Encoding Techniques for Complex Information Structures in Connectionist Systems

John Barnden & Kankanahalli Srinivas

MCCS-90-186

Computing Research Laboratory
Box 30001
New Mexico State University
Las Cruces, New Mexico 88003

Encoding Techniques for Complex Information Structures in Connectionist Systems*

John Barnden & Kankanahalli Srinivas

Computing Research Laboratory and Computer Science Department
New Mexico State University
Box 30001-3CRL
Las Cruces, New Mexico 88003-0001, USA
(505) 646-6235/4535 jbarnden/srini@nmsu.edu

Running Head: **Connectionist Encoding Techniques**

ABSTRACT

Two general information-encoding techniques called "relative-position encoding" and "pattern-similarity association" are presented. They are claimed to be a convenient basis for the connectionist implementation of complex, short-term information processing of the sort needed in commonsense reasoning, semantic/pragmatic interpretation of natural language utterances, and other types of high-level cognitive processing. The relationships of the techniques to other connectionist information-structuring methods, and also to methods used in computers, are discussed in detail. The rich inter-relationships of these other connectionist and computer methods are also clarified. We detail the particular, simple forms that the relative-position encoding and pattern-similarity association techniques take in our own connectionist system, called Conposit, in order to clarify some issues and to provide evidence that the techniques are indeed useful in practice.

* This work was supported in part by grant AFOSR-88-0215 from the Air Force Office of Scientific Research, to Barnden, and grant NAGW-1592 under the Innovative Research Program of the NASA Office of Space Science and Applications, to Barnden and C.A. Fields.

1. INTRODUCTION

Our purpose is twofold: to present and discuss two somewhat atypical techniques for *short-term information structuring* in connectionist systems, and to use them as a starting point for a detailed examination of the space of such techniques. This examination also brings in the connections to basic information-structuring techniques used in computer science — connections which are often ignored and whose significance is underestimated.

The issue of short-term information structuring is one of the major problems facing the application of connectionism to "high-level cognitive processing". We use this phrase as a shorthand to cover, for instance, commonsense reasoning, action planning, and the semantic and pragmatic aspects of natural language understanding and generation. In these areas there is a need for a cognitive system to be able to deal with highly dynamic and unanticipated conglomerations of information. For instance, in understanding the sentence "Mike gets angry whenever Sally talks about going to Tibet", the system must cope with the *particular combination* of ideas presented by the sentence. Although we presume that the system in some sense knows who Mike and Sally are and is familiar with the notions of someone getting angry, someone talking, someone going somewhere, and so on, it is quite possible that the system has never before had to consider *Mike* being angry, *Sally* talking about *going to Tibet*, or of someone's getting angry *whenever* (or *because*) someone talks about going somewhere. It is very likely that the system has never before encountered the particular proposition conveyed by the sentence.

Thus, it is important for a high-level cognitive system to be able to bring together, in some sense or other, some of its representations, concepts, knowledge or whatever, in a way that is *unanticipated in detail*, even though it may have had much experience in using those ideas in other combinations in the past. Moreover, the system must do the bringing together very *rapidly*, and the result of doing it must be that the system is now in a state that enables it to process the combined information in efficient ways.

Strongly related to the unanticipatedness issue is the *arbitrariness* of the way information can be combined. For instance, although normally it is people who are the agents of speaking actions, in children's stories one might find a banana talking; similarly, the system has to have the ability to represent a situation in which an adult *believes* that a banana can speak. It is very difficult to think of any firm constraints on what can be combined with what, especially when one brings in metaphorical language. Although one may legitimately claim that it is too early to start devising detailed connectionist systems that deal with real children's stories, radically unusual beliefs, and metaphorical language (though see Weber 1989 for a start on the last of these), one cannot use this claim to shut one's eyes to the problems that such applications *will obviously* bring in when they are eventually considered.

To the issues of unanticipatedness and arbitrariness we can add the sheer fact that the information structures that need to be processed cover a *wide range of complexity*. For instance, our example sentence above conveys a much more complex information structure than does a sentence like "Mike is angry". The variability of complexity is also self-nourishing in the sense that a "slot", like the agent slot for a talking action, can itself be filled with information of widely varying complexity. A sentence might specify that it is *Sally* who is doing something, or *Martin's mother*, or *the person down the road who always trips over her cat when coming back from work*. The reason we stress the three issues of unanticipatedness, arbitrariness and variability of complexity is that much connectionist research focuses on tasks that to a large extent lack these qualities. For instance, in a printed letter recognition task, only certain highly-restricted, anticipated combinations of features need be recognized, and for any given letter the complexity of the combination of features is essentially fixed (and different letters have roughly comparable levels of complexity of feature combination).

Many of the techniques to be discussed in this paper are promising approaches to dealing with some or all of the three problems of unanticipatedness, arbitrariness, and variability of complexity, though it cannot be claimed that any of these problems have been fully solved. The intent of our discussion is to illuminate the true richness and subtlety of the space of possible ways of structuring short-term information in connectionist systems. We believe that this illumination is boosted by bringing out also the rich connections between connectionist techniques and computer techniques. One motivation for doing this is to resist the excessive paradigmatic distance that has been placed between connectionism and computer science.

The two short-term information-structure techniques we ourselves will be putting forward are called "relative-position encoding" and "pattern-similarity association". We emphasize that it is *not* our purpose to argue that our techniques are superior to others, although we certainly hope the reader will agree they are useful. We should also emphasize that our own connectionist system (Conposit) is presented only in order to clarify the nature of our techniques and to make the discussion of general issues clearer and more precise.

We do not have room in the paper to do more than occasionally touch upon the important issue of the ways in which the various representational techniques that we address facilitate or hinder efficient processing, and allow sufficient "systematicity" of processing (a matter stressed recently by Fodor & Pylyshyn 1988,and which underlies the "variable-binding" issue). We are well aware that representation and processing are indissolubly linked, but must defer attention to details of the linkage to later papers. However, see Barnden (1984b) for an early though detailed treatment.

In the remainder of the paper, we will first review some important existing connectionist information structuring methods, aiming primarily for ones that have interesting relationships with our own techniques. Then in Section 3 we will briefly review the basic information-structuring methods used in computers. Section 4 will describe relative-position encoding and pattern-similarity association. Section 5 will explain how straightforward forms of relative-position encoding and pattern-similarity association are used in Conposit, our connectionist rule-based system. Section 6 will discuss the relationships among the various techniques described in the paper. Section 7 will conclude.

2. SOME CONNECTIONIST INFORMATION STRUCTURING TECHNIQUES

The purpose of this section is to outline some major connectionist information-structuring techniques which we will relate to PSA and RPE, and to each other, in Section 6. We defer the detailed discussion of the inter-relationships until that section. Although we will be covering many of the ideas to be found in the literature, we will not be exhaustive, either with respect to ideas or with respect to the particular systems embodying the ideas. Our review will not attempt to outline each system as a whole, and will omit many interesting aspects of the systems referred to. Nor will we be saying much about the relative strengths and weaknesses of the systems and techniques portrayed. And, we will be ignoring hybrid symbolic/connectionist systems — systems that are partially connectionist but have symbolic-processing aspects that purposely are given no connectionist implementation (see e.g. Hendler 1989, Lehnert 1990).

Our focus throughout this section will be on the following basic question, which lies at the heart of the technical challenge that high-level cognitive processing presents to connectionism: *How* *are pieces of information put into* **temporary association** *with each other in a connectionist system?* For instance, how are the different parts of a complex temporary proposition, or of a dynamically created plan of action, or of a complex temporary description of a visual scene, put together? How are rule variables bound to values? How are frame slots bound to values?

Positional Techniques

The label "positional" is our own name for a very commonly used, though relatively weak, type of technique. It is especially prevalent in connectionist systems directed at visual perception. The general idea is best approached by looking at a simple, prototypical system. This example system is hypothetical, but is abstracted from the word-perception system of McClelland and Rumelhart (McClelland & Rumelhart 1981, Rumelhart & McClelland 1982). The basic ideas to be attended to crop up in many other perceptual systems.

Our hypothetical system is designed to recognize single written words up to a certain length l, and its visual field is divided up into l separate regions, one for each letter position in a word. A word is always presented with its first letter in region 1, its second in region 2, and so on. Corresponding to each region there is a subnetwork dedicated to recognizing the letter appearing in the region. In each such subnetwork, there is for each particular letter a single unit dedicated to indicating, by means of a high level of activity, the presence of that letter in the subnetwork's region. We give the unit that indicates the presence of letter 'T' in position 1 the name T_1, the unit that indicates the presence of letter 'H' in position 2 the name H_2, the unit that indicates the presence of letter 'T' in position 4 the name T_4, and so on. The system also contains a single unit for each word up to length l in some lexicon. The unit for the word 'THAT' is connected (in some way that we do not detail) to the "letter units" T_1, H_2, A_3 and T_4. One of the consequences of these connections is that when these units are highly active, the 'THAT' word unit becomes highly active as well. We call these connections the "basic letter/word connections" for reference.

Let us abstractly think of a word, say 'THAT', as a set of letter instances glued together by temporary associations of type "followed by". So, there is an instance of letter 'T' that does not follow anything and is followed by an instance of letter 'H'. The question we wish to ask is: how are these abstract temporary associations represented in the system? The answer is that *they are implicit in the sheer "positions" of the letter units* T_1, H_2, A_3 *and* T_4 *in the system as a whole.* By the "position" of a unit we mean something abstract, namely the way it is connected into the network as a whole. The sheer fact that T_1 and H_2 are highly active represents the fact that a 'T' instance is followed by an 'H' instance. Further, the rest of the system "knows" that, say, T_1 represents a 'T' in *first* position. This "knowledge" is embodied in the fact that T_1 has basic letter/word connections *only* with word units for words whose *first* letter is 'T'. An analogous observation holds in the case of H_2 and word units for words whose second letter is 'H'.

Now, letter units are of course what represent the letter instances (in our abstract view) themselves. Therefore, no temporary state (activation values etc.) other than the temporary states of *the units representing the abstractly-associated items themselves* is involved in the representation of the abstract associations. In particular, there is no change of connection weights, and no setting of particular activation values at units other than the letter units.

In sum, the space of possible abstract associations of interest has been in a sense "hardwired" into the gross structure of the system, We like to talk in terms of the "position" of units in the whole network, because it provides a more evident link to other ideas we will be discussing. It is important to realize also that the label "positional" does *not* derive from the fact that what is encod*ed* in many applications of the technique is physical position, in words or some other type of space. Nor does it derive from any notion of physical position in a physical realization (biological or electronic) of the network itself.

A less obvious instance of the positional category is evident in applications of a method derived from Wickelgren (1969) and well-known (as the "Wickelfeatures" technique) for its appearance in the past-tense model of Rumelhart & McClelland (1986). For our current illustrative purposes, the basic idea is to have different units represent different *triples* of letters, and to represent the presence of a word by lighting up the units for the triples of consecutive letters in the word. For the word 'THAT' we would have high activity at the units for the triples '–TH', 'THA', 'HAT' and 'AT–', where a dash indicates the absence of a letter. (A set of such triples does not necessarily represent a word unambiguously — see Pinker & Prince (1988, pp.96-97) — but we will ignore this complication in the following comments.) There would be word units as before, but in place of the basic letter/word connections we would have, analogously, triple-to-word connections. The abstract associations represented by the system on presentation with a particular word would be overlap relationships between successive triples, such as between 'THA' and 'HAT'. Again, we note that the abstract associations between triples in the presented word are entirely encoded in the activity levels of the triple units, given that they are linked to the word units in a particular way.

The positional encodings used in our hypothetical letter-based and triple-based systems are *absolute* positional encodings. The reason we say this is best brought by an example of a *relative* positional encoding of abstract associations. The (quasi-)connectionist sentence-parsing system CONPARSE of Charniak & Santos (1987) uses such an encoding. The model is centered on a two dimensional array of registers, which could in principle be implemented as connectionist subnetworks. See Fig. 1, where each square illustrates a register.

FIGURE 1 ABOUT HERE

By putting the registers into suitable states representing syntactical categories, the array can hold parse trees. The tree structure is encoded partially by the relative positioning of states in registers. Different directions in the array have different representational significance. The up-down dimension is for representing constituency. If a register in the array denotes a particular constituent found in the sentence, e.g. the particular verb phrase "ate the cheese", then the register immediately below it represents a sub-constituent of it, e.g. the noun-phrase "the cheese". On the other hand, directions involving a left-right aspect are for representing identity of constituents. Thus, if a register L in one column represents a particular constituent, then it *may* be that a register R in the column immediately to the right of it denotes the very same constituent. If so, a special, so-called "binding unit" in L has as its value the row number (up-down position) of R. Such settings of binding units are illustrated in the figure by means of arrows.

If we ignore this left-right dimension, we have a straightforward "relative position encoding" of constituency in the up-down dimension. The way that CONPARSE differs from the previous example systems is that the absolute positions of constituent representations in the array are insignificant (from the point of view of the parse tree structure itself — but the absolute positioning does reflect factors such as how far through the sentence the parsing has got). No register permanently represents any particular thing, in contrast to the letter units and triple units above, although of course a CONPARSE register does denote a particular thing *at any given moment*.[1] *All* that needs to be done to represent the fact that one constituent, A, is a direct sub-constituent of another, B, is to ensure that a representation of A is in a register immediately below a register containing a representation of B.

[1] Actually, a register can denote several constituents with varying degrees of confidence, but this complication does not significantly affect the present discussion.

Despite this very significant difference from absolute positional systems, we still have the notion of a "position" as being a matter of the way in which something (a register in the present case) is connected into the rest of the system. The array-ness of the set of registers is of course just a conceptual view, and is not to be taken as reflecting any physical notion of position. The array-ness is reflected in the total network connectivity as the fact that neighboring registers are connected in *special* ways to each other.

Although the actual and hypothetical systems referred to in this subsection have a localist flavor, in that some particular nodes are dedicated to denoting particular things (only temporarily, in the case of CONPARSE), the discussion can be extended to distributed systems as well. If, for instance, letter instances had a distributed representation rather than a localist representation by means of the individual units like T_1, then we would modify our observations by talking about the way that sets of nodes, rather than individual nodes, are connected into the rest of the system.

Path-Marking Techniques

Path-marking methods appear in many different detailed forms in existing connectionist systems. The basic idea is that if a system contains subnetworks A and B representing entities a and b respectively in the domain of discourse, then an abstract association between a and b could be represented by temporarily marking one or more connection paths between A and B in some way. The two main ways this can be done are (i) to change one or more connection weights on each path, and (ii) to change the activation level of one or more intermediate nodes on each path.

Connections allowing fast, short-term weight change have occasionally been suggested (see e.g. Goddard 1980, Hinton & Plaut 1987 and references therein), as has the closely related notion of "multiplicative" connections (Hinton 1981, McClelland 1986, Pollack 1987). However, for the purpose of representing temporary associations in complex information structures, path marking method (ii) is more common than the weight-change method. It is also more powerful, since activation levels of binder nodes can be transmitted from place to place (thus facilitating operations such as the copying of complex temporary structures), whereas in most views of connectionism weights as such cannot be transmitted. Of course, if a weight is maintained at some level as a result of a signal continuously impinging on the connection from some node N, then N's value could be transmitted elsewhere; however, the method is then tantamount to method (ii) for the purposes of the discussion in this paper. Also, a large proportion of what we will be saying about method (ii) applies also in modified form to method (i). Hence, for the sake of brevity we will concentrate on method (ii) in this paper. (Some discussion of method (i) can be found in Barnden 1984b).

In method (ii) the intermediate nodes are typically called "binder", "binding" or "bind" units or nodes. We will uniformly use the term "binding node". A clear example of the technique appears in Feldman (1982). A more elaborate form of the idea appears also in the BoltzCONS system of Touretzky (1986) and the distributed connectionist production system (DCPS) of Touretzky & Hinton (1988). Here a production rule can contain variables, as in the following example:

$$(=\mathrm{x\ A\ B})\ (=\mathrm{x\ C\ D}) \Longrightarrow +(=\mathrm{x\ E\ F})\ +(\mathrm{P\ D\ Q})\ -(=\mathrm{x\ S\ T}).$$

=x is the variable here. The rule is enabled if, for some symbol λ the system contains the representation of a triple (λ A B) in a subnetwork called the C1 clause space and the representation of a triple (λ C D) in a subnetwork called the C2 clause space. When this happens the variable =x is, abstractly speaking, bound to the symbol λ. The binding is realized in the "bind space", which is a set of interconnected binding nodes which "coarsely" encode bindings. Specifically, each binding node is thought of as representing the binding of =x to one of a set of three symbols. A binding of =x to a unique symbol λ is represented when about 40 binding nodes, all of which include λ in their symbol-sets, are active. Now, the representation of triples in the two clause spaces is also coarse, so any given triple is represented in a clause space by activity at about 28 different nodes, each of which includes the triple in question in its "receptive field" of triples. For each clause space, some of the nodes that include in their receptive fields a triple starting with the symbol λ are excitatorily connected to the λ clique of binding nodes in bind space. Thus, the binding nodes can be thought of as devices for the marking of connection paths between distributed representations of symbol triples in the clause spaces. We note in this system that not only are the representations that are bound together distributed, but the representation of the binding, as a *set* of binding nodes none of which encodes a unique binding, is itself distributed.

