19

A Propositional Network Approach to Plans and Plan Recognition[*]

Stuart C. Shapiro, Deepak Kumar, and Syed S. Ali
Department of Computer Science
State University of New York at Buffalo
226 Bell Hall
Buffalo, NY 14260
(716) 636-3182

1 Introduction

In this paper, we discuss the current status of a planning/acting component for SNePS, the Semantic Network Processing [Shapiro, 1979; Shapiro and Rapaport, 1987; Shapiro and Group, 1989], and the design of a plan recognition system using this component. First, we present the design, implementation, and use of representations of plans to model a cognitive agent whose behavior is driven by its beliefs, desires, and intentions. We give the motivations underlying our representations for plans, goals, acts, actions, pre-conditions and post-conditions. These representations are designed to satisfy constraints posed by the issues involved in natural language understanding, belief representation, planning and problem solving, plan recognition, and text generation.

1.1 Overview of the system

Our work is proceeding by implementing, experimenting with, and revising a system called SNACTor (for SNePS Actor). SNACTor begins with an empty knowledge-base. In the role of informant, we interact with SNACTor using English sentences about the domain, instructing SNACTor about the various actions that it can perform, and how to solve problems in that domain. The input sentences are analyzed using a Generalized ATN grammar[Shapiro, 1982], the results of which are new beliefs in the knowledge-base. A generation part of the Generalized ATN grammar takes the new beliefs and expresses them back in English to

[*]This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB DC 20332 under Contract No. F30602-85-C-0008, which supports the Northeast Artificial Intelligence Consortium (NAIC).

indicate SNACTor's understanding to the informant. Requests to perform some action are sent to an acting executive that may then generate and execute a plan to fulfill the request. The informant may also ask questions about plans and the way the system would solve various problems.

2 Motivations underlying our representations

Our goals are to design and implement representations of plans to model a rational cognitive agent whose behavior is driven by its beliefs, desires, and intentions. We now give the motivations underlying our representations of plans, goals, acts, actions, pre-conditions and post-conditions. As mentioned before these representations are designed to satisfy constraints posed by issues in natural language understanding, belief representation, planning and problem solving, plan recognition, and text generation. A preliminary version of this work appears as an extended abstract in [Shapiro, 1988]. Since then, there have been changes in the design of our representations and planning techniques used. A detailed account of the syntax, and semantics of our earlier representations can be found in [Kumar *et al.*, 1988].

2.1 Motivation for intensional representations of plans

Georgeff (1987) mentions the importance of "considering planning systems as rational agents that are endowed with the psychological attitudes of belief, desire, and intention" and the problem of using appropriate semantics that give an *intensional* account of these notions. SNePS is an intensional propositional semantic network system [Shapiro and Rapaport, 1987] that has been used for cognitive modeling, belief representation and reasoning, belief revision, and natural language understanding. A basic principle of SNePS is the Uniqueness Principle—that there be a one-to-one mapping between nodes of the semantic network and concepts (mental objects) about which information may be stored in the network. These concepts are not limited to objects in the real world, but may be various ways of thinking about a single real world object, such as The Morning Star *vs.* The Evening Star *vs.* Venus. They may be abstract objects like properties, propositions, Truth, Beauty, fictional objects, and impossible objects. They may include specific propositions as well as general propositions, and even rules. Any concept represented in the network may be the object of propositions represented in the network giving properties of, or beliefs about it. For example, propositions may be the objects of explicit belief (or disbelief) propositions. Rules are propositions with the additional property that SNIP, the SNePS Inference Package, [McKay and Shapiro, 1981; Shapiro *et al.*, 1982] can use them to drive reasoning to derive additional believed propositions from previous believed propositions.

Plans are also mental objects. We can discuss plans with each other, reason about them, formulate them, follow them, and recognize when others seem to be following them. An AI system, using SNePS as its belief structure, should also be able to do these things. Requiring that the system be able to use a single plan representation for all these tasks puts severe constraints on the design of the representation. For instance, understanding natural language dialogue involving plans requires building plan representations from natural language input. In natural language, the explication of plans generally takes the form of a sequence of rather simple rules (*e. g.*, "If you see John, tell him I'm looking for him," "To pick up a block, you must first clear it") The full plan, including preconditions and effects of its component acts, must be constructed from such a sequence of rules.

Once constructed, a plan must be usable as a specification for the behavior of the agent, and must also be usable by the agent to understand other agents' actions. We are not treating plans as schedules of events for third parties (or multiple agents) [Lansky, 1987].

2.2 Intensional representations

We now give an overview of our representations and the motivations that led to them. We use "goal," "plan," "act," and "action" in particular ways, and distinguish among them. A *goal* is a proposition in one of two roles—either the role within another proposition that some *plan* is a plan for achieving that goal (making it true in the then current world), or the role as the object of the *act* of achieving it.

2.3 Planning is different from inference

We view a *plan* as a structured individual mental concept, *i.e.,* it is not a proposition or rule that might have a belief status. A plan is a structure of *acts.* (Among which may be the achieving of some goal or goals.) The structuring syntax for plans is a special syntax, differing, in particular, from that used for structuring reasoning rules. This is important both for semantic clarity and to allow a system to be implemented that can both reason and act efficiently. For contrast, consider standard (non-concurrent) Prolog or some arbitrary production rule system. Such a system relies on a semantic ambiguity between the logical & and the procedural **and then**. For example,

(1) $$p(X) \;:-\; q(X),\; r(X).$$

either means "For any X, $p(X)$ is true if $q(X)$ and $r(X)$ are true" or it means "For any X, to do p on X, first do q on X and then do r on X." Guaranteeing the proper ordering of behavior in the procedural interpretation is only possible by giving up the freedom to reorder, for efficiency, the derivations of $q(X)$ and $r(X)$ in the logical interpretation. The example is made more striking by appending

(2) $$q(Y) \;:-\; s(Y),\; t(Y).$$

(3) $$r(Z) \;:-\; s(Z),\; u(Z).$$

and considering the query

(4) $$?-\; p(a).$$

Under the logical interpretation, it would be efficient for the system to try finding if $s(a)$ holds only once, instead of once when rule 2 is being used and once when rule 3 is being used. This is the way SNIP has been implemented (see [McKay and Shapiro, 1981]). However, under the procedural interpretation, it may be perfectly reasonable to perform $s(a)$ twice, so the behavior that optimizes logical reasoning destroys procedural rule following. The fact that SNIP is optimized in this way for reasoning, and so cannot use its reasoning rules as procedural rules, was what originally motivated this project to design a planning/acting component for SNePS.

