PROCEEDINGS

Third Annual Workshop on

Conceptual Graphs

Sponsored by AAAI in conjunction with AAAI – 88
445 Burgess Dr. Menlo Park, CA 94025
(415) 328–3123

August 27, 1988
St. Paul, Minnesota

Ed. John W. Esch, Unisys

Representing Plans and Acts*

Stuart C. Shapiro
Department of Computer Science
State University of New York at Buffalo
226 Bell Hall
Buffalo, NY 14260-7022
(716) 636-3182
shapiro@cs.buffalo.edu

July 14, 1988

In this paper, I discuss the current status of a planning/acting component for SNePS, the Semantic Network Processing System [8,10]. First, I will motivate our representation of plans, goals, acts, actions, pre-conditions and post-conditions. Second, I will present the acting executive loop that can carry out these plans. Third, I will present the syntax and semantics of our representation of the primitive control acts that constitute the structure of our plans. Last, I will present a rule that is the beginning of a plan recognition component based on this representation.

In SNePS, all concepts represented in the network are represented as nodes. Labelled arcs of a SNePS network represent non-conceptual binary relations between nodes. The basic meaning of a node may be determined by the set of arcs emanating from it, the nodes they go to, the arcs emanating from those nodes, etc. In comparing SNePS networks and conceptual graphs, Sowa states, "Although the diagrams look very different, there is a direct mapping between them" [11, p. 139]. Given that, this paper will use SNePS terminology.

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, [5,9] 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 representation.

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. This will become clearer as we proceed.

A *plan* is 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.)

*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). The author is grateful to Dr. Norman K. Sondheimer of USC/Information Sciences Institute for his hospitality and support during the author's sabbatical year.

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) $$r(Z) : -s(Z), u(Z)$$

Under the logical interpretation, it would be efficient for the system to try finding true instances of $s(X)$ only once, instead of once when rule 2 is being used and once when rule 3 is being used. This is, in fact, the way SNIP has been implemented (see [5]). However, under the procedural interpretation, it is perfectly reasonable to perform $s(X)$ twice for a given X, 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. The plan structuring syntax we have designed is discussed below.

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 very same act which, 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 [1], rather than like more traditional case-based or frame-based representations. In what I am 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 approximately similar ways at similar distances (measured by path length of arcs or slots). Almeida, however, took seriously the fact 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. When (s2) is processed, the symbol representing the proposition that Mary walked to the store would be connected to the very same symbol for walking to the store used for (s1). This symbol represents what I am calling an act, and using it in the representation of both propositions follows by the Uniqueness Principle from interpreting (s1) and (s2) as saying the John and 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. Thus, John and Mary are rather directly connected to a plan that they may be engaged in.

Finally, 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," I will say "performing the action x," and hope the reader will note the distinction between acts and 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*. Non-primitive acts, which we will term *complex*, can only be performed by decomposing them into a structure of primitive acts. The syntax of that structure is the same procedural syntax as used in plans. So we close the inductive definition of plans by including plans among the acts, and note that a plan can be a plan for achieving some goal, or it can be a plan for performing some complex act. That some plan p is a plan for achieving some goal g is a proposition. Also, that some plan p is a plan for carrying out some complex action a, is a proposition. We have already designed representations for several different types of propositions in SNePS (see [10]), so we have now almost finished a tour of plans and acts with the only radically new syntactic structure needed being that of plans.

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 [4] 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: the pre-condition of some act a is the proposition p; the post-condition of some act a is the proposition p. That is, 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 for acts in which pre- and post-conditions are special fields.

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, we can simulate other actions by appropriate printed messages. The acting system is composed of a queue of acts to be carried out, and an acting executive, which currently is the following loop:

```
while act-queue is not empty do
      if   the first-act on the act-queue has preconditions
           then insert the achieving of them on the front of the act-queue
      else remove the first-act from the act-queue;
           retrieve 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
```

From this 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. Also at this stage, choosing one of a set of alternative plans for carrying out a complex action is done arbitrarily, unless one of the set is the no-op action of doing nothing, in which case it is chosen.

