Knowledge Preconditions for Actions and Plans

Leora Morgenstern
New York University
Department of Computer Science
New York, Ny. 10012
(212)781-6539
morgenst@nyu.csd2.edu.arpa

Abstract

Agents who operate in complex environments must often construct plans on the basis of incomplete knowledge. In such situations, the successful agent must incorporate into his plans actions which obtain information. These plans are intrinsically sketchy to begin with and become more specified as the agent proceeds through his plan. A theory which allows for such flexible planning will have to provide solutions to two problems: (1) how can an agent reason that he knows how to perform an action? (Knowledge Preconditions Problem for Actions) and (2) if an agent must construct an underspecified plan due to incomplete knowledge, when can we say that he can successfully execute his plan? (Knowledge Preconditions Problem for Plans)

This paper provides solutions to both these problems. We develop a robust and highly expressive theory of action and planning which allows for actions of varying granularity, primitive as well as complex acts, multi-agent plans, and partially specified plans. We demonstrate that this theory lends itself in a natural manner to solutions to the Knowledge Preconditions Problems.

1. Introduction

An agent who operates in a complex environment must often construct plans on the basis of incomplete knowledge. Agents frequently don't know detailed procedures for the tasks they set out to accomplish. They likewise generally fail to keep up with a piece of information that is constantly changing, such as the location of a moving object. These gaps in knowledge mean that an agent may not have enough information to do the task that he wishes to perform, or to draw up completely detailed plans as he starts the planning process. Consider, for example, an agent who enters a chemistry lab for the first time and is asked to neutralize an acid. It is likely that he will not be able to perform the task. Moreover, he cannot even completely specify a simple plan such as:
1) ask a friend to tell him the procedure P
2) friend tells him P
3) perform P, that is, neutralize the acid.
Both 2) and 3) will be unspecified at the start of the plan, since the agent doesn't know P. In particular 2) will remain unspecified until it is completed.

Agents who function well in complex environments must be able to construct and execute plans despite their lack of knowledge. In particular, they should be able to successfully plan to get the information that they need to perform specific actions. An intelligent agent, for example, should realize that the above plan for neutralizing the acid is a reasonable one, while a plan such as
1) swim the English Channel
2) neutralize the acid
makes no sense at all.

A theory that allows for such flexible planning will have to provide solutions to at least two problems. Firstly, it must explain under what circumstances an agent knows how to perform an action. We call this the Knowledge Preconditions Problem for Actions. Secondly, even if an agent does not know how to perform an action, it must explain how he can execute a plan that gets the action done. We call this the Knowledge Preconditions Problem for Plans.

In this paper, we develop a flexible and expressive theory of action and planning, and present solutions to these two problems.

2. Previous Work

Most AI planners have ignored both of the Knowledge Preconditions Problems. This is true both of classic plannine programs such as OPS [Ernst and Newell 1969] and STRIPS [Fikes and Nilsson 1971] as well as more recent planners such as NOAH [Sacerdoti 1975], Non-Lin [Tate 1977], and TWEAK [Chapman 1985]. The theories underlying these planners have all implicitly assumed that agents always have complete knowledge; planning thus reduces to some sort of search through a pre-packaged list of action operators, pre-conditions, and post-conditions.

McCarthy and Hayes [1969] were the first to argue that an agent does in fact reason about his ability to perform an action, thus addressing themselves to the first of the Knowledge Preconditions Problems. They suggested writing down explicit knowledge precondition axioms for each action, so that a planning program could reason that it knew how to do an action if the relevant knowledge precondition axioms were true. This proposal however, fails on at least two counts: (1) it leads to an explosion of axioms, a large search space, and thus unacceptably slow proofs, and (2) it provides no explanation as to why or how agents come to know how to perform actions.

In [Moore 1980], Moore presented an elegant solution to the first Knowledge Preconditions Problem lor a limited class of actions. Moore used the modal logic of knowledge S4 with possible worlds semantics and the simple situation calculus model of actions, in which actions are regarded as functions on situations. Central to Moore's argument were two concepts from possible worlds theory: that of the rigid designator, an entity such as a name or number denoting the same object across possible worlds, and that of the rigid function, a function on rigd designators. Moore argued mat all actions were (axiomatically composed out of) rigid functions. Furthermore, all agents knew all axioms and knew how to perform actions as long as they knew rigid designators for the actions. An agent thus knew how to perform an action if he knew rigid designators for the parameters of the action.

Unfortunately, Moore's system is insufficiently general for the following reasons: (1) The S4 model entails that all agents know all axioms and that they thus have the same procedural knowledge; this reduces the entire Knowledge Preconditions Problem to a toy problem, (2) the first order modal logic that Moore uses is severely inexpressive, and does not allow us to express an agent's partial knowledge, (3) the McCarthy-Hayes situation calculus that Moore uses is too rigid to serve as the basis for a flexible theory of planning.

Since the first two objections follow from Moore's use of a modal logic of knowledge, and the third objection follows from Moore's use of the McCarthy-Hayes model of action, it makes sense to develop a theory which avoids both. We will build our theory upon a first order logic of knowledge, in which knowledge is represented as a relation on strings, and integrate it with an expressive, set-theoretic model of action. The resulting theory will lend itself in a natural manner to solutions to both of the Knowledge Preconditions Problems.

