Generic Plan Recognition for Dialogue Systems

George Ferguson, James F. Allen

University of Rochester
Rochester, NY, 14627-0226

ABSTRACT

We describe a general framework for encoding rich domain models and sophisticated plan reasoning capabilities. The approach uses graph-based reasoning to address a wide range of tasks that typically arise in dialogue systems. The graphical plan representation is independent of but connected to the underlying representation of action and time. We describe types of plan recognition that are needed, illustrating these with examples from dialogues collected as part of the TRAINS project. The algorithms for the tasks are presented, and issues in the formalization of the reasoning processes are discussed.

1. Introduction

Plan recognition is an essential part of any dialogue system. Traditional approaches to plan recognition are inadequate in one of two ways. Those that are formally well-specified tend to be highly restricted in the phenomena they can accomodate and are therefore unsuitable for a general purpose dialogue system. On the other hand, the heuristically-motivated systems have been difficult to formalize and hence to understand. In both cases, the representation of plans is insufficient for a collaborative dialogue-based system.

The research reported here is part of the TRAINS project [1]. The goal of this project is an intelligent planning assistant that is conversationally proficient in natural language. In this paper we concentrate on the plan recognition procedures of the domain plan reasoner component of the system.

As examples of the phenomena that arise in discourse and affect plan recognition, consider the following utterances gathered from TRAINS dialogues:

1. Utterances that suggest courses of action, *e.g.*,
   (a) Send engine E3 to Dansville.
   (b) Move the oranges to Avon and unload them.

   This is the prototypical case studied in the literature, and most systems are limited to handling only this case.
2. Utterances that identify relevant objects to use, *e.g.*,
   (a) Let's use engine E3.
   (b) There's an OJ factory at Dansville.

   The second sentence is an example of an indirect suggestion to use the OJ factory.
3. Utterances that identify relevant constraints, *e.g.*,
   (a) We must get the oranges there by 3 PM.
   (b) Engine E2 cannot pull more than 3 carloads at a time.
4. Utterances that identify relevant lines of inference, *e.g.*,
   (a) The car will be there because is it attached to engine E1.
5. Utterances that identify goals of the plan, *e.g.*,
   (a) We have to make OJ.
6. Utterances that introduce complex relations, *e.g.*, purpose clauses such as
   (a) Use E3 to pick up the car.
   (b) Send engine E3 to Dansville to pick up the oranges.

Our approach to plan reasoning is motivated by examples such as these. It is a *generic* approach because the details of the algorithms do not depend directly on properties of the underlying knowledge representation. Rather, the approach assumes that certain operations are exported by the underlying reasoner (such as entailment, $\models$), and it uses these to validate plan reasoning steps.

We first describe our representation of plans and its connection to the underlying knowledge representation scheme. We then present plan recognition algorithms for the dialogue phenomena and we discuss how they interact with other modules of the system. Finally, we discuss related and future work.

2. Plan Graphs

We assume that the underlying knowledge representation formalism can be effectively partitioned into two types of formulas:

- *Event formulas* state that something happened that (possibly) resulted in a change in the world.
- *Fact formulas* are everything else, but typically describe properties of the world (possibly temporally qualified).

In our temporal logic,[1] the former are of the form *Occurs*(*e*) and the latter are, for example, *At*(*eng*3, *dansville*, *now*). For formalisms where there are no explicit events (*e.g.*, the situation calculus), we can extend the language—an example of this is given below.

We then define a graphical notion of plans, based on viewing them as *arguments* that a certain course of events under certain explicit conditions will achieve certain explicit goals. A *plan graph* is a graph over two types of nodes: *event nodes* are labelled with event formulas, *fact nodes* are labeled with fact formulas. These can be connected by four types of arcs:

event-fact: Achievement

fact-event: Enablement

event-event: Generation

fact-fact: Inferential

The link types correspond roughly to an intuitive classification of the possible relations between events and facts (*c.f.*, [5]). The *goal nodes* of a plan graph are its sinks, the *premise nodes* are its sources.

