SRI International

COMMUNICATION AND INTERACTION IN MULTI-AGENT PLANNING

December 9, 1984

Technical Note 313

By: Michael Georgeff, Computer Scientist

Artificial Intelligence Center
Computer Science and Technology Division

APPROVED FOR PUBLIC RELEASE:
DISTRIBUTION UNLIMITED

SRI Project 8871

This research was supported in part by
ONR Contract N00014-80-C-0296 and in
part by AFOSR Contract F49620-79-C-018.

This paper appeared in *Proceedings of the National Conference on Artificial Intelligence*, Washington, D.C. (1983).

SRI International®

333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 326-6200 • TWX: 910-373-2046 • Telex: 334-486

Abstract

A method for synthesizing multi-agent plans from simpler single-agent plans is described. The idea is to insert communication acts into the single-agent plans so that agents can synchronize activities and avoid harmful interactions. Unlike most previous planning systems, actions are represented by *sequences* of states, rather than as simple state change operators. This allows the expression of more complex kinds of interaction than would otherwise be possible. An efficient method of interaction and safety analysis is then developed and used to identify critical regions in the plans. An essential feature of the method is that the analysis is performed without generating all possible interleavings of the plans, thus avoiding a combinatorial explosion. Finally, communication primitives are inserted into the plans and a supervisor process created to handle synchronization.

§1 Introduction

One of the things robots and other agents need to be able to do is to organize their activities so that they can co-operate with one another and avoid conflicts. For example, we might want two robots to co-operate in building a component, one holding some part while the other attaches some other part to it, or we might want each to pursue different goals, making sure that they both don't attempt to use the same resource at the same time. One way to organize such robots is to carefully time each of their activities in such a way that this sort of co-operation and conflict avoidance is guaranteed. However, in many real-world situations, it is not possible to time events with enough accuracy to enable this approach to work, and some run-time synchronization of activities is needed. Further, because these robots like to be as autonomous as possible, pursuing their own goals at their own speed, we should not impose ordering constraints on their activities unless it is absolutely necessary. Such synchronization can only be achieved by getting the robots (or some observers) to talk to each other or to some

supervising agent.

This paper describes a relatively simple method for achieving this synchronization, given that the plans of the individual robots have already been constructed. The method also extends to single-agent planning, where one tries to achieve subgoals separately and defers decisions as to how these subplans will finally be interleaved (e.g., as in NOAH [Sacerdoti 77]). Note that we are not concerned with some of the more problematic issues in multi-agent planning, such as questions of belief or persuasion (e.g., [Konolige 82]). Similarly, the type of communication act that is involved is particularly simple, and provides no information other than synchronizing advice (cf. [Appelt 82]).

Most approaches to planning view actions (or events) as a mapping from an old situation into a new situation (e.g., [McCarthy 68, Sacerdoti 77]). However, in cases where multiple agents can interact with one another, this approach fails to adequately represent some important features of actions and events (e.g., see [Allen 81, McDermott 82]). For example, consider the action of tightening a nut with a spanner. To represent this action by the changes that it brings about misses the fact that a particular tool was utilized *during* the performance of the action. And, of course, this sort of information is critical in allocating resource usage and preventing conflicts. Wilkins [Wilkins 82] recognized this problem with the STRIPS formulation, and extended the representation of actions to include a description of the resources used during the action. However, these resources are limited to being objects, and one cannot specify such properties of an action (or action instance) as "I will always remain to the north of the building", which might help other agents in planning to avoid a potentially harmful interaction.

In this paper we show how representing actions as *sequences* of states allows us to take account of both co-operative and harmful interactions between multiple agents. We assume that the duration of an action is not fixed, and that we can only know that an action has been

completed by asking the agent that performed the action (or some observer of the action). This does not mean to say that we cannot take into account the expected duration times of actions, but rather that we are concerned with problems where this information is not *sufficient* for forming an adequate plan. For example, if some part in a machine fails, then knowing that delivery of a new part takes about 24 hours can help in planning the repair, but a *good* plan will probably want a local supervisor or agent to be notified on delivery of the part.

