University of Otago
Te Whare Wananga O Otago
Dunedin, New Zealand

A Model for Exploiting Associative Matching in AI Production Systems

N.K. Kasabov
S.H. Lavington
S. Lin
C. Wang

The Information Science Discussion Paper Series

Number 94 / 5
February 1994
ISSN 1172-6024

NOTE

The text of this paper was converted from an older electronic version; some diagrams that could not be converted from their original formats were scanned from the original paper document. While every effort has been made to reproduce the original paper content and layout as closely as possible, there inevitably may be some minor discrepancies or loss of quality, for which we apologise.

Any queries regarding this paper should be directed to the Discussion Paper Series Coordinator: <dps@infoscience.otago.ac.nz>

May 2005

A Model for Exploiting Associative Matching in AI Production Systems

N.K. Kasabov[1]
Department of Information Science
University of Otago

S.H. Lavington, S. Lin and C. Wang
Department of Computer Science
University of Essex

February 1994

Abstract

A Content-Addressable Model of Production Systems, `CAMPUS', has been developed. The main idea is to achieve high execution performance in production systems by exploiting the potential fine-grain data parallelism. The facts and the rules of a production system are uniformly represented as CAM tables. CAMPUS differs from other CAM-inspired models in that it is based on a non-state saving and `lazy' matching algorithm. The production system execution cycle is represented by a small number of associative search operations over the CAM tables which number does not depend, or depends slightly, on the number of the rules and the number of the facts in the production system. The model makes efficient implementation of large production systems on fast CAM possible. An experimental CAMPUS realisation of the production language CLIPS is also reported. The production systems execution time for large number of processed facts is about 1,000 times less than the corresponding CLIPS execution time on a standard computer architecture.

Keywords: Associative matching, content-addressable memory (CAM), production systems

---

[1] Address correspondence to: Dr N.K. Kasabov, Senior Lecturer, Department of Information Science, University of Otago, P.O. Box 56, Dunedin, New Zealand. Fax: +64 3 479 8311 Email: nkasabov@commerce.otago.ac.nz

1 Introduction

Production systems (PS) are widely used as a basis for the development of expert systems and for many other AI problem-solving mechanisms, as well as a model of cognition. Two major issues concerning PS have been intensively discussed in the literature:
• the functionality of PS;
• the efficiency of PS.

What concerns the first issue: PS in their symbolic form, as postulated by Post, are general purpose universal computational models equivalent to the Turing machine. It is well known that every problem which is algorithmically solvable can be realised as a PS. PS are also a natural way for representing heuristic knowledge. Production languages as OPS5 [Brownston, 1985] and CLIPS [Giarratano and Riley, 1989] were developed to facilitate building PS. Except the classical symbolic PS, new PS paradigms have more recently been developed which facilitate approximate reasoning and dealing with inexact data. In this respect, fuzzy production systems [Yager and Zadeh, 1992]; connectionist production systems [Touretzky and Hinton, 1988; Kasabov and Shishkov, 1993]; and hybrid connectionist production systems [Hendler, 1989; Kasabov, 1993; Wang, 1993] have been implemented and used.

What concerns the second issue is that many implementations of PS are found to be very inefficient for realistic volumes of information. The problems of time and space complexity of PS are crucial when they are used, for example as a deductive component of an expert database system [Kershberg 1987, 1988]. Furthermore, many real-life PS applications require very high performance. For example, speech recognition utilising thousands of facts, needs a rate of about 200,000 data changes per second [Gupta, 1987].

In this paper we represent a model called CAMPUS for symbolic PS implementation. The model is based on using associative search operations over content-addressable memory (CAM) tables. The model results in a dramatic increase in the efficiency of PS realisations in comparison to using standard PS languages on conventional computers. Experimental implementation of CAMPUS on an Intelligent File Store [Lavington, 1993] and its efficiency are also reported on briefly.

2 Production systems and their realisations

A PS consists of three main components: a set P of production rules captured in a production memory PM, a set W of facts placed in a working memory WM and an inference engine E, i.e. PS= (P,W,E). The most common symbols in a PS are constants, variables and object symbols. Every ordered sequence of an object symbol and every finite number of constants, is a fact.