This paper extends the work of [Morgenstem 1986a] in which we first addressed the problem of constructing a first order logic of knowledge and action. There, we focussed upon developing a first-order logic of knowledge which avoids paradox and obeys the classical inference rules, and which could be integrated with various models of action. In this paper, we construct a detailed theory of action and planning and present our solution to the Knowledge Preconditions Problems within that context

3. The Logical Language

We will be using a language *L,* which is an instance of the first order predicate calculus. *V* s symbols consist of the logical symbols, such as v , - i , and 3 , constants, variables, predicates, and functions. We will feel free to substitute the English equivalents for logical symbols when it improves legibility. Constants are numbers or begin with an upper case letter. Predicates begin with upper case; functions and variables begin with lower case. Predicates and constants, functions and variables, will be disambiguated by context In our axioms and definitions, variables will be assumed to be universally quantified unless otherwise specified.

Our logic is sorted; sorts are distinguished by their first letters). Some of the more commonly used sorts are s, ranging over situations, i, ranging over intervals, a, ranging over agents, act, ranging over actions, str, ranging over strings. Other sorts will be introduced as needed, and will be understood from context

The quotation construct is an important feature of *L.* We assume a meta-function of L, G, which maps distinct expressions of *L* into distinct numbers. G is invemble; the analogue of G^{-1} in *L* is *g^{-1}*. In the proper context, a number of *L* is said to 'stand for' the expression of *L* which maps into it When used in this way, numbers are known as strings. A string is written as the expression it represents, surrounded by quotation marks. For example, the string 'At(John,NYC,s3)' represents the expression At(John,NYC,S3). Strings thus provide us with a way of talking *about* expression of *L within L.*

Various predicates of *L* take strings as their arguments; the most important of these are the predicates True and Know. Know takes three arguments: an agent, a string representing a sentence, and a situation. For example, to say that John knows something that Bill doesn't know, we write:

3 p (Know(John,p,S5) & -i Know(Bill,p,S5))

Note that by using strings, we are effectively quantifying over sentences while remaining in a first order language; we can use this trick to 'quantify' over functions and predicates by quantifying over strings which represent them. Note also the ease with which we can express partial knowledge.

Achieving full expressivity in languages with quotation can be complicated since the quoted constructs are opaque. For example to state the principle of positive introspection, that if an agent knows something, he knows that he knows it, we cannot simply say:

V a,p,s Know(a,p,s) ■> Know(a,'Know(a,p,s)',s)

This would mean that all agents always know the string 'Know(a,p,s)'. If we try to substitute values for a,p, and s, we will not be able to substitute the values within the quoted context: a string is a constant To solve this problem, we introduce some syntactic abbreviations: a name-of-operator, @, where @ applied to an object yields the name of that object, ! !, where ! ! applied to a string variable yields the string the variable stands for, and * *, where * * applied to a string variable yields the string the variable stands for, stripped of surrounding quotes. (*) As an example of their use, we now write the principle of positive introspection correctly:

V a,p,s Know(a,p,s) «> Know(a,'Know(@a,!p!,@s)',s)

Our axioms on Know correspond closely to the standard S4 axiomatization of knowledge. We assume veridicality, positive introspection, and consequential closure. We do not, however, assume necessitation: agents are required to know all logical axioms and axioms on knowledge, but not all axioms about the world.

4. Theory of Action

4.1. Requirements

Underlying every theory of action is an explicit or implicit ontology of action. The ontology of action that we choose will be determined by the set of requirements that are placed upon the theory. These requirements, in turn, are determined by the problems that our theory seeks to solve. Below, we briefly list the more salient requirements which we place upon our theory:

[1] Fidelity of Temporal Representation: A theory of action must be able to talk about both instants of time and intervals of time. Actions take place over intervals of time; facts may be true over instants or intervals. Instants are necessary so that we can describe what it true at a particular moment Time intervals are necessary so that we can describe how the world changes over time.

[2J Granularity: The theory should be able to view actions with varying degrees of granularity. That is, the term 'action' should encompass broadly general as well as detailed descriptions of actions. For example, both driving a car and driving a red Alfa Romeo with the roof down on a hot summer's day should be considered actions. This is especially important for the purposes of planning. At the start of the planning process, agents often think in general terms, about coarse-grained actions. However, these coarse-grained actions may become transformed into increasingly finer grained actions as the planning process continues and plan refinement occurs.

[3] Interval Dependent Actions: There are many actions whose very descriptions depend on the time during which

(*) These operators are defined in terms of standard features of *L* such as concat For further details, see [Morgenstem 1986b].

they take place, such as going to the top of the tallest building in New York. The values of descriptions such as 'the tallest building in New York' change with time. We wish to have a theory which allows us to describe actions in this manner.

[4] Composability: Most actions are formed by composing simpler actions in various ways. For example, the action of making blackberry jam can be thought of as a sequence of the action of crushing blackberries, mixing with sugar and pectin, bringing to a hard boil, and pouring into Mason jars. Our theory should provide a mechanism for composing simple actions to form complex actions, using the standard operators of a concurrent programming language, such as sequences, conditionals, while loops, and concurrency.

[5] Multiple Agents: Many of the actions we are interested in, such as communicative actions, are *interagent actions,* which involve at least two agents. Our theory must therefore be flexible enough to describe how multiple agents act and interact. In particular, we should be able to talk about agents acting simultaneously. We should also be able to talk about individual plans which are constructed out of actions done by many agents.