§2 Formalizing the Problem

We consider an action to be a *sequence* $S_1, S_2, \ldots S_n$ of sets of states, intuitively those states over which the action takes place.* The *domain* of the action is the initial set of states S_1, and the *range* is tbe final set of states S_n. The intermediate sets of states $S_2, \ldots S_{n-1}$ are called the *moments* of the action.

A *planning problem* P consists of a set of states, S; a designated set of initial states, I, in S; a set of primitive actions, A, which can be performed by the various agents operating in the domain; and a set of goal states, G, in S. For any given planning problem, a *single-agent [unconditional] plan* P is a description of a sequence of actions $a_1, a_2, \ldots a_n$ from A such that

i. a_1 is applicable to all initial states I (i.e., the domain of a_1 contains I)

ii. for all i, $1 < i \leq n$, the action a_i is applicable to all states in the range of a_{i-1}

iii. a_n achieves tbe goal G (i.e., the range of a_n is contained in G).

A *multi-agent plan* for a problem P is a collection of plans for subproblems of P which are synchronized to be applicable to all initial states I and to achieve the goal G.

We will describe the problem domain using a predicate-calculus-like representation

*More generally, an action may be a *set* of such sequences. While this generalization can easily be accommodated within tbe formalism, it needlessly complicates our exposition.

and assume that all actions satisfy the so-called "STRIPS assumption" [Nilsson 80]. Under the STRIPS assumption, all conditions that cannot be proved [under some suitable restriction] to have changed by the performance of an action are assumed to remain unchanged.

Further, we are only concerned with problems in which the components of a world state involving distinct agents are sufficiently decoupled to permit us to assume that the effects of actions of one agent are largely independent of any other. Although violation of this restriction would not affect the validity of any solutions obtained, we would then be less certain of finding solutions, even if they existed.

The representation of actions that we will use is a generalization of the standard STRIPS representation. Each action description contains a *pre-condition* and a *post-condition*, denoting the domain and range of the action. In addition, we need to represent what happens *during* the action. This is achieved by specifying an ***unordered*** set of conditions to denote the moments (intermediate state sets) of the action. We will call these conditions the *during* conditions of the action. Again, under the STRIPS assumption, all other conditions are assumed to remain unchanged during the performance of the action, unless it can be proved otherwise.

For example, here is a possible description for the blocks world action that places one block on another:

puton(x,y)

pre:	**holding(x) and clear(y)**
during:	**{ holding(x) and clear(y), holding(x) and on(x,y) }**
post:	**clear(x) and handempty and on(x,y)**

In the above problem domain, we could assume also that there was a *static domain constraint* [Rosenschein 82] saying that **holding(x)** always implies **clear(x)**.

§3 The Method

Let us assume that, given a planning problem, we have decomposed the original goal into appropriate subgoals. Without loss of generality, we will only consider decomposition into *two* subgoals. Also assume that we have separately generated plans for solving each of these subgoals (using some simple search technique, for example). Our problem now is to combine the two plans into a multi-agent plan that avoids conflicts and allows as many actions to proceed in parallel as possible.

The first thing we have to work out is the manner in which individual actions may interact with one another. Then we need to determine which of the feasible situations are "unsafe" (i.e., could lead us into deadlock) and finally we need to insert synchronization primitives into the two subplans (single-agent plans) so that these unsafe situations can be avoided.

3.1 Interaction Analysis

Our first task is to establish which situations occurring in the two single-agent plans are incompatible with one another. For example, if, in one single-agent plan, a situation occurs where block A is on top of block B, and, in the other single-agent plan, a situation occurs where block B is required to be clear, then these two situations are clearly incompatible. Similarly, if one agent expects a component of some assembly to be held by some other agent, and that other agent is not holding it, then the situations are again incompatible. We will now make this notion a little more precise.