A production rule Ri: Ai-> Ci consists of a left-hand side (LHS) - Ai of the rule, which is the condition part, and of a right-hand side (RHS) Ci - the actions to be executed over the working data if the conditions Ai are satisfied by the existing facts. Typical actions are: assert a fact to the WM; retract a fact from the WM; perform an input or an output operation; execute a procedure. Two exemplar productions are given below. They are written in the CLIPS syntax (CLIPS is a "C Language Interpreter of Production Systems" developed by NASA) [Giarratano and Riley, 1989]:

```
(defrule p1 this is the first production rule
;the following is a LHS, consisting of three conditions:
?CE1<-(class1 A ?x)
(class2 B ?x ?y)
?CE3<-(class3 C ?y)
=>
;the RHS contains two actions
(printout t " Data " ?y)
(retract  ?CE1 ?CE3))
(defrule p2 this is the second production rule
?CE1<-(class1 D ?x)
(class2 E ?x ?)
?CE3<-(class3 F ?x)
=>
(printout t " Info " ?x)
(retract  ?CE1 ?CE3)
(assert (class2 B ?x d))
```

The following initial facts are to be placed in the WM before the PS run:

```
(deffacts initial
(class1 A a)
(class3 C d)
(class1 D a)
(class2 E a b)
(class3 F a)
(class2 E a c)
(class2 E a d))
```

The inference engine realises a cyclic process of: determining the possible rules for application over the working data; choosing one of them, and transforming the data according to its RHS. The phases of that succession are: match- M, select- S and act- A. Therefore the

inference mechanism is specified as E= (M,S,A).

In the matching phase M, after matching all the facts and all the LHS of the productions, the satisfied rules and the collections of facts which satisfy them (called `instantiations') are added to a conflict set. The majority, (usually over 80-90 percent) of PS run-time is spent in the matching phase [Gupta 1987]. Effort has therefore been spent on devising faster matching algorithms. Popular methods for conventional computers are the RETE [Forgy, 1982] and TREAT [Miranker, 1990] algorithms. In order to decrease the number of comparison operations (which are the main ones during the matching process), the rules and facts are initially compiled into special kinds of discrimination networks. In another approach, special-purpose parallel architectures have been designed to support the above techniques. Examples are DADO [Stolfo 1984], NON-VON [Shaw 1984], PSM [Gupta 1989]. However analysis shows only a modest increase in speed: typically 10 fold [Gupta 1987]. This is because the implicit parallelism in the PS is quite limited.

Now, thanks to the new hardware technologies, opportunities to benefit from the fine grain parallelism arise. Two are directions for exploiting this parallelism. The first one is using neural networks, or as they are called, `connectionist models'. The connectionist models facilitate a massive parallelism and approximate reasoning, but the well spread neural network types can only cope with a relatively small number of working memory elements (facts). This is due either to a local representation, or to the small inherent capacity of the associatively learned patterns.

Another approach to the search for an increase in PS realisation efficiency, is that of using non-connectionist CAM [Lavington, 1988]. Building large amounts (i.e. Megabytes) of CAM is becoming cost-effective when implemented using SIMD techniques. Such active memory, in which data are stored and retrieved according to their content rather than their location addresses, is for example the Intelligent File Store IFS/2 [Lavington 1993].

Before we present the CAM based model for PS realisation (CAMPUS) we would like to summarise some discussions on PS realisations which continue to take place in the AI community:

• *State preserving versus state non-preserving matching algorithms.*

The former preserves the current state after the match phase of each execution cycle, so that when a new change of facts (WMEC) occurs, only the current state is updated and the old already-matched facts are no longer processes. Such a matching algorithm is RETE. State non-preserving algorithms are those algorithms which do not keep a track of previous cycles. They are more economical in space. An example of these is TREAT. CAMPUS is also based on the same idea.