Believing is a state of knowledge; acting is the process of changing one state into another. Reasoning rules pass a *truth* or a *belief* status from antecedent to consequent, whereas acting rules pass an *intention* status from earlier acts to later acts. A reasoning rule can be viewed as a rule specifying an act—that of believing some previously non-believed proposition, but the believe action is already included in the semantics of the propositional connective, and, as pointed out above, there is no reason to believe a proposition more than once (unless it's disbelieved in the interim). The distinction between "believing and acting" in SNePS was first outlined in [Morgado and Shapiro, 1985].

2.4 The distinction between "acts" and "actions"

Lifschitz (1987) attempts to give a semantics of STRIPS by viewing STRIPS as a form of logic and STRIPS operators as rules of inference in this logic. For us, an *act* is a structured individual mental concept of something that can be performed by various actors at various times. This is important for plan recognition—we must be able to recognize that another agent is performing the same act that, if we were performing it, we would be in the midst of carrying out one of a certain number of plans. By the Uniqueness Principle, a single act must be represented by a single SNePS node, even if there are several different structures representing propositions that several different actors performed that act at different times. This argues for a representation of propositions more like that of Almeida [Almeida, 1987], rather than like more traditional case-based or frame-based representations. In what we are calling "more traditional representations", there is a structure representing the proposition with slots or arcs to the actor, the action, the object, *etc.* For example, to represent the proposition,

(s1) John walked to the store.

there would be four representational symbols, one for John, one for walking (or PTRANSing), one for the store, and one for the proposition itself, and the first three would be connected with the fourth in nearly similar ways at similar distances (measured by path length of arcs or slots). See Figure 1 for a SNePS representation based on Shapiro & Rapaport (1987). Almeida, however, took seriously that one could follow (s1) by

(s2) Mary did too.

and understand by that that John and Mary performed the same act—that of walking to the store. The representation for (s1) would have to introduce a fifth symbol, for walking to the store, which would be connected to the representation of the proposition at the same distance as the representation of John. Now, however, the symbols for walking and the store would be further from the symbol for the proposition (see Figure 2). When (s2) is processed, the symbol representing the proposition that Mary walked to the store would be connected to the same symbol for walking to the store used for (s1) (node M7 in Figure 3). This symbol represents what we are calling an act, and using it in the representation of both propositions follows by the Uniqueness Principle from interpreting (s1) and (s2) as saying that John and

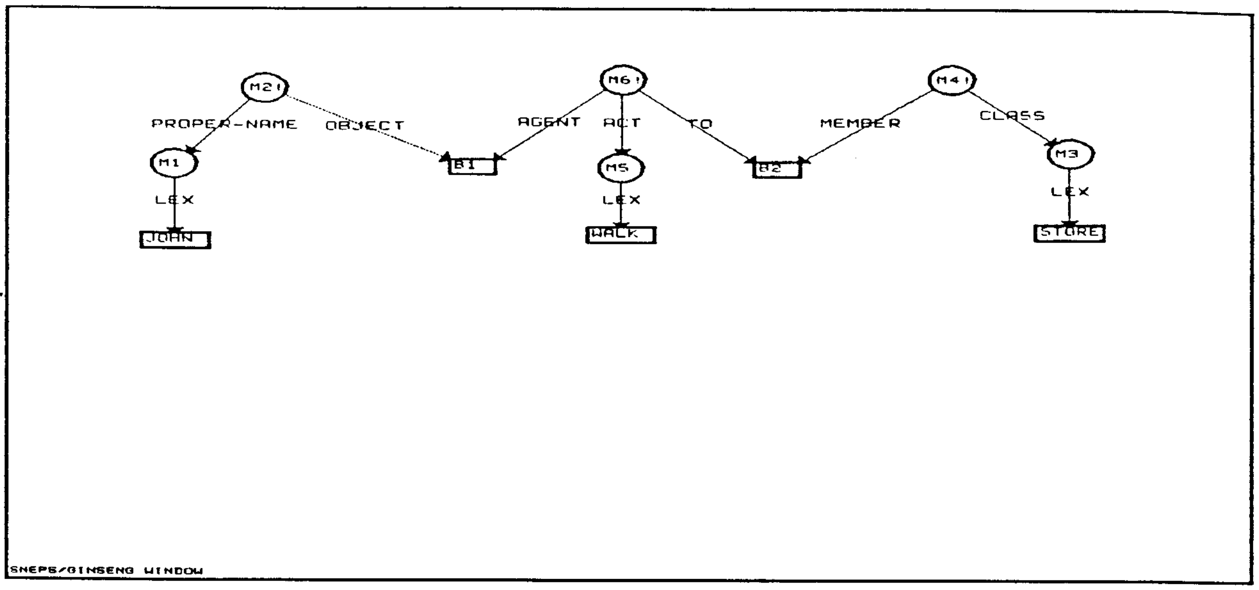

Figure 1: A traditional representation of "John walked to the store" (ignoring tense). Node B1 represents John; node B2 represents the store; node m6! represents the proposition that John walked to the store.

