

Malik GHALLAB

LABORATOIRE d'AUTOMATIQUE et d'ANALYSE des SYSTEMES du C.N.R.S.
7, avenue du Colonel Roche - 31400 TOULOUSE - France

ABSTRACT

Pattern-matching remains a very costly operation, and the main factor in the computational inefficiency of production systems. Following previous work devoted to the design of improved pattern-matchers, we propose here a method which compiles the set of production-rules conditions into a near-optimal decision tree. This approach leads to an algorithm for matching data against conditions with a sub-linear complexity.

1. INTRODUCTION

Pattern-matching is a fundamental technique in AI, but it remains a very costly operation and the main factor in the computational inefficiency of many systems, particularly Production Systems (PSs) to which the following discussion is restricted, (see [1,8] for overviews and main definitions of PSs). According to experimental data [2,5], the interpreter of some PSs spends 90 to 98 % of its time in the matching task.

A straight-forward pattern-matcher tests at each cycle the conditions of each production rule individually against all the data of the working memory (WM). Taking advantage of 1) the matchings found in the previous interpreter cycle, and 2) the occurrence of identical or similar condition elements in more than one production rule, improved algorithms were designed using either a partitioning and filtering technique [5,7], or more successfully a matching network technique [2,4]. But even this last approach suffers from some inefficiencies and leaves room for substantial improvements mainly because 1) the resulting algorithm cannot avoid redundant computations (a data element may have more than one path in the network), and 2) it relies on a network arbitrarily designed.

2. COMPILING PRODUCTION RULES CONDITIONS

The pattern-matching method herein developed compiles the set of production-rules conditions into an optimized decision tree. The nodes of this tree are tests (i.e., evaluation of characteristic features) to be performed on the data modified in WM at each cycle, the leaves are pointers to stacks containing all data actually matching a given condition. The method applies to any PS of the OPS type [3]. We first restrict segment variables (representing list fragments of undefined length) to tails of sublists, and assume that any datum in the WM matches necessarily some condition of the system.

Let us consider as an example the following 5 production rules (only rules antecedents are given ; x,y,z, are variables ; σ is a segment variable ; (D x) and

(C B σ) are negative conditions in R2) :

R1 : { (A(B C x)) (D x) (x F y) (E F y)) + ...

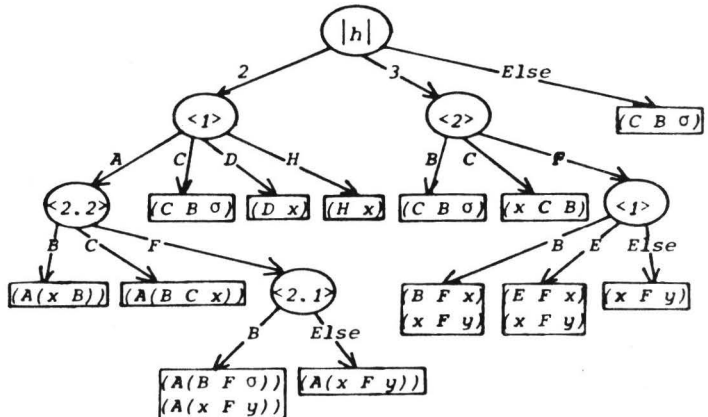
R2 : { (A(B C x)) $\neg$ (D x) (x F y) (B F y) $\neg$ (C B σ)) + ...

R3 : { (A(x F y)) (H x) (x F z)) + ...

R4 : { (A(B F σ)) (B F y) (z C B)) + ...

R5 : { (A(x B)) (x C B)) + ...

