Accounting for Script Generation: A Model with Connectionist and Symbolic Components

by

Suzanne M. Mannes and Stephanie M. Doane

Institute of Cognitive Science

University of Colorado
Campus Box 345
Boulder, CO 80309-0345

ICS Technical Report #89-8

Accounting for Script Generation: A Model with Connectionist and Symbolic Components

Suzanne M. Mannes and Stephanie M. Doane[1]

Institute of Cognitive Science
University of Colorado

[1]The research reported in this paper was supported by grant number ARIMDA903-86-C0143 from the Army Research Institute and grant number IRI-8722792 from the National Science Foundation. We gratefully acknowledge the insightful comments of Mike Mozer on an earlier draft of this paper.

Abstract

For many years researchers have attempted to model human behavior in familiar situations using the notions of schemata and scriptre presentations. Much psychological data exists to support the notion that humans use script or schema-like representations in their interactions with the world (e.g., Bower, Black, and Turner, 1979). Purely symbolic models and purely connectionist models each have difficulty accounting for both the contents and sequentiality of scripts. This paper will review several of the approaches that have been proposed to account for human behavior regarding scripts and schemata. Following this, a hybrid model is proposed which allows us to model scriptal behavior. It is then shown how the model deals effectively with traditional and non-traditional AI problems which lend themselves to scriptal representations.

A major goal of cognitive science is to provide descriptions of well established human cognitive phenomena. To accomplish this effectively, it is essential that the behaviors chosen to be modeled are well known and supported by a large quantity of experimental data. This assures that cognitive scientists are modeling relevant behaviors and allows significant comparisons between the modeling process and those observed in human subjects.

We describe some experimental work that supports the notion of scripts as psychologically real. We then discuss traditional symbolic and recent connectionist methods for modeling these scripts. It is argued that purely symbolic and connectionist methods each cannot account for all of the data, and a hybrid model with symbolic and connectionist components is described. We use this model to simulate data from subjects describing how they perform routine computing tasks, and subjects producing complex UNIX[2] commands. Benefits of the hybrid formalism will then be presented.

One of the most interesting and consistent findings in the psychological literature concerns the psychological validity of scripts and schemata as representations in memory (e.g., Bower, Black, & Turner, 1979). Traditional approaches to modeling and accounting for schemata have suggested large stable structures (as in the work of Minsky, 1975 and Schank & Abelson, 1977). These have been successful with regard to schemata which change little from experience to experience, but not with unusual or novel instances of schemata. More recent attempts at modeling schemata have quite different characteristics. They tend to approach the concepts of schemata in a bottom up manner: Composing schemata from a set of primitive features (as in the room schemata work of Rumelhart, Smolensky, McClelland, and Hinton, 1986). These methods have been successful in modeling static schemata, but have weaknesses when it comes to modeling schemata of a temporal or causal nature. In light of the particular strengths and shortcomings of these past attempts, the hybrid model we propose appears to be a suitable alternative to either of these.

---

[2]UNIX is a registered trademark of AT&T.

Scripts and Schemata as a Domain

Schematic representations or schemata have typically been viewed as the body of knowledge a person or culture has about a particular event or state of the world; e.g., what a kitchen looks like or what one would typically find at a zoo. Scripts can be seen as a subcategory of schemata - causal schemata - in which the information contained in the schemata is ordered sequentially. Knowledge of what happens when we go to a restaurant for a meal or to the grocery store to buy groceries are examples of the events considered to be scriptal.

The original work of Bartlett (1932) showed how powerful schemata can be in affecting comprehension and retention of stories and pictures. In his experiments, Bartlett often found that topics or ideas which were out of place in the context of the more general topic being learned, were often not remembered at the time of recall. Out of context knowledge which was not forgotten was often changed to be more in accordance with the topic being learned, and contextually appropriate items which were not studied often intruded in subject's recollections. The converging evidence from many studies led Bartlett to postulate the existence of a schema and researchers have been trying to determine the nature of this concept ever since.

Experimental Evidence

Experimental evidence regarding scripts and their role in memory has been provided by a series of experiments by Bower, Black, and Turner (1979). In one experiment, they asked students to tell what typically happens in the sorts of situations presented above, like going to a restaurant for a meal. There was high agreement among the participants' protocols, and the items they produced were almost exclusively produced in a temporal order (i.e. according to the way they would happen in the real world). In a recall experiment, when the script items were presented in a random order, they were recalled by subjects in the appropriate temporal order.

Thus, evidence of Bower et al. supports the existence of scripts, but it does not suggest anything about the structural form of scripts in memory. This was addressed in further studies where Bower et al. found evidence that these scripts do not exist in memory as stable structures. They were able to determine this by assuming the

following: if scripts existed in memory as stable structures, then when subjects were asked to comprehend sentences reflecting events from a script, the further away from event A that event B occurs in time, the longer it should take subjects to comprehend the sentence portraying event B after reading about A. This was not the case. They failed to find this "graded distance" effect. Hence, on the one hand, scripts seem to exist; there is much agreement between subjects on what constitute scriptal activities and subject recall is quite faithful to the ordering that these events follow in real time. On the other hand, evidence suggests that these are not structures which merely reside in memory in a stable, pre-existing state. Everyday experience suggests that exceptions to what we would expect from a stable script are interpreted by people with little difficulty and changes in their environments are adopted quite readily. That is, given a situation in which certain events occur out of their expected temporal sequence, people do not fail to act in the situation but, rather, use their cognitive flexibility to interpret the observed "violations" with respect to their expectations. For example, going to a nice restaurant where you must pay for food before it is served, thus violating expectations, does not prohibit people from eating there. Rather, the experience serves to brings up information relevant to other scripts, for instance a fast food restaurant.

Behaviors which are consistent with scripts as a part of memory are easily confused with the true existence of these structures. In light of this it is not surprising that symbolic approaches to modeling human script-like behavior has tended to represent these scripts in stable structures such as frames, assuming that they exist as stable structures in human memory. This methodology assures that prototypical, script relevant, items can be produced in a correct order. However, it seems to follow from closer examination of the human data that scripts are not represented in human memory in this manner, although they follow fairly strict temporal production, because exceptions are dealt with easily, and are integrated with remaining script material. As an example, consider restaurant patrons who have been seated and have ordered their meals, and then have been asked to move to another table for some reason, i.e. they are reseated. This situation presents no problem (other than inconvenience) for the patrons: They do not try to order their meals again! Thus, an exception to the expected temporal order has been incorporated into their behavior.

Connectionist schemes have also been applied to the domain of scripts (e.g., Rumelhart, Smolensky, McClelland, & Hinton, 1986; Golden, 1986). These assure that all items, even those which are not in strict keeping with a script may be recalled as a result of their becoming connected to actual script items during learning. However, these models have difficulty producing series of temporally ordered items. It seems that a hybrid model which draws on the strengths of each of these approaches would be preferable and might possess characteristics which allow it to meet the following criteria we would require of a model of scriptal behavior.

First, and most importantly, the model must be able to produce items associated with a single script without intrusions from others. This will be referred to as the script criterion. Second, if this model is to simulate the planning and execution of script behaviors, it must be able to reason about these events such that it produces them in a constrained temporal order; the temporal criterion. Third, this model must be able to violate that temporality in cases where steps or actions must be performed out of order, but do not conflict with the causal structure of the episode;the temporal violation criterion. This flexibility is identifiable in the restaurant example (i.e. moving to a different table) given previously. Fourth, the model should be able to begin at any point in a script and provide the remaining actions (i.e. deal with partial scripts), as this ability has been observed in human subjects (e.g., Mannes & Kintsch, 1987); the partial criterion. The following is a description of several established models and the degree to which they do or do not meet the mentioned criteria. Then our hybrid model, fashioned after the Construction-Integration theory of Kintsch (1988), will be evaluated with respect to these same criteria.

Traditional Symbolic Accounts of Scripts

Many symbolic accounts of schema and script memory have been proposed. The most influential of these have been put forth by Minsky (1975) for schemata, and Schank and Abelson (1977) specifically for scripts. Minsky's frame theory gained much popularity due to the ease with which it dealt with visual scenes and their interpretation.

