TOWARD THE DEVELOPMENT OF A REAL-TIME EXPERT SYSTEM

Jack Park
Box 326
Brownsville, CA 95919

Paper to be presented to the 1986 Rochester Forth Conference

Abstract

Application of the Forth environment to applied artificial intelligence projects has led to the development of a concurrent processor, real-time expert system. Steps toward the development of the EXPERT-5 concurrent system have included EXPERT-2, EXPERT-4, and early versions of the EXPERT-5 technology running on a personal computer. The concurrent EXPERT-5 consists of two EXPERT-5 shells running on separate M68000-based co-processors mapped into the memory structure of a third EXPERT-5 based blackboard controller running under MS-DOS on a pc. A "parser-thinker" paradigm is being explored with this architecture.

Introduction

It is reasonable to consider the universe of artificial intelligence (AI) activities as consisting of two disjoint efforts: those fundamentally research oriented, and those fundamentally applications oriented. This classification lets us distinguish certain tool-design approaches discussed in the following paragraphs.

Fundamentally, we take the view that at the implementation level research efforts will require powerful dynamic memory management schemes (including heap managers, list processors, garbage collection, etc) while applications may be ported to (or implemented in) a variety of highly simplified memory structures (such as arrays). Akin to the implementation issue, the notion of a closed universe is appropriate to the applied AI project, while the research project may entail an open universe. That is, the applied project may start with a pre-enumerated universe (knowledge base) and not be expected to extend that universe, while the research effort may similarly start with a pre-enumerated universe, but may extend or otherwise structurally alter that universe.

The pre-enumerated universe notion is the substrate on which we have built and tested a variety of experimental inference engines. These include EXPERT-2 [PARK84a], EXPERT-4 [PARK85a], and EXPERT-5 [PARK85b] (1). In the discussion following, we trace some of the development effort underlying these programs, we discuss some observations made along the way, and we conclude with a look at the future of this effort.

In the present context, an expert system is defined as a program which behaves as a domain (human) expert would when given a set of input states and a knowledge of the desired output response. Our view necessitates a further classification of systems: those oriented toward consultation, and those oriented toward process control. Both consultation and process control entail heuristic classification [CLANCEY85], the underlying task of the system.

EXPERT-2

Given the background motivation to develop a real-time expert system, EXPERT-2 was written (originally) in FIG-Forth on an Apple II computer. That program used as a design model the backward-chaining inference engine described in [WINSTON81]. The objective of the project was to explore the issues of applying Forth to an AI project.

The primary algorithm implemented in the EXPERT-2 inference engine is:

1 select a goal to prove from a pre-enumerated list of goals

2 try to verify the goal

- o see if the goal is already known
  - o if known, exit with truth
- o collect some rules from a pre-enumerated list of rules - rules which contain the goal as a consequent
- o taking the rules - one at a time - find a proof
  - o exit on first rule proven
  - o take each antecedent field of the rule and treat it as a new (sub) goal - try to verify
  - o if an antecedent field (subgoal) does not have any rules to support it, ask the user if the antecedent is true.

From this outline, we see that the EXPERT-2 system simply takes the next goal in line and tries to verify its truth by verifying the truth of rules which support it. Ultimately, verifying truth entails asking the user the truth of underlying antecedent attributes.

As an example, the familiar penguin rule (shortened here) is:

```
IF animal is a bird
AND animal swims
THEN animal is a penguin
```

If we consider this the only rule in the entire knowledge base, then, from the expert procedure outlined above, a goal to prove an animal is a penguin results in the program asking the user if the animal is a bird and if the animal swims. A "true" answer to both of these questions yields the final pronouncement that the animal is a penguin. All, of course, based on the closed-world assumption that this lone rule defines a penguin, and only a penguin.

We can generalize the expert algorithm by substitution of quering a database of world states for merely asking the user. The query

procedure might include asking the user if a given attribute is known "askable." In a real-time process controller, one expects the largest share of attributes that must be checked will be sensed data acquired by a data collection (hardware) system and made available to the inference engine as an array of numerical information.

By generalizing the query portion of the inference routine EXPERT-2 is a candidate real-time shell. Exploration of its use as a consultant in weather prediction is discussed in [PARK84b], and as a consultant in neurological research in [TRELEASE86]. Exploration of its use as a "smart" node in an array processor is discussed in [PARK86a]. That project applied EXPERT-2 to the game of Life as an exercise in designing a systolic array, one potentially capable of dealing with real-time weather prediction. In the Life project, the entire query procedure is redirected to use the array data supplied by the game, just as would a real-time process monitor with predictive capabilities.

A fundamental issue in any real-time process is that the monitor or controller must behave in real-time. This notion leads one to consider the benchmark performance of the expert system. We return to benchmark performance later, but for now, it suffices to note that EXPERT-2 is a particularly slow inference engine.

An analysis of inference engine speed (as measured in logical inferences per second - lips) leads one to look at two issues:

- o underlying hardware speed
- o inference algorithm speed

Underlying hardware speed is an issue of two parts:

- o cpu and memory speed
- o language mapping onto the cpu

Given any cpu and memory such that hardware speed is fixed (often by fiat), mapping of the language on which the domain effort is based is a primary issue. In all cases, the highest run-time speeds will be achieved with careful use of the language that uses the least number of clock cycles of the cpu. The "assembler" language of the cpu is the typical choice in this respect.

A secondary issue is one of project development cost. Here, a real-world focus brings about a fundamental comprimise between run-time speed, and overall project cost. Until the recent introduction of language-specific cpus (e.g. a circuit which executes a high-level language as its native "assembler" language), important run-time speed advantages were nearly always traded for the convenience of project development in a high-level language. For the real-time controller case, until now, the secondary issue of project cost has dominated decisions which affect the primary issue - getting the inferences done in time.