• *`Lazy' versus `eager' matching algorithms.*
In the former, the first instantiation found for the first satisfied rule is executed immediately, while the latter saves all the instantiations from all the satisfied rules and then selects one of them to be executed. The former, has been exploited in [Miranker, 1990]. It has the obvious advantage in that the majority of the instantiations are not used at all and will not be chosen. If a strategy of `recency' is applied, i.e. the instantiation is chosen which has the most recent WME in it, it is better to apply `lazy' matching as it performs the most recent fact matching anyway. The idea of `lazy' matching has been used in CAMPUS.
• *Matching facts against rules versus matching rules against facts.*
The majority of known algorithms use the first option. The rationale is that the number of rules is usually not large (on average it is about 100) and the last changes of facts are only necessary to match against the rules. It can be assumed that there are on average two WM changes per cycle. A mixed strategy of both matchings is also possible. This is the case with the CAMPUS model presented here. First facts match a global representation of the rules, and a small set of possibly satisfied rules from this set is found. Then every rule is matched against the WM and instantiations are found, if they exist.
• *Control versus data parallelism.*
The former means a parallel matching and execution of rules and the latter means parallel matching of all the data fields. In PS which work on large WM the latter is preferable. This is the case in CAMPUS.

An important issue when implementing PS is the problem of variable binding. Variable binding process consumes the majority of the time spent on the matching phase because it involves calculating cross products between sets of values which satisfy the same variable in different condition elements within one rule (called a shared variable). Some schemes for connectionist solution of the variable binding problem are suggested in [Touretzky and Hinton, 1988; Sun, 1992; Kasabov and Shishkov, 1993]. A special representation is introduced here to ease the variable binding process.

3 Uniform representation of rules and data as associative tables

We propose a specific method for improving the PS run time performance, based on exploiting fine grain data parallelism using content-addressable tables as data structures. This way we overcome one of the main drawbacks of the von Neumann architectures - their non efficient, (for many tasks), location-addressable memory. The method allows us to exploit not only the applicable parallelism of the PS but also directly their inherent potential parallelism.

There are two main points in the development of an AI model:
• knowledge and data representation;
• reasoning.

Both points are approached here, with the associative search operations over associative tables in mind. Associative tables are used as a uniform representation for both data and knowledge. This ensures the same operations for searching in data and searching in rules, i.e. for matching data against rules and for matching rules against data.

Two levels of representation are distinguished in both data and knowledge:
• preliminary (meta, higher, constant) level;
• detailed (values, variables) level.

The first level of representation gives the `big picture' of what is in the WM and what is in the PM. It is used by the inference machine as a filter, defining which rules might be matched or whether it is worth checking them further on for a detailed, exact matching. The second level contains all the necessary detailed information about the rules and the facts. For the current implementation, we assume that because "time is more precious that space" there will be some `redundant' data structures which will be used to accelerate the matching process.

Every item (fact, rule, etc.) is uniformly represented as an n-field tuple. A parallel search on all the fields of a tuple is functionally possible regardless of the length of the tuple. Production rules are stored in two associative tables: a higher level constant table - CT, and a lower level condition elements table - CET.

The constant table - CT is built of vectors, corresponding to the rule antecedents. It contains information only for the constants in the rules. The CT for the two rules p1 and p2 is shown below:

Constant Test Table- CT

Rule	Class 1	A	D	Class 2	B	E	Class 3	C	F
1	1	1	?	1	1	?	1	1	?
2	1	?	1	1	?	1	1	?	1

Here "1" denotes that the corresponding constant takes part in the rule, "0" - that it does not appear, and "?" - that it is not important for the rule.

To represent the condition elements of the rules with their real values, wildcards and variables, a condition element table - CET is used. Every production is represented as the same number of tuples as the number of its condition elements. The format of the tuples is the same as the format of the facts in the WMET, except the first two fields which contain

the rule identifier and the CE identifier. In such a way we provide a correspondence for implementing the effective matching between the productions and the working data in both directions. The condition element table - CET - for the two rules above is:

Condition Element Table- CET

Rule-id	CE-id	CE
1	1	Class 1 A ?x 0
1	2	Class 2 B ?x ?y
1	3	Class 3 C ?y 0
2	1	Class 1 D ?x 0
2	2	Class 2 E ?x ?
2	3	Class 3 F ?x 0

The two tables CT and CET represent uniquely the set of production rules. This representation is a redundant one as the CET contains sufficient information. Another structure is also introduced for representing the knowledge base which concerns the variables in the rules. This is the binding constraint matrix (BCM). The table below shows the BCM for the given example:

Binding Constraint Matrix- BCM

Rule-id	Var-name	CE1	CE2	CE3
1	?x	3	3	0
1	?y	0	4	3
2	?x	3	3	3

One tuple from the BCM represents one shared variable in a rule. The elements of the matrix show the positions where a shared variable appear in a CE of a rule. Such a representation is very useful in the process of variable binding when state non-preserving inference algorithms is used which is the case with CAMPUS.

