A PROCEDURAL LOGIC

Michael P. Georgeff *
Amy L. Lansky
Pierre Bessiere

Artificial Intelligence Center
SRI International
Menlo Park, California

Abstract

Much of our commonsense knowledge about the real world is concerned with the way things are done. This knowledge is often in the form of *procedures* or *sequences* of actions for achieving particular goals. In this paper, a formalism is presented for representing such knowledge based on the notion of *process.* A declarative semantics for the representation is given, which allows a user to state *facts* about the effects of doing things in the problem domain of interest. An operational semantics is also provided, which shows *how* this knowledge can be used to achieve given goals or to form intentions regarding their achievement. The formalism also serves as an executable program specification language suitable for constructing complex systems.

1 Introduction

Active intelligent systems need to be able to represent and reason about actions and how those actions can be combined to achieve given goals. Much of this knowledge is in the form of sequences of actions or "procedures" for accomplishing these goals. For example, knowledge about kicking a football, performing a certain dance movement, cooking a roast dinner, solving Rubik's cube, or diagnosing an engine malfunction, is primarily procedural in nature.

Within AI, there have been two approaches to the problem of action and practical reasoning, with a somewhat poor connection between them. In the first category, there is work on theories of action - i.e., on what constitutes an action per se [1, 10, 15]. This research has focused mainly on problems in natural-language understanding concerned with the meaning of action sentences. Second, there is work on planning - i.e., the problem of constructing a plan by searching for a sequence of actions that will yield a given goal [2, 5, 18 - 23]. Surprisingly, almost no work has been done in AI concerning the execution of preformed plans or procedures - yet this is the almost universal way in which humans go about their day-to-day tasks, and probably the only way other creatures do so. To actually search the space of possible future courses of action, which is the basis of planning, is relatively rare.

In attacking this problem, we first have to identify what it is that humans or other active systems do when performing a complex action. We postulate that such systems have some representation of a procedure for achieving given goals, or reacting to particular events, and that they can reason about and execute this procedure to achieve their aims. Just as we might view intelligent systems as having "beliefs** about the world, we consider these systems to have "procedures" for acting in the world. And, just as for theories of belief, the problem here is to provide abstract models for these "mental entities." We call these abstract models *processes.*

There are two aims to our work. One is to develop a theory suitable for building active intelligent agents. In that regard, the theory presented in this paper models only the simplest kind of agent - one with no preserved beliefs and with limited reasoning abilities. We define a declarative semantics for our formalism, as well as an operational semantics. Together these provide a suitable semantics for simple action sentences in natural language and a method of practical reasoning about how to accomplish given goals.

The other aim is to provide a basis for the design of improved programming languages - in particular, languages that allow users to represent their knowledge about the behavior of systems declaratively, are amenable to verification, and operationally are flexible and responsive to environmental changes. In this sense, our work can be viewed as the basis for executable specification languages.

It is important to point out that the theory presented here is not just another variant of the standard logics for describing dynamic behaviors. In particular, there is no existing logic (temporal, dynamic or interval-based) known to us that can both (1) express the same complexity of

'Also affiliated with the Center for the Study of Language and Information at Stanford University.

This research has been made possible In part by a gift from the System Development Foundation, by the Office of Naval Research under Contracts N00014-80-C-0296 and N00014-86-C-0251, and by the National Aeronautics and Space Administration under Contract NAS2-11864. The views and conclusions contained In this paper are those of the authors and should not be interpreted as representative of the official policies, either expressed or implied, of the Office of Naval Research, NASA, or the United States government.

action as the formalism proposed here (which can handle sequencing, conditional selection, nondeterministic choice, iteration, and hierarchical abstraction), and (2) be used to *automatically* generate behaviors for achieving goals and to form plans. In this sense, the approach here offers the same kind of advantages as Prolog, but in a dynamic rather than static domain: it can be viewed as a logic describing properties of behaviors, or it can be used as a programming language for generating behaviors to achieve given goals.

Furthermore, the model we use is based on nondeterministic procedures. This nondeterminism is essential for providing the kind of flexibility exhibited by intelligent systems. The model also allows for action failures and tests with side effects, both of which are necessary for handling most real-world domains. Such a model would be very cumbersome to describe in any of the standard temporal or dynamic logics - indeed, we know of none that have attempted to do so.