We see this same possibility of distributed binding in Smolensky's tensor-product framework for role/filler binding (Smolensky 1987). In one type of application of this framework there is a tuple of filler units and a tuple of role units. A particular role (of some concept or frame) is represented by a particular activation vector over the role units. Similarly, a particular filler (for the role) is represented by a particular activation vector over the filler units. Several fillers and roles might be simultaneously represented. There is also a binding node b_{ij} for each pair (r_i, f_j) where r_i is a role unit and f_j a filler unit. Node b_{ij} is connected to both r_i and f_j. If the filler tuple and the role tuple are representing just one filler and role respectively, then the activity of b_{ij} is the product of the *ith* component of the role and the *jth* component of the filler. Otherwise, the activity of b_{ij} is the sum of such products. At one level, b_{ij} represents a degree of binding between a role *unit* and a filler *unit*. At the level of the representations, on the other hand, a binding between a *role* and a *filler* is represented in a possibly distributed way as activity across the set of binding nodes.

As stated in the previous subsection, the CONPARSE system of Charniak & Santos uses "binding units" as an adjunct to its relative positional encoding of constituency. The numerical value of the binding unit in a register indicates which register in the next column to the right is to be taken as temporarily associated with it. The binding unit so described does not immediately fit the concept of binding nodes we have been using. However, it is easy to imagine connectionist implementations of CONPARSE in which binding units are mapped to binding nodes in our sense. For example, if there are n rows in the array, each binding unit could be implemented as a tuple of n binding nodes, with the integer value r of the binding unit being implemented as high activity at the *rth* binding node and zero activity at the others. Because of the dynamic interactions between registers that are required during parsing, each binding node would have to be connected to units within the register to which it "points". Therefore, each binding node would be a vehicle for marking one or more connection paths between its home register and the register it points to.

To return to the hypothetical word-recognizing systems discussed in the subsection on positional encodings, the word units themselves can be said to be binding units, supplying encodings of the abstract associations among the letter instances or triples in the words. High activity at the 'THAT' word unit can be cast as *one* representation the system has for the presence of certain triples or letter instances and certain abstract associations among them. Since a word unit is connected to the triple units or letter instance units that are appropriate for the word represented by the unit, there is a case for viewing it as a binding unit sitting on paths joining those units. We will consider this possibility further in Section 6.

A High-Level Associative Technique

Under this heading we will discuss only the technique for representing symbol-triples, and associative linkages among them, in BoltzCONS (Touretzky 1986). As stated above, triples are represented within a space (actually, more than one space) of nodes, in which each node is deemed to coarsely represent many triples. A given triple is unambiguously and fully represented when the 28 or so nodes that represent that triple are all highly active. Notice that there is no separate representations of the three parts of a triple; essentially, the system is a distributed "conjunctively coded" one that is not parasitic on a more low level representation of the symbols conjoined in triples.

Triples are thought of as being (bidirectionally) linked by virtue of *symbol sharing*. For example, suppose the space is representing the triples (T U V), (U A B) and (V C D). Regard the first symbol in each triple as a name ("tag") for the triple, under the assumption that no two co-active triples start with the same symbol. Then the state of the space can be thought of as consisting of a root triple (the first one) that points to other triples by virtue of including their names, U and V. In fact, triples can be thought of as Lisp-like "cons cells" (the basic nodes used in the implementation of Lisp list structures on a computer). The BoltzCONS system contains mechanisms such that, if it has access a certain sense to the triple (T U V), it can gain access to (U A B) or (V C D), and vice versa. The ability to traverse the "links" between triples in either direction is an important feature of BoltzCONS.

The BoltzCONS associative linkage technique is cast as "high level" because it is naturally described at the level of triples themselves, rather than at the connectionist network level. There is no *straightforward* description of the technique at the latter level.

Reduced Descriptions

A number of connectionist systems are interesting in that they use a "reduced description" (or "reduced representation") technique for assisting in the representation of complex information structures. To abstract away from the details of various versions of it that have appeared — Hinton (1988), Touretzky & Geva (1987), Sumida & Dyer (1989)[2] — we can give the following general, idealized account. Suppose there is some transformational mechanism T that takes as input an existing complex data structure (description, representation), realized as a possibly large pattern of activity D, and produces from it a considerably smaller activity pattern d. Let us call d the reduced description or reduced representation produced from D. Suppose also there is an inverse transformational mechanism t that reconstructs, retrieves or accesses D from d. Suppose that it is required to use the data structure represented by D as a component in a logically larger data structure. Then the activity pattern DD for the larger structure can be built using the reduced pattern d as a "representative" or "ambassador" of D, rather than using D itself. One simple option is for DD to be the concatenation of d with the reduced descriptions for other parts. The pattern DD can in turn be reduced by T if it is required to have the data structure it represents be part of a yet larger data structure.

Different systems may differ on how DD is formed from the reduced descriptions d, and therefore differ on their overall handling of temporary associations. However, the use of reduced descriptions is an important component of the overall handling, since it allows the size of the activity patterns being put into temporary association to be standardized. This has certain benefits in terms of the circuitry needed to effect and use the associations.

Signatures

Lange & Dyer (1989) and Lange & Dyer (in press) describe ROBIN, a connectionist system capable of performing high level inferences, with powerful capabilities for temporary association. A special node called a *signature node* is attached to each concept-representing node. Also, each role (slot) of a frame-like concept in ROBIN has an attached *binding node*. A signature is an activation value or pattern emitted by the signature node attached to a concept node. Different concepts have

[2] The Recursive Auto Associative Memory scheme of Pollack (1988) for encoding stacks and trees can also be cast as using reduced descriptions.

different signatures. Temporary associations are achieved between a concept and a role of another concept when the binding node for the role is supporting the concept's signature as its activation pattern. ROBIN performs inference by a combination of the spreading of ordinary activation over the knowledge-level semantic network containing the concept nodes and the spreading of signatures over another control network parallel to the semantic network.

A Time-Phase Technique

An interesting way of dealing with temporary associations is to implement them in the time dimension. Clossman (1987) and Shastri & Ajjanagadde (1989) present schemes of this sort. In the latter, "constant" nodes representing objects (e.g. John and Mary) in the domain of discourse, and nodes acting as the argument positions of a predicate, are *phase-sensitive* binary units that become active in the phase i of every clock cycle once activated in the *ith* phase of a clock cycle. An argument node is considered to be bound to a constant node simply by virtue of being active at the same phase of each clock cycle. Therefore, the representation of the proposition that "John loves Mary" would involve (amongst other things) two argument nodes, one for the "lover" argument and one for the "lovee" argument, where the lover argument node has the same phase as the John node, and the lovee argument node has the same phase as the Mary node.

3. INFORMATION STRUCTURING IN COMPUTERS

Here we look briefly at the three basic ways in which information is structured in computer memories: *sequential allocation*, *pointers* and *associative addressing*. It is convenient to consider the *hashing* technique here as well, although it is a less basic technique and rests on the others. Readers wishing for further information may turn to, say, Hwang & Briggs (1984) for associative addressing and to an introductory data structures text (e.g. Standish 1980) for the others.

Although much of what we say in this section will of course be familiar to many if not all readers, there are certain observations and perspectives that may yet be a little unfamiliar and that are important in the context of the paper as a whole.

Sequential Allocation

The "sequential allocation" technique can be found to some extent in virtually all data structures as implemented on existing computers.[3] One form in which it commonly appears is that an abstract body of information, containing information about, say, a particular employee in a company, could be realized in computer memory as some "concrete record" — a group of *consecutive* cells in primary memory. In saying this we rely on the idea that the memory consists of a linear sequence of cells, so that it makes sense to talk about the (absolute and) relative *positions* of cells in memory. Indeed, cells are conventionally viewed as being consecutively numbered, starting at 0, say. Let the lowest-numbered cell in our concrete record have number (i.e. address) b, the rest being numbered $b+1$, $b+2$, etc. One simple scheme for mapping the information about the employee into the concrete record might be to have the employee's identification number in cell b, his/her name in cell $b+1$, his/her salary in cell $b+2$, etc. However, more complex schemes are possible: for instance, two or more such pieces ("fields") of information might fit into one cell, or one field might have to be spread over several cells. We will suppress these complications in what follows, but what we say could easily be adjusted to take account of them.

[3] The adjective "sequential" is to do with the linear nature of computer memory, and *not* with sequentiality of processing.

An important point for our purposes is that the significance of any given cell in a concrete record is determined, at least in part, by its *relative position* with respect to the base cell b. Notice here that the absolute position of the whole record in memory will typically be insignificant, and crucially so. For instance, it may be required to move/copy a record around in memory in order to have it take part in various larger data structures. It is then assumed that a given piece of program can access and modify the information in the record merely by knowing the base address b and the scheme whereby information is packed into the record, where that scheme is independent of b itself.

A particular form of a concrete record is the sequentially allocated array — that is, an array (of any dimension) implemented in the familiar way in consecutive memory locations starting at some base address b. In the simplest case, a one-dimensional array is implemented so as to make the first array element's value be in cell b, its second element's value be in cell $b+1$ and so forth. Once again, the absolute location in memory of the whole concrete record is not significant, as long as any piece of program that needs to use the array knows the base address b. The crucial point, therefore, is the relative positioning of the array values with respect to the base cell. Also, the relative positioning of cells within the record may have a more general significance, in that neighboring elements in the (one-dimensional) array map to neighboring cells in the record. (A somewhat more complex relationship holds for higher-dimensional arrays).

We have so far described a pure form of sequential allocation, in that there is nothing *within* any given cell in the record to say what its significance is — only its displacement from b does that. A less pure form is to allow the fields to occur in any order starting at b, though still occupying consecutive cells, and to have in some part of each cell some information, such as a field name or array subscript, specifying the cell's exact significance. Such a scheme can be useful if concrete records are able to miss fields out, either because fields are sometimes not applicable (e.g. spouse name in the case of an under-age employee) or because if a field is missing it is to be taken as having a default value. However, in an impure form of sequential allocation like this the notion of relative position is still crucial: the represented fields for the concrete record based at b are still bunched in consecutive cells starting at b — it is not the case that just *any* cell in memory containing a given field number (say) is part of *that* concrete record, or indeed any concrete record.

Another type of impurity is for the value of one cell to help specify the significance of others. This is found, for instance, in the concrete records implementing the abstract "variant records" of the programming languages Pascal (Jensen & Wirth 1974) and Modula-2 (Wirth 1985). In a concrete record representing a person, a cell in a specific position relative to the base address might say whether the person is a child or an adult. The cells following it would then encode child-relevant fields in one case and adult-relevant fields in the other.

Sequential allocation is also commonly used at higher levels of conceptual organization than assumed in the above examples. For instance, the concrete records for some set of employees might themselves be placed consecutively in memory.

Pointers

The "pointer" technique for information structuring relies on interpreting the contents of some cells as being cell names ("addresses"). For this purpose it is not necessary in principle for addresses to be in any way thought of as numbers, although of course they are so in practice. The use of pointers is almost always parasitic on a non-trivial use of the sequential allocation technique, since what is pointed at is almost always some sort of concrete record containing more than one cell. (The pointer is typically the base address of the concrete record pointed at.) Further, pointers are almost always in particular cells of concrete records. The most familiar example of this is provided by a linked list of concrete records, where, say, the *third* cell in each record always contains the address of the "next" record. Conversely, the use of sequential allocation is almost always parasitic on at least a minimal use of pointers, since a piece of program can only use a concrete record in a conventional computer memory if it knows its base address (i.e. has a pointer to the record).

It is rarely pointed out in discussions of traditional symbolic processing as contrasted with connectionist processing that there are compromises between the pointer technique and the sequential allocation technique. Pointers can be relative as opposed to absolute addresses, so that some base value needs to be added to them to produce absolute addresses. In particular, size fields in different-sized concrete records that are arranged consecutively in memory can act as relative addresses, in that the base address of a given record plus the value in its size field equals the base address of the next record.

Associative Addressing

The third basic type of information structuring in computers is "associative addressing" (also called "content addressing"). It is much less prevalent than sequential allocation and pointers, but is nevertheless used for cache memory in ordinary computers (e.g. the Vax 11/780 — see Stallings 1987, pp.114) as well as for general purposes in some unusual types of computers (usually highly parallel ones).

The basic idea, on which there are many variants, is to use the value in a specific portion (or all) of a memory cell C as a key with which to access other cells D whose corresponding portions contain the same value. The key value obtained from cell C is broadcast in some way to all cells, and these independently and in parallel decide whether they contain that value. If so, they "light up" in some way that is visible to other machinery.

Associative addressing can achieve what absolute or relative pointers do. It also naturally allows bidirectional pointing, since if cell C associates to cell D then (in typical schemes) D also associates to C. Further, associative addressing gets further away from the linear structure of memory, since concrete records that are interlinked by associative addressing can be moved about freely within memory.

In an associative memory, sequential allocation and associative addressing are typically mutually parasitic in much the way sequential allocation and ordinary pointers are. When a value is used to associatively access a given cell, it is likely that the cell is in a concrete record that is of interest. Conversely, a concrete record is likely to have special cells containing associative keys for accessing other records.

Hashing

Hashing is a way of storing items of information in a concrete record — the "hash table" — in such a way that the position in the hash table where a given item is stored can on average be computed very efficiently from the item itself. In one standard application, the items are words (variable identifiers, function names, etc.) in a particular program, and there is a "hashing function" h that applies some transformation to the bit-string representations of the characters in any given word in order to come up with a relative address in the hash table. The word itself together with important information associated with the word (such as a data type if the word is a variable identifier) would be stored in that position in the table. This account is an oversimplification, because there must be some way of dealing with collisions — that is, cases when two different items are mapped by the hashing function to the same address. Several different methods are in common use (see e.g. Standish 1980).

Two features of hashing are important for us here. First, hashing can be applied to items much more complex than character strings, such as board positions in games. or frame structures as used in AI. Second, hashing is in a sense a way of emulating a form of associative addressing in non-associative memories. This is because it enables a program to gain efficient access. on the basis merely of I itself, to the place where information related to I is stored.

4. RELATIVE-POSITION ENCODING and PATTERN-SIMILARITY ASSOCIATION

One promising way to approach high-level cognitive processing in connectionist systems is to encode complex temporary information structures by means of two techniques called "Relative-Position Encoding" (RPE) and "Pattern-Similarity Association" (PSA). These are answers to the Temporary-Association Question presented at the start of Section 2. The present section gives an overview of the general ideas of RPE and PSA. Our own connectionist system's particular forms of RPE and PSA are described in Section 5.

Relative-Position Encoding (RPE)

A simple instantiation of the general idea of relative-position encoding is as follows. Imagine a network that can be viewed as a two-dimensional array of "component subnetworks". See Fig. 2, where each square illustrates a component subnetwork.

FIGURE 2 ABOUT HERE

Assume that there is a set of activity patterns any of which can be temporarily installed in any of the component subnetworks. Consider two *adjacent* component subnetworks h and j. Suppose that at some moment they hold, respectively, an activity pattern HUNGRY representing the idea of being hungry and an activity pattern JOHN representing a particular person, John. Finally. take this "subconfiguration" of adjacent activity patterns to encode the temporary proposition that John is hungry. It could also have been encoded by using any two other adjacent component subnetworks in the array in a similar way. The absolute position of the subconfiguration. as well as the direction of the adjacency within each subconfiguration, is insignificant.

To encode a two-or-more place proposition, such as that John loves Mary, we could basically follow the same approach. We could have activity patterns LOVES, JOHN and MARY in component subnetworks l, j and m respectively, where j and m are adjacent to l (but not necessarily to each other). This is not quite adequate, since we need to be able to distinguish the proposition that John loves Mary from the proposition that Mary loves John. A simple way to ensure this is to let a "Role" part of each component subnetwork be available for containing an activity pattern which signifies the role, if any, that that subnetwork is playing in some proposition. Thus, in the John-loves-Mary proposition, j's Role part would contain an activity pattern signifying "first argument", while m's Role part would contain an activity pattern signifying "second argument". In principle, there is no need to assume that such activity patterns have a fixed meaning in themselves — rather, it could be that the pattern used to specify the first-argument role in LOVES propositions could be the very same pattern used to specify the second-argument role in HATES propositions. This is not to say, however, that there might not be good reasons for giving fixed meaning to the patterns.

Different propositions can be simultaneously maintained in different regions of the array. The propositions can share relationships or arguments — say JOHN — since the JOHN pattern or any other can be simultaneously instantiated in several different component subnetworks at once.

More sophisticated versions of the RPE idea could depart from the simple example in a number of directions. The closest departure would be to have an array whose number of dimensions was not two. A further departure would be to have a network with a non-array structure while still providing a useful notion of relative position. On the other hand, keeping to the idea of an array, it could be more like a TV screen on which patterns (like images on a screen) can appear in any position on a quasi-continuous range, rather than being confined to discrete component subnetworks.[4]

Further, in any array-based version of RPE, whether using discrete component subnetworks or not, one could allow relative-position relationships other than strict adjacency to have associational significance. In particular, we could simply allow, say, the agent-denoting component subnetwork in a love proposition to be two steps away from the one containing the LOVES pattern, rather than requiring it to be closer. More interestingly, different degrees of closeness could conceivably be used to encode different degrees of association. Also, the array could be used analogically as a map of some physical space (or of some domain mathematically cast in terms of a space), so that long-distance relative positional relationships in the array, not just adjacency relationships, would have structural significance. See Barnden (1985, 1987) for suggestions along these lines.

Another departure from the simple RPE described above is for component subnetworks not to contain separate parts (main part and Role part). Rather, a component subnetwork would contain a monolithic activity pattern. If the subnetwork was to be made to contain a representation of John as first argument of a loving proposition, the system would insert a John-as-first-argument pattern. If on the other hand the subnetwork was to be made to contain a representation of John as second argument of a loving proposition, the system would insert a John-as-second-argument pattern. For the purposes of inference there would have to be some way of detecting that these two patterns represent the same person. For instance, if the system is to reason from the premises that A loves B and B loves C (where C is different from A) to the conclusion that A is jealous of C, then it must recognize that the same person B appears appropriately in two loving propositions.

