Notes on Using SNePS for Interactive Fiction

Phil Goetz
Dept. of Computer Science, SUNY, Buffalo NY 14260

December 15, 1993

1 SNePS (skip this if it bores you)

SNePS, a **S**emantic **Ne**twork **P**rocessing **S**ystem developed primarily at the University of Wisconsin, Indiana University, and the University of Buffalo over the past 25 years, is a knowledge representation (KR) and reasoning system. It uses semantic networks to represent knowledge.

1.1 Semantic networks

A semantic network is usually a directed graph in which the directed edges, or **arcs**, specify relationships between the entities represented by the vertices, or **nodes**. In SNePS, a proposition is a closed (no unquantified free variables) predicate logic formula, and is represented in the network by a node. The set of arcs leading from the node forms a caseframe, which identifies the logical predicate in the proposition.

A semantic network representation displays a system of knowledge in a form easier to understand than a list of propositions, because each object of a proposition is only written once, and all the rules concerning a given object can be found merely by noting what is connected to that object. The distinction between semantic networks and logic is merely one of notation.

1.2 Philosophical commitments of SNePS

1.2.1 Propositions are represented by nodes

Every proposition that can be asserted by the system must be subject to modification. Most semantic network representation for "Jack is a fish" would consist essentially of a node representing Jack, a node representing the class of fish, and an arc labelled "isa" going from $Jack$ to $fish$. But how do you say "Jack is not a fish"? The SNePS solution is to represent "Jack is a fish" by the network consisting of a node $Jack$ representing Jack, a node $fish$ representing the class of fish, and a node M representing the proposition that Jack is a fish. An arc labelled "member" leads from M to $Jack$, and one labelled "class" leads from M to $fish$.

In SNePS we can represent any proposition by a list of pairs, where each pair is an arc label leading from the node representing the proposition, followed by the representation for the proposition or base node (something like $Jack$ or $fish$) or other type of node that the arc leads to. There is no need to label a propositional node in specifying a network. So, the proposition "Jack is a fish" can be represented by a node whose structure is described as

(member *Jack* class $fish$)

Now you can negate the proposition by asserting

(not (member *Jack* class $fish$))

1.2.2 Intensional representation

An unusual feature of SNePS is its commitment to intensional representations [Maida and Shapiro 1982]. Roughly, an extensional object is one in the real world. An intensional object is an object of thought. The **SNe**PS **R**esearch **G**roup (**SNeRG**) argues that knowledge representations should be intensional for several reasons. One is that unicorns and other imaginary objects have no extensions, yet we think and talk about them. Another is that you may have different beliefs about objects of thought that actually, unbeknownst to you, are the same extensional object. For example, if a man has just won the lottery but doesn't know it yet, and you ask him "Are you rich?", he will say no, but if you ask him "Is the man who just won the $10 million lottery rich?", he will say yes.

1.2.3 Uniqueness principle

Every concept represented in the network is represented by one and only one node [Shapiro 1986].

1.2.4 UVBR

The **Unique Variable Binding Rule**, or **UVBR**, states that no instance of a rule is allowed in which two distinct variables of the rule are replaced by the same term, nor in which any variable is replaced by a term already occurring in the rule. The reason is that different variables used in a single rule are meant to represent distinct concepts. Therefore, it would be an abuse of the intended meaning of the rule to instantiate it so that two distinct concepts were collapsed into one [Shapiro 1986]. If Jack, a single extensional object, is thought of as filling both the agent and object roles in a rule, then he is represented by two different intensions: one representing Jack the agent, and one representing Jack the object.

2 The Goals of Using SNePS for Interactive Fiction

2.1 Interactive fiction as it could be

Here is a possible interactive fiction version of *Of Mice and Men,* mangled for my own didactic purposes:

>enter barn

The barn is quiet now. Lenny is standing near the doorway, crying. "I didn't mean to do it, George," he sobs. The straw has been thrown up all around him. Something like a large sack lies on the floor behind Lenny, but you can't tell what just yet.

This is canned text, since this scene is central to the plot.

>Ask Lenny "What did you do?"

Lenny stops crying.
Lenny says, "George, nothing."