A system based on the proposed representation has been implemented and is currently being used for an intelligent robot and for fault isolation and diagnosis on the space shuttle. An early version of an implemented system is described in Georgeff and Bonollo [6] and the latest work in Georgeff and Lansky [8], The more recent work includes many capabilities not decribed in this paper, including a database of preserved "beliefs" and more powerful reasoning abilities represented as metalevel processes.

2 Processes and Actions

Most previous work in representing actions has been based on state change models [5, 13, 18]. However, existing models can describe only a limited class of actions and are too weak to be used in dealing with multiagent or dynamic worlds.

Some attempts have recently been made to provide a better underlying theory for actions. McDermott [15] considers an action or event to be a set of sequences of states, and describes a temporal logic for reasoning about such actions and events. Allen [Ij also considers an action to be a set of sequences of states, and specifies an action by describing the relationships among the intervals over which the action's conditions and effects are assumed to hold. However, while it is possible to state arbitrary properties of actions and events, it is not obvious how one could use these logics to achieve, or form intentions to achieve, one's goals.[1]

Our notion of action is essentially the same as that of McDermott and Allen; namely, we consider actions to be sets of sequences of world states. However, in modeling intelligent agents, it is convenient to consider not only states of the external world, but also various "mental entities," such as beliefs, goals and intentions. In the same way, it is important to be able to model not only the actions that occur in the real world, but the internal mental "procedures" that agents use to generate their external behaviors. We will call these entities *processes* (see [7] and, for some early work based on similar ideas, [10]).

We assume that, at any given instant, the world is in a particular *world state.* A *process* is some abstract mechanism that can be executed to generate a sequence of world states, called a *behavior* of the process. The set of all behaviors of a process constitutes the *action* (or *action type)* generated by the process. In this paper we restrict our attention to sequential (nonconcurrent) processes.

Each process is modeled by a labeled transition network, with distinguished start and finish nodes. The nodes of the network are called *control points,* and are labeled with *state conditions.* These conditions can be viewed as representing constraints on possible world states. Each arc of the network is labeled by a *goal,* which can be considered to represent a particular type of *behavior* to be achieved.[2] Associated with each network is a *purpose,* which is the goal that will be achieved if the process is successfully executed.

A process is *executed* in the following manner. At any moment during execution, the process is at a given control point *c.* An outgoing arc *a* may be traversed if (1) the current state of the world satisfies the state condition labeling *c* and (2) the goal labeling *a* is *successfully* achieved. If no outgoing arc from *c* can be traversed, process execution *fails.* Execution begins with control at the initial control point and *succeeds* if control reaches the final control point.

In some ways, a process may be viewed as just a convenient way of specifying actions. However, processes also allow us to make a distinction that is critical for practical reasoning - we can distinguish between behaviors that are *successful* executions of the process and those that are unsuccessful (or have *Jailed).* Since actions often fail to achieve their intended goals, it is important to be able to reason explicitly about the consequences of action failure. We thus need to be able to represent the behaviors that correspond to failed actions as well as successful ones. This is particulary important if the model is to be extended to handle multiagent and dynamic environments (e.g., see [11]). Similarly, in natural-language understanding, it is important to have a denotation for action sentences (such as "he was painting a picture**) that allows for action failure, even in mid-performance ("he was painting a picture when killed by lightning").

The notion of action failure also allows us to represent

[1] Allen |2] proposes a method of forming plans that is based on his representation of actions. However, he does not use the temporal logic directly, and actions are restricted to a particularly simple form (e.g., they do not include conditionals).

[2] In Section 6 we show how a goal to achieve a given *$state* can be represented as a type of behavior.