For example, using the temporal logic, we might have a plan graph like that shown in Figure 1(a). The functions *blk*1 and *blk*2 are *role functions* that denote objects participating in the event; the functions *pre*1 and *eff*1 are *temporal* role functions denoting intervals related to the time of the event. In a formalism such as the situation calculus, actions are terms and there is no equivalent of the *Occurs* predicate. However, we can introduce one as a placeholder, and then we might get a plan graph like that shown in Figure 1(b).

A plan graph makes no claim to either correctness or completeness. It represents an argument from its premises to its goals, and as such can be "correct," "incorrect," or neither. The previous examples are intuitively correct, for example, but are incomplete since they don't specify that the block being stacked must also be clear for the stacking to be successful.

A translation of plan graphs into a first-order logic with quotation is straightforward. With this, one can declaratively define properties of plans represented by plan graphs (such as "correct") relative to the underlying representation's entailment relation. For example, a node n in a plan graph P might be *supported* if its preconditions (nodes with arcs incident on n) are sufficient to ensure the truth of n, formally:

$$supported(n, P) \equiv \bigwedge\{\pi \mid \exists n'.\langle n', n\rangle \in P \wedge \pi = Label(n')\} \models Label(n)$$

[1]Space precludes a detailed description of this representation, see [2, 3, 4]. In what follows, we will rely on intuitive descriptions of the relevant aspects of the logic.

The antecent of the entailment must, of course, also be consistent.

Unfortunately, such an analysis is not particularly illuminating in the case of plans arising from dialogue since such plans are often too poorly specified to meet such criteria. In particular, they are often based on assumptions that the system makes in the course of its interpretation of the manager's statements. We feel that making such assumptions explicit is crucial since they often drive the discourse. To illustrate this, we will present the algorithms used by the TRAINS plan reasoner to reason with plan graphs. We will return to the issue of axiomatizing them in the final section.

3. Plan Graph Algorithms

We characterize plan reasoning for dialogue systems as search through a space of plan graphs. The termination criterion for the search depends on the type of recognition being done, as will be described presently. Since the plan graph formalism sanctions arbitrarily complex graphs labelled with arbitrarily complex formulas, searching all possible plan graphs is impossible. We therefore rely on additional properties of the underlying representation to restrict the search.

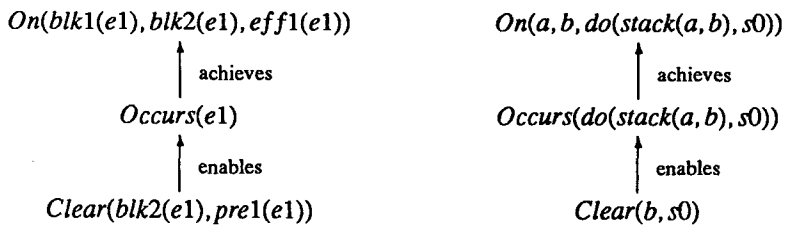
First, we assume the ability to test whether two objects (including events and facts) unify and, optionally, to determine assumptions under which they would unify. Simple objects use simple equality. In the temporal logic, two events are equal if their roles are equal. Two facts unify if there are assumptions that make them logically equivalent. This use of equality and inequality corresponds to the posting of *codesignation* constraints in traditional planners.

Second, we assume that events be defined using relations corresponding to *enablers*, *effects*, and *generators*. This should not be controversial. In the temporal logic, these descriptions can be obtained from the event definition axioms. For a STRIPS system, they correspond to the add- and delete-lists. Existing plan recognition systems use an event taxonomy, which corresponds to the generators slot. There can be multiple definitions of an event type, thereby allowing alternative decompositions or conditional effects.

The search then only considers plan graphs that reflect the structure of the event definitions, we call such plan graphs *acceptable*. In this respect, the search will only find plan graphs that agree with the assumed-shared "event library." However, information returned from failed searches can be used to guide the repair of apparent incompatibilities at the discourse level.

3.1. Incorporation

Plan recognition using plan graphs operates by searching the space of acceptable plan graphs breadth-first. The search



(a) Temporal logic plan graph

(b) Situation calculus plan graph

Figure 1: Simple example plan graphs

frontier is expanded by the function `expand-graph`, shown in Figure 2. The use of breadth-first search implements a "shortest-path" heuristic—we prefer the simplest connection to the existing plan. The plan reasoner exports several interfaces to the basic search routine, each motivated by the discourse phenomena noted at the outset. The discourse module of the system invokes these procedures to perform domain reasoning.