Primitive actions fall into three classes: external actions that affect the world; mental actions that affect the system's beliefs; control actions that affect the acting queue. At this point, the only external action that our system can actually perform is printing something on the screen; all other external actions are simulated by printing an appropriate message. The two mental actions we have implemented are *believing* a proposition, and *disbelieving* a proposition. The syntax and operational semantics of our current set of control actions are:

Syn. 1: *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.

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

Syn. 2: *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.

Sem. 2 If no *proposition* is true, does nothing. Otherwise, arbitrarily chooses one *acti* whose corresponding *propositioni* is true, and puts it on the front of the act queue. (Based on Dijkstra's guarded if [2].)

Syn. 3: *iteration* ::= ACTION: SNITERATE
OBJECT1: {CONDITION: *propositioni*
THEN: *acti*}

Sem. 3 If no *proposition* is true, does nothing. Otherwise, arbitrarily chooses one *acti* whose corresponding *propositioni* is true, and puts on the front of the act queue a sequence whose OBJECT1 is *acti* and whose OBJECT2 is the *iteration* node itself. (Based on Dijkstra's guarded loop [2].)

Syn. 4: *achieve* ::= ACTION: ACHIEVE
OBJECT1: *proposition*

Sem. 4 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.

Syn. 5: *no-op* ::= ACTION: NOOP

Sem. 5 Does nothing.

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.

Notice that deduction is used in two places: in the executive loop to find plans for complex acts; and as part of the achieve action, to find a plan to achieve some goal. This constitutes the active planning the system does. When the project advances to the point that hypothetical reasoning is needed for planning, SNeBR will be used as described in [3].

The two propositions that relate plans to complex acts and to goals are represented as follows:

Syn. 6: *plan-act-proposition* ::= PLAN: *act1*
ACT: *act2*

Sem. 6 *act1* is a plan for carrying out *act2*.

Syn. 7: *plan-goal-proposition¿* ::= PLAN: *act*
GOAL: *proposition*

Sem. 7 *act* is a plan for achieving *proposition*.

An examination of the above syntax shows that the SNePS path-based inference [7,12] rule:

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

defines the 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. Therefore, an initial rule for plan recognition is:

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 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 initial approach to plan recognition.

The representation shown in this paper has been implemented in SNePS-2, a new implementation of SNePS written in Common Lisp and running on HP 9000 series workstations, Texas Instrument Explorers, and Symbolics Lisp Machines. Simple plans have been represented and carried out by the new SNePS acting component. The plan recognition rule given above has been tested and has worked. A Generalized Augmented Transition Network parsing/generation grammar [6] has been written to interact with SNePS and its planning/acting component in the domain of the blocks world.

References

[1] 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.

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

[3] João P. Martins and Stuart C. Shapiro. Hypothetical reasoning. In *Applications of Artificial Intelligence to Engineering Problems: Proceedings of The 1st International Conference*, pages 1029–1042, Springer-Verlag, Berlin, 1986.

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

[5] 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.

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

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

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

[9] 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.

[10] Stuart C. Shapiro and William J. Rapaport. SNePS considered as a fully intensional propositional semantic network. In Nick Cercone and Gordon McCalla, editors, *The Knowledge Frontier: Essays in the Representation of Knowledge*, pages 262–315, Springer-Verlag, New York, 1987.

[11] John F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley, Reading, MA, 1984.

