Technical Report

UCAM-CL-TR-154
ISSN 1476-2986

Number 154

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

A natural language interface to an intelligent planning system

I.B. Crabtree, R.S. Crouch, D.C. Moffat,
N.J. Pirie, S.G. Pulman, G.D. Ritchie, B.A. Tate

January 1989

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

https://www.cl.cam.ac.uk/

This project is supported by the Alvey Directorate and the
Science and Engineering Research Council (Alvey IKBS 179,
SERC GR/D/83507). This paper is to appear in the
Proceedings of the 1988 UK IT Conference, Swansea.

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

https://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

AIMS OF PROJECT

An intelligent planning system is an example of a software aid which, although developed by specialists in artificial intelligence, is intended to be used by non-programmers for a wide variety of tasks. That is, the domain-specific definition of requirements, resources, etc. for a particular application will be specified by people who are trained in their own trade or profession but who are unlikely to be acquainted with the technicalities of automatic planning systems. There is therefore a need for a communication medium which allows the application specialist, and the non-expert user of the eventual domain-tailored system, to specify their needs without knowing any of the details of the planning notation or the actual operations of the planning system.

This kind of system is one where the 'mice and menus' approach is unlikely to be able to provide a very flexible interface, since it is characteristic of the type of interaction that one would like to be able to have with a planner that the range and type of potential queries is not predictable in advance, and thus not reducible to choices within some predetermined set of options. Furthermore, it is often desirable to have the planner try out a range of hypothetical or counterfactual situations that could not be represented in any obvious graphical form: some kind of language, either artificial or natural, is a necessity here.

The aim of this project is to experiment with the use of English language as the medium of communication. The kind of system we would like to evantually be able to build would be one where a user could use the planner to organise some type of external activity, perhaps trying out several alternative scenarios based on differing assumptions, and then interact with the system during the course of execution of the resulting plans, making adjustments when parts of the plan turn out for some unforeseen reason to be unachievable.

The following are examples of the kind of query we would like to be able to handle, in our current sample domain concerning field engineers repairing equipment faults.

1. Has the fault at Ipswich been fixed?
2. Will Brown go to Ipswich from Base before anybody does Job2?

3. What jobs will be worked on while Smith is at Ipswich?

Queries like this require that notions of time, in relation to particular points in the current plan, or to the current state of execution of the plan, should be handled correctly.

4. Could Brown do Smith's job at Ipswich?
5. If Smith goes to Ipswich, could Brown go to Martlesham?

Here the interpretion of the queries depends on the ability to consider alternatives to the current plan which still allow the goals to be achieved. It would also be useful to be able to frame queries asking for explanation or justification for some aspects of plans:

6. Why does Smith have to fix the fault at Ipswich before he goes to London?

During the first year of this project (1987) we have developed a simple prototype system which is capable of handling queries like those in 1 - 3. In the next section of the paper we describe this system, and in the final section we will discuss some of the problems that arise in trying to extend this to a successor system capable of handling the remaining type of query, those in 4 - 6.

THE PROTOTYPE SYSTEM

A very simple prototype ('Version 1') was developed during the first year), which processes queries about a single plan produced by the NONLIN planner. This prototype system consists of several modules: the Planner, the Sentence Analyser, the Translator, and the Query Evaluator. The medium of communication between several of these modules is what we call a 'General Planning Language' (GPL): a language in which plans and goals can be represented, queried and reasoned about. GPL allows the expression of planning-oriented notions such as occurrences of actions, effects and orderings of actions, and initial world state, all of which can be the subject of two different types of query, corresponding linguistically to 'yes-no' and 'wh-' questions.

The actual planner used in Version 1 is Tate's NONLIN developed at Edinburgh some ten years ago (Tate, (5)). (A clear description of this type

of planner is given in Charniak and McDermott (1), section 9.3). NONLIN uses a library of 'hierarchical', or composite, actions; these are used to 'expand' a plan from a high level specification of the requirements to a more detailed set of actions that must be performed to achieve the objectives. The library of actions, the statement of requirement for a plan and the initial conditions of the world can be specified via a 'Task Formalism' language. It produces 'non-linear' plans, in that they only minimally specify the order in which actions must be executed to achieve the goals. The 'intent' of any specific action with respect to the goals and sub-goals that are desired at points in the plan is embedded in the so-called 'Goal Structure' of the plan. This information is used to ensure that interacting effects and conditions are spotted and can be corrected by the introduction of necessary action orderings.