The procedure `incorp-event` takes as parameters a plan graph and an event (a term or a lambda expression representing an event type). For example, sentence (1a) results in the following call:

```
(incorp-event
  (lambda ?e*Move-Engine
        (And (Eq (eng ?e) ENG3)
             (Eq (dst ?e) DANSVILLE)))
  THE-PLAN)
```

where `?e*Move-Engine` is an event variable of type `Move-Engine`.

The plan reasoner first checks if the given event unifies with an event already in the plan. If so, the plan reasoner signals that nothing needed to be added to the plan (except possibly unifying assumptions, which are also indicated). Otherwise, it attempts to add an event node to the plan graph labelled with (an instance of) the event. The search continues until one or more unifying event nodes are found.[2] An example of the search in progress for the previous call is given in Figure 3, assuming that the plan already includes moving some oranges to Dansville (event `e1`). At this point (two levels of search), the given `Move-Engine` event unfies uniquely with a leaf node, so the search terminates successfully. The connecting path (double arrows) indicates that moving the engine is done to move a car that will contain the oranges, thus moving them. Note that we do not know yet which car this will be.

[2]In fact, we also use a depth bound, based on the intuition that if the connection is not relatively short, the user's utterance has probably been misinterpreted.

If more than one match is found at the same depth, the plan reasoner signals the ambiguity to the discourse module for resolution. Otherwise the connecting path is returned as a list of things that need to be added to the plan to incorporate the given event. These are usually interpreted by the discourse module as being implicatures of the user's utterance. They are added to a plan context and are used both for subsequent planning and plan recognition steps and to generate utterances when the system gets the turn.

The procedure `incorp-role-filler` is used for statements that mention objects to be used in the plan (example (2) previously). In this case, the termination criterion for the search is an event node labelled by an event that has a role that unifies with the given object (a term or lambda expression). For example, the sample sentences result in the following calls:

(2a) `(incorp-role-filler ENG3 THE-PLAN)`

(2b)
```
(incorp-role-filler
    (lambda ?x*OJ-Factory (At ?x DANSVILLE NOW))
    THE-PLAN)
```

Finally, there is the procedure `incorp-fact` that searches for a fact node that would unify with the given one. This is used for utterances like the examples (3) and (4), since the plan graph representation supports inferential (fact-fact) links. Again however, the search space of potential unifying formulas is infinite. We therefore only consider certain candidates, based on syntactic considerations. These include facts that the underlying reasoning system is particularly good at, such as temporal constraints or location reasoning. Continued use of the system will identify which inferences need to be made at this level, and which are best left to management by higher-level discourse manager routines.

3.2. Goals

These `incorp-` routines all take an existing plan graph as argument and expand it. This could come from an initial specification, but utterances like example (5) require that the plan reasoner be able to incorporate goals. There is therefore an `incorp-goal` procedure that takes a sentence and a (possibly

```
function expand-graph (g p)
  foreach n ∈ leaf nodes of g
    if n is an event node
    then e ← Label(n)
         foreach f ∈ Enablers(Type(e))
           add (plan-enabler f_e e p) to g
         foreach e' ∈ Generators(Type(e))
           add (plan-generates e' e p) to g
    else f ← Label(n)
         foreach event type T
           foreach f' ∈ Effects(T) s.t. Unify(f, f', φ)
             add (plan-enables (lambda T φ) f p) to g
```

Figure 2: Function expand-graph (subscript e indicates substitution)

empty) plan graph as arguments. If the sentence is *Occurs(e)*, then the plan graph is searched for a matching event node. If one is found, then the plan reasoner returns relevant assumptions and marks the node as a goal. Otherwise, a new event node is added to the plan and marked as a goal. Similar processing is done for fact goals. In our dialogues, the user often begins by communicating a goal that the rest of the dialogue is concerned with achieving. There is no point in doing much work for goals (beyond checking consistency) since it is likely to be immediately elaborated upon in subsequent utterances. Proper treatment of subgoals expressed as goals is part of our current work on subplans.

