A Hybrid/Symbolic Connectionist Production System

Katsuaki Sanou*  Steve G. Romaniuk†  and  Lawrence O. Hall†

Department of Computer Science and Engineering
University of South Florida
Tampa, Florida 33620
phone: 813-974-4195  fax: 813-974-5456
e-mail: sanou@sol.usf.edu  romaniuk@sol.usf.edu  hall@sol.usf.edu

Abstract

The task of implementing a simple rule-based production system in terms of a hybrid symbolic/connectionist architecture is discussed in order to achieve the parallel execution of rules. The architecture, described here, uses a local representation and builds upon prior work, which resulted in the symbolic/connectionist expert system development tool SC-net. The Hybrid Symbolic/Connectionist Production System (HSC-PS) supports 2 types of working memory elements, the attribute/value pair and the object/attribute/value triplet. HSC-PS can provide variable binding in an individual condition and across conditions and variable value instantiation from the left hand side to the right hand side in a rule. The network is constructed from production rules such that only one value is bound to a variable at any one time. An example implementation of the Farmer’s Dilemma planning problem is given to illustrate the potential of this approach.

1: Introduction

The Hybrid Symbolic/Connectionist Production System (HSC-PS) is based on the network structures of the connectionist expert system tool SC-net [3, 11]. It has been pointed out that a connectionist-based production system provides advantages such as learnability, robustness, and parallelism over conventional schemes [4, 7]. It is parallelism that is aimed at in this approach. That is, the ultimate goal of this approach is to achieve the parallel execution of rules and to realize it on adequate hardware.

*on leave from Kawasaki Steel Corporation, 2-3, Uchisaiwai-cho 2-chome, Chiyoda-ku, Tokyo 100, Japan
†Research partially supported by grants from Florida High Technology and Industry Council, Computer Integrated Engineering and Manufacturing section and Software section.

In order to accomplish the goal, it is essential to be capable of representing explicit rules in terms of neuron-like computing units. The computing unit has to be primitive enough to be realized by means of one simple processor. Moreover, consistent variable bindings have to be ensured in the connectionist network. The implementation of a production system in the SC-net structure, which provides a clear "local" form of connectionist knowledge representation, is the approach adopted to satisfy the above requirements.

\[ CA_i \cdots \text{cell activation for cell } C_i, \text{ } CA_i \in \{0,1\} \]
\[ CW_{ij} \cdots \text{weight for connection between cell } C_i \text{ and } C_j, \text{ } CW_{ij} \in \text{R} \]
\[ CB_i \cdots \text{cell bias for cell } C_i, \text{ } CB_i \in \{-1, +1\} \]
\[ |CB_i| = \min_{0, \ldots, i, \ldots, n} \{CA_i \cdot CW_{ij}\}, \text{ } C_i \text{ is a min cell} \]
\[ |CB_i| = \max_{0, \ldots, i, \ldots, n} \{CA_i \cdot CW_{ij}\}, \text{ } C_i \text{ is a max cell} \]
\[ CA_i' := |CB_i| \cdot |\sum_{j=0}^{n} CA_j \cdot CW_{ij}|, \text{ } C_i \text{ is a ltc cell} \]
\[ 1 - (CA_i \cdot CW_{ij}), \text{ } C_i \text{ is a negate cell} \]

The SC-net network structure consists of simple cells modeling the fuzzy t-norm and co-norm operators min and max [8], linear threshold cells, or negate cells. We can think of every cell in a network accommodating n inputs \(I_n\) with associated weights \(CW_n\). Each cell contains a bias value, which indicates what type of fuzzy function a cell models, and its absolute value represents an upper threshold on the activation a cell can take on. Every cell \(C_i\) with a cell activation of \(CA_i\) (except for input cells) computes its new cell activation \(CA_i'\) according to the formula in Figure 1. If cell \(C_i\) (with \(CA_i\)) and cell \(C_j\) (with \(CA_j\)) are connected then the weight of the connecting link is given as \(CW_{ij}\), otherwise \(CW_{ij}=0\). The output of cell \(C_i\) is then calculated as \(O_i = \min\{1, \max\{0, CA_i'\}\}\).
2: Architecture