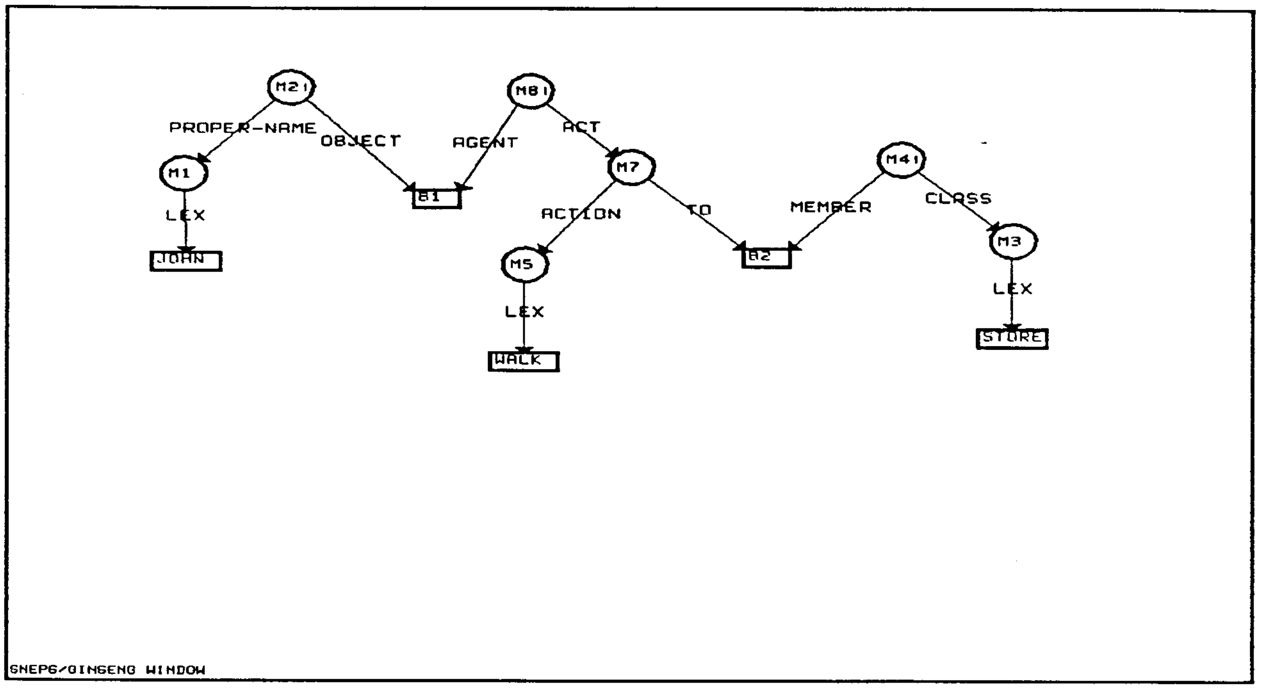

Figure 2: A representation of "John walked to the store" based on Almeida (1987). Node M7 represents the act of walking to the store; node M8! represents the proposition that John walked to the store.

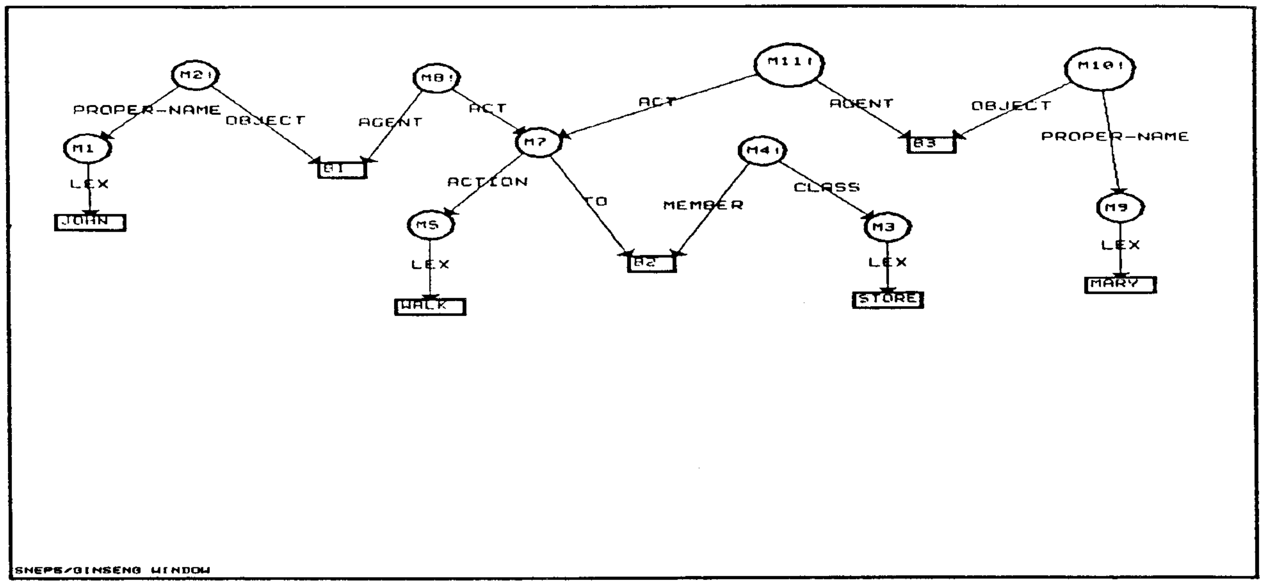

Figure 3: The network of Figure 2 with a representation of "Mary did too." added. Node M11! represents the proposition that Mary performed the very same act (represented by node M7) that John did.

Mary performed the same act. Moreover, if the network contains the representation of any plan that involves walking to the (same) store, that same act node would be used in the structure representing that plan (see Figure 4). Thus, John and Mary are directly connected to a plan that they may be engaged in.

An *action* is that component of an act that is what is done to the object or objects. In (s1) and (s2), the action is walking. Achieving some goal is an act whose action is achieving, and whose object is the particular proposition that is serving as the goal. Unfortunately for our remaining discussion, but consistently with what has gone before, one can only *perform* something that is an act (an action on an appropriate object), so instead of saying "performing an act whose action is x," we will say "performing the action x," and hope the reader will note the distinction between acts and actions.

Our representation of an act is a node with an ACTION arc to a node that represents the action, and OBJECT1, ..., OBJECTN arcs to the required objects of the action. Thus, the general syntax[1] of an act is

Syntax 1: *act* ::= ACTION: *action*
OBJECT1: *object1*
...
OBJECTN: *objectN*

[1]Specific actions might have their objects on differently labeled arcs. For example, in Figures 2–4 the WALK action uses a TO arc, and in Figure 4 the BUY and OBTAIN actions use OBJECT arcs.