4.2. Choosing an Ontology

There are two major questions which must be addressed when choosing an ontology:
1) What arc actions? and
2) How can we describe actions?

Mainstream AI research has provided us with two answers for each of these two questions.

1) Actions have been *regarded* as
   a) functions on states, or
   b) sets of intervals.
2) Actions have been *described* using
   c) functional descriptions such as put-on(a,b), or
   d) set theoretic descriptions of the form (i I <(i) } where <K0 is a well formed formula free only in i.

It is important to realize that these approaches can be combined in a variety of ways. Three of the four possible combinations are coherent and have in fact been incorporated into AI theories. McCarthy and Hayes [1969] have regarded actions as functions on states and have used functions to describe these actions. Both McDermott [1982] and Allen [1984] have argued that actions are sets of intervals. However, Allen uses functional descriptions, while McDermott makes use of the set theoretic description of actions.

These approaches can be contrasted in different ways. There is a clear advantage to approach b) over approach a) : that of *oncological realism.* There are many actions, such as running around the block or waiting on the corner, which do not seem to involve a state change, and which are not the null action.

Each of approaches c) and d), however, has some advantage over its rival. It is clear that d) is more expressive than c). There are many actions, such as our example of driving a red Alfa Romeo with the roof down which cannot be described in any natural manner using functional descriptions. However, it is easy to describe such an action using a set theoretic description:

{i 13 a,c Alfa-Romeo(c) and Red(c) and Down(roof(c)i)
    and Drives(a,c,i))

If an action cannot be described using function descriptions, we say that it is *composite.*

In point of fact, however, no researcher using approach d) has utilized its expressive potential in any systematic manner. Moreover, approach c) has an important advantage over approach d): it is less cumbersome and easier to use for those actions which are functionally describable, such as driving a car (drive(c)) or putting one block on another (put-on(x,y)).

We aim to construct a theory which maximizes ontological realism, expressivity, and ease of use. With these ends in mind, we will regard actions as sets of intervals. To achieve expressivity, we will describe actions using set theoretic descriptions. We will show in a systematic manner how we can exploit the notation's expressive potential. To maximize ease of use, we will identify those classes of actions which are functionally describable, and use functional descriptions for these actions whenever possible.

4.3. Presentation of Theory

We now present a theory of action which satisfies the requirements discussed in section 4.1. The basic building block of our theory is the situation or state. States are ordered by the < relation, indicating precedence in time. (*) An interval is defined as a set of contiguous instants, all of which are linearly ordered. It is determined by its beginning and end points; these points name the interval. Thus, the interval starting at SII and ending at S25 is denoted [SI 1,S25]. Intervals in general are closed.

We define actions and events as sets of intervals, intuitively those in which the action or event takes place. Actions are those events in which the performing agent is not specified.

We place in *L* a set of action functions, such as put-on(bll,bl2), dial(x) and drive(c). These functions map their arguments onto sets of intervals, or actions. We can associate in a natural manner an n+2-place predicate with each n-place action function. The extra arguments are used for the agent performing the action, and the interval during which the action is performed. For example, we associate with the action functions above the predicates: Puts-on(a,bll,bl2,i), Dials(a,x,i) and Drives(a,c,i). We call these predicates *action predicates.*

Actions are introduced using these action predicates. Since an action is a class, it must be of the form {i I <(>(i)}, where $ is a wff free only in i. If Act « {i I *ty* (i)}, we call ⌖ the descriptive wff of Act. ⌖(i) must contain an action predicate.

We can achieve our second requirement, varying granularity of action, by expanding or restricting ⌖ in a systematic manner. We give some examples of this below:

(1) {i I 3 a Puts-on(a,BII,B12,i)} - the act of putting block BII on block B12

(2) {i I 3 a,bll,bl2 Puts-on(a,bll,bl2,i)} - the act of putting one block on another

(3) (i I 3 a,bll,bl2 Red(bll) and Puts-on(a,bll,bl2,i)} - the act of putting a red block on some other block

(4) (i I 3 a,bll,bl2 Equal(bll,favorite-block(Mary)) and Equal(bl2,favorite-block(Jane)) and
    Puts-on(a,bll,bl2,i)} - the act of putting Mary's favorite block on Jane's favorite block.

It will be noted that only some of these actions are functionally describable. (1), (2), and (4) are: their func-

(*) If this order is total, we have linear time; if it is partial, we have branching time. Neither assumption is crucial for the work done in this paper.

tional descriptions arc, respectively: put-on(Bll,B12), put-on(bl 1 ,bl2) and put-on(favorite-block(Mary),favorite-block(Jane)). There is, however, no way to produce a functional description for (3), unless we are willing to create functions such as put-red-block-on on the fly.

In general, we can show that an action Act $= \{i \mid \phi(i)\}$ is functionally describable if $\phi(i)$ is one of the following forms:

[1] ϕ contains no quantified variables, other than the variable representing the performing agent(s).

[2] ϕ contains quantified variables; these are all existentially quantified. Furthermore, does not contain any predicates involving these variables other than the action predicate(s) or the predicate Equal.