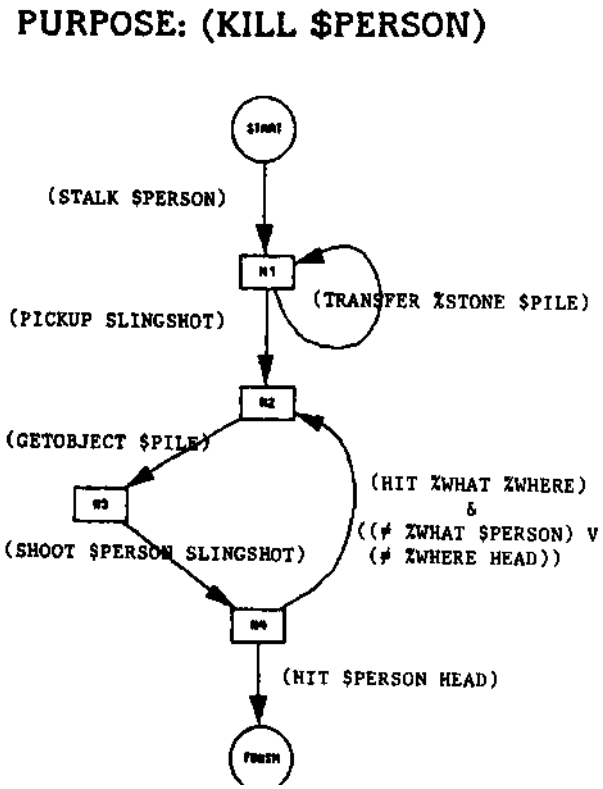

Figure 1: David and Goliath

tests on world states as actions, without the introduction of knowledge or belief structures (cf. [16]). To test whether a particular condition is true, one need simply perform an action that can only succeed when the condition is indeed true. (Of course, action failure cannot, in general, be equated with the falsity of the condition being tested.)

3 Process Descriptions

In this section we develop a formalism for describing processes and for reasoning about the behaviors they generate. Each process description consists of a *purpose description* and a *body*. The body is a network isomorphic to the network of the described process. The state conditions labeling the control points of the underlying process are modeled by expressions which have as their denotation world states; the goals labeling the arcs of the underlying process are modeled by expressions whose denotations are behaviors (sequences of world states). The purpose description also denotes a set of behaviors.

A typical process description using the formalism is shown in Figure 1. It describes a procedure for killing someone with a slingshot.

The process involves gathering stones, placing them in a pile, getting a slingshot, and then repeatedly taking up a stone and shooting it until the foe ((person) is hit on the head. In this particular domain, hitting someone on the head with a stone hurled by a slingshot always results in that person's death. The procedure is nondeterministic and allows agents to gather as many stones as they wish, limited only by their ability to continue gathering them. The procedure is not guaranteed to be successful - it may fail if any one of the actions labeling the arcs of the network fails. However, if there are only a finite number of gatherable stones, the procedure is guaranteed to terminate.

It is important to note how the process description captures *implicit* knowledge of the problem domain. This knowledge is of two kinds: one concerning the validity of the killing procedure, the other heuristic. For example, hitting a person on the head with an object propelled from a slingshot will not always kill them (e.g., if it's a cotton ball), but will if it's a stone (in this particular domain). Thus, the validity of the conclusion depends critically on the first part of the procedure, which ensures that only stones are placed in the pile. (Strictly, the procedure should also ensure that the pile is initially empty or contains nothing but stones.)

The procedure also captures heuristic knowledge in that earlier actions may make subsequent actions more likely to succeed. For example, the slingshot may require a certain size and weight of stone; however, instead of this being represented as an *explicit* precondition of the shooting action, it is represented implicitly by the context established by the procedure. In this case, the assumption is that any stone that can possibly be gathered will most likely possess the appropriate characteristics. Note that this does not affect the validity of the procedure; if a stone does not have the necessary properties, the action of shooting the slingshot will fail.

We now give a definition of the formalism. A *process description* is a tuple

$$P = \{S, F, N, E, \delta, n_I, n_F, C, A, G\} ,$$

where

- S is a [possibly infinite] set of *state descriptions*
- F is a [possibly infinite] set of *action descriptions*
- N is a set of *nodes*
- E is a set of *arcs*
- $\delta : N \times E \rightarrow N$ is the *process control function*
- $n_I \in N$ is the *initial node*
- $N_F \subset N$ is a set of *final nodes*
- $C : N \rightarrow S$ associates a state description with each node

- *A* : *E* —► *F* associates an action description with each arc
- *G* is an action description called the *purpose* of the process.

