A Framework for Investigating Production System Formulations with Polynomially Bounded Match

Milind Tambe
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Paul S. Rosenbloom
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292

Abstract

Real time constraints on AI systems require guaranteeing bounds on these systems' performance. However, in the presence of sources of uncontrolled combinatorics, it is extremely difficult to guarantee such bounds on their performance. In production systems, the primary source of uncontrolled combinatorics is the production match. To eliminate these combinatorics, the unique-attribute formulation was introduced in (Tambe and Rosenbloom, 1989), which achieved a linear bound on the production match. This formulation leads to several questions: is this unique-attributes formulation the best conceivable production system formulation? In fact, are there other alternative production system formulations? If there are other formulations, how should these alternatives be compared with the unique-attribute formulation?

This paper attempts to address these questions in the context of Soar. It identifies independent dimensions along which alternative production system formulations can be specified. These dimensions are based on the fixed class of match algorithms currently employed in production systems. These dimensions create a framework for systematically generating alternative formulations. Using this framework we show that the unique-attribute formulation is the best one within the dimensions investigated. However, if a new class of match algorithms is admitted, by relaxing certain constraints, other competitor formulations emerge. The paper indicates which competitor formulations are promising and why. Although some of the concepts, such as unique-attributes, are introduced in the context of Soar, they should also be relevant to other rule-based systems.[1]

1. Introduction

Soar is an architecture for a system that is intended to be capable of general intelligence. It is based on formulating all symbolic goal-oriented behavior as search in problem spaces (Laird, Newell, and Rosenbloom, 1987). The primitive acts of the system, called *decisions*, are those required to pursue this search: the selection of problem spaces, states, and operators, plus the application of operators to states to generate new states. The information necessary for the performance of these primitive acts can be provided in one of two ways: from Soar's knowledge base, which is implemented as a production system, or by the recursive use of problem space search in subgoals. Both can result in adding new working memory elements (wmes) to the system's existing working memory. Soar learns by converting subgoal-based search into productions that generate comparable results under similar conditions (Laird, Rosenbloom, and Newell, 1986). The actions of the new productions are based on the results of the subgoals. The conditions are based on those wmes in parent goals upon which the results depended. This *chunking* process is a form of explanation-based learning (Rosenbloom and Laird, 1986).

[1]This research was sponsored by the Defense Advanced Research Projects Agency (DOD) under contract numbers F33615-87-C-1499 and N00039-86C-0033 (via subcontract from the Knowledge Systems Laboratory, Stanford University), by the National Aeronautics and Space Administration under cooperative agreement number NCC 2-538.

This paper is focused on providing Soar with an efficient and non-combinatorial (polynomially bounded) production match, particularly in the presence of continuous chunking. Production match in Soar is a key performance bottleneck. Soar's current production match is OPS5-based (Forgy, 1981), i.e., it is NP-hard (Tambe and Newell, 1988). Production match occurs at every decision in problem solving in Soar, and unpredictable combinatorial processing can occur in the match at any such decision.

This combinatorial match leads to various problems in Soar: (1) It prevents Soar from operating in real time (Newell, 1989). (2) It leads to the problem of *expensive chunks*, i.e., productions learned in the course of problem solving that can cause a severe degradation in Soar's performance (Tambe and Newell, 1988). (3) It is problematical for Soar's quest of modeling human cognition (Newell, 1990). (4) It leads to load-balancing problems in parallelization (Acharya and Tambe, 1989, Gupta, et. al., 1989, Tambe and Acharya, 1989). An efficient and bounded production match could alleviate all these problems.

In (Tambe and Rosenbloom, 1989), the *unique-attribute* formulation was introduced to guarantee an efficient and bounded production match. This formulation eliminated combinatorics from the match by trading off some expressive power. This formulation leads to several questions: is this unique-attributes formulation the best conceivable alternative to the current production system formulation? Are there any other alternative production system formulations? If there are, how should they be compared with the unique-attribute formulation?

This paper attempts to address these questions in the context of Soar. The paper identifies different dimensions along which alternative production system formulations can be specified. These dimensions are based on the fixed class of match algorithms currently employed in production systems. These dimensions create a framework for systematically identifying the different production system formulations. This paper shows how the unique-attribute formulation fits into the framework presented. Using this framework the paper shows that unique-attributes are the best possible formulation within the dimensions investigated. However, if a new class of match algorithms is admitted, other competitor formulations emerge. The paper indicates which competitor formulations are promising and why. Although these results are introduced in the context of Soar, Section 9 discusses their relevance to other systems.

All alternative formulations investigated in this paper tradeoff expressive power for production match complexity, thus providing some additional data points in understanding the general tradeoff in knowledge representation and reasoning (Levesque and Brachman, 1985, Patel-Schneider, 1989). Additionally, Soar's chunking provides a unique opportunity to gain a better understanding of how learning interacts with this tradeoff.

The paper is organized as follows: Section 2 provides an implementation-independent model of the production match, since this paper depends on a deeper understanding of production match. Section 3 presents the unique-attribute formulation. Section 4 discusses issues in evaluating alternative production system formulations. Section 5 introduces the framework for generating alternative formulations and shows how unique-attributes fit in the framework. Section 6 introduces tokenless match, the basis for the new class of match algorithms. Section 7 shows how promising new alternative formulations emerge with the introduction of tokenless match. Section 8 provides some evidence about the generality of the formulations presented here, by showing how marker passing — a formulation quite different from the standard production match — maps onto it. Section 9 outlines the contributions of this paper and its relevance to other research.