Functionally describable actions can further be subdivided into deterministic and non-deterministic actions. An action is said to be deterministic if all the arguments of its functional description evaluate to constants; otherwise it is non-deterministic. (1) and (4) are deterministic; (2) is not. We furthermore say that all composite actions are non-deterministic.

We close this section by giving two more examples of actions:

(5) $\{i \mid \exists$ a,bl1,bl2 Smallest-block(biU) and Largest-block(bl24) and Puts-on(a,bll,bl2,i)) - the act of placing the smallest block on the largest block

Note that this is an example of an interval dependent action.

(6) {i} - the set of all intervals. We will call this the null action, Null.

We can easily define such programming language operators as sequence(actl,act2), cond(p,actl,act2), while(p,act) and concurrent(actl,act2) using standard set theoretic notation. In particular, we can recursively define while loops in terms of sequences, conditionals, and the null act: While(p,act) = cond(p,sequence(act,while(p,act)),Null). (see [Moore 1980].) We say that sequence, cond, while, and concurrent are action functions.

4.4. Events

Events are sets of intervals; intuitively, those intervals in which some agent performs an action. Every action is an event; thus, examples (1) - (6) above are all events. However, the following are also events:

(7) $\{i \mid \exists$ a,bl1,bl2 Child(a) and Puts-on(a,bl1,bl2,i)$\}$ - the event of a child placing one block on another

(8) $\{i \mid \exists$ bl1,bl2 Puts-on(Bill,bl1,bl2,i)$\}$ - the event of Bill placing one block on another

These are not actions because the performing agent is in some way restricted.

If the performing agent in deterministically specified, we can express an event as do(a,act), where a is the performing agent, act is the action, and do is the function mapping agents and actions onto events. For example, (8) can be expressed as do(Bill,put-on(bll,bl2)).

We introduce the predicates on events R and Occur. If Ev is an event R(Ev,SI,S2) is true if S2 is the result of EvI's occurrence in SI. Occur(Ev,SI,S2) is true if Ev occurs between SI and S2. The two predicates are equivalent in a model of linear time. Note that the predicate R, while superficially similar to the Result predicate of the situation calculus, can in fact be defined in terms of our interval based ontology:

Def. **R(ev,s1,s2) iff [s1,s2] $\in$ ev**

In a model of branching time, in which only some states are actualized, or *real states,* we define:

Def. **Occur(ev,s1,s2) iff R(ev,s1,s2) and Real-state(s1) and Real-state(s2).**

5. Solution to the Knowledge Preconditions Problem for Actions

Our solution is based upon five general observations about actions, the agents who perform them, and the knowledge these agents possess:

[1] Agents need to have explicit procedural knowledge for the actions they perform. Agents often don't have the knowledge they need, and thus are not able to perform the actions. If an agent does not know the explicit procedure for making a souffle, he will not be able to do it

[2] All agents in a community know how to perform some basic actions. This assumption is a necessary precondition for meaningful teaching to occur.

[3] An agent needs more than procedural knowledge in order to perform an action. He also needs to know definite descriptions for the parameters of the action he is performing. For example, even if an agent knows how to dial a phone number, he will not be able to perform the action Dial(tclno(Mary)) if he doesn't know what Mary's number is. This observation has been the major focus of Moore's research on the Knowledge Preconditions Problem for Actions.

[4] Action descriptions make a difference. The same action can be described in a number of different ways; an agent may be able to reason that he knows how to perform the action in only one of its guises. For example, an agent might know how to perform sequence(beat-cggs,fry-eggs), but not how to perform make-omelette; he might know how to perform Dial(460-7100) but not Dial(telno(Courant)).

[5] An agent knows how to compose his knowledge. If an agent knows how to perform two actions, for example, he will generally know how to perform a sequence of those actions.

Our solution to the Knowledge Preconditions Problem for Actions will synthesize and formalize these notions. We present our solution in two stages, first giving the solution for functionally describable actions, and then giving the solution for composite actions.

5.1. Solution for Functionally Describable Actions

5.1.1. Deterministic actions

5.1.1.1. Primitive Actions:

We begin by designating a class of action functions as *primitive.* This class will include such simple action types as move, put-on and dial. In general, an action is primitive if it cannot be further decomposed into simpler actions. The intuitive idea is that all agents know the basic procedures for these simple, non-decomposable acts.

As mentioned, an agent needs to know more than the basic procedure in order to do an action; he must also know definite descriptions for the parameters of the action functions. We say that an agent knows a definite description for a parameter if he knows some *standard identifier* e.g., a constant, for the parameter. So, if f(argl...argn) is an action, f is primitive, and an agent knows standard identifiers for each of argl...argn, he *knows how to perform* those actions. This principle is expressed in Axiom 1. Note that Stidstr is a predicate that ranges over strings; it is true of a string iff the string is the quoted form of a standard identifier of L.

Arinm 1:

Knows-how-to-perform(a,'@f(@arg1,...,@argn)',s)
if
primitive-action(f) and
∃ p1 (Stidstr(p1) and Know(a,'equal(^p1^,@arg1,@s)',s)
and ... and
∃ pn (Stidstr(pn) and Know(a,'Equal(^pn^,@argn,@s)',s)

For example, if put-on is a primitive action, then A knows how to perform put-on(favorite-block(Mary),favorite-block(Jane)) if he knows standard identifiers for those blocks.