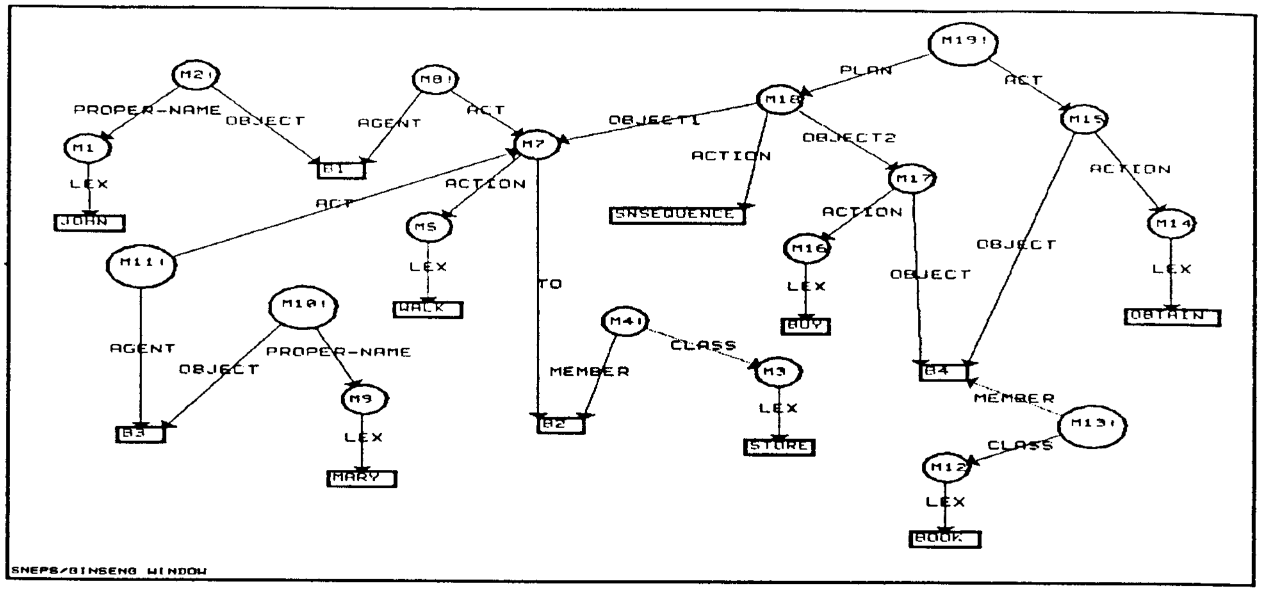

Figure 4: The network of Figure 3 with a representation of the proposition that a plan for obtaining some book is to walk to the store and buy the book. Node M19! represents the proposition using a syntax to be introduced in Section 2.5; node M18 represents a sequence of two acts, using a syntax to be introduced in Section 4; the first act in that sequence, represented by node M7, is the very same act that John and Mary did.

Semantics 1: *act* is a structured individual node representing the act whose action is *action* and *object1, ..., objectN* are the objects of *action*. For example, the SNePSUL (the SNePS User Language) command for building a node representing the act of saying "FOO" is:

```
(build action say object1 FOO)
```

2.5 Primitive and Complex actions

Any behaving entity has a repertoire of *primitive actions* it is capable of performing. We will say that an act whose action is primitive is a *primitive act*. That an action is primitive is a belief held by SNACTor after we tell it. The belief is represented in the form of an assertion saying that the action is a member of the class of primitive actions. This is similar to the MEMBER-CLASS proposition used by CASSIE [Shapiro and Rapaport, 1987]. For example, the SNePSUL command for asserting that saying is a primitive action is

```
(assert member say class primitive)
```

Non-primitive acts, which we will term *complex*, can only be performed by decomposing them into a structure of primitive acts, the syntax of which is the same procedural syntax as used in plans. That some plan *p* is a plan for carrying out some complex act *a*, is a proposition we can assert to SNACTor using the following representation:

Syntax 2: *plan-act-proposition* ::= ACT: a
PLAN: p

Semantics 2: *plan-act-proposition* is a proposition node that represents that the proposition p is a plan for carrying out act a.

The plan p is a structure of acts. The structuring syntax for plans is described in terms of control actions which are described later. That some plan p is a plan for achieving some goal g is also a proposition we can assert to SNACTor:

Syntax 3: *plan-goal-proposition* ::= GOAL: g
PLAN: p

Semantics 3: *plan-goal-proposition* is a proposition node that represents that p is a plan for achieving goal g.

The goal g is expressed as a domain specific proposition. Examples of these are given in a later section (see Section 4).

When the time comes for the agent to perform a complex act, it must find a plan that decomposes it. Using the above representations SNACTor may be told such plans. SNACTor is also capable of doing classical planning in case it does not already know any decompositions for a complex act. This is discussed later.

2.6 Pre– and post–conditions

The remaining notions we must consider are preconditions and effects (postconditions). Whether we think of them as pre- and post-conditions of plans or of acts is irrelevant since plans are kinds of acts. A pre-(post-)condition is just a proposition that must be (will be) true or false before (after) an act is performed. But the proposition that a proposition p is false is itself a proposition, so we can say that a pre-(post-)condition is a proposition that must be (will be) *true* before (after) an act is performed. (We will rely on SNeBR, the SNePS Belief Revision System [Martins and Shapiro, 1988a] to remove inconsistent beliefs after believing the effects of an act.) We have thus reduced the storage of pre- and post-conditions to two simple kinds of propositions:

Syntax 4: *precondition-proposition* ::= ACT: a
PRECONDITION: p

Semantics 4: *precondition-proposition* is a node that represents that the pre-condition of some act a is the proposition p.

For example,

```
(assert forall $block
        ant (build member *block class block)
        cq (build act (build action pickup object1 *block)
                 precondition (build property clear object *block)))
```

is the SNePSUL command to assert that before picking up any block it must be clear (i.e. nothing must be on top of it).

Syntax 5: *postcondition-proposition* ::= ACT: a
EFFECT: p

Semantics 5: *postcondition-proposition* is a node that represents that the post-condition of some act a is the proposition p.

For example,

```
(assert forall $block
        ant (build member *block class block)
        cq (build act (build action pickup object1 *block)
                  effect (build property holding object *block)))
```

is the SNePSUL command to assert that after picking up any block, it is being held.

Thus, effects and preconditions of an act are represented in the same way as other beliefs about other mental objects; we do not need a special data structure (or an operator formalism) for acts in which pre- and post-conditions are special fields. Such a representation also enables us to assert context-dependent effects of actions[Wilkins, 1988], i.e. the effects of doing some action are determined by the context in which the action is performed. For example,

```
(assert forall ($block $support)
        &ant ((build member *block class block)
              (build member *support class object)
              (build rel on arg1 *block arg2 *support))
        cq (build act (build action pickup object1 *block)
                  effect (build property clear object *support)))
```

asserts that if a block is on some support then after you pick up the block the support is clear. The scope of the context being referred to is the set of beliefs held by the system at the time the action is about to be performed. Using rules like these, context-dependency is guaranteed by ensuring that the effect is conditional on the antecedents being true before the act is performed. This is a more natural way of modeling actions and avoids the need for specifying multiple operators for doing the same action in different situations, which is a major criticism of earlier planners[Drummond, 1987].

2.7 Types of actions

We discussed three kinds of acts: a *primitive* act is unstructured and is in the repertoire of the agent; a *complex* act is unstructured—to perform it, the agent must find a plan for it; a *plan* is a structured act—the structure determines how the agent performs the component acts.

The structure of a plan can determine how the agent performs the component acts, because the structure, itself, is a primitive action.

Primitive actions fall into three classes:

- external actions that affect the world;
- mental actions that affect the agent's beliefs;
- control actions that affect agent's intentions.

External actions are domain specific actions like pickup, putdown etc. in the Blocksworld. The two mental actions that we have are *believing* a proposition, and *disbelieving* a proposition. Our repertoire of control actions includes *sequencing, conditional, iterative, and achieve* actions. A *sequencing* action represents the agent's *intention* to perform its object actions in a given sequence. The *conditional* and *iterative* actions are modeled after Dijkstra's guarded-if and guarded-loop commands respectively [Dijkstra, 1976]. The *achieve* action deduces plans for achieving some proposition and forms the intention of performing one of them. The *conditional* and *iterative* control actions enable the specification of non-linear partial plans. We have also designed a control action that can be used for posting constraints on plan variables (as in [Wilkins, 1988]).