RPE in any of its forms, but most clearly when based on an array of discrete component subnetworks, can be viewed as an extrapolation from the sequential allocation technique in computers. More exactly, it is an extrapolation from the "impure" sequential allocation technique described in Section 3, under which the structuring of a concrete record is indicated partly by special tags in cells. These tags are very like the Role activity patterns proposed above for RPE.

The (quasi-)connectionist system, CONPARSE, of Charniak & Santos that was mentioned in Section 2 clearly uses a form of RPE, in that much of the structuring of a parse tree is embodied in the relative positional relationships of cell states. In fact, as far as we are aware, amongst systems other than our own this one comes closest to fitting the above general description of RPE.

In principle, none of the above comments imply that the array structure of the pool of component subnetworks be reflected in any particular *physical* arrangement of the subnetworks. "Adjacent" subnetworks need not be physically adjacent in any hardware or biological realization of the system. However, if the network does have a physical embodiment, then there will be consequences for the physical layout of the array. For instance, one operation that is likely to be

[4] In early work leading to our own system, Conposit, arrays like this were adopted (Barnden 1984a). These were later replaced (Barnden 1985) by arrays of discrete component subnetworks so as to allow a system that could be more easily simulated and analyzed. Also, a mechanism for processing subconfigurations is much easier to devise for the discrete case. However, it is possible that there is ultimately more future in a "TV-screen" version, especially from the point of view of brain theory. This shows one respect in which Conposit is to be viewed as heuristic guidance for the development of future, more sophisticated systems.

required is the traversal of an association. Suppose the system is in some sense attending to one component subnetwork N — so perhaps some special "degree-of-attention" units within N have a particularly high level of activation. One type of traversal operation then consists in switching attention to the neighboring component subnetwork that has some given activity pattern in its Role part. Presumably, there must be circuitry that allows the activity on the degree-of-attention units in N to spread to neighboring subnetworks. The neighbor that has a suitable Role pattern then lights up its degree-of-attention units. Thus, each component subnetwork must have special connections, whether direct or indirect, to its *neighbors*— special in the sense that they are different in function from any other connections that might exist between subnetworks. Thus, the overall special connectivity between neighbors is the network realization of the notional array structure. Moreover, since the traversal operation mentioned should be as efficient as possible, it could well follow that, other things being equal, it was advantageous to have *notionally* adjacent component subnetworks be *spatially* adjacent as well, assuming the system has a physical embodiment. This naturally suggests a spatial array as a reasonable and motivated embodiment of the notional array structure.

Pattern-Similarity Association (PSA)

A simple illustration of the general PSA idea is as follows. We assume as before that there is a pool of discrete "component subnetworks", but now the pool is just regarded as a set, and does not need to have any overall structuring, adjacency relationships, etc. Imagine two component subnetworks *h* and *j*, each divided into two parts called the Information Part and the Associator Part. See Fig. 3.

FIGURE 3 ABOUT HERE

The temporary proposition that John is hungry could be encoded by letting the Information Parts of *j* and *h* contain activity patterns JOHN and HUNGRY representing John and the property of being hungry, respectively, and letting the Associator Parts of *h* and *j* contain the same activity pattern X. The component subnetworks *h* and *j* are deemed to be temporarily associated by virtue of this equality of their "associator" patterns (X).

Clearly, this example is a simple extrapolation from a possible use of associative addressing in a computer. For the latter, there would be two-cell concrete records instead of subnetworks *h* and *j*. In each record, the first cell (say) would be called the Information cell and the second would be called the Associator cell. Bit-strings would correspond to the connectionist activity patterns. Some bit-strings would represent JOHN, the HUNGRY property, and so on, and would go into Information cells, while some would be reserved to act as associative addressing keys, and would go into Associator cells.

Of course, we do not assume in general that *h* (or similarly *j*) is restricted to containing the patterns HUNGRY and X. On another occasion it might contain TIRED and Y instead. Conversely, it should be clear that we do not assume that the HUNGRY and JOHN patterns, etc., can *only* go into the particular subnetworks *h* and *j*. By default, we assume that any Information part can contain any of the information patterns, and any Associator parts can contain any of the associator patterns. It does not matter, logically, which component subnetworks are used — all that matters is that the right temporary associations between subnetworks be embodied in the assignment of activity patterns to subnetworks. Nevertheless, one can imagine a type of PSA in which a component subnetwork could be reserved for use in some particular class of propositions (e.g. propositions about personal relationships).

We need to be able to disambiguate the roles in a two-or-more place proposition, such as that John loves Mary. We can proceed analogously as in the case of RPE, introducing a Role part into each component subnetwork, as well as an Information part and Associator part. For John loving Mary, we then have component subnetworks *l*, *j* and *m*, where *l* is analogous to *h* and has the LOVES pattern in its Information part, *j* is as before, and *m* is like *j* but with the MARY pattern in its Information part. All three subnetworks have the same associator pattern X. Each subnetwork's Role part contains an activity pattern specifying its role: relationship, first argument or second argument. As in the case of the role patterns discussed above for RPE, there is no need to assume that they have a fixed meaning in themselves.

Different propositions can be simultaneously maintained if they use different associator patterns. In essence, the associator pattern, X or whatever, used in a particular proposition can be regarded as temporarily standing for the proposition as a whole. The propositions can share relationships or arguments — say JOHN — since the JOHN pattern can be simultaneously instantiated in several different Information parts.

In more sophisticated versions of the PSA idea, the required similarity of associator patterns would not have to be equality. One might even seek to encode strength of temporary association in the degree of similarity of associator patterns. One does not need to assume that a given part of a component subnetwork always has to hold associator patterns: for instance, it could hold associator patterns on some occasions and information patterns on others. More radically, we might be able to get away from the idea of having separate parts in the component subnetworks, and have a more distributed representation within each subnetwork. There would be a single pattern in each subnetwork, and association would be incorporated into the notion of similarity among such patterns. Consider the propositions "John is hungry" and "John loves Mary". For the former, an *h* subnetwork and a j_1 subnetwork would contain patterns that are similar in some way which we can call X, whereas for the latter proposition an *l*, a j_2 and an *m* subnetwork would contain patterns that are similar in some way Y. Further, the j_1 and j_2 patterns would be similar in yet a third way we can call JOHN, and also in a fourth way that we can call "first argument". That is, the associator, information and role patterns that we had originally have been replaced by respects in which relatively monolithic patterns are similar. However, we have not investigated this possibility in detail.

Dependencies between RPE and PSA

In a PSA scheme in which component subnetworks have separate parts, the parts within a given component subnetwork need not in principle be physically arranged in any particular way (given that the system has a physical embodiment in the first place). For instance, they need not occupy separate spatial regions of the subnetwork — they merely need to have disjoint sets of nodes. Of course, it is still open to us to regard the parts in the subnetwork as being *notionally* arranged in some way that we think of as spatial — e.g. as a linear sequence. (Also, a notional spatial arrangement may be reflected in the structure of interconnections among the parts.) Now, whether or not we think of the parts as being spatially arranged, there is a sense in which a relative positional technique is in play. This is because the significance of a part P of some component subnetwork N at any moment is best taken as a function of which set of nodes P is *within some component subnetwork N*, and of the significance of N.

Thus, there is a sense in which a local type of RPE is in operation within each component subnetwork. This is analogous to the point that typical uses of associative addressing in computers exploit the relative positional arrangements within concrete records.

Conversely, there are opportunities for RPE to be partially dependent on PSA. The reader may have wondered how the described simple form of RPE using a discrete 2D array could deal with propositions that have more than eight arguments, since the component subnetwork representing the relationship aspect of the proposition must be adjacent to the subnetworks representing the arguments. A possible solution is to divide the proposition up into two or more subconfigurations, and to include in each one a component subnetwork that (temporarily) acts as an Associator subnetwork. Each of these temporary Associator subnetworks would contain the same main activity pattern X, and their Role parts would contain an activity pattern saying that they are Associator subnetworks. In this way, different groups of arguments of a single proposition could be realized in different subconfigurations.

An interesting aspect of a scheme of this sort is that, viewed from the PSA perspective, the component subnetworks are *subconfigurations of* the components of the array, and these subconfigurations only have a temporary existence; moreover, an Associator part is designated as such by virtue of a special subpattern in an array component. Thus, the form of PSA in use is more labile and sophisticated than the basic sort described in the previous subsection. We also observe that each "component subnetwork" from the PSA perspective uses a straightforward form of RPE internally. As we are about to see in the next section, Conposit rests on such a mixture of RPE and PSA.

We close this section by observing that, although RPE and PSA have been presented as if the component subnetworks are localized, separate subnetworks, they could in principle have a more distributed nature, in the sense that their node sets could overlap to significant degrees.

5. CONPOSIT

Conposit is a connectionist rule-based system that exists in several different simulated versions. It rests on a particular, intimate mixture of simple forms of Relative-Position Encoding and Pattern-Similarity Association, similar but not identical to the forms used for illustration in the previous section. Our focus will be on Conposit's use of RPE and PSA, and the reader should refer to Barnden (1988, 1989, 1990) for the fine detail of other aspects of Conposit. Conposit's forms of RPE and PSA give it the same sort of data structuring power and flexibility that sequential allocation, pointers and associative addressing confer on existing computers.

The overall structure of simulated versions of Conposit is shown in Fig. 4, the main flow of information being indicated by the arrows.

FIGURE 4 ABOUT HERE

The simulated versions of Conposit are *entirely* concerned with *short-term* data structures and manipulations. There is *no* long-term memory of data, except insofar as the production rules can be viewed as holding long-term information. Equally, the versions perform no learning, and involve no weight changing. The structure of the system as implemented as a connectionist network is completely and intricately hand-crafted.

The productions or rules are shown in the figure as being split into condition parts, bundled up in the Subconfiguration Detection Module, and action parts, bundled up in another connectionist subnetwork called the Rule Action Parts Module. Through the Subconfiguration Detection Module, the rules detect relevant forms of short-term data structure held in the short-term memory (working memory). This memory consists of a two-dimensional array of component subnetworks, to use the terminology of the section above on relative-position encoding. In Conposit, the component subnetworks are called "(active) registers" and the array is called the "Configuration Matrix" (CM).

The rules are hardwired as permanent subnetworks of the system. The system operates in a cyclic way. In each cycle, the presence of data structures in the working memory causes one rule to fire, changing the contents of working memory. Then another cycle starts. A typical rule is one that can be paraphrased as: "if A loves B and B loves C, where C is not A, then assert that A is jealous of C". Note that this involves three variables that must be bound on any give firing of the rule, and that the condition part of the rule imposes more than one constraint on each variable. (Conposit's variable-binding facility is only partially described below, but more fully in Barnden 1990.) Complex rules are also to be found in a version of Conposit (Barnden 1989) that performs syllogistic reasoning on the basis of Johnson-Laird's mental-model theory (Johnson-Laird 1983).

A rule's action part effects a series of actions, generated by a flowchart-like subnetwork that can include branching and loops. An action can be, effectively, the calling of another such subnetwork as a routine. However, such calling cannot be directly or mutually recursive. This is not a restriction on the logical power of the system, since, if necessary, the stacks necessary to get the effect of recursion could be implemented in working memory. As regards the traversal of recursive data structures such as trees in working memory, this can be readily achieved in Conposit. In fact, such traversal is aided by the bidirectionality of RPE-based and PSA-based linkages. This is similar to the advantage conferred by bidirectionality of linkage in BoltzCONS, namely that the use of explicit stacks for traversal housekeeping is avoided. One version of Conposit does the same type of natural language parse tree manipulations that Touretzky (1986) reports BoltzCONS as doing.

The rules change the contents of the CM by sending "command signals" to it. These are moderately complex messages, each one consisting of a vector whose components are either binary or ternary valued. The vector travels over a "cable" of connections from some node within a rule subnetwork to the CM. For reasons of space, we omit any further description of the Subconfiguration-Detection and Action-Parts modules, although they do throw further light on the nature of RPE and PSA in Conposit (Barnden 1988, 1990).

Two deficiencies of Conposit are the massive size of the Subconfiguration Detection Module, which contains numerous subcomponents each isomorphic to the Configuration Matrix, and the sequentiality of the rule-based processing (although, as pointed out in Barnden 1990, Conposit provides opportunities for limited within-rule parallelism). These and other drawbacks are being corrected in the development of a case-based reasoning (or analogy-based reasoning) version of Conposit, a preliminary version of which is described in Barnden & Srinivas (to appear). However, the new version uses forms of RPE and PSA almost identical to those used in the rule-based versions.

Much of the interest of Conposit lies at an intermediate, "register-machine" level of description that will be adopted in most of this section. The mapping to the lower, connectionist network level is for the most part straightforward. As for the one respect in which it is not straightforward (a so-called "temporal-winner-take-all" selection mechanism sketched below), the details are not highly germane to the present paper, and can be found in Barnden, Srinivas & Dharmavaratha (1990).

The Configuration Matrix: Register-Machine Level

The CM is a two-dimensional (32×32) array of "active registers" (playing the role of "component subnetworks" in the general description of RPE). Each register is connected to all its neighbors, and also to a module called the CM's "Parallel Distributor". One of the purposes of the parallel distributor is to allow indirect communication between any two registers. However, no register knows the identity of any other register except for its neighbors, and non-neighborly communication is based purely on the *broadcasting* of register states to all other registers via the parallel distributor.

The state of a register at any moment has two parts, called its current "symbol" and its current "highlighting state". At the register-machine level, a symbol is a member of a large set of symbols, which are just unanalyzed, atomic objects of some sort. The highlighting state, at the register-machine level, is a collection of ON or OFF values, one for each of a set of "highlighting flags". We refer to the flags by names of various sorts, which we introduce below as needed. Any combination of ON or OFF values is allowable across the highlighting flags of a register. All registers have the same, fixed set of highlighting flags, though of course they generally differ on the ON/OFF values they have for them. Different versions of Conposit have had different numbers of highlighting flags, up to a maximum of 21. One of the uses of highlighting — to act as the role-specifiers needed by RPE — will be detailed in a later subsection. For now we concentrate on the symbols.

Any symbol can be placed in any register, by the action of a rule or as part of the initialization of a run of the system. Many symbols have a specific, *constant* representational function, such as denoting a particular person (e.g. John), or a particular set of situations (e.g. the set of conceivable situations in which someone is hungry, or the set of conceivable situations in which one person loves another). The symbols are very much like the individual constants in a logic-based representation scheme that has a situation-based ontology. There is also a "null" symbol that does not denote anything, and a set of "unassigned" symbols that are very like the variables in a logic-based representation scheme. The unassigned symbols are what play the part of the "associator patterns" in Conposit's version of PSA.

If a register contains a constant, non-null symbol s at some moment, then the register is deemed to be denoting *at that moment* whatever it is that the symbol denotes. Thus, we say that a register containing the JOHN symbol denotes a particular person John (temporarily). As for a register containing an unassigned symbol, it acquires a temporary denotation by virtue of the symbols and highlighting in adjacent registers. We may also view the unassigned symbol itself as temporarily picking up that denotation, if we wish. We defer the details for a little while.

Configuration Matrix: Connectionist Network Level

Each register in the CM is implemented as a small connectionist subnetwork, and each of these subnetworks is connected to each of its neighboring subnetworks (eight of them, except at the edges). As subnetworks, the registers are all isomorphic to each other, except for differences occurring at the edges. A symbol is implemented as an activation pattern that can be imposed on a certain subnetwork in any register. In the current version of the connectionist implementation, the activation pattern for a symbol takes the simple form of a vector of on/off values on a set of units. The size of this vector is a parameter of the model, but we typically take it to be ten. The arbitrary positioning of symbols within the CM is possible because of the network isomorphism between registers. The highlighting flags are represented in each register by means of a tuple of units, each of which can sustain either one level of activity called OFF or another level called ON.

As well as the circuitry for merely holding the register state, a register also contains circuitry for interpreting command signals received from the action parts of rules. The interpretation typically involves the register sensing its own highlighting state and the highlighting states of its neighbors, and broadcasting its symbol to all registers, via the parallel distributor.

Traditional Symbolic Data Structures in the CM

Conposit is an exercise in "implementational connectionism" — connectionism used as an implementation tool for the manipulation of complex symbolic data structures very similar to those used in traditional AI programs. Our justifications for considering implementational connectionism to be a valid research strategy are beyond the scope of this paper, but are set out in Barnden (1988, 1990). We should emphasize that the potential usefulness of RPE and PSA is not confined to system developed according to the strategy.

The way Conposit realizes complex symbolic structures is illustrated by Fig. 5. The figure shows a possible state of an 8×8 region within the CM, containing a realization of a traditional symbolic representation of the proposition that *Bill hopes that John loves Mary.*

FIGURE 5 ABOUT HERE

Each square in the figure illustrates a register. The following shows what the various items in a square in the Figure signify as regards the current state of the register illustrated by the square:

capitalized word or letter	:	occurrence of a (non-null) symbol
∇	:	ON value of "white" highlighting flag
•	:	ON value of "black" highlighting flag
r, *g*	:	ON value of "red", "green" flags resp.

A square not showing any symbol illustrates a register containing a special "null" symbol. If a square does not show an ON value for a highlighting flag, then that flag is OFF in the register. When, say, the black highlighting flag is ON in a register, we say that the register is highlighted in black, or simply that it is black.

