Inferring Complex Plans

Kristina Höök, Jussi Karlgren, Annika Wærn
Swedish Institute of Computer Science, Box 1263, S-164 28 Kista, Sweden
kia@sics.se, jussi@dsv.su.se, annika@sics.se

January 1993

Abstract

We examine the need for plan inference in intelligent help mechanisms. We argue that previous approaches have drawbacks that need to be overcome to make plan inference useful. Firstly, plans have to be inferred - not extracted from the users' help requests. Secondly, the plans inferred must be more than a single goal or solitary user command.

Keywords

Plan inference, interactive help systems, configuration problem

INTRODUCTION

Plan inferencing is the task of detecting what the current goal of a user is, and what plan she is carrying out to fulfil this goal. Plan inference is a theoretical tool in several different research areas in cognitive psychology and linguistics to explain processes of understanding and language production, and more generally, to explain the underlying processes in human action and behavior. In this type of research, the formalization of plan inference has as its main purpose to provide explanatory strength. In computer science, the study of plan inference has a different purpose: to examine how plan inference can be used in practice for speeding up computations or making computer systems more user friendly: to provide computational efficiency. As computer scientists, the two questions regarding plan inference we must ask ourselves are then "In what type of situations is plan inference useful?" and "Can plan inference processes be designed to be efficient?"

In this paper we argue that plan inference under certain conditions indeed is both useful and feasible. We give an example of a type of interactively solvable problems where the system will have more use of and practical possibilities to perform plan inference than in the traditional interactive applications of today: configuration of interactive systems to suit specific needs of specific users.

This type of problem is likely to become more prevalent in future applications. As an example domain for configuration problems we give systems for management of electronic mail messages.

We find that the suggestions found in the existing literature on plan inference all have shortcomings that make them unfeasible for practical usage in building interactive systems. We end the discussion by some specifications of the functionality we need, and a set of problems that must be overcome in order to yield a feasible solution to the problem.

Do People Use Plan Inference?

Human agents are able to perform a substantial amount of communication without seeming to use plan inference. That people get by without doing much plan inference has to do with the fact that people are people, and we all have the same processing units. Inferring plans for an agent with a similar inferencing process as oneself is naturally easier than if one has to assume the possibility of completely different processes. When communicating with another person, we can assume that our counterpart reasons and acts the same way we would, and our interpretation of utterances and actions is guided by what we would have done ourselves in the same situation.

The less evidence we have of a common ground and common understanding, the more difficult becomes the communication. This scale of processing mutualness comes to an extreme case when the counterpart is a machine. Several usual features of discourse disappear: users seem to feel little need to produce connected discourse (Dahlbäck, 1991). The working mechanisms of a computer system are very different from human thinking, and users know it (Joshi, 1982; Karlgren, 1992) When working with a computer system, users must have a complex and conscious understanding of the system's functionality, such as of what tasks it can perform and how and when it goes about to perform them. The converse is also true: for a computer system to act in a usefully user-adapted way, it should be equipped with an understanding of what users can do, and made capable of understanding of how and when they do what.

Space of Possible Plans Is The Problem

Human agents have a hierarchy of goals for every action they perform, ranging from trivially high level goals such as having a good time, procreating, and reaching a state of grace on the one hand, to trivially low level goals on the other; for example, if a person queries the time from somebody else, her immediate goal is to find out what time it is. Neither of these extremes are interesting for the immediate purposes of building better interactive interfaces. The interesting goals from a system engineering perspective are goals that immediately subsume the actions a user performs at an interface.

Given a system with a specific functionality - interpreted as a set of possible and conceivable goals - the possibility of interpreting a sequence of actions as an attempt to fulfil a goal or a set of goals varies with the expressivity of the interaction language. A system with a very complex functionality and a low level interaction language will have more trouble inferring an appropriate goal than a system with the same functionality and an interaction language which allows and handles commands or expressions on a higher level. Similarly, a system with a small range of functionalities will have less trouble inferring goals than a system with large range of possible tasks.