Assume that "What did you do?" is parsed so that Lenny comes up with a proposition like (agent Lenny action kill object Sarah) as the true answer. Lenny creates a simulated belief space of George, adds that proposition to it, and sees that that knowledge may cause George to hit him. This contradicts with his own preservation goals. Lenny decides to return no proposition: "George, nothing." The act of speaking has as a precondition that the speaker must not be crying too hard to speak, so Lenny first stops crying. This action is reported.

>go behind Lenny

Lenny stops you.

Assume that the spatial representation of the barn is such that "go behind Lenny" makes sense. It may be best to allow each nearby character to examine each request for action by the participant before it is executed, in case that character wants to interfere. In this case, Lenny simulates a world like the present one except in which George is behind him. Lenny doesn't have to consciously realize that George will then see the body; he simply runs the simulation: the proposition (object Sarah property dead) will be added to the simulated George, setting off a goal of finding a plausible cause. Assuming (agent Lenny action kill object Sarah) can be hypothesized, this hypothesis might cause the simulated George to hit Lenny, or shout at Lenny. Lenny merely sees the resulting action in the simulated George, and decides that he will prevent that.

Note the the participant (George) is simulated by pretending he is a non-player characters like Lenny.

>push Lenny

Lenny staggers back.
Behind Lenny you see dead Sarah.
Lenny says, "George, don't yell at me."

Lenny sees you see Sarah, and derives that you may yell at him. He tries to prevent this by a request.

>hit Lenny

Lenny runs out of the barn.

Lenny is a reactive agent. When something hits him, he runs away.

2.2 Interactive fiction as it is

>enter barn

The barn is quiet now. Lenny is standing near the doorway, crying. "I didn't mean to do it, George," he sobs. The straw has been thrown up all around him. Something like a large sack lies on the floor behind Lenny, but you can't tell what just yet.

>Ask Lenny "What did you do?"

Lenny isn't interested in talking.

Lenny can't talk.

>go behind Lenny

You are now behind Lenny.
You see Lenny and dead Sarah.

Lenny doesn't care what you do.

>hit Lenny

That wouldn't be nice.
You are in a barn.
You see Lenny and dead Sarah.

"That wouldn't be nice" means "I can't let you do that because it would make me look silly." Lenny wouldn't react. Nothing interesting happens in a world with static characters unless you do it yourself.

3 SNePS and the Knowledge Representation Needs of Interactive Fiction

3.1 Interactive fiction programming methodologies

3.1.1 Verb handlers

The earlier interactive fiction (IF) implementations, such as ADVENT, used verb handlers. That is, a separate subroutine was written for every verb in the game. Each subroutine was responsible for checking for any unusual reactions to special objects or circumstances.

3.1.2 Object-oriented programming

The first adventure to use object-oriented programming was DUNGEON, which collected special-case code around objects instead of in verb handlers. Recently, many programmers, using systems like TADS, have come to appreciate that object-oriented simulated worlds are easier to maintain and debug, since items can be removed and added more easily. Also, objects can send messages to each other. Instead of needing a central control program to handle interactions between objects, each object can have its own methods of dealing with different signals. So, for example, if the signal is speech, most objects ignore it, non-player characters (NPCs) respond to it, and portals (e.g., doors) may pass a "muffled speech" signal to the room they connect to, upon which the room passes the message to every object in it.

3.1.3 Rule-based systems

There is an old argument in KR over whether procedural or declarative knowledge is better [Winograd 1975]. In the eighties, the AI community in general moved toward declarative representations [Brachman 1990]. I favor declarative knowledge for IF.

Non-player characters should be able to reason about the IF world and formulate plans on the fly to accomplish their goals. This requires understanding the world and being able to predict the results of their actions. The code that specifies what effects an action will have must be available to that character

for introspection at the knowledge level [Allen 1981], meaning the most abstract level of representation at which precise results can be predicted, or roughly the level at which humans symbolize problems. That is, though a character doesn't need access to the compiler, she needs access to rules like "If X is in Y and drops Z, then Z is in Y."

The source code of a procedural language is very difficult to reason with. To use traditional AI reasoning techniques, which use logics, the procedures which specify how to update the world in response to actions should be *rules* in a rule-based system like Prolog.

