_2) \wedge$$
$$\text{“}All\ files\ in\ the\ current\ directory\ are\ deleted\text{”}$$

With **All files in the current directory are deleted**, the effects of the **rm *** command on the world model are described. We omit the exact formalization because we are mainly interested in the process of plan recognition.

5.2.4. *The ‘cd ..’ command*

$$\forall x, y, z, t_1, \exists t_2 \qquad (4)$$
$$HOLDS(CurrentDir = x|y, t_1) \wedge$$
$$HOLDS(WasUsed(cd(z)), t_1) \wedge$$
$$HOLDS(Parameter(cd(z), \ldots), t_1) \wedge$$
$$MEETS(t_1, t_2)$$
$$\Rightarrow$$
$$HOLDS(CurrentDir = x, t_2) \wedge$$
$$HOLDS(Occur(cd(z), t_1), t_2) \wedge$$
$$HOLDS(Occur(Parameter(cd(z), \ldots), t_1), t_2)$$

With **x|y**, the name of the **CurrentDir** is divided into two parts. **x** is the name of the parent directory and **y** is the name of one of its subdirectories.

5.2.5. *The ‘rmdir’ command*

$$\forall x, y, z, t_1, \exists t_2 \qquad (5)$$
$$HOLDS(CurrentDir = x, t_1) \wedge$$
$$HOLDS(IsDirIn(y, x), t_1) \wedge$$
$$HOLDS(WasUsed(rmdir(z)), t_1) \wedge$$
$$HOLDS(Parameter(rmdir(z), y), t_1) \wedge$$
$$MEETS(t_1, t_2)$$
$$\Rightarrow$$
$$HOLDS(Occur(rmdir(z), t_1), t_2) \wedge$$
$$HOLDS(Occur(Parameter(rmdir(z), y), t_1), t_2) \wedge$$
$$\text{“}The\ directory\ with\ name\ y\ was\ deleted\text{”}$$

With **IsDirIn(y, x)**, it is guaranteed that the **rmdir** command is only used if the specified parameter of the command **y** is a directory in the current

directory. With **The directory with name y was deleted**, the effect of the **rmdir** command on the world model is expressed.

5.3. *Description of atomic plans*

A *plan* is composed of one or more commands or sub-goals which form the *body* of the plan and the *goal*, which is persued by the commands and the sub-goals. The body of an *atomic plan* consists only of commands. In the body of a *non-atomic* plan, sub-goals and commands are mixed.

In the following, we formalise those atomic plans needed to model the insertion of plans.

5.3.1. *The 'cd' plan*

The following example represents an atomic plan:

$$\begin{aligned} &\forall x, t_1, t_2, t_3 \\ &HOLDS(Occur(cd(x), t_1), t_2) \wedge \\ &HOLDS(Occur(Parameter(cd(x), ..), t_1), t_2) \wedge \\ &(MEETS(t_2, t_3) \vee BEFORE(t_2, t_3)) \\ &\Rightarrow \\ &HOLDS(Goal(ChangeDir([cd(x)]), t_1), t_3) \end{aligned} \tag{6}$$

The second line states the fact that the **cd** command occured in time interval t_1. This was recognized within the interval t_2 (simple action recognition). In the third line, it is specified that the parameter of the cd command must not be .. (the parent directory). In the fourth line, the relationships between the intervals is specified in which the used command was recognized and in which the plan recognition with this atomic plan should be performed:

- *MEETS*(t_2, t_3) states that the recognition of the goal of the atomic plan can immediately follow the recognition of the use of the command.
- *BEFORE*(t_2, t_3) specifies that the recognition of the goal must not immediately follow the recognition of the employment of the command. There can be intervals between t_2 and t_3. So, it is possible to separate goal recognition from that of its employment or from the use of the commands.

In the first line following the right arrow, the goal of this atomic plan is specified. **HOLDS(Goal(ChangeDir([cd(x)]),** t_1**),** t_3**)** states that the goal **ChangeDir** was persued in t_1 with the command list **[cd(x)]** and the goal of this atomic plan was recognized in t_3.