> A frame is a data-structure for representing a stereotyped situation like being in a certain kind of living room... We can think of a frame as a network

> of nodes and relations. The 'top levels' of a frame are fixed, and represent things that are always true about the the supposed situation. The lower levels have many terminals- 'slots' that must be filled by specific instances or data. (p.355)

In fact, these terminals can often be filled by default values and these frames can be interconnected with other frames. In the example of visual analysis the interconnection provides something like different perspectives on the scene unfolding before the subject. These frames have to be specialized and can perform quite competently in well described fields, such as the micro-world of the blocks domain, but fail to be effective in poorly circumscribed areas, such as natural language understanding. In addition to this shortcoming, the frames 'top levels', those parts which describe a global series of events, were necessarily but unfortunately too rigid, thus not allowing for the temporal violation criterion. If one thinks carefully about the kind of data these structures were proposed to account for, it becomes clear that there are many instances rigid frames could not encompass. For example, a bird without wings is still a bird.

Schank and Abelson's script theory gained notoriety for its ability to deal with the typical activities of events like going to the grocery store. In Schank and Abelson's formalism, knowledge about the world is organized in small packets much like frames. (These structures contain information about what usually happens in the described situation - general knowledge - as well as slots to be filled in by the particular episode under consideration - specific information.) While this format afforded flexibility it was incurred at some cost. Much general knowledge may be brought to bear in understanding what typically happens in a restaurant, but exceptions to the rule have to be explicitly filled in, sometimes contradicting a piece of general knowledge from the currently active script. The most popular example of this deviation from the norm comes from the Legal Seafood Restaurant in Boston. This is the restaurant mentioned earlier where customers are expected to pay for their food before it is served. As mentioned, this is consistent with a fast food restaurant script for a place like McDonald's, but as anyone who has ever been to Legal Seafood knows, it is not like a fast food establishment in any other respect (particularly the price). In this example, we need to incorporate disparate pieces of information from at least two existing scripts; the current 'nice place' script and

the 'fast food' one. In this model script determination is difficult and in some instances no suitable pre-existing scripts can be found, again, prohibiting the accomplishment of the temporal violation criterion. Paying before being served is difficult to achieve in script based systems because the necessary information resides in two separate scripts.

The inability of these approaches to deal flexibly with the wing-less bird (or a sink-less kitchen, for that matter) and the script combination problems that occur when trying to understand, for example, the Legal Seafood restaurant have led artificial intelligence researchers and psychologists alike to search for a more suitable representations for scriptal knowledge.

A Recent Symbolic Attempt

A more flexible, but still essentially symbolic, account of the script data has been put forth by Kintsch and Mannes (1987). By adding an additional control mechanism for dynamic cue composition (a cue being an item or pair of items which is used to sample other items from memory) to a standard psychological model of retrieval from associative memory (Raaijmaker and Shiffrin, 1981), Kintsch and Mannes were able to account for script retrieval data which they collected, as well as that of Bower, Black and Turner. Instead of representing scripts as large structures they use much smaller portions of scripts as the basic unit of analysis. In their model, memory is represented as an associative network with the matrix entries standing for the degree to which the two connected items are related to one another, this relationship being determined by experimental evidence about the co-occurrence of and temporal ordering between these items. The content of the matrix nodes is represented propositionally with items such as MAKE(YOU, RESERVATIONS) or TAKES(WAITRESS, ORDER). When given the title of a script to be retrieved the model forms a search cue with the title and searches the matrix probabilistically for items highly associated with the cue. The cue changes dynamically when an associated item is found. The associated item is included in the retrieval cue and the search continues, now looking for items which are highly associated with both elements of the cue - the title which was given (T) and the particular proposition which was recalled (P). The model continues retrieving cue associates, dynamically replacing (P) with subsequently retrieved propositions until no more associates to the current cue can be found. The control process then takes over and

goes on a search for a proposition containing some temporal information about what happens next in the script. A simple pattern match will find a proposition of the form AFTER(X, Y) where X is a proposition currently in the cue. The newly found proposition is added to the cue and the associative search is restarted using the new ever-changing, compound cue. This cycle of probabilistic search and controlled search continues until the entire script has been recalled. The model produces most of the same actions as subjects in the previously mentioned studies. It produces them in the correct temporal order (the temporal criterion). It does not produce intrusions from similar scripts although they are represented in the network and there is much overlap between the different scripts in terms of actions (e.g., no irrelevant intrusions from going to the shoe store occur when trying to recall a grocery store script even though they share features such as getting to the store, selecting an item, and paying for purchases) (the script criterion), and the model can be started at any point in the script and complete it successfully (the partial criterion). In this formalism, then, scripts do not exist as static structures, but are created on the spot in response to whatever task or situation demands exist.

One major limitation of the Kintsch and Mannes work is that the temporal relations between different script episodes are static, thus not allowing for the temporal violation criterion. For the model, when going to the grocery store, one must always go to the store before one can peruse the aisles. This would therefore require a separate set of temporal relation propositions to take into account a circumstance where perusal may precede shopping, as in looking at a store catalogue to decide what to buy before visiting the store. Although representing scriptal information in this distributed manner allowed for greater flexibility in terms of what piece of scriptal information was produced when, the temporal ordering was still somewhat constrained thus violating the temporal violation criterion, that of being able to violate a strict ordering when necessary.

A Local Connectionist Framework

An approach similar to Kintsch and Mannes' has been presented by Golden (1986). In Golden's model, the basic concept is called a "causal relationship" (CR) and is represented in a manner quite similar to that used by Kintsch and Mannes, particularly for their propositions which reflect the temporal information mentioned

previously. His node contents are of the form (S3,A3,S5), indicating that through an action A3, state S3 is transformed into state S5. While Golden tries to make an argument for interpreting his model in neural terms this is not necessary for the model to be useful and comprehensible. Partial situations (e.g., the start of an event such as going to a restaurant) are described in the model by multi-dimensional subvectors of a total state vector whose entries are real numbers representing the activity (or rate of neural firing in Golden's terms) of each of the "neurons" in his system.

Unlike Kintsch and Mannes, Golden's model learns the connection strengths by use of the autoassociative Widrow-Hoff learning rule. As it learns, the model tries to minimize the difference between some given, but improbable, vector X and a more likely interpretation. It is in essence an error correcting model, correcting the current state of the system to coincide more readily with the input it is given.[3] That is, given a story to learn containing a stimulus (Y), it tries to update the weights in the network to maximize the future probability of Y in that same story. When the learning is complete, stimuli which have been experienced will have a much higher likelihood of being produced than those which have not. Interestingly, Golden models the Bower, Black, and Turner data differently than do Kintsch and Mannes. He addresses the issue of what script items are recalled, regardless of ordering, rather than the function according to which this information is recalled. His data make no claims about the sequentiality of scripts. Hence, the Golden and Kintsch and Mannes results are not entirely comparable, but some conclusions can be drawn nonetheless.

---

[3]As a related point, it is quite interesting to note the relationship many connectionist models have to the traditional concept learning literature. For many years psychologists debated about exactly when and under what circumstances learning took place. Through extensive contrived experimentation most researchers came to the conclusion that human learning in a concept identification task occurred only on trials where errors had been made. Only when a subject was incorrect about a hypothesis he or she held regarding the rules describing what an instance of the concept was did they learn anything about the concept to be formed. It was shown that, in traditional modeling efforts, the appropriate learning descriptions could only be produced when this assumption, learning on error trials only, was made. In the most typical connectionist learning models this is also the case. Weight adjustment, or learning of the system, only occurs when there is discrepancy between the system "hypothesis" and the desired output which can be reduced according to some rule. When the system "knows" what it is supposed to do and performs correctly, no weight adjustment (i.e., no "learning") takes place.

The first step for Golden was to create the long-term memory which Kintsch and Mannes had built in based upon subject data. His auto associative network consisted of 107 possible causal relations; the weights were trained using the Widrow-Hoff rule such that the activity patterns representing the training stories became stable configurations of the network. The model was then presented with new stories and was tested for retrieval. While the stories were quite impoverished, consisting of only five causal relationships each, there were 24 of them, enough to present a challenge to the model. During this phase

> an incomplete C(ausal) R(elationship) representing an initial situation (e.g., (S1,0,0)) is presented to the BSB model which reconstructs the action field of the CR. The effect of this action upon the environment results in a new situation (e.g., (S2,0,0)) that, in turn, can be used by the BSB model to reconstruct the second action in some action sequence. (p. 18)

This is analogous to the dynamic cue composition which takes place in the Kintsch model and the effect is quite similar. As in the Bower, Black and Turner study, mentioned items were recalled with a higher frequency than unmentioned items and scriptal items which were not mentioned were still recalled with greater probability than nonscriptal intrusions. Again, it is impossible to know whether these items were produced in an acceptable temporal order (the temporal criterion) or whether Golden even intended for this to be so.