Given an underlying hardware system, and a high-level language, inference speed is now an issue of two parts:

- the basic inference algorithm
- the search algorithm

The basic inference algorithm for the backward-chaining (goal driven) EXPERT-2 was described above as: select a goal to prove, then try to prove it. The search algorithm of EXPERT-2, source code for which is in [PARK84a], details a simplistic (e.g. slow) pattern-matcher oriented to an array of ordered rules.

In the EXPERT-2 search process, the objective is to collect all rules which have as a consequent field the same symbolic reference (e.g. absolute memory address) as the current goal. For example, if the current goal is the address of the concept "penguin" then the search routine will find all rules which have that same address in one of their consequent (e.g. THEN penguin) fields. A reference to each rule found will be pushed onto the Forth parameter stack. For a knowledge base of a hundred rules, search is reasonably quick, and the program is able to operate at a few hundred logical inferences per second. As the knowledge base expands, search time will increase. A candidate for inference engine design change is rule search.

EXPERT-4

For historical purposes, EXPERT-4 grew out of EXPERT-3, which was simply EXPERT-2 with an associative memory structure - a lattice structure of rules and pointers.

The EXPERT-4 inference engine further implemented some fundamental changes to the way a user creates a knowledge base. Whereas EXPERT-2 adhered strictly to the notion of easily-readable rules (see for example the penguin rule above) where entire query strings were imbedded directly in each rule clause, EXPERT-4 introduced the predefinition of symbols and their associated query string. Thus, the rule for a penguin would be declared as follows:

```
(declarations)
     S: penguin " animal is a penguin"
     S: bird    " animal is a bird"
     S: swims   " animal swims"
(rule)
     IF bird
     AND swims
     THEN penguin
```

A slight amount of readability has been traded for an enormous improvement in effort required to create a knowledge base.

An associative memory structure is one which offers a response to a request based on the data contained, rather than the address. In EXPERT-2, search took the "next" address in the rule space and asked "what's in here?" In EXPERT-4's rule space, search takes the symbolic pointer (e.g. an absolute memory address of the goal concept) as something quite different from just another address in rule space. The symbolic pointer is the address of the beginning of a list of all occurances of the symbol. Search no longer needs to take an address

and look there for a match; each address is a match known a priori.

An interesting observation here is that EXPERT-4 does its searching at compile time rather than run time. In fact, this is an attractive notion that should be applied wherever possible. Applied to the EXPERT-4 shell, it results in a doubling of inference speed for systems with knowledge bases under a hundred rules. EXPERT-4's performance with large knowledge bases has not been characterized, but the program is the substrate for an acute pediatrics program of more than 1000 rules, and its longest search time appears less than two seconds.

As a consultant, a 2-second search time for a thousand rules is quite reasonable. As a real-time process manager, however, one looks, again, for areas of improvement. This time, we turn from search to the underlying inference algorithm.

EXPERT-5

EXPERT-5 is best described as a production system [DAVIS77] which is message-passing, object-centered and uses a blackboard [HAYES-ROTH85] as its working memory.

EXPERT-5 is a different inference engine altogether from EXPERTs -2 and -4. Here, the fundamental change is to the inference structure. Instead of an interpreter (the inference engine) looking for something to prove and trying to prove it, each object in the universe described by the knowledge base is its own inference engine.

The change in fundamental thinking came from our own reasoning behind using Forth (say, instead of Lisp) as a substrate environment. The premise is that the closer one is to the native cpu language, the faster will be the system. Forth offers an environment which does its searching at compile time so that its run-time interpreter is small and typically quite efficient. In retrospect, Forth offers a model for some aspects of inference engine design. But, we failed in our original designs to notice another regularity in Forth that would greatly improve the performance of our inference engines. This regularity was pointed out by Dana Redington in [REDINGTON84].

Forth's own "next" is quite capable of functioning as the host inference engine. Thus, two properties of the Forth environment make it suitable as an inference engine. Forth's use of compile-time searching to build data structures needs only to be exploited such that the data structures built with "find" and "create" are useful for inference processing. Further, those structures need only capture the behaviors necessary to perform inferences such that each structure, a first-class entity (an object capable of behaving according to a message it receives) in the expert system's pre-enumerated universe, is capable of behaving as its own inference engine using "next." Thus was created EXPERT-5.

The EXPERT-5 "inference engine" no longer is a body of Forth code built to find a goal and prove it. Instead, the program is just an extension of the Forth compiler for creating memory structures which are capable of behaving as individual inference engines. For example, in the EXPERT-4 rule for a penguin, a body of Forth code reads the

symbol for penguin as a goal and searches for the truth for the penguin (that truth is either already known or needs to be found by use of supporting rules). Thus, penguin is not a self-executing entity in EXPERT-4. In EXPERT-5, however, penguin would be an object with behaviors defined by a body of Forth code known as "methods." When passed a message such as "truth", the object penguin will use its truth-finding methods to determine it's own truth. Fundamentally, the response will be the same (e.g. some powerful regularities exist across all reasoning methods), but the object's approach will not rely on an external interpreter; any rules which support penguin inferences are, ultimately, self-executing - they are small bodies of Forth code written to execute the necessary reasoning chains. A fundamental difference between EXPERT-4 and EXPERT-5 is that the programmer in EXPERT-5 is responsible for the entire reasoning process (e.g. building the proper knowledge structure, invoking the proper messages, using the blackboard correctly, etc.) The programmer in EXPERT-4 need only concentrate on the semantic correctness of the rules, and their ordering; the external inference engine applies a predefined procedure to the knowledge base.

Using the game of Life as a benchmark, whereas EXPERT-2 times at approximately 200 lips, EXPERT-5 (written on 16-bit F83 on a stock PC class personal computer) runs nearly 2,700 lips, an important improvement. Timing the same EXPERT-5 on a 16-bit MVP microcoded coprocessor [KOOPMAN86] running a 280 nanosecond clock returned 41,600 lips. Thus, there are important improvements to be found in looking closely at the tools one can exploit from the host environment.