2. Modeling Soar's Production Match

The *k-search model* (Tambe and Newell, 1988) of production match covers match algorithms that find all possible solutions, without the aid of heuristics. This includes widely used match algorithms such as Rete (Forgy, 1982) and Treat (Miranker, 1987). The k-search model is based on the notion of tokens, or partial instantiations. Consider the (simplified) production **Length-3** shown in Figure 2-1-a. In the figure, up-arrow (^) indicates an attribute, and angeled bracket (<>) indicates a variable. Figure 2-1-b shows the working memory of the production system, which describes the graph in Figure 2-1-c. On the creation of the wme **(current-position A)**, the production **Length-3** will match the working memory, generating tokens, e.g., **(2; <x> = A, <z> = B)**. The first number in the token indicates the number of conditions matched and the other elements indicate the bindings for the variables. Thus, tokens indicate what conditions have matched and under what variable bindings. These tokens can be represented in the form of a *k-search* tree, as shown in Figure 2-1-d. This k-search tree represents the search conducted by the matcher, using tokens, to match the production. The result of the k-search is the four tokens (instantiations) shown at the leaves of the k-search tree. The time/token is approximately constant (Tambe and Newell, 1988). Therefore, for Soar productions, the *number of tokens* in the k-search tree is a reasonable estimate of the time spent in match.

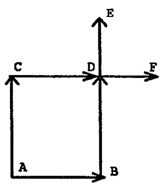
The causes of Soar's combinatorial production match can now be explained. All the wmes in Soar's production system are *a priori* candidates to match a condition, leading to a k-search tree with a number of tokens greater than $\text{wmes}^{\text{conditions}}$. However, variables bound in the conditions prior to the current condition, and constants, can provide a strong filter on the match. Soar conditions have four fields: *(class identifier attribute value)*. The class and attribute fields are constant (almost always), the identifier field is prebound (almost always) and the value field can be a constant, a prebound variable, or an unbound variable. Thus, *an unbound variable should only occur in the value field*, and multiplicity only occurs in matching a Soar condition if there are multiple possible values corresponding to the three already fixed fields, i.e., there is more than one value for an attribute. This is referred to as a *multi-attribute*. For instance, in Figure 2-1-b, **connected-to** is a multi-attribute of the objects A and D — point A is **connected-to** both B and C, while D is **connected-to** to both E and F. As shown in Figure 2-1-d, the k-search tree branches out

```
(Production Length-3

 (current-position <x>)
 (point <x>  ^connected-to <z>)
 (point <z>  ^connected-to <y>)
 (point <y>  ^connected-to <w>)
-->
 (write path of length 3 from
                    <x> to <w>))
```

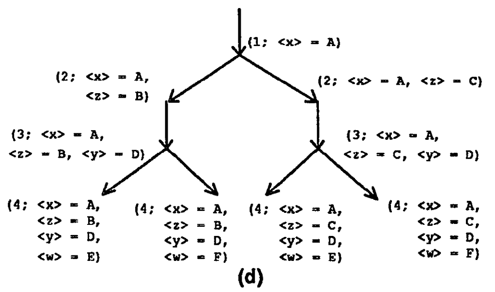
(a)

```
/** The working memory **/

(point A  ^connected-to B)
(point B  ^connected-to D)
(point A  ^connected-to C)
(point C  ^connected-to D)
(point D  ^connected-to E)
(point D  ^connected-to F)
```

(b)



(c)


(1; <x> = A)
(2; <x> = A, <z> = B)
(2; <x> = A, <z> = C)
(3; <x> = A, <z> = B, <y> = D)
(3; <x> = A, <z> = C, <y> = D)
(4; <x> = A, <z> = B, <y> = D, <w> = E)
(4; <x> = A, <z> = B, <y> = D, <w> = F)
(4; <x> = A, <z> = C, <y> = D, <w> = E)
(4; <x> = A, <z> = C, <y> = D, <w> = F)

(d)

Figure 2-1: An example production system.

in matching the multi-attributes with conditions 2 and 4. Thus, combinatorics can occur in Soar's production match only in the presence of multi-attributes. (Preferences — special control elements in Soar — can also contribute to the combinatorics. However, their impact is much smaller than multi-attributes, and they are ignored here. See (Tambe, Newell and Rosenbloom, 1990) for details.)

3. Unique-attributes

If multi-attributes are eliminated completely, the branching of the k-search tree is also eliminated. This will then limit the number of tokens in the k-search tree to the number of conditions in the production. Thus, *the cost of a production will be linear in the number of conditions*, i.e., the match cost will be bounded linearly rather than being NP-hard. For productions containing variables, this O(conditions) match bound is optimal, since all the conditions of a production must be examined for the match in any event. The new formulation is referred to as the *unique-attribute* formulation (Tambe and Rosenbloom, 1989). In contrast, Soar's current production system formulation will be henceforth referred to as the *unrestricted* formulation, i.e., a formulation with no representational restrictions. (This unrestricted formulation was referred to as the multi-attribute formulation in (Tambe and Rosenbloom, 1989, Tambe, Newell and Rosenbloom, 1990)).

The principle computational impact of going with unique-attributes is the removal of the combinatorial k-search from the match — all combinatorics will now occur as search in problem spaces. In (Tambe and Rosenbloom, 1989), detailed experiments on unique-attributes demonstrated the ability of the unique-attributes to eliminate the combinatorics from the production match, and to outperform the unrestricted Soar formulation in various tasks.