Distributed Connectionist Accounts

Of those distributed models proposed to deal with schemata that of Rumelhart, Smolensky, McClelland and Hinton (1986) has been the most successful and popular. Although they have not addressed the problem of temporally organized schemata to the extent that Golden and others have, their formalism is worthy of mention because it deals with schemata in a distributed, rather than localized, memory system.

Rumelhart et al. tried to instill the concept of room schemata in their network. Forty different items which typically occur in rooms or are descriptors of rooms were presented to subjects and they were asked about how likely these items were to occur in rooms they were asked

to imagine.[4] This enabled the modelers to create a network that represented all of the items and their degree of interrelatedness. (It is important to realize that although this might seem localist in the sense that, for example, a television is represented locally, the content of room schemata, which is what is at issue here, is indeed distributed across the elements of the network.) The question, given this network, concerned what the model would do when presented with information regarding the presence of one of these items: Did it have a sense of the schemata necessary to determine what room was being implied by that item? This question was investigated by "clamping on" one of the items and examining the network's pattern of activation after it had been allowed to settle. Clamping in this sense refers to setting the activity of the item to one. That is, the system is told that this one item definitely occurred and its job is to provide the other items which most likely occurred with it. Subjects were asked to make judgements about five room types and there are five maxima (i.e. five peaks in the multi-dimensional space which represents the network) which represent each of the rooms respectively. Although there are a number of possible states that the model can get into, when one item is clamped on to begin, it only settles into one of the possible five room states.

> Schemata are not "things". There is no representational object which is a schema. Rather, schemata emerge at the moment they are needed from the interaction of large numbers of much simpler elements all working in concert with each other. Schemata are not explicit entities, but rather are implicit in our knowledge and are created by the very environment that they are trying to interpret-as it is interpreting them. (p. 20)

In a room schemata example, two elements might represent the concept of a sofa and a television, with the link between specifying that whenever one is present the other is very likely to be present also. Note, though, that it is not necessary for these things to co-

---

[4]The fact that only two subjects were used and that the rooms were presented to them to use as criteria for their decisions has been criticized by Schvaneveldt (1987). It is interesting, and I think important, that when Schvaneveldt's subjects were asked merely to judge the co-occurrence of pairs of items with no mention of the room in which they might occur, very similar results are obtained and the same room schemata emerge out of the resulting network.

occur in order for the network to settle on an interpretation of the stimuli. If one element is missing, the model's representation might not be as stable (i.e. Smolensky's (1986) harmony measure might not be as high) as it otherwise would be, but it does not have the sorts of problems a symbolic representation, e.g., a production system, would have.

Production system views of expertise entail a compilation of many simple production rules, and we might expect such a system to possess a rule that stated: IF (there is a sofa) AND (there is a television) AND (there is a lamp) THEN (conclude that the room is a living room). In this type of formalism, if one of the conditions doesn't match, the system must either decide it doesn't know what is being viewed or a complicated resolution procedure must be invoked to reconcile the apparently inconsistent information.

A connectionist network does not have the same limitations regarding missing and inconsistent data. The Rumelhart et al. network settles upon one of the five possible room interpretations almost exclusively, but more interestingly, it could be made to settle in the state representing, for example, an efficiency apartment where both a bed and a stove are present. Although this might seem irrelevant, it becomes a desirable feature when we return to the world of scripts and schemata on a larger level. It is only with this flexibility that such a network can make sense out of situations like the Legal Seafood Restaurant, the wing-less bird or finding an aspirin for a headache in a restaurant.

The Rumelhart et al. approach dealt quite effectively with the room schemata and can probably also account for other schema type phenomena. This model does have a limitation though. At the present time it is not obvious that this model can deal with the temporal information required to represent causal schemata or scripts (the temporal criterion). This is not a characteristic of all connectionist networks however, connectionist approaches that can describe the temporality necessary for this type of behavior have been proposed by, among others, Jordan (1986), and Kohonen, Oja, and Lehtio (1981).

Jordan and Kohonen et al. propose similar mechanisms to account for temporal behavior in their models: recurrency and delayed feedback. In recurrent networks and networks with delayed feedback, it is possible for a unit to cycle back and affect its own activation. Hence,

the input a unit receives is not only a function of the environmental input, but is also a function of the previous state of the elements. Briefly, the Jordan and Kohonen et al. approaches amount to adding a set of state nodes which essentially keep track of where in a sequence the model is currently performing. This enables the model to figure out what the next move or state should be. As elements of a script are processed they superimpose their activation on these state nodes allowing the total pattern of activation that the system experiences after, for example, item A as the first element of a list to be different from the pattern of activation that would result if item A were repeated in the same sequence. For the item sequence ABA the activation the system would receive on the first cycle would be the activation of item A alone whereas when it tried to encode the second item A, the pattern it would receive would be a combination of the (second) item A pattern coupled with the activation of the state nodes resulting from previous exposure to item A and item B. This eliminates the problem of keeping track of the correct place within a sequence at which the model is currently working and also eliminates the problem of choosing a next action when a script contains multiple instances of similar actions (like the generic action of ordering in a restaurant, which can occur any number of times for drinks, main course, dessert, etc.).

Though the Jordan and the Kohonen et al. models have been quite successful in mimicking whole script retrieval and representation they still fall short in one major area. Psychological data has shown that when subjects are presented with an activity from an event they are able to come up with the succeeding activity with very little effort almost immediately. That is, they can deal with partial scripts (the partial criterion). At this time, it seems that the models which have been proposed do not have the capability to report the remainder of a sequence when provided with a concept or activity from the middle of the sequence. (One possible exception to this is the cumbersome Geman (1981) formalism which we do not review because it has not been fully fleshed out.) Because of the superposition of information occurring in the stable state nodes, it seems unlikely that the appropriate pattern of activation can be imposed on these nodes by providing only one event from within the causally connected chain of events. That is, it is difficult for these models to recreate the appropriate context without actually going through the sequence needed to create it initially. Hence, in a sentence comprehension (or verification) paradigm like that used by

Bower, Black, and Turner, we would expect to see the graded distance effect which was absent in humans.

It seems clear that the parallel distributed models of memory have overcome some of the initial pitfalls which burdened symbolic models of memory for schemata, of both causal and non-causal natures. The approaches which have been taken to model room schemata, for example, are much more flexible than the original formulations in terms of frames and stable scripts. They allow the system to make decisions in the face of incomplete and inconsistent information, such as in the case of the efficiency apartment which is neither a bedroom nor a kitchen. The models which have tried to address the problem of the temporality inherent in scripts have also been successful, though the models have not been designed to handle all of the complexities of the issue.

Although the models of Jordan and Kohonen et al. are quite capable of producing script like recall from start to finish, there remain human data which must also be accounted for before any one of these models can claim to have captured the essence of human information processing in this domain. It appears that none of the models presented are able to meet the four criteria mentioned at the start of this paper: non-intrusiveness of other related scripts (the script criterion), sequentiality of item production (the temporal criterion), robustness to violations in script item order when necessary (the temporal violation criterion), and ability to deal with partial scripts (the partial criterion). Entirely connectionist schemes have particular difficulty with the matter of sequentiality, whereas purely symbolic frame and schema systems have difficulty with the other three criteria because they require more flexibility than they possess. It seems clear that other, perhaps combined, sophisticated methodologies are required in order to deal effectively and completely with the human phenomena of schemata and script representation. It is with this in mind that we now turn to the hybrid model which we have used to model a particular type of scriptal behavior, the Construction-Integration model of Kintsch (1988).

The Construction-Integration Model

The Construction-Integration model (Kintsch, 1988) is a hybrid model initially developed to explain certain phenomena of text comprehension, such as word sense disambiguation. The model has

been extended to model the planning of scriptal behavior, and it will be shown how it can meet criteria the "pure" models previously mentioned cannot. It uses a set of symbolic production rules to *construct* an associative network of interrelated items. These rules are weak in that they construct connections between items without respect for the current context or task at hand. This network is the basis for a second phase in which the *integration* takes place via connectionist constraint-satisfaction search. This process propagates activation throughout the network, serving to strengthen the connections between items which are consistent with each other and the context, and deactivating those items which initially were connected to others in the network, but are inconsistent. This model is implemented in the NETWORK[5] simulation. (Note: A model with similar goals and constitution has been proposed by Elman, 1989, but is beyond the scope of this paper.)