For the first simple prototype, the application domain concerned the scheduling of telephone engineers to fix faults in the network. Men are allocated work depending on their location and special skills. The plan which forms the subject matter of the dialogue is first created by the NONLIN planner from a specification (in NONLIN's own task formalism), displayed graphically, and converted via a planner specific routine to an equivalent representation in GPL. Until a new session is begun, there is no further invocation of NONLIN. Thereafter, queries about this plan can be typed in. A 'Sentence Analyser' parses the input and produces a 'logical form' (a symbolic representation of its meaning). This logical form, together with some (at present rudimentary) contextual information is then passed to a 'logical form to GPL' Translator which converts this into a GPL query form. This query is then evaluated by the Query Evaluator against the current plan. The results of this evaluation (usually a list of variable bindings) is printed out directly: no effort has been expended so far on producing linguistically natural output - in itself a considerable research problem.

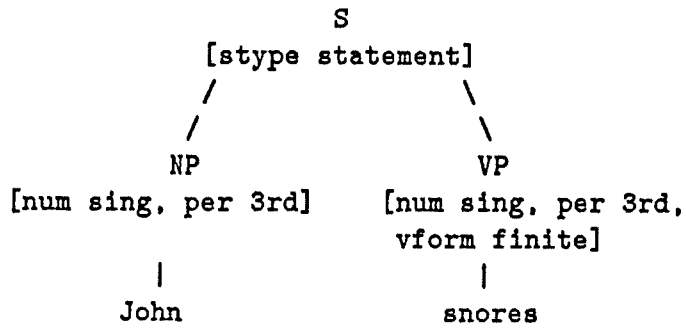
The Sentence Analyser accepts English sentences and produces logical forms representing the literal meaning of the sentences. First, the sentence is parsed using a unification enriched context free grammar, a lexicon and a bottom up chart parser. This produces the syntactic structure of the sentence, which is used to determine what the logical form of the sentence should be. Each syntactic rule in the grammar has one or more corresponding semantic rules saying how the meanings of the constituents should be

combined to give the meaning of the whole. (When there are several semantic rules, the instantiations of syntactic features determine which one is applicable). To give a simple example, consider the grammar rule

```
syntax:  S[stype statement] -->
           NP[number @num, person @per]
           VP[number @num, person @per, vform finite]

semantics: S' = NP'(VP')
```

This says that a sentence can consist of a noun phrase followed by a verb phrase, provided the NP and VP agree in the values given to their number and person features and that the VP is finite. Agreement between features is enforced by unification: feature values beginning with '@' are variables. The semantic part of the rule says that given a sentence built up from an NP and a VP combined in this way, form the meaning of the sentence (S') by applying the meaning of the NP (NP') to that of the VP (VP'). For a very simple sentence we might have:



NP' = [λP. P(john)]
VP' = snore
S' = [λP. P(john)] (snore)
= snore(john)

The meanings of the constituents are expressions in a typed higher order logic based on a simplified form of Montague's intensional logic (see Dowty, Wall, and Peters (2)). These meanings are ultimately built up from logical expressions assigned as the meanings of individual words by function application or composition. In most cases, when all the lambda (beta) reductions have been carried out, the result is a first order expression.

The 'logical-form-to-GPL' translator is implemented within a forward and backward chaining inference system which recursively goes through a logical form. The translation rules for atomic clauses are expressed as conditionals of the form:

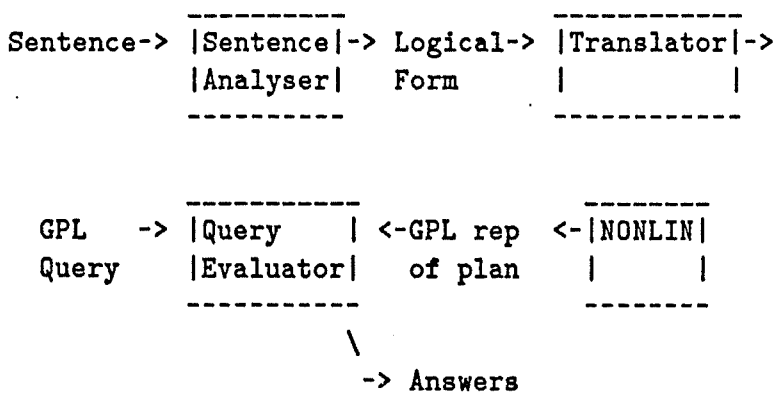
IF ⟨conditions⟩ THEN translates(⟨GPL-expr⟩, ⟨logical-form⟩)

This allows the translation to take into account contextual factors (which will be among the 'conditions' previously asserted) and allows virtually arbitrary rewritings of a logical form into a GPL representation, via rules which are in a purely declarative and readable form.