Basically, the unique-attribute formulation trades off some expressive power to gain efficiency in the production match. This loss in expressive power is not inconsequential. It manifests itself in three issues. First, multi-attributes are used in encoding unstructured sets in working memory. For instance, in Figure 2-1-b, points B and C are an unstructured set of points connected to point A. With unique-attributes, all sets in working memory have to be structured (e.g., lists), which may not always be easy (Tambe and Rosenbloom, 1989). Second, the loss of multi-attributes causes a loss in chunk generality. A much larger number of unique-attribute chunks may be required to gain the same amount of coverage as an unrestricted chunk. Until such coverage is obtained, the unique-attributes have to perform potentially expensive problem-space search. So far, in the tasks encoded in the unique-attribute formulation, these two issues have not been a major problem (Tambe and Rosenbloom, 1989).

The third implication of the unique-attributes is the inability to process arbitrary sets in a single decision, which is important in cognitive modeling tasks such as syllogisms (Polk, Newell, and Lewis, 1989). The unrestricted formulation allows this type of processing. With unique-attributes, sets are structured and have to be processed in multiple decisions (or a large number of chunks have to be learned to allow the processing to occur in a single decision) (Tambe and Rosenbloom, 1989). These three issues imply that the unique-attribute formulation is *not the best conceivable* alternative to the current formulation — its expressive power can potentially be improved.

4. Evaluating Alternative Formulations

The unique-attribute formulation is not the best conceivable formulation. But, in practice, *is it the best possible?*, i.e., is the tradeoff in unique-attributes the *best tradeoff?* This is a difficult question. As shown later, different formulations, which polynomially bound the match, restrict the production system in different ways and engage in different tradeoffs. The concept of *the best tradeoff* is imprecise.

Thus, it is impossible to devise an independent test to determine if the unique-attribute, or any other formulation is the best possible in practice, without a comparison with other formulations. That is, to determine the best, different formulations have to be compared and ranked. Even this comparison is quite difficult. However, we can enumerate a set of requirements, which will help in comparing different formulations. These requirements can be divided into *absolute* and *relative.* Absolute requirements allow formulations to be evaluated independent of each other, while relative requirements only enable comparative evaluations. The absolute requirements are:

1. *Polynomial bound on match complexity:* The match complexity of the desired representation should be polynomially bounded (in number of conditions in a production and wmes in the system).
2. *Closure under chunking:* If the productions and the working memory meet the restrictions imposed by a particular representation scheme before chunking, the chunks should also meet the restriction. If chunking violates the restrictions and creates expensive chunks, clearly that defeats the purpose of this exercise.
3. *Correctness of match:* The match should provide correct results.

The relative requirements are:

1. *Expressive adequacy:* How easy/difficult is it to encode various existing Soar tasks in a given representation? The previous section discussed this issue with respect to unique-attributes. Schemes that allow easier encoding of various Soar tasks are preferred.
2. *Relative efficiency:* Within the space of polynomially bounded formulations, schemes with smaller polynomial bounds are preferred.
3. *Chunking generality:* This refers to the number of chunks required in a particular representation to cover a given concept. A formulation providing higher chunking generality is preferred.
4. *Principle of uniformity:* Formulations that do not introduce arbitrary divisions in productions or working memory are preferred. Soar strongly adheres to the principle of uniformity in various architectural mechanisms (Newell, 1990). This requirement extends that principle to the production system formulation.

These are diverse requirements, possibly conflicting with each other. Hence these requirements have to be prioritized: the requirements are listed above in their order of priority. Given the goals of the current research, we adopt the position that the absolute requirements must be met by any candidate formulation. Even with this, the candidate formulations are not well ordered. First, the relative requirements are quite subjective. Second, there are interesting tradeoffs/interactions among these requirements, e.g., it may be possible that a formulation with an $O(n^2)$ match bound may provide better chunking generality than a formulation with a $O(n)$ match bound. Even with these shortcomings, the requirements and priorities outlined above are of great help in the search for alternative formulations.

As an example of the use of these requirements, consider a combination of unique- and multi-attributes that attempts to introduce limited amounts of multi-attributes to gain expressibility without sacrificing efficiency. First, suppose this formulation adheres to the principle of uniformity and does not explicitly separate out multi- and unique-attributes from each other. Then, it has no way of controlling the number of multi-attributes matching a production — it is exactly like the unrestricted Soar system. The match cost of productions becomes unpredictable and the polynomial match bound requirement is violated.

Now, suppose the violation of the principle of uniformity is accepted: the system explicitly labels and separates unique- and multi-attributes. It can then bound the number of multi-attribute-matching conditions in any single production — thus controlling the match cost. However, it is possible for such a system to create chunks where the number of multi-attribute-matching conditions in the production exceeds the specified bound — generating expensive chunks. This violates the requirement of closure under chunking. Thus, this unique- and multi-attribute combination does not work.

Note that in this paper, schemes that impose arbitrary time-based cutoffs on the match or those that require Soar to chunk selectively, are rejected as possibilities. Such schemes do not integrate well with the rest of the Soar architecture, i.e., they do not conform with the assumptions underlying the Soar architecture (Tambe, Newell and Rosenbloom, 1990).

5. Formulations Based on Existing Match Algorithms

The previous section showed that the combination of unique- plus multi-attributes does not meet all the absolute and relative requirements. Generating a formulation that satisfies all the requirements is difficult — there is no method that, given the requirements, will directly provide a formulation satisfying them. Combined with the need for comparing different formulations to determine the best among them, this situation dictates a strategy for *exhaustively* searching the space of alternative formulations; then for each formulation, testing if it meets the requirements, and then comparing its properties with other formulations to test if it is better.