3.3. Purpose clauses

One construction that uses subgoals and subplans and that arises repeatedly in collaborative dialogue is the use of purpose clauses, such as example sentences (6). To accomodate these, the incorp- functions all accept an optional "purpose" argument (an event). For example, the sample sentences result in the following calls:

```
(6a) (incorp-role-filler ENG3 THE-PLAN
       :purpose (lambda ?e*Move-Car
                  (Eq (car ?e) THE-CAR)))

(6b) (incorp-event
       (lambda ?e1*Move-Engine
         (And (Eq (eng ?e1) ENG3)
              (Eq (dst ?e1) DANSVILLE)))
       THE-PLAN
       :purpose (lambda ?e2*Load
                  (Eq (obj ?e2) THE-ORANGES)))
```

If the purpose argument is present, it is first incorporated using incorp-event. If this fails, then the discourse module is notified—presumably this is some kind of presupposition failure requiring discourse-level action. If it succeeds, then the original item is incorporated but with the search *restricted* to the (sub-)plan graph rooted at the purpose event.

This simple modification of the basic plan recognition algorithms is effective at reducing the ambiguity that would otherwise be detected if the entire plan graph were searched. It is likely not adequate for all types of purpose or rationale clause, in particular those that involve the mental state of the agent rather than domain events. However, the generality of the plan graph formalism does allow it to handle many of the cases arising in our dialogues.

4. Example

To further illustrate our approach to plan reasoning, we present a sample TRAINS dialogue and describe how it is processed by the system. This dialogue was gathered from simulations where a person played the role of the system. A previous version of the TRAINS system processed the dialogue correctly—the current implementation will also once it is completed.

The manager starts by communicating her goals, making several statements, and asking a question. The system replies and makes a proposal, which is then accepted by the manager. The complete transcript is as follows:

1. M: We have to make OJ.
2. M: There are oranges at Avon and an OJ factory at Bath.
3. M: Engine E3 is scheduled to arrive at Avon at 3pm.
4. M: Shall we ship the oranges?
5. S: Ok.
6. S: Shall I start loading the oranges into the empty car at Avon?
7. S: Ok.

The manager's first utterance results in the following call to the plan reasoner:

```
(incorp-goal (lambda ?e*Make-OJ
               (Eq (agent ?e) SYSHUM))
             THE-PLAN)
```

As described above, this results in an event node begin added to the (formerly empty) plan.

Utterance (2) could be interpreted simply as statements about the world. However, since the system already knows these