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

APPENDIX A

```
> (sneps)

   Welcome to SNePS-2.0
   8/17/1988 17:46:47

*(demo "sneps2.snactor;snactor.demo")
File sneps2.snactor;snactor.demo
is now the source of input.
 CPU time : 0.38     GC time : 0.00

;;; Basic SNACTOR network

;;; Required arcs
(define action lex object1 object2  object3
 act plan goal effect then condition until
 do member class)

(ACTION LEX OBJECT1 OBJECT2 OBJECT3 ACT
  PLAN GOAL EFFECT THEN CONDITION UNTIL
  DO MEMBER CLASS)
 CPU time : 0.85     GC time : 0.00

;;; Declaration of primitive actions
(describe
  (assert member (build lex snsequence) = SNSEQ
          class (build lex primitive) = PRIMITIVE))

(M3! (CLASS (M2 (LEX PRIMITIVE)))
     (MEMBER (M1 (LEX SNSEQUENCE))))
 CPU time : 1.08     GC time : 0.00

(describe
  (assert member (build lex snif) = SNIF
          class *PRIMITIVE))

(M5! (CLASS (M2 (LEX PRIMITIVE)))
     (MEMBER (M4 (LEX SNIF))))
 CPU time : 0.38     GC time : 0.00

(describe
  (assert member (build lex sniterate)
          = SNITERATE
          class *PRIMITIVE))

(M7! (CLASS (M2 (LEX PRIMITIVE)))
     (MEMBER (M6 (LEX SNITERATE))))
 CPU time : 0.20     GC time : 0.00

(describe
  (assert member (build lex achieve) = ACHIEVE
          class *PRIMITIVE))
(M9! (CLASS (M2 (LEX PRIMITIVE)))
     (MEMBER (M8 (LEX ACHIEVE))))
 CPU time : 0.30     GC time : 0.00

(describe
  (assert member (build lex say) = SAY
          class *PRIMITIVE))

(M11! (CLASS (M2 (LEX PRIMITIVE)))
      (MEMBER (M10 (LEX SAY))))
 CPU time : 0.23     GC time : 0.00

(describe
  (assert member (build lex noop) = NOOP
          class *PRIMITIVE))

(M13! (CLASS (M2 (LEX PRIMITIVE)))
      (MEMBER (M12 (LEX NOOP))))
 CPU time : 0.28     GC time : 0.00

(describe
  (assert member (build lex believe) = BELIEVE
          class *PRIMITIVE))

(M15! (CLASS (M2 (LEX PRIMITIVE)))
      (MEMBER (M14 (LEX BELIEVE))))
 CPU time : 0.18     GC time : 0.00

(describe
  (assert member (build lex forget) = FORGET
          class *PRIMITIVE))

(M17! (CLASS (M2 (LEX PRIMITIVE)))
      (MEMBER (M16 (LEX FORGET))))
 CPU time : 0.32     GC time : 0.00

;;; Some tests
(describe
  (build action *say
         object1 hello)) = say-hello

(M18 (ACTION (M10 (LEX SAY)))
     (OBJECT1 HELLO))
 CPU time : 0.23     GC time : 0.00

(snact *say-hello)

 HELLO
 CPU time : 0.93     GC time : 0.00

(describe
  (build action *say
         object1 there)) = say-there

(M19 (ACTION (M10 (LEX SAY)))
     (OBJECT1 THERE))

(snact
  (build action *snseq
         object1 *say-hello
         object2 *say-there))

 HELLO
 THERE
 CPU time : 0.47     GC time : 0.00

(describe
  (build action *snif
         object1
           (build condition
                  (build lex permission) = permission
                  then *say-hello)))

(M23 (ACTION (M4 (LEX SNIF)))
     (OBJECT1
      (M22 (CONDITION (M21 (LEX PERMISSION)))
           (THEN (M18 (ACTION (M10 (LEX SAY)))
                      (OBJECT1 HELLO))))))
 = if-have-permission-say-hello
 CPU time : 0.02     GC time : 0.00

(snact *if-have-permission-say-hello)
```

```
 CPU time : 0.37     GC time : 0.00

(assert lex permission)
(M21!)
 CPU time : 0.02     GC time : 0.00

(snact *if-have-permission-say-hello)

 HELLO
 CPU time : 0.42     GC time : 0.00

(snact
  (build action *forget
         object1 *permission))

Now doing:  DISBELIEVE:
(M21 (LEX PERMISSION))
 CPU time : 0.42     GC time : 0.00

(snact *if-have-permission-say-hello)

 CPU time : 0.30     GC time : 0.00

