90-9

PRODIGY:
Its Exploration and Use

Ralph H. Freeman

90-9
MLI-90-10

PRODIGY

ITS EXPLORATION AND USE

Ralph H Freeman

Center for Artificial Intelligence
George Mason University
4400 University Drive
Fairfax, Va 22030

Abstract

The purpose of this paper is to document the exploration and use of a general purpose planner and learning system called *Prodigy*. The planner was designed to be domain independent and to incorporte an Explanation-Base Learner (EBL). Since its purpose was to serve as testbed for research, the planner was exercised by creating a 2-Robot, cooperative, three dimensional world. This new world stesses the existing paradigm by requirings a virtual doubling of operators and search rules with an order of magnitude increase in search time.

Acknowledgements

Prodigy is a general purpose problem-solving architecture that serves as the basis for research in planning, machine learning, apprentice-type knowledge-refinement interfaces, and expert systems. It includes both a closed world type planner and an Explanation Based Learning (EBL) module. The program was developed by Steven Minton, et. al. at the School of Computer Sciences, Carnegie Mellon University, Pittsburgh, Pa. 15213.

It is with sincere gratitude that I acknowledge the work performed by Steve Minton and the many others that worked with him in development of *Prodigy*. It provided an excellent basis for my research and understanding into the fundamentals of planners of the type typified by Prodigy and their extension into example based deductive learning systems.

In the day-to-day environment, maximum support for this project was received from Ken Kaufman, Pavel Stefanski and Ryszard Michalski of the GMU Machine Learning & Inference Laboratory. Without their help, the project would quite literally never have begun.

This research was conducted in the Center for Artificial Intelligence at George Mason University. Research of the Center for Artificial Intelligence is supported in part by the Defense Advanced Research Projects Agency (DARPA) under grant, administered by the Office of Naval Research, No. N00014-87-K0874, in part by the Office of Naval Research under grant No. N00014-88-K-0026, and in part by the Office of Naval Research under grant No. N00014-88-K-0397.

CONTENTS

1. RESEARCH PURPOSE

This report is designed to document the findings made on three different perspectives of the *Prodigy* planning system. The first is a general discussion of the program, its functions, utility and ease of use. In a larger sense, this discussion may be considered as an extension of the Prodigy 2.0: User Manual and Tutorial [4].

A second purpose is to provide a basis for understanding the Explanation Based Learning (EBL) Module that accompanies the basic program. In implementation, this package is called Explanation Based Specialization (EBS). The EBL section contains some implementation hints as well as a review of the lessons learned.

The final section of the paper contains the findings of the implementation of a two-robot variation of the 3-Dimensional GRIDWORLD domain.

The *Prodigy* program is written in Common Lisp and and a high level shorthand language, *Prodigy* Description Language (PDL). While it may not be strictly necessary to understand the syntax of LISP to appreciate this report, it is certainly a prerequisite to using *Prodigy* in order to define new domains. Example runs of existing domains can be made using only *Prodigy* provided user commands.

2.0 INTRODUCTION TO PRODIGY

Prodigy is a general purpose testbed planner that was developed by Steve Minton et. al. at Carnegie Mellon University. Its basic purpose is to provide a vehicle for testing concepts for machine learning and planning. Towards this ends, the system contains a planner, a learning module and a set of example domains. The program used for this project was Version 2.0 and was completed May 1989. The disclaimers of experimental software should be noted. All facets of the software are still under revision and examination of the code indicates numerous instances of "hacked" code. Nonetheless, for an academic or research environment, precisely the purpose for which *Prodigy* was designed, it is an excellent learning paradigm.

2.1 System Overview *Prodigy*

It is not the intent of this paper to reproduce the user's manual that has been provided for the planner. The work done by Minton in the documentation file which accompanies the program is excellently written and covers the essentials needed for operation of the program. Despite their efforts, however, the manual provides only a superficial view of some very complex coding and file interactions. Once a working knowledge of the detailed code is gained, many of the subtleties buried in the existing manual can be exorcised. The converse is not possible. It is not possible to read the manual and gain these insights into program operation at the code level. For this reason, an extension of the Manual and Tutorial is provided for those who do not have a grounding in either planners as a class of machine reasoning systems or in *Prodigy*. [4]

Prodigy comes from a class of planners known as STRIPS world planners. The name is derived from a robot-domain world created as the STanford Research Institute Problem Solver. The purpose of the original program was to provide a small robot with a plan to navigate between several rooms and retrieve blocks of various sizes and descriptions. The basic thought process was based upon still earlier work in the General Problem Solver (GPS). Both STRIPS and EXTENDED-STRIPS domains are provide for user exploration but take heed of the warnings provided.

To accomplish this task, STRIPS and *Prodigy* make some basic decisions about the world in which they would operate. The first of these is called the closed world assumption. Under this idea, everything that is true in the world is expressed in the state description of the world. Note that anything not explicitly mentioned in the state description is assumed false (hence the name: closed world). Furthermore, the only way to change the state description is through the application of some "operator". This basic structure assumption leads to the creation of operators which have two basic parts to them. The first part of the operator consists of the preconditions which must be true if the operator is to be applied and the second part of the operator is the description of those things in the state description which must be altered to describe the "new world" after execution of the action. In STRIPS this function was accomplished through the ADD and DELETE lists of the operators. In *Prodigy* it is accomplished directly through the "rhs" effects condition of an operator.

As a practical planner a STRIPS approach has several fallacies. One of the most damaging is the inability of any designer to fully capture all of a complex domain in explicit

statements that are true in the real world. One method of circumventing this problem is to list the relevant facts of the world and then establish domain axioms, inference rules, that explain the relationship between known data and their implications. Thus, many propositions can be deduced from the enumerated state description. For instance, the fact that a robot is in an ARMS-EMPTY condition is deducable from the world state where no condition of the form ROBOT-HOLDING BLOCKx exists. *Prodigy* is an extension of the basic STRIPS approach which captures this capability. The user is allowed to define any number of INFERENCE rules and this capability provides a limited "open world" within the *Prodigy* planner.

The search approach for the planner is the heart of the problem for machine reasoning systems. In GPS, the technique espoused was called a MEANS-END analysis. Under this concept, the planner would compare its current state to its goal state and derive some measure of <u>distance</u> to the goal. By prior knowledge it had a measure of how much <u>distance</u> could be reduced by the application of each candidate operator. It then picked the most likely candidate, executed it, and moved on to the next step. In the event that a path failed to reach the goal state somewhere along the path, the planner regressed to the last successful position and tried the next candidate operator. In this MEANS-END fashion, a sequence of operators (a plan) was generated. In classical planning systems, this process is considered complete in the sense that if a set of operators exists to get from state_1 to state-end, the planner would eventually find it. Nothing, however, is said about the timeliness of the answer.

Prodigy, like STRIPS, follows this same basic search process but guides the search through a different heuristic. The

utility function for distance is implicitly describe in the SEARCH-CONTROL-RULES developed by each domain designer. All rules are defined into four categories, each of which has three discrete subsets. The rules are unique for dissimilar domains (worlds). First consider the earlier discussion on the search technique. Each time GPS stopped to consider its world and compare it to the desired final state, it was at a NODE. A NODE embodies the state of all the known variable assignments (BINDINGS), their relation to each other (PREDICATES), the GOAL or SUBGOAL needing satisfaction, and the OPERATOR applied to try to achieve it. In *Prodigy*, 'NODE, 'GOAL, 'OPERATOR, and 'BINDINGS, in this precedence, form the four basic control mechanisms for the search routines. Within each of these, three subsets are possible: whether the effect of the rule is to be SELECTED, PREFERRED, or REJECTED in that precedence. The default setting in the program is to take the SC-RULES in the order presented. Using a rule, it is matched to the existing world state and the search options available and the first rule which matches is used to select first the NODE, then the GOAL, then OPERATOR and finally a set of variable instantiations (BINDINGS). In this fashion *Prodigy* steps through a search tree looking for a plan. Examples of SC-rules, domain-runs, operators etc., can be found in the third section of this report on the implementation of the two-robot GRIDWORLD.

