HPM:
A COMPUTATIONAL FORMALISM FOR HEURISTIC PROCEDURE MODIFICATION[1]

Robert Neches
Learning Research and Development Center
University of Pittsburgh
Pittsburgh, PA 15260
USA

Abstract

The **HPM** (Heuristic Procedure Modification) system is a model of strategy learning and optimization, implemented as a processing environment within the **PRISM** production system package [4]. This paper describes progress in getting **HPM** to emulate children's discoveries about basic addition procedures. HPM's goal-trace and production-trace formalisms allow it to maintain a history of its actions which is both goal-ordered and time-ordered. Heuristics about meaningful patterns in these traces guide the construction of new productions, which modify procedures by replacing or circumventing pre-existing productions.

1. Introduction

Young children (approx. 4 years) often add by a counting procedure called the "SUM method". This method consists of (a) counting out a set of objects to represent the first addend; (b) counting out another set of objects to represent the second addend; (c) merging the two sets, and counting the number of objects in the new set. One strategy used by older children is the "MIN method", which starts with the larger number and increments it the number of times given by the smaller addend. This development can be modelled as an additive production system, in which the MIN procedure is obtained from the SUM procedure by the insertion of a series of productions that mask or circumvent pre-existing productions.

Given an initial production system for the SUM method coded in its goal-structured formalism, the **HPM** system can discover for itself some of the productions in the additive set. Its goal formalism allows **HPM** to maintain a history of its actions which is both time-ordered and goal-ordered. Strategy transformation productions, which can fire in parallel with performance productions, respond to patterns in this history by building productions which predict future outcomes, in addition to productions which produce changed performance. The patterns which evoke initial strategy transformation productions each represent a different heuristic for suggesting when and where a certain type of transformation might be fruitful.

Like Barr [2], I am concerned with exploring how meta-knowledge can be exploited in an intelligent learning system. The key concept in **HPM** is the specification of conventions to be followed when processing goals and passing information between them. Their existence allows informal strategy change heuristics to be respecified formally as productions with their conditions expressed as propositions in the goal-description formalism. This enables the system to carry out strategy transformations by having those productions construct new productions which mask or circumvent pre-existing productions. The formalism which implements the conventions also has the benefit of imposing constraints that reduce the reasoning power required to construct appropriate changes.

HPM is part of a larger project on procedure learning and optimization [7][2]. The program is a self-modifying production system [5] implemented in MACLISP on a PDP KL-10. Its design has been influenced by analyses of human performance in procedure learning tasks. These analyses indicate that procedure modifications involve pre-compiled heuristics for applying *transformations*, simple procedures for producing particular kinds of changes.

Neches [7] describes 21 application heuristics associated with various transformation types. The heuristics subsume most of those suggested by Anzai & Simon [1]. This paper, however, will present only the heuristics actually implemented in the **HPM** system:

- **Result still available**: *IF a procedure is about to be executed with a certain input, but the result of that procedure with the same input is recorded in working memory, THEN try to borrow that result now and in the future.* (Requires procedure/goal, input, result, time, and processing information.)
- **Untouched results**: *IF a procedure produces an output, but no other procedure receives that result as input, then try deleting the procedure.* (Requires procedure/goal, input, result, and episode information.)
- **Effort difference**: *IF a difference in expended effort is observed when the same goal is operating on the same input(s) at different times, THEN set up a goal to find a difference between the methods used, and try to produce the circumstances which evoked the more efficient method.* (Requires effort, procedure/goal, input, event, and subgoal information.)

As the parenthesized notes indicate, each of these heuristics depends on the availability of certain information about a procedure's processing history. The formalism described in Section 2 is designed to capture the information required by such heuristics. Retaining that information, although necessary to enable learning, makes **HPM** vulnerable to breakdown under a glut of extraneous information. Section 3 briefly describes mechanisms for handling the information explosion problem.

[1] The research reported in this paper was supported in part by NIMH Grant # MH07722, and by ARPA Grant # F33615-78-C-1551.

[2] This section has been greatly abbreviated due to limited space. Please consult this reference for more details.