These improvements come at some cost. We have found large knowledge bases built on the present level of EXPERT-5 to be difficult to build and to maintain. This is a the same tradeoff one makes when one leaves a "readable" environment (say, English sentences as computer instructions) and employs a constrained language (like Forth). The situation is made even more difficult by a more subtle change in program design from EXPERT-4 to EXPERT-5 (at its present level). Whereas EXPERT-4 includes a rule compiler that builds the lattice memory structure necessary for search, no such rule compiler has yet been developed for EXPERT-5; the task of visualizing the structure of any tree or graph in an EXPERT-5 knowledge base falls solely on the knowledge engineer. Brighter prospects for EXPERT-5's future will include the development of a more-powerful user interface to the system.

Interesting arguments run both sides of the symbolic vs. procedural approaches to software specification [DOYLE85], and it may be that an optimum lies somewhere within specific, and carefully crafted combinations of both [SUTHERLAND86]. We have assumed a design approach that enables both symbolic inferences and numeric (or procedural) computation to coexist without interference; in EXPERT-5, all knowledge base entries are ultimately executable Forth code. A program may reason about that code, or it may execute it. Thus, as with Lisp, EXPERT-5 permits programs to reason about themselves. Programs as data may not be an important paradigm in the realm of real-time process controllers, but the capability to go beyond will be important in the discovery (scientific theory formation [LENAT83], [LANGLEY83]) arena.

The penguin example is inadequate to expose the design of an EXPERT-5 knowledge base, so we include in the appendix a small program that performs a simple robot control exercise (a monkey game). The test program is a functional copy of the program included in [BROWNSTON85], an OPS5 programming manual. The monkey's goal is to hold the banannas. It's behaviors include moving from where it is initially at to where the banannas are. If the location of the banannas is initially some position in the room and suspended from the ceiling, then the monkey "knows" to move the ladder to that location and climb it to get the banannas. The monkey is only allowed to hold one physical object at a time; if in its initial posession is a blanket, it must drop the blanket before holding, say, the ladder. An initial "bug" in the implementation included here arose when the monkey, standing at the top of the ladder, discovered it must drop the ladder before holding the banannas. It took considerable effort to "teach" the monkey a more appropriate behavior. Mention of the OPS5 environment queues further observations and these will be briefly mentioned below.

The example program is a rule-based knowledge base used to control the behavior of a robot - a monkey in this case - the underlying goal of which is to cause the robot to acquire a physical object - a bananna. Rules in the EXPERT-5 implementation have the same names as those in the reference. The reference offers english tranlations and OPS5 versions of each rule - the example program here is nearly a one-to-one match with its OPS5 counterpart. The exceptions lie in two areas: EXPERT-5 permits collecting concepts (e.g. rules) into lists which may be structurally organized as trees, tangled hierarchies, or other forms of graphs, OPS5 does not (on the surface) support this. Further, and more subtle, OPS5 relies on a process called conflict resolution [FORGY79] (3) to select the rule to fire; EXPERT-5 relies on partial and total ordering of rules - the first rule to fire in a given line of reasoning wins. A further difference, not illustrated in the example program, is the ability, in EXPERT-5 to apply both the data-driven and goal-driven approaches to the inference process. OPS5 is fundamentally a data-driven inference engine; the primary algorithm of the data-driven (forward chaining) system is: take some data item and see where it leads. Recall that the goal-driven algorithm is: take some goal and find the data to prove it.

The example program illustrates many of the EXPERT-5 features (2):

- o allocation of a working memory to serve as a Blackboard
- o rules
- o primitive behaviors
- o messages
- o lists

Not illustrated in this program is the use of frames [MINSKY75] as a paradigm for knowledge representation. Recent versions of EXPERT-5 have made frames first-class objects; they are easily implemented as

lists and behaviors.

Given that all entities in an EXPERT-5 program are capable of behaving as their own inference engine, they are all generally executable Forth code. On inspection, the test program reveals several self-executing entities:

o GOAL-SATISFIED - a common Forth colon-defined procedure which supports the user-defined blackboard. The procedure grabs a variety of entities from the blackboard and saves them on a goal-stack for later recall. This is a self-garbage-collection mechanism; popping a stack performs an analagous function to releasing list cells to a heap.

o ON-FLOOR - a rule which conditionally fires the primitive side-effect: POST something to the blackboard. Note that rules read like Forth colon definitions (which they ultimately are), and '-->' stands for "implies" and reads like Forth's IF. Entities are either POSTed to the blackboard or RECALLed from it. The blackboard, in this program, is a three-dimensional array: an entity is accessed by the level (multilevels support e.g. past, current, and future events), the name, and a value. This example rule reads:

```
IF the current goal is to be on the floor
AND the monkey is not on the floor
THEN put the monkey on the floor and tell somebody about it.
```

Thus, this rule captures a conditional behavior of the monkey.

o PHYSOBJ - a disjunctive list. EXPERT-5 permits two types of lists (type refers to their inference behavior) - disjunctive and conjunctive. If not used as inference engines, a list can behave either as a database or as an agenda. This example list has arity=1 and a lone parameter placeholder (unused in this implementation). A disjunctive list, when passed the message "TRUTH" will behave just like the rule:

```
IF a OR b OR c THEN true ELSE false.
```

For the conjunctive list, substitute AND for each instance of OR. Lists are collected in a heap [DRESS86]. The example PHYSOBJ list is used as a database.

o STARTUP - a Forth definition the purpose of which is to initialize the blackboard with startup data and present an initial (perhaps the only) goal.

o RUN - the classic Forth word designed to set the program off on its mission (as defined by STARTUP). The inference process starts with the message TRUTH passed to the object RULES. RULES is a disjunctive list behaving as a conditional agenda (exit on the first agenda item found true). This agenda captures all the knowledge contained in the knowledge base; the ultimate organization is a tree, the major nodes of which are the rules: ?ON, ?HOLDS, and ?AT, and a couple of minor nodes: CONGRATS, and SORRY - the meanings of which are intuitively obvious. This partial ordering of rules was experimentally found to raise the rule speed of this knowledge base from firing 85

rules per second without parital ordering to firing 103 rules per second with partial ordering (note that rules per second is a different type of performance figure from lips).