Consider two (single-agent) plans P and Q, and let p and q be some state descriptions occurring at some point in the action sequences for P and Q, respectively. We will denote by $<p,q>$ the situation (set of states) where both p and q hold. If p and q are contradictory (i.e., we can prove that p and q cannot both be true at the same time), then of course $<p,q>$ will

denote the empty set and we will say that $<p,q>$ is *unsatisfiable*. Otherwise, we will say that $<p,q>$ is *satisfiable*.

Now consider what happens when we try to execute actions in parallel. Let us begin by describing the sequence of state sets defining an action by a *sequence* of conditions. Then, given two actions $a = p_1, p_2, \ldots p_m$ and $b = q_1, q_2, \ldots q_n$, what can we say about the way they can be executed?

Assume we are in some situation $<p_i, q_j>$. To establish feasibility and safety, we need to know what are the possible successor situations. Say that, at this given instant, action a continues next, while action b remains at its current point of execution. Then, clearly, in the next situation p_{i+1} will hold. But will q_j also hold in this new situation? In the general case, we would need to use the properties of the problem domain to determine what in fact does happen next. However, under the STRIPS assumption, we are guaranteed that q_j holds in this new situation, provided $<p_{i+1}, q_j>$ is satisfiable. Similarly, if action b proceeds before action a, then p_i will continue to hold in the new situation, provided again that this new situation is satisfiable. Thus the possible successors of the situation $<p_i, q_j>$ are just $<p_{i+1}, q_j>$ and $<p_i, q_{j+1}>$.

The STRIPS assumption is thus seen to be very important, because it allows us to determine the manner in which actions can be interleaved solely on the basis of satisfiability of the pairwise combination of the conditions defining the actions. If this were not the case, we would have to examine every possible interleaving of the actions, inferring as we went just what the successor situations were and whether or not they were satisfiable. Even without taking into account the cost of performing the necessary inferences, the complexity of this process is of order $(n+m)!/(n!\ \ m!)$, compared with a complexity of order $n \times m$ if we make the STRIPS assumption (and thus need only examine all possible pairs of conditions). Furthermore, in the general case it would not be possible to specify the during conditions as an unordered set — we

would have to specify the actual order in which these conditions occur during the performance of the action. This complicates the representation of actions and, in any case, may not be information that we can readily provide.

We are now in a position to determine how actions as a whole can be safely executed. Consider two plans $P = a_1, a_2, \ldots a_m$ and $Q = b_1, b_2, \ldots b_n$, and assume actions a_i and b_j are next to be executed.

One possibility is that actions a_i and b_j can he executed in parallel. Because we have no control over the rates of the actions, all interleavings of the actions must therefore be possible. Under the STRIPS assumption, this will be the case if, and only if, all situations $<p, q>$ are satisfiable, where p and q are any condition defining the actions a_i and b_j, respectively. Such actions will be said to *commute.*

Alternatively, action a_i could be executed while b_j is suspended (or vice versa). For this to be possible, we require that the preconditions of b_j he satisfied on termination of some action that follows a_i in plan P. We will in fact impose somewhat stronger restrictions than this, and require that the preconditions of b_j be satisfied on termination of a_i itself.* This amounts to assuming that the preconditions for one of the actions appearing in one of the plans are unlikely to be achieved by the other plan (or that, in worlds where interactions are rare, so is serendipity). It is clear that, for actions satisfying the STRIPS assumption, and under the restriction given above, action a_i can be executed while b_j is suspended if, and only if, (1) the situation consisting of the preconditions of both actions is satisfiable and (2) the situation consisting of the postcondition of a_i and the precondition of b_j is satisfiable. If actions a_i and b_j have this property, we will say that a_i has *precedence* over b_j.

---

*This is simply a restriction on the ***solutions*** we allow, and simplifies the analysis. The fact that one of the plans might fortuitously achieve the preconditions for one or more actions in the other plan does not ***invalidate*** any solution we might obtain — it just means that the solution we obtain will not make constructive use of that fact.