The constant symbol LOVES permanently denotes the class of all conceivable situations in which one person loves another. Now, any white-highlighted register adjacent to a black-highlighted register containing a symbol denoting a *class* of things is deemed to denote, temporarily, a *member* of the class. Thus, in the example, the white register next to the black register containing the LOVES symbol denotes some loving situation or other, currently. Further, any red register adjacent to a white register denoting a loving situation is deemed to denote the agent of the loving; similarly for green highlighting and the object of the loving. Thus, the top-right "subconfiguration" of register states centered on the mentioned white register encodes the proposition that John loves Mary. We call the white register the "head" register of the proposition. The encoding of the structure of the John-loves-Mary proposition in the Figure relies only on the *undirected adjacency* relationships between registers in the CM, as refined by the use of highlighting of registers to differentiate their roles with respect to one another.

We see here that a crucial form of temporary abstract structuring in Conposit is realized through RPE. In the terms of our general description of this technique in Section 4, the registers are the "component subnetworks", the symbol in a register is the content of the register's "main part", and the highlighting state is the content of its "Role part".

The other subconfiguration in the Figure can be interpreted in a way analogous to the first one. The white register in it denotes some hoping situation, with agent Bill. The object (patient) of the hoping is denoted by the green register adjacent to the white register. This register contains the "unassigned" symbol X, which also appears in the white register denoting the situation of John loving Mary. Because X appears in the latter register, we deem it to be temporarily representing that loving situation. We also deem *any* register containing X to be temporarily denoting whatever it is that X denotes. Hence, the green register in the "hopes" subconfiguration denotes the situation of John loving Mary, and altogether we have an encoding of Bill's hoping that John loves Mary.

Clearly, what we have here is a version of the PSA technique, since the hoping subconfiguration is associated with the love subconfiguration through the use of a shared, unassigned symbol X, and a symbol is an activity pattern. The unassigned symbol is acting as the "associator" pattern. The type of similarity required for an association to be deemed to be present is equality (to within the resolution of which activity-pattern comparators are capable). In the terms of our general description of PSA, the simplest view is to take the association to be between two individual registers (the ones containing X), in which case the "component subnetworks" from the point of view of PSA are just the registers. However, it is often intuitively reasonable to take the association to be between the temporary subconfigurations in which those registers lie, in which case the "component subnetworks" from the point of view of PSA are the subconfigurations. We met this idea of component subnetworks being subconfigurations in Section 4, although the subconfigurations in Conposit differ in detail from the ones assumed in that earlier discussion. The love-proposition subconfiguration, for instance, is best regarded as having several Information parts, namely the registers other than the one containing X. That register is of course the Associator part. There is no separate role part. Rather, the significance of the subconfiguration with respect to others is implicit in the local structure within each of those subconfigurations. That local structure is encoded by means of RPE.

Conposit's PSA can also be deployed to encode propositions that have more than seven participants (a white register denoting a situation as in Fig. 5 can only have seven neighbors other than the black, class-denoting one.) The sharing of unassigned symbols can be used to split up the representation of a single simple proposition (situation) into several subconfigurations. For the sake of illustration, a split up of a loving situation is shown in Fig. 6, although normally a proposition with fewer than seven arguments would be represented in Conposit by means of single subconfiguration. A rule that creates such a proposition would only split it up if forced to by the absence of enough contiguous free space in the CM for a single subconfiguration to be used.

FIGURE 6 ABOUT HERE

Just as before, the white register containing X temporarily denotes the loving situation; and then so does every other register containing X, by the symbol-sharing semantic rule. The interpretation of the red and green registers is then *exactly* as it was in the case of the loving situation in Fig. 5.

With the split up as shown, we have a mixture of RPE and PSA much like that discussed at the end of Section 4. Each subconfiguration, or "component subnetwork" from the point of view of the PSA in this mixture, is a pair of registers. The one containing X is again the Associator part. The Information part, from the point of view of Section 4, is the the symbol part of the other register. There is also a Role part consisting of the highlighting part of that register.

It should be clear that Conposit can encode more deeply nested propositions, such as that Tom believes that Bill hopes that John loves Mary. The only limitations are ones of resources: the size of the CM and the number of unassigned symbols available (current simulations provide for thirty). Also, the techniques presented can also be used to realize the equivalent of logical formulae involving connectives and quantifiers. This matter is detailed in Barnden (1988, 1990).

When a rule creates a new data structure in the CM, it must find enough "free space" for it. If there is not enough space, the rule simply fails to do what it should be doing. Conposit as it currently stands does not try to avoid such failures or take special action when they occur. This is one respect in which the system lacks qualities of graceful degradation. It should be noted, however, that one could design enhanced versions of the system that would contain multiple "back up" CMs that would hold data structures not currently in the focus of attention and for which there is no room in the main CM. Indeed, the new version of Conposit that performs massively parallel case-based reasoning does have multiple CMs, and data structure processing occurs in many CMs simultaneously.

An idea of the capacity of a CM can be gained from considering the maximum number of two-argument propositions it could conceivably hold. A two-argument proposition requires four registers, and (in the current Conposit) none of the registers is allowed to touch any of the registers for another proposition. The propositions can be packed most tightly if each one has its registers configured as a square group, where groups must be separated by at least one register width of free space. In this way each proposition effectively consumes a square of nine registers, so that the 32×32 CM can in principle hold up to 100 two-argument propositions. However, Conposit does not currently seek to pack each proposition into a four-register square, but instead arranges the relationship and argument registers in a random way around the head register. This makes the capacity difficult to analyze exactly, but a crude lower bound of 64 for the maximum number of propositions results from assuming each one consumes a sixteen-register square including separating space.

Although Conposit, as we have just implied, does not seek to minimize the space each proposition consumes, it does seek to pack them tightly, in the sense that a new proposition is placed as close as possible to existing data structures. This placing can easily be achieved by simple highlighting manipulations.[5]

On the Use of a 2D Register Array

Our decision to make Conposit's array be *two* dimensional, and indeed the decision to use and *array* at all, have sometimes been questioned. The two-dimensionality is relatively insignificant from some points of view: given that we wanted an array, we had to choose some dimensionality, and two was a good choice both from the point of view of graphical presentation and directness of simulation on present-day array processors (Conposit has been simulated on the Massively Parallel Processor and the Connection Machine, both of which have two-dimensional arrays of processors). We do have two more substantive motivations, though. One is that we are interested in the applicability of Conposit to spatial reasoning by using the CM to represent a 2D projection of a region of space (see Section 6 for more on this), and it is simpler to confine attention initially to two dimensional spatial reasoning. The other motivation was that we are interested in the possibility that a Conposit-*like* system (but not Conposit as it currently stands) could be the basis of a theory of high-level cognition in the brain, by taking the 2D matrices to be idealizations of regions of cerebral cortex.

As regards having an array at all, that simply reflects our desire to incorporate as conceptually simple as possible a version of RPE, in order to investigate the potential of that technique. Some commentators have asked why we do not simply have an unstructured collection of what we may call "big registers", each of which consists essentially of two of our actual registers. Essentially, the proposal is to do away with RPE and to treat the big registers as component subnetworks in a PSA scheme.[6]. One of the registers in a big register would be the Associator part and would contain an unassigned symbol, while the other would be the Information part and would in simple

[5] The free-space grabbing mechanism in current Conposit simulations is an advance over that reported in other papers, such as Barnden (1988, 1990). Conposit as described there did not try to place new propositions near to old structures, and always split a new proposition with at least one argument up into two-register subgroups in the way described above (although rules always had the ability to detect and process existing non-split propositions). Nevertheless, the special highlighting manipulations used to effect free-space grabbing are similar in kind and complexity to those reported before.

[6] Though, in line with a point made in Section 4, there is still a limited form of RPE within individual component subnetworks.

cases contain a constant symbol. This is certainly a logically possible technique, but we see the following advantages in including the present array-based RPE.

First, we again have the point that we are interested in using an array-structured CM as a spatial analogue. Second, the inclusion of array-based RPE makes for more efficient processing, in that it leads to PSA-based links being processed less often, both in data structure detection during rule-enabling and in data structure modification in rule-execution. For instance, without RPE it would be more time consuming to find the agent of a love proposition, given access to the "big register" L acting as the head of the proposition. This is because we would need both a broadcast of the associator symbol in L (combined with a highlighting check) to get to the Associator part of the big register A encoding the agent of the loving, *and* a move of attention from the Associator part of A part to its Information part. On the other hand, to find the agent of an RPE-encoded proposition merely requires a move of attention from the proposition's white head register to the neighboring red register. Worse, symbol broadcasts in the service of PSA-based association-following have to be serialized on pain of getting the symbols mixed up, whereas RPE-based associations can be followed in parallel.

Moreover, if Conposit as it stands had a physical realization in which the CM was implemented as a physical array, then moves of attention to neighbors would be over a much shorter distance on average than moves between different "registers" in a RPE-free version. Although Conposit itself does not have a physical realization, other Conposit-like systems might.

Since data structures can appear anywhere in the CM, and the subconfiguration for a given data structure could have any one of possibly many different shapes, the Subconfiguration Detection Module is faced with a "pattern invariance" problem. However, as the details in Barnden (1988, 1990) show, there is no particular difficulty in solving the problem. In fact, the "spatial" regularity of Conposit's arrays actually give it an advantage, in simplifying and regularizing the circuitry needed. The Subconfiguration Detection Module is an acyclic graph of matrices isomorphic to the CM and connected to each other and/or the CM in a simple, near-topographical way. More important, analogous pattern-invariance problems will arise in *any* system that can recruit components for temporary representational purposes, no matter how unlike Conposit it is in other respects: the system must have the ability to process a given information structure no matter where and exactly how it might be realized within the network on any given occasion.

(Non-)Synchrony, and Time-Based Selection

A command signal sent by a rule actually goes to the CM's parallel distributor, whereby it is identically distributed to all the registers. Each register decides for itself whether and how to react to the signal, and the reacting registers proceed *more or less* simultaneously and more or less at the same rate. However, we assume that in a physical realization there are likely to be random variations (between registers, and between different occasions for a given register) in the speeds of reaction and the times at which reaction steps occur. Also, we assume that the copies of the command signal distributed to the register may take different times to reach their destination registers (allowing both for systematic biases and random effects). Therefore, although at the register-machine level of description the system is best thought of as synchronous, with registers reacting literally simultaneously, at the detailed level of connectionist circuitry there is no strict synchrony assumption.

Indeed, the lack of strict synchrony is exploited in an important, and apparently novel, way in Conposit. A command signal often needs to cause just *one, arbitrary* register, out of the set satisfying a highlighting condition transmitted in the signal, to react. This arbitrary selection of a register is achieved through a "temporal-winner-take-all" (TWTA) selection mechanism. When a register detects that it satisfies the signal's highlighting condition, it transmits an announcement to the parallel distributor. Because of the types of asynchrony noted (plus variation in the time taken to decide to send announcements), the announcements are spread out in time, and the parallel distributor tries to select the register sending the earliest arriving announcement. However, because of delays within the parallel distributor itself, this cannot always be done unambiguously, so that a further round of announcement may well be necessary (with a reduced set of registers), and so on. The TWTA mechanism is further described and analyzed in Barnden, Srinivas & Dharmavaratha (1990). It turns out that the number of rounds of announcements is roughly logarithmic in the number of registers initially sending announcements.

Variable Binding in Conposit

RPE and PSA allow arbitrary temporary associations among information items to exist within the CM. In particular, the examples of the use of PSA and unassigned symbols in Section 3 show that PSA/RPE-based associations provide a sort of variable binding: if an unassigned symbol is regarded as a variable, then placing it, say, at the head register of a love proposition "binds" it to that proposition. Another particular effect of RPE and PSA is to allow role-filling: for example, the *agent* of a love proposition is represented by whatever red-highlighted register is adjacent to the proposition's head register or to a register linked to that head register by PSA. We could, if we wanted to, regard role-filling as a type of variable binding, because a role could be viewed as a sort of variable.

So, Conposit achieves a type of variable-binding (and role-filling) *within* the CM. But there is also the question of the variable-binding performed in the process of firing and executing hardwired rules: this is a type of binding operating between the CM and mechanisms *outside* the CM. Barnden (1990) looks at how Conposit achieves variable-binding for rules. Overall, Conposit's variable-binding facility is fully general, although special types of variable binding are effected particularly quickly by the Subconfiguration Detection Module, the remaining types being achievable only by the execution of rule action parts. We will confine attention in the present paper to the aspects of rule variable binding that can be described by reference only to the CM.

One version of Conposit contains a rule that can be paraphrased as *"If a man M loves a woman then M is hungry."* In the sense that the action part of the rule, when it acts upon a particular detected love situation, is able to access the register representing the man, we can say that Conposit is binding the variables M to that man. The action part of the rule sends a couple of command signals that have the effect of switching on a certain highlighting flag *l* at the agent register of an arbitrary one of the love propositions detected by the Subconfiguration Detection Module. Note that it is *l* highlighting that identifies the agent for the purposes of building the hungriness proposition. The installation of the register clump for that proposition copies *whatever symbol* is in the *l*-highlighted register. The marking of a register with *l* highlighting can therefore be viewed as binding the variable M in the rule paraphrase. Since bindings are represented by highlighting, sooner or later, and since highlighting is an intrinsic part of the encoding scheme within the CM, the usage of bindings is just a special case of the way CM data structures are manipulated by command signals.

It is also important to observe that the variable binding achieved through highlighting is most simply viewed as a binding of rule variables to *CM registers* rather than either to the entities *denoted by* those registers or to the *symbols in* those registers. If we wish to view bindings as reaching out to the symbols (or denoted entities), as is often convenient, we must note the crucial played by the fact that a register can have different values at different times. That is, the system's capability to bind rule variables to different symbols (or denoted entities) on different occasions depends *both* upon the fact that the rule is able to impose appropriate highlighting (by *l* in our example) on selected registers *and* on the fact that a given register can have different values at different times.

In fact, variable-binding as construed as binding to *registers* is an example of what was called "processing-locus identification" in Barnden (1984b), in that highlighting is being used to identify CM registers as loci of processing. Variable-binding-to-symbols/entities can therefore be regarded as a combination of processing-locus identification and value settings of registers. The splitting of the issue into two parts is useful, because the same proposition can be realized in a large number of different ways within the CM (for instance, it can appear in different places in the CM): therefore, the *same* variable-binding (to a symbol/entity) can require *different* processing-locus identifications in different cases. The splitting of the issue would also be useful in discussing any other connectionist system in which the same piece of information can be represented in more than way. Most discussions of variable binding in connectionism fail to take proper account of this.

6. DISCUSSION

In this section we discuss some relationships that RPE and PSA bear to the connectionist information-structuring techniques reviewed in Section 2. This gives us the opportunity also to examine the relationships between the various connectionist techniques, and between connectionist techniques and the methods reviewed in Section 3 for information structuring in computers. In discussing RPE and PSA we intend our comments to provide insight into these techniques in general, not just into the specific versions of them to be found in Conposit. However, we will often draw upon the Conposit versions in order to make a point clearer or more precise, in the hope that the observations can be carried over in some form to other versions.

RPE and Sequential Allocation

RPE is a natural extrapolation from the sequential allocation technique in computers. In particular, Conposit's version of RPE is a particularly simple extrapolation. Importantly, it preserves the bidirectionality of association that sequential allocation provides: in Conposit it is just as easy to go from the agent register of a LOVES proposition to the head register as it is is to go in the opposite direction. This bidirectionality can be expected also to appear in more sophisticated versions of RPE.

One aspect of sequential allocation that we normally expect RPE to abandon is the global naming scheme for memory cells. In computers, one goes from a cell R to a neighboring one by taking the address of R, adding 1 or -1 to it, and then passing the new address through the memory-addressing circuit of the computer. However, in connectionist RPE we make no general assumption that the "component subnetworks" — the objects corresponding to memory cells — have names (addresses) in any interesting sense, let alone ones that can be computed from other names.

Instead, we make a more basic assumption, namely that there is *some* machinery whereby the system can (a) "transfer attention" efficiently from one component subnetwork C to other "related" ones, and/or (b) ensure that the changes at a given component subnetwork can depend on information in "related" registers. What the phrases "transfer attention" and "related" mean here depends on the particular form of RPE in question. Thus, in Conposit as described in Section 5, "related" merely means "adjacent", and to "transfer attention" is to cause certain highlighting changes at the destination component subnetwork (i.e. register) and at C. The machinery whereby

(a) and (b) are achieved in Conposit relies on connections between adjacent registers, allowing the highlighting state of a register to be sensed by any neighboring register.

No direct register-register connections other than the neighbor-neighbor ones are needed in Conposit as presented in this paper, because its RPE exploits only immediate-adjacency relationships within the CM. Sequential allocation in computers, however, may involve concrete records of any length, so that the ability to make large hops between memory cells is desirable. Therefore, it is beneficial to have a way of computing an address within a concrete record by adding a (possibly large) hop amount to another address within the record. In principle it would be sufficient (from the point of view of moving about within a concrete record) to have direct connections between memory cells, but these would lead to major inefficiencies in dealing with large records. Also, of course, the machinery for accessing cells by global addresses is used to support pointers between cells. Conposit lacks pointers in the usual sense — and so might other RPE-based systems — so that one reason for having global addresses disappears.

The direct neighbor-neighbor connections in Conposit can be viewed as a way in which any given register R "knows which registers are its neighbors". Notice, however, that there is no need for R to be able to distinguish its neighbors from each other, because a command signal only ever directs a register to consider whether *all* or *some* of its neighbors obey some particular highlighting condition. Conposit's indistinguishability of neighbors is to be contrasted with the distinguishability of the two neighbor cells of any cell in a computer memory. Versions of RPE in systems other than Conposit as described here could, nevertheless, have some ability to distinguish neighbors from each other.