The facts are also represented by two structures. The higher-level one uses the format of the constant test table CT. It is a single tuple vector, which represents to some degree the current state of the WM. It consists of data flags for all the constant values which appear in the condition elements of the rules. A value of "1" means that the corresponding value for a class of objects is present in the WM. This vector will be called `constant test interrogant vector' (CTIV) because of its context. For the initial facts in the example above the CTIV will be:

Constant Test Interrogant Vector- CTIV

Class 1	A	D	Class 2	B	E	Class 3	C	F
1	1	1	1	0	1	1	1	1

The detailed-level representation of the facts is through a working-memory element table - WMET. It includes the system working data in the following form: <fact-identifier><fact-information>

For the given exemplar PS, the initial facts would be represented as 5 field tuples as follows:

WMET

f-1	Class 1	A	a	0
f-2	Class 3	C	d	0
f-3	Class 1	D	a	0
f-4	Class 2	E	a	b
f-5	Class 3	F	a	0
f-6	Class 2	E	a	0
f-7	Class 2	E	a	d

The structure of the WMET is similar to the structure of the CET. The first element is an identifier which represents the time moment when the fact is asserted in the WM (the recency). It is also a unique label of the corresponding fact. Representing facts and rules in a same (or similar format) enables directly both ways for matching. Representing knowledge and data in the same format might be also psychologically and physiologically plausible. Unfortunately, the way knowledge and data are represented in the brain is still not well known.

4 PS inference engine based on parallel associative match operations

The matching phase of the PS inference engine described in this section is based on associative search through associative tables. The matching phase consists of five sub-phases. In the first sub-phase the constant test-interrogant vector; CTIV is matched against the constant tests table; CT. As a result, a set - S1 of all possibly satisfied rules is found. In the second sub-phase the last working-memory element changes - WMEC is matched to the condition element table CET. Another set - S2 is obtained containing partially affected rules. In the third sub-phase, a set S3 of possible active rules is derived as an intersection between S1 and S2.

At the fourth stage a kind of selection is made first. One production rule **r** is chosen from the set S3 for continuing the matching process. This complies with the `recency' strategy, i.e. the most recent (the last) WMEC has been matched with the CET and possibly some satisfied rules are found. In addition to this specificity can also be imposed when a rule is chosen for further treatment. The applied strategy is close to LEX [Brownston 1985]. If for this rule an instantiation is found from the WM, the rule is fired. This is the `laziness' of the algorithm implemented in the last sub-phase of the matching algorithm. The condition elements of the selected rule **r** are used as tuples to search in the WMET for instantiations but after substituting all the already known values for the variables in r. If an instantiation for the rule **r** is found, it is executed immediately. If not, another rule r' from S3 is chosen and the fourth sub-phase is repeated. The fourth sub-phase includes also a variable binding consistency check which means that the same values should be found to satisfy shared variables in different condition element of the rule. This is done by using the BCM to check to consistency of the found instantiations.

The inference engine cycles through different phases and sub-phases at each cycle in order to process the latest working memory element change WMEC. The changes are piled on a stack-like structure. The last WMEC is placed on the top and processed first. The facts are marked: whether they have been matched against the rules or not; whether they have been deleted meanwhile; or whether they stay in the WMET. The initial state of the stack for the example above, looks as follows:

```
Stack-bottom                                   Stack-top
f-1(n) f-2(n) f-3(n) f-4(n) f-5(n) f-6(n) f-7(n)
```

where: "n" denotes not used; "u" denotes used; "d" means deleted.