2.8 Modeling external effects of actions

As mentioned above, external actions are domain specific actions that affect the outside world. For example, if the agent has an arm and is asked to pick up a block, the arm actually moves to the block, grasps it, and then lifts it up. Depending on the set of interfaces provided to the agent (like an arm, a speech synthesizer, etc.) we need to be able to carry out the action in the external world. This is done by writing Common Lisp functions that access the external interface. For instance, we can model the external effects of the "say" action by driving a speech synthesizer or by simply printing the message on the screen. The `define-primaction` function enables us to do this. Thus, to model saying something by printing it on the screen, we will have

```
(define-primaction say (n)
  "n is the node representing the act of saying. The node at the
   end of object1 arc is printed. choose.ns and pathfrom are
   SNePS interface functions to access parts of a structured node."
   (format t " ~A " (choose.ns (pathfrom 'object1 n))))
```

Thus when the agent executes the action represented by

```
(build action say object1 FOO)
```

the above code for say is executed, resulting in "FOO" appearing on the screen. How an action gets scheduled to be executed is discussed in the next section.

3 The Planning Paradigm

Besides having a current set of beliefs about the world, the system also has beliefs about plans for achieving goals, and about how complex actions can be decomposed into partial plans. The overall architecture of the system is similar to that of the PRS system [Georgeff, 1988].

The acting executive (called an *interpreter* or a *reasoning mechanism* in PRS) manipulates these components. It maintains an acting queue (referred to as a *process stack* in PRS) that contains all the scheduled actions to be performed as a part of some plan, thereby representing the system's *intentions.* The system can also form its own intentions in response to changing beliefs. SNIP, the SNePS inference package is used for several tasks: to find plans for complex tasks; as part of the achieve action, to find a plan to achieve some goal; and also as the *truth criterion* (also called the *question answering procedure*, see [Drummond and Tate, 1987]). Hence, it is used as the *plan decision procedure* in our system. SNIP is implemented on a simulated multi-processing system[McKay and Shapiro, 1980]. In the future, we will be able to do hypothetical reasoning using SNeBR [Martins and Shapiro, 1988b], for state-based plan projection.

3.1 The acting executive

We want the system to carry out plans, as well as to discuss them, reason about them, and recognize them. Certainly, since the system is currently without eyes, hands, or mobility, its repertoire of primitive actions is small, but, for now, as shown above, we can simulate other actions by appropriate printed messages. SNACTor, the acting system, is composed of a queue of acts to be carried out, and an acting executive. The queue of acts represents the system's intentions for carrying out the acts on the queue in that order. Intentions are formed by either an explicit request from a user to do something, or by committing to a plan that needs to be executed to fulfill a complex act or a goal. Explicit requests are made using the `perform` command. For example,

```
(perform (build action say FOO))
```

is an explicit request to the system to say "FOO" The act is put on the act-queue and the acting executive takes charge. Currently, the acting executive is the following loop:

```
while act-queue is not empty do
      if   the first-act on the act-queue has preconditions
           and they are not currently satisfied
           then insert the achieving of them on the front of the act-queue
      else remove the first-act from the act-queue;
           deduce   effects of first-act,
                    and insert the believing of them on the front
                    of the act-queue;
           if   first-act is primitive
                then perform it
           else deduce plans for carrying out first-act
                (using SNIP and available rules),
                choose one of them,
                and insert it on the front of the act-queue
           end if
      end if
end while
```

Notice that the effects of the act about to be performed are retrieved and scheduled to be believed before the act is actually performed. This guarantees that proper effects of the act are retrieved depending on the context that exists at that time. This flexibility in dynamically determining the effects of acts is what enables us to avoid having multiple operators for the same action.

When preconditions for an act exist and some of them are found not to be true, we schedule the achieving of all of them on the queue. The intention to perform the act is now pushed behind the intention to achieve these preconditions. Once all the preconditions are achieved, and we are ready to perform the act, they are checked again (just in case achieving some precondition renders another one false). Later on, we intend to incorporate critics, that will enable detecting of such conflicts and more sophisticated reasoning about plans.

From the above loop, it can be seen that at this stage of our work, we are assuming that a plan will be found for every complex act, and that every act will be successful. These assumptions will be removed as we proceed. SNACTor can also be made to do classical planning in case it is not able to find a plan to achieve a goal. This is done in the spirit of STRIPS[Fikes and Nilsson, 1971] by reasoning about effects of actions. As mentioned above, SNeBR can be used for hypothetical reasoning.

4 Syntax and semantics of control actions

We are now ready to examine the syntax and operational semantics of our current set of control actions.

Syntax 6: *sequence* ::= ACTION: SNSEQUENCE
OBJECT1: *act1*
OBJECT2: *act2*

This means that a *sequence* act is represented by a node with an ACTION arc to the node SNSEQUENCE, an OBJECT1 arc to an *act* node, and an OBJECT2 arc to another *act* node.

Semantics 6: *act*2 is inserted on the front of the act queue, and then *act*1 is inserted in front of it.

For example, a plan to get a block on a support is to pick it up and then put it down on the support. This can be derived using the *plan-goal-proposition* and *snsequence* as

```
(assert forall ($block $support)
        &ant ((build member *block class block)
              (build member *support class object))
        cq (build plan (build action snsequence
                              object1 (build action pickup object1 *block)
                              object2 (build action putdown
                                             object1 *block object2 *support))
                   goal (build rel on arg1 *block arg2 *support)))
```

Another example of a sequence is represented by node M18 in Figure 4. Since either or both of *act*1 and *act*2 can themselves be *snsequence* acts, we have a general structure for plans of sequential actions.

Syntax 7: *do-one* ::= ACTION: DO-ONE
OBJECT1:{*acti*}

This means that a *do-one* act is represented by a node with an ACTION arc to the node DO-ONE, and OBJECT1 arcs to an arbitrary number of *act* nodes.

Semantics 7: Chooses one *acti* and puts it on the front of the act queue. As currently implemented, the choice is arbitrary. However, we intend to implement a *do-one* that will reason about the *acti* and pick the "best" one.