Once these issues are clarified, it is possible to examine the operationalization and application of these heuristics, the topic of the remaining sections.

2. Execution history: The goal trace and the production trace

HPM fundamentally operates by manipulating propositions stated as node-relation-object triples. An object can be either a node, a proposition, a list of nodes, or a list of propositions. Activation is associated with propositions.

The condition, or left-hand side (LHS), part of productions match against sets of propositions in active memory. When a production matches successfully and is selected for firing, its action part, or right-hand side (RHS), contains (a) propositions which are added to the semantic network and made active; and/or, (b) actions, or RHS-functions, which perform computations without necessarily adding propositions to memory.

HPM uses **PRISM**'s trace-data option to invoke a procedure after each production firing that adds a description of the firing to active memory. The description reflects the production instantiation rather than the production itself.

HPM represents processes in a hierarchical goal structure similar in organization to Sacerdoti's [8] planning nets. Goals are decomposed into partially ordered lists of subgoals until executable goals are reached. Each goal is represented in terms of relations from a common node to other objects which define it. The rules of the representation constrain the form of **HPM** productions; procedures are productions that build goal structures.

For example, one of the first rules in a production system for addition by the SUM method is, "*To add two numbers, generate sets corresponding to them and then count how many elements the two sets contain.*" Figure 1a shows the goal-trace propositions involved in this production, with those matched as conditions shown with dark lines and those added as actions shown with lighter lines. Figure 1b shows the corresponding production-trace propositions.

The production responds to a goal to *ADD* by building structures in the network representing goals to *GENERATE-SETS* and *COUNT-UP*. These are linked by a *subgoal* relation from the initial goal, and a *then* relation from the first subgoal to the second. These establish the goal-subgoal hierarchy and the ordering of subgoals. *Input* and *result* relations from the goal nodes point to nodes describing operands of the goals. These are described in terms of various relations, with the most important being *value*, which indicates the concept instantiated by an operand.

The *goal* and *value* relations are **HPM**'s means of representing the type-token distinction discussed by Woods [10]. **HPM**'s strong type-token distinction means that objects are represented in network structures of some potential complexity. To avoid comparison problems, "*formal-value*" tags, associated with nodes in the semantic network, are constructed in a canonical fashion which causes nodes representing equivalent structures to have identical tags. This enables **HPM** to immediately recognize equivalent objects by comparing their tags.

As Figure 1 illustrates, a production must only specify the propositions relevent to its own processing; it need not indicate all relations from a node Nevertheless, it is required to construct a goal-trace representation by using only a restricted set of propositions and obeying their semantics. The semantics of the goal-trace representation allow only the following types of nodes: (a) *GOAL* nodes; (b) *DATA* nodes, which instantiate concepts; (c) *SET* nodes, used for concepts which represent sets of *data nodes*; (d) *FIRING-EVENT* nodes, discussed previously; (e) *PREDICTION* nodes, discussed in section 4; and, (f) *EFFORT* nodes, which have a numeric value associated with them representing the estimated processing effort of a goal.

An **HPM** production system for solving a task primarily consists of productions which add goals (such as was just illustrated), productions which set up data structures for goals requiring iteration by establishing *has-data* relations from the goals to *DATA* nodes, and productions which terminate processing of a goal by manipulating *result*, *value*, and *status* relations.

These goal structures are augmented by system productions which collect information. For example, effort estimation is managed by two **HPM** system productions. The estimate is linked to *goal* nodes by the **effort* relation.