Since GPL includes (approximately) the inferential capabilities of first-order logic, the evaluation of a query against the representation of a single plan by the Query Evaluator is in fact a form of theorem-proving. Version 1 actually has two different types of Query evaluator: one which interprets quantifiers in a way reminiscent of Woods (6) 'procedural semantics' for the Lunar system and one which turns the query into a normal form

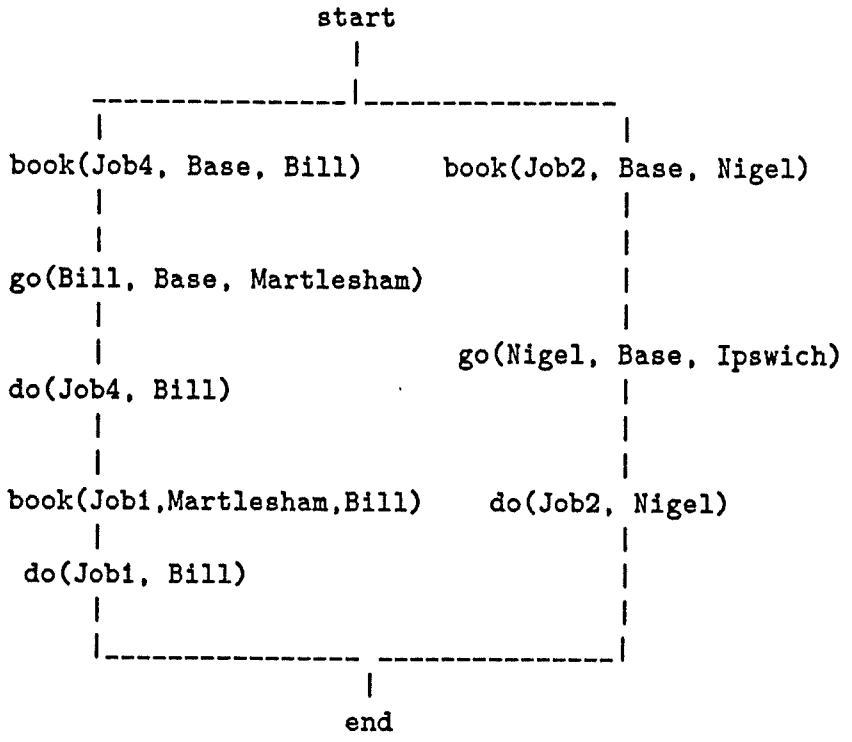
and uses unification and backward chaining of a familiar sort. However, the intention is that 'procedural evaluation' of a more far-reaching kind (procedurally evaluated operators and predicates) will also be used in future versions of GPL to allow more complex types of interaction with the back-end planning system in the course of evaluating a query.

The overall structure of Version 1 can be represented as:



The initial prototype was tested using (extremely simple) plans in our sample domain of engineers fixing faults at telephone exchanges. If we consider the simple plan drawn below and pose the following queries before any of the plan is executed, we would get these responses:

)What does Bill do?
Job1,Job4
)Who will do Job2?
Nigel
)Will anyone go from Base to Ipswich?
Yes
)Will Nigel go to Ipswich from Base before Nigel does Job2?
Yes



As can be seen, the questions that naturally arise in this, as in most planning domains, concern the ordering of various actions in the plan and when they happen. The linguistic analysis has therefore concentrated on providing an account of tense and temporal reference. The range of constructions covered includes: simple past, present and future tenses, complex tenses, aspects like the progressive and the perfect, temporal adverbials like 'at 2pm', and some temporal connectives like 'before', 'after' and 'when'.

The basic assumption behind the semantic treatment of time is that the logical forms of sentences refer to states and events: e.g. the event of someone going from Base to Ipswich, or the state of someone being in Ipswich. The various temporal constructions place restrictions on the time intervals during which these states or events can occur. The simplest sort of restriction is imposed by the tenses. The past tense says that the state or event must occur at some point prior to the time at which the sentence was uttered. (The time of utterance is in turn related to some point during

the presumed execution of the plan). Adverbials like 'at 2pm' and 'yesterday' can further narrow down the range of times at which the event could have occurred. Temporal connectives like 'before' serve to mutually restrict two event times: for instance, in 'Bill went to Ipswich before he went to Martlesham', both events are past and one is constrained to occur before the other.