5.3.2. *The 'lpr' plan*
In the following atomic plan two commands are used:

$$\begin{aligned} &\forall x, y, z, u, t_1, t_2, t_3, t_4, t_5 \\ &HOLDS(Occur(lpr(x), t_1), t_2) \wedge \\ &HOLDS(Occur(Parameter(lpr(x), y), t_1), t_2) \wedge \\ &HOLDS(Occur(lpr(z), t_3), t_4) \wedge \\ &HOLDS(Occur(Parameter(lpr(z), u), t_3), t_4) \wedge \\ &(t_1 < m\ t_3) \wedge \\ &(t_2 < t_6) \wedge (t_4 < m\ t_6) \wedge \\ &STARTS(t_1, t_5) \wedge FINISHES(t_3, t_5) \\ &\Rightarrow \\ &HOLDS(Goal(PrintTwoFiles([lpr(x), lpr(z)]), t_5), t_6) \end{aligned} \tag{7}$$

We assume that if commands occured and if they form a plan, then the interval in which the goal of the plan is persued must contain the intervals in which the commands occured, so, t_1 and t_3 must be **IN** t_5. Also, because the commands form the plan, the interval of the first command **STARTS** the interval of the goal and the interval of the last command **FINISHES** the interval of the goal. The use of the second command can immediately follows the use of the first command, but intervals between $(t_1 < m\ t_3)$ are possible.

These intervals represent the basis for modelling inserted sub-plans.

5.3.3. *The 'rm *' plan*

$$\begin{aligned} &\forall x, t_1, t_2, t_3 \\ &HOLDS(Occur(rm(x), t_1), t_2) \wedge \\ &HOLDS(Occur(Parameter(rm(x), *), t_1), t_2) \wedge \\ &(MEETS(t_2, t_3 \vee BEFORE(t_2, t_3)) \\ &\Rightarrow \\ &HOLDS(Goal(DeleteAllFiles([rm(x)]), t_1), t_3) \end{aligned} \tag{8}$$

5.3.4. *The 'cd ...' plan*

$$\begin{aligned} &\forall x, t_1, t_2, t_3 \\ &HOLDS(Occur(cd(x), t_1), t_2) \wedge \\ &HOLDS(Occur(Parameter(cd(x), ..), t_1), t_2) \wedge \end{aligned} \tag{9}$$

$$(MEETS(t_2, t_3) \vee BEFORE(t_2, t_3)) \Rightarrow HOLDS(Goal(ChangeToParentDir([cd(x)]), t_1), t_3)$$

5.3.5. *The 'rmdir' plan*

$$\forall x, t_1, t_2, t_3 \quad (10)$$
$$HOLDS(Occur(rmdir(x), t_1), t_2) \wedge HOLDS(Occur(Flag(rmdir(x), -r), t_1), t_2) \wedge (MEETS(t_2, t_3) \vee BEFORE(t_2, t_3)) \Rightarrow HOLDS(Goal(DeleteADir([rmdir(x)]), t_1), t_3)$$

5.4. *Description of non-atomic plans*

In a non-atomic plan, the body can contain not only commands but sub-goals as well. For example, in the following formula, the plan to reach the goal **delete a directory together with its content** is formalized:

$$\forall x, y, z, u, w, t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8 \quad (11)$$
$$HOLDS(Goal(ChangeDir(x), t_1), t_2) \wedge HOLDS(Goal(DeleteAllFiles(y), t_3), t_4) \wedge HOLDS(Goal(ChangeToParentDir(z), t_5), t_6) \wedge HOLDS(Goal(DeleteADir(u), t_7), t_8)) \wedge HOLDS(Occur(Parameter(GetCommandToken(x), w), t_1), t_2) \wedge HOLDS(Occur(Parameter(GetCommandToken(u), w), t_7), t_8) \wedge (t_1 < m\ t_3) \wedge (t_3 < m\ t_5) \wedge (t_5 < m\ t_7) \wedge (t_2 < m\ t_{10}) \wedge (t_4 < m\ t_{10}) \wedge (t_6 < m\ t_{10}) \wedge (t_8 < m\ t_{10}) \wedge IN(t_1, t_9) \wedge IN(t_3, t_9) \wedge IN(t_5, t_9) \wedge IN(t_7, t_9) \wedge STARTS(t_1, t_9) \wedge FINISHES(t_7, t_9) \Rightarrow HOLDS(Goal(DeleteADirAndAllFiles([x, y, z, u]), t_9), t_{10})$$

The body consists of the four sub-goals **ChangeDir, DeleteAllFiles, ChangeToParentDir**, and **DeleteADir**. In line six and seven, the parameters

used in the first and fourth plan in reaching the appropriate sub-goal must be the same.