HSC-PS consists of four separate but connected network structures, which feed into each other. The Left Hand Attribute Layer (LHAL) is a single layer of input cells which represent the different attributes that are known to the system. The inputs to the subset consisting of these cells are provided initially by the user or from some initial working memory configuration and in subsequent Recognize/Act cycles from the Right Hand Attribute Layer (RHAL). The production rules are represented in the Encoded Rule Network (ERN), which is composed of two network structures, the Left Hand Side Layer (LHSL) and the Right Hand Side Layer (RHSL), and finally leads into the Right Hand Attribute Layer (RHAL). The RHAL duplicates the LHAL and contains the new attribute values after one forward pass through the HSC-PS network. Finally, the Working Memory Recognition Network (WMRN) detects whether the activations stored in the LHAL are identical to the ones stored in the RHAL. The Recognize/Act cycle should be terminated when such is the case.

HSC-PS supports two types of working memory elements. One is the attribute/value pair and the other is the object/attribute/value triplet. The attribute of an attribute/value pair is represented by a single cell and its activation encodes the current value of the attribute. This cell and its duplication are placed in the LHAL and the RHAL respectively. For instance, the cells A.l and A.r are placed in the LHAL and RHAL respectively for the attribute A. The activation of a cell in the LHAL, for example A.l, encodes the current value of the attribute and the activation of a cell in the RHAL, for example A.r, encodes the new value of the attribute after one forward pass through the HSC-PS network. The same principle can be applied to the representation of the object/attribute/value triplet. The attribute of an object is represented by a single cell and its activation encodes the current value of the attribute. For instance, in order to handle the attribute C of object B, the cells B.C.1 and B.C.r are placed in the LHAL and the RHAL respectively. The relation between the cell in the LHAL and its counterpart in the RHAL is the same as that in the case of the attribute/value pair.

Because the value of the attribute is encoded by the activation of the cell which represents the attribute and a cell has only one output, the attribute can take only one value at any time. One object can have an arbitrary number of attributes at the same time under the condition that for every object all the attributes which may be used during the course of the inference are provided to HSC-PS in advance. This declaration of attributes is crucial in solving the problem of variable binding and variable value instantiation.

The rule is composed of rule-name, which is optional, left-hand-side which consists of any number of conditions, and right-hand-side which also consists of any number of actions. A condition is either a pattern or relational-expression and an action is a pattern. The pattern is composed of two parts. The first is the symbol + or - and the second is either an attribute/value pair or an object/attribute/value triplet. In the left hand side, the symbol + represents the presence of the pair or the triplet and the symbol - represents its absence. In the right hand side, the symbol + represents the assertion of the pair or the triplet and the symbol - represents its retraction.

There is no limitation regarding the number of variables to be used in the left hand side and any variable that is used in the left hand side can be used in the right hand side of the same rule. In a rule, it is guaranteed that the same value is bound to the variables whose names are identical. It is also possible to use variables in relational expressions.

; the declaration of attributes
; this declaration has to precede the description of rules
( (A ( ) ) ; the declaration of an attribute/value pair
 ; a pair whose attribute is A
( (B ( ) ) ) ; the declaration of
 ; two object/attribute/value triplets
 ; a triplet whose object is B and attribute is C
 ; a triplet whose object is B and attribute is D

; the description of a rule
( (rule-1) ; the name of the rule
( (A 0.1) + (B C ?x) ) ) ; LHS
( (A ?x) + (B D 0.1) ) ) ; RHS

Given an initial working memory of (A 0.1) (B C 0.2), the final result is (A 0.2) (B C 0.2) (B D 0.1).