Although part of the search has already been conducted (Section 3 and 4), how should the exhaustive search proceed further? The problem here is the absence of a device to systematically generate required alternatives. Without such a device, we cannot understand and usefully exploit the structure of the space of alternative formulations. Therefore, this paper introduces a framework for systematically generating alternative production system formulations. To specify the framework, it is necessary to first identify the independent dimensions of this framework. Roughly, these dimensions can be divided into two categories: (1) those that do not require any modification to the current set of token-based match algorithms (introduced in Section 2) (2) those that require some modification. Section 6 introduces dimensions from the second category. This section introduces dimensions only from the first category. These dimensions are based on restrictions on Soar's working memory format: **(class identifier attribute value)**. They are:

1. *Forward-attribute*: Given a fixed identifier, this dimension imposes restrictions on the number of attributes for that identifier. For example, suppose the number of attributes is restricted to one. If **B1** is an identifier, then **(class1 B1 attribute-1 val1)** and **(class1 B1 attribute-2 val2)** are not allowed simultaneously to exist in working memory.
2. *Values-per-attribute*: Given a fixed identifier, this dimension imposes restrictions on the number of values-per-attribute for that identifier. If the number of values-per-attribute is restricted to one, the unique-attributes formulation is obtained. Multi-attributes refer to multiple values-per-attribute.
3. *Reverse-attribute*: Given a fixed value, this dimension imposes restrictions on the number of attributes for that value. This is symmetrical to the forward-attribute dimension. Thus, if **val1** is a value, then **(class1 B1 attribute-1 val1)** and **(class1 B2 attribute-2 val1)** are not allowed simultaneously to exist in working memory, as **attribute-1** and **attribute-2** are two different attributes with the same value **val1**.
4. *Identifiers-per-attribute*: Given a fixed value, what are the restrictions on the number of identifiers-per-attribute for that value. This is symmetrical to the values-per-attribute dimension.

(The class slot in Soar's wmes does not serve any semantic role. Hence the paper does not refer to the class slot.) Figure 5-1 shows the four dimensions in a tabular format. There are two co-ordinates along each of the four dimensions — 1 and *. Here, an asterisk (*) refers to an arbitrary number of values, i.e., no restrictions on the values. In the figure, to lay out these four dimensions in two dimensions, the forward-attribute and values-per-attribute dimensions are paired. Similarly, the reverse-attribute and identifiers-per-attribute dimensions are paired. The figure shows that different combinations of co-ordinates along these dimensions identify different working memory representations.

The choice of 1 and * for the co-ordinates is an interesting issue. This choice of co-ordinates covers both the unique-attribute and the unrestricted working memory and yields semantically meaningful representations like the *Tree* representation (introduced below). Furthermore, if the number of values-per-attribute are increased to two or more, match becomes combinatoric and does not admit the linear bound of unique-attributes. However, whether no other co-ordinates besides 1 and * are useful, remains unclear.

In the unrestricted working memory representation, with a fixed identifier, it is possible to have an arbitrary number of forward-attributes and values-per-attribute. Similarly, with a fixed value, it is possible to have an arbitrary number of reverse-attributes and identifiers-per-attribute. Therefore, the unrestricted working memory occupies the square in the center of the table, where all four dimensions have a value of *. For unique-attributes, the restriction is only on the values-per-attribute — for a fixed identifier, there can be only one value-per-attribute. Its other dimensions are unrestricted and take the value of *.

<ua	Indicates more restrictive than unique-attributes
<ui	Indicates more restrictive than unique-identifiers
*	Indicates an arbitrary number
<ts	Indicates more restrictive than the tree-structures
→	Indicates increasing restrictiveness

		(forward-attribute, values-per-attribute)				
		(1, 1)	(*, 1)	(*, *) [Multi-Attr]	(1, *) [Multi-Attr]	(1, 1)
(reverse-attribute, identifiers-per-attribute)	(1, 1)	<ua <ui <ts	<ua <ui <ts	Tree <ui	<ui <ts	<ua <ui <ts
	(*, 1)	<ua <ui	<ua <ui	Unique-ident	<ui	<ua <ui
	(*, *)	<ua	Unique-Attr	Current Unrestr Represnt		<ua
	(1, *)	<ua	<ua			<ua
	(1, 1)	<ua <ui <ts	<ua <ui <ts	<ui Tree	<ui <ts	<ua <ui <ts

Figure 5-1: Dimensions of alternative representations.

In Figure 5-1, the square marked as *Tree* restricts the reverse-attributes and identifiers-per-attributes to one and leaves the other dimensions unrestricted. This representation implies that two different working memory elements cannot have the same value in their value fields. More semantically, this representation corresponds to a tree-structured organization of the working memory. The square marked as *unique-identifiers* restricts the identifiers-per-attribute to one, but does not restrict any other dimension. The table shows that the Tree and unique-identifiers allow an arbitrary number of values-per-attribute, i.e., they allow multi-attributes. In fact, all the working memory representations in two of the columns allow multi-attributes; these columns are labeled with **[Multi-attr]**.

The table shows the unrestricted working memory as the least restrictive form of representation. It occupies the center square in the table. The other representations form a restriction lattice. The first column and the first row are repeated to show the symmetry in this lattice. The most restrictive working memory representation is the one where all the dimensions have the value 1. The table also shows that the Tree structures are more restrictive than the unique-identifiers; but unique-attributes are unrelated in terms of restrictions to either one of these.