As the domains of human-computer interaction, computer usage, and intelligent system design have grown, our work has gravitated towards modeling scriptal behavior in the computer domain. Specifically, we are interested in how experienced computer users accomplish standard computer tasks, such as command production, in a seemingly automatic way. We hypothesized that because these procedures are overlearned for our users, they formed a certain type of domain dependent script and we are modeling them as such. Looking at users interacting with a computer to accomplish particular goals one may observe them producing orderly sequences of events (the temporal criterion) without intrusions from other tasks (the script criterion). They are able to modify their interactions depending upon the context (the temporal violation criterion) and they can complete a task regardless of the point at which they begin (the partial criterion).

Two sets of experimental studies (Mannes & Kintsch, 1988; Doane, Pellegrino, & Klatzky, in press) provide evidence for this approach. In one set, subjects were asked to provide verbal protocols about performing computer tasks, while in the other, subjects were asked to produce typed protocols consisting of UNIX commands to accomplish particular goals. In the verbal protocol study, seasoned computer users were asked to provide protocols according to the

---

[5]Note that in the remainder of this paper NETWORK will be used to refer to the program which implements the model, whereas, network in lower case will be used to refer to associative networks as they are traditionally understood.

standards set forth by Ericsson and Simon (1984). In general, the experimenter derived representations of the coded protocols resembled quite closely those obtained from researchers investigating standard scripts (e.g., Bower, Black, & Turner, 1979; Kintsch & Mannes, 1987). This work is described in detail in Mannes & Kintsch, 1988). In the other set of studies (Doane, Pellegrino, & Klatzky, in press) subjects were asked to produce efficient UNIX commands that varied in complexity while working at a computer terminal.

Initial Model Construction

The areas mentioned above, routine computing and UNIX command composition, were modeled using the NETWORK system and will be described below. After evaluating the verbal protocols three types of information could be identified; information subjects produced about their plans of action (or script for the particular task), meta information where general knowledge (e.g., about computing and computers) played a role in the solution attempt, and also keystroke information which is of no interest in the current investigation.

Both the plan of action and the meta information were propositionalized according to standard procedures (Bovair & Kieras, 1985; Turner & Greene, 1978) and each proposition then became a node in the network representation of the domain. In this format each proposition is an atomic unit which contains a predicate and some number of arguments. For example, the propositionalization of the sentence "Mike throws a ball." would appear as (THROW MIKE BALL). (Note: Propositions may also take as arguments other propositions, resulting in propositional embedding.)

Plan information was described as a set of plan elements, simple actions out of which entire plans could be synthesized. These were represented in an extended propositional format with three propositional fields, a name, preconditions, and outcomes. By representing information propositionally, knowledge about the domain and even a particular task is represented in a distributed manner although the node content remains symbolic and identifiable. The manner in which the relationships between these items were derived is described next.

Several rules were used to establish connections among items in the network. That is, to assign the various weights relating propositions.

Because this is an extension of a model of text comprehension to planning command sequences and we are treating this effortless type of problem solving as a matter of comprehension, certain linguistic relations, such as argument overlap and propositional embedding, were used for deriving links between propositions. These provide a crude approximation to the types of metrics people are hypothesized to use when comprehending a text (Kintsch & van Dijk, 1978) and thus, when propositions shared an argument or were embedded within one another a weight specifying this relationship was entered in the matrix representing the network.

In addition, several relationships of a non-textual nature played a role in specifying the weights in the development of the network. In particular, items which were associated to each other, as determined by a free association study (see Mannes & Kintsch, 1988), were related with each other with a particular weight, and the plan elements which NETWORK uses to produce a plan of action dynamically, step by step, were linked to each other in a causal chain. The causal chain was derived by assessing matches between the precondition and outcome fields of the plan element's representation, such that plan elements which provided as their outcome a precondition for another plan element would receive a link from that plan element. Also, a plan element which could destroy as a result of its execution a precondition for another plan element received an inhibitory link from that plan element. This allowed for plan elements which provided essential preconditions to receive activation from the plan element requiring that state of affairs during the integration phase. Likewise, the inhibitory connections between plan elements allowed for the flow of inhibition. Figure 1 shows a small subset of plan elements and the manner in which they would have become related via causal chaining as shown by the matrix in the bottom panel. All of this information is related to form the system's long-term memory. This memory is used as a source of knowledge for all of the tasks NETWORK can perform.

An Example Solution for Conflicting Subgoals

In order to assess the functionality of this approach simulations of several tasks were done. Although several of these tasks were taken directly from the domain which we endeavored to simulate, routine computing, others were modeled after those found in literature on artificial intelligence. This section is included to contrast methods used by the traditional AI programs and those used by NETWORK to

Three plans in their generic (unbound) form.

NAME	(FIND FILE)
PRECONDITION(s)	(AT-LEVEL SYSTEM) (EXIST FILE)
OUTCOME(s)	(KNOW FILE LOCATION)
NAME	(DELETE FILE)
PRECONDITION(s)	(AT-LEVEL SYSTEM) (EXIST FILE) (KNOW FILE LOCATION)
OUTCOME(s)	NOT (EXIST FILE)
NAME	(ENTER SYSTEM)
PRECONDITION(s)	(AT-LEVEL MAIL)
OUTCOME(s)	(AT-LEVEL SYSTEM)

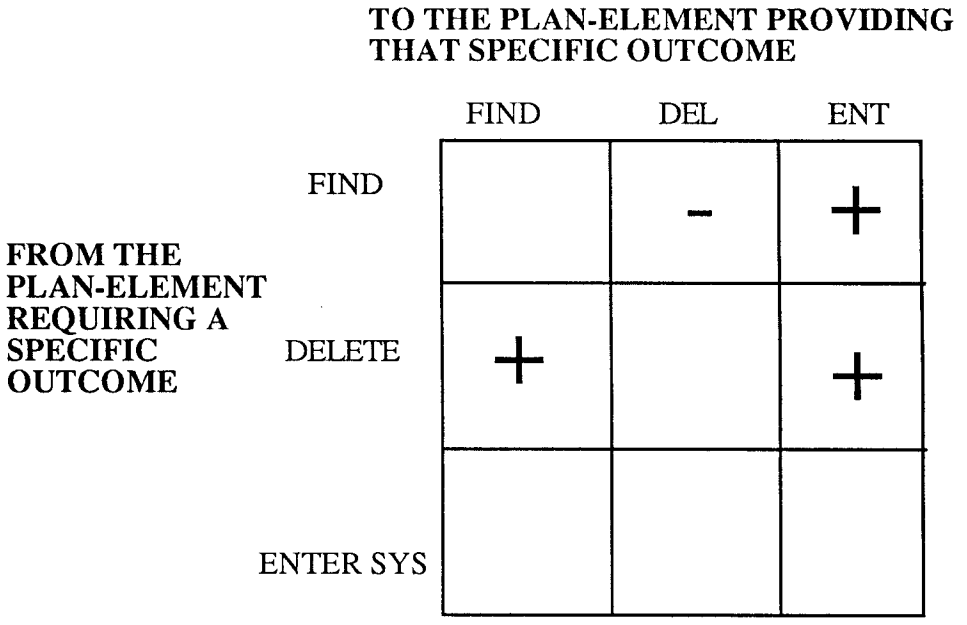

Figure 1. A sample of three plan elements are shown in the top panel and the resulting causal chain in matrix form in the bottom panel.

accomplish one type of task. The task chosen is one which has proven to be difficult for AI systems and has been coined the "conflicting subgoals" task. This is best exemplified by the example of a painter who has been asked to paint both a ceiling and a ladder. In this scenario, the painter must have the foresight to paint the ceiling first, for painting the ladder first would preclude its use as a tool for subsequently painting the ceiling. This task is script-like in nature because it requires the production of a sequence of events which MUST be produced in a particular order. Traditionally, this is a difficult task because problem-solving systems have begun it by performing explicit subgoal decomposition. This neglects the fact that the subgoals in this case interact and has led to the development of a series of specialized critics to search the proposed problem solution for these types of interactions and other problems (e.g., Sacerdoti, 1977; Sussman, 1975). The following describes how the NETWORK system deals with this class of problem and why its hybrid composition allows it to solve the task effortlessly and correctly.