Figure 2 Example rule-base

Figure 2 shows an example rule-base. The general structure of the network for this rule-base is shown in Figure 3. In the LHSL, there exist three cells, A.l, B.C.1, and B.D.1 which are recruited for the attribute A, the attribute C of object B, and the attribute D of object B, respectively. And three cells, A.r, B.C.r, and B.D.r are also recruited in the RHAL. In the LHSL, two groups of cells are recruited; one is for the condition (+ (A 0.1)) into which the activation of A.l is fed and the other is for the condition (+ (B C ?x)) into which the activation of B.C.1 is fed. The outputs of the two groups of cells are fed into the min cell whose
Figure 3 Generalized network for an example rule-base

The first type of condition is the one which is composed of only one pattern. It is divided into the following three groups. Each can also be regarded as a primitive used to construct networks for conditions composed of multiple patterns. The groups are:

- conditions without variables
- conditions with one variable for a value
- conditions with one attribute variable

Conditions without variables: The simplest condition is the one without a variable. (+ (B C 0.3)) is a typical example. This pattern is matched if the value of the attribute C of object B is 0.3 and is not matched otherwise. In HSC-PS, an attribute is represented by a single cell, even if it belongs to some object, and its activation encodes the current value of the attribute. Therefore, the network structure for the pattern is shown in Figure 4. In Figure 4 and later figures, the weight "inf" means the output is multiplied with the largest representable positive number on a computer. The UK cell always takes on an activation of 0.5 and is used to provide a threshold. The input of the Itc cell from the UK cell is 0.3 because the activation of the UK cell is 0.5 and the weight of the link from the UK cell to the Itc cell is 0.6 (= 2.0 * 0.3). Hence, if the cell A1 receives an activation of 0.3 as its input, the Itc cell will take on an activation of 0 and the negate cell will be activated.

Conditions with a variable for a value: The next type is the condition with one variable for a value of hand side of a rule are developed. The most important issue that needs to be addressed here is the problem of variable binding. Since all rules will be executed in parallel in HSC-PS, given their left hand side matches the contents of some working memory elements, it is important to ensure that only one value is bound to the same variable at any one time. To ensure consistency of variable bindings, HSC-PS builds the LHSL Network in such a way that only one value can be bound to a variable at any one time. The following items have to be considered:

- conditions composed of a single pattern
- conditions composed of multiple patterns
- conflict resolution.

Each is going to be explained in a corresponding subsection below.

3.1: Conditions Composed of a Single Pattern

The first type of condition is the one which is composed of only one pattern. It is divided into the following three groups. Each can also be regarded as a primitive used to construct networks for conditions composed of multiple patterns. The groups are:

- conditions without variables
- conditions with one variable for a value
- conditions with one attribute variable.

Conditions without variables: The simplest condition is the one without a variable. (+ (B C 0.3)) is a typical example. This pattern is matched if the value of the attribute C of object B is 0.3 and is not matched otherwise. In HSC-PS, an attribute is represented by a single cell, even if it belongs to some object, and its activation encodes the current value of the attribute. Therefore, the network structure for the pattern is shown in Figure 4. In Figure 4 and later figures, the weight "inf" means the output is multiplied with the largest representable positive number on a computer. The UK cell always takes on an activation of 0.5 and is used to provide a threshold. The input of the Itc cell from the UK cell is 0.3 because the activation of the UK cell is 0.5 and the weight of the link from the UK cell to the Itc cell is 0.6 (= 2.0 * 0.3). Hence, if the cell A1 receives an activation of 0.3 as its input, the Itc cell will take on an activation of 0 and the negate cell will be activated.

Conditions with a variable for a value: The next type is the condition with one variable for a value of
Figure 4 Network structure for (+ (B C 0.3))

Figure 5 Network structure for (+ (B C ?y))

Figure 6 Network structure for (+ (A ?x 0.3))

Figure 7 Network structure for ((+ (A ?x))(+ (B C1 ?y)))