The state descriptions labeling the nodes are called *[partial] correctness assertions;* the one labeling the initial node is called the *precondition* of the process. The action descriptions labeling the arcs are called *goal assertions.*

We choose predicate calculus as the state description language. A state description can be viewed as denoting a set of states; namely, those in which it is true. We distinguish between *local* and *global* variables. Informally, the interpretation of a local variable is fixed in the interval during which a given arc is transitted, but can otherwise vary. A global variable, on the other hand, has a fixed interpretation during the execution of the entire process. (Local variables are needed especially in loops where it is necessary to identify different elements from one iteration to the next). A state description is any formula in this calculus in which all global variables are free and all local variables are bound. In the example of Figure 1, global variables are prefixed by $ and local variables, assumed to be existentially quantified, by *%.* All correctness assertions are assumed to be *true.*

An action description consists of an action predicate applied to an n-tuple of terms. Action descriptions denote *action types* or *sets* of state sequences. That is, an expression like "walk (a, b)" is considered to denote the set of walking actions from point a to b. Any sequence of states satisfies the action description if it is in the set so denoted. In Section 6 we augment the action description language to include various temporal operators.

4 Declarative Semantics

The declarative semantics of process descriptions is intended to describe what is *true* about the underlying system of processes and the world in which they operate. Such a semantics says nothing about *how* such knowledge could be used to achieve particular goals — rather, it simply allows one to state *facts* about certain behaviors.

On an intuitive level, the declarative semantics is straightforward. The intended meaning of a process description *P* is that every behavior that satisfies the goal and correctness assertions for some path through the net also satisfies the purpose of *P.* Alternatively, one may view the body of *P* as denoting a set of behaviors - namely, those that satisfy the goal and correctness assertions for some path through the net. Then the intended meaning of *P* is that each behavior in the set satisfies the purpose of P.

Unfortunately, allowing only simple paths through the net will not do. For example, if a node has multiple outgoing arcs, we need to allow several of these arcs to be tried until one is found successful. This is exactly the sort of behavior required *of* any useful conditional plan or program; if a test on one branch of a conditional fails (returns false), it is necessary to try other branches of the conditional. The problem in this case is that an attempted test may change the state of the world. Thus, paths through the network must allow behaviors that explicitly include failed attempts at realizing tests and actions as well as successful ones.

A formal definition of the semantics of process descriptions is given in [9]; here we will simply give an informal outline. The approach is similar to that used for most temporal logics. We first consider single states. A *state s* consists of a set of elements from a domain *D* together with relations and functions defined over these elements. Assuming a fixed interpretation for each constant symbol in the language, a *state interpretation I* assigns to each variable in the language an element of D, to each n-ary predicate symbol an n-ary relation in D_j and to each n-adic function symbol an n-adic function in *D.* The truth-value of a state assertion *w* in a state *s* with respect to a state interpretation / is defined in the standard way (variables ranging over elements of *D).* We can also view *w* as denoting the set of states in which *w* is true.

While state interpretations may vary from state to state in the course of a behavior, the interpretation of global variables must remain the same. For a process description *P,* a *global variable assignment a* is defined to be an assignment of an element in *D* to each global variable in *P.* Similarly, for each arc in P, we have a local variable assignment that associates a value with each local variable used by the goal assertion of that arc. In the course of a behavior satisfying the goal assertion, its local variables may take on at most one value. A state interpretation / is said to be *consistent* with a given a (or *β)* if the assignment to global (local) variables in / is the same as their assignment in). Note that we do not require a fixed interpretation for predicate symbols or function symbols over the sequence of states in a behavior. We define a *process instance* to be a process description together with consistent global and local variable assignments.

Following the discussion above, we consider the set of behaviors denoted by the body of a process instance as falling into either of two classes, one of which we will call the *success set* of the process instance and the other the *failure set* The success set represents all those behaviors that constitute successful executions of the underlying process; the failure set represents all those executions that fail somewhere along the way.

Let *P* be a set of process instances and let n be a node in a process instance *P*. An element *Q* of *P* is said to be applicable to an arc *a* emanating from n if its purpose is included in the set of behaviours described by the goal assertion of *a*.

The allowed behaviors starting at node n are those in which each applicable process instance at n is tried *at most once* until one succeeds or they all fail. * Let *succ(n, a)* be the set of behaviors consisting of some arbitrary number of unsuccessful attempts by applicable process instances (at most one per process instance) on the arcs emanating from n, followed by a behavior of an applicable process instance that succeeds for some arc a. Each of these attempts, both successful and unsuccessful, must begin in a state that satisfies the correctness assertion at node n. Similarly, let *fail(n)* be the set of all behaviors that fail to reach a successor node of n, i.e., behaviors consisting of failed attempts of all applicable processes. In this case, an attempt may fail because it cannot satisfy the correctness assertion at node n, or because the applicable process instance itself fails.