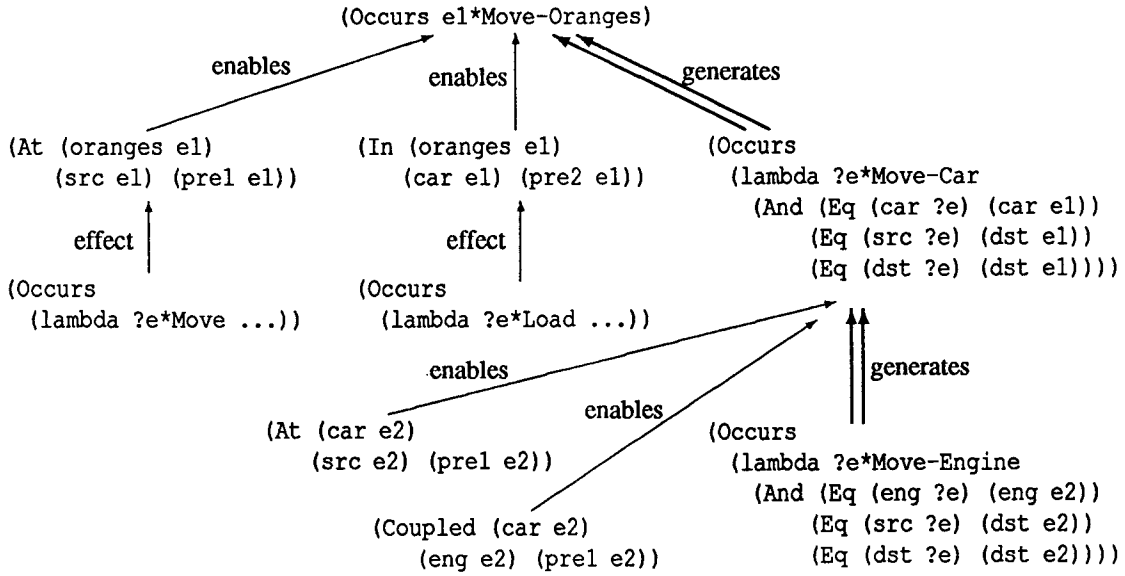


Figure 3: Incorporating moving the engine

facts (and assumes the manager knows it knows, *etc.*), the utterance is interpreted as suggesting use of the objects, resulting in the following calls:

```
(incorp-role-filler o1 THE-PLAN)
(incorp-role-filler f1 THE-PLAN)
```

The constants `o1` and `f1` are determined by the scope and reference module.

For the first call, the `Make-OJ` event has a role for some oranges, but there is a constraint that they must be at the location of the factory. While the system does not yet know which factory this will be, it can deduce that Avon cannot be that city since there is no factory there. Since the system knows only that the oranges are at Avon now (by assumption), they cannot be used directly for the `Make-OJ`. The plan reasoner therefore searches the space of acceptable plan graphs breadth-first, as described above. A connection is found by assuming that the oranges will be moved from Avon to the factory (wherever it turns out to be) via a `Move-Oranges` event. A description of this path (with assumptions) is returned to the discourse module. For the second call, the factory is acceptable as a role of the `Make-OJ` event, so only the required equality assumption is returned. This has the additional effect of determining to where the oranges are shipped (Bath).

Utterance (3) is also non-trivial to connect to the plan. We presume that the system already knows of E3's imminent arrival in the form of a sentence like `(Occurs e0*Arrive)`. Again therefore, the statement is therefore taken to suggest the use of E3 in the plan. The system can reason about the effects of the `Arrive` event, in this case that E3 will be at Avon at 3pm. Even so, there is no event with a role for an engine in the plan yet, so the space of acceptable plans is again searched breadth-first. In this case, a connection is possible by postulating a `MoveCar` event that generates the previously-added `Move-Oranges` event, and a `Move-Engine` event that generates the `Move-Car`.

The manager then makes the query (4), thereby relinquishing the turn. The dialogue module evaluates the query by calling the plan reasoner with:

```
(incorp-event (lambda ?e*Move-Oranges
                (Eq? (oranges ?e) o1)) THE-PLAN)
```

The plan reasoner finds the `Move-Oranges` event added a result of utterance (2), and indicates this to the discourse module. The system therefore replies with utterance (5), implicitly accepting the rest of the plan as well.

The plan reasoner is then called to `elaborate` the plan, during which it performs fairly traditional means-ends planning to attempt to flesh out the plan. In so doing, it attempts to satisfy or assume preconditions and bind roles to objects in order to generate a supported plan. It freely makes consistent persistence assumptions by assuming inclusion of one unconstrained temporal interval within another known one. It can ignore some details of the plan, for example the exact route an engine should take. These can be reasoned about if necessary (*i.e.*, if the human mentions them) but can be left up to the agents otherwise.

In the example scenario, many things can be determined unambiguously. For example, the oranges should be unloaded at Bath at the appropriate time, leading to an event of type `Unload`. The choice of car for transporting the oranges, however, is ambiguous: in the scenario, there is an empty car at Avon as well as one attached to E3. The plan reasoner sig-

nals the ambiguity to the discourse module, which chooses one alternative and proposes it, leading to utterance (6).

At this point the manager regains the turn and the dialogue continues until the system believes it has a mutually agreed upon plan. In this example, the manager accepts the system's suggestion, and the plan reasoner determines that the plan is ready for execution by the agents in the simulated TRAINS world.

5. Discussion

Graph-based approaches to representing plans date back to the very beginnings of work on automated planning, from Sacerdoti's procedural nets [6] to SIPE's representation of plans [7]. Often these representations reflected a combination of temporal information and knowledge about the plan. In our view, the temporal reasoning is provided by the underlying knowledge representation and the plan graph represents an *argument* that a certain course of action under conditions will achieve certain explicit goals. The earlier systems' inability to separate the plan representations from their use as data structures in planners made it difficult to predict and explain their behaviour. The plan graph formalism achieves such a separation, but the price we pay is the inability to use directly the efficient algorithms developed previously. Some of the results from the planning community on efficient algorithms can be adapted to the temporally explicit logic of events (*c.f.*, [2]). We are developing a theory of plan graphs that will provide a formal basis for many of the heuristic procedures developed previously.