Note that the argument to the Knows-how-to-perform predicate is the *string* representing the action description in question. This is important so that the predicate can be opaque with respect to action descriptions: otherwise the predicate would be true or false of all action descriptions which evaluated to the same action. If an agent knows how to perform an action description Actl, and Actl and Act2 designate the same action, he will be able to perform Act2 only if he knows they designate the same action.

Axiom 2:

Knows-how-to-peform(a,'@f(@arg1,...,@argn)',s) if
∃ f†,arg1†,...,argm†
(Know(a,'@f(@arg1,...,@argn)
=@f†(@arg1†,...,@argm†)',s)
and
Knows-how-to-perform(a,'@f†(@arg1†,...,@argm†)',s))

5.1.1.2. Complex Actions

Actions that are composed out of simple actions using our four composition operators are designated as complex actions. The knowledge preconditions for complex acts depend in a straightforward manner on the knowledge preconditions for simpler actions.

Sequence: An agent knows how to perform a sequence of two actions if he knows how to perform the first, and knows that as a result of performing the first, he will know how to perform the second.

Axiom 3:

Knows-how-to-perform(a,'sequence(@act1,@act2)',s1) if
Knows-how-to-perform(a,'@act1',s1) and
R(Do(a,act1),s1,s2) =>
Knows-how-to-perform(a,'@act2',s)

Conditionals: An agent knows how to perform cond(p,actl,act2) if he knows p and knows how to perform actl or he knows that p is false and he knows how to perform act2.

Axiom 4:

Knows-how-to-perform(a,'cond(^p^,@act1,@act2)',s) if
Know(a,p,s) and Knows-how-to-perform(a,'@act1',s) or
Know(a,' ¬ ^p^',s) and Knows-how-to-perform(a,'@act2',s)

While Loops: Since while loops are defined in terms of conditionals and sequences (4.3), the knowledge precondition axiom for loops is straightforward:

Axiom 5:

Knows-how-to-perform(a,'while(^p^,@act)',s) if
Knows-how-to-perform
(a,'cond(^p^,sequence(@act,while(^p^,@act),Null))',s)

Concurrency: An agent who knows how to perform concurrent(actl,act2) must know how to perform each of actl and act2. In addition, the intersection of the two actions must be feasible, and the agent must have sufficient resources for both actions.

Axiom 6:

Knows-how-to-perform('concurrent(@act1,@act2)',s) if
Knows-how-to-perform(a,'@act1',s) and
Knows-how-to-perform(a,'@act2',s) and
Feasible(do(a,concurrent(act1,act2))) and
Resource-compatible(a,act1,act2,s)

The predicates Feasible and Resource-compatible are discussed in [Morgenstern 1987].

5.1.2. Non-deterministic Functionally Describable Actions

A non-deterministic functionally describable action is always of the form f(argl...argn) where at least one of the argj does not evaluate to a constant. An example is put-on(bll,bl2). Intuitively, an agent knows how to perform an action of this sort if he knows how to perform at least one deterministic instantiation of the action. So, for example, if B172 and B18 are blocks, and an agent knows how to perform put-on(B172,B18), he also knows how to perform put-on(bll,bl2). The same is true if he knows how to perform put-on(B172,favorite-block(Jane)). This intuition is justified by the set theoretic structure of non-deterministic functionally describable actions, which, we will remember, is (i 13 a **[∃ arg1] ... [∃ argn]** act-pred(a,argl,...,argn,i)J. The 'existential generalization' is true as long as we can find some values to satisfy it.

To formalize this rule, we need only add

Axiom!:

Assume one of argl... argn does not evaluate to a definite description. Suppose further that 3x1 ... xn, such that each xi is a deterministic instantiation of arg 1... argn. and that
Knows-how-to-perform(a,'@f(@x1,...,@xn)',s)
Then, Knows-how-to-perform(a,'@f(@arg1,....,@argn)',s)

This axiom, together with Axiom 1, will achieve the desiredl result.

5.2. Composite Actions

A composite action can be characterized in terms of its descriptive function φ. The following is true of φ:

(i) φ begins with a subexpression of the form '∃ a'

(ii) φ contains other quantified variables besides a

(iii) φ contains non-action predicates involving at least some of these variables.

Although in theory, φ's quantifiers may be either universal or existential, in practice most composite actions of interest contain only existential variables. We can identify the following form as being of particular interest to us:

(•) {I I ∃ a [∃ arg1] ... [∃ argn] P1(arg1,...,argn) conj. P2(arg1,...,argn) conj. ... conj. Pm(arg1,...,argn) and Action-pred(a,arg1,...,argn,i) }
where each Pi is a predicate of 1 through n arguments, and each conj. stands either for ∨ or &.

The expression
P1(arg1,...,argn) conj. ... conj. Pm(arg1,...,argn)
is called the *restriction* of φ.

If an action is of this form, we say it is an REQ (restricted existentially quantified) action. Thus, for instance, example (3) of Section 4.3 and the action of driving a Red Alfa Romeo with the roof down (Section *42)* are both examples of REQ actions.