Only 11 among the 17 conditions of this example are distinct. In order to find which conditions match a given datum, it is sufficient to discriminate between these 11 conditions, since by assumption WM does not contain unnecessary data. The following decision tree achieves that discrimination :



At the beginning of each cycle, the interpreter traverses the tree with each modified datum in WM (entering and leaving data), computing at each tree node a feature and taking the branch corresponding to the value found. If for example datum $d = (A(B C(U V W)))$ is crossing the tree, the root node ($|h|$) computes the length of d at the highest level, and selects the branch labelled by the number found, namely 2. The next node ($<1>$) extracts the 1st subelement of d , finds atom 'A' and takes the left branch. The following node ($<2.2>$) extracts the 2nd element of the 2nd subelement and takes the branch labeled by the result 'C'. This will lead to a leaf containing the only condition matching d .

Note that some branches are labelled "else" : these are to be selected when the value computed by the node does not correspond to any of its other branch labels. Leaves which contain more than one condition correspond to overlapping cases : for example, any datum matching $(B F x)$ also matches $(x F y)$.

Two kinds of features are considered : those which compute the length of a list or a sublist, and those which extract an atom from some specified position. The 1st step in compiling a set of production-rules

conditions into a decision tree consists in enumerating all features of these 2 types relevant for the data at hand. An identification table which plots these features values v.s. the set of conditions, and from which non discriminating features are deleted, is then constructed.

For the previous example, the table is as follows (with '-' and 'φ' denoting undefined and variable feature values, respectively) :

	(A(B C x))	(A(x F y))	(A(B F o))	(A(x B))	(D x)	(H x)	(B F x)	(x C B)	(E F x)	(C B o)	(x F y)
h	2	2	2	2	2	3	3	3	φ	3	
2	3	3	φ	2	φ	φ	0	0	0	0	0
<1>	A	A	A	A	D	H	B	φ	E	C	φ
<2>	-	-	-	-	φ	φ	F	C	F	B	F
<2.1>	B	φ	B	φ	-	-	-	-	-	-	-
<2.2>	C	F	F	B	-	-	-	-	-	-	-

The algorithm which generates the decision tree starts by selecting among the features which are defined everywhere an "efficient" feature f as the root of the tree. From this root, a branch is drawn for each of the possible values of f including an "else" branch if f has ϕ in its row. For each branch the algorithm associates a subtable of the identification table containing the subset of conditions corresponding to the label of the branch and the features discriminating these conditions. It then proceeds recursively at each branch using the associated subtable. The recursion stops if some subtable has an empty set of features : the corresponding branch ends with a leaf containing the subtable conditions.

It is easy to show that 1) in any path from the root to a leaf of the resulting tree, a feature will not be evaluated more than once ; and 2) any datum matching some condition element has one and only one path in this tree, such a path ends with a leaf containing all the matched conditions. Thus the decision tree achieves a complete discrimination with no redundant testing. Moreover the efficiency of a decision tree pattern-matcher may be optimized. Tree optimization is a very well known problem. The previous algorithm gives only an approximate solution, whose goodness depends on the rule used for feature selection. As an example of such a rule, one may choose the feature with minimum number of variable values (ϕ) in its row, and break ties by taking the feature with the range of maximal cardinality. If the evaluation cost of each feature is known, an ϵ -admissible heuristic search is a feasible approach : for any given $\epsilon \geq 0$ it will generate an ϵ -optimal decision tree. This approach can even handle features whose evaluation costs depend on what has been previously evaluated.

Notice that the 2 classes of features considered here for designing a decision tree may be extended to other features (e.g. maximal depth or flattened length of a list). This will improve the tree efficiency in some cases, but will always increase the compiling time.

For most systems, the assumption that any datum in WM necessarily matches some condition is not a very

restrictive hypothesis, since a new datum is not an external input but rather has been generated by some rule consequent. One may check out once for all the data generated by each rule, together with the data in the initial state of the WM. In systems where this is not possible, the decision tree method is still applicable with a slight modification: verification nodes must be added just above some leaves for rejecting or confirming a candidate datum. A tree with verification nodes can also be used to handle external input data.

Providing another slight extension, the method developed here remains applicable to systems where segment variables are only restricted to be at most one per sublist : features extracting atoms from sublist in the reverse order must be considered. Thus, for example, the condition $(A \phi B)$ is characterized by : $\langle 1 \rangle =$, and $\langle 2..1 \rangle = B$.