Intelligent Help

Intelligent help systems are intended to provide on-line help to users making use of the current context, the users' own expertise, and the users' current intentions. Worded differently, an intelligent help system understands what users are trying to do and gives help for that rather than for what they are actually doing. Intelligent help in our terms is thus an almost canonical example of an application domain in which plan inference is needed.

The Configuration Problem

Almost every word processor and operating system affords the user the opportunity to tailor or configure the system according to preferences by installing an init or profile file or some other means. For frequent users this is a powerful way of creating an environment to feel comfortable with. For sporadic users there usually is a default behavior that can be used: changing profile files is difficult for these users (Kass, 1991). In our present work we have found applications within the communication area where configuration is a part of the system functionality rather than simply a feature for frequent users. Examples are advanced storage systems for electronic mail, computer supported cooperative work, multimedia communication, etc. For such communication tools the configuration cannot simply be left as a default, since the environment in which we communicate will change over time.

In these applications the configuration task is necessary to perform even for non-expert users, and thus the need for interactive help and guidance increases. We expect plan inference to be feasible for this purpose, since system configuration is a task in which the command language is fairly high level, while the functionality is reasonably narrow-scope.

EXISTING APPROACHES

Plan inference is used to deduce the underlying intentions of queries posed. But how much of a plan is in fact inferred? Looking closer at work in this research area we find firstly that the plans inferred are plans only in a very limited sense, and that the inference done is of a very simplistic kind.

Plan Debugging

One important use of the inferred plans is to debug them, that is, recognize when a user has an faulty or suboptimal way to fulfil a certain goal. This task has been tackled by Pollack (1986). Although interesting, Pollack's work has some shortcomings which makes it insufficient for the intelligent help domain. Firstly, with Pollack's approach no real plan inference is really taking place. In fact, she assumes that the whole plan is stated by the user in the questions posed to the help system. (This problem has been partly addressed in later work (Konolige and Pollack, 1989).)

Secondly, the plans investigated are so-called simple plans where one act generates the rest of the acts. This definition means that the user of the system will only perform one action whereupon the rest of the actions will happen automatically. Plans where one action will set up the context for the next action necessary to achieve the goal (enablement), are not dealt with in her work.

The "Buggy Model" Approach: Eurohelp

Systems that use true plan inference usually rely on the so-called "buggy model" approach. In this approach, the system monitors the actions of the user, constantly trying to infer the goal the user is trying to reach. The help system contains a library of correct plans as well as incorrect and non-optimal plans, and matches the sequence of user actions against these in order to find one that fits. In fact, this way of doing plan inference might just as well be called pattern matching. No construction of plans is taking place.

The buggy model approach to plan inference has been used for intelligent help in the EUROHELP project (Breuker *et al*, 1987). The purpose of this project was to construct an application-independent shell for intelligent help. As we know of no substantial applications where this shell has been used, it is difficult to say how successful the EUROHELP plan recognition attempt was.

The main critique against this model for plan inference is that the set of buggy and suboptimal plans needs to be very large in order to cover a useful set of user behavior. This puts a large burden on the designer of the plan library. Its design must also be guided by knowledge of human cognitive processes and by particular studies of the application at hand in order to yield a useful result.

An additional difficulty for the system configuration task we are envisioning is the dynamics of the system. The set of envisionable goals and plans will change over time, making it very difficult for a rigid, pre-programmed hierarchy to cope. In the extreme case, even the alphabet of the command language can change.