One final point on the basic methodology being applied. *Prodigy* actually searches from the goal state backwards to the current state. It is a process of backward chaining. Specifically, if a robot needs to be at LOCATION A and is initially at LOCATION B, the searcher starts at LOCATION A and asks itself to satisfy this goal. If it can not, it then considers where it can move that is towards the initial state

conditions.

2.2 System Architecture

Prodigy is organized as a set of COMMON LISP files. In many cases both a compiled (binary) version as well as the source file exists. The startup routines are structured to seek the binary file if it exists. There are three fundamental groups of programs. Under the directory "/prodigy/system" are two sub-directories containing the EBL learning module and the Planning engine, called /ebl and /planner, respectively. The file directory /ebl contains all files necessary to execute the Learning system while the file directory /planner has the independent domains that describe various problem worlds. All three of these directories contain a file called "startup.lisp" which is the primary controller used to load all other required files in that particular directory as well as intitialize various control parameters. Among other things, each *startup.lisp* contains the *World-PATH* attribute for its directory, which describes the file structure to all the other files. This attribute must be reset to the *Prodigy* file structure established for the target host computer and userfiles. **HINT:** To execute the planner on the SUN Machines in the MLI Center, change your work directory to the domain of interest e.g. cd prodigy/domains/gridworld, and then execute the following LISP command, (load " ~ username/prodigy /system/planner/startup.lisp") This presupposes you have a copy of all *Prodigy* files in the appropriate file structure under "username" and have corrected the *WORLD-PATH* attribute in the "startup.lisp" file of the planner and domains. The learning system must be loaded separately and after the planner is loaded. After the files are loaded, refer to Minton's Manual and Tutorial for instructions on the operation of the numerous user commands and options.

One final word on the implementation as of this writing. *Prodigy* was designed to operate under X-Windows. Neither conversion of *Prodigy* to the SUN windows or conversion of X-Windows to a SUN implementation has been done as of this report. As a result, none of the graphics capability possessed by the program is accessible. Attempts to use any function of graphics will lead to LISP run-time errors. Understanding the default text output is both tedious and time consuming. The printouts are about as well organized as they can be made but they are voluminous (see Appendix). In addition, analytical tools exist within the graphics package for tracing the trees and understanding the program's actions. Without them, the learning process on planners is less than one might expect. Correction of this deficiency should be considered of primary importance if *Prodigy* is ever to attain its full potential.

3.0 EXPLANATION BASED SPECIALIZATION (EBS)

The EBL directory of *Prodigy* comes with an explanation of how to use the EBS program module but it is much less clear than the planner documentation. It covers operation but does not well explain exactly what the learning system is, what its attributes are, and what can be expected as an outcome. Fortunately, additional information is contained in Minton's book / thesis which can serve as both a *Prodigy* manual and a tutorial on Explanation-based Learning Systems [3]. In a fashion similar to that used in Section 1.0, this chapter will provide a user viewpoint on the use of the EBL/EBS system provided within the *Prodigy* program. It should be carefully noted that the planner can/does work independently of the learner. The converse is not true. A knowledge and understanding of the learning mechanism is NOT a prerequisite to using the planner and, in fact, the default (toggle-ebl) is set to NIL to disable the learner.

Explanation Based Learning (EBL) is one of several learning paradigm currently in vogue. As implemented in *Prodigy*, its purpose is to learn new search control rules that will help guide the direction of node expansion as the planner attempts to attain a goal (see Section 1). As a technique, EBL is an analytic paradigm, based on learned examples, that uses deduction as its primary inference tool. It can not discover "new" facts but can only recombine existing facts into new combinations or clusters/segments. In concept, this is little different than the development of macro-operators used in the solution of the traditional 8-puzzle problem. (This is the child's puzzle with 8 numbered squares inside a 3 x 3 block. The object is to rearrange the numbers into a desired sequence. One solution technique is to develop a function

which maps the sequence of moves needed to move from any position to any other position on the board, leaving all other pieces unchanged - a macro operator. After the macro-operator is developed, no search is needed to solve any problem. A call to the appropriate macros provides a solution plan [5:19]). The problem with this approach, of course, is that if the facts are already contained in the domain, then it is a trade-off between search time to find the path in the domain versus search time to find the right macro operator. If large numbers of "new" operator rules are learned, it could take longer to search the list of new rules than to re-solve the problem directly. These issues are addressed in Minton's EBS module for *Prodigy*.

3.1 EBL and *Prodigy*

The EBL method employed in *Prodigy* is called Explanation-Based Specialization. The concept is quite simple but the implementation and coding is equally complex. The system starts with a target concept, in the case of *Prodigy*, the concepts are SUCCESS, FAIL, GOAL-INTERACTION. A specific condition is observed, called a training example. Using the domain rules of manipulation syntax, EBL attempts to find a set of transformations which explain why the training example is a specialization of the target concept. The set of transforms under which this proof is true, is called a learned description and becomes a "new" search control rule for the planner.

In addition to the basic process of EBL, *Prodigy* employs two other features which are fully described in Minton's book. Thes features are designed to improve the human understanding of the new rules and to address the expected value of adding the rule to the planner. After generating a proposed learning

rule, the learner critiques itself by simplifying the expression (called compression by Minton) and then evaluating its usefulness through a predefined utility function. The function is an "estimated" time dependent algorithm which seeks to compare the apriori solution time against the posterior search time with the rule included. Control rules which do not score sufficiently well are excluded from the new rules list, stored under *SAVED-RULES*. (*Prodigy* keeps with the naming convention of placing *s before and after global variables, e.g. *SAVED-RULES*).

3.2 Methodology

For each of the search control types mentioned in Section 1, there are four target concept types predefined in *Prodigy*, e.g. GOAL-INTERACTION. These four are only one of many learner strategies or concept sets that could be used. The *Prodigy* EBL list is extensible by the user [3:52-53]. The learner, refered to as OBSERVER in the program, searches the nodes of a tree for interesting examples of the identified target concepts. It defaults to searching children before parents and from left to right. The documentation claims the process can be interactive or applied after a solution is reached, however, in this report only the latter was tried.

3.2 EBS Application

After a training example is selected, it must pass some heuristic tests of "promise" to be passed to the learner. Succeeding in this, the EBS module will attempt to specialize the target concept, guided by (instantiated?) the training problem. The process in *Prodigy* goes as follows:

Within the program are two sets of domain rules. The first

are handcoded architectural schemas and the others are derived by the system from the domain specifications, domain-level-schemas. The former is a domain independent description of what it takes to SUCCEED (node, operator, goal) for instance. The latter is a descriptor that transitions the system from domain independence to dependence. The descriptors take the variables described in the architectural schemas and explain them in terms of the domain. Suppose SUCCEED happens if (WORKS operator). The descriptor for WORKS might be defined as IF SHOVEL. Obviously shovel would only be appropriate in domains requiring one whereas a water environment might define WORKS as IF FINS. Thus at this level, *Prodigy* transitions to domain dependence. Given these schemas, the EBS module performs a theorem_proving match, recursively, on the definitions until only primitives are left. It then instantiates the learned description and uniquely renames all the variables. The output is a macro-operator that states something like this - use operator$_x$ at node$_n$ IF goal$_z$ & IF (precondition, precondition, etc). This process is not a search technique but rather a matching. As a result, it grows O(n) with the number of nodes expanded.

3.3 Utility Function

A major point of all EBL systems concerns the timeliness of a response. For *Prodigy*, Minton expressly addresses the utility of his EBS system through the use of a utility function based solely on time savings [3:79]. The premise is succinctly stated as finding the time savings from the new rule by subtracting the old time it took from the new time. Unfortunately, the means of collecting the precise value of the new solution time requires a repeated run of the problem which has already been solved. This not being practical in as

a routine way of business, Minton implemented a strategy for estimating the new run time. The derivation is contained in his thesis. Weeding of the generated rules then takes place across a set of problems with the rules that survive being saved for planner execution. A separate list contains the discard file for user review. This is necessary because of the inherent serial application of rules within *Prodigy*. It is possible a few "better" rules may be placed on the inactive list because they were shadowed by a preceding less capable rule which always fired first. Thus the second rule would never receive "credit" as being useful even though it might actually be a superior rule.