Indeed, Barnden (1987) proposes a version of Conposit in which neighbors need to be distinguished by direction. In this version, the 2D space of registers in the CM is used as an analogue representation of a plane in real space. Th system is for emulating the type of spatial reasoning to which part of Johnson-Laird's mental-model theory is directed (Johnson-Laird 1983). The focus of his discussion is on inferring the relative spatial positions (left-of, above) of some objects in a planar configuration, given some information about relative spatial positions of some of the objects. In this context, it is desirable to be able to perform operations like the following:

IS-RIGHT?:

> given that two registers are highlighted using some special flag *f*, where the registers may be widely separated in the CM, determine whether one of them is to the "right" of the other (assuming that some fixed direction in the CM corresponds to the direction "right" in the real spatial plane being represented).

Currently, the way we propose that this operation be carried out is that the system make one of the registers send out a highlighting "wave" in a rightwards direction (using a flag *w*), and then wait to see if some register becomes highlighted in both *w* and *f*. The wave spreads by means of a series of steps, at each of which any register more or less to the right of a register highlighted in *w* becomes highlighted in *w*. By "more or less to the right" we mean the register whose position is one further along in one dimension of the CM and possibly one further along or back in the other dimension, so that each non-edge register has three registers more or less to the right of it. Clearly, the process can only be done if registers have some ability to distinguish among their neighbors on the basis of direction.

Notice also that in this version of Conposit it is no longer the case that only short-range relative position relationships are important. The question therefore arises of whether something akin to the global numerical addressing in a computer memory should be added in order to speed up the implementation of the operation just discussed. We observe here the obvious but important fact that arbitrarily long-range relative-position relationships can be discovered in a computer memory merely by numerically comparing addresses (assuming, crucially, that we already know what the relevant cells' addresses are — if we did not, we might need to engage in a very time consuming search process). Nevertheless, we have resisted the temptation of adding some sort of numerical addressing to speed up the spatial-analogue version of Conposit. The reason is that the CM is assumed to contain far fewer locations than an ordinary computer memory does; hence, the iterative "wave" process described is not too time-consuming.

These allusions to spatial analogue representations should remind us that sequential allocation is the standard tool for implementing arrays, of any dimension, in a computer memory. The use of a CM to support a spatial analogue representation is no more radical than the very common practice, in scientific programming for instance, of using a conventionally-implemented array in a computer memory as an isomorph of some array of points in real space. Notice also that the spatial-analogue use of a CM can be temporary, just as the use of a particular region of computer memory to support an array representation can be temporary. At other times, the CM might be used in the way described earlier in this paper. Indeed, there is no reason why both styles of representation should not be mixed together simultaneously in a CM, as is suggested in Barnden (1985, 1987).

RPE in CONPARSE

As we pointed out earlier, the CONPARSE system of Charniak & Santos (1987) uses a form of RPE. This form is again a simple extrapolation from sequential allocation, though different in detail from Conposit's RPE. The model's use of RPE makes it Conposit's closest neighbor in the space of existing connectionist systems.[7] CONPARSE's RPE is less uniform than that of Conposit, in that different directions in CONPARSE's array have different representational significance. Recall that only the up-down dimension is for representing constituency relationships, whereas the left-right dimension, with the aid of the "binding units", is for representing identity of constituents.

It is probably best to regard a CONPARSE binding unit as a sort of pointer, since a "binding unit" in the connectionist field is generally a unit that, when active, is taken to bind two fixed portions of the network, rather than a unit that is able to bind different things together at different times. The interesting thing about the binding units in CONPARSE is that they are a hybrid of RPE and the usual notion of pointer: a CONPARSE "pointer" from a register in a given column is not a global address, but rather an address *within* the column immediately to the right of C. However, we also pointed out in Section 2 that each CONPARSE binding unit could be replaced by (or implemented as) r binding nodes of the more conventional sort, where r is the number of rows in the array.

RPE and Absolute-Positional Techniques

We turn now to the relationship between RPE and "absolute-positional" techniques, as defined in Section 2. To take an example mentioned there, suppose there is a letter unit T_1 that represents the presence of the letter 'T' in position 1, and a letter unit H_2 that represents the presence of the letter 'H' in the next position, 2. Then we may certainly abstract from this description and say that the simultaneous activity of T_1 and H_2 represents the presence of

[7] Conposit was developed independently of the Charniak & Santos system — an early version, using much the same type of RPE (and PSA) as current versions do, was presented in Barnden (1985). We should also emphasize that Conposit bears very little relationship to the memory-field proposal of Kohonen, Oja & Lehtiö (1989). A similarity between memory fields and Conposit's CM has sometimes been claimed, but in fact the former make no use of RPE or PSA.

a 'T' and and 'H' together with the fact the 'H' follows the 'T' in the word viewed. Hence, a contiguity relationship (a type of "abstract association") in a word is represented by making particular choices of the units for representing the two letters. This brings out a loose similarity to RPE: an RPE-based representation of an abstract association rests on particular choices of component subnetworks with which to represent the things associated.

We can make this point more precise by regarding the set of letter units for any given position i as forming a component subnetwork, called C_i say. Furthermore, we regard the C_i for increasing i as forming a linear array of component subnetworks. In a proper representation of a word, any given component subnetwork has at most one of its units on. So, the activity pattern in a component subnetwork consists either of OFF values at all units or of an ON value at just one unit. To represent two contiguous letters we turn on the appropriate letter units in C_i and C_{i+1} for some i. We have now described a situation that fits roughly in the general description of RPE in Section 4.

Of course, what we have suppressed here is the fact that we are not allowed to choose *any* i: the i must be 1 if the earlier of the two letters is at position 1 and so on. Therefore, we have an extra restriction — reflecting the use of an ***absolute*** positional encoding — that we did not include in the general description of RPE. This is not to say that we are necessarily violating RPE — we could instead say that we have a very special, or perhaps degenerate, case of it. A modified form of the scheme that *would* be RPE in a more typical sense is as follows. Suppose we have a linear array of component subnetworks C_i of the form just mentioned, but no longer taken to correspond to particular positions in words. Let us introduce an extra unit in each C_i, with the intention that if the unit is on then the C_i is *currently* taken to correspond to the first position in the word, so that only one C_i will have the unit on at any time. (In essence, the extra unit is like a Conposit highlighting unit.) Let it still be the case that if a C_j and C_{j+1} have letter units on then they represent contiguous letters, somewhere, in the word being viewed. Then, to a represent a word of length N, any contiguous series of N C_i subnetworks could be used. The absolute positions in the array are no longer important. We are not saying that this modified, RPE-based scheme is better than the original, absolute-positional version. We are simply making a point about the relationships of techniques.

As might be expected, there are intermediate possibilities between a truly absolute positional scheme and a fully relative positional scheme. The dimension of variation in between is that of the extent to which one is forced to choose particular component subnetworks (as opposed to choosing a particular relative "position" of those subnetworks). Just for the sake of illustration, one intermediate possibility would be a Conposit-like scheme in which different portions of the CM would be dedicated to particular representational purposes. For example, personal relationships would have to be represented in one particular region, physical position relationships in another, and so on. Further, we assume that the system's interpretation of a subconfiguration would depend on which region it was in. Then, although the usual RPE (and PSA) would reign in each portion, and indeed PSA linking might be able to cross between portions at will, the relative-position encoding would be "less relative" than in Conposit as it stands.

Absolute-Positional Encoding and Binder Nodes

Consider the following variant scheme for representing a word, based on binding nodes. It again uses a set of component subnetworks, each of which represents (at most) one letter at any time. However, the subnetworks are not organized into an array, and no subnetwork corresponds to any specific position. Instead, for each pair of subnetworks (C, D) there is a binding node connected to C and D. When the binding node is active, C and D are taken to correspond to contiguous positions in the word. (We defer until later the issue of specifying whether it is the C position that follows the D position or the other way round.) There is also some means, perhaps a highlighting node again, for specifying which component subnetwork corresponds currently to the first position in the word.

Now, in the absolute-positional scheme, contiguity was represented by means of the activation of particular units (letter units). And, in the present binding-node scheme, contiguity is still represented by means of the activation of particular units: in this case, a coordinated choice of letter units and binding nodes. Hence, simply saying that particular units are activated does not distinguish the schemes. The obvious next step is to say that the critical difference from the absolute-positional scheme is that nodes *other than* the letter units are involved. However, matters are not as clearcut as they seem, since if one considers extra machinery that might be present in a system using the absolute-positional scheme, we are likely to find things similar to binding nodes elsewhere in the system. To take a simple instance of this, suppose there is a single output node O_{THE} that represents the word 'THE' and which therefore lights up when T_1 (i.e. the 'T' unit in C_1) is on, H_2 is on, E_3 is on, and no other letter unit is on. Then, O_{THE} can, if we wish, be considered as a sort of binding node, since it connects some component subnetworks and is active just when they are to be taken as being "bound together" — in a rather specific way. A less extreme example of the same point can be made by considering possible digram units, or other subword units such as the triple units in Section 2, that might be present.

An important difference from the binding nodes postulated at the beginning of this subsection is perhaps that nodes like O_{THE} become active as a result of activity in a *self-sufficient* lower level representation, in this case the letter-level representation of the word 'THE'. Unit O_{THE} could be turned off without destroying our ability to say that the system is encoding this word. By contrast, the earlier binding nodes' activity was an essential *part of* the representation of the word — they could not be turned off without destroying our view of the system as encoding 'THE'. Notice that the distinction holds even if O_{THE} *dynamically contributes, by top-down feedback, to the establishment, and even the maintenance, of the letter-level representation.* For, we can fall back on the following counterfactual statement: *if* it were possible to turn off O_{THE} without turning anything at there letter level on or off, then the system could still be seen as representing 'THE'.

The moral from this is that the description of a given system as using a given encoding technique can depend very much on one's view of the system and of the allowable variation in the technique — on the level of description, on how one parcels up the parts of the system as to function (representation and recognition, for example), and, in our case, on what one is prepared to accept as a "binding node".

RPE and Binder Nodes

Having seen that absolute-positional techniques can implicitly bring in binding nodes, we should ask whether RPE can do so. We answer this by looking, for definiteness, at Conposit's version of RPE. Barnden (1990) shows that Conposit's RPE as manifested in the required circuitry in the Subconfiguration Detection Module brings in binding nodes in a somewhat straightforward way. The nodes temporarily bind nodes in the rule action parts to CM registers.

If, however, we look at Conposit's RPE as manifested in the role it plays in the CM, we find that it can again be seen as bringing binding nodes in, but only in a rather complex, forced and artificial sense. We observe first that an adjacency relationship within the CM is construed as representing a temporary association only if the registers concerned have suitable highlighting. For example, a register A adjacent to a register S that represents a love situation is only construed as representing the agent if it (A) is highlighted in red. Consider what happens if a rule wants to find the agent register A, on the assumption that S is already marked with "detected" highlighting.

say. Let us say that the rule must mark A with highlighting *l*. Then what the rule does is to send a command signal to the CM, telling every red register adjacent to a "detected" register to turn highlighting *l* on. Some component signal in this command signal must therefore bring into play some sort of connection path P from S's "detected" highlighting flag to A's *l* highlighting flag. Thus, ONness at A's red flag acts as temporary facilitation of some connection path that can be traced through the register-internal logic of A and S and the connections between A and S. In this sense we might be tempted to say that A's red highlighting unit is acting (for the moment) as a binding node between A and S.

However, the red highlighting unit in a given register A is not preferentially connected into any one of the neighboring registers, whereas a binding node normally does mediate between two particular nodes or sets of nodes. We must pursue the above account in a little more detail, referring to Fig. 7. The figure shows register A and three of its neighbors, including S. The small boxes illustrate highlighting flags, with *r* standing for red, *d* for "detected", *n* for "neighborly", and *ls* for "locally-satisfying". If a small box has a double wall then the flag is ON.

FIGURE 7 ABOUT HERE

The command signal mentioned actually causes every register with "detected" highlighting on to turn its so-called "neighborly" highlighting flag on, and every red register to turn its "locally-satisfying" flag on. These effects are illustrated by the lines joining the *d* and *r* boxes to the *n* and *ls* boxes respectively. (These lines do not stand for direct connections.) In A (as in every register) there is a unit that does an OR operation on the "neighborly" highlighting units of all of A's neighbors. The result of this OR is then ANDed with the value of A's "locally-satisfying" flag. If the AND result is ON, then A is a register that performs the state change (namely, the switching on of *l* highlighting) dictated by the command signal. Altogether, then, the binding of A and S is not actually encoded by activity at A's red highlighting unit alone, but actually by the *combination of* activity at A's red highlighting unit and activity at S's "detected" highlighting unit. Thus, we should really say that it is this pair of units that together act as a binding node. Moreover, A's red highlighting unit is only connected in an elaborately indirect way to S's "detected" unit.

Highlighting has uses other than specifying roles in propositions. For example, we have seen that *l* highlighting marks registers that are to be worked on by a rule. Therefore, the binding-node view of *some* highlighting flags at *some* occasions during processing is a relatively superficial aspect of a more fundamental matter.

Although we have concentrated on Conposit in this subsection, it is to be expected that other versions of RPE are likely to be amenable to a similar analysis.

RPE and Time Phases

The reader will no doubt have noticed the space/time duality between RPE and the time-phase method mentioned in Section 2. The latter uses relative "position" in time (actually, *simultaneity* of periodic pulses) to encode associations, as opposed to relative position in network "space" (CM space in the case of Conposit). We find, however, the time-phase method has a more substantive relationship to PSA than to RPE. We look at the relationship to PSA below.

RPE and the Typing/Orienting of Associations

Clearly, the highlighting combinations involved in Conposit's RPE-based associations serve to "type" the associations and to specify their "directions" or "orientations". For example, a red register adjacent to a white register denoting a love-situation denotes the agent of the situation, whereas a green register adjacent to the situation register denotes the object. The importance of this association typing and "orienting" facility should not be underestimated: it is evidently a very important capability, but is one that is not trivial to realize in connectionist systems, especially those that seek to encode associations as facilitated connection paths, whether the facilitation is by weight-enhancement or activity at binding nodes.

In a binding node scheme, there are difficulties enough just in orienting, and we will comment only on this point here. For definiteness, we can go back to our hypothetical introduction of binding nodes into a word-representing system. One solution might be to have, for each subnetwork-pair C, D, two binding nodes, connected in exactly the same way as each other to C and D, but having different interactions with the the rest of the system, so that is is up to rest of the system as a whole to "know" the differing significance of each of the binding nodes. A further possibility is to have two binding nodes, but connect them differently to C and D; for instance, one binding node could have a stronger connection to C than to D, vice versa for the other binding node. This asymmetry might in principle be a suitable basis on which the rest of the system can proceed. (The obvious idea of using different directions for the connections is problematic, because the orientation of a binding has nothing to do with the directions in which the system might have to *traverse* it.) Another proposal is as follows. There is a binding node b is connected to both C and D, and in the same way to each. Another node b_C is connected to C and b, in the same way to each. Similarly, a third node , b_D, is connected to D and b. Node b is on if C and D represent contiguous positions, either way round. When the C position is meant to be the earlier one, node b_C is also on; and similarly for D. This scheme is an implementation of the standard set-theoretical device for representing ordered pairs by means of unordered sets: represent the pair (C, D) as the set $\{C, \{C, D\}\}$ and the pair (D, C) as the set $\{D, \{C, D\}\}$.

We do not dwell further on these possible solutions, which all involve a considerable increase in the elaborateness of the whole system.

the role-distinguishing highlighting flags in Conposit can be seen as binding nodes, as discussed a moment ago. The orientation problem is solved by the fact that a highlighting unit's relationship to the rest of the circuitry in its own register is different to that to the circuitry in adjacent registers. To put it another way, no register confuses any neighbor's highlighting state with its own.

The orienting issue for binding nodes has received very little attention in the connectionist literature. There is an interesting reason for this: proposals generally confine binding nodes to mediate between subnetworks of markedly different types or which have clearly different roles the distinction between which is hardwired into the rest of the system. For instance, in Smolensky's tensor-based system (applied as presented in Section 2) a binding node sits between a frame-role subnetwork and a filler subnetwork. There is an assumption that the system already knows, so to speak, which subnetwork constitutes the role or rule and which the filler. Similarly, in DCPS (see Section 2) a binding node can be viewed as binding something in one clause space to something in another. Again, there is an assumption that the system already knows which clause space is which — they *permanently* play different roles in the whole system. Thus in both systems the required asymmetry is built-in. However, there is no such built-in asymmetry in the case of the component subnetworks in the word-representation example.

PSA and Associative Addressing

Association by symbol sharing, which is Conposit's version of PSA, is a simple extrapolation from associative addressing in computers: clumps of CM registers linked by PSA are like concrete records linked by associative addressing. In fact, Conposit's PSA is probably more like associative addressing than its RPE is like sequential allocation in computers. Conposit's PSA obviously preserves the bidirectionality of association that is provided by (the simpler forms of) associative addressing in computers, and more advanced versions of PSA can be expected to preserve it as well.

In a more complex version of PSA, we might still have separate Associator, Information, and other parts in component subnetworks, but the required similarity associator patterns might be a looser relation than strict equality. Indeed, in the new, case-based reasoning version of Conposit being developed (Barnden & Srinivas, to appear), head registers of propositions contain unassigned symbols in the form of activity patterns that are automatically computed on the fly by the system. The computation is includes a hashing transformation applied to the symbols and highlighting states in the registers representing the relationship and the arguments. The computation of the unassigned symbols also involves the addition of a small random perturbation to each component of the activity pattern, in order to prevent unwanted symbol-sharing relationships (i.e unwanted PSA). PSA no longer requires strict equality of symbols as activity patterns, but instead allows small differences at each component of the pattern.