SNePS has all the advantages of rule-based systems, as well as a procedural attachment mechanism. This lets you define predicates with Lisp code, not just in terms of other rules. There is a way to interface rules with Lisp code, so that you can call a Lisp function to answer a question in such a way that a reasoner need not understand the Lisp code.

Rule-based systems can be written in an object-oriented manner. The rules can be entered in groups organized around objects. Message handlers can be expressed with rules. Very general inheritance hierarchies are easy to make using inheritance rules.

3.2 Belief spaces

3.2.1 Viewpoint persona

I must digress to explain what the "viewpoint persona" is. The participant interacts with the IF world through the point of view of one character. Adventures usually address the player in second person, as if the player were identified with that character. A more powerful technique is to address the player in first person, postulating a viewpoint "puppet" which the player controls. This lets the player ask questions which the program can answer ("Where is Belboz?"), lets the program offer advice ("Maybe we should let the nice dragon sleep,") and lets the puppet respond to commands with phrases such as "No way I'm going into a pool of molten lava!" I refer to this puppet as the *viewpoint persona.*

The use of a viewpoint persona is more straightforward than second-person address. A viewpoint persona obviously requires a cognitive model of that persona, whereas supposedly second-person address lets the participant be his own cognitive model. That is not true. If the participant is in a library with a door to a hallway and an undiscovered secret door behind a bookcase, and types "Open door," you don't want the program to ask, "Which door do you mean, the library door or the secret door?" It must keep track of what the participant knows. Second-person address is also odd in that the commands given by the participant are also in second person. Typing "[You] eat the plant" and being told "You don't like its taste" raises the eyebrows of people unacclimated to this convention: who ate the plant, the viewpoint persona or the participant?

3.2.2 Different beliefs

All interactive fiction architectures written to date have only one representation of the world, which is interpreted as complete and correct. All characters, including the viewpoint persona, refer to this knowledge base (KB). This is inadequate. Not all the characters in the IF world should have the same beliefs. Some have knowledge that others don't. None, including the viewpoint persona, should have complete knowledge. It is quite jarring, in text adventures, to ask "Where is the green jar?" and be told "Zane has it" if your persona hasn't yet met Zane.

The minimal architecture needed would have a KB representing the IF world, and separate cognitive architectures for each character and for the participant's persona. Each cognitive architecture would consist

of at least a KB mostly duplicating the world KB (hence an efficient method of referring to instead of duplicating items in the world KB or in other agent's minds is needed), plus any special information that character has; and a planning/acting component which moves perceived information from the world KB into its private KB, and responds to changes in its private KB with purposive action. The user interface would simply be an English parser which places goals on a goal stack of the viewpoint persona. Having an agent architecture for the viewpoint persona that can make and execute plans also lets the participant give orders such as "Enter the kitchen" and have the persona handle details such as unlocking and opening the door. Once we have such a planning and acting viewpoint persona, we need it to have a private KB, because we don't want the participant to type "Save Marty from the flood" and have the persona come up with the entire sequence of moves which the participant was expected to discover.

In SNePS, this can be accomplished by giving the viewpoint persona and all NPCs a separate belief space from the knowledge in the world model. Every perceptual action of any agent transfers information found into that agent's belief space. When the participant enters a room, he implicitly performs a "look around." If it is dark, nothing is reported. If he can see, every piece of information that is reported is also entered into the participant's belief space, with a rule of the form

```
(assert forall $dest $agent $agent-context
   &amp;ant ((build member *dest class thing)
         (build object *agent context *agent-context))
   cq (build
      act (build action (build lex "look") agent *agent dest *dest)
      plan (build
         action (build lex "snsequence")
         object1 (build action (build lex "print")
                        object (build lex "You see "))
         object2 (build action (build lex "withall")
                        vars $obj
                        suchthat (build object *obj in *dest)
                        do (build action (build lex "snsequence")
                                  object1 (build action (build lex "print")
                                                object *obj)
                                  object2 (build action (build lex "believe")
                                                object1 (build
                                                      object *obj
                                                      in *dest
                                                      *agent-context)))))))
```

where the final "(assert object *obj in *dest agent-context)" says that the knowledge that $*obj$ is in $*dest$ is to be added to the agent's belief space. I will use this same code for "look in the barrel," "look through the window," etc., so all knowledge and only that knowledge reported to the participant is available to the viewpoint persona. With the use of object-oriented methods, this can even report false knowledge to the viewpoint persona. For instance, if Greg is disguised as an old woman, the "look" code would ask the object *Greg* to report its identity, which would be given as "old woman."