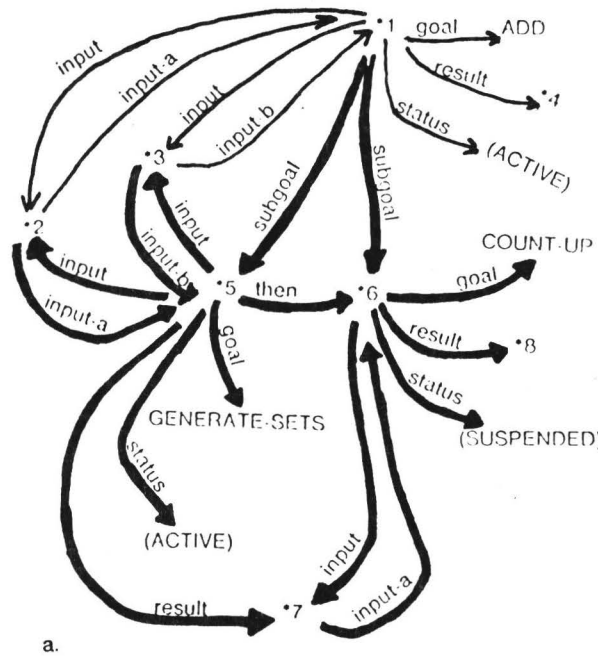

Figure 1.

In summary, productions build goal structures similar to many familiar goal representations. Thus, **HPM** can express a range of procedures and processing constructs. The goal structure remains afterwards, and forms a trace of a procedure's actions. **HPM** also retains a *production trace*, which indicates the conditions under which portions of the goal trace were created or modified. Section 4 will consider how the information contained in the goal and production traces is utilized by strategy modification heuristics. However, that discussion requires some understanding of **HPM**'s mechanisms for managing the information explosion entailed by retention of the two forms of trace data, the topic of section 3.

3. HPM processing mechanisms

By the very nature of learning, a self-modifying system cannot know in advance what it will learn in a specific case. Therefore, it also cannot know in advance what information it will require to do that learning, which in turn implies that it must retain all information which it is likely to need.

The implication of this observation is that a large information base is an unavoidable requirement of a realistic model of performance and learning. This places greater stress on the attention-focusing mechanisms of production systems. Since they must retain a potentially large body of extraneous information in order to avoid missing critical information, it becomes crucial that learning systems have effective means for concentrating attention on important data and avoiding distractions inherent in a larger information space. Two processes affect focus of attention in HPM: associative retrieval and conflict resolution.

3.1. Goal-driven associative retrieval

HPM has a mechanism called "*goal-driven spreading*" for associative retrieval by spreading activation. When any proposition is asserted, the scheme activates all other propositions about nodes contained in that proposition. When a goal becomes active, activation is spread not only from the proposition describing the goal, but also from the proposition(s) describing its inputs and goal type. Similarly, when a goal is terminated, activation is spread from the propositions describing its result, its planned successors, and the goal which invoked it.

Associative retrieval functions as an attention-focusing mechanism by allowing **HPM** to partition its memory into active and non-active sections, with the non-active section automatically eliminated from consideration. Goal-driven spreading activation helps reduce the size of active memory. It enables retrieval of indirect associates most likely to be relevent to the system's immediate performance and learning goals, while avoiding the combinatorial explosion inherent in activating all indirect associates.

3.2. By-class conflict resolution

Even under this context-sensitive associative retrieval method, active memory can still be quite large. This produces an increase in matches -- many more productions are likely to find matching data, and many of those productions are likely to have more than one way to match against the data. This introduces the familiar problem of "conflict resolution".

Parallelism simplifies these problems. First of all, it reduces the demands on conflict resolution by lowering the criterion of success. Finding a single "best" production instantiation is difficult to do satisfactorily; it is much easier to find a set of "good" productions which do not interfere with each other. Second, parallelism -- like the notion of "beam search" in the HARPY system [6] -- helps avoid errors due to premature restriction of attention. Third, it enables reducing active memory size because important data will be attended to promptly upon its assertion and can safely be eliminated from active memory much earlier than in a serial system. Reducing memory size reduces the number of potential uninteresting matches, thereby easing the conflict resolution problem.

HPM emulates a parallel production system. Productions are grouped into six different classes, with a separate conflict resolution policy for each class. The production instantiations fired on a given cycle represent the union of selections from the individual classes. Productions in different classes can be presumed not to interfere with each other, and can safely fire in parallel. The circumstances under which productions in the same class might interfere with each other depend on the task performed by that class. Therefore, conflicts are resolved by class-specific policies.