Our initial concern in this endeavor was to model behavior in the computing domain and our system, therefore, contains plan elements specific to that domain. The amount of effort required to compose a similar set of plan elements for the painting domain would have made this exercise prohibitive, so a routine computing analogue to the paint the ceiling and ladder problem was devised. This allowed us to produce simulation results using the plan elements which had already been created while allowing us to evaluate NETWORK's performance with respect to attempts of other systems. The analogue chosen asked NETWORK to print and delete a file. Here, the system must recognize, without explicit instruction, that deleting the file will make it unavailable for printing and, thus, the (PRINT FILE) command must be executed first. To begin, NETWORK is given the task description in propositional form. This description activates related information in NETWORK's long-term memory, including all of the plan elements it knows about. These plan elements include things such as (DELETE FILE), (PRINT FILE), and (READ MAIL). Then NETWORK uses its production rules to compute all of the types of relationships between propositions (e.g., argument overlap) previously mentioned. This results in a network where all the information to be used for the PRINT/DELETE task is incorporated. This network, written as a matrix, represents the system's understanding of the task to be done, and provides the input for the second phase of the problem solution, integration.

Integration is accomplished through standard vector-matrix multiplication methods. An initial activation vector is created such that there exists an element for each of the nodes in the network. This vector has the characteristic that all nodes representing the initial task description (i.e. delete the file and print the file) are activated and all others are set to zero. This vector is post multiplied with the matrix representing the network until there is minimal change in vector activation values from one multiplication to the next and the system is considered settled. When the system has settled, activation has propagated from the original task description throughout the network to the plan elements and the most highly activated one which has its preconditions met in the world is allowed to fire. Having its preconditions met requires that all of the propositions of a plan element's second field are present in the world proposition list.

The firing of a plan element has the consequence of adding its outcome(s) to the state of the world and an updated network is constructed adding this new information to old. The integration phase is then restarted. This series of steps continues until a state of the world is achieved in which the original task goal has been satisfied.

The following is the proposition list given to NETWORK at the beginning of the PRINT/DELETE task.

(EXIST FILE^EGGPLANT)
(REQUEST PRINT FILE^EGGPLANT)
(REQUEST DELETE FILE^EGGPLANT)
(OUTCOME-OF-REQUEST-PRINT EXIST HARDCOPY FILE^EGGPLANT)
(OUTCOME-OF-REQUEST-DELETE NOTEXIST FILE^EGGPLANT)
(AT-LEVEL SYSTEM)

This description activates associated information and all plan elements from NETWORK's long-term memory. A network is constructed using all of these propositions; task description, related knowledge and plan elements, and then integration is performed. For the first step of the PRINT/DELETE task, (FIND FILE) is the most highly activated plan element with its preconditions met after activation ceases to spread further. It fires, producing the location of the file to be printed and deleted. Note from the plan elements shown in Figure 2 that this provides a precondition for both the

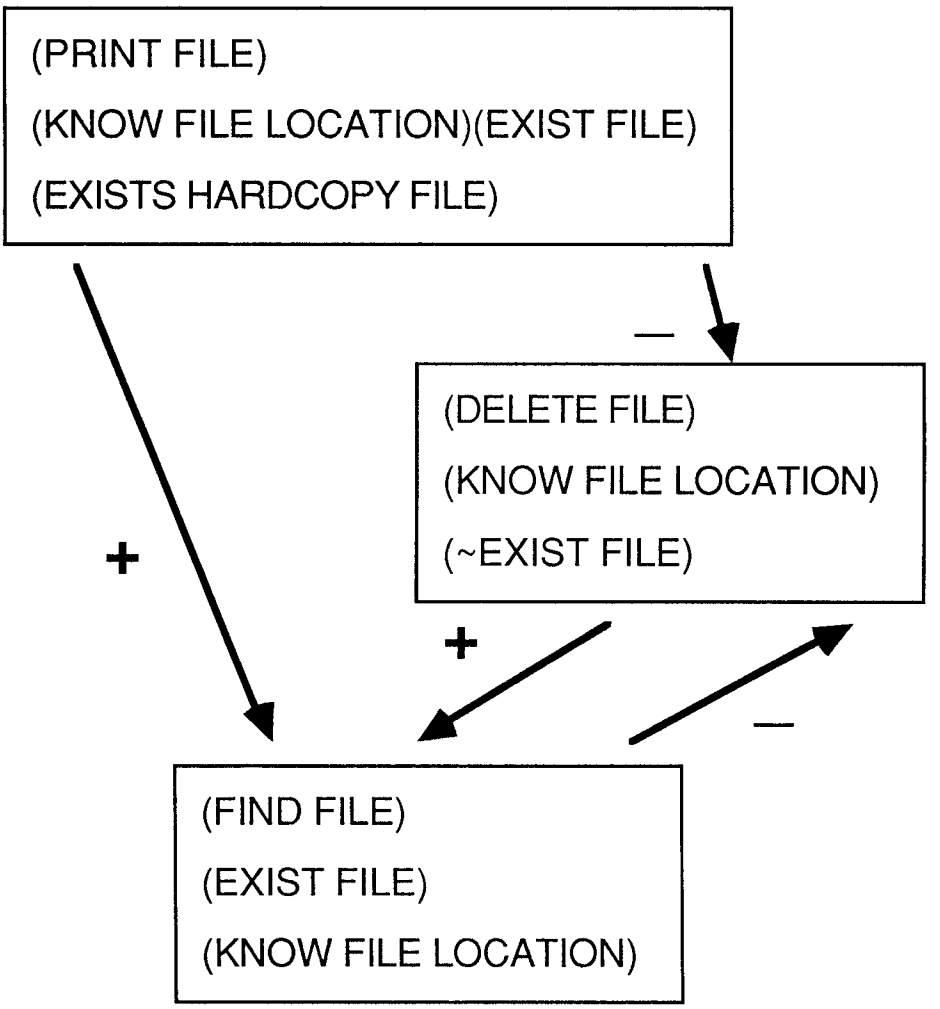

Figure 2. This figure shows how PRINT and DELETE both send positive activation to FIND and how PRINT and FIND both inhibit DELETE as a result of causal chaining.

explicit goals, hence the plan element which produces this outcome receives activation from allplan elements which require it as a precondition during integration.

At this time, one more method of determining connectivity between the network nodes during construction must be explained. Propositions representing the outcomes of plan elements inhibit the plan elements which produce them. This means that if the outcome of a plan element has already been established, either because it was a part of the original task description or, as in this case, it has become true as the result of a plan element having fired, it inhibits that plan element from firing again. Here, once the location of the