Conditions with an attribute variable: It does not occur that more than one value can be bound to the variable for a value of an attribute. However, there exist attribute variables to which more than one value can be bound. The principle problem of variable binding in HSC-PS is to ensure that only one value is bound to a variable at any one time, as noted earlier. Therefore, for the purpose of handling the problem of variable binding of this type, it is enough to select one of several possible bindings. Consider the typical example (+ (A ?x 0.3)). This pattern is matched if the object A has attributes whose value is 0.3 and requires that the particular attribute be bound to the variable ?x. That is, though the object A can have simultaneously more than one attribute whose value is 0.3, only one value must be bound to the variable ?x at the end. The significant issue is that the attributes which the object A can have are known to be, for example, B1, B2, and B3. The actual network is shown in Figure 6. For each attribute, the cells for checking whether the value of the attribute is 0.3 or not are recruited, which are surrounded by the dashed lines in Figure 6, and the result is fed into a min cell together with the value representing the attribute itself which is part of the link weight from the UK cell. By taking the maximum value of the activations of the min cells, one of the values which can be bound to the variable ?x appears as the activation of the max cell.

3.2 Conditions Composed of Multiple Patterns

Conditions composed of more than one pattern need to be considered as more general cases. If there are variables whose names are equal in different patterns within a rule, the variables impose constraints across those patterns because the same value has to be bound to them. Hence, conditions composed of multiple patterns can be divided into the following two groups: conditions without common variables and conditions with common variables.

Conditions without common variables: The simplest condition in this group is the one which includes only variables for values. A typical example is ((+ (A ?x))(+ (B C1 ?y))). This condition requires that the value of attribute A be bound to the variable ?x and the value of attribute C1 of object B be bound to the variable ?y if there exist two instantiated working memory elements for the attribute A and the attribute C1 of object B. There is no constraint between the variables ?x and ?y. Hence, the network structure for the condition is shown in Figure 7. It is constructed by merging the network for (+ (A ?x)) with the one for (+ (B C1 ?y)).
A more complicated condition is the one which includes one attribute variable and one or more variables for values. Consider the typical example \((+(A \ ?x))(+(B \ ?y \ ?z))\). This condition requires that the value of the attribute \(A\) be bound to the variable \(?x\), the attribute of object \(B\) be bound to the variable \(?y\) and its value be bound to the variable \(?z\) if there exists an instantiated working memory element for the attribute \(A\) and at least one working memory element for the attribute which belongs to the object \(B\). There is no constraint among the variables \(?x\), \(?y\), and \(?z\).

The actual network is shown in Figure 8. The significant issue is that the attributes which the object \(B\) can have are known to be, for example, \(C1\), \(C2\), and \(C3\). Thus, in this case, there can exist the following three triplets of variable bindings; \(\{?x = A.1, ?y = "C1", ?z = B.C1.1\}\), \(\{?x = A.1, ?y = "C2", ?z = B.C2.1\}\), and \(\{?x = A.1, ?y = "C3", ?z = B.C3.1\}\). One of them has to be selected because only one value can be bound to a variable at any one time in HSC-PS. The selection is performed based on the values of the attributes of object \(B\). That is, if the maximum value among \(B.C1.1\), \(B.C2.1\), and \(B.C3.1\) is \(B.C1.1\), then the first triplet is selected, and if the maximum is \(B.C2.1\), then the second triplet is selected, and otherwise, the third triplet is selected. The max cell in the middle part of Figure 8 propagates that maximum value. Only one of the three negate cells in Figure 8 takes on the activation value of 1.0 and the activation pattern of the three negate cells, that is, \(\{1, 0, 0\}\), \(\{0, 1, 0\}\), or \(\{0, 0, 1\}\) shows which triplet of variable bindings is going to be selected.

Conditions with common variables: The simplest condition in this group is the one with common variables all of which are variables for values. The example \((+(A \ ?x))(+(B \ ?z))\) shows this. This condition is matched if and only if the value of attribute \(A\) is equal to that of attribute \(C1\) of object \(B\) and also requires that the value be bound to the common variable \(?x\). The network structure for the condition is shown in Figure 9. In the network, the negate cell will only take on an activation of 1 if the output of the \(A.1\) cell is equal to that of the \(B.C1.1\) cell. By feeding the output of the negate cell together with that of the \(A.1\) cell into the min cell, it will take on an activation of the \(A.1\) cell if the negate cell is activated and 0 otherwise.