Note that it is possible for both actions to have precedence over each other, meaning that either can be executed while the other is suspended. Also, neither action may have precedence over the other, in which case neither can be executed. In the latter case, we will say that the actions *conflict.*

In problem domains that are best described by predicate calculus or some parameterized form of action description, the above conditions need to be determined for the *instances* of the actions that occur in the particular plans under consideration. However, in many cases these conditions can be established for the primitive actions, irrespective of the particular instance. For example, in the blocks world, **handempty** conflicts with **holding(x)**, irrespective of the value of **x**. Furthermore, one can often establish relatively simple *isolation conditions* under which classes of actions will or will not commute irrespective of the particular instance. Thus although the deductions necessary for determining satisfaction of situations may be time consuming, much of the analysis can be done *once only* for any given problem domain.

3.2 Safety Analysis

We can now use these properties to set up the safety conditions for individual actions. Consider two plans $P = a_1, a_2, \ldots a_m$ and $Q = b_1, b_2, \ldots b_n$. Let $begin(a)$ denote the beginning of an action a and $end(a)$ the termination of the action. Let the initial conditions of the plans P and Q be denoted by $end(a_0)$ and $end(b_0)$, respectively. For each pair of actions a_i and b_j occurring in P and Q we then have the following:

i. If a_i and b_j do not commute, then $<begin(a_i), begin(b_j)>$ is unsafe.

ii. If a_i does not have precedence over b_j, then $<begin(a_i), end(b_{j-1})>$ is unsafe.

The set of all such unsafe situations is called the *interaction set.*

However, we still need to determine whether these unsafe situations give rise to other

unsafe situations — that is, we must determine which of all the possible situations occurring in the execution of the plans P and Q could result in deadlock. The rules that govern the safety of a given situation s are as follows:

i. If $s = <begin(a_i), begin(b_j)>$, then s is unsafe if either successor situations are unsafe.

ii. If $s = <begin(a_i), end(b_j)>$, then s is unsafe if $<end(a_i), end(b_j)>$ is unsafe.

iii. If $s = <end(a_i), end(b_j)>$, then s is unsafe if both successor situations are unsafe.

iv. Together with those situations occurring in the interaction set, these are all the unsafe situations.

Unfortunately, to use these rules to determine which of all feasible situations are unsafe requires the examination of all possible interleavings of the actions comprising the plans, and the complexity of this process increases exponentially with the number of actions involved. However, in the kinds of problem domain that we are considering, actions rarely interact with each other, and as a result long subsequences of actions often commute. The following theorem, which is not difficult to prove, allows us to make use of this fact.

Commutativity Theorem. *Let $a_1, a_2, \ldots a_m$ be a [consecutive] subsequence of actions in a plan P and $b_1, b_2, \ldots b_n$ be a subsequence of actions in a plan Q. If all the actions a_i, $1 \leq i \leq m$, commute with the actions b_j, $1 \leq j \leq n$, then all possible situations occurring in all possible interleavings of these sequences will be unsafe if, and only if, the situations $<end(a_m), begin(b_1)>$ and $<begin(a_1), end(b_n)>$ are unsafe. Further, all situations occurring in all interleavings of these sequences will be safe if, and only if, $<end(a_m), end(b_n)>$ is safe.*

This theorem means that, if any two subsequences of actions commute with each other, then we need only consider those situations that occur on the "boundaries" of the sequences. Exactly what states within those boundaries are safe and unsafe depends only on the safety or otherwise of the boundary states, and this can be determined in a straightforward manner. As commutativity is common when interactions are rare, this result allows us to avoid the

exploration of a very large number of interleavings and to substantially reduce the complexity of the problem. In particular, actions that commute with all actions in the other plan can simply be removed from consideration.