The example program illustrates both the implementation aspects of EXPERT-5 and its user program, and a useful, if simplistic, application of the environment. Interesting prospects for this environment would include the implementation of an experimental environment like KL-ONE [BRACHMAN84].

Current Status

EXPERT-5 has been ported to a concurrent processor system comprised of two M68000 coprocessors [HALLOCK85] each with one megabyte of memory (benchmarking EXPERT-5 at about 9000 lips in a 32-bit dtc Forth environment [BRADLEY86]). Each of the two coprocessors runs a full implementation of EXPERT-5 along with a Qualitative Process Theory shell [FORBUS85]. One processor will run a "parser" and the other will run a "thinker." In a temporal reasoning environment [ALLEN85], these two processors will be capable of monitoring and controlling real-time processes. A third EXPERT-5 serves as a "smart" blackboard monitor, running under 16-bit F83 in the host PC environment. The two M68000-based EXPERT-5 systems pass messages through the blackboard system, and communicate with the outside world (acquiring data, controlling objects, communicating with files or operators) through the same blackboard system. For the process environment, this division of tasks (reading the environment, responding to the environment) represents a logical task split, potentially doubling the throughput of a real-time process controller.

Looking Ahead

We already know what EXPERT-6 looks like. This variant of EXPERT-5 uses a list handler similar to [TRACY85] for building and maintaining its memory structure and is oriented strictly toward research in discovery systems.

Notes

(1) We claim revolutionary breakthroughs in naming conventions for computer programs.

(2) This recalls some useful observations on where "features" really come from.

(3) An interesting insight into conflict resolution in Forth, as well as further discussion on Forth in AI along with a Forth-based inference engine implementation, is found in [MATHEUS86].

References

[ALLEN85] Allen, James F. and Henry A. Kautz; A Model of Naive Temporal Reasoning; in Hobbs, Jerry R. and Robert C. Moore, eds; FORMAL THEORIES OF THE COMMONSENSE WORLD; 1985, Ablex Publishing Corp.

[BRACHMAN84] Brachman, R.J. and J.G. Schmolze; An Overview of the

KL-ONE Knowledge Representation System; Cognitive Science 9 no. 2 April-June, 1984

[BRADLEY86] Bradley, Mitch; Forth system user's manual

[BROWNSTONE85] Brownston, Lee, Robert Farrell, Elaine Kant, and Nancy Martin; Programming Expert Systems in OPS5; 1985, Addison-Wesley

[CLANCEY85] Clancey, William J.; Heuristic Classification; in Artificial Intelligence 27 (1985), pp 289-350

[DAVIS77] Davis, Randall, Bruce Buchanan, and Edward Shortliffe; Production Rules as a Representation for a Knowledge-based Consultation Program; Artificial Intelligence 8 (1977) pp 15-45

[DOYLE85] Doyle, Jon; Expert Systems and the "Myth" of Symbolic Reasoning; IEEE Transactions on Software Engineering, vol SE-11, no 11, November 1985 pp 1386-1390

[DRESS86] Dress, W.B.; A Forth Implementation of the Heap Data Structure for Memory Management; 1986 JFAR, in press

[FORBUS85] Forbus, Kenneth D.; Qualitative Process Theory; in Bobrow, Daniel G., ed;QUALITATIVE REASONING ABOUT PHYSICAL SYSTEMS; 1985, MIT Press

[FORGY79] Forgy, C.L.; On the Efficient Implementation of Production Systems; PhD thesis, Department of Computer Science, Carnegie-Mellon University, February 1979

[HALLOCK85] Hallock Systems Company; PRO-68 owners manual, 1985

[HAYES-ROTH85] Hayes-Roth, B.; A Blackboard Architecture for Control; Artificial Intelligence 26 (1985) pp 251-321

[KOOPMAN86] Koopman, Phil, Jr., and Glen Haydon; User's Manual - MVP-FORTH CPU BOARD; 1986, Mountain View Press.

[LANGLEY83] Langley, Pat, Gary L. Bradshaw, and Herbert A. Simon; Rediscovering Chemistry with the BACON System; in Michalski, R.S., J.G. Carbonell, and T.M. Mitchel eds; Machine Learning - An Artificial Intelligence Approach; 1983, Tioga Publishing Company

[LENAT83] Lenat, Douglas B.; The Role of Heuristics in Learning by Discovery: Three Case Studies; in Machine Learning - An Artificial Intelligence Approach

[MATHEUS86] Matheus, Christopher J.; The Internals of FORPS - A Forth-based Production System; 1986 JFAR, in press

[MINSKY75] Minsky, Marvin; A Framework for Representing Knowledge; in Winston, P. ed; The Psychology of Computer Vision; 1975, McGraw-Hill

[PARK84a] Park, Jack; Forth Expert System; 1984, Mountain View Press

[PARK84b] Park, Jack; Expert Systems and the Weather; Dr. Dobb's Journal, April 1984, pp 24-31

[PARK85a] Park, Jack; EXPERT-4 User's Manual; unpublished manuscript

[PARK85b] Park, Jack; EXPERT-5 User's Manual; unpublished manuscript

[PARK86a] Park, Jack; A Cellular Automaton in EXPERT-2; Dr. Dobb's Journal, April 1986, pp 42-44

[REDINGTON84] Redington, D.; Outline of a Forth Oriented Real-Time Expert System for Sleep Staging: a Fortes Polysomnographer.; 1984 FORML Proceedings, Forth Interest Group.