Here we present the CAMPUS lazy matching algorithm in an abstract language, defined on the C- like syntax:

```
CAMPUS-inference-engine
{
 while ( not-empty( stack))
  {
   pull(stack, WMEC-id)
   if ((not-used(WMEC-id)) and (not-deleted (WMEC-id)))
      Match-lazy( WMEC-id )
  }
}
Match-lazy( WMEC )
{
 S1 = match-I( CTIV,CT )
 if (null (S1 )) return
 S2 = match-II( WMEC,CET )
 if ( null (S2 )) return
 S3 = intersect-III( S1,S2 )
 while ( not-empty( S3 ))
  {
   r = select-iV( first( S3 ))
   possible-instantiations =
       assoc-search( WMEC-affected-r-CE, WMET )
   while ( not-empty (possible-instantiations for r))
    {
     instantiation =
     consistency-check( r, BCM, first-possible-instantiation)
     if (null (instantiation))
        continue (next-possible-instantiation)
        break
    }
   if (null (instantiation))
      continue( next_rule (S3))
      {
       if ( not-executed( r,instantiation))
        {
         save (instantiation)
         act  (r with instantiation)
        }
       if (created-WMEC )
        {
         update(CTIV)
         push( stack, new-WMEC-id )
         return
        }
      }
  }
}
```

The procedure of finding consistent variable binding needs special attention. First, the dependencies between the variables in a chosen rule **r** are quickly summoned from the binding constraint matrix BCM. According to this, a unique fact-pattern is formed for the affected condition element in **r**. Fact-patterns are formed for the condition elements one at a time and serve as seeds for associative search in the WMET. The condition elements in **r** are taken one by one, in such a way as to start with the one which has the largest number of variables with the current fact-pattern. A new fact-pattern is made by substituting the variables with their values. Looking for facts which match a condition element CEi in a rule **r** is a process of searching for fact-patterns in the WMET:

```
consistency-check( r, BCM, a-possible-instantiation )
{
 rule-BCVs = fetch( r,BCM )
 CI1 = substitute( CE1,rule-BCVs,a-possible-instantiation )
 while ( not-empty( set-of-r-CEs )
  {
   CEi = take-condition( set-of-r-CEs )
   CIi = substitute( CEi,rule-BCVs,CI-set )
   new-CIi_assoc-search( CIi,WMET)
   if ( null (new-CIi))
      return
      {
       CI-set = add( new-CIi )
       continue
      }
  }
  if ( not ( null ( CI-set ))) return
}
```

The CAMPUS lazy matching algorithm presented above is on average much faster than the corresponding `eager' scheme. In the latter all the instantiations for all the rules from S3 are found and then one of them chosen for execution. In the worst case matching time for the lazy algorithm is the same as in the best case in the `eager' matching algorithm. At two points of the CAMPUS matching algorithm presented above, the 'lazy' algorithm has as an asymptotic bound the 'eager' one. The points are:

1) Selecting a rule **r** from the set S3 of possibly satisfied rules. If the extracted from the set S3 production rule **r** occurs to be satisfied, the matching is successful and the production cycle terminates. But if no instantiation for **r** is found, the matching continues with the remaining rules from the set S3 and the upper bound of this is, (the worst case), the same as when the whole set of rules is processed. This is equivalent to the `eager' variant.

2) Finding an instantiation for the rule **r**. Variables consistency check is performed on the first from the valid-for-the-moment possible instantiations and it is found to satisfy the affected rule condition elements. If this fails, the tests continues with the remaining

instantiations. Comparisons with all of the possible instantiations lead to exposure of the whole set of instantiations, which is exactly what the `eager' algorithm does.

The CAMPUS lazy matching algorithm is a general one. That is, any PS can be realised in it. If the PS potentially finds the solution to a problem, that will be achieved faster than the same solution can be achieved by applying the `eager' matching algorithm.

In order to evaluate the CAMPUS efficiency, we use average values for the most important parameters of a PS. Some statistically-obtained values are [Gupta, 1987]: number of condition elements in a rule - Nce=3; number of attributes in a CE (fields in a CE tuple) - Nattr=3; number of shared variables in a rule - Nshv=2; number of non-shared variables - Nnshv=2.