Intuitively, an agent a knows how to perform an REQ action if there exist some constants CI ... Cn satisfying the restriction, for which he knows how to perform the associated action **{i | ϕ / a,C1...Cn,i }**. Suppose, for example, that Act is example (3) of section 4.3:

{i | ∃ a,bl1,bl2 Red(bl1) and Puts-on(a,bl1,bl2,i)}

If there is some constant value for MI, say B155, such that
(i)Red(B155)and
(ii) Knows-how-to-'**{i | ∃ a,bl2** uts-on(a3l55,bl24)}^),
we say that a knows-how-to-perform Act as well. We formalize this notion in the following axiom:

Axiom 8:

Let act be of the form described in (•), an REQ action.
Suppose that there exist n strings, strl... strn,
such that Stidstr(strl) and... and Stidstr(strn).
Suppose further that Pit(g'str) for each i in a through n;
that is, that for each i, the predicate Pi of (•),
is true of the constant denoted by the standard identifier stri.
Then,
if Knows-how-to-oerform
(a,'{i | Act-pred(a,^str1^,...,^strn^,i)',s),
then Knows-how-to-perform(a,'@act',s)

We defer for the present those composite actions which contain universal quantifiers. These cases will be considered in future research.

6. Solution to the Knowledge Preconditions Problem for Plana

6.1. The Concept of a Plan

Standard AI planning research [Fikes and Nilsson 1971, Sacerdoti 1975] has regarded plans as sequences of actions that are performed by a single agent (*) This planning paradigm, while adequate for simple single-agent toy domains like the blocks world, is nonetheless inadequate for more robust planning systems for at least two reasons:

(1) Planning may involve interactions between two or more agents. Consider even a simple plan such as my petting to the airport: it consists of my hailing a taxi, the driver driving to the airport, and my paying the taxi. The taxi driver s action is an essential part of my plan.

(2) Constructs other than sequence, such as concurrency, are widely used in real life plans. An army general s plan to attack a city might consist of his attacking the city's eastern front, while his colonel attacks the city's western front

These observations lead us to define a plan as any structure of events constructed with our standard event operators of sequence, conditionals, while loops, and concurrency. For example, we can express the general's plan, above, as:

concurrent(do(General,attack(eastern-front(City))),
do(ColonelX,attack(western-front(City))))

Note that this notation permits a particularly flexible kind of plan construction in which the planner need not even fully specify all the agents who will be performing the actions. For example, if the general is simply planning for *one* of his colonels to attack the city's western front, he might construct his plan as:

concurrent(do(General,attack(eastern-front(City))),
{i| ∃ a Colonel(a) and Under-command(a,General)
and Attacks(a,western-front(City),i)})

We introduce the function actors, which applied to a plan, yields the set of actors involved in the plan, and the function actions, which applied to a plan, yields the set of actions involved in the plan.

6.2. Plan Execution

To solve the second of the Knowledge Precondition Problems, we must explain under what circumstances an agent can successfully execute a plan. Intuitively, an agent can execute a plan if he can in some sense 'make sure' that all the events in the plan get done. More precisely, we can say that an agent can execute a plan if he knows that he will be able to perform all the actions in the plan for which he is the actor, and he can predict that the other events in the plan will take place at their proper time. In the taxi plan above, for example, I can successfully execute the plan if I know that I will be able to perform the actions of hailing a taxi and paying the driver, and I can predict that the taxi driver will indeed drive me to the airport

This section is devoted to formalizing this seemingly *simple* concept We will begin by examining how *agents* can execute simple single-agent plans and subsequendy extend our analysis to more complex plans.

We first define the concept of a simple plan. We introduce the predicate Simple-plan, which is true of plans consisting of events done by a single agent Thus, for example, Simple-plan(do(Adrive(car))do(A,park(car))) is true; Simple-plan(do(Adrive(car)),do(B,park(car))) is false. A plan that is a simple plan can always be re-written as a single event If Pin is a simple plan, we write ev(Pln) to denote the single event associated with this plan. Note that for simple plans, the function actors evaluates to a singleton.

If P is a simple plan, and A is the performing agent in this plan, A can execute P if he knows he is able to perform the action construct associated with P. We say that an agent is able to perform an act if he knows how to perform it and if the physical preconditions are satisfied. (++)

***Def*. Can-Perform(a,'@act',s) iff**
Knows-how-to-perform(a,'@act',s) and
Physprecondsat(a,act,s)

(*) It is a common misconception that NOAH allows for concurrent and multi-agent plans. In fact, NOAH simply allows for the representation of unordered actions during the intermediate stages of planning. This is not the same as representing concurrency: the final output of NO AH is always an ordered sequence of actions. Moreover, NOAH has no explicit representation for the actions of other agents (although this could conceivably be built in). Since in any case, NOAH cannot handle concurrency, general multi-agent planning is clearly impossible.

(*+) In our full theory [Morgenstem 1987] we say that an agent is able to perform an action if he knows how to perform it, if the physical preconditions are satisfied, and if certain social protocols are satisfied. These protocols are an important part of our theory of communication. We omit this condition here for simplicity.

Axiom 9:

```
actors(pln) = {a} =>
  Can-execute-Plan(a,pln,s) iff
  Know(a,'Can-Perform(@a,@action(@pln),@s)',s)
```

If A is not the performing agent for a simple plan, we say that A can execute the plan if he can predict its occurrence.