[SUTHERLAND86] Sutherland, John W.; Assessing the Artificial Intelligence Contribution to Decision Technology; IEEE Transactions on Systems, Man, and Cybernetics, vol SMC-16, no 1, January/February 1986 pp 3-20

[TRACY85] Tracy, Marten; Forth List Handler; disk available from Forth Interest Group.

[TRELEASE86] Trelease, Robert; JFAR in press

[WINSTON81] WINSTON, P.H., and B.K.P. HORN; LISP; 1981 Addison-Wesley.

Appendix

```
Scr # 0          B:MONKEY.BLK
  0 MONKEY GAME WRITTEN FOR EXPERT-5  (Ref: OPS-5 book)
  1 VERSION MONKEY1.BLK
  2   USES FILTER RULES
  3 ON EXPERT-5.2 (16-bit F83 on a stock PC)
  4 RUNS ~135 RULES IN ~1.8 SEC = 75 RULES/SEC
  5 without diagnostic depth .  ~ 85 rules/sec
  6 ON EXPERT-5.2 WITH IMPROVED MESSAGE MEMORY
  7 RUNS 1.3 SEC = 103 RULES/SEC - times include screen printing
  8 MONKEY4.BLK runs on EXPERT-5.6 with a heap manager and
  9   default (single) blackboard level.
 10
 11 Presently cannot handle the case of blanket on banannas on floor
 12
 13
 14
 15

Scr # 1          B:MONKEY.BLK
  0 \ MONKEY4 ( latest version )  LOAD SCREEN
  1
  2 : WALL ;       \ something to "forget"
  3
  4 E5INIT         \ initialize EXPERT-5
  5 RELEASE-HEAP   \ clear the heap
  6
  7 2 32 THRU      \ load monkey program
  8
  9 CR .( Monkey ready - type RUN)
 10
 11
 12
 13
 14
 15

Scr # 2          B:MONKEY.BLK
  0 EXIT
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Scr # 3          B:MONKEY.BLK
  0 \ MONKEY  working memory ( blackboard ) design
  1
  2 1 ( level ) 8 ( cells ) 4 ( entries ) WM  ( allot space )
  3 NEWCELL
  4 WM-CELL GOAL    NEWENTRY
  5   WM-ENTRY TYPE
  6   WM-ENTRY STATUS
  7   WM-ENTRY NAME
  8   WM-ENTRY TO
  9 WM-CELL MONKEY  NEWENTRY
 10   WM-ENTRY AT
 11   WM-ENTRY ON
 12   WM-ENTRY HOLDS
 13 WM-CELL COUCH
 14   WM-ENTRY WEIGHT
 15 WM-CELL LADDER

Scr # 4          B:MONKEY.BLK
  0 \ MONKEY  wm ( blackboard ) design
  1
  2 WM-CELL BLANKET
  3 WM-CELL BANANNAS
  4 WM-CELL FLOOR
  5 WM-CELL CEILING
  6
  7 1 1 OR: PHYSOBJ
  8   COUCH LADDER BLANKET BANANNAS L;
  9
 10
 11 : GOAL-SATISFIED GS> GOAL TO POST GS> GOAL NAME POST
 12   GS> GOAL TYPE POST ;
 13
 14 : NEWGOAL GOAL TYPE RECALL >GS GOAL NAME RECALL >GS
 15   GOAL TO RECALL >GS ;

Scr # 5          B:MONKEY.BLK
  0 \ MONKEYS  rules    ON cluster
  1
  2 R: ON-FLOOR
  3   GOAL NAME RECALL ['] FLOOR = ( goal )
  4   MONKEY ON RECALL ['] FLOOR <>  AND  ( state )
  5     --> CR ." 1 monkey jumps on floor "
  6       ['] FLOOR MONKEY ON POST
  7       GOAL-SATISFIED TRUE R;
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Scr # 5          B:MONKEY.BLK
  0 \ MONKEY  rules ON
  1
  2 R: ON-FLOOR-SATISFIED
  3   GOAL NAME RECALL ['] FLOOR = ( goal )
  4   MONKEY ON RECALL ['] FLOOR =    AND ( state )
  5     --> CR ." 2 monkey already on floor "
  6       GOAL-SATISFIED TRUE R;
  7
  8
  9
 10
 11
 12
 13
 14
 15

Scr # 7          B:MONKEY.BLK
  0 \ MONKEY rules ON
  1
  2 R: ON-PHYSOBJ
  3   GOAL NAME RECALL ?NIL NOT
  4   GOAL NAME RECALL ['] PHYSOBJ MEMBER ( goal ) AND
  5   GOAL NAME RECALL >OBJECT
  6     AT RECALL ( xy )
  7     MONKEY AT RECALL ( xy ) =  ( same location ? ) AND
  8   MONKEY ON RECALL GOAL NAME RECALL <> AND
  9   MONKEY HOLDS RECALL GOAL NAME RECALL
 10     >OBJECT ON RECALL <> AND
 11     --> CR ." 3 monkey climbs on "
 12       GOAL NAME RECALL SHOW
 13       GOAL NAME RECALL MONKEY ON POST
 14       GOAL-SATISFIED TRUE R;
 15

Scr # 8          B:MONKEY.BLK
  0 \ MONKEY  rules ON
  1
  2 R: ON-PHYSOBJ-HOLDS
  3   GOAL NAME RECALL ?NIL NOT
  4   GOAL NAME RECALL ['] PHYSOBJ MEMBER ( goal ) AND
  5   GOAL NAME RECALL
  6    >OBJECT AT RECALL ( xy )
  7    MONKEY AT RECALL = ( same location ? )  AND