3.4 Execution

The documentation stored within ebl.doc is quite sufficient for implementation of the ebl module. The *EBL-WORLD-PATH* must be set correctly in /ebl/startup.lisp. For efficiency, it is suggested that only the compiled binary version be loaded to improve execution time. The number of source lines of code for ebl is greater than that of the planner. The planner should be loaded first, followed by ebl and then your domain and specific problem/problem-sets. If it should be necessary to implement a different domain, it is necessary to exit and reload both the planner and the ebl module. On several occasions, program bugs made it necessary to exit the LISP environment completely and reload before continuing. The planner is fairly straightforward but the ebl module has numerous hidden hooks which cause it to get hung up occasionally. Once the two packages have been loaded, it will be necessary to "toggle-ebl" from either within *Prodigy* or by enclosing the command in parenthesis, as a LISP statement e.g. (toggle-ebl). Other commands are available as needed.

4.0 Execution in a 2-Robot Gridworld

This chapter will cover the execution of both the planner and the learning module against an adaptation of the *Prodigy* 3-Dimensional Gridworld. Several of the *Prodigy* provided domains were also used for comparative purposes. It was not the intent of this project to provide simply another domain but rather to study *Prodigy* and its implementation. Thus in many cases I terminated further development of a problem area when it became clear what would have to be done and concentrated on exploring additional facets of the program. Within that framework, I found a number of things *Prodigy* could do and several that it could not. What follows is a compilation of the different things *LEARNED*. If you are not familiar with Minton's The Manual and Tutorial as well as Sections 2 and 3 of this report, it is strongly suggested that the process of learning will continue more quickly if the readings are accomplished first.

4.1 Gridworld

The major problem used in this report to explore the boundaries of *Prodigy* was a mutated Gridworld, i.e. the adding of a second robot to Gridworld. Gridworld is a three-dimensional world that is 5-squares on a side. The world is composed of robots and blocks. The robots have a reach of 2 units and can coexist with a square once it is picked up. Furthermore, although the squares can be of size greater than one cube, once retrieved by a robot, it is handled as if it were only size = one. (This allows it to pass between obstacles that would otherwise be impassable). The robots can move, pickup and putdown. They can not exist unsupported in

the air, but they can build steps, arches and navigate themselves.

4.1 Domain Definition

In order gain an understanding of the task to be preformed, it was first necessary to investigate how a domain was established. The documentation explained what it would do and what the basic elements were but provided very little insight at the primitive level of how to construct a domain.

Domains are created around four basic files: SC-RULES.LISP, FUNCTIONS.LISP, /PROBS/filename.lisp and DOMAIN.LISP. These four files contain, respectively, the search control rules including INFERENCE-rules, the primitive functions of the domain, the problem specification, and the domain OPERATORS. *Prodigy* was designed as a domain independent planner. The file architecture is setup to clearly differentiate the domain specific data from the basic planner engine. However, it is not well documented that the domain files call predicate primitives that are defined in the host LISP environment and back in the Planner directory. The only mention of a non-domain interconnection is a planner file called META-FNS LISP which houses some user defined predicates of global interest. For instance, in the domain unigue directory, FUNCTIONS.LISP contains predicate definitions similar to the following:

```
(defun next-to (loc sqloc)
  (cond (( and (is-variable loc) (is-variable sqloc))
               'no-match attempted)
        (( and (is-variable sqloc) (is-location loc))
               (binding-list sqloc (next-to-gen loc)))
        (( and (is-location loc) (is-square-location
                sqloc))  (NOT (null (member sqloc
                          (next-to-gen loc) :test
                          #'equal))))
        (T (type-error 'next-to))))
```

NEXT-TO is a boolean operator to determine if location "loc" is next-to a square-location "sqloc." Since FUNCTIONS.LISP is in the domain architecture, it appears at first glance to be domain independent. However, the predicate IS-VARIABLE is defined in the Planner under DATA-TYPES.LISP. The function MEMBER is the standard LISP operator and BINDING-LIST is another FUNCTIONS.LISP operator. Since these functions, plus several hundred more, form the allowable syntax and semantics of the planner, it is extremely difficult to determine how to construct legal statements without "reverse engineering" all of *Prodigy*. There is no guidance at the coding abstraction level in the documentation and reverse engineering is extremely difficult because of the cross-connections between files in the large planner program. The problem of locating the files can be alleviated by creating an index of what functions are defined in which files (see Appendix 6.6). Once the basic primitives are understood, it is possible to move to the two higher level files written in PDL. Thes files, SC-RULES.LISP and DOMAIN.LISP, contain the search rules and operators for the domain.

The Prodigy Description Language (PDL) forms the higher level abstraction language needed to describe SC-Rules in user compatible terms. For example purposes consider the following:

```
(PUT-DOWN-ANYWHERE
   (priority 0)
   (lhs  (current-node nx)
         (current-goal nx (¬(holding obj))
         (current-op nx PUT-DOWN)
         (known nx
            (and (at RMG robot-loc)
                 ( next-to robot-loc new-loc1)
  (rhs (select bindings (obj new-loc1 robot-loc))))
```

In section 2.0, the basis of these search control rules were discussed. The intent is to match the left side rules (lhs) and then to perform the action indicated by the effects lists of the right hand side (rhs). This example brings out several points of interest. First, the existing *Prodigy* domain in Gridworld treated all search rules at the same priority. This means that the default bias of the system is predicated upon the order in which the rules are entered. This cultural bias or equally likelihood approach does not appear to be harmful on small problems but as the number of search criteria went up, it appears that increasing attention is needed to structuring the search rules. How you would set the precedence, apriori, is an open question. Second, note that this rule is a bindings-select rule. There are three categories of search-rules which the existing EBL module can not handle NODE SELECT, NODE-PREFERENCE, and BINDING-SELECT. Thus, it is necessary to delete the bindings select rules if the Learning System is to be enabled. When this is done, a detrimental effect on the speed of the planner's search will be apparent. Finally, PUT-DOWN-ANYWHERE, like the other SC-RULES, makes use of non-domain predicates. For instance, the NEXT-TO predicate is from the domain functions list, but the CURRENT-GOAL predicate is from the Planner definitions and the AT turns out not to be a predicate at all but is instead a constant. Trying to understand the PDL in order to describe the search criteria is no small task. As a further item of complexity, many of the predicates used are called GENERATORS. This means that if their arguments are constants, they return a T/F on the semantic meaning of the predicate, but if the arguments are a variable, they return a list of bindings which fulfill the predicate. Thus, the instantiation at the time of the call determines the response. In the example above, KNOWN is such a generator. In this search rule, it returns a bindings-list but in other contexts it will be used to test a

True/False condition.

The final element in the domains architecture is the Operators. Operators actually come in two forms, inference-rules and action operators. The initial Gridworld consisted of only three actions, PICKUP, PUTDOWN, and MOVE. There were six inference-rules that allowed the robot to deduce things from its state description. Since they are syntactically the same consider the Inference operator INFER-CLEAR.

```
(INFER-CLEAR
  (params nil)
  (preconds
    (exists (loc) (at obj loc)
       (exists (above-loc) (above-loc top-loc loc)
         (vacant-loc top-loc))))
  (effects
    (( add (clear obj)))))
```

This operator uses two calls of the generator, EXIST, to develop bindings for AT and ABOVE-LOC. In words, if a location exists with an object on it and there is a location above the object location which is vacant, then the state description can be inferred to include that the object is clear. As the domain scales upward, the number of inference operators possible grows as a polynomial function. There are several limitations to creating these domain operators when EBL is to be used. The definitions must exclude IF clauses in the effects list and they must not have a form of the type $\neg EXISTS_x$. If generators are used, they can not include disjuncts of generators e.g. (AND (OR $Exists_x$ $Known_y$)).