The six conflict-resolution classes currently distinguished by **HPM** are:

- *Goal-manipulation*: productions which operate upon goal-trace structures.
- *Data-bookkeeping*: productions which maintain the correctness of **HPM**'s representation of complex data objects.
- *Goal-bookkeeping*: productions which maintain correctness of goal structures.
- *Data-description*: productions which add information representing knowledge about properties of data objects.
- *Strategy-change-noticing*: productions which detect or predict situations relevent to development of a strategy change.
- *Strategy-change-maker*: productions that perform actions which effect changes to a procedure, usually by building a new production.

4. An example: discovering improved addition strategies

This section is concerned with showing how the **HPM** goal trace formalism applies to the simulation of cognitive processes.

4.1. The SUM strategy for addition

Psychological researchers have shown that very young children who solve addition problems by a SUM or "counting-all" method can discover MIN [3]. The essential properties of SUM are that external objects (e.g., fingers or blocks) are counted out to represent each addend, and that these sets of objects are then merged and the combined set counted in order to produce the sum.

The **HPM** production system for SUM requires 14 productions. Following is a summary of its operation, with the names of key productions given in parentheses. Unless otherwise stated, the productions belong to the goal-manipulation class.

The production system solves addition problems by asserting goals to GENERATE-SETS and COUNT-UP the sets (*Addition-plan*), in response to a goal to ADD (*Goal-to-add*). The first goal is decomposed into two goals to MAKE-A-SET, one for each addend (*Goal-to-generate-sets*). These goals produce sets of objects corresponding in size to the addends. Both spawn a series of CORRESPOND-ELEMENT goals, which produce pairings of

objects with numbers (*Make-a-set-of-fingers*, *Goal-to-correspond-elements*). Each of these objects is appended to a set associated with the MAKE-A-SET goal. In both cases, the series of CORRESPOND-ELEMENT goals is terminated when an object is appended to the set that has a number assignment matching the appropriate addend (*Have-a-set*). After the two sets are constructed by the MAKE-A-SET goals, a set representing their merger is constructed during termination of the GENERATE-SETS goal (*Sets-have-been-generated*).

The COUNT-UP goal then becomes active (*Continue-planned-subgoal* in the goal-bookkeeping class), and counting of that set is initiated (*Start-counting-up*). The counting involves finding objects in the set of objects-to-be-counted, assigning a new number to them, and appending them to a set of counted-objects. This process is also accomplished by a sequence of CORRESPOND-ELEMENT goals (*Count-element*). The sequence is terminated when the set of counted-objects matches the set of to-be-counted objects (*Finished-counting-up*). The answer is then given by the size of the counted-objects set (*Finished-adding*).

4.2. Strategy transformations in HPM

The goal-trace formalism is designed to facilitate domain-independent heuristics for modifying procedures, like the three mentioned in Section 1. This section discusses their application in modifying the SUM strategy; the next section illustrates how they are operationalized by discussing the implementation of the first heuristic.

HPM gets from SUM to MIN through a series of incremental refinements. The first involves eliminating redundancies in counting up the combined set of objects representing the two addends. Since counting consists of creating a set of objects in which numbers have been assigned sequentially to each object in the set, there is a point where the counting-up process creates a set of objects and number assignments equivalent to that created when building a set to represent one of the addends. After that point, the process continues adding to the set by taking objects corresponding to the other addend, which makes the set unique.

At this intermediate point, however, it is possible to recognize through the *Result-still-available* heuristic that the counting-out for one addend (i.e., the *result* of a MAKE-A-SET goal) can also serve as part of the counting-up process (i.e., as an intermediate result in processing the COUNT-UP goal). When a production is built that implements the change, the resulting procedure differs in that the COUNT-UP goal starts its iteration of CORRESPOND-ELEMENT subgoals with a set containing the elements produced by one of the MAKE-A-SET goals, rather than starting with an empty set. This causes processing to pick up with the result of the other MAKE-A-SET goal. Rather than counting all of the objects in the combined set, the new procedure therefore counts only the objects representing one of the addends, but counts them starting from the number given by the other addend instead of starting from 1.