Natural Language Interface Systems

Work on natural language interfaces has developed from intrasentential syntactic and semantical analysis to levels where pragmatic analysis is necessary. Part of the pragmatic analysis involves inferring goals that a communicator has when producing an utterance (Kaplan, 1982). A common strategy used in this type of systems is to examine utterances one at a time, and to classify them according to their functional type: so called speech act analysis. In the Unix Consultant system UC, for instance, the underlying goal is inferred from a single utterance or question (Wilensky, 1983). Since the analysis is based on single questions to a separate consultant system, it requires either that the user provides the context in the question posed, or that the answer is not dependent on knowing the context. The following example shows a case where the context is provided by the user in the question itself (Chin, 1989):

```
  I tried typing "rm foo"
but I got the message
"rm: foo not removed".
```

Although UC is capable of inferring the goals of the user, it does not attempt to infer the plans by monitoring the users actions with the target system. This makes it difficult or impossible to correct the users behavior when attempting a goal by a faulty plan. Plan inference would also make it possible to give correct answers to context dependant help requests. Furthermore the UC approach requires that the user is capable of expressing her goal which may not always be the case.

Example Scenario: Managing Mail Messages

The example scenario in this section is a configuration problem, where successful plan inference would enhance the system functionality. It is based on PostMaster, a system for managing electronic mail messages currently under development at SICS. PostMaster is similar to the Information Lens system developed at MIT, and does not provide help based on plan inference (Malone *et al*, 1987). Users can automatically sort their incoming electronic mail using rules that they have provided to the system. The rules utilize information found in the header fields of the messages. Example rules are shown in the figure below. Assume that a user has created a new mailbox named PPIG[1] and then moved a set of old messages to it. She then turns to the system with a vague help request:

```
"What now?"

IF sender OR receiver = Tom Ormerod
THEN put in folder Ormerod
IF sender = David Gilmore
THEN put in folder Gilmore
```

The system will search for a consistent pattern in the actions the user has performed. In this case, the user has moved all letters from Tom Ormerod and David Gilmore (who have one mailbox each) into the new PPIG mailbox, excepting a subset of letters which all contain the word "workshop" in the subject header. Due to the intended functionality of mailboxes, the system knows that if a mailbox has been created, there ought to be at least one rule routing letters to it. The system now assumes that what the user wants to do is to reconfigure the system, so that all letters that previously went to Tom's and David's mailboxes now should go to the PPIG mailbox, possibly excepting the "workshop" letters. The

[1]Psychology of Programming Interest Group

system confirms its guess by asking[2]:

```
"Do you want to put all incoming
mail that previously went to the
'Ormerod' and 'Gilmore' mailboxes
into the PPIG folder?"
```

The user (relieved by the apparent understanding of her predicament) answers yes. The system then creates a rule by merging the old rules for the Ormerod and Gilmore mailboxes and displays it for the user to edit.

```
PPIG-rule1:
IF sender OR receiver = Tom Ormerod
THEN put in folder PPIG
PPIG-rule2:
IF sender = David Gilmore
THEN put in folder PPIG
```

Since the creation of these rules was based on guessing the user's goal of replacing the Ormerod and Gilmore mailboxes by the PPIG mailbox, the rules move a wider span of mail to the new mailbox than was indicated by the set of actions performed by the user. For example, the user may not have moved any mail where Tom Ormerod only was a receiver.

But the problem is not yet solved completely - the letters with "workshop" in the header were not moved. What does the user want with these letters? The system asks again:

```
"Do you want to do anything
particular with letters to
the PPIG folder, if the header
contains the word "workshop"?"
```

The user now replies

```
"PPIG workshop dates should also
be added to my calendar."
```

[2]In the example we use an interaction language resembling written English to make the help functionality clear. This may not necessarily be the best choice for an interaction language, but this issue will not be explored further here.

This causes the system to create and display the rule

```
IF
    PPIG-rule1 OR PPIG-rule2 applies
AND
    subject contains ''workshop''
THEN
    put in folder PPIG
AND
    start calendar program
```

Potentially, it could also interpret the answer as an implicit request to perform this rule on the letters remaining in the old folders. Finally, the functionality of the old mailboxes of Ormerod and Gilmore is now completely redundant, and if the system was right in its guess, these should be deleted. The system asks:

```
"Do you want to delete the
folders 'Ormerod' and 'Gilmore'?"
```

If the answer is yes, the system deletes these folders and the rules routing letters to them.