The success and failure sets for a node n, denoted *S(n)* and *F(n)* respectively, are then defined recursively as follows: [4]

1. If n is a final node, then *S(n)* is the set of states satisfying the correctness assertion at *n* and *F(n)* is the set of states that fail to satisfy the correctness assertion at n.

2. If n has arcs a_i to nodes m_i, $1 \leq i \leq k$, then
$S(n) = \bigcup_i succ(n, a_i).S(m_i)$ and
$F(n) = \bigcup\{fail(n), \bigcup_i succ(n, a_i).F(m_i)\}$

The success and failure sets of a process description *P* are then taken to be the success and failure sets, respectively, of the initial node of *P*. The semantics of *P* is that any behavior in the success set of *P* satisfies the purpose of P.

As an example, consider the process networks shown in Figure 2 where the arcs are labeled with applicable process instances. For a process instance P_1 let *(P)* denote the set of its successful behaviors, and *(P)F* the set of its failed behaviors. Then the success and failure sett for each of the process networks in Figure 2 are as follows:

[3]The deciiion to try each process instance at most once allows us to realiEe the control constructs of standard programming languages; various alternatives are possible without substantially affecting the results presented here.

[4]If $w_1 = s_1, \ldots s_k$ and $w_2 = s_k, \ldots s_n$, then $w_1.w_2 = s_1, \ldots s_{k-1}, s_k, s_{k+1}, \ldots s_n$. This operation is extended to sets of sequences in the usual way. Note that this formulation allows a single state to satisfy a sequence of goals.

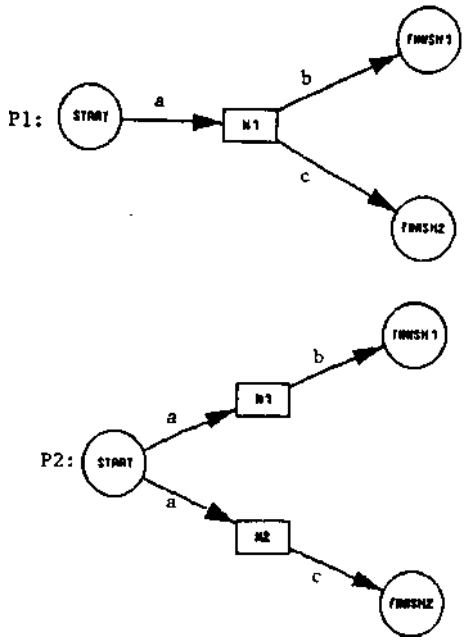

Figure 2: Sample Process Networks

$(P1)$: $\langle a\rangle\ \langle b\rangle$; $\langle a\rangle\ \langle c\rangle$; $\langle a\rangle\ \langle b\rangle_F\ \langle c\rangle$; $\langle a\rangle\ \langle c\rangle_F\ \langle b\rangle$

$(P1)_F$: $\langle a\rangle_F$; $\langle a\rangle\ \langle b\rangle_F\ \langle c\rangle_F$; $\langle a\rangle\ \langle c\rangle_F\ \langle b\rangle_F$

$(P2)$: $\langle a\rangle\ \langle b\rangle$; $\langle a\rangle\ \langle c\rangle$

$(P2)_F$: $\langle a\rangle_F$; $\langle a\rangle\ \langle b\rangle_F$; $\langle a\rangle\ \langle c\rangle_F$

Notice that backtracking upon failure occurs only up to the current node being exited, and no farther.

Because process descriptions can be recursive, and because loops in process networks introduce self-reference into the definitions of *S* and *F* given above, a formal specification of the semantics of process descriptions requires a fixed-point construction. That is, for a given set of process instances $P = P_1 \ldots P_n$, we need to define a transformation *T* that maps n-tuples of pairs of success and failure sets into additional such n-tuples. The definition of *T* is based on the definition of success and failure sets given above. If one assumes a set of primitive tests and actions, the least fixed point of *T* applied to these primitives can be taken as the denotation of $P_1 \ldots P_n$.