In line eight, the temporal relationships between the sub-goals are given. in the last two lines before the implication sign, the relationships between the intervals in which the sub-goals are persued and the interval in which the overall goal is persued are described. Note that the interval of the goal does not begin before the first sub-goal interval and does not end after the last sub-goal interval.

5.5. *Axiom for the detection of inserted plans*

In order to detect the inserted sub-plans, we must have several goals. If one of these goals are persued in a time interval which is a sub-interval of a second goal, then we say the plan used to reach the first goal is inserted into the plan employed to reach the second goal. We formalize this in:

$$\begin{aligned} &\forall x, t_1, t_2, t_3, t_4, t_5 \qquad (12)\\ &HOLDS(Goal(x, t_1), t_2) \wedge HOLDS(Goal(y, t_3), t_4) \wedge \\ &\qquad IN(t_1, t_3) \wedge (t_2 < m\ t_5 \wedge (t_4 < m\ t_5)\\ &\qquad\qquad \Rightarrow \\ &\qquad HOLDS(InsertedSubPlan(x, t_1, y, t_3), t_5 \end{aligned}$$

We must extend the above formula to exclude those deductions which simply state that **x** is inserted in **y** in the case that **x** is a sub-goal of **y**. We omit this extension here.

5.6. *How to handle insertion in the deductive model*

We have formalized different commands, several atomic plans, and a non-atomic plan. Now, we show how the world model changes if the commands are executed and how plan recognition proceeds. The example used here is the same as in section 3.1.

In this example, the time structure used is given by the following predicates:

(T1 m T2) $\wedge$ (T3 m T4) $\wedge$ (T5 m T6) $\wedge$
(T7 m T8) $\wedge$ (T9 m T10) $\wedge$ (T11 m T12) $\wedge$
(T1 <m T3) $\wedge$ (T3 <m T5) $\wedge$ (T5 <m T7) $\wedge$
(T7 <m T9) $\wedge$ (T9 <m Tll) $\wedge$
(T2 <m = T13) $\wedge$
(T4 <m T15) $\wedge$ (T6 <m T15) $\wedge$ IN(T3, T14) $\wedge$ IN(T5, T14) $\wedge$

STARTS(T3, T14) ∧ FINISHES(T5, T14) ∧
(T8 <m = T16) ∧
(T10 <m = T17) ∧
(T12 <m = T18) ∧
(T13 <m T19) ∧ (T15 <m T19) ∧ (T16 <m T19) ∧ (T17 <m T19) ∧
(T18 <m T19) ∧
IN(T1, T20) ∧ IN(T7, T20) ∧ IN(T9, T20)∧ IN(T11, T20) ∧
(T15 <m T21) ∧ (T19 <m T21)