In this scenario, the fact that a configuration was needed was inferred from the user using the system in a recognizable manner, together with the fact that the user requested help. The configuration goal was correctly understood, and required a whole set of actions in order to be consistently fulfilled, some of them in a particular sequence.

Necessary Functionality

We have already stated that we expect plan structures to contain actions related not only by generation but also by enablement. We have also claimed that the system should not be dependent on every possible plan being explicitly represented and preconceived in order to be recognizable. The system must be able to reason backwards from the actions of the user, infer a goal or a set of goals that is compatible with the actions, instead of requiring that the plan is stated in the question posed.

These properties are not easy to achieve. The system must be able to recognize incorrect and suboptimal plans: a sequence of actions by the user may not always be directed towards a single simple goal. The user may interleave several tasks, or abandon a task in mid-plan to start another. Typically, the set of user actions involved in fulfilling a goal is small, and even if the required set of actions is large, we want to recognize this fact early in order to give appropriate help. This implies that the system must be able to pick up a plan from very few user actions.

Necessary Functional Architecture

To avoid the plan library problem, we propose to partition the domain knowledge into two more basic types of knowledge: a library of primitive actions, and their preconditions and effects, and a structured knowledge, possibly a goal taxonomy resembling Wilensky's (1983), consisting of the goals and subgoals the user may want to fulfil or maintain when using the system.

The difficulty now becomes how to connect the users' actions to a goal in the goal hierarchy. The observed sequence of user actions must be matched against the possible set of goals by bottom-up planning. This matching should be based both on objective goal fulfilment or enablement, but also on cognitive knowledge about what users might think would fulfil or partially fulfil a goal. This knowledge can be obtained by empirical studies of the target system at hand, but it is also possible to draw upon general principles from cognitive psychology, yielding a result that is transferable between applications.

The goal structure and the action library can be changed independently of each other which will allow for a more dynamic system: the potential learning capacity of the system will be larger using this approach.

References

Joost Breuker, Radboud Winkels and Jacobijn Sandberg, A Shell for Intelligent Help Systems, in IJCAI, 1987, pp. 167-173.

Chin, David N., KNOME: Modeling What the User Knows in UC, in User Models in Dialog Systems, Alfred Kobsa and Wolfgang Wahlster (eds), Springer-Verlag, Berlin-New York, 1989, pp. 74-107.

Dahlbäck, Nils, Representations of Discourse - Cognitive and Computational Aspects, Linköping Studies in Arts and Science 71, Linköping Studies in Science and Technology 264, Doctoral thesis at Linköping University, 1991

Joshi, Aravind K., Mutual Beliefs in Question-Answering Systems, in N. V. Smith (ed.), Mutual Knowledge, Academic Press, London 1982

Jussi Karlgren *The Interaction of Discourse Modality and User Expectations in Human-Computer Dialog*, Licentiate Thesis at the Department of Computer and Systems Sciences, University of Stockholm, 1992

Kaplan, S. J., Cooperative Responses from a Portable Natural Language Database Query System, Artificial Intelligence, 19:2, 1982, 165-188.

Kass, Robert, Building a User Model Implicitly from a Cooperative Advisory Dialog, User Modeling and User-Adapted Interaction, 1:3, 1991.

Konolige, Kurt and Pollack, Martha, Ascribing Plans to Agents: Preliminary Report, in IJCAI, 1989, pp. 924-930.

Malone, Thomas W., Grant, Kenneth R., Turbak, Franklyn A., Brobst, Stephen A., and Cohen, Michael D., Intelligent

Information-Sharing Systems, Communications of the ACM, 30:5, 1987.

Pollack, Martha, A Model of Plan Inference that Distinguishes between the Beliefs of Actors and Observers, in Proc. of the 24th Annual Meeting of ACL, 1986.

Wilensky, Robert, Planning and Understanding: A Computational Approach to Human Reasoning, Addison-Wesley Publ. Comp., Reading, Massachusetts, 1983.