For example, an act of giving an arbitrary greeting by saying "HELLO" or "JAMBO" or "G-DAY" can be expressed as

```
(build action do-one
       object1 ((build action say object1 HELLO)
                (build action say object1 JAMBO)
                (build action say object1 G-DAY)))
```

Syntax 8: *do-all* ::= ACTION: DO-ALL
OBJECT1: {*acti*}

Semantics 8: Forms the intention of doing all the *acti* by placing them on the front of the act queue in some unspecified order.

For example, an agent's "things-to-do-today" list can be represented using such an act as

```
(build action do-all
       object1 ((build action buy object *BOOK)
                (build action pay object *PHONE-BILL)
                (build action see object *NIAGARA-FALLS)))
```

Syntax 9: *conditional* ::= ACTION: SNIF
OBJECT1: {CONDITION: *propositioni*
THEN: *acti*}

This means that a *conditional* act is represented by a node with an ACTION arc to the node SNIF, and OBJECT1 arcs to an arbitrary number of nodes, each with a CONDITION arc to a *proposition* node and a THEN arc to an *act* node.

Semantics 9: If no *proposition* is true, does nothing. Otherwise, a *do-one* act whose objects are all the *acti* having their corresponding *propositioni* true is put on the front of the act queue. (Based on Dijkstra's guarded if [Dijkstra, 1976].)

For example, an act of saying "HELLO" contingent upon having permission can be expressed as

```
(build action snif
       object1 (build condition (build have permission)
                      then (build action say object1 HELLO)))
```

Syntax 10: *iteration* ::= ACTION: SNITERATE
OBJECT1: {CONDITION: *propositioni*
THEN: *acti*}

Semantics 10: If no *proposition* is true, does nothing. Otherwise, puts on the front of the act queue a *sequence* whose OBJECT1 is a *do-one* act whose objects are all the *acti* having their corresponding *propositioni* true, and OBJECT2 is the *iteration* node itself. (Based on Dijkstra's guarded loop [Dijkstra, 1976].)

For example, the act of repeatedly saying "HELLO" contingent upon having "hello-permission" and saying "THERE" contingent upon having "there-permission" can be expressed as

```
(build action sniterate
       object1 ((build condition (build have hello-permission)
                       then (build action snsequence
                         object1 (build action say object1 HELLO)
                         object2 (build action forget
                                object1 (build have hello-permission))))
                (build condition (build have there-permission)
                       then (build action snsequence
                         object1 (build action say object1 THERE)
                         object2 (build action forget
                                object1 (build have there-permission))))))
```

Syntax 11: *achieve* ::= ACTION: ACHIEVE
OBJECT1: *proposition*

Semantics 11: If *proposition* is true, does nothing. Otherwise, deduces plans for achieving *proposition*, chooses one of them, and puts it on the front of the act queue.

For example, in order to achieve a state in which BLOCKA is clear we'll have the act

```
(perform (build action achieve
                object1 (build property clear object BLOCKA)))
```

Thus, we can write plans for achieving goals as well as plans for decomposing a complex act. The domain normally determines the kinds of plans required (i.e. goal-based or act-decomposition based or both). However, as we will see, in the case of the blocksworld, and possibly in other domains, it may become hard to distinguish between something that characterizes a state and something that expresses an act. For example, "Clear BLOCKA" could be interpreted as a command to perform the act of clearing BLOCKA or a goal to achieve a state in which BLOCKA is clear. We are still exploring this issue. In any case, if required, we can model and use both interpretations.

Other control acts may be defined in the future, in particular a parameterized act that uses a sensory act to identify some object, and then performs some action on the identified object.

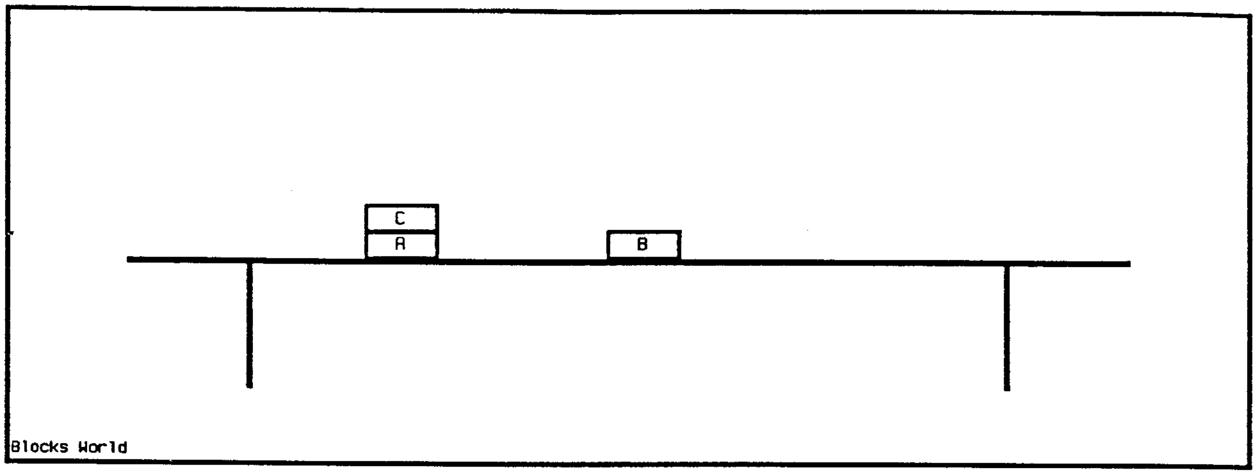

Figure 5: An initial Blocksworld situation

5 An Example

A natural language understanding component has been implemented in a Generalized ATN grammar[Shapiro, 1982] and is used for analyzing sentences and for generating English responses. SNACTor begins with an empty knowledge-base. In the role of informant, we interact with SNACTor using English sentences about the domain, instructing SNACTor about the various actions that it can do, and how to solve problems in that domain. A natural language generation grammar takes the new beliefs and expresses them back in English to show SNACTor's understanding to the informant. Requests to do some action are sent to an acting executive that may then generate and execute a plan to fulfill the request. The informant may also ask questions about plans and the way the system would solve various problems.