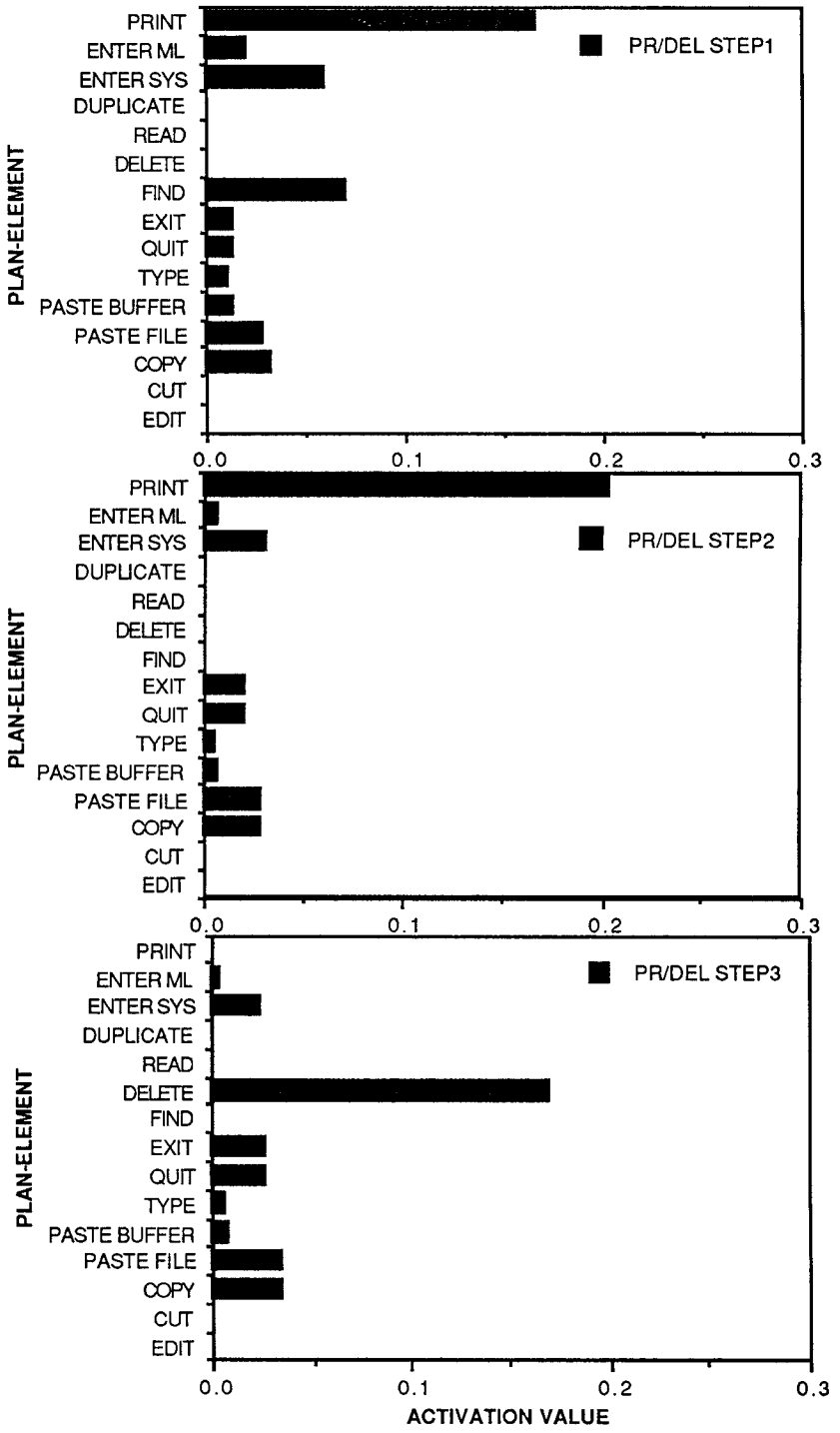

Figure 3. Activation values for some plan elements for the three steps of the PRINT/DELETE task.

file is known, as produced by the (FIND FILE) plan element, (FIND FILE) is inhibited by the proposition reflecting the file's location. With this newly added method of connectivity in force, the PRINT/DELETE network is updated and integration is done once again for the second step of the problem.

In this step, the plan element to (PRINT FILE) is highly active and, as can be seen in Figure 3, the plan element to (DELETE FILE) has no activation at all. (PRINT FILE) fires, and its outcome, that there (EXISTS HARDCOPY FILE), is added to the network. Thus, the system has accomplished one of its two goals and is progressing in the proper order. As in the previous step, the proposition representing the plan element outcome has the ability to inhibit the plan element which produces it. In this case, the plan element to (PRINT FILE) is inhibited by the fact that a hardcopy already exists. This plays an important role as can be seen in the results of the next construction-integration cycle.

The third integration step produces a vector of activation values in which the plan element to (DELETE FILE) is most active (see Figure 3). This state of affairs can come about because of the plan-world inhibition mentioned earlier. In step 2 the plan element to (PRINT FILE) was very active and as a result had a strong inhibitory effect on the (DELETE FILE) plan element via the plan element causal chaining. After its firing, the plan element to (PRINT FILE) becomes inhibited itself and, therefore, no inhibition propagates from its node to the (DELETE FILE) node. This allows (DELETE FILE) to receive the activation the direct request provides and makes it the most active plan element. (DELETE FILE) fires and NETWORK has correctly completed the conflicting subgoals task.

It is not immediately obvious from this example how NETWORK meets the four criteria previously mentioned. It clearly produces the set of script events in the proper order (the temporal criterion) without suggesting other irrelevant plan elements (the script criterion). A second, unreported, simulation of NETWORK solving the PRINT/DELETE task after the file had already been found was successful, as have simulations of other more lengthy tasks started at various stages of completion (the partial criterion). This is a result of the fact that NETWORK takes into account the current state of the world at each step of a task, and changes its connections in response to the ever-changing state of the world. In this case, the temporal

violation criterion is unimportant, but would not be a problem anyway because, in NETWORK, stable scripts don't exist!

The fact that scripts don't exist in NETWORK allows for additional interesting contextually driven phenomena. One example of this can be seen in comparing how NETWORK approaches similar tasks differently depending upon the information is has been given. One task, REVISE, tells NETWORK that it is to revise a manuscript and send it to a colleague. The other task, SEND, tells NETWORK it has received a mail message from a friend of theirs asking for a copy of a revised manuscript. For REVISE, there is no mail message to which NETWORK can reply (perhaps the colleague had called on the phone with her request) so it chooses to send the revised manuscript as a file from the system level (as some operating systems allow). In the SEND task, there is a mail message to respond to so NETWORK chooses to enter mail and reply to the existing message with the file. Thus, a seemingly irrelevant piece of information, that the request was received via computer mail as opposed to through a phone conversation, encourages NETWORK to produce different solutions. This is a result of the additional weighting that the plan element for sending through mail (SEND FILE MAIL) gets from argument overlap (one method of determining network connectivity) with the task description which mentions the mail message. Another example of NETWORK's context sensitivity, and dynamic plan creation will be given in the next, UNIX, section.

A Real Task: The UNIX Domain

Although NETWORK shows proficient performance in solving the PRINT/DELETE task, it is what may be regarded as a "toy" problem. That is, it is a task in which humans show little or no effortful problem solving. It was our desire to simulate more effortful problem solving and to this end we chose a different data set of some complexity to attack. These data (from Doane, Pellegrino, & Klatzky, in press) concern users attempting to perform several tasks of varying complexity in the UNIX operating system. The following is a brief description of those data and an overview of our attempts at modeling them. A brief description of the empirical results is followed by a description of simulating UNIX command production.

Doane et al. examined the development of expertise within the UNIX operating system by measuring users' performance in tasks requiring them to produce UNIX commands. In the production task,

subjects were asked to produce the most efficient (that with the least number of keystrokes) legal UNIX command that they could to accomplish a specified goal. Goals ranged in difficulty from individual, frequently used UNIX commands to composite commands that accomplished several actions, and had to be sequenced appropriately using pipes or other input/output redirection symbols. Correct production of the composite goals required knowledge of individual commands and knowledge of the processes involved in sequencing those commands properly. Tasks involving more elementary commands were designed to include elements that had to be put together to generate a successful composite command. Thus, these are the commands of interest.

An example composite task would be "display the first ten alphabetically arranged lines of file x". A component single would be "display the first ten lines of file y". A multiple would be "display the first ten lines of file y, and arrange the contents of file x alphabetically on the screen". Thus, "single" commands caused just one action, multiple commands caused two or more independent actions and composite commands caused several actions that had to be sequenced appropriately using pipes and/or redirection symbols. These were equal in length to the multiple commands; thus the essential difference between the two was that composites had dependencies among their components whereas multiples did not.

The goals given to the Doane et al. subjects can be seen as requiring the use of scripts. To accomplish composite goals, subjects had to put familiar elements together in a novel fashion. We are not assuming that subjects were recalling fixed scripts from memory. Rather, they were producing script-like action plans on-line. Thus, we are trying to simulate the solution of tasks requiring script-like knowledge.

We modeled a portion of the expert production performance measured in the Doane et al. (in press) research using an extended version of NETWORK. One of our goals was to understand in detail the kinds of knowledge that are sufficient to produce composite commands. In the following sections, we will discuss only a fragment of the UNIX simulation work (for details, see Doane, Kintsch, & Polson, 1989), with the goal of explicating how the symbolic and connectionist portions of NETWORK interact to accomplish the composite command goals.

As in the PRINT/DELETE task described above, NETWORK is initially given three types of information in propositional form; long-term memory knowledge relevant to the domain (e.g., knowledge of UNIX), knowledge of the task description (e.g., "list the first ten lines of file x"), and knowledge of plan elements. The following describes each of these pieces of knowledge as they were given to NETWORK to simulate the production of UNIX composite commands.