Going back to the analysis of the lazy matching generality provided in the previous section and the fact that the most time consuming operations of that algorithm are the associative (CAM) tables search operations; we can evaluate the time efficiency of the algorithm as the total number of CAM-search operations Ncam_s per PS' execution cycle for the match phase:

$$Ncam\_s = 2 + Nwmet\_s$$

where: 2 stands for the two search operations in the CT and CET respectively; Nwmet_s is the number of search operations over the WMET.

For the average statistical value of Nce=3, if the first condition element is matched instantaneously through the WMEC->CET match, and if in the second one the values for the shared variables are derived immediately by checking with the BCM, then there is only one condition element left (on average) to be processed through WMET search operations. The analysis based on the statistical values for the PS parameters as published in [Gupta, 1987] shows that the average value for Nwmet_s is Nce which gives an average value Ncam_s=5. The absolute value for the matching time depends on the time for a CAM search operation. CAM implementations are known as connectionist based [Kohonen, 1990] and as active memory store based [Lavington, 1988, 1993]. The latter have been used for an experimental implementation of the CAMPUS model. Some results are presented in the next section.

The number of WME changes per second can be evaluated by using the formula:

$$Nwmec/sec = 1/(Ncam\_s.\ Tcam\_op)$$

where: Tcam_op is an average time for a CAM search operation.

5 Experimental results: CAMPUS implementation of CLIPS on IFS/2

The problem of expressing Nwmet_s in an analytical form as a function of the PS parameters proved rather difficult. Therefore the CAMPUS model was practically implemented on an existing CAM, and absolute values for the matching time were taken and compared with the values for implementing the same PS in a standard production language. In this case it was CLIPS, on a standard platform.

An experimental implementation of CAMPUS in CLIPS syntax IFS-CLIPS was developed. It utilises a CAM called Intelligent File Store IFS/2 [Lavington, 1993]. IFS/2 has the following parameters: 27 Mbytes of semiconductor CAM, backed by 2.2 Gbytes of associatively accessed discs; time of search operation for a constant tuple is 117 microseconds; time of search operation for a tuple with one wild card is 688 microseconds; time for insertion of a tuple is 339 microseconds; time for deleting a tuple is 113 microseconds.

A synthetic PS with all the average values for the mentioned PS parameters was automatically generated. The program was run in CLIPS on SunSpark (24 MHz, 16KB) and in IFS-CLIPS (Sun 3 as a front-end machine and IFS/2 as a CAM). Figure 1 shows both times in absolute values depending on the number of initial facts in the WM. For a number of initial facts in the WM at the range of 10,000 a speed-up of about 1,000 has been evaluated. When the number of the facts in the WM is less than 100, then a CLIPS realisation would be more efficient than an IFS-CLIPS one.

For the experimented synthetic PS the executed number of WMEC per second was 6,400. For comparison, we shall mention only that a highly parallel special-purpose machine NON-VON [Hillyer and Shaw, 1986] performed 2,000 WMEC per second. More details are given in [Lin, 1991; Lavington et al, 1993].

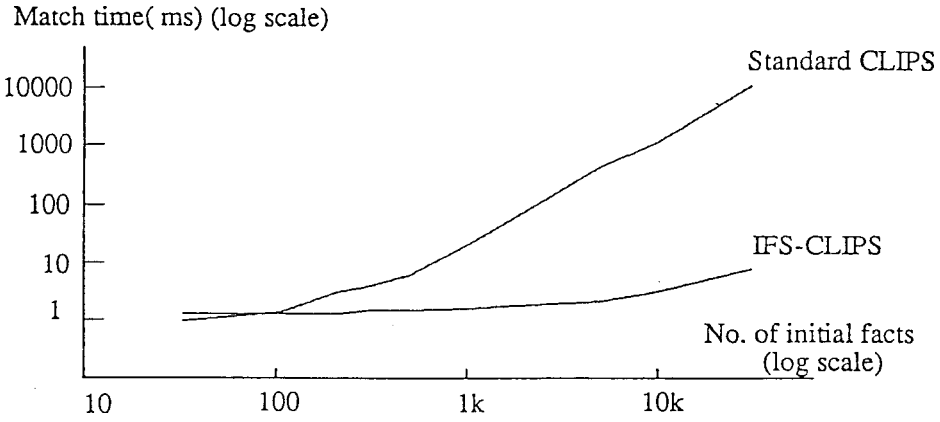

Figure 1: An average match-time per cycle for running a synthetic PS in standard CLIPS and in IFS-CLIPS

The analysis of the experimental results can be summarised by the following statements:
• the IFS-CLIPS average number of search CAM operations per cycle does not depend significantly on the number of the production rules in the PS;
• the matching time per cycle for IFS-CLIPS implementation depends only slightly on the number of the WME, but the corresponding CLIPS time strongly depends on it.