(snact
  (build action *believe
         object1 *permission))

Now doing: BELIEVE:
(M21! (LEX PERMISSION))
 CPU time : 0.38     GC time : 0.00

(snact *if-have-permission-say-hello)

 HELLO
 CPU time : 0.60     GC time : 0.00

(describe
  (build action *sniterate
         object1
           ((build
      condition *permission
      then
(build action *snseq
       object1 *say-hello
       object2
 (build action
 *forget
object1
 *permission)))
    (build
              condition
                (build lex permission2)
 = permission2
              then
(build action *snseq
       object1 *say-there
       object2
 (build
   action *forget
   object1
     *permission2))))))

(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)))))))))
 = repeatedly-with-permission-say-hello-there
 CPU time : 0.12     GC time : 0.00

(snact *repeatedly-with-permission-say-hello-there)

 HELLO

Now doing:  DISBELIEVE:
(M21 (LEX PERMISSION))
 CPU time : 2.23     GC time : 0.00

(snact
  (build action *believe
         object1 *permission2))

Now doing: BELIEVE:
(M28! (LEX PERMISSION2))
 CPU time : 0.45     GC time : 0.00

(snact *repeatedly-with-permission-say-hello-there)

 THERE

Now doing:  DISBELIEVE:
(M28 (LEX PERMISSION2))
 CPU time : 2.78     GC time : 0.00

(snact
  (build action *believe
         object1 *permission))

Now doing: BELIEVE:
(M21! (LEX PERMISSION))
 CPU time : 0.50     GC time : 0.00

(snact
  (build action *believe
         object1 *permission2))

Now doing: BELIEVE:
(M28! (LEX PERMISSION2))
 CPU time : 0.60     GC time : 0.00

(snact *repeatedly-with-permission-say-hello-there)

 HELLO

Now doing:  DISBELIEVE:
(M21 (LEX PERMISSION))

 THERE

Now doing:  DISBELIEVE:
(M28 (LEX PERMISSION2))
 CPU time : 6.50     GC time : 0.00

;;; Beginning of plan recognition
;;;
```

```
;;; A plan node has plan-act or plan-goal arcs
;;; the plan arc points to an act node
(define plan-component)
(PLAN-COMPONENT)
 CPU time : 0.23     GC time : 0.00

 ;;; the plan-component virtual arc points
 ;;; from a plan node to the act nodes
 ;;; within its plan-act
(define-path plan-component
  (compose plan
          (kstar (or (compose
                         (kstar object2)
                         (or object1 object2))
                        (compose object1 then)))))

PLAN-COMPONENT implied by the path
(COMPOSE
  PLAN (KSTAR (OR
       (COMPOSE
(KSTAR OBJECT2)
(OR OBJECT1 OBJECT2))
       (COMPOSE OBJECT1 THEN))))

PLAN-COMPONENT- implied by the path
(COMPOSE
  (KSTAR (OR
  (COMPOSE (OR OBJECT1- OBJECT2-)
   (KSTAR OBJECT2-))
  (COMPOSE THEN- OBJECT1-))) PLAN-)
 CPU time : 0.22     GC time : 0.00

(describe
 (assert
  act give-greetings
  plan
  *repeatedly-with-permission-say-hello-there))

(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)))))))))))
 CPU time : 0.73     GC time : 0.00

(define agent)
(AGENT)
 CPU time : 0.02     GC time : 0.00

; If someone is doing an act which
; is part of some plan, assume that person
; is engaged in the plan.
(describe
  (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)))

(M37! (FORALL V39 V40 V41)
      (&ANT (P39 (ACT V40)
                 (AGENT V39))
         (P40 (ACT V41) (PLAN-COMPONENT V40)))
      (CQ (P41 (ACT V41) (AGENT V39))))
 CPU time : 0.45     GC time : 0.00

(describe
  (add agent john
       act *say-hello))

(M38! (ACT (M18 (ACTION (M10 (LEX SAY)))
                (OBJECT1 HELLO)))
      (AGENT JOHN))
 CPU time : 1.80     GC time : 0.00

(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
```