To simulate expert performance on composites, the model requires more long-term memory knowledge about UNIX than is necessary to perform singles and multiples. In addition to knowledge of the basic types of commands and their syntax (the only knowledge required to execute singles and multiples), the model had to possess knowledge of the ordering of commands in a sequence. For example, to execute the task "display the first ten alphabetically sorted lines of file x", the model must know that the command that sorts files alphabetically (SORT) should be executed on file x prior to executing the command that will display the first ten lines of the file (HEAD). The model must also know about the specific redirection properties of each command. In the example task above, the model must know that the output of SORT can be redirected to another command (in this case HEAD), and that the input to HEAD can be redirected from another command (in this case SORT). Finally, the system must know the syntax required to produce a command sequence (e.g., know that LPR prints a file, and that the pipe symbol redirects input and output between commands).

As in the PRINT/DELETE task, NETWORK is initially given the task description in propositional form. This description activates related information about UNIX in NETWORK's long-term memory, including all of the plan elements it knows about. In Doane et al. (1989), some of the plan elements corresponded to single building-block commands (e.g., SORT, HEAD), and of these, some corresponded to building-block commands which were irrelevant to the simulated tasks. Other plan elements allowed creation of new plans. These latter plan elements are called building plan elements, because they allow NETWORK to build a composite command from the single building-block commands.

Two of the single plan elements, and one of the build plan elements are shown, with their text in abbreviated form in Table 1.

Table 1. A sample of the UNIX domain plan elements.

Plan Name	Preconditions	Outcomes
(SORT FL)	(@SYS) (EXIST FL) (KNOW SORT)	(SORT FL)
(DISP 1ST TEN LINES FL)	(@SYS) (EXIST FL) (KNOW HEAD)	(DISP 1ST TEN FL)
(BUILD PIPE)	(KNOW SORT 1ST) (SORT FL) (KNOW HEAD 2ND) (DISP 1ST TEN FL) (KNOW REDIRECT HEAD OUTPUT) (KNOW REDIRECT SORT INPUT) (KNOW REDIRECT SORT OUTPUT) (KNOW REDIRECT HEAD INPUT)	(USE PIPE PLAN)

As previously described, NETWORK includes three components in each plan, a plan name, preconditions, and outcomes. The preconditions are propositions which represent states of the world which must exist for the plan to be executed. The outcomes are propositions which become states of the world if the plan is executed. Once the propositions corresponding to long-term memory, task description, and plan element knowledge are given to NETWORK, it uses its production rules to compute the relationships between the propositions, and sets the weights in the network which includes all of the information to be used to solve the production task. This weight matrix is NETWORK's representation of the current task, and it provides the input for the integration phase.

Integration is accomplished in the same manner as it was in the PRINT/DELETE task. The initial activation vector is post-multiplied with the memory matrix until the change in activation between activations is small and the system is considered settled. When the system has settled, the most activated plan element whose preconditions are met is allowed to fire. This process is reiterated until the plan element whose outcome accomplishes the specified goal fires.

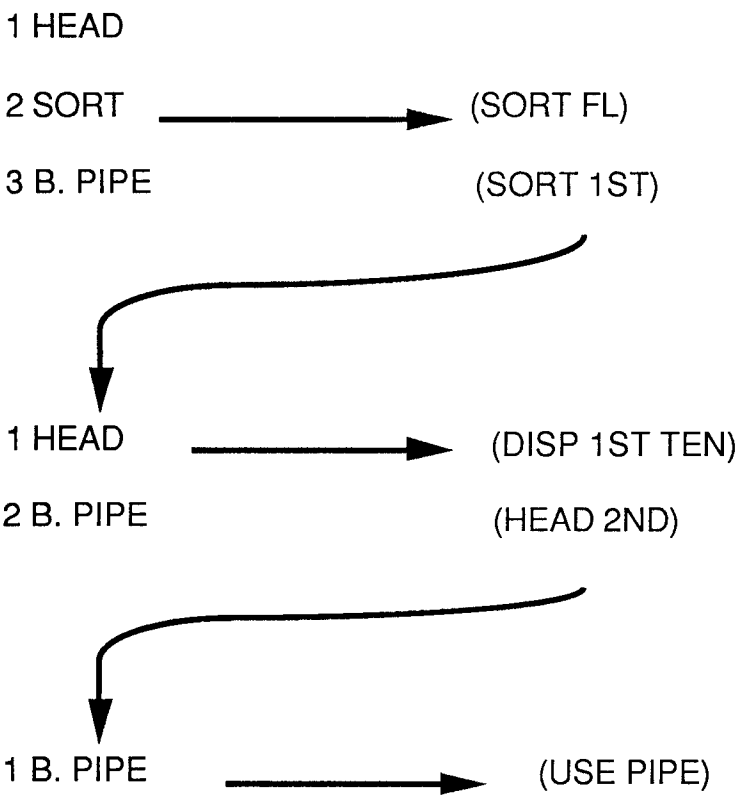

Figure 4. Results of simulating the composite command SORT FILE-X|HEAD. The build pipe plan is abbreviated as B PIPE.

Figure 4 depicts the main results of simulating the component tasks discussed above in the form of a composite. The figure shows only those plan elements considered by NETWORK, along with their relative activation values. The simulation of a single command to sort the contents of file x alphabetically (SORT problem) is very simple for NETWORK, as it was for the subjects in Doane et al. (in press). The system finds all of the preconditions are met, and the plan element is executed immediately. For the multiple problem (SORT, HEAD), the executive first chooses the SORT goal, its preconditions are met, its outcome (the proposition (SORT FILE-X)) is added to NETWORK's knowledge. After a new integration phase, the most activated plan element is HEAD. NETWORK chooses to fire the HEAD plan element, its outcome (HEAD FILE-X) is added to the knowledge base, and the task is complete.

The composite problem (to produce SORT FILE-X|HEAD) is far more complex, since the task requires that the model go through both a planning phase and a building phase. In the single and multiple

goals, there are no interrelationships between sequences of actions, and the model immediately generates user actions upon determination of the correct plan element. In the case of the composites, we argue that the appropriate plan elements must be determined, and then sequenced correctly using the interrelationships among the preconditions for the various plan elements. Following each integration phase, this planning process adds representations of the individual commands and the order in which they should be executed to the in the world knowledge. In our example, the request which is acted on first is SORT; the system would like to execute the HEAD plan element, but its preconditions have not been met. The preconditions for HEAD in this task require the existence of an alphabetically sorted listing of file x. That is, the (EXIST FL) precondition has been bound to a specific file; the required file contains the alphabetically sorted file x. The only plan element with satisfied preconditions is SORT. It is fired, and the outcome (SORT FILE-X) is added to the current knowledge base, along with a representation of its order in the task sequence (that it is first). On the final planning iteration, the HEAD plan element is again the desired plan element, its preconditions are now met, the plan element is fired, relevant in the world knowledge is added, and the planning phase is complete.

Following the planning phase, these new representations are used by the build pipe plan element to create the actual composite command. The model executes the build pipe plan element when its preconditions are met, but the preconditions for the build pipe plan element are extensive. In order for the plan element to fire, the system must have, for each command, knowledge of the command syntax, knowledge of the command order in the sequence, and knowledge of the command redirection properties. Since we are discussing modeling the expert, the system has all of the prerequisite knowledge, and the build pipe plan element fires. The outcome of this plan element is the creation of a "use pipe" plan element, which is bound to the syntactically correct sequence of commands. When the use pipe plan element fires, the task is complete. Again, NETWORK has produced a sequence of commands (the temporal criterion) without intrusions (the script criterion), but this task has not addressed the temporal violation criterion and the partial criterion although we could have started the task at any point in the solution process (thus satisfying the partial criterion).

As when NETWORK solved the REVISE and SEND tasks with different plan elements depending upon subtle differences in the task instructions, the system also differentially performs UNIX tasks according to task instructions. In the SORT FILE-X|HEAD example, NETWORK has the option of activating a (BUILD REDIRECTION) plan as well as the (BUILD PIPE) plan, both of which would accomplish the task. In the task instructions, the model is told that pipe use is preferred, and the overlap of this instruction with the pipe plan-element arguments leads to its higher activation relative to the redirection plan.

In summary, NETWORK used a hybrid approach to accomplish a sophisticated planning problem: It simulated producing composite UNIX commands. The interrelationships between the plan elements in NETWORK played a crucial role in producing the sequence-dependent script-like behavior required for these tasks.

Why Hybrid?