CAMPUS was also implemented to run on IBM/PC platform, where only a small number of useful CAM-operations was software simulated.

6 Towards realising PS for approximate reasoning based on partial associative matching

The CAMPUS model for implementing PS based on CAM-search operations assumes an exact match even though it was not explicitly mentioned. A fact must be 100% present if this is the case and a flag in CTIV and in CT will be set to "1". But what will happen if the facts are not required to match the production rules exactly? Say, for example, (class 1 A(0.7) ?x), that a fact from the WM would match this condition element even though the value of A might not be 100% certain. Connectionist implementation of CAM would be beneficial in this respect. File stores need some extra predicates implemented for an interval-based search, e.g. -3.5 <= A < 200.

A new research problem is to investigate how a CAM can be used for fuzzy matching. For example, imagine that A is a fuzzy value for the attribute **x** of objects which belong to class 1, and A is defined by its membership function. How should a new value of **x** be matched to A if: (a) the new value is a crisp, real number; (b) the new value is a fuzzy value?

7 Conclusions and directions for further research

CAMPUS is an attempt to look at PS from the point of view of CAM associative best-match operations, rather than from the operations realised in the von Neumann computer architecture. In this respect all the main parts of a PS - the production memory and the working memory, are represented as CAM tables. The matching process is represented as a small number of CAM search operations. The search operations are best-match ones using wild cards and variables, not only in the interrogant elements, but in those stored in CAM tuples as well. The experiments with the IFS-CLIPS implementation are encouraging for using PS as a deductive mechanism of large expert databases. A real-life PS with thousands of fact changes per second could be effectively implemented in the CAMPUS model. CAMPUS provides opportunities to exploit the finest-grain data parallelism in PS for further improvement of their time complexity. CAMPUS increases the need to develop fast CAM with more sophisticated search operations with partial match. That would facilitate building PS for approximate reasoning.

Acknowledgements

The authors would like to thank the other members of the IFS group in the Department of Computer Science at the University of Essex for their contributions to this work.

References

Brownston, I., R. Farrel, E. Kant and N.Martin (1985) Programming expert systems in OPS5: an introduction to rule-based programming, Addison-Wesley.

Forgy, C. (1982). Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem, Artificial Intelligence, vol.19, No1, pp.17- 37.

Giarratano, J. and G.Riley (1989). Expert Systems- Principles and Programming, PWS- Kent Publ. Company, Boston.

Gupta, A. (1989) High- Speed Implementations of Rule- Based Systems, ACM Trans. on Comp. Systems, vol.7, No.2, May 1989, pp.119-146.

Gupta, A. (1987). Parallelism in Production Systems, Pitman, Morgan Kaufmann Publ.

Hendler, J. (1989) On the Need for Hybrid Systems, in Connection Science, special issue on Hybrid Connectionist/Symbolic Systems, 1(3).

Hillyer, B. and Shaw, D. (1986) Execution of OPS5 Production System on Massively Parallel Machine, J.of Parallel and Distributed Computing, vol.3, No.2, June 1986.

Kandel, A. (Ed) (1991) Fuzzy Expert Systems. CRC Press.

Kasabov, N. (1993) Hybrid Connectionist Production Systems: An Approach to Realising Fuzzy Expert Systems , Journal of Systems Engineering, vol.3, No.1, pp.15-21, Springer Verlag London.

Kasabov, N. (Ed) (1993) Artificial Neural Networks and Expert Systems (Proc. of ANNES'93) IEEE Computer Society Press, Los Alamitos.