A more complicated condition is the one with common variables one of which is a variable for attribute and the rest are variables for values. A typical example is \((+(A \ ?x))(+(B \ ?y \ ?z))\). This condition is matched if there exists the working memory element for the object \(B\) whose attribute is identical to the value of the attribute \(A\). It requires that the identical value be bound to the variable \(?x\) and also that the value of that attribute of object \(B\) be bound to the variable \(?y\). In this example, the constraint is imposed by the common variable \(?x\) and more than one value can be bound to the variable \(?x\) in the pattern \((+(B \ ?x \ ?y))\) because attributes are to be bound to the variable there. However, the variable \(?x\) is also used in the condition \((+(A \ ?x))\) with a value of an attribute to be bound.

The actual network is shown in Figure 10. Here, the significant issue is that the attributes which the object \(B\) can have are known to be, for example, \(C1\),
Figure 10  Network structure for \((\text{+}(A ?x))(\text{+}(B ?x ?y)))\)

C2, and C3. Thus, in this case, there can exist the following three pairs of variable bindings; \(\{?x = \text{"C1"}, ?y = B.C1.1\}\), \(\{?x = \text{"C2"}, ?y = B.C2.1\}\), and \(\{?x = \text{"C3"}, ?y = B.C3.1\}\). One of them has to be selected because only one value can be bound to a variable at any one time in HSC-PS. The selection is performed based on the values of the attributes of object B among the pairs which satisfy the constraint imposed by the common variable \(?x\). Hence, for each attribute of object B, the cells for checking whether the corresponding attribute is equal to the value of the attribute A are recruited. They are surrounded by the dashed lines in Figure 10. In the group of cells recruited for each attribute, the attribute is specified explicitly at this point. For instance, the cells are recruited for the pattern \((\text{+}(A ?x))(\text{+}(B C1 ?y)))\) instead of \((\text{+}(A ?x))(\text{+}(B ?x ?y)))\) and they check whether the value of the attribute A is equal to \text{"C1"}. The output of each group is taken into consideration in finding the maximum value among B.C1.1, B.C2.1, and B.C3.1. If the output is 0.0, the corresponding value of the attribute of object B is regarded as 0.0 by the respective min cell in the middle part of Figure 10 instead of the actual value of that attribute. If the maximum value after considering the constraint is B.C1.1, then the first pair is selected, and if the maximum is B.C2.1, then the second pair is selected, and otherwise, the third pair is selected. The max cell in the middle part of Figure 10 propagates that maximum value. Only one of the three negate cells in Figure 10 surrounded by the dotted lines takes on the activation value of 1.0 and the activation pattern of the three negate cells shows which pair of variable bindings is going to be selected.

3.3: Conflict Resolution

In the conventional production system, the conflict resolution is done to the entire rule-base but in HSC-PS, it is done within each rule and as a result, more than one rule fires in parallel. As mentioned in previous sections, the conflict resolution in a rule proves to be the selection of one combination of variable bindings out of candidate combinations. Practically, the selection is realized by means of a max cell. The max cells which fulfill this role are the one in Figure 6, the one in the middle part of Figure 8, the one in the middle part of Figure 10 and so on. What should be noted here is that the same binding is propagated as long as the inputs of the max cell are identical. Hence, in order to make the conflict resolution work as expected, the inputs to the max cell which performs it need to be updated explicitly. In describing rules, this restriction has to be taken into consideration.

4: Right Hand Side Layer Network

In this section, the subnetworks to build the right hand side of a rule are described. The most important issue to be addressed here is the problem of variable value instantiation. The following points have to be
considered; actions without variables and actions requiring instantiated variable values. Each is explained in a corresponding subsection below.