  8   MONKEY HOLDS RECALL ?NIL NOT AND  ( state )
  9   --> CR ." 4 need to drop the "
 10       MONKEY HOLDS RECALL SHOW
 11       NEWGOAL ['] HOLDS GOAL TYPE POST
 12               ['] NIL GOAL NAME  POST TRUE R;
 13
 14
 15
```

```
Scr # 9          B:MONKEY.BLK
  0 \ MONKEY  rules ON
  1
  2 R: ON-PHYSOBJ-AT-MONKEY
  3   GOAL NAME RECALL ?NIL NOT
  4   GOAL NAME RECALL ['] PHYSOBJ MEMBER ( goal ) AND
  5   GOAL NAME RECALL
  6    >OBJECT ON RECALL ['] FLOOR = AND
  7   GOAL NAME RECALL
  8    >OBJECT AT RECALL MONKEY AT RECALL <>  AND
  9     --> CR ." 5 need to move " NEWGOAL
 10       GOAL NAME RECALL
 11        >OBJECT AT RECALL GOAL TO POST
 12       ['] AT GOAL TYPE POST
 13       ['] MONKEY GOAL NAME POST TRUE R;
 14
 15

Scr # 10          B:MONKEY.BLK
  0 \ MONKEY rules ON
  1
  2 R: ON-PHYSOBJ-SATISFIED
  3   GOAL NAME RECALL ?NIL NOT
  4   GOAL NAME RECALL ['] PHYSOBJ MEMBER ( goal ) AND
  5   MONKEY ON RECALL GOAL NAME RECALL = AND
  6     --> CR ." 6 monkey already on "
  7       MONKEY ON RECALL SHOW
  8       GOAL-SATISFIED TRUE R;
  9
 10
 11
 12
 13
 14
 15

Scr # 11          B:MONKEY.BLK
  0 \ MONKEY rules HOLDS
  1
  2 R: HOLDS-NIL
  3   GOAL NAME RECALL ?NIL ( goal )
  4   MONKEY HOLDS RECALL ?NIL NOT AND ( state )
  5     --> CR ." 7 monkey drops the "
  6       MONKEY HOLDS RECALL SHOW
  7       ['] NIL MONKEY HOLDS POST GOAL-SATISFIED TRUE R;
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Scr # 12          B:MONKEY.BLK
  0 \ MONKEY rules HOLDS
  1
  2 R: HOLDS-NIL-SATISFIED
  3   GOAL NAME RECALL ?NIL ( goal )
  4   MONKEY HOLDS RECALL ?NIL   AND    ( state )
  5     --> CR ." 8 monkey holds nothing " GOAL-SATISFIED TRUE R;
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15

Scr # 13          B:MONKEY.BLK
  0 \ MONKEY rules HOLDS
  1
  2 R: HOLDS-OBJ-CEILING
  3   GOAL NAME RECALL ?NIL NOT
  4   GOAL NAME RECALL ['] PHYSOBJ MEMBER ( goal ) AND
  5   GOAL NAME RECALL
  6    >OBJECT ON RECALL ['] CEILING = AND
  7   GOAL NAME RECALL
  8    >OBJECT AT RECALL ( xy )
  9   LADDER AT RECALL =  ( same location ? ) AND
 10   MONKEY ON RECALL ['] LADDER = AND
 11   ['] PHYSOBJ ['] ON GOAL NAME RECALL ANY NOT AND
 12     --> CR ." 9 monkey grabs the " GOAL NAME RECALL SHOW
 13       ['] NIL GOAL NAME RECALL >OBJECT ON POST
 14       GOAL NAME RECALL MONKEY HOLDS POST
 15       GOAL-SATISFIED TRUE R;

Scr # 14          B:MONKEY.BLK
  0 \ MONKEY rules HOLDS
  1
  2 R: HOLDS-OBJ-CEIL-ON
  3   GOAL NAME RECALL ?NIL NOT
  4   GOAL NAME RECALL ['] PHYSOBJ MEMBER ( goal ) AND
  5   GOAL NAME RECALL
  6    >OBJECT ON RECALL ['] CEILING =   AND
  7   GOAL NAME RECALL
  8    >OBJECT AT RECALL LADDER AT RECALL = AND
  9   MONKEY ON RECALL ['] LADDER <> AND     ( state )
 10     --> CR ." 10 need to get on ladder "
 11         NEWGOAL ['] ON GOAL TYPE POST
 12       ['] LADDER GOAL NAME POST TRUE R;
 13
 14
 15
```

```
Scr # 15          B:MONKEY.BLK
  0 \ MONKEY rules HOLDS
  1
  2 R: HOLDS-OBJ-CEIL-AT-OBJ
  3   GOAL NAME RECALL ?NIL NOT
  4   GOAL NAME RECALL [']  PHYSOBJ MEMBER AND
  5   GOAL NAME RECALL
  6    >OBJECT ON RECALL [']  CEILING = AND
  7   GOAL NAME RECALL
  8    >OBJECT AT LADDER AT RECALL <> AND
  9     --> CR ." 11 need to move ladder "
 10       NEWGOAL GOAL NAME RECALL
 11       >OBJECT AT RECALL GOAL TO POST
 12       [']  AT GOAL TYPE POST
 13       [']  LADDER GOAL NAME POST TRUE R;
 14
 15

Scr # 16          B:MONKEY.BLK
  0 \ MONKEY rules HOLDS
  1
  2 R: HOLDS-OBJ-NOTCEIL
  3   GOAL NAME RECALL ?NIL NOT
  4   GOAL NAME RECALL [']  PHYSOBJ MEMBER AND
  5   GOAL NAME RECALL
  6    >OBJECT ON RECALL [']  CEILING <> AND
  7   GOAL NAME RECALL
  8    >OBJECT AT RECALL MONKEY AT RECALL = AND