5 Operational Semantics

Process descriptions provide a way of describing the effects of actions in some dynamic problem domain. But how can a system or "agent" *use* this knowledge to achieve its goals? That is, we currently have a knowledge representa-

tion that allows us to state certain properties about actions and what behaviors constitute what actions. We have not explained, however, how an agent's *wanting* something can provide a rationale for or *cause* an agent to *act* in a certain way. This is the basis of so-called *practical reasoning* [3].

One way to view the causal connection between reasoning and action is as an *interpreter* that takes knowledge about actions and goals as input and as a result performs certain acts in the world. An abstract representation of such an interpreter may be considered to be the *operational semantics* of the knowledge representation language.

If a system is to be able to achieve its goals, it must be able to bring about certain actions, and thus be able to affect the course of behavior. Thus, we assume a system with certain effector capabilities. The actions that the system can effect simply by *choosing* to do so will be called *primitive actions.* The system must also be able to sense the world to the extent of determining the success or failure of the primitive actions. In addition, we assume the system has sensor capabilities for detecting satisfaction of all correctness assertions.

The system tries to achieve its goals by applying the following interpreter to applicable process instances. The interpreter works by exploring paths from a given node n in a process description P in a depth-first manner. To transit an arc, it unifies the corresponding arc assertion with the purposes of the set of all process descriptions, and executes those that unify, one at a time, until one terminates satisfactorily. If none of the matching processes terminate successfully, and all leaving arcs fail, the execution of P fails. At each node, we verify that the correctness assertion (c-assertion) is satisfied.

```
function successful (P n)
  if (is-end-node n) then
    if (satisfied (c-assertion n)) then
      return true
    else return false
  else
    arc-set := (outgoing-arcs n)
    pr-a-set := (processes-that-unify arc-set)
    do until (empty pr-a-set)
      if (not (satisfied (c-assertion n))) then
        return false
      pr-a := (randomly-delete pr-a-set)
      pr := (process pr-a)
      a := (arc pr-a)
      if (successful pr (start-node pr)) then
       return (successful P (terminating-node a))
    end-do
    return false
end-function
```

The function processes-that-unify takes a set of arcs and returns the set of processes that unify with some arc in the set, along with the specific arc with which each unifies. The functions process and arc select out the process instance and corresponding arc from each element of this set. The function randomly-delete selects an element from a set, destructively modifying the set as it does so. The order in which selections are made is called the *selection rule.* The function return returns from the enclosing function, not just the enclosing do. The initial system goal is represented by a process description with a single arc labeled with the goal.

Note that, if this theory were to form the basis of the reasoning capabilities of some real-world agent, we would probably want process descriptions to be invoked on the basis of particular facts becoming known as well as because particular goals have been established. A suitable organization for such a system would be to have a list of all applicable process descriptions - some goal-invoked and others fact-invoked - and at each stage of processing select one of these for execution [8]. The above recursive implementation would have to be modified, but the semantics would remain essentially the same.

Of course, it is important that the operational and declarative semantics be consistent with each other. The declarative semantics defines a set of behaviors for each process instance. The operational semantics also defines a set of behaviors for each process instance, but this set depends on the selection rule utilized in the above algorithm. Let $(P)_D$ be the set of successful behaviors for a process instance P as given by the declarative semantics, and let $(P)o,R$ be the set of successful behaviors for P as given by the operational semantics for selection rule R. It is not difficult to show that

$$(P)_{O,R} \subset (P)_D$$

This means that any behavior generated by the interpreter given above will satisfy the declarative semantics. However, the inclusion, in general, is strict. That is, the interpreter may not achieve some given goal even when, according to the declarative semantics, there exists a way to achieve it. But, assuming that all correctness assertions are directly testable, we do have the following:

> If a behavior a is in $(P)D$, there exists a selection rule R such that s is in $(P)O,R$-

This is the best one can really hope for when any particular selection may cause some possibly irreversible action. It means that, provided you are smart enough to choose the right selection rule, the above interpreter will achieve a goal if it is at all achievable. This highlights the importance of reasoning about the selection of applicable processes in any practical implementation (see also |8]). It also means that one can reliably *plan* to achieve goals and be guaranteed of finding a finite plan if one exists.

6 Action Descriptions

So far, action descriptions have been restricted to simple action predicates. However, it is desirable to also allow a class of action descriptions that relate to conditions on world states.