APPENDIX B1

```
> (snaps)

   Welcome to SNePS-2.0
   8/1/1988 17:42:31

(^ (parse -1))
 ATN parser initialization...
 Input sentences in normal English orthographic
convention. May go beyond a line by having a
space followed by a <CR>
To exit parser, write ^end.

;;; Basic SNACTOR network that defines
;;; a Blocksworld.

 : picking up is a primitive act.
I understand that pickup is a primitive act.
 Time (sec.): 6.9

 : putting down is a primitive act.
I understand that putdown is a primitive act.
 Time (sec.): 5.35

 : stacking is a primitive act.
I understand that stack is a primitive act.
 Time (sec.): 4.4166665

 : unstacking is a primitive act.
I understand that unstack is a primitive act.
 Time (sec.): 4.4333334

;;; Effects of acts...

 : after picking up a block
   the block is not clear

I understand that for every V1 ,
        after performing pickups on V1 ,
        exactly 0 of the following
        are true: V1 is clear.
 Time (sec.): 14.2

 : after picking up a block
   the block is not ontable

I understand that for every V1 ,
        after performing pickups on V1 ,
        exactly 0 of the following
        are true: V1 is ontable.
 Time (sec.): 13.583333

;;; Effects of acts......contd.

 : after picking up a block the block is held

I understand that for every V1 ,
        after performing pickups on V1 ,
        V1 is held.
 Time (sec.): 11.233334

 : after putting down a block
   the block is not held

I understand that for every V1 ,
        after performing putdowns on V1 ,
        exactly 0 of the following
        are true: V1 is held.
 Time (sec.): 13.4

;;; Effects of acts......contd.

 : after putting down a block
   the block is clear

I understand that for every V1 ,
        after performing putdowns on V1 ,
        V1 is clear.
 Time (sec.): 10.433333

 : after putting down a block
   the block is ontable

I understand that for every V1 ,
        after performing putdowns on V1 ,
        V1 is ontable.
 Time (sec.): 10.683333

;;; Effects of acts......contd.

 : after stacking a block on another block
   the latter is not clear

I understand that for every V1 and V2 ,
        after performing stacks on V1 and V2 ,
        exactly 0 of the
        following are true: V2 is clear.
 Time (sec.): 28.916666

 : after stacking a block on another block
   the former is not held

I understand that for every V1 and V2 ,
        after performing stacks on V1 and V2 ,
        exactly 0 of the
        following are true: V1 is held.
 Time (sec.): 14.333333

;;; Effects of acts......contd.

 : after stacking a block on another block
   the former is on the latter

I understand that for every V1 and V2 ,
        after performing stacks on V1 and V2 ,
        V1 is on V2.
 Time (sec.): 12.583333

 : after stacking a block on another block
   the former is clear

I understand that for every V1 and V2 ,
        after performing stacks on V1 and V2 ,
        V1 is clear.
 Time (sec.): 12.116667

;;; Effects of acts......contd.

 : after unstacking a block from another block
   the former is not clear

I understand that for every V1 and V2 ,
        after performing unstacks on V1 and V2 ,
        exactly 0 of
        the following are true: V1 is clear.
```

```
 Time (sec.): 15.816667

 : after unstacking a block from another block
   the former is not on the latter

I understand that for every V1 and V2 ,
        after performing unstacks on V1 and V2 ,
        exactly 0 of
        the following are true: V1 is on V2.
 Time (sec.): 15.833333

;;; Effects of acts......contd.

 : after unstacking a block from another block
   the latter is clear

I understand that for every V1 and V2 ,
        after performing unstacks on V1 and V2 ,
        V2 is clear.
 Time (sec.): 12.033334

 : after unstacking a block from another block
   the former is held

I understand that for every V1 and V2 ,
        after performing unstacks on V1 and V2 ,
        V1 is held.
 Time (sec.): 12.6

;;; Some plans for a blocksworld...

 : if a block is on another block
   then a plan to achieve the former is held
   is to achieve the former is clear and
   then unstack the former from the latter

I understand that for every V1 and V2 ,
        if V1 is on V2
        then a plan to achieve V1 is held
        is by achieving V1 is clear and then
        performing unstacks on V1 and V2.
 Time (sec.): 27.8

;;; Some plans for a blocksworld......contd.

 : if a block is ontable
   and the block is clear then
   a plan to achieve the block is held
   is to pick up the block

I understand that for every V1 ,
        if V1 is clear
        and V1 is ontable
        then a plan to achieve V1 is held
        is by performing pickups on V1.
 Time (sec.): 22.7

;;; Some plans for a blocksworld......contd.

 : a plan to achieve a block is ontable
   is to achieve the block is held
   and then put down the block

I understand that for every V1 ,
        a plan to achieve V1 is ontable is
        by achieving V1 is held
        and then performing putdowns on V1.
 Time (sec.): 48.083332
```

```
;;; Some plans for a blocksworld......contd.

 : a plan to achieve
   a block is on another block is
   to achieve the latter is clear
   and then achieve the former is held
   and then stack the former on the latter

I understand that for every V1 and V2 ,
        a plan to achieve V1 is on V2 is
        by achieving V2 is clear
        and then achieving V1 is held
        and then performing stacks on V1 and V2.
 Time (sec.): 32.433334

;;; Some plans for a blocksworld......contd.

 : if a block is on another block
   then a plan to achieve the latter is clear
   is to achieve the former is clear
   and then achieve the former is ontable

I understand that for every V1 and V2 ,
        if V1 is on V2
        then a plan to achieve V2 is clear
        is by achieving V1 is clear
        and then achieving V1 is ontable.
 Time (sec.): 29.466667

 : ~end

 ATN Parser exits...
 CPU time : 383.42     GC time : 0.00
```

```
;;; Plan for building a stack of three blocks
;
;;; To build a stack of three blocks,
;;; B1 on B2 on B3,
;;; first put B3 on the table,
;;; then put B2 on B3,
;;; then put B1 on B2.

(assert forall (*block *other-block $third-block)
    act (build action make-3-stack
               object1 *third-block
               object2 *block
               object3 *other-block)
    plan (build
          action *SNSEQ
          object1
            (build
             action *ACHIEVE
             object1
               (build
                property *ONTABLE
                object *other-block))
          object2
            (build
             action *SNSEQ
             object1
               (build
                action *ACHIEVE
                object1 *ONE-ON-OTHER
             object2
               (build
                action *ACHIEVE
                object1 (build
                         rel *ON
                         arg1 *third-block
                         arg2 *block))))))
 CPU time : 14.58     GC time : 0.00
```