  9   MONKEY ON RECALL [']  FLOOR = AND
 10   MONKEY HOLDS RECALL ?NIL AND
 11   [']  PHYSOBJ [']  ON GOAL NAME RECALL ANY NOT AND
 12     --> CR ." 12 monkey grabs the " GOAL NAME RECALL SHOW
 13       GOAL NAME RECALL MONKEY HOLDS POST
 14       [']  NIL GOAL NAME RECALL >OBJECT ON POST
 15       GOAL-SATISFIED TRUE R;

Scr # 17          B:MONKEY.BLK
  0 \ MONKEY rules HOLDS
  1
  2 R: HOLDS-OBJ-NOTCEIL-ON
  3   GOAL NAME RECALL ?NIL NOT
  4   GOAL NAME RECALL [']  PHYSOBJ  MEMBER  AND
  5   GOAL NAME RECALL
  6   >OBJECT ON RECALL [']  CEILING <> AND
  7   GOAL NAME RECALL
  8   >OBJECT AT RECALL MONKEY AT RECALL <> AND
  9   MONKEY ON RECALL [']  FLOOR <> AND
 10     --> CR ." 13 need to get on floor "
 11       NEWGOAL [']  ON GOAL TYPE POST
 12       [']  FLOOR GOAL NAME POST TRUE R;
 13
 14
 15
```

```
Scr # 18          B:MONKEY.BLK
  0 \ MONKEY rules HOLDS
  1
  2 R: HOLDS-OBJ-NOTCEIL-AT-MONKEY
  3   GOAL NAME RECALL ?NIL NOT
  4   GOAL NAME RECALL ['] PHYSOBJ MEMBER AND
  5   GOAL NAME RECALL
  6   >OBJECT ON RECALL ['] CEILING <>  AND
  7   GOAL NAME RECALL
  8   >OBJECT AT RECALL MONKEY AT RECALL <> AND
  9     --> CR ." 14 need to go to " GOAL NAME RECALL SHOW
 10       NEWGOAL GOAL NAME RECALL
 11       >OBJECT AT RECALL GOAL TO POST
 12       ['] AT GOAL TYPE POST
 13       ['] NIL GOAL NAME POST TRUE R;
 14
 15

Scr # 19          B:MONKEY.BLK
  0 \ MONKEY rules HOLDS
  1
  2 R: HOLDS-OBJ-HOLDS
  3   GOAL NAME RECALL ?NIL NOT
  4   GOAL NAME RECALL ['] PHYSOBJ MEMBER AND
  5   GOAL NAME RECALL
  6   >OBJECT AT RECALL   MONKEY AT RECALL = AND
  7   MONKEY HOLDS RECALL ?NIL NOT AND
  8   MONKEY HOLDS CURLEVEL GOAL NAME RECALL <> AND
  9     --> CR ." 15 need to hold nothing "
 10         NEWGOAL ['] NIL GOAL NAME POST
 11       ['] HOLDS GOAL TYPE POST TRUE R;
 12
 13
 14
 15

Scr # 20          B:MONKEY.BLK
  0 \ MONKEY rules HOLDS
  1
  2 R: HOLDS-OBJ-SATISFIED
  3   GOAL NAME RECALL ?NIL NOT
  4   GOAL NAME RECALL ['] PHYSOBJ MEMBER AND
  5   MONKEY HOLDS RECALL GOAL NAME RECALL = AND
  6     --> CR ." 16 monkey already holds the "
  7       MONKEY HOLDS RECALL SHOW GOAL-SATISFIED TRUE R;
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Scr # 21          B:MONKEY.BLK
  Ø \ MONKEY rules AT
  1
  2 R: AT-OBJECT
  3   GOAL NAME RECALL ?NIL NOT
  4   GOAL NAME RECALL ['] PHYSOBJ MEMBER AND
  5   GOAL NAME RECALL
  6   >OBJECT AT RECALL GOAL TO RECALL <> AND
  7   MONKEY HOLDS RECALL GOAL NAME RECALL = AND
  8     --> CR ." 17 monkey moves the "
  9       GOAL NAME RECALL SHOW
 1Ø       GOAL TO RECALL DUP MONKEY AT POST
 11       MONKEY HOLDS RECALL >OBJECT AT POST
 12       GOAL-SATISFIED TRUE R;
 13
 14
 15

Scr # 22          B:MONKEY.BLK
  Ø \ MONKEY rules AT
  1
  2 R: AT-OBJ-ON-FLOOR
  3   GOAL NAME RECALL ?NIL NOT
  4   GOAL NAME RECALL ['] PHYSOBJ MEMBER AND
  5   GOAL NAME RECALL
  6   >OBJECT AT RECALL GOAL TO RECALL <> AND
  7   MONKEY ON RECALL ['] FLOOR <> AND
  8   MONKEY HOLDS RECALL GOAL NAME RECALL = AND
  9     --> CR ." 18 need to get on floor "
 1Ø         NEWGOAL ['] ON GOAL TYPE POST
 11       ['] FLOOR GOAL NAME POST TRUE R;
 12
 13
 14
 15

Scr # 23          B:MONKEY.BLK
  Ø \ MONKEY rules AT
  1
  2 R: AT-OBJ-HOLDS
  3   GOAL NAME RECALL ?NIL NOT
  4   GOAL NAME RECALL ['] PHYSOBJ MEMBER AND
  5   GOAL NAME RECALL
  6   >OBJECT AT RECALL GOAL TO RECALL <> AND
  7   MONKEY HOLDS RECALL GOAL NAME RECALL <> AND
  8     --> CR ." 19 need to hold the "
  9         GOAL NAME RECALL SHOW
 1Ø         NEWGOAL ['] HOLDS GOAL TYPE POST
 11       \ no need to define goal name - it's the same
 12         TRUE R;
 13
 14
 15
```

```
Scr # 24          B:MONKEY.BLK
  0 \ MONKEY rules AT
  1
  2 R: AT-OBJ-SATISFIED