With respect to plan recognition, Kautz's work [8, 9] provides a formal basis for plan recognition but only dealt with observed events fitting into a hierarchy of event types. Pollack [10] uses a formalism similar to our underlying temporal logic, but includes representations of belief and intention that are not the focus of this paper. We believe that there is a structure to plans independent of the intentions of agents, and that plan graphs seen as arguments provide the proper perspective for reasoning about them at that level.

Carberry [11] describes a model for incremental plan inference in task-related information-seeking dialogues. It uses a "context model" consisting of a tree of goals, with associated plans. Since we see the overall structure of a plan as an argument, there is no such separation in our approach, although we do treat goals specially as described previously. Her "current focused" goal and plan are analogous to our `:purpose` mechanism and to the techniques used by the language and discourse modules for determining focus. The system also uses breadth-first search with "focusing heuristics," several of which correspond to our heuristics described previously. However, the approach lacks a formal description that we believe can be provided by the plan graph formalism.

Several recent approaches to plan recognition [12, 13] rely on the use of a powerful terminological reasoner to place event types in a virtual lattice. This has the advantage that subsumption relationships (corresponding to our unification procedure) can be automatically and incrementally computed. Existing terminological reasoners, however, typically either do not allow complex objects (roles) and equality, or draw only the conclusions about subsumption that are deductively (necessarily) entailed. Neither do they compute the assumptions that would unify facts.

No existing system is as ambitious as the TRAINS domain plan reasoner in providing services required to support dialogue, from representing complex, partial and incorrect plans to providing incremental and interleaved planning and plan recognition. We are currently completing a new implementation of the procedures based on this paper. It is part of our current research to apply work on argument systems directly to justifying these plan graph algorithms in terms of a formal theory of plan graphs.

References

1. James F. Allen and Lenhart K. Schubert. The TRAINS project. TRAINS Technical Note 91-1, Dept. of Computer Science, University of Rochester, Rochester, NY, 1991.
2. James F. Allen. Temporal reasoning and planning. In *Reasoning about Plans*, pages 1–68. Morgan Kaufmann, 1991.
3. George Ferguson. Explicit representation of events, actions, and plans for assumption-based plan reasoning. Technical Report 428, Dept. of Computer Science, University of Rochester, Rochester, NY, June 1992.
4. James F. Allen and George Ferguson. Action in interval temporal logic. In *Proceedings of the Second Symposium on Logical Formalizations of Commonsense Reasoning*, pages 12–22, Austin, TX, 11–13 January 1993.
5. A.I. Goldman. *A Theory of Human Action.* Prentice-Hall, 1970.
6. E.D. Sacerdoti. *A Structure for Plans and Behaviour.* Elsevier, North-Holland, 1977.
7. D.E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, 1988.
8. Henry A. Kautz. *A Formal Theory of Plan Recognition.* PhD thesis, Dept. of Computer Science, University of Rochester, Rochester, NY, May 1987. Available as TR 215.
9. Henry A. Kautz. A formal theory of plan recognition and its implementation. In *Reasoning about Plans*, pages 69–126. Morgan Kaufmann, 1991.
10. Martha E. Pollack. Plans as complex mental attitudes. In P.R. Cohen, J. Morgan, and M.E. Pollack, editors, *Intentions in Communication.* MIT Press, 1990.
11. Sandra Carberry. *Plan Recgnition in Natural Language.* MIT Press, 1990.
12. Barbara Di Eugenio and Bonnie Webber. Plan recognition in understanding instructions. In *Proceedings of the First Intl. Conf. on AI Planning Systems*, pages 52–61, 15–17 June 1992.
13. Robert Weida and Diane Litman. Terminological reasoning with constraint networks and an application to plan recognition. In *Proceedings of KR92*, pages 282–293, Boston, MA, 25–29 October 1992.