As a concrete illustration, the logical form assigned by the sentence analyser to the second query, 'Who will do Job2?', is (variables begin with upper case, constants with lower case):

```
(whq (some (WH Ev EvTime FutTime)
           (and (and (do WH job2 Ev)
                     (time-of Ev EvTime))
                (and (precedes start-of-day FutTime)
                     (during EvTime FutTime)))))
```

i.e. 'is there some WHperson, Event, EventTime, and FutureTime such that the Event is the carrying out by the WHperson of job2 at the time specified by EventTime, where FutureTime must be later than the start-of-the-day (i.e. the time of the utterance), and the EventTime must occur within the FutureTime interval'

The corresponding GPL query is:

```
find([WH],
     exists(type(X, node),
            exists(type(WH, person),
                   and(nameof(X, done(job2, WH)),
                      follows(X, start-of-plan)))).
```

i.e. 'return an instantiation for WH in: is there in the plan a node X, and a person WH such that X is labelled as 'job2 completed by WH' and X is after the initial node of the plan?'

As can be seen, the GPL translation usually omits information present in the logical form. In general, the meanings of sentences carry much more information about time than the rather simple treatment presupposed by this planner and this domain. There can be considerable amounts of

inference involved in going from a linguistically expressed meaning to the most appropriate formulation of the query in GPL for a particular plan.

THE NEXT VERSION

We are currently working on the design of the second prototype, which will remedy some of the weaknesses of the first version and extend the linguistic coverage in various ways. However, there are many difficult research issues to be tackled here.

One issue concerns the exact status of our intermediate representation language, GPL. It would be nice to think of (the final version of) GPL as bearing the same kind of relation to planners as a database query language like SQL does to databases. We would like the language to provide in a precise sense a definition of the range of possible interactions it is possible to have with a planning system. GPL would have a clear denotational or operational semantics, which any 'implementation' of it in terms of a particular planner would have to respect. Then the process of moving our system from one planner to another operating of the same factual domain would be simply that of implementing GPL using the basic mechanisms of the planner in question. (There are of course a different set of problems arising when the domain is different too).

However, the analogy is not one which holds as firmly as we might like. For one thing, database theory and practice is much more advanced in standardisation than is the case with planning. Only if some reasonable convergence on the range of operations and type of structures that planners deal with emerges will it be possible to provide a definition which has a chance of being fairly generally applicable. As it is, there is a wide variety of types of system being used for planning: deductive, procedural, agentless, multi-agent, those that include plan execution and monitoring, or on-the-fly replanning, etc. etc. There seems to be no great measure of agreement (at any useful level of detail) about the primitive concepts of planning systems in general, other than those we have already tried to incorporate. Thus any version of GPL that we are likely to come up with in the next few years is quite likely to be overtaken by events in the world of planning research. Nevertheless, we hope to be able to do something useful with at least the range of concepts that are likely to remain stable: anything recognisable as

a planner is presumably going to involve initial and goal conditions, actions, sequencing of actions, and so on.

A second point where the analogy begins to break down, concerns the relationship of the planner itself to the evaluation of queries. Whereas with database systems the range of possible (sensible) queries is more or less limited to the contents, and to some degree, the organisation of the database, with a planner there are far more things that could sensibly and usefully be required. For some of these type of interactions, there may be little or nothing inside the actual planner for the front end system to use. In fact, NONLIN in our present system provides an example of this: there is actually no explicit temporal information at all in a NONLIN plan. The partial ordering of nodes in a plan places some constraints on how the plan could be temporally executed, but all of this is neutral as to particular approaches to the representation of time. Nevertheless, we want to be able to ask and answer questions about temporal ordering of jobs to be done, and so on. In our current system, this has either to be inferred from domain information, or (as is actually done) treated in an oversimplified manner by associating nodes in a plan with temporally located events. Whereas in this application this is fairly satisfactory, it is not difficult to imagine other applications in which consumption of time resources were important, where this kind of temporal information would have to be superimposed on the actual planner in a way fairly unconnected with how the plans were actually arrived at.

In some more extreme cases, it might be in any case be information straightforwardly about the nature of the domain which is needed to answer the question, information that is totally irrelevant for the operation of the planner itself and thus not represented in it. In our current system we would have to represent this knowledge inside GPL, for the not very good reason that there is nowhere else for it to go. But enriching GPL so as to encompass the ability to answer such questions would not, we feel, be a good idea: a general purpose knowledge representation formalism should not be confused with a planning-specific language. The next version of the system will have to develop some principled approach to the problem of how domain-relevant information which is not planner-relevant is represented and used.