We thus extend the action description language to include actions that achieve a given world state *p* (represented as !p), actions to test for p (?p), and actions that preserve *p* (#p). We define these action descriptions more formally as follows.

We assume a fixed domain *D* and a fixed interpretation for constant symbols. Let *w* be a state assertion, *a* an action description of the above form, and $S = s_1 \ldots s_n$ a behavior. Assume fixed global and local variable assignments and let all local interpretations / be consistent with them. We then have the following truth rules:

1. !*w* is true in *S* if, for some local interpretation I, w is true in s_n
2. *?w* is true in *S* if, for some local interpretation /, *w* is true in s_1.
3. #w is true in *S* if, for all i, $1 < i < n$, there exist local interpretations I_i,- such that *w* is true of all states in *S* or *-iw* is true in all states in 5.

To make effective use of such action descriptions we can use proof rules of the kind given below. We will use the notation *(P) (a)* to mean that *every* successful behaviour associated with the process description *P* satisfies the temporal assertion *a*. *(P)F* denotes failed behaviors. The symbols ";" and T represent sequential composition and [nondeterministic] branching, respectively.

Some typical proof rules are as follows:

Conjunctive Testing

$$\frac{(P_1)(?p) \wedge (P_1)(\#q) \wedge (P_2)(?q)}{(P_1 \,;\, P_2)\,(?(p \wedge q))}$$

Conjunctive Achievement

$$\frac{(P_1)(!p) \wedge (P_2)(!q) \wedge (P_2)(\#p)}{(P_1; P_2)(!(p \wedge q))}$$

Disjunctive Testing

$$\frac{(P_1)(?p) \wedge (P_2)(?q) \wedge (P_1)_F(\#q) \wedge (P_2)_F(\#p)}{(P_1 \mid P_2)(?(p \vee q))}$$

Disjunctive Achievement

$$\frac{(P_1)(!p) \wedge (P_2)(!q)}{(P_1 \mid P_2)(!(p \vee q))}$$

Note that these proof rules are not the only ones, nor are they the strongest, that could be used. For example, in the rule for conjunctive achievement, we need not require that p be unaffected by *(P_2);* all we need do is regress the goal !p through *(P_2)* and set this as the goal of *(P_1)*. However, since in most real-world cases it is difficult to regress conditions through processes, the rules given above prove to be most practical.

The declarative semantics with this extension to the language is standard. The operational semantics simply requires that the interpreter be modified to allow application of the proof rules when necessary.

7 Conclusions

This paper has presented a simple model for action and a means for representing knowledge about procedures. We have indicated the importance of reasoning about *processes* rather than simply histories or state sequences. A declarative semantics for the representation was provided that allows a user to specify *facts* about behaviors independently of context. We have also given an operational semantics that shows *how* these facts can be used by an agent to achieve (or form intentions to achieve) its goals.

This knowledge representation can also be used for planning. Indeed, the operators of many standard planning systems (such as NOAH [19], DEVISER [22] and SIPE [23]) can be viewed as restricted forms of process descriptions. The fact that any behavior allowed by the declarative semantics can also be found using the operational semantics means that a planning algorithm that tried all possible selection rules would be "complete" - that is, it would find a solution if one existed.

By modifying the formalism so that failure sets allow full backtracking, single-state theorem proving of Horn clauses becomes a special case. This modification would also include as a special case the realization of "backtracking through triangle tables," as proposed by Nilsson [17]. However, such modifications present practical problems of verification and efficiency, and would appear useful only in some special cases.

In some ways, the declarative semantics is surprisingly complex and would seem to indicate some undesirable properties of the representation. Most of these difficulties arise from the need to model failed as well as successful behaviors. Of course, if we could fully specify necessary correctness conditions independently of context, *and* test for them, failed behaviors would become irrelevant for practical reasoning; we could always test to make sure conditions were true when needed. But experience with programming languages, and indeed the real world, shows that this can often be impractical if not impossible.

The formalism presented here can also be viewed as an

executable specification language — that is, as a programming language that allows a user to directly describe the behaviors desired of the system being constructed. The fact that the language has a denotational semantics allows *facts* about the behavior of the system to be independently stated and verified. The operational semantics provides a means for directly *executing* these specifications to obtain the desired behavior. In this sense the language has much in common with Prolog, except that it applies to dynamic domains instead of static domains.