4.1: Actions without Variables

The simplest action is one without a variable: \((+ (B C 0.3))\) is a typical example. This action requires the value of attribute \(C\) of object \(B\) to be set to \(0.3\). Figure 11 shows the HSC-PS network that implements the action. The cell \(B.C.1\) comes from the LHAL and the cell \(B.C.r\) forms the corresponding RHAL. Whenever there exist in the rule-base one or more actions which are performed on an attribute \(A_i\), one group of five cells is recruited. They are Detector-\(A_i.r\), Neg-Detector-\(A_i.r\), Instantiator-\(A_i.r\), Update-\(A_i.r\), and Default-\(A_i.r\), which are surrounded by the dashed lines in Figure 11. The Update-\(A_i.r\) cell and the Default-\(A_i.r\) cell are connected to the attribute cell \(A_i.r\) in the RHAL.

The Detector-\(A_i.r\) cell takes on an activation of 1.0 whenever some action to an attribute \(A_i\) is to be performed and 0.0 otherwise. The Neg-Detector-\(A_i.r\) cell has two inputs. One is from the Neg-Detector-\(A_i.r\) cell and the other is from the attribute cell in the LHAL, that is, \(A_i.1\). Hence, the Default-\(A_i.r\) cell propagates the negated value of the activation of the Detector-\(A_i.r\) cell. The Default-\(A_i.r\) cell has two inputs. One is from the Neg-Detector-\(A_i.r\) cell and the other is from the attribute cell in the LHAL, that is, \(A_i.1\). Hence, the Default-\(A_i.r\) cell propagates the negated value of the activation of the Detector-\(A_i.r\) cell. The Default-\(A_i.r\) cell propagates the negated value of the activation of the Detector-\(A_i.r\) cell. The Default-\(A_i.r\) cell is connected to the attribute cell \(A_i.r\) in the RHAL.

In the above example, the value the Binder-B.C.r-0.3 cell propagates has to be 0.3. This can be represented as the input from the UK cell because the activation of the UK cell is 0.5 and the weight of the link from the UK cell to the Binder cell is 0.6 \((= 2.0 \times 0.3)\).

4.2: Actions Requiring Instantiated Variable Values

The next type of action is the one with variables. This type of action requires instantiated variable values for its execution. It can be divided into the following two groups; actions with a fixed attribute and actions with a variable attribute.

Actions with a fixed attribute: In the action of this group, it is possible to uniquely determine, before the beginning of the inference, the cell to which the
action is to be performed because the attribute is explicitly specified in the action. One typical example of this group is (+ (B C ?y)). This action requires the value bound to the variable ?y be set as the value of the attribute C of object B. Since an attribute is represented by a single cell, even if it belongs to some object, and the activation of the cell encodes the current value of the attribute, it is possible to use the same network structure as in the case of actions with no variables to implement the action. Figure 12 shows this. The difference between Figure 12 and Figure 11 which is its variable-less counterpart lies in the links to the Binder cell. One comes from the Cond cell. This is common between them, but the other input is different. The one in Figure 11 comes from the UK cell and that in Figure 12 comes from the variable ?y. Because the problem of variable binding is solved in the LHSL Network, it is enough to use that result. In other words, during the construction of the LHSL Network it is necessary to relate each variable to the cell which propagates the value bound to the variable, in order that the relation can be used to instantiate the variable value.

Actions with a variable attribute: In the action of this group, because of the existence of the variable it is impossible to uniquely specify, before the beginning of the inference, the cell to which the action is to be performed. That is, the cell to which the action is to be done depends on the value bound to the variable. One typical example of this group is (+ (A ?x 0.3)). This action requires 0.3 be set as the value of the attribute of object A and at that time the attribute is specified by the value bound to the variable ?x. Hence, 0.3 is set to one of the cells for the object A in the RHAL in such a way that the attribute is equal to the value bound to the variable ?x.

The significant premise is that the attributes which the object A can have are known to be limited to, for example, B1, B2, and B3. The actual network is shown in Figure 13. For each attribute, the cells for checking whether the value bound to the variable ?x is equal to the attribute itself or not are recruited. They are surrounded by the dashed lines in Figure 13. The output of each group of the dash boxed cells is either 1 in the case that the corresponding attribute is equal to the value bound to the variable ?x or 0 otherwise. When the output is 0, for the corresponding attribute, the value of the attribute cell in the LHAL is set to its counterpart in the RHAL and when it is 1, 0.3 is the value to be set. This function is performed in the cells surrounded by the dotted lines in Figure 13. Hence the cells for the action (+ (A B1 0.3)) are recruited, but they are never activated unless the value bound to the variable ?x is "B1".