3. PRODUCTION RULES INSTANTIATION

The decision tree enables us to identify at each cycle the conditions matching entering or leaving data in WM. A stack is associated with each condition. An entering/leaving datum is added to/deleted from the stacks of its matching conditions. Modified stacks may imply changes in the validity of the production rules containing the corresponding conditions. If the entire conflict set is required at each cycle, production-rule instantiation is mainly the Cartesian product of the concerned stacks with consistent variable bindings. Computation may be reduced by looking for groups of conditions common to many rules. Some refinements can be made to the basic method developed in [2].

More significant improvements are achieved in the rules instantiation task if, instead of generating the complete conflict set, one anticipates on the conflict resolution. It is fairly easy to check if some desired rule R_i is valid and to generate its instantiation which uses the most recent data (top of R_i 's conditions stacks).

4. CONCLUSION

In a system containing $|PM|$ production rules with $|C|$ distinct conditions, a straightforward pattern-matcher has a time complexity of $O(|PM| \times |WM|)$. Both, the filtering technique [5] and the network matching technique [2] have a worst case complexity linear in $|PM|$ or in $|C|$, and independant of $|WM|$. The decision tree approach inherits this independancy. If each node in the decision tree is implemented as a search tree, the number of operations performed to match a datum is proportional to the sum, over the features measured, of the logarithm of the number of their branches. Assuming that the decision tree has a number of leaves linear in $|C|$, in a favorable case one or few features discriminate completely all the conditions, and the complexity is of $O(\log |C|)$. Implementing each node as a hash table can even improve the best performance to a constant complexity, but this will not ameliorate the worst case in which the decision tree would have only binary features. In such a tree the average path length (over all binary trees) is of $O(|C|^{1/2})$.

In summary, the method proposed here leads to a completely discriminating pattern-matcher which exempts from any further matching, as needed by the filtering technique [5,7]. Because of its tree structure, this matching algorithm avoids the redundant computations encountered by the network

matching method [2], and has a sub-linear time complexity. Taking into account the tree optimization possibilities, the expected performances of the method developed seems more promising.

ACKNOWLEDGEMENTS

The author wishes to thank R.P. Sobek for constructive discussions, and implementation and testing of these ideas. Dr. M. Georgeff's referee comments were helpful and greatly appreciated.

REFERENCES

- [1] Davis R., King J., "An overview of production systems", In Machine Intelligence 8, Elcock and Michie Ed., Wiley, New-York, 1976, pp.300-332
- [2] Forgy C.L., "On the efficient implementation of production systems", Ph.D. Dissertation, Carnegie-Mellon Univ., Pennsylvania, Feb.1979.
- [3] Forgy C.L., "OPS4 user's manual", T.R.,C.S. Dep., Carnegie-Mellon Univ., Pennsylvania, July 1979.
- [4] Hayes-Roth F., Mostow D.J., "An automatically compilable recognition network for structured patterns", Proc. 4th IJCAI, Thillissi, 1975, pp.246-251.
- [5] Mc Dermott J., Newell A., Moore J., "The efficiency of certain production system implementations", In Pattern-Directed Inference Systems, Waterman and Hayes-Roth ed., Academic Press, New-York, 1978, pp.155-176.
- [6] Mc Dermott J., Forgy C.L., "Production system conflict resolution strategies", In Pattern-Directed Inference Systems, Waterman and Hayes-Roth ed., Academic Press, New-York, 1978, pp.177-199.
- [7] Rychener M.D., "Production systems as a programming language for artificial intelligence applications", Ph.D. Dissertation, Carnegie-Mellon Univ., Pennsylvania, Dec. 1976.
- [8] Waterman D.A., Hayes-Roth F., "An overview of pattern-directed inference systems", In Pattern-Directed Inference Systems, Waterman and Hayes-Roth ed., Academic Press, New-York, 1978, pp.3-22.