PSA and Binder Nodes

Earlier we saw that there was a rather strained sense in which RPE can be viewed as bringing in binding nodes (at least in Conposit's version, and quite probably in other versions). An analogous observation can be made about PSA, except that the view is even more strained.

Suppose several CM registers in Conposit contain the same unassigned symbol X. The way this association would actually be used within the CM is for a command signal to cause a symbol to be broadcast from one of the registers, and subsequently some change to take place in the other registers, by virtue of their noticing that their own symbols are equal to the one just broadcast. The broadcast goes through the CM's parallel distributor. The circuitry in each register R contains a subnetwork S_R whose activity pattern constitutes the symbol in the register. The symbol-broadcast machinery therefore involves indirect connection paths joining each S_R to every other $S_{R'}$, through the parallel distributor. Let S be the total subnetwork formed by the S_R networks for all the registers R in some set mutually associated by symbol sharing.

A sharing of a symbol X by the registers in the set in question is equivalent to the presence of a particular activation pattern over S. For every other unassigned symbol Y, another activation state of S is similarly deemed to define a Y-based temporary association among the same registers. Assume that such an activation state of S is binary (ON/OFF) at each unit, so that we can identify the state by the portion of S that is ON. Then we can say that each such portion is a complex, distributed binding "node" which binds the registers in question.

Clearly, this view of Conposit's PSA as involving binding nodes is a highly complex one, at best. However, connectionist research has countenanced quite complex binding node arrangements, witness Smolenky's and Touretzky & Hinton's distributed binding node schemes reviewed in Section 2. The view of PSA as involving binding nodes is merely at an extreme point on the same path of increasing elaborateness.

There is nothing very special to Conposit in the above argument, since any system using PSA is going to need circuitry directly or indirectly allowing associator patterns to be transmitted among component networks.

PSA and Signatures

It should be clear that a "signatures" technique such as that used in the ROBIN system reviewed in Section 2 is a special case of PSA, if we assume that a signature is implemented as some sort of activity pattern, be it as simple as an activity value at a single node. A signature is an associator pattern in our terms. A special feature of ROBIN's PSA is that the signature node for a concept (one type of component subnetwork) contains a *constant* signature (i.e. associator pattern), whereas the binding node for a role of a frame (another type of component subnetwork) can contain any signature. Also, in this view, a component subnetwork (either (a) a concept node plus associated signature node, or (b) a frame subnetwork together with bind nodes) has a semantics that is wholly or partially fixed, whereas generally in PSA there is no assumption that a component subnetwork is semantically fixed at all.

Each slot part of a ROBIN frame counts as a distinct associator part. The possibility of a component subnetwork containing several associator parts was countenanced in Section 4. Note also that Conposit has a similar feature, since several registers in a register clump can contain unassigned symbols, linking the clump to various other clumps.

Since the signature in a concept subnetwork is constant, and different from the signatures for other concepts, a signature actually identifies *a particular, constant subnetwork*, as well as identifying, at a higher level of description, a particular *concept*. Hence, a signature is somewhat like an address or pointer as well as being like an associative addressing key. Certainly, the machinery used in a signature-based system for using a system to access a concept subnetwork may be different in detail from the address-decoding machinery used in a computer. However, the notion of an address or pointer is not tied to any particular mechanism for using a given address or pointer to access what it identifies.

In order to buttress their inclusion of signatures, Lange & Dyer (1989) note that signatures might conceivably appear in the brain as phase-locked patterns of oscillation produced by pacemaker neuron assemblies and central pattern generators. We may also note that Eckhorn, Reitboeck, Arndt & Dicke (1989) have proposed that observed phase-locking of oscillations in possibly widely separated parts of visual cortex (in the cat) might be used to transiently link together information extracted from different parts of an image. This fits in well with the PSA idea, since similarity of oscillation frequency and phase can be taken as a valid notion of associator pattern similarity.

PSA, the Time Phase Technique, and RPE/PSA Duality

In Section 2 we looked at a connectionist information-structuring technique based on time phases. The last paragraph of the previous subsection leads us to the observation that the time-phase technique is a PSA technique in which every associator pattern is *purely* temporal in nature, and where, furthermore, the temporal quality is purely a matter of phase. The pulsations of activity at different nodes in the system of Shastri & Ajjanagadde (1989) are all at the same frequency, but possibly at different phases. A node's pulsation at a given phase serves as an associator pattern that links the node to other nodes pulsing at the same phase.

In this view of diversely phased pulsations as providing a form of PSA we get away from the idea in ROBIN of using an associator pattern (signature) as an address of a particular, fixed subnetwork, so that the time-phase technique is more typical as a PSA technique than the signature technique is. Now, we remarked earlier that the time-phase technique is a temporal dual of RPE. We can now observe that this is just a special case of the fact that RPE in general and PSA in general are dual to each other in the following sense. RPE exploits *similarity of spatial position* of activity patterns (the similarity consisting in adjacency, in the case of Conposit) but not their similarity as patterns, whereas PSA exploits *similarity of pattern* but not similarity of spatial position.

The negations here should be taken as being partial, so that the duality is only approximate. For instance, it is possible to imagine a PSA-based system in which different modules used independent PSA-based representations, so that the presence of the same associator pattern in component subnetworks in two different modules did not constitute a cross-module association. In this case, the PSA in the system as a whole involves a crude type of similarity of spatial position, in the sense that an association requires the associator pattern instances to be in the same module and thereby to be in "similar" positions. Conversely, one could imagine a form of RPE in which two adjacent component subnetworks were only taken as being associated if they were in similar states in some sense — for example, if they both had a certain highlighting flag on, if the system were somewhat Conposit-like. (We should recall that in Conposit adjacency only signifies association if the adjacent registers are in suitable highlighting states. However, it is difficult to see this dependence on highlighting as a type of pattern *similarity*.)

PSA and BoltzCONS

At a high level of description based on symbol triples, BoltzCONS uses an associative addressing scheme. Given that PSA is also associative, the question arises of what relationship the BoltzCONS technique bears to PSA.

An active triple (a,b,c) in a space called Cons Memory is considered to be linked by its CAR field (its second component) to the (assumed unique) active triple having b in its TAG (i.e. first) field, and similarly by its CDR field (its third component) to the active triple having c in its TAG field (assuming there are such triples, i.e. that b and c are not just "basic" symbols denoting objects in the domain of discourse).

However, triples are not implemented by separate component subnetworks as would be required for a simple view of BoltzCONS as using PSA. Each triple is represented by about 28 units in Cons Memory, and the unit-sets for different triples can overlap. Therefore, the question arises of whether one can discern a form of PSA using *distributed* component subnetworks (= triple representations), a possibility mentioned in Section 4. The simple answer is that we cannot: Cons Memory is simply an unstructured set of units, and the 28 or so units representing (a,b,c) *need* have no overlap with the set of 28 or so units representing the triple starting with b. We fail to see any useful sense in which activity over the former set is similar to activity over the second, in general.

On the other hand, there does remain a sense in which BoltzCONS can be seen as using PSA. The way a triple in Cons Memory is actually used in processing is for it to be converted to patterns of activity over the so-called TAG, CAR and CDR spaces. See Fig. 8.

FIGURE 8 ABOUT HERE

The triple (a,b,c) is converted into the pattern in the TAG space representing the symbol a, the pattern in the CAR space representing the symbol b, and the pattern in the CDR space representing the symbol c. Similarly, on a different occasion the triple (b,d,e) could be converted into the b, d and e patterns over the TAG, CAR and CDR spaces respectively. Now, as far as we can determine from Touretzky (1986), the TAG, CAR and CDR spaces are isomorphic, with one-to-one connectivity between corresponding units, in order to support the simple copying of symbol-representing patterns between the spaces. Under this assumption, the (a,b,c) and (b,d,e) triples lead to the same symbol pattern (for b) being present, only the pattern is instantiated in the CAR space in the case of the former triple and in the TAG space in the case of the latter. A typical operation in BoltzCONS is to convert (a,b,c) in Cons Memory into the representation over the TAG/CAR/CDR spaces, copy the b pattern from the CAR space into the TAG space, set the other spaces to zero activity, use the TAG space to cause stimulation of the representation of (b,d,e) in Cons Memory, and then convert this representation into the corresponding representation over the TAG/CAR/CDR spaces.

The upshot of this is that the combined TAG/CAR/CDR space can be regarded as a single component subnetwork that represents different triples at different times, where triples appear dynamically through an operation based on pattern similarity. Thus, we might call this scheme a "diachronic" PSA scheme, where, instead of having ***several*** component subnetworks simultaneously present and containing triples *synchronically* inter-linked by pattern similarity, there is a *single* component subnetwork containing triples *diachronically* inter-linked by pattern similarity. Clearly, these are extreme ends of a spectrum, and we could presumably design schemes that intermix synchronic and diachronic PSA.

Suppose now that the TAG and CAR spaces are not isomorphic, and there is no obvious sense in which the pattern representing b in TAG space is similar to the pattern representing b in CAR space. Rather, there is some more or less complex arrangement of connections joining the two spaces, such that the presence of the b pattern in one space can be used to cause the presence of the b pattern in the other space. Can we still say we have a diachronic PSA scheme? Our inclination is to say yes, regarding the b patterns in the two spaces as being similar in an advanced sense. However, for the reader who objects to this liberal view of similarity we suggest that BoltzCONS can be viewed as using a diachronic, "pattern-relationship association" (PRA) technique. PRA is just a simple generalization of PSA:

Pattern-Relationship Association

> PRA is defined by modifying the general presentation of PSA in Section 4 as follows. An associator pattern X in one component subnetwork is considered to associate to patterns XX in other component subnetworks that are *related* to it in some specified way, rather than similar to it.

Then, PSA is simply an especially important species of PRA. We will in any case be bringing in PRA for other reasons in a moment.

PRA, Reduced Descriptions, and PSA

In our description of the reduced descriptions technique (RDT) in Section 2, we talked as if the activity patterns D and DD must sit in distinct subnetworks. Although this is not in fact a necessary aspect of RDT, we will assume it to be the case for simplicity. Then, if we call these subnetworks component subnetworks, it should be clear that RDT falls clearly under the notion of pattern-relationship association (PRA) described at the end of the previous subsection. In RDT, a pattern is "related" to another if it is a reduced version of the other or vice versa, reduction being defined as the transformation effected by T. (Of course, we need to know which way round the reduction is going here, but that is no problem in the systems cited, since the set of component subnetworks is non-homogeneous).

In this way, RDT and PSA can be viewed as sibling techniques, with PRA as their parent.

Notice also that a new version of Conposit (Barnden & Srinivas, to appear) referred to earlier can be viewed as using a version of RDT that includes PSA as a subcomponent. In Fig. 5 (for "Bill hopes that John loves Mary") the unassigned symbol X in the new version would be computed from the states of the registers holding the LOVES, JOHN and MARY symbols. The combined state of those registers can be taken as the activity pattern D in the previous paragraph. X is therefore the corresponding reduced representation d for "John loves Mary". The register clump for the "Bill hopes" aspect of Fig. 5 uses the reduced representation X (i.e. d). The symbol/highlighting activity pattern over this clump can be taken as the pattern DD. The reduction mechanism T is the subsystem for computing unassigned symbols from states of appropriate registers. The inverse mechanism t is less obviously isolatable, but can be taken as the collection of mechanisms that takes an unassigned symbol X, broadcasts it to all registers, and is able to switch the system's attention (e.g. by highlighting changes) to neighbors of registers that contain X. Under this RDT view, the PSA arising from the sharing of X can be seen as a part of t.

Reduced Descriptions and Hashing

Earlier we said that the transformation performed by T in the new Conposit was a type of "hashing". We have also just said that this transformation can be regarded as the reduction transformation in an application of RDT. This is just a special case of the fact that RDT and hashing in computers are strongly related. Surprisingly, there has been little attention by connectionists to this strong relationship, despite the important role that hashing plays in computer science and artificial intelligence. We bring out the similarity of RDT and hashing as follows.

In hashing, a hashing function T maps data structures D to integer hash keys d that are used as the positions of entries in a hash table HT. The hash key d is usually a much smaller datum than D itself. The entry that d indexes in HT typically contains D itself in full form, together with extra information associated with D. Therefore the table can be regarded as a mechanism t that (among other effects) delivers D on presentation with d. There can, however, be "collisions", when several different D map to the same d. This complicates the picture, but does not significantly weaken the relationship to RDT, since the latter also can have several or many D mapped to the same d. (We suppressed this complication in our description of RDT, and will continue to suppress consideration of collisions in our comments on RDT and hashing, in the interests of abbreviating the discussion.)

There are certainly some differences between hashing and RDT, but they are relatively superficial. First, hashing is predominantly used merely as affording a mode of data structure storage: unlike RDT, hashing is not generally used as a way of building data structures. However, there is nothing in the hashing technique that prevents it being used for this purpose.

The second difference one might point to between RDT and hashing is that the values d produced by the hashing function T are normally addresses of storage locations, in essence, whereas in RDT the values d produced by T are associative addressing keys of a sort (in that they allow access via the expansion mechanism t to the patterns D from which they came). However, the fact that hash keys are normally location addresses (in essence) merely reflects the fact that hashing occurs mainly in ordinary computer memories, rather than in associative memories. After all, ordinary memories are much more common than associative ones. However, in a computer with an associative memory, d can indeed be used as an *associative* addressing key rather than as a table-entry address, making the d values in hashing look much more like the d values in RDT. This method is used in the "set-associative" scheme for cache memory in the Vax 11/780 computer (see Stallings 1987, p.114): a subsequence d of a bit-string D is used as an associative addressing key for accessing information related to D. (The D bit-strings are in fact addresses of blocks in main memory, but that does not disturb our point.)

A third difference is that it can be desirable for the reduction mechanism T to preserve pattern similarity, so that the reduced descriptions can take part in associative operations that reflect associative operations that could be effected more elaborately and accurately on the full representations (Hinton 1988). On the other hand, in hashing, one often tries hard to ensure that similar data structures hash to markedly *dis*similar hash keys, in order to minimize the probability of collisions. However, this feature of hashing is really driven by special considerations to do with the types of task to which hashing tends to be applied, and has nothing to do with the essence of the technique. For instance, in the typical case when the hashed data structures are programming language symbols, symbols which are similar as strings of characters usually have no semantic relationship which the language processor is expected to respect, so that there is no reason for wanting them to hash to the same or similar keys. However, such collisions or near-collisions could be desirable if they result from *meaningful* similarities among the hashed data structures. For instance, if the data structures are board positions in a game, the preservation of board similarity by the hashing function could enable positions similar to a given one, P, to be found efficiently by looking at the positions indexed by hashing function values approximately equal to the hashing function value for P.

Hashing is used in computers precisely because it is a way of *avoiding sequential search* to a large extent. On average, it allows very fast retrieval, with almost no search of memory, since (ideally) the value d gives immediate access to D. (The possibility of search arises because of collisions, but with the right parameters and methods the effect of collisions can be made small on average.) This point is highly significant from the point of view of seeing how connectionism relates to other areas of computation science. It is too often assumed that one sharp contrast between connectionist systems and standard symbolic systems is that the former substitutes fast (parallel) memory access mechanisms for slow (sequential) memory-search mechanisms. Although there is a significant amount of truth in this, it greatly oversimplifies the true situation.

Finally, we observe that the new Conposit's hashing is appreciably more like conventional hashing in computers than prototypical RDT is. This is because we can regard the whole CM itself as the analogue of a hash table. Going back to the example we used in Fig. 5, the unassigned symbol (hash key) X provides access to the "hash table entry" consisting of the register clumps that involve X. In RDT as found in the systems cited, there is no such clear analogue of a hash table. Indeed, in those systems the mechanism t is not a straightforward *access* mechanism as it is in the new Conposit and in computer hashing, but is more straightforwardly speaking a *reconstruction* mechanism. Having said this, it would be difficult to draw a sharp line between the notions of access and reconstruction.

Pointers and Associative Addressing

Many people probably think of associative addressing and pointers in computers as very different techniques. However, they are more similar, both at a conceptual level and at the hardware level, than is usually observed. In discussing this issue briefly it will become apparent that the notion of pointers is less antithetical to connectionism than is often assumed.

To take the conceptual level first, we view an instantaneous state of a computer memory as a function σ from the set M of memory cells to the set B of bit-strings of the right length. Now, a function from M to B is mathematically just a special type of relation from M to B; and a relation from M to B is in turn just some set of ordered pairs (m, b) where m is in M and b is in B. Also, there is a one-to-one correspondence between M and the set A of bit-strings that can be interpreted as addresses of memory cells. (A is a subset of B.) Therefore, we may conceptually recast a memory state σ as a set of ordered pairs (a, b) where a is in A and b in B. We may further recast by replacing each ordered pair (a, b) by the concatenation ab of the strings a and b. We have therefore recast a memory state σ as an unordered set of bit-strings. Hence, the following of a pointer p held in a memory cell is to be recast as finding the unique bit-string of the form pb (for some b) in the current (recast) memory state. Viewed this way, pointer following is just a type of associative addressing. It is, of course, a very special type, in that the set of possible associative tags p is regarded as a set of consecutive integers in some range, allowing arithmetical operations on the tags (and therefore, for instance, allowing the sequential allocation technique to be used as well).