Now, when double-counting of the one addend is eliminated in this fashion, an inefficiency is introduced which can be detected through the *Untouched-results* heuristic. When the objects produced under MAKE-A-SET goal are no longer counted under the COUNT-UP goal, they are not really used at all. That is, there is no point in creating these objects if they aren't going to be counted. The only function which they still serve is to give the starting number for counting up the objects representing the other addend. That information can be gotten elsewhere, however, since the size of the set representing an addend is (of course) given by the addend itself.

This change is realized by building a production which responds to the MAKE-A-SET goal used in the first strategy change to speed up processing of the COUNT-UP goal. The production asserts that the MAKE-A-SET goal has assigned to an object the number given by the addend which was input to the goal. This satisfies its termination conditions, causing the goal to complete with a single element set as its result. That single element has the correct number assignment for initializing counting of the objects representing the other addend. Note that this new production would give an erroneous result if it fired to all instances of MAKE-A-SET goals. However, the conditions of the new production are constructed from the production trace. Therefore, the conditions describe the context for asserting the particular goal-instance that is used in the first shortcut, and the new production will not fire in the general case. The result of this second change is a procedure which counts out objects corresponding to one of the addends, and then counts up those objects starting from the number after the other addend.

When one or both of these changes are made, the opportunity is created for HPM to discover effort differences between different trials with the same problems. This is because the effort involved depends on whether the addend treated specially is the larger or smaller of the two. Effort is minimized in the case where objects are generated to represent the smaller addend and their counting is initialized by the larger addend. The result is a procedure in which, for problems in which the other shortcuts would treat the smaller addend specially, the problem is first transformed into an equivalent form where the larger addend is taken as special. Problems in which the shortcuts would initially be applied to the larger addend are left as they are. This new procedure has the properties of the MIN procedure: effort is now proportional only to the size of the smaller addend, but there is a small effect of the order of the addends[3].

It is important to note that the independence of these last two strategy changes means that they can take place in either order, which means that HPM can follow several different paths from SUM to MIN.

5. Operationalization of heuristics

We have seen that very straightforward heuristics can be used to account for transitions between procedures. Operationalizing these heuristics in HPM requires dealing with the following set of problems: (a) detecting instances of situations described in the heuristics' conditions; (b) determining the precursors of those situations, i.e., learning to predict them; (c) constructing appropriate new actions for productions which implement strategy changes; and, (d) discovering the range of application for a new production. This paper is primarily concerned with (a) through (c), where the goal-trace conventions permit making assumptions that greatly simplify the problems.

5.1. Determining applicability of heuristics

The approach taken in HPM to operationalizing conditions for heuristics basically consists of re-expressing the conditions as patterns in sets of goal-trace propositions. It is not essential that all of these patterns co-exist in time, because of the production trace.

---

[3]The latter observation was first reported by Svenson [9], and has been confirmed in recent studies of his reported in a personal communication.

In the *Result-still-available* heuristic, the stated condition is that a previously computed item is needed again. However, this explicit condition really consists of several implicit conditions which go into deciding that an item is "needed again" In **HPM**, the first step begins when the system notices that the input for a goal is something that has been used before. This post hoc discovery comes too late for **HPM** to change its course of action on that trial; **HPM** cannot tell that it will re-use a computed value, only that it just has re-used one. Since inputs can come to a goal either as the *result* of another goal or through a *has-data* relation from a higher-level goal, two versions are needed of the production which makes this discovery. Figure 2 illustrates the production which applies when the current source of a goal's input is a *has-data* node. This is the production which initiates the first strategy transformation in the development from SUM to MIN.

Consider-keep-partials-for-data-node:

IF a goal has just been asserted with a particular data node
as input, there is a different data node with an equivalent
value, the goal was established in response to a *has-data*
relation between the input and a higher goal,
and trace-data is available for the production asserting the
goal and for the actions of the production which terminated
the goal that had the other data node as its result,

THEN build a production to predict that this relationship will recur:

IF another instance of this goal is asserted with this
input under the equivalent conditions, and another
instance of the other goal terminated in the same way as
observed this time,
THEN predict that the result of that other goal will be
equivalent to the input of the new goal-instance.

Figure 2.

The conditions for the production can be put into three groups: conditions governing when it fires (ensuring that it will fire as early as possible), main conditions checking that criteria for firing the production are met (in this case, that the goal's input has a potential alternative source), and conditions which pick up data needed on the action side of the production (by inspecting the production trace to find the context in which the earlier source completed and the current source was asserted).