3.2.3 Hypothetical reasoning

SNePS attaches to each proposition in its knowledge base a set of hypothesis. If that set is nil, the proposition is always true. If a character wonders what will happen if she performs an act, a new context

will be created which inherits all the knowledge of the character, and adds the hypothesis that the character performed that act. If she had complete knowledge of the world, reasoning in that new space would produce the same results that performing the action would in the IF world, but the IF world is not affected since the derived results have an additional hypothesis in their hypothesis set. If the character doesn't have complete and correct knowledge, or if the world changes after her planning, she may reach a conclusion which will prove false when she executes her plan.

3.3 Beliefs and actions

3.3.1 Integrating acting and inference

NPCs need to be able to plan and act on plans to do anything interesting. Decomposable plans, which consist of subplans, allow characters to have reactive plans, meaning they can use different subplans depending on the environment. This is necessary in order for them to overcome attempts by the participant to foil their actions.

The participant's persona can also benefit from planning capability. The planning system would take care of details like automatically picking up a book you ask to read, or opening a box you want to look inside.

I favor a rule-based system so characters can have maximal knowledge available to them for reasoning. For the same reason, plans and beliefs should be integrated. Traditional planners use three different representations — one for the world model, one for operators or actions, and one for plans. As a result, the system has to perform three types of reasoning: reasoning about the world, about actions, and about plans. The result is that the inference engines for reasoning about actions and plans get shortchanged, and don't have the power of general inference. In the **SN**ePS **A**cting and Inference **P**ackage (**SNAP**), plans and actions, though distinct from beliefs and having different denotations, are represented in the same language; hence, the same inference rules can operate on them [Kumar and Shapiro 1991a].

So, using SNAP, you can not only execute plans, you can reason about them, e.g., to decide which plan is best or to guess when another character is executing a plan. Since you can reason about plans, and plans are decomposable, a character can take a collection of facts and derive a plan which the IF designer never foresaw.

In most architectures, reasoning is performed by an inference engine and acting is done by an acting executive. The SNAP approach uses only one component. Belief is viewed as an action, and other types of actions are supposed to be executed in a similar manner to believing by an external acting component [Kumar and Shapiro 1991a]. SNAP integrates beliefs and action with the use of **transformers**. Belief-belief transformers are ordinary inference rules. Belief-act transformers under forward chaining have either the interpretation "once the agent believes B, it intends to do A", e.g., "if the building is on fire, exit," or "if the agent wants to know the truth value of P, perform A". Under backward chaining, they specify preconditions: "If you want to play soccer, you need to have a ball." The transformers for forward and backward chaining are distinct. (You wouldn't want your character to set fire to his house because he wanted to leave it.) Action-belief transformers represent effects of actions and plan decompositions. Action-action transformers represent sequences of actions. Together, these transformers allow a character's beliefs to affect his actions in ways that take into account all the rules governing the IF world, and they let actions be governed by goals.

In this way SNAP is like Meehan's TALESPIN. Both have distinct representations for plans and beliefs, and a means of executing plans, although SNAP uses a STRIPS planner and TALESPIN does not [Kumar and Shapiro 1991b p. 129, Meehan 1980 p. 52-53]. Both have a class of inferences that connect beliefs to the acts from which the beliefs result, inferences that connect acts to the beliefs that enable

them, inferences that connect acts to reasons why those acts might be done, and so on [Meehan 1980 p. 42]. But TALESPIN characters cannot reason about their plans, since rules of inference do not operate on plan components.

The main motivation for using this planning system in IF is so that the non-player characters can make and execute plans, detect the player's plans, and form plans to assist or thwart the player. What is needed is a more selective method of detecting plans that another agent might be following (currently SNAP returns all that are consistent with the actions taken — which is possibly infinite), and a way for an agent to consider what consequences the success of another agent will have for his own goals and to choose to help or hinder the other agent. It's not clear whether this can be done with SNAP as it exists, merely by adding rules.