APPENDIX B2

```
;;; We now describe the current blocksworld
;;; and ask SNACTOR to perform some action.
;;;
(^ (parse -1))

 ATN parser initialization...
 Input sentences in normal English orthographic
convention. May go beyond a line by having
a space followed by a <CR>
To exit parser, write ^end.

 : blockc is clear

I understand that blockc is clear.
 Time (sec.): 5.9666667

 : blockc is ontable

I understand that blockc is ontable.
 Time (sec.): 4.516667

 : blockb is clear

I understand that blockb is clear.
 Time (sec.): 6.1

 : blockb is ontable

I understand that blockb is ontable.
 Time (sec.): 4.616667

 : blocka is clear

I understand that blocka is clear.
 Time (sec.): 6.25

 : blocka is ontable

I understand that blocka is ontable.
 Time (sec.): 4.7

 : pick up blockb

I understand that you want me to perform
        the action of pickups on blockb.
 Time (sec.): 6.35

Now  doing: PICKUP BLOCKB from table.

Now doing:  DISBELIEVE:
(M40 (OBJECT (M39 (LEX BLOCKB)))
     (PROPERTY (M12 (LEX CLEAR))))

Now doing:  DISBELIEVE:
(M41 (OBJECT (M39 (LEX BLOCKB)))
     (PROPERTY (M15 (LEX ONTABLE))))

Now doing: BELIEVE:
(M50! (OBJECT (M39 (LEX BLOCKB)))
      (PROPERTY (M14 (LEX HELD))))
 CPU time : 2.93     GC time : 0.00

 : put down blockb

Now doing: PUTDOWN BLOCKB on table.
```

```
Now doing: BELIEVE:
(M40! (OBJECT (M39 (LEX BLOCKB)))
      (PROPERTY (M12 (LEX CLEAR))))

Now doing: BELIEVE:
(M41! (OBJECT (M39 (LEX BLOCKB)))
  (PROPERTY (M15 (LEX ONTABLE))))

Now doing:  DISBELIEVE:
(M50 (OBJECT (M39 (LEX BLOCKB)))
     (PROPERTY (M14 (LEX HELD))))
 CPU time : 3.05     GC time : 0.00

 : pick up blockc

Now  doing: PICKUP BLOCKC from table.

Now doing:  DISBELIEVE:
(M37 (OBJECT (M36 (LEX BLOCKC)))
     (PROPERTY (M12 (LEX CLEAR))))

Now doing:  DISBELIEVE:
(M38 (OBJECT (M36 (LEX BLOCKC)))
     (PROPERTY (M15 (LEX ONTABLE))))

Now doing: BELIEVE:
(M68! (OBJECT (M36 (LEX BLOCKC)))
      (PROPERTY (M14 (LEX HELD))))
 CPU time : 2.98     GC time : 0.00

;;; SNACTOR Trace for the problem:
;;;----
;;; C|                         A
;;;----            ==>         B
;;;     A  B                   C
;;;  -----------          -----------
;
; Make a 3-stack using A, B, and C

(snact (build action make-3-stack
              object1 (build lex blocka)
              object2 (build lex blockb)