The important conclusion that can be made from the table is: *unique-attributes provide the best possible formulation within the four dimensions investigated.* That is, the unique-attributes provide the best fit to all the absolute and relative requirements. All other formulations are either combinatoric, so that they violate the absolute requirement of a polynomial match bound; or they are more restrictive than the unique-attributes, so that unique-attributes fit the relative requirements better. First, consider the formulations in the third and fourth columns of the table. They include the Tree and unique-identifiers, which restrict the identifiers-per-attribute. These representations allow multi-attributes. As described in Section 2, the match in the presence of multi-attributes is combinatoric: thus, all these formulations violate the polynomial match bound requirement.

Second, consider the formulations in the first and second columns of the table. This includes the unique-attributes and the formulations more restrictive than the unique-attributes. The formulations that are more restrictive than the unique-attributes do not reduce the match bound: it is optimal to begin with. Furthermore, these formulations are guaranteed to not be better than unique-attributes along the expressive adequacy or chunking generality requirements, since they are more restrictive.

More generally, we can also conclude that if a given formulation meets all the absolute requirements, *then more restrictive formulations, which do not reduce the match bound, need not be investigated.* If the more restrictive formulations possess the same match bound as the given formulation, only the remaining relative requirements like expressive adequacy, choose the better among them. However, the more restrictive formulations are guaranteed to *not* meet these remaining relative requirements better than the given formulation.

Thus, the unique-attribute formulation is the best possible one in practice, within the four dimensions investigated. The general conclusions drawn here illustrate the power of the framework: the conclusions exploit the structure of the search space made explicit by the framework. Thus, the need for detailed evaluation of each formulation is eliminated.

6. Tokenless Match: A New Match Scheme

The formulations introduced in the previous section were based on the the token-based match scheme (henceforth called token match). The conclusion about unique-attributes being the best among these formulations is based on this token match. In a token match, a single token indicates what variable bindings go together. For example, the token **(2; <x> = A, <z> = C)** from Figure 2-1-d indicates that the binding A for <x> and C for <z> go together. The outcome of the match is a set of instantiations, indicating which bindings go together.

If a *tokenless match* is allowed, then new formulations emerge: the unique-attribute formulation is no longer guaranteed to be the best. In *a tokenless match*, each variable obtains a list of bindings, *independent* of the bindings of other variables. The outcome of this match is a set of bindings for each variable, rather than the separate instantiations. If the production **Length-3** in Figure 2-1-a is matched with the working memory from Figure 2-1-b, the result is a set of bindings for the variables as follows:**<x> = A; <z> = B, C; <y> = D; <w> = E, F.** The matcher guarantees that these bindings are consistent with each other; however, it does not explicitly create tokens and instantiations. This consistency requirement is explained in the subsection below with the help of a mapping. This mapping also provides ready-made algorithms to perform tokenless match.

6.1. Tokenless Match and Constraint Satisfaction

A constraint-satisfaction problem is defined as follows: given a set of N variables each with an associated domain and a set of binary constraining relations between the variables, find all possible N-tuples such that each N-tuple is an instantiation of the N variables satisfying the constraining relations (Mackworth and Freuder, 1985). This problem can be represented as a *constraint-graph* where the variables are represented by nodes and the constraints by arcs. Each constraint specifies the set of permitted pairs of values for the two variables involved. Thus, if X_i and X_j are two variables with domains D_i and D_j respectively, then the constraint R_{ij} between X_i and X_j is a subset of the cartesian product of their domains, i.e., $R_{ij} \subseteq D_i \times D_j$

The match for a single Soar production maps on to the constraint satisfaction problem as follows. The variables in the conditions form the variables in the constraint satisfaction problem. For example, the production in Figure 2-1-a can be represented as the constraint network in Figure 6-1. The symbols in working memory, i.e., symbols occupying the identifier and value fields of working memory elements, form the domains of the variables. A condition containing two variables is a constraint between the two variables. The condition specifies (or selects) the wmes with its attribute, so that each wme represents a permitted pair of values for the variables linked by the condition. If a condition contains A1 as an attribute, it specifies all the wmes with the attribute A1. Thus, the conditions from Figure 6-1 with the attribute **connected-to** specify all the wmes from Figure 2-1-b. Finding all possible solutions of the constraint satisfaction problem formed by a production will result in finding all possible instantiations of the production.

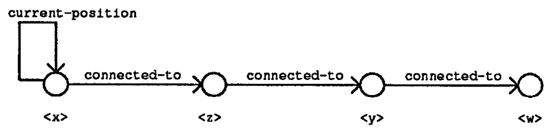


Figure 6-1: Mapping tokenless match to constraint satisfaction.

The constraint satisfaction literature distinguishes between obtaining consistent bindings and forming an instantiation (Dechter and Pearl, 1988). Once consistent bindings are obtained, individual instantiations from the bindings are obtained separately. The tokenless match achieves a similar consistent set of bindings, without forming any instantiations. Thus, there is a mapping between the tokenless match and obtaining consistent bindings in constraint satisfaction problems.

6.2. Implications of the Mapping

The mapping between the tokenless match and constraint satisfaction problems provides an important notion: the relation between the structure of a production and the effort required in obtaining consistent bindings for that production. The structure of the production refers to the equality tests across the value fields of the production's conditions. For instance, if a Soar production admits no equality tests between two variables in the value fields of its conditions, then the production has a tree structure. (To understand why such productions are tree-structured, consider a graph where the variables in the identifier and value fields of the production's conditions are nodes; and the attributes in the conditions are links between the nodes. This graph is tree-structured.) Figure 2-1-a presents one such production without any equality tests across its value fields. To demonstrate how the production structure can be exploited, we need the concept of *arc consistency* in constraint-satisfaction problems.