3.3.2 Belief revision

A truth-maintenance system (TMS) is neeeded to maintain consistency in a rule-based world representation. Inferences will be drawn from facts; when those facts change, those inferences must be withdrawn. For example, I may have a rule that states that the effective weight of an object is its intrinsic weight plus the weight of all objects on or in it. I might add an effect to the action (put object *obj dest *dest) which says to add the weight of $*obj$ to that of $*dest$, and to subtract the weight of $*obj$ from that of its previous container. But that side-effect approach wouldn't be consistent. If A were put in B, B were put in C, and A were removed from B, A's weight would at first be added to but later not subtracted from C's. A procedural world simulation would make recursive calls to maintain consistency. To implement that recursion in a rule-based approach, we need truth maintenance. C's weight depends on B's weight. When B's weight changes, the assertion about C's weight will be removed. When we once more want to know C's weight, it will be re-derived. **SNeBR**, the **SNe**PS **B**elief **R**evision System, does this.

This same action takes effect on higher levels. Say your NPC Annie has decided to throw herself off the cliff based on her belief that Jethro doesn't love her anymore, and Jethro comes and tells her he loves her. Without a belief revision system, Annie will say "Good" and throw herself off the cliff. Her plan to do so was originally derived from a belief B, but removing B doesn't remove the inferences and plans based on it. With a belief revision system, all the inferences and plans based on B will also be removed when B is removed, so Annie won't throw herself off the cliff.

Actually, this example won't work yet, because SNeBR can't decide which belief to remove when two beliefs clash. I hope to add a few heuristics to the belief revision system so it can replace old information with new information. Now it will stop and ask the user which to believe.

3.4 Disadvantages of SNePS

I've been trying to implement an IF engine in SNePS 2.1. I've come across several problems:

1. SNePS cannot easily implement a default logic. Many times I want to say, "Do B unless C". For example, the following rules say, "If character C asks to move west, and $by(C, X)$, and Y is west of X, retract $by(C, X)$ and assert $by(C, Y)$. If that rule is not applicable, but $in(C, L)$ and M is west of L, retract $in(C, L)$ and assert $in(C, M)$."

   1. west(C) & by(C,X) & westof(X,Y) ⇒ not(by(C,X)) & by(C,Y)
   2. Default: west(C) & in(C,L) & westof(L,M) ⇒ not(in(C,L)) & perform(in(C,M))

There is no mechanism for default logic in SNePS 2.1. There is an older version of SNePS, SNePSwD, with default rules, but it exploits bugs that have been corrected in SNePS 2.1, so porting it to SNePS 2.1 would not be easy.

SNePS is **monotonic**: Any derived proposition will still be true when other non-contradictory information is added to the knowledge base. I can't say "If A then by default B, unless C." If A then B, period. C doesn't have anything to say about it.

When we have a finite number of rules, we should be able to implement a default logic if we **assume closure**: Assume that if we can not derive P, $not(P)$ holds. Then we can provide disjoint antecedents for each of the rules. In the example above, the default rule would have the added antecedent *not (exists (X Y) (by(C,X) & westof(X,Y)))*.

I can't ask if *exists (X Y) P(X,Y)* because the existential quantifier hasn't been implemented yet in SNePS, and probably won't be for at least two years. The workaround is to use Skolem functions. But they can only be used to derive the existence of an object, not to derive the fact that no such object exists. So I still can't ask if *not (exists (X Y) . . .)*.

There is an alternative form of *not(exists(X) P(X))* : *forall (X) not(P(X))*. But I can't use such a rule in SNePS, because SNePS doesn't assume closure. Just because it can't prove P doesn't mean it will assume $not(P)$. So I generally can't effectively query whether $not(P(X))$ holds. You can prove $not(in(C, M))$ if you know $in(C, L)$ and $in(C, X) \Rightarrow not(in(C, Y))$ (this latter rule is correct due to UVBR). But you can't prove $not(by(X, Y))$ because there is no necessity for X to be *by* anything.

There is a need for a form of **autoepistemic logic**. An autoepistemic logic assumes closure in an agent's mind for propositions about that agent, since the agent should know things about himself. What we need is a logic that would assume closure for propositions which the believing agent could verify empirically. In this case, the character can verify that he is not by any object by looking around himself.