5.2. Finding conditions for the new production

The action of the first production builds a new production which essentially predicts that the same situation, if observed again, would produce the same relationships between goal-trace propositions. That is, the prediction is that any goal terminating with equivalent assertions to the observed earlier source will have its result equivalent to the input of any goal asserted under conditions equivalent to those for the observed current goal.

Since a new prediction production is only built when the prediction is known to be true for the current case, **HPM** behaves as if the prediction had been made and tested for that case. This causes a strategy transformation production to fire and build a second production which will change the strategy. The strategy transformation production is shown in Figure 3.

Keep-partial-set-for-data:

IF a goal has just been asserted with a particular data node
as input, there is a different data node with an equivalent
value, the goal was established in response to a *has-data*
relation between the input and a higher goal,
and trace-data is available for the production asserting the
goal and for the actions of the production which terminated
the goal that had the other data node as its result,
AND a prediction has been asserted and confirmed that the
goal's input and the other data node would match,
AND the input is a set with an element as current member,
AND the other data node has some relation associating it
with an element equivalent to that current member,

THEN construct a production to avoid recomputation of the set
by copying the prior data node over:

IF the evoking conditions are present for the production
which asserted the *has-data* relationship between
the higher goal and the current input of the current goal,
andan instantiation of the goal which produced the
alternative source has both been asserted and completed
under the same circumstances as this occasion,
and that goal's result has relations that correspond
to those needed to predict what the current member
should be,
THEN assert that the higher goal *has-data* to a node which
has the set represented by the alternative source as
its subset; assert a current-member for this new set,
along with any other relations used in the conditions
of the goal of interest.

Figure 3.

As before, there are several versions of the production in order to allow for each of the alternative information-passing methods which can be used in the goal-trace formalism. The one shown is the version which applies in making the first strategy change. Its first conditions require that a successful prediction has been made. Like the noticing production, it also contains conditions which identify the context in which the new source goal was terminated. In addition, it contains conditions which identify the circumstances under which the to-be-replaced goal was asserted. Finally, it has conditions specialized for *has-data* data nodes which represent sets; these conditions test whether the current goal makes use of any propositions allowed in describing sets (e.g., the *current-member* relation) or other data objects. Related to those conditions are others which seek to find or construct the analogous propositions for the result of the potential new source goal.

When the production actually builds the production for the strategy change, the conditions of that production are derived from the conditions just described. First, they borrow the conditions which led to assertion of the data's current source. This ensures that the new production can fire at the time the source would have been asserted, thus enabling it to override the production which would do so. Second, they borrow the terminating conditions of the goal which first produced the data. This ensures that the value is available to be copied, and that the new production has the ability to do so. Third, they borrow the initiating conditions of the first goal, expressed in terms of the production trace since they may no longer be true at the critical point in time. These conditions increase the likelihood that the new production will fire only in siuations closely equivalent to the

current situation. Finally, the new production is given the conditions needed to specify auxilary propositions such as *current-member* relations. Figure 4 provides an example of a new production resulting from this process. This new production, which implements the first shortcut for the problem 2 + 3, will compete with the *Start-counting-up* production mentioned in section 4.1.

IF a set containing a left-thumb, a left-index-finger, a right-thumb, a right-index-finger, and a right-middle-finger is input to an active COUNT-UP goal,
AND that goal has no *has-data* node,
AND a goal to MAKE-A-SET that produced a left-thumb and a left-index-finger is done, and that goal was asserted as a subgoal of a GENERATE-SETS goal, and the set produced by the MAKE-A-SET goal has a left-thumb as its first and a left-index-finger as its last element, and the set is of size 2,

THEN assert that the COUNT-UP goal *has-data* to a node which has the MAKE-A-SET goal's result as a subset, the left-thumb as its first member, the left-index-finger as its current and last member, and the number two as its size.

Figure 4.