In constraint-satisfaction problems, arc-consistency is a form of local consistency. Arc-consistency does not solve the general constraint-satisfaction problem. A constraint graph is arc-consistent if each of its arcs is arc-consistent. An arc between variables X_i and X_j is arc-consistent iff for any value $x \in D_i$

there is a value $y \in D_j$ such that $R_{ij}(x, y)$ (Mackworth, 1977). Here $R_{ij}(x, y)$ stands for the assertion that (x, y) is permitted by the explicit constraint R_{ij}. We add the following small symmetric requirement to the situation above: if there is a value $y \in D_j$ then there is a value $x \in D_i$ such that $R_{ij}(x, y)$.

In terms of the production match mapping, the following has to be satisfied for a condition with an attribute A1 to be arc consistent: If there is a binding x for the variable <x> in its identifier field, then there is a binding y for its variable <y> in its value field, such that there exists a wme with attribute A1, identifier x and value y. Symmetrically, if there is a binding y for its variable <y> in the condition's value field, then there exists a binding x for the variable <x> in its identifier field such that there exists a wme with attribute A1, identifier x and value y.

Arc consistency is used along with the production structure in the following result based on an important result from (Dechter and Pearl, 1988):

*If a production is tree structured, and it is made arc-consistent, then the bindings obtained for its variables are consistent. Furthermore, this arc-consistency can be achieved in O(wmes*conditions).*

Here, wmes refers to the number of working memory elements. Thus, if productions are tree-structured, then a tokenless match can be obtained in polynomial time, i.e., the match guarantees consistent bindings for the variables in the production using arc consistency. Note that the tokenless match is important in achieving this bound. A token match, despite tree structured productions, is still exponential — $O(\text{wmes}^{\text{conditions}})$ (see section 4 of (Tambe and Rosenbloom, 1989) for a demonstration of this effect).

Note, however, that if a production is not tree structured, i.e., it has an equality test across the value fields of its conditions, arc consistency may provide a wrong result. That is, it might provide a binding for a variable, when a token match for the same production would have provided none (Tambe and Rosenbloom, 1990). Thus, there is a tradeoff in the complexity of the tokenless match and the restrictions on the equality tests. By restricting the structure of the production in specific ways, different (and increasingly complex) bounds on the tokenless match can be obtained (Dechter and Pearl, 1988). With this result, the structure of the productions emerges as an important dimension. *Thus, two new dimensions are now available in the framework: tokenless match and production structure.*

7. Tokenless Match: New Formulations Emerge

The token/tokenless match and production structure dimensions, along with the four dimensions introduced previously (see Figure 5-1), provide us with a total of six dimensions. From this expanded space, new formulations emerge, and unique-attributes are no longer guaranteed to be the best fit to the absolute and relative requirements. To illustrate this, consider a specific formulation, called the *unrestricted-tree formulation*, that works with an unrestricted working memory representation (from Figure 5-1) and a tokenless match, but restricts productions to be tree-structured. This formulation satisfies all of the absolute requirements. First, polynomial match bound is guaranteed because of the use of a tokenless match and tree-structured productions. As noted in Section 6.2, this combination yields polynomial match bound — O(wmes*conditions) — irrespective of the representation used in working memory. Second, with a small modification to chunking — disallowing introduction of new equality tests across value fields of conditions — this formulation provides closure under chunking. That is, only tree-structured productions will be chunked. Third, due to the consistency requirements imposed, this formulation provides correct results. Therefore, all the absolute requirements are satisfied.

When the relative requirements are compared for the unrestricted-tree and unique-attribute formulations, it becomes clear that neither formulation dominates the other, and hence neither can be pruned *a priori*. In particular, in terms of expressive adequacy, in the unrestricted-tree formulation, working memory restrictions are absent (compared to unique-attributes); however, since equality tests across value fields of conditions are not available, productions have to be written in a fairly different manner. (If such tests become absolutely necessary, productions can directly test constants.) Therefore, we need to implement a set of tasks with this new formulation, and compare its performance with unique-attributes. Initial analysis seems to indicate some promising results for the unrestricted-tree formulation. For example, in the grid task (Tambe and Rosenbloom, 1989), the unrestricted-tree formulation should be able to outperform the unique-attribute and unrestricted formulations. The unrestricted-tree formulation would provide chunks with polynomial match cost, where similar chunks in the unrestricted formulation require an exponential match cost (see Section 4 of (Tambe and Rosenbloom, 1989)); but simultaneously it would avoid the loss in chunk generality that afflicts the unique-attributes (Tambe and Rosenbloom, 1990). The work of implementing tasks in this new formulation is currently in progress and remains a key issue for future work.

Besides the unrestricted-tree formulation, other formulations based on tokenless match are also possible (Tambe and Rosenbloom, 1990). For instance, by adopting consistency algorithms more constraining than arc consistency (Dechter and Pearl, 1988), it is possible to introduce limited equality tests across value fields of conditions of productions. Such new formulations may perform better along the expressive adequacy dimension than the unrestricted-tree formulation, but require a higher polynomial bound. Other formulations that are based on the tree and unique-identifier working memory representations from Figure 5-1 are also possible (Tambe and Rosenbloom, 1990). All these formulations are competitors for the unique-attribute formulation. Further investigations of these new formulations is another key issue for future work.

8. Relationship to Marker Passing Systems

This section develops a mapping between the unrestricted-tree formulation and marker passing systems. This mapping illustrates the generality of the framework presented in this paper for alternative production system formulations — it manages to capture a fairly different production system formulation. This mapping is illustrated using the NETL parallel marker passing system (Fahlman, 1979).