  3   GOAL NAME RECALL ?NIL NOT
  4   GOAL NAME RECALL ['] PHYSOBJ MEMBER  AND
  5   GOAL NAME RECALL
  6   >OBJECT AT RECALL GOAL TO RECALL = AND
  7   --> CR GOAL NAME RECALL SHOW
  8     ." 20 already at location "
  9     GOAL-SATISFIED TRUE R;
 10
 11
 12
 13
 14
 15

Scr # 25          B:MONKEY.BLK
  0 \ MONKEY  rules AT
  1
  2 R: AT-MONKEY
  3   GOAL NAME RECALL ?NIL
  4   MONKEY AT RECALL GOAL TO RECALL <> AND
  5   --> CR ." 21 monkey walks to location "
  6     GOAL TO RECALL MONKEY AT POST
  7     GOAL-SATISFIED TRUE R;
  8
  9
 10
 11
 12
 13
 14
 15

Scr # 26          B:MONKEY.BLK
  0 \ MONKEY rules AT
  1
  2 R: AT-MONKEY-OBJ
  3   GOAL NAME RECALL ['] MONKEY =
  4   GOAL TO RECALL MONKEY AT RECALL <> AND
  5   MONKEY HOLDS RECALL ['] PHYSOBJ MEMBER AND
  6   --> CR ." 22 monkey walks, carrying "
  7     MONKEY HOLDS RECALL SHOW
  8     GOAL TO RECALL DUP MONKEY AT POST
  9     MONKEY HOLDS RECALL >OBJECT AT POST
 10     GOAL-SATISFIED TRUE R;
 11
 12
 13
 14
 15
```

```
Scr # 27          B:MONKEY.BLK
  0 \ MONKEY rules AT
  1
  2 R: AT-MONKEY-ON
  3   GOAL NAME RECALL ['] MONKEY =
  4   GOAL TO RECALL MONKEY AT RECALL <> AND
  5   MONKEY ON RECALL ['] FLOOR <> AND
  6   --> CR ." 23 jump to floor "
  7       NEWGOAL ['] ON GOAL TYPE POST
  8     ['] FLOOR GOAL NAME POST TRUE R;
  9
 10
 11
 12
 13
 14
 15

Scr # 28          B:MONKEY.BLK
  0 \ MONKEY  rules AT
  1
  2 R: AT-MONKEY-SATISFIED
  3   GOAL NAME RECALL ['] MONKEY =
  4   GOAL TO RECALL MONKEY AT RECALL = AND
  5   --> CR ." 24 monkey already at location "
  6     GOAL-SATISFIED TRUE R;
  7
  8
  9
 10
 11
 12
 13
 14
 15

Scr # 29          B:MONKEY.BLK
  0 \ MONKEY rules DONE
  1
  2 variable DONE  \ a flag set by CONGRATS or SORRY to end run
  3
  4 R: CONGRATS
  5   GOAL TYPE RECALL ?NIL
  6   --> CR ." congratulations " CR TRUE DONE ! TRUE R;
  7
  8 R: SORRY
  9   GOAL TYPE RECALL ?NIL NOT
 10   --> CR ." sorry " CR TRUE DONE ! TRUE R;
 11
 12
 13
 14
 15
```

```
Scr # 30          B:MONKEY.BLK
  0 \ MONKEY  RULELIST
  1
  2 1 1 OR: {ON}
  3   ON-FLOOR ON-FLOOR-SATISFIED ON-PHYSOBJ
  4   ON-PHYSOBJ-HOLDS ON-PHYSOBJ-AT-MONKEY
  5   ON-PHYSOBJ-SATISFIED L;
  6
  7 1 1 OR: {HOLDS} HOLDS-NIL HOLDS-NIL-SATISFIED HOLDS-OBJ-CEILING
  8   HOLDS-OBJ-CEIL-ON  HOLDS-OBJ-CEIL-AT-OBJ
  9   HOLDS-OBJ-NOTCEIL HOLDS-OBJ-NOTCEIL-ON
 10   HOLDS-OBJ-NOTCEIL-AT-MONKEY HOLDS-OBJ-HOLDS
 11   HOLDS-OBJ-SATISFIED L;
 12
 13 1 1 OR: {AT}  AT-OBJECT AT-OBJ-ON-FLOOR AT-OBJ-HOLDS
 14   AT-OBJ-SATISFIED AT-MONKEY AT-MONKEY-OBJ
 15   AT-MONKEY-ON AT-MONKEY-SATISFIED  L;

Scr # 31          B:MONKEY.BLK
  0 \ MONKEY controls
  1 : STARTUP  ['] NIL GOAL TYPE POST NEWGOAL
  2   ['] HOLDS GOAL TYPE POST ['] NIL BLANKET ON POST
  3   ['] BANANNAS GOAL NAME POST \ 11 BLANKET AT POST
  4   ['] CEILING BANANNAS ON POST 11 BANANNAS AT POST
  5   ['] COUCH MONKEY ON POST \ 43 MONKEY AT POST
  6 \ ['] FLOOR MONKEY ON POST 23 MONKEY AT POST
  7   ['] FLOOR COUCH ON POST 43 COUCH AT POST
  8   ['] FLOOR LADDER ON POST 41 LADDER AT POST
  9   ['] BLANKET MONKEY HOLDS POST ;
 10 R: ?ON GOAL TYPE RECALL ['] ON =
 11   --> {ON} R;
 12 R: ?HOLDS GOAL TYPE RECALL ['] HOLDS =
 13   --> {HOLDS} R;
 14 R: ?AT GOAL TYPE RECALL ['] AT =
 15   --> {AT} R;

Scr # 32          B:MONKEY.BLK
  0 \ MONKEY  control
  1
  2 1 1 OR: RULES CONGRATS ?ON ?HOLDS ?AT SORRY L;
  3
  4 : RUN  ( DARK ) E5INIT
  5   STARTUP FALSE DONE ! 0 >LEVEL
  6   ." Starting goal is " GOAL TYPE RECALL SHOW  CR
  7   BEGIN   \ the main loop
  8     ['] RULES TRUTH DROP   \ RULES gets the message TRUTH
  9     DONE @                 \ drop truth returned by RULES
 10   UNTIL ;                  \ and check variable DONE
 11
 12
 13
 14
 15
```