Prodigy is a well formed program. The domain architecture attempts to simplify the user's ability to create new domains. The only caveat is that a thorough understanding of the primitives and idiosyncracies is necessary before

experimentation can begin. Suggested improvements would be providing a listing of all predicates legally available to a domain description from other files and a separate listing of all generators. If coupled with either a cookbook example of a new domain construction or a fully documented explanation of an existing domain, it would dramatically reduce user confusion.

4.2 2-Robot Domain, the Deltas

To modify the Gridworld Domain to accept two robots, a number of changes were made and as of this report are still being made. The initial aim was to create an adversarial position between the two robots. This occurs when each robot has a competing goal. Because of the way *Prodigy* works, this has proven to be impractical. The planner as currently coded assumes a sequential ordering of tasks for the agent of interest. Adversarial systems assume a parallel ordering. The planner algorithm coded is tailored to suit the former and not the latter. To work around this basic assumption using SCR_RULES and OPERATORS would generate an explosion of rules that would slow the planners sequential search algorithm to the length of finite time. For instance, in the MULTI-ROBOT domain of an extended STRIPS world, the two cooperating robots whose taskings allow sequential stepping, required 17 search reject rules. Nearly all of these deal with the relationship between the robots. Because of the sequential nature of the problem, if two robots are necessary to move the object, the planner eventually gets around to bringing the other robot. In an adversarial role, the other robot needs to be simultaneously moving to negate his opponents move. *Prodigy* does not have any practical way of providing this.

Nonetheless, it did prove practical to modify the Gridworld

for two robots when they were to act in a non-aggressive, cooperating goal world. At the same time the two robots were being added, it was necessary to make changes to allow the system to operate with EBL. The first change attempted was to remove the -Existential in the INFERENCE RULE used in Section 4.1: Infer-Arms-Empty. This was done by altering the existential generator, (¬(exists (ob) (holding ob))) to the universal generator with a corresponding negation of the predicate, (FORALL x (¬(Holding x))). While this was an elementary logical procedure, its effectivity and reliability depended on how the two generators had been coded to the program. Line by line comparison of instantiations between runs using the modified and standard domain indicated equivalent node expansion. The new domain was slower however. This was attributable to the fact that before the change, the planner needed to return a list for only those items in the HOLD state, a linear matching time, but after the change, all variables in the state had to be iteratively tried, resulting in many more system cycles and a slower response. For small problems, it is a minimal time increase (averaging a second in the runs observed) but larger complexity problems would be expensive. The change was necessary to evaluate the EBL module.

Changes in the operators were constrained by the limitations of the learning system and the requirement to add a second robot. In order to define the domain, a distinction must be made as to which robot is using PICK-UP, PUT-DOWN, MOVE, or any of the inference rules. Simultaneously, a limitation is imposed by the learner which disallows any conditional IF clauses in the effects list of the operators. To accomplish the former and circumvent the latter, it was necessary to duplicate each of the search rules for the second robot rather than use the more elegant approach taken in the MULTI-ROBOT

domain. The Domain.lisp file in the Appendix describes MOVE-II, PUT-DOWN-II, etc. These modified operators were run against the original baseline problem sets to determine what effect the alteration had.
Again, the planner was able to reach solution with the duplicated operators but it began to exceed the default *PRODIGY-TIME-BOUND* of 300 seconds. This is not surprising since depth first searches increase polynomially with nodes and the operator list had just been doubled.

An examination of the solution sets indicated that the planner was attempting paths using a brute force search technique. Binding were being attempted which were obviously inappropriate. For instance, (PUT-DOWN-II rmg loc-x) is obviously incorrect since the II signifies a robot2 action and the object of investigation is not an object but robot1. At this point, several alternative solutions are available, all dealing with the search control rules. First, a binding selection rule could be created of the following form:

```
(PICK-PDII
   (lhs (and (current-node n)
             (current-op n PUT-DOWN-II)
             (candidate-bindings n (thing)
             (¬(is-type thing ROBOT)))
    (rhs (select bindings (thing))))
```

The effect of this rule would be to select bindings only when "thing" is not of type ROBOT. In order to implement this rule, a new data type must be created, ROBOT. This is done by using the data type in two places. First, the type must be used in a search control rule. This fact was discovered through trial and error useage of the extensive *Prodigy* run-time error message facility. It may be that a new-type can also be specified if it is used in an OPERATOR but this was one of the areas where furhter experimentation was curtailed

in order to explore other facets of *Prodigy*. Furhtermore, the error message encountered specified only the "effects" list, rhs, of a rule is legal for creating new data types, but an experiment with using it on the lhs of the rule worked. Therefore, the requirement may be simply for it to be used in one or more of the domain descriptions. Second, it must be specified in the state description of the problem set. Failure to do this resulted in *Prodigy* defaulting to the premise that the robots were of type OBJECT which was indistinguishable from BLOCKS. As such, they were used to instantiate the wrong variable.

Using binding-select option, however, is inappropriate due to the old nemesis, EBL. The learner is not coded to be able to handle select_NODES or select_BINDINGS or prefer_NODES. An alternative but more cumbersome solution does exist. The same functionality can be accomplished through the use of reject_BINDINGS:

```
(PICK-PDII-reject
   (lhs (and (current-node n)
             (current-op n PUT-DOWN-II)
             (candidate-bindings n (thing)
             (is-type thing ROBOT)))
    (rhs (reject bindings (thing))))
```

This modification serves the same purpose but is more time consumming.

Each of the above changes was eventually identified and the domain was modified to incorporate them. Numerous runs were then made comparing the interaction of robots. Analysis of the runs continued to show the necessity of ever more complex search rules. In addition, requirements were identified for inference conditions which had not been anticipated. Of

been made to see if some improvement could be made in solution times. In this fashion, test data would be generated for analysis of EBL's capability and if warranted, new rules could be added to improve the 2-Robot operation.

In accordance with the EBL prohibitions, additional modifications were made to the existing search control rules to remove the two SELECT-BINDINGS rules. One of these was PUT-DOWN-ANYWHERE.

```
(PUT-DOWN-ANYWHERE
  (lhs (and ( current-node n)
            ( current-goal n (¬(holding x)))
            (current-op n PUTDOWN)
            (known n
               (and (at rmg rmg-loc))
                    (next-to rmg-loc new-loc)))))
  (rhs (select bindings ( x new-loc rmg-loc))))
```

This rule had been added to the list of constructs as a time-saver. It simply stated that if the robot had to meet a goal of ARMS-EMPTY then it could put its block down in any new-loc next_to its current location. It shortened the search time by mandating a select-rule rather than just considering the instantiation as one of many to be expanded. Consideration of this rule, however, revealed an interesting connection between the closed world assumption, the goal choice and the bindings. In order for the current-goal to be ¬Holding block_x as specified in the precondition, the proposition must not already exist in the current world state. This is true because if it already existed, the system would not be attempted to expand it at this new node. Since its negation is implied by the closed world assumption, the robot must be Holding $block_x$. Therefore, when writing the control rule, it is not necessary to test EXISTS robot Holding $block_x$ -- that is already assured by the precondition goal test. The closed

world assumption has, in effect, acted as an inference mechanism rather that a constraint and simplified a search control rule.

MOVE-DELETE_RMG was the other rule to be removed. If it was needed that the robot not be located at some location, this rule picked a location binding to move the robot to a new location. Both rules had a robot2 version which was also deleted for a total of four rules removed from the 2-Robot domain. Neither pair of these rules appeared as primary domain rules but rather were secondary effects added to improve the speed of solution. The effect of their deletion was hidden was not apparent in the test problems run.