The system modeled in this paper has no database, and thus no storage for knowledge or beliefs. We have a practical implementation of a system that includes such a database, but have yet to formalize it. This introduces all the standard planning issues, such as the frame problem [14] and consistency maintenance [4]. We also need to investigate concurrency, and extend the model to deal with it. The notion of process failure and correctness assertions play a particularly important part when multiple agents or dynamic environments are allowed, and bear some relationship to formalisms for concurrent program verification. Some work in this direction is described by Georgeff [7].

References

[1] Allen, J. F., "A General Model of Action and Time," Computer Science Report TR 97, University of Rochester, Rochester, New York (1981).

[2] Allen, J.F., "Maintaining Knowledge about Temporal Intervals," *Comm. ACM,* Vol. 26, pp. 832-843 (1983).

[3] Davidson, D., *Actions and Events,* Clarendon Press, Oxford (1980).

[4] Doyle, J., "A Truth Maintenance System," *Artificial Intelligence* Vol. 12, No.3 (1979).

[5) Fikes, R. E., and Nilsson, N. J., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence* Vol. 2, pp. 189-208 (1971).

[6] Georgeff, M. P. and Bonollo, U., "Procedural Expert Systems," *Proceedings of the Eighth International Joint Conference on Artificial Intelligence,* Karlsruhe, West Germany (1983).

[7] Georgeff, M. P., "A Theory of Action for Multiagent Planning," *Proceedings of the Fourth National Conference on Artificial Intelligence,* Austin, Texas (1984).

[8] Georgeff, M.P. and Lansky, A.L., "Practical Reasoning Using Procedural Knowledge," forthcoming Technical Note, Artificial Intelligence Center, SRI International, Menlo Park, California (1985).

[9] Georgeff, M.P., and Lansky, A.L., "A Procedural Logic: Declarative and Procedural Semantics," forthcoming Technical Note, Artificial Intelligence Center, SRI International, Menlo Park, California (1985).

[10] Hendrix, G.G., "Modeling Simultaneous Actions and Continuous Processes," *Artificial Intelligence,* Vol. 4, pp. 145-180 (1973).

[11] Hoare, C.A.R., Brookes, S.D., and Roscoe, A.W. "A Theory of Communicating Sequential Processes," Technical Monograph PRG-16, Oxford University Computing Laboratory, Oxford, England.

[12] Kowalski, R., *Logic for Problem Solving,* North Holland Publishing Company, Now York, Now York (1979).

[13] McCarthy, J., "Programs with Common Sense," in *Semantic Information Processing,* M. Minsky (ed.), MIT Press, Cambridge, Massachusetts (1968).

[14] McCarthy, J., and Hayes, P.J., "Some Philosophical Problems from the Standpoint of Artificial Intelligence, in *Machine Intelligence 4*, pp. 463-502 (1969).

[15] McDermott, D., "A Tomporal Logic for Reasoning about Plans and Processes," Computer Science Research Report 196, Yale University, New Haven, Connecticut (1981).

[16] Moore, R.C., "Reasoning about Knowledge and Action," Technical Note 191, Artificial Intelligence Center, SRI International, Menlo Park, California (1980).

[17] Nilsson, N.J., "Triangle Tables: A Proposal for a Robot Programming Language", Technical Note 347, Artificial Intelligence Center, SRI International, Menlo Park, California (1985).

[18] Rosenschein, S.J., "Plan Synthesis: A Logical Perspective," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence,* pp. 331-337 (1981).

[19] Sacerdoti, ED. *A Structure for Plans and Behaviour,* Elsevier North Holland Publishing Company, New York, New York (1977).

[20] Stefik, M. "Planning with Constraints," *Artificial Intelligence,* Vol. 16, pp. 111-140 (1981).

[21] Tate, A. "Goal Structure - Capturing the Intent of Plans," *Proceedings of the Sixth European Conferencs on Artificial Intelligence,* pp. 273-276 (1984).

[22] Vere, S., "Planning in Time: Windows and Durations for Activities and Goals," Jet Propulsion Laboratory, Pasadena, California (1981).

[23] Wilkins, D.E., "Domain Independent Planning: Representation and Plan Generation," *Artificial Intelligence,* Vol. 22, pp. 269-301 (1984).