Although SNePS can say *forall (x) [P(x) → Q]*, SNePS apparently can't say *[forall (x) P(x)] → Q*. So even if I could query whether $not(by(C, X))$ held, I couldn't express the correct meaning of the rule *[forall (X) not(by(C,X))] → P*.

I can't assert two separate rules for the two cases. There is no guarantee what order rules will be executed in, and in the case above, executing the wrong (default) rule (1) first destroys the preconditions for the correct rule (2). (This is a side-effect of the "perform" predicate, which invokes a procedure.)

I can't combine the two rules into one, because of the way variables must be mentioned in a SNePS rule. All variables in SNePS rules (that aren't Skolem functions) must be universally quantified. You must place restrictions on them in the rule antecedent, or else the variables will be matched to every node in the database, which makes rule use very slow. (Also, the rule is not guaranteed to work if the variable is not restricted.) But this means that, in order for a rule to fire, bindings for all free variables must be found. The two rules have different free variables. If we combined them into one, it could not fire unless bindings for *all* the variables were found, even though they may be irrelevant. I can work around this by saying everything is a *thing*, and using $thing(X)$ in the antecedent. However, this makes inference very slow, since every *thing* will be matched to X.

There is a way to use a limited default logic. I can use SNAP to say, "When you want to know if $not(P)$ is true, do action A." Action A is Lisp code which tries to derive $not(P)$ assuming closure. This is similar to the way Cyc implements default logic [Guha 1990]. But there are limitations on when this can be used. First, at present a different action must be written for each type of node P, since you have to know some arcs coming into P. Second, the problems concerning parenthesizing quantifications and implications still hold.

The tremendous difficulties that the combination of monotonic logic and lack of closure create make

me think that knowledge representation researchers would not have wasted so many decades on such logics if they had tried to use them for IF.

2. Like all logics, SNePS is poor at performing procedures on collections. You can say "For all objects X in object Y perform Z." But you can't say "Add the weight of all objects X in object Y." Adding iteratively to a variable is impossible since a variable in logic has one value, and is not free to change with every iteration of a rule firing. You can't say "If W=N then retract W=N and assert W=N+X" because retracting the antecedent of the rule will cause belief revision to retract its consequent. You can write a procedural attachment to compute the sum of the weights of all objects in object Y, but if you then assert that the effective weight of Y is its intrinsic weight plus that sum, that assertion depends on the effective weights of all the objects in Y. Since you used a procedure to compute that sum, all the assertions you used about the weights of various objects will not be added to the support set of the derived proposition about the effective weight of Y. This means that this latter proposition will not be retracted if an object is removed from Y, or if the effective weight of an object in Y changes. Sung-Hye Cho is working on an extension of SNePS to handle reasoning about collections.

3. Preconditions and effects are given for actions. Currently I represent "John hit the ball with the bat" by (act (action hit agent John object ball instr bat)). This is because many actions either have preconditions concerning the agent, or affect the agent, so we need to talk about the agent in the action. But an action is something that can be done by many people, and we want to be able to say Jean did the same act as John if every slot except *agent* has the same filler. So perhaps we should have (agent John act (action hit object ball instr bat)) and have a means of giving preconditions and effects for a different type of action represented by that entire proposition.

4. John can't be bound to both agent and object due to UVBR. This means that we need separate rules for the cases "John hit Sue with the paddle" and "John hit himself with the paddle." It is significant that our language often uses different words for reflexive actions; possibly they do have different meanings to us. But writing multiple rules for every action that can be performed with oneself as one of the non-agent slot-fillers is inefficient in terms of memory and makes the knowledge base hard to maintain. On the plus side, it automatically prevents actions such as "Hit the hammer with the hammer."

5. SNeBR currently needs user intervention to resolve conflicts. An NPC would not be able to retract old information if it conflicted with new information. This would not be a problem if we used only (infallible) deduction. But then a character could never make use of any information you told her, because the truth value of a proposition P cannot be deduced from the proposition *told(John, me, P)*. SNePSwD can resolve conflicts without user intervention.

6. If an agent tries to perform an action, and reaches a precondition that has no plan specifying how to satisfy it, SNAP goes into an endless loop. The problem is that there will always be such a final precondition unless the action ultimately succeeds. This is not likely to be fixed soon since Deepak Kumar, the author of SNAP, claims it is a feature.