With all learner caveats met, the *Prodigy* planner and EBL were loaded. In the first test, EBL was given only a single problem set - MOVE ROBOT2 from his existing location, 1 space, to a new location. (See Appendix 6.3) The result was the generation of four new search rules. Consider just one of these, R-1-10 in the generation listing or "sr2" in the saved rule listing:

```
(R-1-10
    (lhs (and (current-node n)
              (current-goal n (AT ROBOT2 location_x))
              (candidate-operator n MOVE-II) ; move robot2
              (known n (¬(object ROBOT2 y)))))
    (rhs (select operator MOVE-II)))
```

The context of this learning example was the following: When the planner expanded N2, it first tried to PUTDOWN RMG2 to attain the goal. When this branch eventually failed because no further operators or bindings remained to be tried and the goal had not been achieved, the system backtracked to N2. Its second attempt was with MOVE-II. This eventually led to a successful attainment of the goal. This triggered the target-

down the tree. This means that even though the correct operator might be chosen at a node, the planner is forced to expand incorrect operators before preceding. Using this option, the first problem forced the module to create twenty additional rules, twelve of which passed the utility test. It also deleted several of the older versions which had not lived up to expectation.

The EBL module proved to be quite successful within the scope of these experiments. It was effective at finding useful search control rules which upon subsequent application were valuable additions to the domain controller. Further research would undoubtably reveal more insight into both the module and the newly created 2-Robot domain.

4.4 Summary

The experiment with defining a domain in a 3-Dimensional world using 2-robots is continuing. There are many intriguing issues to explore and many more search rules to be added if the planner is to function in a time effective and descriptive manner. The EBL module will help with the first but not the second. This is without addressing the problem of a competitive goals for the robots. Working with the domain descriptions is not particularly difficult, *once you know what constitutes a legitimate predicate.* Deciding what predicates, search rules, and operators to add IS THE DIFFICULT TASK. The user and EBL have the same problem, in that each addition drives up the time to solution by a factor of at least n^2. Thus small "toy" domains, easily described, are handled well by *Prodigy*. The larger and more complex the problem, the less satisfactory the domain descriptions were able to perform. This summation encompasses both an assessment of the time value and the

efficiency of the plan. As the number of nodes increase, the ability to trace the logic of complicated, nested expansions deteriorates. The machine can do it but from a user perspective it is tedious. This notwithstanding, the purpose of *Prodigy* was to provide an experimentation testbed for exploring planning and machine learning. This it does admirably. It is an excellent graduate level training tool.

5.0 CONCLUSION

In reflection, *Prodigy* is an excellent learning tool for the advancement and understanding of serial planners and explanation based learning. It is robust and reasonably well documented at the higher abstraction levels. Steve Minton and the other designers quite obviously spent a large amount of their effort in designing a planner which not only works, but has features for discovery and correction of errors - deliberate or unknowns. The EBL system is most useful as an adjunct to domain development. It is less robust than the planner and much more complicated. It "feels" much more experimental than the planner. Perhaps this is because much less work has been done on machine learning than the development of serial planners.

For those that have not tried to code a large scale planner of their own, *Prodigy* provides an interesting study. The attempt at domain independence and the file structure are particularly well thought through. Without the appropriate LISP development tools, however, reverse engineering the program is extremely time consumming and often confusing.

On the down side, this planner appears to be restricted to toy problems of a particular type. First, there are some domains for which *Prodigy* would be the wrong tool. The adversarial planning problems fit in this category as do those where facts are probabilistic rather than TRUE/FALSE. Second, the current planner works well on simple worlds having very limited numbers of operators, goals, and bindings. Yet even here, when the operators and rules are simply doubled to handle 2-robots, the increase in solution time was dramatic. In any real domain world, the number of agents and alternative search

rules possible would number in the hundreds or more. This problem is not, strictly speaking, a *Prodigy* problem but rather is endemic to this whole class of planners. However, before a user gets enamored of *Prodigy*, the distinction between simple STRIPS problems and real domain worlds should be carefully considered.

The EBL module is fascinating to watch and intriguing in application. By iteratively playing the EBL module learning rules against modifications made in the 2-Robot world, it was possible to use the module to deduce better search-rules which could then be installed in the planner. The module did not, as predicted, learn anything new but was quite effective in capturing rules within the stricture of the defined domain. Even with the expanded operator base it was timely in its application. It output explanations of why it "learned" what it did were understandable and descriptive of the concept trying to be captured. Overall, an excellent tool.

5.1 Other Areas of Investigation

This research report was performed to gain a first insight into a newly available planner and learning system. The full breath of the testbed's usefulness has not been explored. There remain several areas which would be immensely entertaining and educational to explore:

X-Windows needs to be implemented so that the graphics capability can be investigated and used in investigation of domain interactions.

A step-by-step cookbook for defining new domains is needed. This would enable the quick assimilation of new researchers interested in machine reasoning.

Another useful project would be to incorporate Michalski's MULTI_STRATEGY learning system as an adjunct of the planner.

EBL only restates knowledge already available to the planner. It would be interesting to study the planner's response to a inductive generalization.

The inability of the planner to handle simultaneous actions by two or more agents and to perform in the face of adversarial goals is another limitation on the domains of interest. I suspect, however, that addressing this problem would necessitate a complete strategy change to the program structure and require recoding the whole system. It should not be begun lightly.

The EBL module needs to be recoded to remove the limitations noted earlier. The current exceptions are needlessly time consuming.

The 2-Robot domain needs a lot more work if it is to be included in the training sample for *Prodigy*. For those that like domain applications this might be a place to start.

The list of additional things to be learned is endless and that, afterall, is why *Prodigy* is an excellent educational testbed.