              object3 (build lex blockc)))

Want to ACHIEVE:
(M38 (OBJECT (M36 (LEX BLOCKC)))
     (PROPERTY (M15 (LEX ONTABLE))))

Want to ACHIEVE:
(M68! (OBJECT (M36 (LEX BLOCKC)))
      (PROPERTY (M14 (LEX HELD))))
Already Achieved.

Now doing: PUTDOWN BLOCKC on table.

Now doing: BELIEVE:
(M37! (OBJECT (M36 (LEX BLOCKC)))
      (PROPERTY (M12 (LEX CLEAR))))

Now doing: BELIEVE:
(M38! (OBJECT (M36 (LEX BLOCKC)))
      (PROPERTY (M15 (LEX ONTABLE))))

Now doing:  DISBELIEVE:
(M68 (OBJECT (M36 (LEX BLOCKC)))
     (PROPERTY (M14 (LEX HELD))))

Want to ACHIEVE:
(M75 (ARG1 (M39 (LEX BLOCKB)))
     (ARG2 (M36 (LEX BLOCKC)))
     (REL (M13 (LEX ON))))

Want to ACHIEVE:
(M37! (OBJECT (M36 (LEX BLOCKC)))
      (PROPERTY (M12 (LEX CLEAR))))
Already Achieved.

Want to ACHIEVE:
(M50 (OBJECT (M39 (LEX BLOCKB)))
     (PROPERTY (M14 (LEX HELD))))

Now  doing: PICKUP BLOCKB from table.

Now doing:  DISBELIEVE:
(M40 (OBJECT (M39 (LEX BLOCKB)))
     (PROPERTY (M12 (LEX CLEAR))))

Now doing:  DISBELIEVE:
(M41 (OBJECT (M39 (LEX BLOCKB)))
     (PROPERTY (M15 (LEX ONTABLE))))

Now doing: BELIEVE:
(M50! (OBJECT (M39 (LEX BLOCKB)))
      (PROPERTY (M14 (LEX HELD))))

Now doing: STACK BLOCKB on BLOCKC.

Now doing: BELIEVE:
(M40! (OBJECT (M39 (LEX BLOCKB)))
      (PROPERTY (M12 (LEX CLEAR))))

Now doing:  DISBELIEVE:
(M50 (OBJECT (M39 (LEX BLOCKB)))
     (PROPERTY (M14 (LEX HELD))))

Now doing:  DISBELIEVE:
(M37 (OBJECT (M36 (LEX BLOCKC)))
     (PROPERTY (M12 (LEX CLEAR))))

Now doing: BELIEVE:
(M75! (ARG1 (M39 (LEX BLOCKB)))
      (ARG2 (M36 (LEX BLOCKC)))
      (REL (M13 (LEX ON))))

Want to ACHIEVE:
(M77 (ARG1 (M42 (LEX BLOCKA)))
     (ARG2 (M39 (LEX BLOCKB)))
     (REL (M13 (LEX ON))))

Want to ACHIEVE:
(M40! (OBJECT (M39 (LEX BLOCKB)))
      (PROPERTY (M12 (LEX CLEAR))))
Already Achieved.

Want to ACHIEVE:
(M106 (OBJECT (M42 (LEX BLOCKA)))
      (PROPERTY (M14 (LEX HELD))))

Now  doing: PICKUP BLOCKA from table.

Now doing: BELIEVE:
(M106! (OBJECT (M42 (LEX BLOCKA)))
       (PROPERTY (M14 (LEX HELD))))

Now doing:  DISBELIEVE:
(M43 (OBJECT (M42 (LEX BLOCKA)))
```

```
     (PROPERTY (M12 (LEX CLEAR))))

Now doing:  DISBELIEVE:
(M44 (OBJECT (M42 (LEX BLOCKA)))
     (PROPERTY (M15 (LEX ONTABLE))))

Now doing: STACK BLOCKA on BLOCKB.

Now doing:  DISBELIEVE:
(M106 (OBJECT (M42 (LEX BLOCKA)))