7. It is hard to program in logic. A 3-line procedure can become a 30-line rule.

8. When SNePS crashes, it in some cases crashes at the LISP level instead of at the SNePS level. Debugging is then impossible unless you know how SNePS works, which you don't want to know.

9. SNePS requires Common Lisp. Public domain versions (such as Kyoto Common Lisp) are available in the public domain, but any version probably needs 10M RAM. A Scott-Adams-sized IF world would need perhaps an additional 10M (based on extrapolation from what I've implemented). I anticipate that within 3 years it will be common for home systems to have this much memory.

10. With the very simple rulebase I have created, and no NPCs, the prompt-to-prompt time (between when you enter a simple command and when you get the next prompt) is about ten minutes at first, though it goes down to around three minutes after common inferences have been made once. (SNePS remembers the results of past inferences, even when they result in rules rather than definite propositions [Choi and Shapiro 1991].) With a minimal world simulation that dealt with sizes, capacities, stacks, and weights, this time might increase by a factor of less than ten. A fifteen-minute wait (longer if you don't have a 10 MIPS Sparcstation) is not acceptable. If a rule is coded inefficiently, its firing time increases exponentially with the number of objects in the database.

    On the plus side, SNePS parallelizes naturally, since variable bindings for rules are meant to be tested in parallel. A near-future desktop computer with 32 processors each equivalent to a DEC Alpha might run it comfortably.

4 Conclusion

There are many reasons why SNePS in its current incarnation is not ready to be used as an IF engine. But I believe that systems like SNePS will supersede traditional programming languages in the near future for interactive fiction. All of the knowledge-representation problems in SNePS that I have listed (except for logic being difficult to program in) are slated to be corrected within the next 3 or 4 years, so perhaps someday soon we will see SNePS IF.

5 References

Ronald J. Brachman (1990). "The future of knowledge representation," *Proceedings Eighth National Conference on Artificial Intelligence* (AAAI-90), Cambridge, Mass.: MIT Press, p. 1082-1092.

Joongmin Choi and Stuart C. Shapiro (1991). "Experience-based deductive learning," *Proc. of the 1991 IEEE Int. Conf. on Tools for AI,* San Jose, CA.

Ramanathan V. Guha (1990). "The representation of defaults in Cyc," *Proceedings Eighth National Conference on Artificial Intelligence* (AAAI-90), Cambridge, Mass.: MIT Press, p. 608-614.

Deepak Kumar and Stuart C. Shapiro (1991a). "Architecture of an intelligent agent in SNePS," *SIGART Bulletin,* vol. 2 no. 4, Aug. 1991, p. 89-92.

Deepak Kumar and Stuart C. Shapiro (1991b). "Modeling a rational cognitive agent in SNePS," *Proceedings of EPIA 91, the 5th Portugese Conference on Artificial Intelligence,* eds. P. Barahona and L. Moniz Pereira, Heidelberg: Springer-Verlag, p. 120-134.

Anthony S. Maida and Stuart C. Shapiro (1982). "Intensional concepts in propositional semantic networks," *Cognitive Science,* vol. 6, Oct.-Dec. 1982, p. 291-330. Reprinted in *Readings in Knowledge Representation,* eds. R.J. Brachman and H.J. Levesque, Los Altos, CA: Morgan Kaufmann, 1985, p. 169-189.

James R. Meehan (1980). *The Metanovel: Writing Stories by Computer.* NY: Garland.

Allen Newell (1981). "The knowledge level," *AI Magazine,* vol. 2 no. 2, p. 1-20.

Stuart C. Shapiro (1986). "Symmetric relations, intensional individuals, and variable binding," *Proceedings of the IEEE,* vol. 74 no. 10, p. 1354-1363, Oct. 1986.

Terry Winograd (1975). "Frame representations and the declarative/procedural controversy,"

Representation and Understanding: Studies in Cognitive Science, eds. D.G. Bobrow and A.M. Collins, NY: Academic Press, 1975, p. 185-210. Reprinted in *Readings in Knowledge Representation,* eds. R.J. Brachman and H.J. Levesque, Los Altos, CA: Morgan Kaufmann, 1985, p. 357-370.