5.3. Determining appropriate actions for new productions

The actions of the new production are almost entirely determined by the constraints of the goal trace formalism. This is exactly the point of having such a formalism; the amount of reasoning which the system must do in order to construct a change is minimized. In this case, the intent to borrow a previously generated set in order to avoid reconstructing it completely determines the actions. Having found that the set was constructed as a *has-data* node attached to a higher-level goal, the first thing to do is to replace that node with one representing the previously computed set. However, it is only necessary to set up the new linkages, since the production's conditions ensure that it fires in place of the productions which would have added the old linkages. Since the constraints of the formalism say that elements may be added to a set while an active goal has a *has-data* link to it, the production asserts propositions creating a new set which has the previously computed set as a subset. This maintains the correctness of the goal trace if additional members are added to the set, as is the case here, because the additional members are represented as belonging to the "copy" of the set rather than the original set. The goal-trace formalisms for sets, and the **HPM** system productions which maintain them, guarantee that the original and the copy are treated identically when their values are accessed but are distinguished when querying the source of those values.

The other actions of the new production are also determined by the constraints of the formalism. When a data node representing a set is being made available to subgoals by a *has-data* relation, there is a restricted set of possible propositions which might be accessed by those subgoals. The remaining actions of the productions assert propositions which duplicate those which appeared in the set being copied.

This new production will fire just after assertion of a COUNT-UP goal, skipping to the point just after counting of the set representing the first addend. Its specialized conditions make it applicable only for the particular problem, 2 + 3. Methods for generalizing such productions are discussed in [5].

This change opens the way for the *Effort-difference* and *Untouched-results* heuristics to apply in later strategy changes.

6. Closing notes

Three features of **HPM** are especially important. Formal-value tags let **HPM** represent subtle type/token distinctions without losing the ability to detect higher-level identities between the objects represented. Goal-driven associative retrieval helps **HPM** ameliorate problems of large working memory size while still ensuring that potentially relevent data will be in active memory. By-class conflict resolution lets **HPM** fire productions with fewer concerns about introducing unintended interferences or database inconsistencies.

The most important aspect of **HPM** is the notion of a goal-trace formalism. These conventions for specifying procedures and actions in the system cause those procedures to leave a history of their execution behind them. This history is a hierarchical semantic network representation of the process that preserves information about relations between goals and data. The parallel notion of a production trace preserves information about both the time-ordering of events and the context in which goals were initiated and terminated.

The existence of the formalism greatly simplifies the implementation of heuristics for improving procedures, by providing a formalism for specifying them and by imposing constraints which reduce the reasoning power required to construct appropriate modifications. Furthermore, the domain-independent conventions move the system closer to the goal of being able to demonstrate domain-independent learning capabilities.

References

[1] Anzai, Y., & Simon, H.A. The theory of learning by doing. *Psychological Review*, 1979, *86*(2), 124-140.

[2] Barr, A. Meta-knowledge and cognition. *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, 1979, 31-33.

[3] Groen, G.J., & Resnick, L.B. Can preschool children invent addition strategies? *Journal of Educational Psychology*, 1977, *69*, 645-652.

[4] Langley, P. and Neches, R. PRISM User's Manual. Technical Report, Department of Psychology, Carnegie-Mellon University, 1981.

[5] Langley, P.W., Neches, R., Neves, D.M., & Anzai, Y. A domain-independent framework for procedure learning. *Policy Analysis and Information Systems*, 1981, *4*(2), 163-197.

[6] Lowerre, B.T. *The HARPY speech understanding system*. Pittsburgh: unpublished PH.D. thesis, Computer Science Department, Carnegie-Mellon University, 1976.

[7] Neches, R. *Models of heuristic procedure modification*. Pittsburgh, PA: Psychology Department, Carnegie-Mellon University, Ph.D. thesis, 1981.

[8] Sacerdoti, E.D. *A structure for plans and behavior*. New York: American Elsevier Press, 1977.

[9] Svenson, O. Analysis of time required by children for simple additions. *Acta Psychologica*, 1975, *39*, 289-302.

[10] Woods, W.A. What's in a link: foundations for semantic networks. In D.G. Bobrow & A. Collins (Eds.), *Representation and understanding*. San Francisco: Academic Press, 1975, 35-82.