Our suspicion is that this rational reconstruction of pointers as associative addressing will strike many readers as being contorted. If it does, it should bring home the extent to which our normal views of such notions as pointers are heavily loaded with assumptions about the specific sort of low-level architecture used to *implement* the notions. We do not think of pointers just as tags of some sort, but actually as names of *small, localized parts* of the system. However, if we are in the business of comparing/contrasting connectionism with computers, and in particular in seeing in what way pointers (etc.) might or might not carry over to connectionism, we are forced to cut loose from ordinary architectural assumptions. But if we do that, we must then be much more explicit and careful about what we mean by "pointing" etc. If we stick to an abstract view such as used in Section 3 and exploited in the previous paragraph, we may conclude that as far as connectionism is concerned there is not much difference between pointers and, say, associative addressing, and then, given that associative recall is meant to be something connectionism is good at and that it is similar to associative addressing, pointing just becomes a quite natural special case of a familiar connectionist feature. If, on the other hand, we wish to include less abstract qualities in our view of pointing, we may come to a different conclusion.

The question then is, what low level aspects of computer pointing, if any, might we extrapolate to a pointing notion that is relevant to connectionism? The most radical thing to do is to extrapolate the notion of a computer memory cell as small, localized part of the computer. We then get the notion of a connectionist pointer as being a value (whether a spatial pattern, temporal signal, or spatiotemporal pattern) that identifies a small, localized part – a single unit perhaps – in the connectionist network. There is already a difficulty here, in that if the identified place is *not* just a single unit, we may be tempted to assume it is a group of units in a "localized region" of the network. But what can a "localized region" be, unless we are imagining the network as having some *physical* realization? We note, however, that most connectionist research is presented with no commitments to any particular physical layout (in the form, say, of a VLSI circuit or biological neural net). But if we then mean "localized" in some abstract sense, we are in danger of losing sight of any useful intuitive sense at all. For instance, we might say that the part pointed to must be a group of tightly inter-related units — but without further assumptions about the size of such a group and what "tightly inter-related" means, it could be an arbitrarily large subset of the set of units in the whole network.

Because of these conceptual difficulties, it seems to us best to say that *in general* a connectionist pointer would be a value that identifies some subnetwork, of whatever sort and however large, of the total connectionist system. But this does not mean that in *particular systems* there cannot be a tighter but still natural notion of pointer. One example is provided by the signatures in ROBIN. To take just one other possible instance, in a Conposit-like system a pointer could be a value that identifies a particular register in a CM. If there were more than one CM, a different type of pointer could be a value that identifies a particular, whole CM. Many other connectionist systems, completely unlike Conposit, might also allow of natural notions of pointer. The moral is, however, that the notion of pointer (when not just the very general notion of a subnetwork-identifying value) is *system relative*. Therefore, it is dangerous to discuss the role of pointers in connectionism without making the discussion relative to specific systems or classes of system.

Finally, it is worth observing that *the address-decoding circuitry used to effect pointer following* in computers is hardly ever taken into account in connectionism/computer discussions. This circuitry is a network of logic gates, operating in a highly parallel way, that can be conceptually described as converting a binary vector (the address as a bit-string) into another binary vector. The latter vector has N elements, N being the number of memory cells, and obeys the strong constraint that only one of its elements is 1. The position of this 1 identifies the addressed cell. In other words, without any conceptual contortions, we may view *the address decoder as a connectionist* *recoding network.* It follows that a connectionist could without embarrassment include in a system an address decoder, and therefore implement something directly analogous to computer pointing.

Part of the reason why points such as these are usually not made in comparisons between connectionism and computers is that the comparisons are usually "skewed" as to levels of description. When considering a connectionist system explicitly as a network of interacting units (rather than at some more abstract or functional level of description) a proper comparison with computers would be with them as *networks of logic gates and other fundamental units*, not as more abstractly described systems of unanalyzed cells having unanalyzed addresses (integers in a certain range) and containing unanalyzed values (e.g. addresses or floating point numbers). But it is the latter view that is usually used in comparisons.

7. CONCLUSION

We have put forward the relative-position encoding (RPE) and pattern-similarity association (PSA) techniques as a possible basis for short-term information structure manipulation in high-level cognitive processing by connectionist systems. As an earnest of their usefulness we have detailed the form they take in a connectionist rule-based system, Conposit, which can manipulate complex short-term information structures. We have also discussed connections, of varying strengths, that the techniques have both to techniques proposed by other connectionists and to methods used in computers. In doing this we have also dwelt upon the rich inter-relationships of these other connectionist and computer techniques. Although we have often pointed out similarities rather than contrasts, and certainly our discussion brings out the point that the distinctions between different information-structuring techniques are often far less clearcut than is normally assumed, our aim has been to provide a basis for an increased understanding of the rich space of different techniques and how one can set about representing short-term information in connectionist systems.

The matter of descriptive level is very important in this area. We have seen that when one looks at the low level circuitry need to support a given technique one sees signs of other methods. For instance, in the case of RPE one sees a type of binding-node technique, albeit an atypical one. The lesson here is *not* that one should focus on the low-level details, but rather the reverse: the interesting properties of RPE in Conposit, and the most fruitful insights about how to achieve information structuring with it, sit at the register-array level of description we used, not at the level at which it begins to look like a binding node technique. The general moral — a commonplace in computer science — is that it is often very useful to place one or more intermediate, mechanistically detailed levels of description between the high-level task and the low-level circuitry. This is *not* to say, though, that one can have a proper, full appreciation of what goes on at a given level without considering other levels, nor does it in any sense negate the possibility that the nature of the low-level circuitry places important constraints on what is practical at the higher levels.

A part or corollary of this moral is that a given technique can live at different levels of description. For instance, one could imagine RPE within a notional register array that was much more distant from, and bore a much more complex realization in, connectionist circuitry than is the case with Conposit's register array. However, many important issues would stay the same, notably the question of how to use the register array to support high-level cognitive processing. Some important things would change too, such as how the data structuring scheme is integrated with other parts of the total cognitive system.

In the Introduction we raised the central issues of unanticipatedness, arbitrariness, and variability of complexity. RPE and PSA decouple association itself from what is associated, in the sense that the RPE/PSA-based linkages operate in the first instance between component subnetworks as such, as opposed to whatever information those subnetworks currently contain. Hence, RPE and PSA are very good at dealing with the three issues — in fact they are roughly as good at dealing with them as are the computer methods which exhibit the same dissociation effect. We remind the reader that PSA and RPE as such are not to be held responsible for the rule-based Conposit's excessive sequentiality and rigidity — witness the new massively parallel, case-based reasoning version (mentioned at the beginning of Section 5) that uses straightforward RPE and pattern-similarity association much as in the rule-based Conposit, but which escapes the sequentiality and rigidity.

Similarly, the signature and time phase techniques are potentially good on the issues, because they are forms of PSA. However, it should be remembered that the signatures in the former are identifiers of specific, constant concepts (a major departure from prototypical associator patterns in PSA). And, in the time-phase technique, there is an apparent problem that will get in the way of storing short-term structures into long-term memory: whereas Barnden & Srinivas (to appear) describes a simple way of recoding Conposit CM states into connection weight settings for the purpose of long-term memory, it is more difficult to see how this could be done in a system where the PSA associator patterns have a temporal aspect. The BoltzCONS high-level associative technique is a good approach to all three issues, provided that systems with much larger symbol vocabularies are still capable of representing all or most triples defined over the vocabulary.

Absolute-positional techniques are usually poor on all three issues, precisely because of the techniques' defining feature of having associations wired in in a direct sense. It is difficult to see how to extrapolate the techniques used in our hypothetical, absolute-positional word-recognition system to deal with the representation of the meaning of complex natural language sentences, for instance.

The reduced-descriptions technique can be a useful component in a package of information-structuring techniques, as it allows the variability of complexity to be hidden. Also, the idea tends to lead to at least a limited use of register-like subnetworks and therefore tends to lead to good performance on unanticipatedness and arbitrariness.

Path-marking techniques (weight-change and binding-node techniques) are good at variability of complexity, provided that it is possible to dynamically recruit nodes to stand temporarily for complex information structures. (We include in this notion of recruitment the possibility of recruitinga register-like subnetwork by loading into it some pattern representing a complex information structure.) Path-marking techniques *can* also be good at unanticipatedness and arbitrariness, if the subnetworks bound together do not have constant representational significance, but instead can be recruited ones. This is because a binding facility needs to cope not only with binding to basic, already existing representational items (e.g. ordinary concepts of particular people), but must also with complex, dynamically constructed representational structures, such as the person-description "the person down the road who always trips over her cat when coming back from work".

ACKNOWLEDGMENTS

We are grateful to two anonymous reviewers for suggestions leading to improvements in this paper. We also benefited from communication with Venkat Ajjanagadde and Trent Lange.

REFERENCES

Barnden, J.A. (1984a). Pattern-recognition in a pattern-based neurophysiological model of short-term information-processing. In *Procs. 6th European Conference on Artificial Intelligence*, Pisa, Italy.

Barnden, J.A. (1984b). On short-term information processing in connectionist theories. *Cognition and Brain Theory*. *7* (1), 25–59.

Barnden, J.A. (1985). Diagrammatic short-term information processing by neural mechanisms. *Cognition and Brain Theory*, *7* (3&4), 285–328.

Barnden, J.A. (1987). Simulation of an array-based neural net model. In *Proceedings of the First Symposium on the Frontiers of Massively Parallel Scientific Computation*. NASA Conference Publication 2478.

Barnden, J.A. (1988). Conposit, a neural net system for high-level symbolic processing: overview of research and description of register-machine level. *Memoranda in Computer and Cognitive Science*, No. MCCS-88-145, Computing Research Laboratory, New Mexico State University,

Barnden, J.A. (1989). Neural-net implementation of complex symbol-processing in a mental model approach to syllogistic reasoning. In *Procs. 11th Int. Joint Conf. on Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann.

Barnden, J.A. (1990). Encoding complex symbolic data structures with some unusual connectionist techniques. In J.A. Barnden & J.B. Pollack (Eds.), *Advances in Connectionist and Neural Computation Theory, Vol. 1*. Norwood, N.J.: Ablex Publishing Corp.

Barnden, J.A. & Srinivas, K. (to appear). Overcoming rule-based rigidity and connectionist limitations through massively-parallel case-based reasoning. To appear in *Int. J. Man-Machine Systems*.

Barnden, J.A., Srinivas, K. & Dharmavaratha, D. (1990). Winner-take-all networks: time-based versus activation-based mechanisms for various selection goals. In *Procs. IEEE International Symposium on Circuits and Systems*, New Orleans, May 1990.

Charniak, E. & Santos, E. (1987). A connectionist context-free parser which is not context-free, but then it is not really connectionist either. In *Procs. 9th Annual Conference of the Cognitive Science Society*. Hillsdale, N.J.: Lawrence Erlbaum. A revised version appears in J.A. Barnden & J.B. Pollack (Eds.), *Advances in Connectionist and Neural Computation Theory, Vol. 1*. Norwood, N.J.: Ablex Publishing Corp.

Clossman, G. (1987). A model of categorization and learning in a connectionist broadcast-system. Ph.D. Thesis, Computer Science Dept., Indiana University, Bloomington, IN.

Eckhorn, R., Reitboeck, H.J., Arndt, M. & Dicke, P. (1989). Feature linking via stimulus-evoked oscillations: experimental results from cat visual cortex and functional implications from a network model. In *Procs. 1st Int. Joint Conf. on Neural Networks*, *Vol. I*, IEEE.

Feldman, J.A. (1982). Dynamic connections in neural networks. *Biological Cybernetics*, *46*, pp. 27–39.

Fodor, J.A. & Pylyshyn, Z.W. (1988). Connectionism and cognitive architecture: a critical analysis. In S. Pinker & J. Mehler (Eds.), *Connections and symbols*, Cambridge, Mass.: MIT Press, and Amsterdam: Elsevier. (Reprinted from *Cognition*, *28*, 1988.)

Goddard, G.V. (1980). Component properties of the memory machine: Hebb revisited. In P.W. Jusczyk & R.M. Klein (Eds), *The Nature of Thought: Essays in Honor of D. O. Hebb.* Hillsdale, N.J.: Lawrence Erlbaum.

Hendler, J.A. (1989). Marker-passing over microfeatures: towards a hybrid symbolic/connectionist model. *Cognitive Science, 13,* pp. 79–106.

Hinton, G.E. (1981). A parallel computation that assigns canonical object-based frames of reference. In *Procs. 7th Int. Joint Conf. on Artificial Intelligence,* Vancouver, British Columbia.

Hinton, G.E. (1988). Representing part-whole hierarchies in connectionist networks. In *Procs. 10th Annual Conf. of the Cognitive Science Society.* Hillsdale, N.J.: Lawrence Erlbaum.

Hinton, G.E. & Plaut, D.C. (1987). Using fast weights to deblur old memories. In *Procs. 9th Annual Conf. of the Cognitive Science Society.* Hillsdale, N.J.: Lawrence Erlbaum.

Hwang, K. & Briggs, F.A. (1984). *Computer architecture and parallel processing.* New York: McGraw-Hill.

Jensen, K. & Wirth, N. (1974). *PASCAL user manual and report.* 2nd. ed. New York: Springer-Verlag.

Johnson-Laird, P.N. (1983). *Mental models.* Harvard University Press: Cambridge, Mass.

Kohonen, T., Oja, E. & Lehtiö, P. (1989). Storage and processing of information in distributed associative memory systems. In G.E. Hinton & J.A. Anderson (Eds), *Parallel Models of Associative Memory,* Updated Ed. Hillsdale, N.J.: Lawrence Erlbaum.

Lange, T.E. & Dyer, M.G. (1989). Dynamic, non-local role bindings and inferencing in a localist network for natural language understanding. In D.S. Touretzky (Ed.), *Advances in Neural Information Processing Systems I.* San Mateo, CA: Morgan Kaufmann, pp. 545–552.

Lange, T.E. & Dyer, M.G. (in press). High-level inferencing in a connectionist network. *Connection Science, 1* (2).

Lehnert, W.G. (1990). Symbolic/subsymbolic sentence analysis: exploiting the best of two worlds. In J.A. Barnden & J.B. Pollack (Eds.), *Advances in Connectionist and Neural Computation Theory, Vol. 1.* Norwood, N.J.: Ablex Publishing Corp.

McClelland, J.L. (1986). The programmable blackboard model of reading. In J.L. McClelland, D.E. Rumelhart and the PDP Research Group, *Parallel Distributed Processing, Vol. 2.* Cambridge, Mass.: MIT Press.

McClelland, J.L. & Rumelhart, D.E. (1981). An interactive activation model of context effects in letter perception: Part 1. *Psychological Review, 88,* pp. 375–407.

Pinker, S. & Prince, A. (1988). On language and connectionism: analysis of a parallel distributed processing model of language acquisition. In S. Pinker & J. Mehler (Eds.), *Connections and symbols,* Cambridge, Mass.: MIT Press, and Amsterdam: Elsevier. (Reprinted from *Cognition, 28,* 1988.)

Pollack, J.B. (1987). Cascaded back-propagation on dynamic connectionist networks. In *Procs. 9th Annual Conf. of the Cognitive Science Soc.* Hillsdale, N.J.: Lawrence Erlbaum.

Pollack, J.B. (1988). Recursive auto-associative memory: devising compositional distributed representations. In *Procs. 10th Annual Conf. of the Cognitive Science Soc.* Hillsdale, N.J.: Lawrence Erlbaum.

Rumelhart, D.E. & McClelland, J.L. (1982). An interactive activation model of context effects in letter perception: Part 2. *Psychological Review, 89,* pp. 60–94.

Rumelhart, D.E. & McClelland, J.L. (1986). On learning the past tenses of English verbs. In J.L. McClelland, D.E. Rumelhart and the PDP Research Group, *Parallel Distributed Processing, Vol. 2.* Cambridge, Mass.: MIT Press.

Shastri, L. & Ajjanagadde, V. (1989). A connectionist system for rule-based reasoning with multi-place predicates and variables. Tech. Rep. MS-CIS-8905, Computer and Information Science Dept., University of Pennsylvania, Philadelphia, PA 19104.

Smolensky, P. (1987). On variable binding and the representation of symbolic structures in connectionist systems. Tech. Rep. CU-CS-355-87, Dept. of Computer Science and Institute of Cognitive Science, University of Colorado, Boulder, CO.

Stallings, W. (1987). *Computer organization and architecture.* New York: Macmillan.

Standish, T.A. (1980). *Data structure techniques.* Reading, Mass.: Addison-Wesley.

Sumida, R.A. & Dyer, M.G. (1989). Storing and generalizing multiple instances while maintaining knowledge-level parallelism. In *Procs. 11th Int. Joint Conf. on Artificial Intelligence.* San Mateo, CA: Morgan Kaufmann.

Touretzky, D.S. (1986). Representing and transforming recursive objects in a neural network, or "Trees *Do* Grow on Boltzmann Machines". In *Procs. IEEE Conf. on Systems, Man and Cybernetics.*

Touretzky, D.S. & Geva, S. (1987). A distributed connectionist representation for concept structures. In *Procs. 9th Annual Conf. of the Cognitive Science Society.* Hillsdale, N.J.: Lawrence Erlbaum.

Touretzky, D.S. & Hinton, G.E. (1988). A distributed connectionist production system. *Cognitive Science, 12* (3), 423–466.

Weber, S.H. (1989). A structured connectionist approach to direct inferences and figurative adjective-noun combinations. Tech. Rep. 289 (Ph.D. Thesis), Computer Science Dept., University of Rochester, NY, May 1989.

Wickelgren, W.A. (1969). Context-sensitive coding, associative memory, and serial order in (speech) behavior. *Psychological Review, 76,* pp. 1–15.