6.0 Appendix

6.1

2-Robot SCR-Rules

```
;add robot2-----------------------------------------------------------------
        (DONT-PUTDOWN-RMG-II
         (lhs (and (current-node <node>)
                   (current-goal <node> (at rmg2 <loc>))
                   (candidate-op <node> put-down-II)))
         (rhs (reject op put-down-II)))

; add  Don't event consider putting down a block until you
; have to
        (RULE-1-9
    (LHS
      (AND (CURRENT-NODE <@!R9>)
           (CURRENT-GOAL <@!R9> (AT <@!R6> <@!R5>))
           (CANDIDATE-OP <@!R9> PUT-DOWN)
           (KNOWN <@!R9>
                  (~ (OBJECT <@!R6> <@!R4>)))))
    (RHS (REJECT OPERATOR PUT-DOWN)));close rule



      )); end quote,end op-reject


(setq *SCR-BINDINGS-REJECT-RULES*
  '(
; This rule prevents the use of a block that is already either in position
; to support something or it is on the goal stack for supporting something
; else.  Note that it checks for the case where there is a block under
; the location, but it is covering the sqloc.
 (SUPPORTING-OBJ
    (lhs (and (current-node <node>)
              (current-op <node> INFER-OCCUPIED-SQLOC)
              (candidate-bindings <node> (<sqloc> <ob1>))
              (known <node>
                     (and (on-goal-stack <node> (supported-loc <loc>))
                          (or (at <ob1> <ob-loc>)
                              (on-goal-stack <node> (at <ob1> <ob-loc>)))
                          (~ (in-location <sqloc> <ob-loc>))
                                                ;  not already there
                          (under-loc <loc> <ob-loc>)))))
    (rhs (reject bindings (<sqloc> <ob1>))))
     ))

(setq *SCR-NODE-PREFERENCE-RULES* nil)
(setq *SCR-GOAL-PREFERENCE-RULES* nil)
(setq *SCR-OP-PREFERENCE-RULES* '(
        (Rule-1-11
    (LHS
      (AND (CURRENT-NODE <@!R23>)
           (CURRENT-GOAL <@!R23> (AT RMG2 <&@!R19>))
           (CANDIDATE-OP <@!R23> MOVE-II)
           (KNOWN <@!R23>
                  (IS-TYPE <&@!R19> LOCATION))
           (KNOWN <@!R23>
                  (T-ADJACENT-LOCS <&@!R19> <&@!R18>))
           (KNOWN <@!R23> (AT RMG2 <&@!R18>))
           (FORALL (<@!R17>)
                   (KNOWN <@!R23>
                          (SUPPORTING-SQLOCS <&@!R19> <@!R17>))
                   (KNOWN <@!R23>
                          (OCCUPIED-SQLOC <@!R17>)))
           (FORALL (<@!R16>)
                   (KNOWN <@!R23>
                          (AT <@!R15> <@!R16>)
                   (KNOWN <@!R23>
```

```
                          (DISJOINT <&@!R19> <@!R16>)))
            (CANDIDATE-OP <@!R23> <@!OTHER-OP>)
            (NOT-EQUAL MOVE-II <@!OTHER-OP>)))
    (RHS (PREFER OPERATOR MOVE-II <@!OTHER-OP>))
     )));end titel,quote, setq prefer-ops
(setq *SCR-BINDINGS-PREFERENCE-RULES*
      '((TOWARDS
        (priority 0)
          (lhs (and (current-node <node>)
                    (current-op <node> MOVE)
                    (candidate-bindings <node> (<adj-sq2> <to-sq2>))
                    (candidate-bindings <node> (<adj-sq1> <to-sq1>))
                    (known <node>
                           (and (at RMG <from-loc>)
                                (distance <from-loc> <adj-sq1> <dis1>)
                                (distance <from-loc> <adj-sq2> <dis2>)
                                (less-than <dis1> <dis2>)))))
          (rhs (prefer bindings (<adj-sq1> <to-sq1>)
                                (<adj-sq2> <to-sq2>))))


;add robot2-----------------------------------------------------------------
        (TOWARDS-II
        (priority 0)
          (lhs (and (current-node <node>)
                    (current-op <node> MOVE-II)
                    (candidate-bindings <node> (<adj-sq2> <to-sq2>))
                    (candidate-bindings <node> (<adj-sq1> <to-sq1>))
                    (known <node>
                           (and (at RMG2 <from-loc>)
                                (distance <from-loc> <adj-sq1> <dis1>)
                                (distance <from-loc> <adj-sq2> <dis2>)
                                (less-than <dis1> <dis2>)))))
          (rhs (prefer bindings (<adj-sq1> <to-sq1>)
                                (<adj-sq2> <to-sq2>))))


        (CLOSEST-SIDE-FOR-PICK-UP
          (priority 0)
          (lhs (and (current-node <node>)
                    (current-op <node> PICK-UP)
                    (candidate-bindings <node> (<b1> <b2> <bloc>))
                    (candidate-bindings <node> (<a1> <a2> <aloc>))
                    (known <node>
                           (and (at RMG <rmg-loc>)
                                (distance <rmg-loc> <aloc> <dis1>)
                                (distance <rmg-loc> <bloc> <dis2>)
                                (less-than <dis1> <dis2>)))))
          (rhs (prefer bindings (<a1> <a2> <aloc>)
                                (<b1> <b2> <bloc>))))

;add in robot 2-------------------------------------------------
        (CLOSEST-SIDE-FOR-PICK-UP-II
          (priority 0)
          (lhs (and (current-node <node>)
                    (current-op <node> PICK-UP-II)
                    (candidate-bindings <node> (<b1> <b2> <bloc>))
                    (candidate-bindings <node> (<a1> <a2> <aloc>))
                    (known <node>
                           (and (at RMG2 <rmg2-loc>)
                                (distance <rmg2-loc> <aloc> <dis1>)
                                (distance <rmg2-loc> <bloc> <dis2>)
                                (less-than <dis1> <dis2>)))))
          (rhs (prefer bindings (<a1> <a2> <aloc>)
                                (<b1> <b2> <bloc>))))
```

```
#|
**************************************************************************
PRODIGY Version 2.01
Copyright 1989 by Steven Minton, Craig Knoblock, Dan Kuokka and Jaime Carbonell

The PRODIGY System was designed and built by Steven Minton, Craig Knoblock,
Dan Kuokka and Jaime Carbonell.  Additional contributors include Henrik Nordin,
Yolanda Gil, Manuela Veloso, Robert Joseph, Santiago Rementeria, Alicia Perez,
Ellen Riloff, Michael Miller, and Dan Kahn.

The PRODIGY system is experimental software for research purposes only.
This software is made available under the following conditions:
1) PRODIGY will only be used for internal, noncommercial research purposes.
2) The code will not be distributed to other sites without the explicit
   permission of the designers.  PRODIGY is available by request.
3) Any bugs, bug fixes, or extensions will be forwarded to the designers.

Send comments or requests to: prodigy@cs.cmu.edu or The PRODIGY PROJECT,
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.
**************************************************************************|#


;                       Search Control Rule Evolution
;
;The rules listed below are an adaptation of the Prodigy Gridworld Domain.
; The rules have been modified to introduce a second robot to the domain
; and to meet the criteria necessary to use the EBL module.  For this
; particular domain the modifications include the addition of search
; control rules derived by the EBL module after successfully completing a
; problem.  These are identified in their name by the form "R-x-xx".  The
; were also modified to remove Binding Selection rules and ~Exists predicates.
; One weakness of these rules is that the system treats the robots as resources
; rather than independent actors.
;

(setq *SCR-NODE-SELECT-RULES* nil)

(setq *SCR-GOAL-SELECT-RULES*
      '((SELECT-FIRST-GOAL
          (lhs (and (current-node <node>)
                    (not-top-level-node <node>)  ; optional
                    (primary-candidate-goal <node> <goal>)))
          (rhs (select goal <goal>)))
       ))

(setq *SCR-OP-SELECT-RULES* '(
  (R-1-10
    (LHS
      (AND (CURRENT-NODE <@!R14>)
           (CURRENT-GOAL <@!R14> (AT RMG2 <@!R11>))
           (CANDIDATE-OP <@!R14> MOVE-II)
           (KNOWN <@!R14>
                  (~ (OBJECT RMG2 <@!R10>)))))
    (RHS (SELECT OPERATOR MOVE-II))

  )));  end title, quote,setq op-select
(setq *SCR-BINDINGS-SELECT-RULES*
        '((PUT-DOWN-ANYWHERE
            (lhs (and (current-node <node>)
                      (current-goal <node> (~ (holding <ob>)))
                      (current-op <node> PUT-DOWN)
                      (known <node>
                             (and (at RMG <RMG-loc>)
                                  (next-to <RMG-loc> <new-loc>)))))
            (rhs (select bindings
                         (<ob> <new-loc> <RMG-loc>))))
```

```
          (MOVE-DELETE-RMG
              (lhs (and (current-node <node>)
                        (current-goal <node> (~ (at RMG <adj-loc>)))
                        (current-op <node> MOVE)
                        (known <node>
                               (t-adjacent-locs <adj-loc> <to-loc>))))
              (rhs (select bindings (<adj-loc> <to-loc>))))

; add in robot2-------------------------------------------------------------
          (MOVE-DELETE-RMG2
              (lhs (and (current-node <node>)
                        (current-goal <node> (~ (at RMG2 <adj-loc>)))
                        (current-op <node> MOVE)
                        (known <node>
                               (t-adjacent-locs <adj-loc> <to-loc>))))
              (rhs (select bindings (<adj-loc> <to-loc>))))

;add robot2----------------------------------------------------------------
(PUT-DOWN-ANYWHERE-II
            (lhs (and (current-node <node>)
                      (current-goal <node> (~ (rmg2-holding <ob>)))
                      (current-op <node> PUT-DOWN-II)
                      (known <node>
                             (and (at RMG2 <RMG2-loc>)
                                  (next-to <RMG2-loc> <new-loc>)))))
            (rhs (select bindings
                         (<ob> <new-loc> <RMG2-loc>))))


))

(setq *SCR-NODE-REJECT-RULES*
      '((BLOCK-SUPPORT-RULE-1
           (lhs (and (primary-candidate-node <node>)
                     (candidate-goal <node> (at <obj> <loc1>))