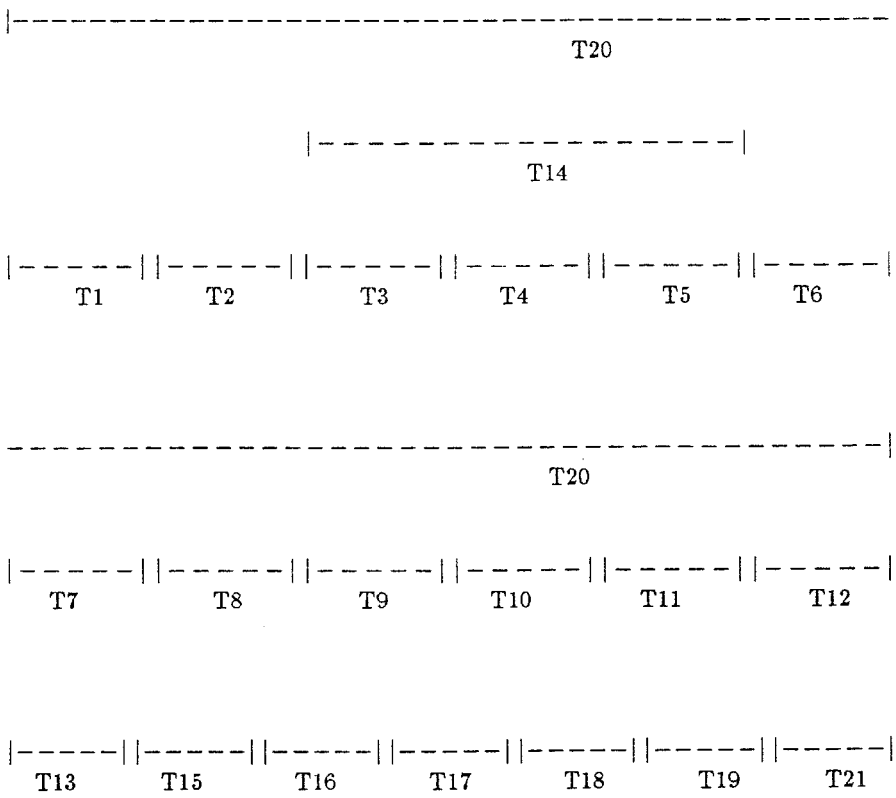

Figure 4. Example of the time structure

One representation of the time structure is given in Figure 4. Note that other representations are possible.

First command: After the employment of the command **cd letters**, the subsequent formulae are valid in the deductive model:

HOLDS(WasUsed(cd(K1)), T1) ∧
HOLDS(Parameter(cd(K1), letters), T1) ∧
HOLDS(CurrentDir = /usr/matt/, T1)

In the first two lines, the command used is specified. The constant **K1** names this specific use of the **cd** command. **T1** is the name given to the time interval in which the command occured.

Because there is an intervall **T2** with **MEETS(T1, T2)** in the time structure, the description of the **cd** command (s. formula 1) can be used. The following theorems are deduced:

HOLDS(Occur(cd(K1), T1), T2) ∧
HOLDS(Occur(Parameter(cd(K1), letters), T1), T2) ∧
HOLDS(CurrentDir=/usr/matt/letters/, T2)

Second command: After the employment of the command **lpr juliane**, the subsequent formulae are valid in the deductive model:

HOLDS(WasUsed(lpr(K2)), T3) ∧
HOLDS(Parameter(lpr(K2), juliane), T3) ∧
HOLDS(CurrentDir = /usr/matt/letters/, T3) ∧
HOLDS(IsFileIn(juliane, /usr/matt/letters/), T3)

Here we are faced with the *frame problem*. We must specify through *frame axioms* what happens with those formulae valid in an interval with respect to subsequent intervals. A frame axiom must guarantee that in **T3**, the **CurrentDir** must be the same as in **T2**. We assume that appropriate frame axioms are available. For a discussion of the frame problem cf. Brown 1987.

With the theorem **MEETS(T3, T4)** and the command description in formula 2, we can deduce the subsequent formulae:

HOLDS(Occur(lpr(K2), T3), T4) ∧
HOLDS(Occur(Parameter(lpr(K2), juliane), T3), T4)

Third command: After the employment of the command **lpr sabine**, the subsequent formulae are valid in the deductive model:

HOLDS(WasUsed(lpr(K3)), T5) ∧
HOLDS(Parameter(lpr(K3), sabine), T5) ∧
HOLDS(CurrentDir = /usr/matt/letters/, T5) ∧
HOLDS(IsFileIn(juliane, /usr/matt/letters/), T5)

With the theorem **MEETS(T5, T6)** and the command description in formula 2, we can deduce the subsequent formulae:

HOLDS(Occur(lpr(K3), T5), T6) ∧
HOLDS(Occur(Parameter(lpr(K3), sabine), T5), T6)

Forth command: After the employment of the command **rm ***, the subsequent formulae are valid in the deductive model:

HOLDS(WasUsed(rm(K4)), T7) ∧
HOLDS(Parameter(rm(K4), *), T7)

With the theorem **MEETS(T7, T8)** and the command description in formula 3, we can deduce the subsequent formulae:

HOLDS(Occur(rm(K4), T7), T8) ∧
HOLDS(Occur(Parameter(rm(K4), *), T7), T8) ∧
"All files in the current directory are deleted"

Fifth command: After the employment of the command **cd . .**, the subsequent formulae are valid in the deductive model:

HOLDS(WasUsed(cd(K5)), T9) ∧
HOLDS(Parameter(cd(K5), . . .), T9) ∧
HOLDS(CurrentDir = /usr/matt/letters/, T9)

With the theorem **MEETS(T9, T10)** and the command description in formula 4, we can deduce the subsequent formulae:

HOLDS(Occur(cd(K5), T9), T10) ∧
HOLDS(Occur(Parameter(cd(K5), . . .), T9), T10) ∧
HOLDS(CurrentDir = /usr/matt/, T10)

Sixth command: After the employment of the command **rmdir letters**, the subsequent formulae are valid in the deductive model:

HOLDS(WasUsed(rmdir(K6)), T11) ∧
HOLDS(Parameter(rmdir(K6), letters), T11) ∧
HOLDS(IsDirIn(letters, /usr/matt/), T11) ∧
HOLDS(CurrentDir = /usr/matt/), T11)

With the theorem **MEETS(T11, T12)** and the command description in formula 5, we can deduce the subsequent formulae:

HOLDS(Occur(rmdir(K6), T11), T12) ∧
HOLDS(Occur(Parameter(rmdir(K6), letters), T11), T12) ∧
"The directory with name y was deleted"

Use of atomic plans: The employment of the commands are recognized. Now we use the atomic plans to see which goals are being persued.

Since the preconditions of plan no. 6 are satisfied (the recognition of the used command was in interval T1 which is before T13), the following theorem is infered:

HOLDS(Goal(ChangeDir([cd(K1)]), T1), T13)

With plan no. 7 we get:

HOLDS(Goal(PrintTwoFiles([lpr(K2), lpr(K3)]), T14), T15)

With plan no. 8 we get:

HOLDS(Goal(DeleteAllFiles([rm(K4)]), T7), T16)

With plan no. 9 we get:

HOLDS(Goal(ChangeToParentDir([cd(K5)]), T9), T17)

With plan no. 10 we get:

HOLDS(Goal(DeleteADir([rmdir(K6)]), T11), T18)

Use of the non-atomic plan: Because the preconditions for the non-atomic plan (s. formula 11) are satisfied, we can deduce:

HOLDS(Goal(DeletADirAndAllFiles([[cd(K1)], [rm(K4)], [cd(K5)], [rmdir(K6)]]), T20), T19)

Deduce the insertion: with formula no. 12, we now can deduce that an insertion exists.
The antecedences are:

HOLDS(Goal(PrintTwoFiles([lpr(K2), lpr(K3)]), T14), T15) ∧
HOLDS(Goal(DeletADirAndAllFiles([[cd(K1)], [rm(K4)], [cd(K5)], [rmdir(K6)]]), T20), T19)

Because T24 is **IN** T20, T15 is **BEFORE** T21, and T19 is **BEFORE** T21, we receive:

HOLDS(InsertedSubPlans(PrintTwoFiles([lpr(K2), lpr(K3)]), T14, DeletADirAndAllFiles([[cd(K1)], [rm(K4)], [cd(K5)], [rmdir(K6)]]), T20), T21)

5.7. *Limits of this approach*

We have seen that the inserted sub-plans can be modelled quite simply, if we use an interval-based time logic.

One deficiency, which is actually a deficiency of the used interval-based time logic, is that it is impossible to specify the maximum amount of time between the first command before and after an inserted sub-plan. For example, if you don't have the possibility to specify this maximum of interruption time, it might be possible that commands interrupted for several hours are treated as belonging to the same plan.

This deficiency is actually due to the lack of not being able to specify the duration time of an interval. This concept is unknown in the logic of time used by Allen.

The most elaborated formal theory of plan recognition was developed by H. Kautz (cf. Kautz 1987, Kautz and Allen 1986). Because his time logic is also taken from Allen, his theory is also unable to handle the insertion appropriately. But, with respect to other related topics of plan recognition, the theory of H. Kautz is far more elaborated.

6. Possible Extensions of the Model

Because the model presented is only a first step towards an overall theory of plan recognition, we briefly mention those topics which *might* be handled in such a theory:

1. The time logic used should be altered so that we can model the duration of time intervals. Another interesting question is the necessity of a time-point based logic of time.
2. As already pointed out, the frame problem occurs.
3. Non-monotonic plan recognition should be handled.
4. The question arises if plan recognition should be modelled with respect to different agents[4].
5. How can incremental plan recognition be handled?

7. Summary

We have briefly described the realized active help system SINIX Consultant. A more detailed description of the plan recognition component REPLIX was given. The experience gained in realizing the plan recognition component was taken as a first-step basis towards a formal theory of plan recognition. We have shown how we can use an interval-based time logic in order to formulate the theory. The insertion of sub-plans was modelled, but at once, we are faced with a problem which cannot be appropriately handled in this initial approach.