      (PROPERTY (M14 (LEX HELD))))

Now doing: BELIEVE:
(M43! (OBJECT (M42 (LEX BLOCKA)))
      (PROPERTY (M12 (LEX CLEAR))))

Now doing:  DISBELIEVE:
(M40 (OBJECT (M39 (LEX BLOCKB)))
     (PROPERTY (M12 (LEX CLEAR))))

Now doing: BELIEVE:
(M77! (ARG1 (M42 (LEX BLOCKA)))
      (ARG2 (M39 (LEX BLOCKB)))
      (REL (M13 (LEX ON))))
 CPU time : 112.75     GC time : 0.00

;;; Achieve a state where Block-B is being held.
(snact
  (build action (build lex achieve)
         object1 (build
                  property (build lex held)
                  object (build lex blockb))))

Want to ACHIEVE:
(M43 (OBJECT (M38 (LEX BLOCKB)))
     (PROPERTY (M14 (LEX HELD))))

Want to ACHIEVE:
(M45 (OBJECT (M38 (LEX BLOCKB)))
     (PROPERTY (M12 (LEX CLEAR))))

Want to ACHIEVE:
(M37! (OBJECT (M36 (LEX BLOCKA)))
      (PROPERTY (M12 (LEX CLEAR))))
Already Achieved.

Want to ACHIEVE:
(M56 (OBJECT (M36 (LEX BLOCKA)))
     (PROPERTY (M15 (LEX ONTABLE))))

Want to ACHIEVE:
(M60 (OBJECT (M36 (LEX BLOCKA)))
     (PROPERTY (M14 (LEX HELD))))

Want to ACHIEVE:
(M37! (OBJECT (M36 (LEX BLOCKA)))
      (PROPERTY (M12 (LEX CLEAR))))
Already Achieved.

Now doing:  UNSTACK BLOCKA from BLOCKB.

Now doing: BELIEVE:
(M45! (OBJECT (M38 (LEX BLOCKB)))
      (PROPERTY (M12 (LEX CLEAR))))

Now doing: BELIEVE:
(M60! (OBJECT (M36 (LEX BLOCKA)))
      (PROPERTY (M14 (LEX HELD))))

Now doing:  DISBELIEVE:
(M39 (ARG1 (M36 (LEX BLOCKA)))
     (ARG2 (M38 (LEX BLOCKB)))
     (REL (M13 (LEX ON))))

Now doing:  DISBELIEVE:
(M37 (OBJECT (M36 (LEX BLOCKA)))
     (PROPERTY (M12 (LEX CLEAR))))

Now doing: PUTDOWN BLOCKA on table.

Now doing: BELIEVE:
(M37! (OBJECT (M36 (LEX BLOCKA)))
      (PROPERTY (M12 (LEX CLEAR))))

Now doing: BELIEVE:
(M56! (OBJECT (M36 (LEX BLOCKA)))
      (PROPERTY (M15 (LEX ONTABLE))))

Now doing:  DISBELIEVE:
(M60 (OBJECT (M36 (LEX BLOCKA)))
     (PROPERTY (M14 (LEX HELD))))

Now doing:  UNSTACK BLOCKB from BLOCKC.

Now doing: BELIEVE:
(M43! (OBJECT (M38 (LEX BLOCKB)))
      (PROPERTY (M14 (LEX HELD))))

Now doing:  DISBELIEVE:
(M45 (OBJECT (M38 (LEX BLOCKB)))
     (PROPERTY (M12 (LEX CLEAR))))

Now doing:  DISBELIEVE:
(M41 (ARG1 (M38 (LEX BLOCKB)))
     (ARG2 (M40 (LEX BLOCKC)))
     (REL (M13 (LEX ON))))

Now doing: BELIEVE:
(M91! (OBJECT (M40 (LEX BLOCKC)))
      (PROPERTY (M12 (LEX CLEAR))))
 CPU time : 96.13     GC time : 0.00
```