Conceptually, NETL is composed of a large collection of object nodes, like semantic nets, and a large collection of bi-directional links or arcs organized so that each object node can be linked arbitrarily to any number of other nodes in the system. Markers (signals) may be propagated along different links in parallel. Queries are posed to the NETL system, and answers are retrieved by passing markers between nodes. (No cancellation links are assumed, hence the problems from (Touretzky, 1986) associated with marker passing systems are not relevant here.)

The mapping onto NETL relies on describing each symbol in the working memory as an object node of NETL. The attributes in working memory are links between nodes. The production to be matched is the query to the NETL system. Match is performed by passing markers between symbols. Matching a single condition is equivalent to sending a marker from a symbol bound to the identifier field of the condition (since the identifier of a condition is bound before matching the condition) to the symbols linked via the attribute in the condition. The result of passing this marker is a set of bindings in the value field of the condition. This allows bindings to be obtained for the next condition via marker passing. Again, the result of the match is a set of variable bindings, without the token information.

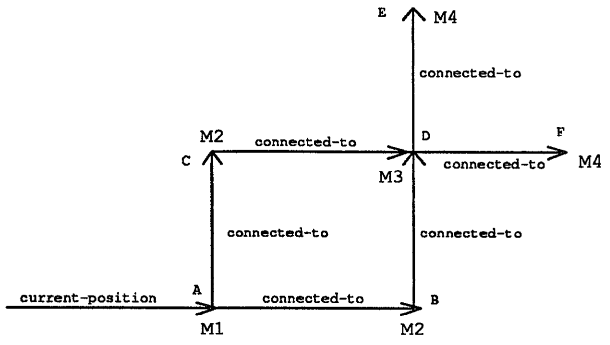


Figure 8-1: Mapping onto marker passing.

Figure 8-1 shows this mapping with the help of the simple example from Figure 2-1. The figure shows the structure described by the working memory in Figure 2-1-b, with the addition of the wme **(current-position A)**. This working memory is to be matched with the production in Figure 2-1-a. Matching the first condition results in the marker **M1** being sent to the symbol A. This binds the variable <x> to A. Matching the second condition results in marker **M2** being sent to the symbols **B** and **C**, which become the bindings for the variable <z> in the value field of the second condition. Matching the third condition results in passing marker **M3** and obtaining binding **D** for the variable <y>. Note that when two **M3** markers from **B** and **C** reach **D**, they are ORed together. Finally, passing marker **M4** obtains bindings for variable <w>. The result of the match is the same as the result of the arc consistency match:<x> = **A**; <z> = **B**, **C**; <y> = **D**; <w> = **E**, **F**.

As in the previous modification to the definition of arc consistency, markers actually have to be passed in a bidirectional manner to achieve complete consistency in tree-structured productions. That is, once the markers have reached the leaves of the production, they are transmitted back toward the root. In this example, the backward marker propagation phase would begin by sending marker M4 back and then continuing onwards with M3, M2 and M1. (In an actual implementation, correctness requires that the backward marker propagation be done first. However, for expository purposes, we have reversed the order of marker propagation.)

The bidirectional marker passing results in the following: suppose there is a binding x for a variable <x> in the identifier field of a condition with attribute A1. Now, with marker passing in the forward direction along the A1 link, bindings from the value fields of wmes with attribute A1 are obtained for the variable <y> in the value field of the condition. Symmetrically, if there is a binding y for its variable <y> in the condition's value field, then with marker passing in the reverse direction along the A1 link, bindings from the identifier fields of the wmes with attribute A1 are obtained for the variable <x> in the identifier field of the condition. This is exactly what the arc consistency match achieves.

Why are tree-structured productions (queries) needed in this mapping? In parallel marker passing systems, the particular equality test issues are well known as the *copy confusion* problems. That is, only tree-structured productions can be matched accurately.

9. Summary and Relevance to Other Work

A combinatorial production match is problematical for Soar for several reasons. This paper was focused on eliminating these combinatorics by introducing alternative production system formulations. The contributions of this paper can be summarized as follows: (1) it introduces absolute and relative requirements for evaluating alternative formulations; (2) it introduces a framework for generating alternative formulations; (3) using the framework it shows that the unique-attribute formulation is the best *within* the dimensions investigated, assuming a token match; (4) it introduces the tokenless match and maps it onto constraint satisfaction; (5) it shows that with tokenless match, other formulations may fit the absolute and relative requirements better than the unique-attributes; (6) it shows how an entirely different formulation — the marker-passing formulation — maps onto a formulation generated via the dimensions introduced in this paper, providing some evidence for the generality of the formulations considered here.

An important question is the relevance of this research for the non-Soar community. The combinatorial production match is not a Soar speciality. It is observed in various other systems — OPS5-based systems (Brownston, Farrell, Kant, and Martin, 1985) and rule-based systems like Prodigy (Minton, 1988a). In all these systems, there is continuing research on achieving real time performance (Barachini and Theuretzbacher, 1988, Parson and Blank, 1989), on eliminating expensive learned rules (Chase et al., 1989, Minton, 1988b) and load balancing schemes for attaining high parallelism (Acharya and Tambe, 1989, Miranker, 1987, Tambe and Acharya, 1989) However, in general, the area of alternative production system formulations for solving the problems facing these systems has not been investigated. The representations in these systems are based on attribute-values, very similar to Soar's representation — allowing a mapping between the results based on Soar's representation to those systems. For instance, the utility of the distinction between unique- and multi-attributes for Prodigy is shown in (Etzioni, 1990). Furthermore, representations in frame-based systems like Theo (Mitchell et. al., 1989) map quite well into attribute-values, in fact, the version used by Theo already corresponds to the unique-attributes (Tambe and Rosenbloom, 1989). Thus, the results derived in this paper would appear to be relevant to all these systems. In particular, the idea of a tokenless match, that would eliminate a significant amount of combinatorics, appears to be very relevant to these systems. We hope that our research in Soar and related research in these other systems will allow us to gain a better understanding of the tradeoffs in knowledge representation, efficiency and learning.