The current linguistic coverage of the system is adequate for the 'one shot'

exchanges permitted by Version 1, although the absence of pronouns or any reference to prior context often makes for some rather linguistically unnatural exchanges. Extending the linguistic coverage to include pronouns and ellipsis, and thus to cover more extended dialogues presents some problems, but nothing that is specific to this application, and we are confident that the next version of the system will be able to handle a wide range of types of pronominal usage, and some useful types of ellipsis. However, we will probably impose some more or less arbitrary limit on the amount of prior context that is considered relevant at any point in the dialogue to avoid complexities of 'local' vs. 'global' focus.

A more challenging set of issues is presented by the proposed extension to modal and conditional sentences. Our initial approach to this is based on the well-known Kripke (3) possible worlds semantics for modal logic, and Stalnaker's (4) development of this for conditionals. Informally, worlds or states of affairs are taken to be related by 'accessibility'. A sentence of the form 'possible p' is true if there is some accessible alternative to the actual world in which p is true. For conditionals, we must assume also that it makes sense to talk of a 'similarity' relationship between worlds. Then a sentence of the form 'if p then q' is true if in the world most similar to the actual world in which p is true, q is also true.

We can substitute plans for worlds and get a first approximation to a reasonable interpretation for questions like our earlier 4:

4. Could Brown do Smith's job at Ipswich?

in terms of a query as to whether there is a plan satisfying all the goal conditions of the current plan, but in which Brown does the job at Ipswich that Smith is currently scheduled to do. Similarly, 5:

5. If Smith goes to Ipswich, could Brown go to Martlesham?

will be interpreted as a query as to whether there is a plan satisfying all the goals of the current plan, but in which Smith goes to Ipswich and Brown goes to Martlesham. (Notice that if there is not, a fully cooperative system should tell us whether this is because there is no plan satisfying the antecedent, or whether there is, but all such plans necessitate Brown not going to Martlesham).

Clearly, this is a simple and intuitively satisfying treatment. Different modalities (logical possibility, practical possibility in the domain, physi-

cal possibility) can be handled by having different accessibility relations between plans. Unfortunately, of course, the accessibility and similarity relations, which are taken as primitive undefined notions in the formal semantic treatments cited, have to be actually computed for our suggested use of them to be possible. This raises many issues of principle and of practice. To say the least, it is not obvious what is going to count as the relevant notion of 'similarity' between plans.

'Why' questions present a further host of difficulties. A question like 6:

6. Why does Smith have to fix the fault at Ipswich before he goes to London?

is most unlikely to be able to be answered on the basis of anything that an existing planner can do. The reasons why particular plans are the way they are can vary enormously: the control strategy of the planner, the requirements of some other distant part of the plan, or real world constraints on the domain. Most planners discard information about how they arrived at the eventual plan, and so information of the first two types may not be available. Domain specific information raises all the familiar problems of what constitutes a reason or causal explanation of something at the appropropriately relevant level. For example, logically valid answers to 6 include:

> He doesn't.
> Because otherwise the fault will not be fixed.
> Because otherwise he will not go to Ipswich.
> Because Ipswich and London are different places.
> Because he can't fix a fault in Ipswich if he is in London.
> Because all the valid plans below a given threshold of complexity order the nodes that way.
> etc. etc.

At present it appears unlikely that we are going to be able to do anything very sophisticated in answering 'why' questions in the near future.

REFERENCES

1. Charniak, E and McDermott, D. (1985) 'Introduction to Artificial Intelligence', New York: Addison-Wesley.
2. Dowty, D., Wall, R., and Peters, S. (1981) 'An Introduction to Montague Semantics', Dordrecht: D. Reidel Publishing.
3. Kripke, S. (1963) 'Semantical Analysis of Modal Logics', I, *Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik*, *9*, 67-96.
4. Stalnaker, R. 1968 'A Theory of Conditionals', in N. Rescher, ed. *Studies in Logical Theory*, Oxford: Basil Blackwell.
5. Tate, A (1977) 'Generating Project Networks', IJCAI-77, Cambridge, Ma. USA.
6. William A. Woods, W. A. (1986) 'Semantics and quantification in natural language question answering' in Barbara J. Grosz, et al (eds) *Readings in Natural Language Processing*, Los Altos: Morgan Kaufmann, 205-248.