Axiom 10:

```
Simple-plan(pln) and actors(pln) ≠ {a} =>
  Can-execute-plan(a,pln,s) <=>
  Know(a,'∃ s2 Occur(ev(@pln),@s,s2)',s)
```

6.2.1. Complex Plans

If a plan is not simple, we say that it is complex. A complex plan must always be constructed via our four compositional operators. The knowledge preconditions for complex plans turn out to be considerably more difficult to state than the knowledge preconditions for complex actions.

Sequences: Our first attempt at a knowledge precondition axiom for plan sequences might parallel the axiom for action sequences:

(*wrong axiom*:)

```
Can-execute-plan(a,seq(pln1,pln2),s) if
 Can-execute-plan(a,pln1,s) and
 R(pln1,s1,s2) => Can-execute-plan(a,pln2,s2)
```

This axiom, however, places overly strong demands on the knowledge of the executing agent. Suppose Jones, a dying man, constructs the plan

```
sequence(Do( Jones,
        sequence(write(Will),die)),
      Do(Smith,execute(Will)))
```

Jones can conceivably execute the plan if he knows he can write a will, knows he will die, and can trust his attorney Smith. However, if we accept the above axiom attempt, Jones will not be able to execute the plan because once he is dead he does not know that Smith will execute the will. Clearly, what is important here is that Jones know what is going on at the beginning of the plan. Once he leaves the picture, we do not care what he knows. (*)

It turns out to be impossible to formalize the Can-execute-plan axiom for plan sequences in terms of the Can-execute-plan axioms for simple plans. We must in fact introduce a level of indirection, via the predicate Control. An agent is said to control a plan if he can perform the action(s) associated with it, or if the plan will occur. (**) The axioms on Control are self-explanatory:

```
Controls(a,pln,s) if
 actors(pln) = {a} and Can-Perform(a,'action(@a,@pln)',s)
```

(*) The slightly unusual case of a dying man is only one of the more salient examples of a basic truth: agents often lose information after they construct a plan; they arc nonetheless capable of executing the plan. Consider, for example, a busy executive who plans a conference in detail, convinces herself that the plan will work, delegates the plan to a secretary, and then proceeds to forget the details of the plan. At the time she delegates the plan, it makes sense to say that she can execute the plan.

(**) This use of control is unintuitive in some cases; for example, I control the plan in which humans land on Mars, if it will occur.

```
Controls(a,pln,s) if
 ¬ a ∈ actors(pln) and ∃ s† occur(pln,s,s†)

Controls(a,sequence(plnl,pln2),sl) iff
 Controls(a,plnl,s) and
 R(pln1,s1) => Controls(a,pln2,s2)

Controls(a,cond(p,pln1,pln2),s) iff
 p => Controls(a,pln1,s) and
 ¬ p => Controls(a,pln2,s)

Controls(a,while(p,pln),s) iff
 Controls(a,
  cond(p,sequence(p,pln,while(p,pln)),Null),s)

Controls(a,concurrent(plnl,pln2),s) iff
 Controls(a,plnl,s) and Controls(a,pln2,s)
```

We now say that an agent can execute a sequence of two plans if he knows that he can control the first and he knows that as a result of the first plan he will control the second.

Axiom 11:

```
Can-execute-plan(a,sequence(pln1,pln2),s1) iff
 Know(a,'Controls(@a,@pln1,@s)',s1) and
 Know(a,'R(@pln1,@s1,s2)
  => Controls(@a,@pln2,s2)',s1)
```

In the Jones-Smith example, above, Jones can execute his plan because at the beginning he knows that Smith will execute his will.

Conditionals: As with sequences, it is easy to overstate the knowledge preconditions for conditional plans. We might say that an agent can execute cond(p,plnl,pln2) if he knows p and can-execute plnl or he knows that p is false and can execute pln2. It turns out, however, that an agent can often successfully execute a plan without knowledge of the crucial condition. Consider Smith's plan to play the stock market by listening to the advice of his stockbroker Brown. Smith knows that Brown is watching out for the earnings report of IBM. Smith can construct the following plan:

```
cond(Favorable(eamings-rept(IBM)),
   sequence(do(Brown,tell(Smith,'Buy')),
        do(Smith,buy-shares(IBM))),
   sequence(do(Brown,tell(Smith,'Sell)),
        do(Smith,sell-shares(IBM))))
```

Smith can execute this plan even though he doesn't at the start know anything about IBM's earnings. Intuitively, this is true because he is not involved in the first part of the plan. It is *Brown* who must know IBM's earnings. More precisely, Smith's actions at the beginning of the plan (here Null) are not affected by the conditions of the plan.

We thus say that an agent executing a conditional plan is required to know the condition only if his actions at the beginning of the plan would be affected by the condition. We introduce the function first-action(a,pln) which returns the actions done by a during the first part of pin.

Axiom 12:

```
Can-execute-plan(a,cond(p,pln1,pln2),s) iff
 (first-action(a,pln1) ≠ first-action(a,pln2))
   => p => Know(a,p,s) and
     ¬ p => Know(a,'¬ ^p^',s) and
  (p and Can-execute-plan(a,pln1,s))
       or
  (¬ p and Can-execute-plan(a,pln2,s))
```

While Loops: Since while loops are defined in terms of sequence and conditionals, our axiom for while loops is simply:

Axiom 13:
```
Can-execute-plan(a,while(p,pln),s) iff
  Can-execute-plan
    (a,cond(p,sequence(pln,while(p,pln)),Null),s)
```

Concurrency: We say that an agent can execute a plan consisting of two concurrent plans if he can execute each plan. In addition, he must know that they are physically feasible, and that there are sufficient resources available.