After instructing SNACTor about the various actions and plan formulations we describe a particular state, e.g.,

```
Input: Blocka is on the table. Blockb is clear and on the table.
       Blockc is clear and on blocka.

Response: I understand that blocka is on the table. I understand that
          blockb is clear. I understand that blockb is on the table.
          I understand that blockc is clear. I understand that blockc
          is on blocka.
```

For the Blocksworld, we can also instruct SNACTor about constructing a graphic analog of the world so that when we describe a state, it can actually construct it in a graphics window (shown in Figure 5). Now we are ready to ask SNACTor to do things. E.g.,

```
Input: Pile blocka on blockb on blockc.
```

```
Response: I understand that you want me to perform the act of piling
          on blocka and blockb and blockc.
```

SNACTor now goes into its acting executive which realizes that piling is a complex act and so needs to be decomposed. This is where the earlier dialog comes in handy and it finds decompositions:

```
Contd.: A plan to pile blocka on blockb on blockc is to achieve
        that blockc is on the table and then achieve that blockb is
        on blockc and then achieve that blocka is on blockb.
```

And this decomposition continues depth-first until it finds an appropriate action to execute

```
Contd.: Want to achieve blockc is on the table.
        ...
        Want to achieve blockc is held.
        ...
        Want to achieve blockc is clear. Already achieved.
        Unstack blockc from blocka. Disbelieve blockc is on blocka.
        Believe blocka is clear. Disbelieve blockc is clear. Believe
        blockc is held. Putdown blockc on the table.
        ...
```

Effects of an action are derived as and when an action is performed. SNACTor can also be made to do classical planning in case it is not able to find a plan to achieve a goal. This is done in the spirit of STRIPS[Fikes and Nilsson, 1971] by reasoning about effects of actions. As mentioned above, SNeBR can be used for hypothetical reasoning.

5.1 Discussing plans

We have seen how we can instruct SNACTor about planning in a domain and how we can describe situations to it and subsequently ask it to do things by using the plans it derives. We can also discuss plans with SNACTor by asking questions about solving problems in specific or generic situations. E.g.,

```
Input: How would you pile blocka on blockb on blockc?
       How would you clear a block?
```

And SNIP, the plan decision procedure, will derive an appropriate plan and respond to the query.

6 Plan Recognition

In this section we will briefly look at the beginning of a plan recognition component based on our representations. The initial idea that we are exploring is based on the observation that if an actor is performing an act that, when we perform it we are are in the process of executing some plan, the actor could possibly be performing the act as a part of a similar plan. Expressed more clearly, we have the rule

if an actor x performs an act $a1$,
  and $a1$ is a PLAN-COMPONENT of a proposition p
  then if $a2$ is the ACT of p
      then x may be engaged in carrying out $a2$
  and if g is a GOAL of a proposition p
      then x may be trying to achieve g.

We can express what it means to be a plan-component using SNePS path-based inference [Shapiro, 1978; Srihari, 1981] as:

```
(define-path PLAN-COMPONENT
  (compose PLAN
          (kstar (or (compose (kstar OBJECT2) (or OBJECT1 OBJECT2))
                     (compose OBJECT1 THEN)))))
```

This defines a virtual arc PLAN-COMPONENT to be one that goes from a *plan-act-proposition* or a *plan-goal-proposition* to every act within the plan. For example, the following SNePS node represents that the act of greeting someone (`give-greetings`) can be accomplished by a plan to repeatedly say "HELLO THERE" thus using the act defined in Section 4 above.

```
(M36! (ACT GIVE-GREETINGS)
 (PLAN
  (M32 (ACTION (M6 (LEX SNITERATE)))
   (OBJECT1
    (M27 (CONDITION (M21 (LEX PERMISSION)))
     (THEN
      (M26 (ACTION (M1 (LEX SNSEQUENCE)))
           (OBJECT1 (M18 (ACTION (M10 (LEX SAY))) (OBJECT1 HELLO)))
       (OBJECT2 (M24 (ACTION (M16 (LEX FORGET))) (OBJECT1 (M21)))))))
    (M31 (CONDITION (M28 (LEX PERMISSION2)))
     (THEN
      (M30 (ACTION (M1)) (OBJECT1 (M19 (ACTION (M10)) (OBJECT1 THERE)))
       (OBJECT2 (M29 (ACTION (M16)) (OBJECT1 (M28))))))))))
```

Next we give the plan recognition rule to SNACTor. This rule says that if someone is doing an act which is part of some plan, assume that that person is engaged in the plan.

```
(assert forall ($agent $reported-act $planned-act)
        &ant ((build agent *agent act *reported-act)
              (build plan-component *reported-act act *planned-act))
        cq (build agent *agent act *planned-act))
```

Now we can tell the system that John performed the act of saying "HELLO"

```
(add agent john act (build action say object1 HELLO))
```

and ask about the act(s) that John performed

```
(describe (deduce agent john act $johns-acts))
(M38! (ACT (M18 (ACTION (M10 (LEX SAY))) (OBJECT1 HELLO))) (AGENT JOHN))
(M52! (ACT GIVE-GREETINGS) (AGENT JOHN))
 CPU time : 6.65     GC time : 0.00
```

As we can see, it comes back with a response saying that John is performing the acts of saying "HELLO" as well as `give-greetings`. We do not yet have a way of dealing with "may be engaged in" nor with "may be trying to achieve," but this rule indicates our approach to plan recognition within the design of the planning/acting SNePS component described in this report.

7 Discussion

Our goal is to model a rational cognitive agent whose behavior is driven by its beliefs, desires, and intentions. We want our agent to do natural language understanding, reason about beliefs, act rationally based on its beliefs, do plan recognition, and plan based text generation. Doing all these tasks in a single coherent framework poses several constraints. We are discovering that SNePS and its underlying theories contribute effectively towards our goal. We have designed and implemented intensional propositional representations for plans. This is a major advancement over operator-based descriptions of plans. Operator-based formulations of actions tend to alienate the discussion of operators themselves. Operators are usually specified in a different language than that used for representing beliefs about states. Moreover, plans (or procedural networks) constructed from these operators can only be accessed by specialized programs (critics, executors) and, like operators, are represented in still another formalism. Our representations for acts, actions, goals, and plans build upon and add to the intensional propositional representations of SNePS. This framework enables us to tackle various tasks mentioned above in a uniform and coherent fashion.

Our current system is being advanced in several directions. In the context of planning, there are issues associated with conjunctive goals[Waldinger, 1977], non-linear plans [Sacerdoti, 1977; Tate, 1977; Drummond and Tate, 1987], and dealing with the effects of actions. As mentioned in [Drummond, 1987] explicitly specifying the disbelieving of propositions as a result of performing some action is not natural. We propose to use belief revision (SNeBR) to detect inconsistencies after asserting the effects of an action.

Language used in planning contexts, is slightly more constrained than in arbitrary discourse. Sentences describing plans tend to be declarative, with a syntactically decomposable structure involving goal, effect, and plan definition. Handling reference is simplified by the assumption that common noun phrases correspond to typed variables. Indefinite noun phrases introduce new variables, definite noun phrases refer to previously introduced variables. Natural language generation of plans and rules involves careful selection of relevant attributes of these variables.