                     (on-goal-stack <node> (at <obj> <loc2>))
                       ;  DONT really need KNOWN test
                     (known <node> (under-loc <loc1> <loc2>))))
           (rhs (reject node <node>)))

; This rule is no longer needed because the domain rules for clear was
; such that it would never attempt to clear my picking up the block.  This
; was done by changing the universal quantifier to an existential quantifier
; and as such it no longer attemps to make the set empty by removing the object.
;        (DONT-CLEAR-BY-PICKING-UP
;          (lhs (and (primary-candidate-node <node>)
;                    (candidate-goal <node> (~ (at <obj> <loc>)))
;                    (direct-supergoal-of <node> (~ (at <obj> <loc>))
;                        (clear <obj>))))
;          (rhs (reject node <node>)))
        ))


(setq *SCR-GOAL-REJECT-RULES* nil)

(setq *SCR-OP-REJECT-RULES*
      '(
; Dont ever try to putdown rmg.  Its not possible to have the robot put itself
; down.
         (DONT-PUTDOWN-RMG
          (lhs (and (current-node <node>)
                    (current-goal <node> (at rmg <loc>))
                    (candidate-op <node> put-down)))
          (rhs (reject op put-down)))
```

```
(CLOSEST-SIDE-FOR-PUT-DOWN
    (priority 0)
    (lhs (and (current-node <node>)
              (current-op <node> PUT-DOWN)
              (candidate-bindings <node> (<a> <b> <loc1>))
              (candidate-bindings <node> (<e> <f> <loc2>))
              (known <node>
                     (and (at RMG <rmg-loc>)
                          (distance <rmg-loc> <loc1> <dis1>)
                          (distance <rmg-loc> <loc2> <dis2>)
                          (less-than <dis2> <dis1>)))))
    (rhs (prefer bindings (<e> <f> <loc2>)
                          (<a> <b> <loc1>))))

;add robot2-------------------------------------------------------------------
(CLOSEST-SIDE-FOR-PUT-DOWN-II
    (priority 0)
    (lhs (and (current-node <node>)
              (current-op <node> PUT-DOWN-II)
              (candidate-bindings <node> (<a> <b> <loc1>))
              (candidate-bindings <node> (<e> <f> <loc2>))
              (known <node>
                     (and (at RMG2 <rmg2-loc>)
                          (distance <rmg2-loc> <loc1> <dis1>)
                          (distance <rmg2-loc> <loc2> <dis2>)
                          (less-than <dis2> <dis1>)))))
    (rhs (prefer bindings (<e> <f> <loc2>)
                          (<a> <b> <loc1>))))



;(PREFER-OBJ-ALREADY-THERE
;    (lhs (and (current-node <node>)
;              (current-op <node> INFER-OCCUPIED-SQLOC)
;              (current-goal <node> (occupied-sqloc <sqloc>))
;              (candidate-bindings <node> (<sqloc> <obj1> <b> <c>))
;              (candidate-bindings <node> (<sqloc> <obj2> <e> <f>))
;              (known <node> (at <obj2> <sqloc>))))
;    (rhs (prefer bindings (<sqloc> <obj2> <e> <f>)
;                          (<sqloc> <obj1> <b> <c>))))


(CLOSEST-SIDE-FOR-OCCUPIED
    (priority 0)
    (lhs (and (current-node <node>)
              (current-op <node> INFER-OCCUPIED-SQLOC)
              (candidate-bindings <node> (<sqloc> <obj1>))
              (candidate-bindings <node> (<sqloc> <obj2>))
              (not-equal <obj1> <obj2>)
              (known <node>
                     (and (at RMG <rmg-loc>)
                          (at <obj1> <loc1>)
                          (~ (in-location <sqloc> <loc1>))  ; not there already
                          (at <obj2> <loc2>)
                          (distance <rmg-loc> <loc1> <dis1>)
                          (distance <rmg-loc> <loc2> <dis2>)
                          (less-than <dis2> <dis1>)))))
    (rhs (prefer bindings (<sqloc> <obj2>)
                          (<sqloc> <obj1>))))

;add robot2-------------------------------------------------------------------
(CLOSEST-SIDE-FOR-OCCUPIED-II
    (priority 0)
```

```
    (lhs (and (current-node <node>)
              (current-op <node> INFER-OCCUPIED-SQLOC)
              (candidate-bindings <node> (<sqloc> <obj1>))
              (candidate-bindings <node> (<sqloc> <obj2>))
              (not-equal <obj1> <obj2>)
              (known <node>
                     (and (at RMG2 <rmg2-loc>)
                          (at <obj1> <loc1>)
                          (~ (in-location <sqloc> <loc1>))  ; not there already
                          (at <obj2> <loc2>)
                          (distance <rmg2-loc> <loc1> <dis1>)
                          (distance <rmg2-loc> <loc2> <dis2>)
                          (less-than <dis2> <dis1>)))))
    (rhs (prefer bindings (<sqloc> <obj2>)
                          (<sqloc> <obj1>))))



; (ALREADY-THERE
;    (lhs (and (current-node <node>)
;              (current-op <node> INFER-OCCUPIED-SQLOC)
;              (candidate-bindings <node> (<sqloc> <ob1>))
;             (candidate-bindings <node> (<sqloc> <ob2>))
;             (known <node>
;                    (and (at <ob2> <ob-loc>)
;                         (in-location <sqloc> <ob-loc>)))))
;    (rhs (prefer bindings (<sqloc> <ob2>)
;                          (<sqloc> <ob1>))))


  (TOWARD-SUPERGOAL
      (priority 0)
      (lhs (and (current-node <node>)
                (current-op <node> MOVE)
                (current-goal <node> (~ (at RMG <rmg-loc>)))
                (candidate-bindings <node> (<rmg-loc> <to-sq2>))
                (candidate-bindings <node> (<rmg-loc> <to-sq1>))
                (not-equal <to-sq1> <to-sq2>)
                (known <node>
                       (and (on-goal-stack <node> (at <obj> <rmg-loc>))
                            (at <obj> <loc>)
                            (distance <loc> <to-sq1> <dis1>)
                            (distance <loc> <to-sq2> <dis2>)
                            (less-than <dis1> <dis2>)
                            (~ (under-loc <to-sq1> <loc>))
                            (forall (<ob-loc>) (at <ob> <ob-loc>)
                                    (~ (in <to-sq1> <ob-loc>)))))))

;                           (~ (is-occupied-sqloc <node> <to-sq1>))))))
      (rhs (prefer bindings (<rmg-loc> <to-sq1>)
                            (<rmg-loc> <to-sq2>))))

;add robot 2
  (TOWARD-SUPERGOAL-II
      (priority 0)
      (lhs (and (current-node <node>)
                (current-op <node> MOVE)
                (current-goal <node> (~ (at RMG2 <rmg2-loc>)))
                (candidate-bindings <node> (<rmg2-loc> <to-sq2>))
                (candidate-bindings <node> (<rmg2-loc> <to-sq1>))
                (not-equal <to-sq1> <to-sq2>)
                (known <node>
                       (and (on-goal-stack <node> (at <obj> <rmg2-loc>))
                            (at <obj> <loc>)
                            (distance <loc> <to-sq1> <dis1>)
```

```
                              (distance <loc> <to-sq2> <dis2>)
                              (less-than <dis1> <dis2>)
                              (~ (under-loc <to-sq1> <loc>))
                              (forall (<ob-loc>) (at <ob> <ob-loc>)
                                      (~ (in <to-sq1> <ob-loc>)))))))

;                             (~ (is-occupied-sqloc <node> <to-sq1>))))))
      (rhs (prefer bindings (<rmg2-loc> <to-sq1>)
                            (<rmg2-loc> <to-sq2>))))



   (NEXT-TO-OBJ-BEFORE-VACATING

       (lhs (and (current-node <node>)
                 (current-op <node> MOVE)
                 (current-goal <node> (~ (at RMG <rmg-loc>)))
                 (candidate-bindings <node> (<rmg-loc> <to-sq2>))
                 (candidate-bindings <node> (<rmg-loc> <to-sq1>))
                 (not-equal <to-sq1> <to-sq2>)
                 (known <node>
                        (and (on-goal-stack <node> (at <obj> <rmg-loc>))
                             (at <obj> <loc>)
                             (in-location <to-sq1> <loc>)))))
       (rhs (prefer bindings (<rmg-loc> <to-sq1>)
                             (<rmg-loc> <to-sq2>))))

;add robot2-----------------------------------------------------------------
   (NEXT-TO-OBJ-BEFORE-VACATING-II

       (lhs (and (current-node <node>)
                 (current-op <node> MOVE-II)
                 (current-goal <node> (~ (at RMG2 <rmg2-loc>)))
                 (candidate-bindings <node> (<rmg2-loc> <to-sq2>))
                 (candidate-bindings <node> (<rmg2-loc> <to-sq1>))
                 (not-equal <to-sq1> <to-sq2>)
                 (known <node>
                        (and (on-goal-stack <node> (at <obj> <rmg2-loc>))
                             (at <obj> <loc>)
                             (in-location <to-sq1> <loc>)))))
       (rhs (prefer bindings (<rmg2-loc> <to-sq1>)
                             (<rmg2-loc> <to-sq2>))))



   (UNOCCUPIED-SQLOC-FOR-PICK-UP
      (lhs (and (current-node <node>)
                (current-op <node> PICK-UP)
                (candidate-bindings <node> (<ob2> <ob-loc2> <rmg-loc2>))
                (candidate-bindings <node> (<ob1> <ob-loc1> <rmg-loc1>))
                (not-equal <rmg-loc1> <rmg-loc2>)
                (known <node>
                       (and (is-occupied-sqloc <node> <rmg-loc2>)
                            (~ (at RMG <rmg-loc2>))))))
       (rhs (prefer bindings (<ob1> <ob-loc1> <rmg-loc1>)
                             (<ob2> <ob-loc2> <rmg-loc2>))))

;add robot2-----------------------------------------------------------------
   (UNOCCUPIED-SQLOC-FOR-PICK-UP-II
       (lhs (and (current-node <node>)
                 (current-op <node> PICK-UP-II)
                 (candidate-bindings <node> (<ob2> <ob-loc2> <rmg2-loc2>))
                 (candidate-bindings <node> (<ob1> <ob-loc1> <rmg2-loc1>))
                 (not-equal <rmg2-loc1> <rmg2-loc2>)
                 (known <node>
```

```
                    (and (is-occupied-sqloc <node> <rmg2-loc2>)
                         (~ (at RMG2 <rmg2-loc2>))))))
    (rhs (prefer bindings (<ob1> <ob-loc1> <rmg2-loc1>)
                          (<ob2> <ob-loc2> <rmg2-loc2>))))


)
```