Kasabov, N., Shishkov, S. (1993) A Connectionist Production System and the Use of its Partial Match for Approximate Reasoning, Journal "Connection Science", vol. 5, Nos.3&4, pp. 275-305, Carfax Publishing Company

Kerschberg, L. (Ed) (1987). Proceedings of the First International Conference on Expert Database Systems, Benjamin/Cummings Publ.Company, Menlo Park, CA.

Kerschberg, L. (Ed) (1988). Proceedings of the Second International Conference on Expert Database Systems, Benjamin/Cummings Publ.Company, Menlo Park, CA.

Kohonen, T. (1988) Self-Organisation and Associative Memory (2nd ed.), Springer Verlag.

Lavington, S. (1988) Technical Overview of the Intelligent File Store, Knowledge-Based Systems, vol.1, No.3, pp.166-172.

Lavington, S. (1993) A novel architecture for handling persistent objects, Microprocessors and Microsystems, vol.17, No.3, April 1993, pp.131-138.

Lavington, S., C.Wang, N.Kasabov and S.Lin (1992). Hardware support for data parallelism in Production Systems, Proc. of the International Workshop on VLSI for AI and Neural Networks, Oxford, 1-2 Sept.1992, pp. A1-A12.

Lavington, S., Waite, M., Robinson, J. and Dewhurst, N. (1993) Exploiting Parallelism in Primitive Operations on Bulk Data Types: Some Results, Computers and Artificial Intelligence, vol.12, No.4, pp.313-336.

Lin, S. (1991) An Investigation of Search Parallelism in Production Systems Using Parallel Associative Hardware, M.Phil Thesis, Department of Computer Science, University of Essex.

Miranker, D. (1990). TREAT: A New and Efficient Match Algorithm for AI Production Systems, Pitman Publ., London.

Miranker, D., D.Brant,B.Lofaso,D.Gadboys (1990). On the Performance of Lazy Matching in Production Systems, Proc. National Conf. on Artificial Intelligence, AAAI- 90, 1990, pp.685- 692.

Pratibha, P. and P.Dasievicz (1990). A CAM Based Architecture for Production System Matching", Int. Conference on VLSI for AI and Neural Networks, Oxford, Kluwer Academic Press.

Stolfo, S. and D.Shaw (1982). DADO: A Tree-Structured Machine Architecture for Production Systems, Proc. of National Conference on Artificial Intelligence, AAAI-1982, Pittsburgh, Pa., Aug., 1982, pp.242-246.

Sun, R. (1992) On Variable Binding in Connectionist Networks. "Connection Science", vol.4, No2, pp.93-124.

Touretzky D. & G.Hinton (1988) A Distributed Connectionist Production System. Cognitive Science, vol.12, pp.423-466.

Wang, C., Khor,E., and Tang, S. (1993) NCLIPS - A platform for implementing hybrid expert systems, in: N.Kasabov (Ed) Artificial Neural Networks and Expert Systems, IEEE Computer Society Press, Los Alamitos, pp. 202-205.

Yager, R. and Zadeh (Eds) (1992) An Introduction to Fuzzy logic Applications in Intelligent Systems, Kluwer Academic Publishers.

University of Otago

Department of Information Science

The Department of Information Science is one of six departments that make up the Division of Commerce at the University of Otago. The department offers courses of study leading to a major in Information Science within the BCom, BA and BSc degrees. In addition to undergraduate teaching, the department is also strongly involved in postgraduate programmes leading to the MBA, MCom and PhD degrees. Research projects in software engineering and software development, information engineering and database, artificial intelligence/expert systems, geographic information systems, advanced information systems management and data communications are particularly well supported at present.

Correspondence

This paper represents work to date and may not necessarily form the basis for the authors' final conclusions relating to this topic. It is likely, however, that the paper will appear in some form in a journal or in conference proceedings in the near future. The authors would be pleased to receive correspondence in connection with any of the issues raised in this paper. Please write to the authors at the address provided at the foot of the first page.

Any other correspondence concerning the Series should be sent to:

DPS Co-ordinator
Department of Information Science
University of Otago
P O Box 56
Dunedin
NEW ZEALAND
Fax: +64 3 479 8311
email: workpapers@commerce.otago.ac.nz