8 Summary

In this report, we have described the design, and aspects of the implementation, of an intensional representation for plans. These representations have been constrained by issues in cognitive modeling, belief representation, reasoning, and natural language understanding. Plans are structured individual mental concepts, consisting of a structure of acts. Acts are structured individual mental concepts of an action process independent of actor and time. Actions are primitive or complex and fall into three classes—external, mental, and control. The system models intentionality with a queue of acts, and may form new intentions based on its current belief status. Currently, we have an implementation of a Blocksworld involving natural language dialogues about plans, planning and execution of Blocksworld plans, and some examples of plan recognition.

References

[Almeida, 1987] Michael J. Almeida. *Reasoning About the Temporal Structure of Narratives.* PhD thesis, Department of Computer Science, SUNY at Buffalo, Buffalo, NY, 1987. Technical Report No. 87-10.

[Dijkstra, 1976] Edsger W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, Englewood Cliffs, NJ, 1976.

[Drummond and Tate, 1987] Mark Drummond and Austin Tate. *AI Planning: A Tutorial and Review.* Technical Report AIAI-TR-30, Artificial Intelligence Applications Institute, University of Edinburgh, Edinburgh, November 1987.

[Drummond, 1987] Mark E. Drummond. A representation of action and belief for automatic planning systems. In Michael P. Georgeff and Amy L. Lansky, editors, *Reasoning about Actions and Plans - Proceedings of the 1986 Workshop*, pages 189–212, AAAI and CSLI, Morgan Kauffmann, Los Altos, CA, 1987.

[Fikes and Nilsson, 1971] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 5:189–208, 1971.

[Georgeff, 1987] Michael P. Georgeff. Planning. In *Annual Reviews of Computer Science Volume 2*, pages 359–400, Annual Reviews Inc., Palo Alto, CA, 1987.

[Georgeff, 1988] Michael P. Georgeff. An embedded reasoning and planning system. In Jay Webber, Josh Tenenberg, and James Allen, editors, *Advance Proceedings of The Rochester Planning Workshop–From Formal Systems to Practical Systems*, pages 79–101, October 1988.

[Kumar *et al.*, 1988] Deepak Kumar, Syed S. Ali, and Stuart C. Shapiro. Discussing, Using, and Recognizing Plans in SNePS–Preliminary Report. In P. V. S. Rao and P. Sadanandan, editors, *Modern Trends in Information Technology—Proceedings of the Seventh Biannual Convention of South East Asia Regional Computer Confederation (SEARCC88)*, pages 177–182, Tata McGraw-Hill Publishing Company, New Delhi, India, 1988.

[Lansky, 1987] Amy L. Lansky. A representation of parallel activity based on events, structure, and causality. In Michael P. Georgeff and Amy L. Lansky, editors, *Reasoning about Actions and Plans - Proceedings of the 1986 Workshop*, pages 123–160, AAAI and CSLI, Morgan Kauffmann, Los Altos, CA, 1987.

[Lifschitz, 1987] Vladimir Lifschitz. On the semantics of STRIPS. In Michael P. Georgeff and Amy L. Lansky, editors, *Reasoning about Actions and Plans - Proceedings of the 1986 Workshop*, pages 1–10, AAAI and CSLI, Morgan Kauffmann, Los Altos, CA, 1987.

[Martins and Shapiro, 1988a] João P. Martins and Stuart C. Shapiro. A model for belief revision. *Artificial Intelligence*, 35(1):25–79, May 1988.

[Martins and Shapiro, 1988b] J. P. Martins and S. C. Shapiro. A model for belief revision. *Artificial Intelligence*, 35(1):25–79, 1988.

[McKay and Shapiro, 1980] D. P. McKay and S. C. Shapiro. MULTI — a LISP based multiprocessing system. In *Proceedings of the 1980 LISP Conference*, pages 29–37, Stanford University, Stanford, CA, 1980.

[McKay and Shapiro, 1981] Donald P. McKay and Stuart C. Shapiro. Using active connection graphs for reasoning with recursive rules. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 368–374, Morgan Kaufmann, Los Altos, CA, 1981.

[Morgado and Shapiro, 1985] E. J. Morgado and S. C. Shapiro. Believing and acting: a study of meta-knowledge and meta-reasoning. In *Proceedings of EPIA-85 ("Encontro Portugues de Inteligencia Artificial")*, pages 138–154, Oporto, Portugal, September 1985.

[Sacerdoti, 1977] Earl D. Sacerdoti. *A Structure for Plans and Behavior.* Elsevier North Holland, New York, NY, 1977.

[Shapiro, 1978] S. C. Shapiro. Path-based and node-based inference in semantic networks. In D. Waltz, editor, *Tinlap-2: Theoretical Issues in Natural Languages Processing*, pages 219–225, ACM, New York, 1978.

[Shapiro, 1979] S. C. Shapiro. The SNePS semantic network processing system. In N. V. Findler, editor, *Associative Networks: The Representation and Use of Knowledge by Computers*, pages 179–203, Academic Press, New York, 1979.

[Shapiro, 1982] S. C. Shapiro. Generalized augmented transition network grammars for generation from semantic networks. *The American Journal of Computational Linguistics*, 8(1):12–25, 1982.

[Shapiro, 1988] Stuart C. Shapiro. Representing plans and acts. In John W. Esch, editor, *Proceedings of the Third Annual Workshop on Conceptual Graphs*, Sponsored by AAAI, St. Paul, MN, August 1988.

[Shapiro and Group, 1989] S. C. Shapiro and The SNePS Implementation Group. *SNePS-2 User's Manual.* Department of Computer Science, SUNY at Buffalo, 1989.

[Shapiro and Rapaport, 1987] S. C. Shapiro and W. J. Rapaport. SNePS considered as a fully intensional propositional semantic network. In N. Cercone and G.

McCalla, editors, *The Knowledge Frontier*, pages 263–315, Springer–Verlag, New York, 1987.

[Shapiro *et al.*, 1982] Stuart C. Shapiro, João P. Martins, and Donald P. McKay. Bi-directional inference. In *Proceedings of the Fourth Annual Meeting of the Cognitive Science Society*, pages 90–93, Ann Arbor, MI, 1982.

[Srihari, 1981] * R. Srihari. *Combining Path-based and Node-based Reasoning in SNePS.* Technical Report 183, Department of Computer Science, SUNY at Buffalo, 1981.

[Tate, 1977] A. Tate. Generating project networks. In *Proceedings 5th IJCAI*, pages 888–93, 1977.

[Waldinger, 1977] R. Waldinger. *Achieving Several Goals Simultaneously*, pages 94–136. Ellis Horwood, Chichester, England, 1977.

[Wilkins, 1988] David E. Wilkins. *Practical Planning–Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, Palo Alto, CA, 1988.