Acknowledgements

We thank Allen Newell, Brian Milnes, Anurag Acharya, Dirk Kalp and Gary Pelton for useful comments on earlier drafts of this paper. We also thank Kathy Swedlow for technical editing.

References

Acharya, A. and Tambe, M. (1989). Production systems on message passing computers: Simulation results and analysis. *Proceedings of the International Conference on Parallel Processing*. pp. 246-254.

Barachini, F. and Theuretzbacher N. (1988). The challenge of real-time process control for production systems. *Proceedings of the National Conference on Artificial Intelligence*. pp. 705-709.

Brownston, L., Farrell, R., Kant, E. and Martin, N. (1985). *Programming expert systems in OPS5: An introduction to rule-based programming*. Reading, Massachusetts: Addison-Wesley.

Chase, M. P., Zweben, M., Piazza, R. L., Burger, J. D., Maglio, P. P. and Hirsh, H. (1989). Approximating learned search control knowledge. *Proceedings of International Workshop on Machine Learning*. pp. 218-220.

Dechter, R., and Pearl., J. (1988). Network-Based Heuristics for Constraint-Satisfaction Problems. *Artificial Intelligence*, *34*(1), 1-38.

Etzioni, O. (1990). *A structural theory of search control*. Ph.D. diss., School of Computer Science, Carnegie Mellon University. In preparation.

Fahlman, S. E. (1979). Representing and using real-world knowledge. In Winston, P. H. and Brown, R. H. (Eds.), *Artifical Intelligence, An MIT perspective*. Cambridge, Massachusetts: MIT Press.

Forgy, C. L. (1981). *OPS5 User's Manual* (Tech. Rep. CMU-CS-81-135) Computer Science Department, Carnegie Mellon University.

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, *19*(1), 17-37.

Gupta, A., Tambe, M., Kalp, D., Forgy, C. L., and Newell, A. (1989). Parallel implementation of OPS5 on the Encore Multiprocessor: Results and analysis. *International Journal of Parallel Programming*, Vol. *17*(2).

Laird, J. E., Newell, A. and Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*(1), 1-64.

Laird, J. E., Rosenbloom, P. S. and Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, *1*(1), 11-46.

Levesque, H. J., and Brachman, R. J. (1985). A fundamental tradeoff in knowledge representation and reasoning. In Brachman, R. J., and Levesque, H. J. (Eds.), *Readings in knowledge representation and reasoning*. Morgan Kaufmann Publishers, Inc.

Mackworth, A. K. (1977). Consistency in Networks of Relations. *Artificial Intelligence*, *8*(1), 99-118.

Mackworth, A. K., and Freuder, E. C. (1985). The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems. *Artificial Intelligence*, *25*(1), 65-74.

Minton, S. (1988). *Learning Effective Search Control Knowledge: An explanation-based approach.* Ph.D. diss., Computer Science Department, Carnegie Mellon University.

Minton, S. (1988). Quantitative results concerning the utility of explanation-based learning. *Proceedings of the Seventh National Conference on Artifical Intelligence*. pp. 564-569.

Miranker, D. P. (1987). *Treat: A New and Efficient Match Algorithm for AI Production Systems.* Ph.D. diss., Computer Science Department, Columbia University.

Mitchell, T.M., Allen, J., Chalasani, P., Cheng, J., Etzioni, O., Ringuette, M., and Schlimmer, J.C. (1989). Theo: A framework for self-improving systems. In VanLehn, K. (Ed.), *Architectures for Intelligence*. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Newell, A. (1989). The Quest for Architectures for Integrated Intelligent Systems. Talk at IJCAI 89 on Recieving the Research Excellence Award.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, Massachusetts: Harvard University Press. In press.

Parson, D. E., and Blank, G. D. (1989). Constant-time pattern matching for real-time production systems. *SPIE Vol. 1095 Applications of Artificial Intelligence VII*. pp. 971-982.

Patel-Schneider, P. F. (1989). A four-valued semantics for terminological logics. *Artificial Intelligence*, *38*(3), 319-351.

Polk, T.A., Newell, A., and Lewis, R.L. (1989). Toward a unified theory of immediate reasoning in Soar. *Proceedings of the Annual Conference of the Cognitive Science Society*. pp. 506-513.

Rosenbloom, P. S. and Laird, J. E. (1986). Mapping explanation-based generalization onto Soar. *Proceedings of the Fifth National Conference on Artificial Intelligence*. pp. 561-567.

Tambe, M. and Acharya, A. (1989). Parallel implementations of production systems. *VIVEK: A quarterly in artificial intelligence*, *2*(2), 3-22.

Tambe, M. and Newell, A. (1988). Some chunks are expensive. *Proceedings of the Fifth International Conference on Machine Learning*. pp. 451-458.

Tambe, M. and Rosenbloom, P. (1989). Eliminating expensive chunks by restricting expressiveness. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*. pp. 731-737.

Tambe, M. and Rosenbloom, P. (1990). Investigating alternative production system formulations. School of Computer Science, Carnegie Mellon University, In preparation.

Tambe, M., Newell, A., and Rosenbloom, P. S. (1990). The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning*, Vol. *5*. (To appear).

Touretzky, D. S. (1986). *The mathematics of inheritence systems*. Los Altos, California: Morgan Kaufmann Publishers, Inc.