Wirth, N. (1985). *Programming in MODULA-2.* 3rd, corrected ed. New York: Springer-Verlag.

FIGURE CAPTIONS

Fig. 1: Representation of a parse in CONPARSE's array.
(Derived from Fig. 6 in Charniak & Santos, 1990).

Fig. 2: Basic illustration of Relative-Position Encoding.

Fig. 3: Basic illustration of Pattern-Similarity Association.

Fig. 4: Overall structure of simulated versions of rule-based Conposit.

Fig. 5: CM subconfigurations for "Bill hopes that John loves Mary".

Fig. 6: Split-up subconfigurations for a loving situation.

Fig. 7: Some within-register detail.

Fig. 8: A simplified view of part of BoltzCONS.

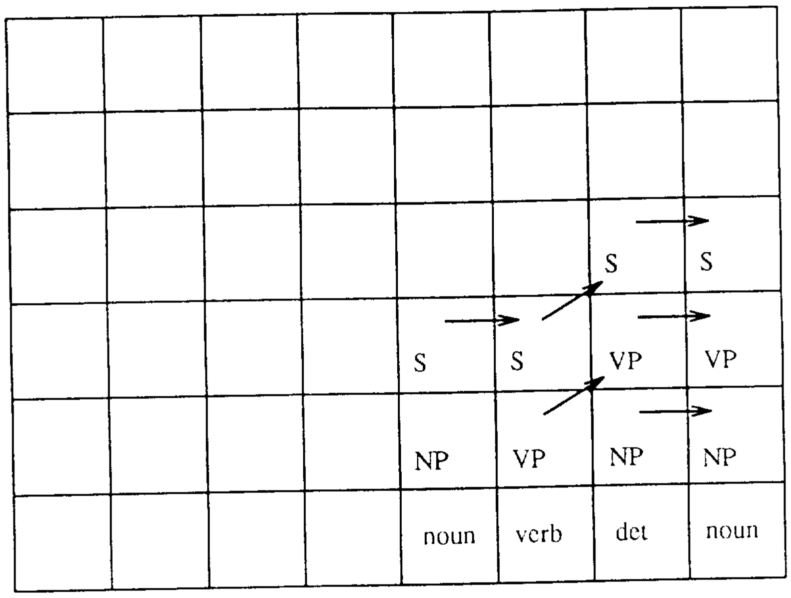

Figure 1

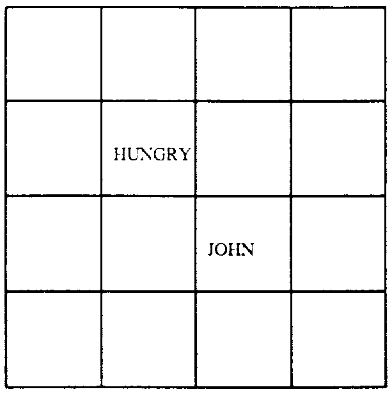

Figure 2

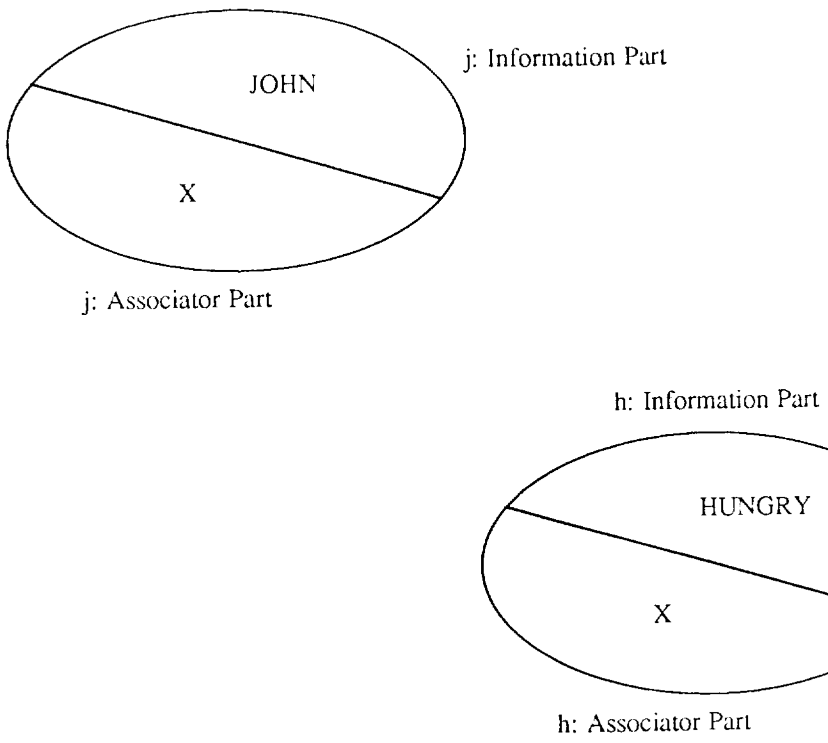
j: Information Part
JOHN
X
j: Associator Part
h: Information Part
HUNGRY
X
h: Associator Part

Figure 3

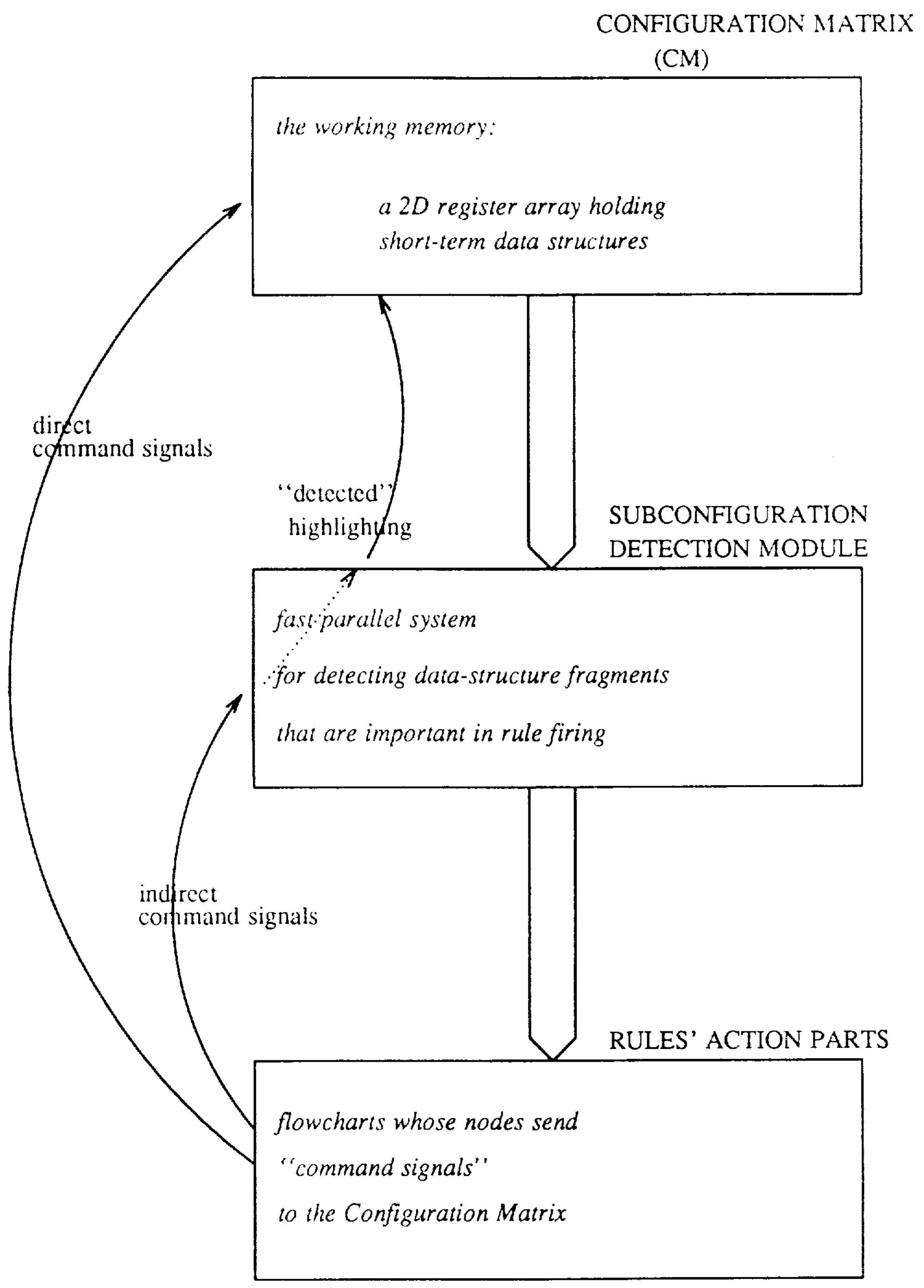
CONFIGURATION MATRIX (CM)
the working memory:
a 2D register array holding short-term data structures
direct command signals
"detected" highlighting
SUBCONFIGURATION DETECTION MODULE
fast parallel system for detecting data-structure fragments that are important in rule firing
indirect command signals
RULES' ACTION PARTS
flowcharts whose nodes send "command signals" to the Configuration Matrix

Figure 4

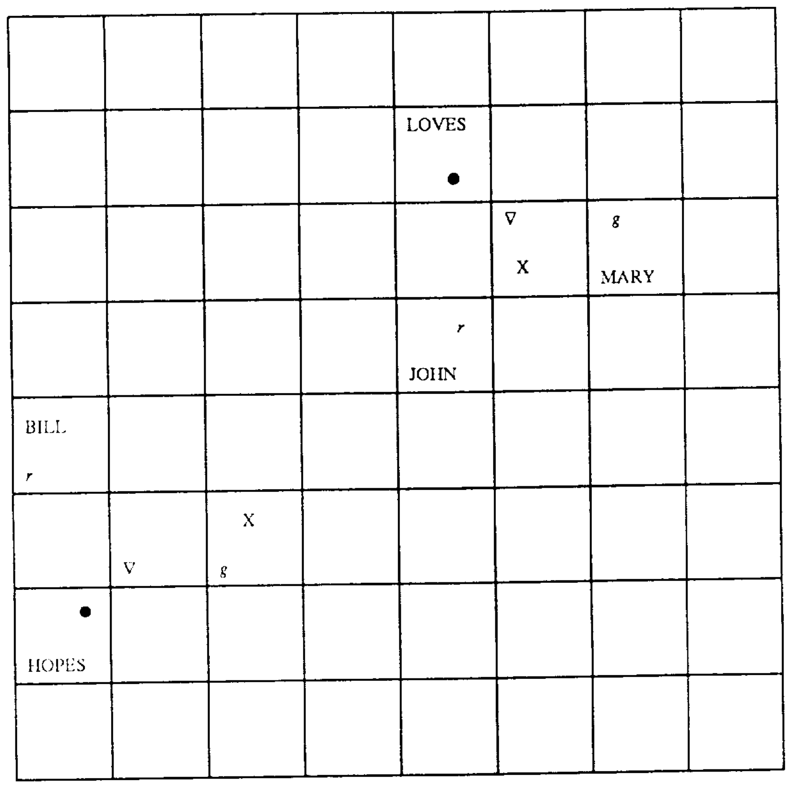

Figure 5

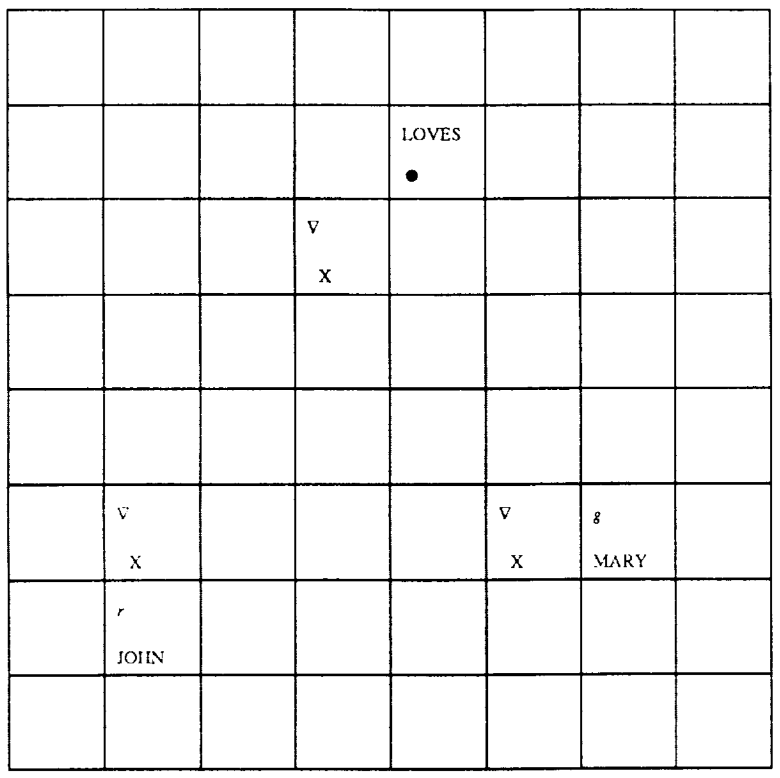

Figure 6

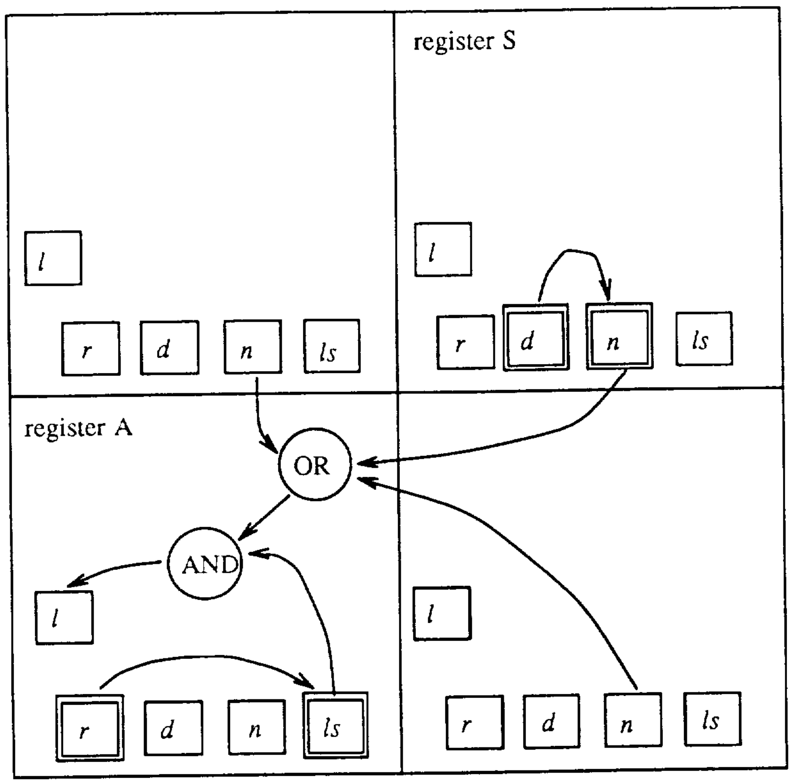

Figure 7

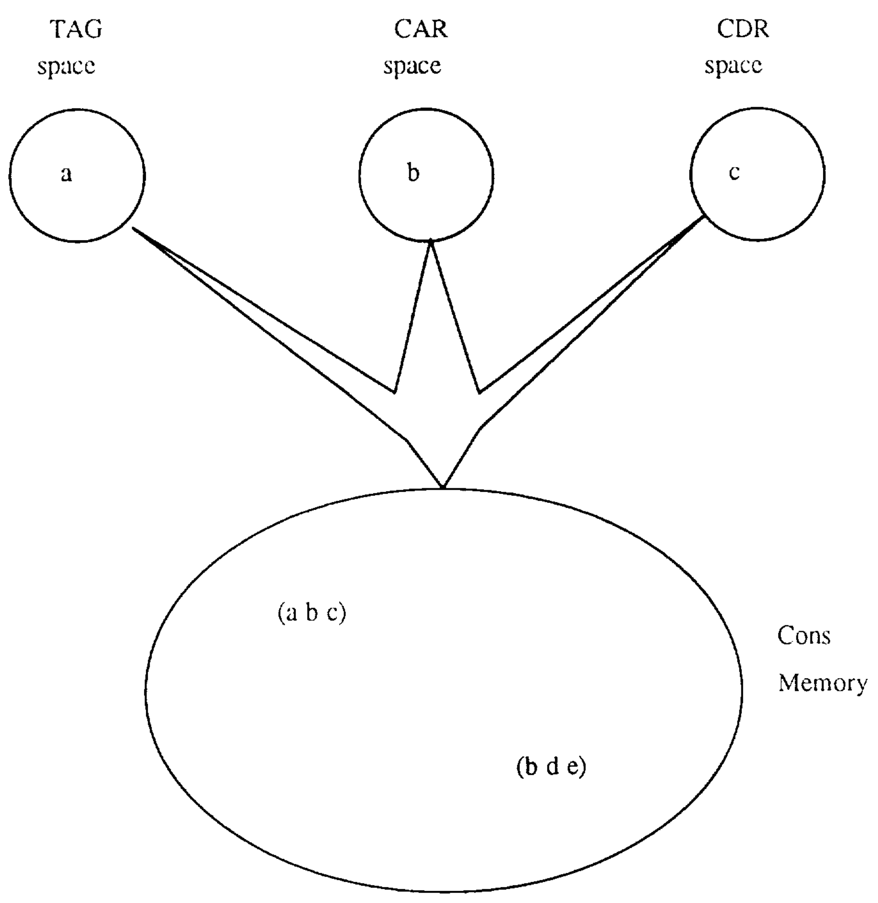

Figure 8