Axiom 14:

```
Can-execute-plan(a,concurrent(pln1,pln2),s) iff
 Can-execute-plan(a,pln1,s) and
 Can-execute-plan(a,pln2,s) and
 Know(a,'Physically-feasible(concurrent(@pln1,@pln2))',s)
      and
 Know(a,'∀ a1 ∈ actors(concurrent(@pln1,@pln2))
  (Resource-compatible
     (a1,action(@pln1),action(@pln2)),@s)',s)
```

63. Example

We now demonstrate how our theory works in practice. We consider again the case introduced in Section 1. An agent A, entering a chemistry lab for the first time, is asked to neutralize an acid; he has no idea how to perform the procedure. We assume that A knows that some agent B knows how to neutralize the acid, and that A and B are cooperative agents. For the purposes of this brief paper, we furthermore assume that the following is true of our planning domain:
1) all communicative acts are primitive
2) friendly agents wish to do what they're asked to do
3) if an agent wishes to do an act and he can, then he will
4) friendly agents are constrained to tell the truth.

Finally, we assume that the physical preconditions for the actions here are satisfied. (These assumptions are dropped in [Morgenstern 1987], where an isomorphic problem is worked out in detail.) We can then show that A can successfully execute the following plan. The actions introduced below should be self-explanatory.

The plan consists of a sequence of three steps:

```
sequence
 {Step 1} do(A,request-act(B,
     '{i∃ p Procedure-string(p) and
       Tells(B,A,
         'do(A,neutralize-acid) = do(A,^p^)',i}')

 {Step 2} {i∃ p Procedure-string(p) and
          Tells(B,A,'do(A,neutralize-acid)= do(A,^p^)',i)}

 {Step 3} {i∃ p Procedure-string(p) and
          do(A,^p^)=do(A,neutralize-acid)
          and Does(A,^p^,i)}
```
Equivalently, do(A,neutralize acid)

Since communicative acts are primitive, A knows that he automatically knows enough to ask B to perform the requested action. In addition, the physical preconditions for this action are satisfied. Thus, A knows mat he is able to perform the action of the first step in this plan. Moreover, since A and B are friendly, A knows that B will perform the favor that he has requested, telling him how to neutralize the acid, if B possibly can. In point of fact, since B knows how to neutralize the acid, he can tell A how to perform the action. Thus, A can predict the occurrence of the second step. Once B tells A the procedure, A will know what the procedure is. So A can predict that he will be able to perform the act of neutralizing the acid. A can thus reason that he can successfully execute the plan consisting of the sequence of Stepl, Step2, and Step3.

7. Conclusion

We have constructed a highly flexible model of action and planning, and have demonstrated that it is well suited for partially specified plans and for multi-aaent interactions. We have presented solutions to both of the Knowledge Preconditions Problems within that context, explaining how agents can reason about their ability to perform actions and execute plans.

This paper represents the second stage of a three-stage research effort to develop a robust logic of knowledge, action, and communication. In a future paper, we present a logic of communication based upon an Austinian model of speech acts [Austin 1962], and discuss how we can integrate this theory with our solutions to the Knowledge Preconditions Problems.

Acknowledgements: This research was supported in part by NSF grant DCR-8603758. Thanks to Ernie Davis for many helpful ideas, suggestions, and discussions.

BIBLIOGRAPHY

Allen, James: Toward a General Theory of Action and Time', *Artificial Intelligence,* Vol. 23, No.2,1984

Austin, J.L.: *How to Do Things With Words,* Harvard University Press, Cambridge, 1962

Chapman, David: *Planning for Conjunctive Goals,* MIT TR 83-85,1985

Ernst, G. and Newell, Allan: *GPS: A Case Study in Generality and Problem Solving,* Academic Press, New York, 1969

Fikes, R.E. and Nils Nilsson: 'STRIPS: a New Approach to the Application of Theorem Proving to Problem Solving,' *Artificial Intelligence,* Vol 2,1971

McCarthy, John and Patrick Hayes: 'Some Philosophical Problems from the Standpoint of Artificial Intelligence' in Bernard Meltzer, ed: *Machine Intelligence 4,*1969

McDennott, Drew: 'A Temporal Logic for Reasoning About Processes and Plans,' *Cognitive Science,* 1982

Moore, Robert: *Reasoning About Knowledge and Action,* SRI Technical Note 191,1980

Morgenstern, Leora: 'A First Order Theory of Planning, Knowledge, and Action', *Proceedings of the Conference on Theoretical Aspects of Reasoning About Knowledge,* Morgan Kaufmann, Los Altos, 1986

Morgenstern, Leora: *Foundations of a Logic of Knowledge, Action, and Communication,* forthcoming NYU PhD. thesis, 1987

Morgenstern, Leora: 'Preliminary Studies for a First Order Logic of Knowledge and Action; NYU Technical Report 262; 1986

Sacerdoti, Earl: *A Structure for Plans and Behavior,* American Elsevier, New York 1977

Tate, Austin: 'Generating Project Networks', *Proceedings, Fifth International Conference on Artificial Intelligence,* 1977