Let's evaluate what was obtained from this hybrid methodology by summarizing the characteristics of each component starting with the rule-based, symbolic network creation.

Symbolic benefits. Because the plan elements of NETWORK are interconnected via their causal chaining, the system will not be tricked into performing the (DELETE FILE) plan element first in the PRINT/DELETE task or into firing the HEAD plan element before SORT has taken place. Unlike several traditional AI systems, NETWORK does no explicit subgoaling. As a result of this it is unnecessary to introduce new conflict resolution procedures. This also eliminates the need for representing tasks to be done at various levels of abstraction - an approach which has been taken by others (Sacerdoti, 1977; Stefik, 1981) to prevent problems of conflicting subgoals. That is, NETWORK is a non-hierarchical system which is economical in terms of both space and memory limitations. The causal chaining allows for NETWORK to "see" the effects of its plan elements in a manner similar to systems like NOAH. NOAH requires each of his plans to maintain ADD and DELETE lists; list which specify concepts which will either be added or deleted from the state of the world upon execution of that plan. Unlike those lists however, the causal chaining in NETWORK is created dynamically so that future actions need not be anticipated. For systems, like NOAH, utilizing those types of lists it is impossible to know the possible outcomes of an

action without knowing about the current state of the world. Thus, the lists must be continuously updated.

Having our network contents as identifiable entities, rather than truly distributed, also has benefits. We are able to evaluate the relative activation for our plan elements quite easily and we have maintained a method of representation, the proposition, which is supported by data as psychologically real (Kintsch & Keenan, 1973).

Connectionist benefits. What we have done is different from standard connectionist methods in that we create a network out of long-term memory dynamically for each task rather than utilizing one large network in which only a portion becomes active for any given task. However, we still obtain many of the benefits from connectionist methodologies. With regard to the benefits of connectionist methods, it seems as if the rule-based creation of the plan element causal chaining and other network relations ought to constrain the possible plans of action for NETWORK to take. However, if one looks at the activation values for all plan elements in the PRINT/DELETE task before activation has been allowed to spread via integration, the plan elements to (PRINT FILE) and (DELETE FILE) are equally active, since they are both explicitly mentioned in the task request. In this situation of plan elements with tied activation, NETWORK chooses one at random to fire. This means that there is a 50% chance that the file will be deleted before it ever has a chance to be printed. By allowing activation to propagate through the network, the (PRINT FILE) plan element has the effect of inhibiting (DELETE FILE) which would ruin one of its preconditions, that the file to be printed exists, to the extent that (PRINT FILE) itself is active. The result of the spread of activation is that the plan element to (PRINT FILE) is the most active in the first step. After it fires, it becomes inhibited by the state of the world. (PRINT FILE) has now lost its inhibiting power through becoming inhibited itself and the plan element to (DELETE FILE) takes on activity. It receives this activation through the explicit request to delete the file, and can then fire resulting in the completion of this multiple goal task. In the UNIX example, the flow of activation from the plan element HEAD to other plan elements which produce its preconditions, again allow for these steps to be accomplished without the need for explicit subgoaling.

There is another class of benefits we glean from utilizing connectionist methods as opposed to those in traditional planners.

Not only does NETWORK save space and time in the absence of fully established plans, but NETWORK is also able to settle on a *best* solution to a problem rather than *any* solution. That activity is allowed to flow throughout the network until it settles allows for optimal solutions to be selected, as shown in the contrast between how NETWORK differs in its solutions to the REVISE and SEND tasks and its ability to solve the sort and head problem in at least two ways.

Also, the fact that one plan element does not have to be fully supported, but only activated to a larger extent than competing plan elements, allows NETWORK to find a "best fit" solution even if a plan element which exactly matches the goal is absent.

Conclusion

What we have done is interesting and important for two different reasons. First, we have taken a model of text comprehension and adapted it for simulating data in a very different domain, routine computing. More importantly, we have combined traditional rule-based symbolic contents with connectionist methods for producing and interpreting activation values for those contents. Each of these gives us something in return, and keeps us relatively faithful to the psychological data we wish to model. That the elements of our networks are symbolic and in fact quite large in their being propositions, provides interpretable node contents while allowing for the task descriptions and knowledge about the domain to be represented in a distributed manner. There is much evidence to suggest that propositions are psychologically real entities and therefore are intuitively suitable as the basic units for a psychological model. On the other hand, the use of connectionist methods allows for behavior to arise which is not precompiled, but in many cases appears so. Thus, we can produce seemingly high-level, controlled, phenomena in a flexible, contextually dependent, bottom-up manner.

In addition, the fact that a plan element needs merely to be supported by the current context to a greater extent than others in order to fire, is a very valuable characteristic of our formalism. In many rule based systems, a single rule must be chosen for firing depending upon the satisfaction of the rule's preconditions. Here, plan elements will not fire if all of their preconditions are not met, but often spurious activation flows to plan elements which one would

Ericsson, K. A., & Simon, H. A. (1984). Protocol analysis: Verbal reports as data. Cambridge, MA: MIT Press.

Geman, S. (1981). Notes on a self-organizing machine. In G. E. Hinton and J. A. Anderson (Eds.), Parallel models of associative memory. Hillsdale, NJ: Lawrence Erlbaum.

Golden, R. M. (1986). Representing causal schemata in connectionist systems. Cognitive Science Society Proceedings. Hillsdale, NJ: Lawrence Erlbaum.

Jordan, M. I. (1986). Serial order: A parallel distributed processing approach. Technical Report number 8604 of the Institute for Cognitive Science: University of California, San Diego.

Kintsch, W. (1988). The role of knowledge in discourse comprehension: A construction-integration model. Psychological Review, 95, 163-182.

Kintsch, W., & van Dijk, T. A. (1978). Toward a model of text comprehension and production. Psychological Review, 85, 363-394.

Kintsch, W., & Keenan, J. M. (1973). Reading rate and retention as a function of the number of propositions in the base structure of sentences. Cognitive Psychology, 5, 257-274.

Kintsch, W., & Mannes, S. M. (1987). Generating scripts from memory. In J. Hoffman & E. van der Meer (Eds.), Knowledge aided information processing. North-Holland Publishers: Amsterdam.

Kohonen, T., Oja, E., & Lehtio, P. (1981). Storage and processing of information in distributed associative memory systems. In G. E. Hinton and J. A. Anderson (Eds.), Parallel models of associative memory. Hillsdale, NJ: Lawrence Erlbaum.

Kolodner, J. L., & Simpson, R. L. (1984). Experience and problem solving: a framework. In Proceedings of the Sixth Annual Conference of the Cognitive Science Society.

Mannes, S. M., & Kintsch, W. (in preparation). Planning routine computing tasks: Understanding what to do.

Mannes, S. M., & Kintsch, W. (1988). Action planning: Routine computing tasks. In A. A. Turner (Ed.), Mental models and user centered design. Institute of Cognitive Science, University of Colorado, Technical Report No. 88-9.

Minsky, M. (1975). Frame-system theory. In R. C. Schank and B. L. Nash-Weber (Eds.), Theoretical issues in natural language processing. MIT Conference proceedings.

Raaijmaker, J. G., & Shiffrin, R. M. (1981). Search of associative memory. Psychological Review, 88, 93-134.

Rumelhart, D. E., Smolensky, P., McClelland, J. L., & Hinton, G. E. (1986). Schemata and sequential thought processes in PDP models. In J. L. McClelland, D. E. Rumelhart, and the PDP Research Group, Parallel distributed processing. Cambridge, MA: MIT Press.

Sacerdoti, E. D. (1977). A structure for plans and behavior. New York: Elsevier North-Holland.

Schank, R. C., & Abelson, R. (1977). Scripts, plans, goals and understanding. Hillsdale,NJ: Lawrence Erlbaum.

Schvaneveldt, R. (1987). Schemata and proximities. Paper presented at the Rocky Mountain Psychological Association, Albuquerque, New Mexico.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In J. L. McClelland, D. E. Rumelhart, and the PDP Research Group, Parallel distributed processing. Cambridge, MA: MIT Press.

Stefik, M. (1981). Planning with constraints (MOLGEN: Part 1). Artificial Intelligence, 16, 111-140.

Sussman, G. J. (1975). A computer model of skill acquisition. New York: American Elsevier.

Turner, A. A., & Green, E. (1978). Construction and use of a propositional text base. JSAS Catalogue of selected documents in psychology, MS-1713.