Notes

[1] The SINIX Consultant project is partially funded by the Siemens AG as part of the III (Innovative Information infrastructures) project, a cooperation between the University of Saarbrücken and the Siemens AG.
SINIX is the UNIX derivate of the Siemens AG

[2] move file 'letter1' to the directory 'invoices'

[3] REPLIX was designed and implemented by D. Dengler, M. Gutmann, G. Hector and M. Hecking.

[4] For a treatment of this topic see Hecking 1988.

References

Allen, J. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM* **26**: 832–843.

Allen, J. (1984). Towards a general theory of action and time. *Artificial Intelligence* **23**: 123–154.

Boggs, W. M., Carbonell, J. G. & Monarch, J. G. (1984). *DYPAR – I: Tutorial and Reference Manual*. Technical Report, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh (PA).

Brown, F. M. (1987). *The Frame Problem in Artificial Intelligence*, Morgan Kaufmann Publishers, Los Altos, California.

Dengler, D., Gutmann, M. & Hector, G. (1987). *Der Planerkenner REPLIX*. Memo No. 16, Dept. of Computer Science, University of Saarbrücken, W. Germany.

Finin, T. W. (1983). Providing help and advice in task oriented systems. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, 176–178.

Finkler, W. & Neumann, G. (1986). *MORPHIX – Ein hochportables Lemmatisierungsmodul für das Deutsche*. Memo No. 8, Dept. of Computer Science, University of Saarbrücken, W. Germany.

Genesereth, M. R. & Nilsson, N. J. (1987). *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers, Los Altos, California.

Hecking, M. (1987). How to Use Plan Recognition to Improve the Abilities of the Intelligent Help System SINIX Consultant. *Proceedings of the Second IFIP Conference on Human-Computer Interaction, held at the University of Stuttgart, Federal Republic of Germany, 1–4 September*, 1987, 657–662.

Hecking, M. (1988). *Towards a Belief-Oriented Theory of Plan Recognition*. Report No. 50, Dept. of Computer Science, University of Saarbrücken, W. Germany.

Hecking, M. & Harbusch, K. (1987). *Plan Recognition through Attribute Grammars*. Memo No. 17, Dept. of Computer Science, University of Saarbrücken, W. Germany.

Hecking, M., Kemke, C., Nessen, E., Dengler, D., Gutmann, M. & Hector, G. (1988). *The SINIX Consultant – A Progress Report*. Memo No. 28, Dept. of Computer Science, University of Saarbrücken, W. Germany.

Kautz, H. A. & Allen, J. F. (1986). Generalized plan recognition. *Proceedings of the 5th National Conference of the American Association on Artificial Intelligence, Philadelphia, Pennsylvania*, 32–37.

Kautz, H. A. (1987). *A Formal Theory of Plan Recognition*. Report No. TR 215, University of Rochester, Department of Computer Science, 5.

Kemke, C. (1987). Representation of Domain Knowledge in an Intelligent Help System. *Proceedings of the Second IFIP Conference on Human-Computer Interaction, held at the University of Stuttgart, Federal Republic of Germany, 1–4 September*, 1987, 215–220.

Konolige, K. (1986). *A Deduction Model of Belief. Research Notes in Artificial Intelligence*, Morgan Kaufmann Publishers, Los Altos, California.

Prior, A. (1957). *Time and Modality*. Clarendon Press, Oxford.

Quilici, A., Dyer, M. & Flowers, M. (1986). AQUA: AN INTELLIGENT UNIX ADVISOR. *Proceedings of the 7th European Conference on Artificial Intelligence*, 33–38.

Th. Schwab. (1984). *AKTIVIST – Ein aktives Hilfesystem für den bildschirmorientierten Editor BISY*. Diplomarbeit Nr. 232, Institut für Informatik, Universität Stuttgart.

Wilensky, R., Arens, Y. & Chin, D. (1984). Talking to UNIX in English: An Overview of UC. *Communications of the ACM* **27**: 574–593, 6.

Wilensky, R., Myfield, J., Albert, A., Chin, D., Cox, C., Luria, M., Martin, J. & Wu, D. (1986). *UC – A Progress Report*. Report No. UCB/CSD 87/303, Computer Science Division (EECS), University of California, Berkeley, **7**.