We will now use these results as a basis for our method of safety analysis. Assume we have constructed two single-agent plans and have performed the interaction analysis. All references to actions that commute with the other plan in its entirety (i.e., which do not appear in the interaction set) are removed from the plans, and the beginning and termination points of the remaining actions are explicitly represented. We will say that the resulting plans are *simplified.* Then, beginning with the initial situation, the conditions of safety given above are applied recursively to determine all situations that are feasible yet unsafe. However, whenever we reach a situation where following subsequences of actions commute, we use the commutativity theorem to avoid the explicit exploration of all possible interleavings of these subsequences.*

3.3 Interaction Resolution

The set of unsafe situations is next analyzed to identify contiguous sequences of unsafe situations. These represent *critical regions* in the single-agent plans. Once these critical regions have been determined, standard operating-system methods can be used to enforce synchronization of the actions in the plans so that conflicting critical regions will not both be entered at the same time.

We will use CSP primitives [Hoare 1978] for handling this synchronization. A program in that formalism is a collection of sequential processes each of which can include interprocess communication operations. Syntactically, an interprocess communication operation names the

*In fact, the analysis of safety can be further simplified. These details need not concern us here, our intention being primarily to establish the importance of the STRIPS assumption and the commutativity theorem to avoid a combinatorial explosion.

source or destination process and gives the information to be transmitted. In Hoare's notation, the operation "send s to process P" is written

P!s

and the operation "receive s from process P" is

P?s

Semantically, when a process reaches a communication operation, it waits for the corresponding process to reach the matching communication operation. At that point the operation is performed and both processes resume their execution.

The synchronization is achieved as follows. At the beginning and end of each critical region R we set up a communication command to a supervisor S, respectively **S!begin-R** and **S!end-R**. The supervisor then ensures that no critical regions are allowed to progress at the same time. Placing the communication commands in the original single-agent plans is clearly straightforward. So all we now have to do is construct the scheduler, which is a standard operating-systems problem.

3.4 Example

We will consider an example where two robots are required to place some metal stock in a lathe, one making a bolt and the other a nut. Only one robot can use the lathe at a time.

We will not formally provide the details of the actions and the problem domain, but only sufficient to give the idea behind the analysis and the solution. The fact that the lathe can only be used by one robot at a time is represented as a static constraint on the problem domain.

The actions are informally as follows:

move1: agent 1 moves to the lathe

move2: agent 2 moves to the lathe

place1: agent 1 places metal stock in lathe

place2: agent 2 places metal stock in lathe

bolt1: agent 1 makes a bolt

nut2: agent 2 makes a nut

end1: agent 1 moves to end

end2: agent 2 moves to end

The preconditions and during conditions for actions **bolt1** and **nut2** include the constraint that the lathe must be in the possession of the appropriate agent, as do the postconditions and during conditions for actions **place1** and **place2**.

Assume that a simple planner produces the following single-agent plans:

move1 → place1 → bolt1 → end1

move2 → place2 → nut2 → end2

The following precedence and commutativity properties can then be established:

i. actions **bolt1** and **nut2** conflict with one another

ii. actions **place1** and **place2** each have precedence over the other, but do not commute.

iii. action **bolt1** has precedence over **place2**, but not vice versa.

iv. action **nut2** has precedence over **place1**, but not vice versa.

We now proceed to determine the unsafe situations. First, the interaction set is determined:

<begin(bolt1),begin(nut2)> <begin(bolt1),end(place2)>

<end(place1),begin(nut2)> <begin(place1),begin(place2)>

<begin(bolt1),begin(place2)> <end(place1),begin(place2)>

<begin(place1),begin(nut2)> <begin(place1),end(place2)>

We next form the simplified solutions:

begin(place1) → end(place1) → begin(bolt1) → end(bolt1)

begin(place2) → end(place2) → begin(nut2) → end(nut2)

Then we perform the safety analysis, which, in this case, returns the set of unsafe situations unchanged from the interaction set. On concatenating consecutive elements, we get only two critical regions: **begin(place1) → end(bolt1)** conflicts with **begin(place2) → end(nut2)**.

Finally we insert CSP commands into the original plans:

Solution for agent 1 (P)

move1 → S!begin(place1) → place1 → bolt1 → S!end(bolt1)→ end1

Solution for agent 2 (Q)

move2 → S!begin(place2) → place2 → nut2 → S!end(nut2) → end2

Solution for the synchronizer (S)*

```
[    not N ; P?begin(place1)  →  M := true
 [] not M ; Q?begin(place2)  →  N := true
 [] true ; P?end(bolt1)      →  M := false
 [] true ; Q?end(nut2)       →  N := false]
```

Both **M** and **N** are initially set to "false".