5: Example: Farmer's Dilemma

The problem of the farmer's dilemma is similar to the well-known puzzle of missionaries and cannibals [6]. The farmer's dilemma problem can be stated as follows.
A farmer is on the bank of a wide river with a fox, a goat, and a cabbage, wanting to cross it with them. There is a boat on the bank. Only the farmer can paddle it and it can only hold either the fox, the goat, or the cabbage other than the farmer. Furthermore, if left alone with the goat, the fox will eat it and if left alone with the cabbage, the goat will eat it. The problem is to find a sequence of boat trips that will get all of them across the river without the loss of anything.

This is a typical problem of state space search. Hence, the generation and deletion of states is the way to solve this problem. In this case, one state is represented as a group of object/attribute/value triplets whose attributes are identical. HSC-PS requires the declaration of all possible attributes which are going to be used during the inference. Usually it is impossible to know how many states are needed in advance in solving a problem of state space search. Therefore, in order to solve this type of problem by HSC-PS, a sufficient number of attributes which are going to represent states have to be provided. For this problem, one state is represented as a group of eight object/attributes/value triplets and eighty attributes are declared for states, that is, ten is assumed as the maximum number of states which can exist simultaneously.

Figure 14 shows a part of the rule-base. Figure 15 is a part of the simulation result for Farmer’s Dilemma.
to the variable ?state because of the function of the max cell every time the rule select-next-state fires. By retracting the matched working memory element, a different value can be bound the next time it fires, for a different value is propagated as the activation of the max cell at that time. If a rule has to fire for each value which can be bound to variables in the rule for which more than one value can be bound, an action is required. The action is usually a retraction of a matched working memory element as shown here, so that different values can be bound to the variable in different cycles.

6: Summary

In this paper, a hybrid symbolic/connectionist implementation of a production system was presented. The model uses the previously developed SC-net architecture and expands upon its basic components.

HSC-PS supports two types of working memory element; one is the attribute/value pair and the other is the object/attribute/value triplet. The attribute of an attribute/value pair is represented by a single cell and its activation encodes the current value of the attribute. The same principle is applied to the attribute of an object/attribute/value triplet. The most important issue that needs to be addressed in the construction of a connectionist production system is the problem of variable binding and variable value instantiation. HSC-PS has been able to handle a wide variety of cases by using the characteristic that an attribute can take only one value at any time and an a priori declaration of all the attributes which may be used during the course of the inference process.

HSC-PS has been able to solve some problems, but there exist several issues to be investigated. The system cannot handle effectively the problem of variable binding in some cases. For example, a condition composed of multiple patterns without common variables but with more than one attribute variable, or a condition composed of two object/attribute/value patterns in which two attribute variables and one common variable for values are used. If such cases are treated directly, the methods shown here cause the problem of combinatorial explosion, though such cases do not have to be treated directly. That is, it is possible to avoid the combinatorial explosion by realizing them in terms of the combination of the cases shown here. Another difficulty is the large number of cells needed to implement complicated production rules. There are 5,128 cells required to encode 17 rules in the rule-base of Farmer's Dilemma problem. The number of cells has already been reduced by partially sharing the network but sufficient reduction has not been reached yet. At least partial solutions to these problems are under experimentation. Owing to the lack of space, the issue of scaling will be discussed in another paper. HSC-PS provides no method for tuning a network but once a new rule is learned by some technique, it can be put in the HSC-PS network. The hardware implementation of HSC-PS is another important task because the ultimate goal of this approach is to achieve the parallel execution of rules and to realize it on adequate hardware.

In closing, this "non-conventional" connectionist model is able to implement production systems in which variable binding is necessary. The performance is illustrated by a relatively simple, but representative example.

References


53