The solution obtained is, of course, the obvious one. Both agents must advise the supervisor that they wish to put stock in the lathe, and can only proceed to do so when given permission. Both agents must also advise the supervisor when they have finished with the lathe. On his part, the supervisor makes sure that only one agent at a time is putting stock

*The form "[] <guard> → <command>" is a *guarded command* (see [Hoare 78]), and the command following the symbol "→ " can only be executed if the execution of the guard (i.e. the boolean expression and the input command preceding "→ ") does not fail.

into the lathe and using it. Notice that the synchronizer allows *any* interleaving or parallel execution of the single-agent plans that does not lead to deadlock. Further, the synchronizer allows the plans to be continually executed, which is useful for production-line planning.

Although the problem described above involved the avoidance of harmful interactions (mutual exclusion), the method can equally well be applied to problems that require co-operation between agents. The reason is that unless the actions are synchronized to provide the required co-operation, situations will arise which are unsatisfiable. For example, if two agents are required to co-operate to paint a block of wood, one holding the piece and the other painting it, then any situation where one agent was painting the wood while the other was *not* holding it would be unsatisfiable.

The multi-agent plan synthesizer described in this paper has been used to solve a number of tasks involving both co-operation and interaction avoidance. These problems include two arms working co-operatively to bolt subassemblies together, some typical blocks world problems requiring "non-linear" solutions, and various "readers and writers" problems.

§4 Conclusions

We have presented a simple and efficient technique for forming flexible multi-agent plans from simpler single-agent plans. The critical features of the approach are that

i. actions are represented as sequences of states, thus allowing the expression of more complex kinds of interaction than would be possible if simple state change operators were used, and

ii. the STRIPS assumption and commutativity conditions are used to avoid the explicit generation of all possible interleavings of the actions comprising the plans, thus avoiding a combinatorial explosion.

While the approach does not guarantee solutions to some classes of problem involving

complex interactions between single-agent plans, it has wide applicability in many real-world settings, such as in automated factories and co-operative robot assembly tasks. Future work will extend the formalism to include conditional plans and hierarchical planning techniques.

Acknowledgment

The author wishes to acknowledge the contribution of Graham Eddy in helping to clarify the ideas in this paper.

References

[1] Allen, J.F., "A General Model of Action and Time", University of Rochester, Comp. Sci. Report TR 97, 1981.

[2] Appelt, D. "Planning Natural Language Utterances", in *Research on Distributed Artificial Intelligence*, Interim Report, AI Center, SRI International, Menlo Park, Ca., 1982.

[3] Hoare, C.A.R., "Communicating Sequential Processes", *Comm. ACM*, Vol. 21, pp 666-677, 1978.

[4] Konolige, K. "A First Order Formalization of Knowledge and Action for a Multiagent Planning System", in *Research on Distributed Artificial Intelligence*, 1982.

[5] McCarthy, J., in Minsky (ed.) *Programs with Common Sense*, MIT Press, Cambridge, Mass., 1968.

[6] McDermott, D., "A Temporal Logic for Reasoning about Processes and Plans", Yale University Comp. Sci. Research Report 196, 1981.

[7] Nilsson, N.J. *Principles of Artificial Intelligence*, Tioga Press, Palo Alto, Ca., 1980.

[8] Rosenschein, S. "Plan Synthesis: A Logical Perspective", *Proc. IJCAI-81*, Vancouver, Canada, pp. 331-337, 1981.

[9] Sacerdoti, E.D. *A Structure for Plans and Behaviour*, Elsevier, North Holland, New York, 1977.

[10] Wilkins, D.E., "Parallelism in Planning and Problem Solving: Reasoning about Resources", Tech Note 258, AI Center, SRI International, Menlo Park, Ca., 1982.