6.2

2-Robot Operators

```
#|
**************************************************************************
PRODIGY Version 2.01
Copyright 1989 by Steven Minton, Craig Knoblock, Dan Kuokka and Jaime Carbonell

The PRODIGY System was designed and built by Steven Minton, Craig Knoblock,
Dan Kuokka and Jaime Carbonell.  Additional contributors include Henrik Nordin,
Yolanda Gil, Manuela Veloso, Robert Joseph, Santiago Rementeria, Alicia Perez,
Ellen Riloff, Michael Miller, and Dan Kahn.

The PRODIGY system is experimental software for research purposes only.
This software is made available under the following conditions:
1) PRODIGY will only be used for internal, noncommercial research purposes.
2) The code will not be distributed to other sites without the explicit
   permission of the designers.  PRODIGY is available by request.
3) Any bugs, bug fixes, or extensions will be forwarded to the designers.

Send comments or requests to: prodigy@cs.cmu.edu or The PRODIGY PROJECT,
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.
**************************************************************************|#

;                          Operator Evolution
;
;
; The operators listed below are a modification of the Prodigy Gridworld to
; incorporate a second robot in the domain.  One lack of elegance in the
; approach taken is that it was necessary to duplicate all of robot1's
; operators for robot2.  This was done for two reasons.  First, the Prodigy
; EBL module does not allow the use of Conditionals in the effects list of
; the operators.  Since a major requirement existed to be able to exercise the
; learning module, the operators had to be stand-alone.  Second, by having
; separate operators, it is possible to easily change the capabilities of
; one robot and not the other.  A second modification was necessary to the
; inference rule ARMS-EMPTY.  ~EXISTS is not a permissable construct to the EBL
; module, so the inference rule was changed to include FORALL ~X instead.
; Future changes needed include an expansion of the inference rules section
; to include descriptions on which robot should be picked for an action.  For
; instance if robot1 is holding and candidate-op is Move, infer robot1 BUSY.
; Such an inference can be used as a toggle in the search control rules for
; preferring one set of bindings over another.
;
;  The changes made are marked below.
;
;
; added in rmg2-reach ------------------------------------
(setq *WORLD-X-SIZE* 5 *WORLD-Y-SIZE* 5 *WORLD-Z-SIZE* 5)
(setq *RMG-reach* 2 *RMG2-reach* 2)


; Operators

(setq *OPERATORS* '(


(PICK-UP
 (params (<ob> <ob-loc> <RMG-loc>))
; (vars (<ob> <ob-loc> <size> <RMG-loc>))
 (preconds
       (exists (<ob> <size>) (object <ob> <size>)
        (exists (<ob-loc>) (at <ob> <ob-loc>)
         (exists (<RMG-loc>) (next-to <ob-loc> <RMG-loc>)
            (and (at RMG <RMG-loc>)
                 (within-height <ob-loc> <RMG-loc> *RMG-reach*)
                 (clear <ob>)
                 (arm-empty)
    )))))
```

```
 (effects (
   (del (at <ob> <ob-loc>))
   (add (holding <ob>)))))

;added in rmg2 for pickup-----------------------------------------------
(PICK-UP-II
 (params (<ob> <ob-loc> <RMG2-loc>))
; (vars (<ob> <ob-loc> <size> <RMG-loc>))
 (preconds
       (exists (<ob> <size>) (object <ob> <size>)
        (exists (<ob-loc>) (at <ob> <ob-loc>)
         (exists (<RMG2-loc>) (next-to <ob-loc> <RMG2-loc>)
            (and (at RMG2 <RMG2-loc>)
                 (within-height <ob-loc> <RMG2-loc> *RMG2-reach*)
                 (clear <ob>)
                 (rmg2-arm-empty)
   )))))
 (effects (
   (del (at <ob> <ob-loc>))
   (add (rmg2-holding <ob>)))))


;  Temporarily hacked to use next-to as a generator
;  note: plenty of ordering problems

(PUT-DOWN
 (params (<ob> <new-loc> <RMG-loc>))
;  (vars (<ob> <new-loc> <RMG-loc> <loc-size> <ob-size>))
 (preconds
    (exists (<new-loc>) (vacant-loc <new-loc>)
     (exists (<ob> <ob-size>) (object <ob> <ob-size>)
      (exists (<RMG-loc>) (next-to <new-loc> <RMG-loc>)
       (and
            (exists (<loc-size>) (location-size <new-loc> <loc-size>)
            (equal-p <loc-size> <ob-size>))
            (supported-loc <new-loc>)
            (supported-loc <RMG-loc>)
            (holding <ob>)
            (at RMG <RMG-loc>)
            (within-height <new-loc> <RMG-loc> *RMG-reach*))))))
 (effects (
   (del (holding <ob>))
   (add (at <ob> <new-loc>)))))



(MOVE
 (params (<adj-loc> <to-loc>))
;  (vars (<adj-loc> <to-loc>))
 (preconds
       (exists (<to-loc>) (is-type <to-loc> location)
         (and (supported-  c <to-loc>)
              (vacant-lo  to-loc>)
              (exists (<  -loc>) (t-adjacent-locs <to-loc> <adj-loc>)
                 (at RMG  dj-loc>)))))
 (effects (
   (del (at RMG <adj-loc>))
   (add (at RMG <to-loc>)))))

;add robot2-------------------------------------------------------------
(PUT-DOWN-II
 (params (<ob> <new-loc> <RMG2-loc>))
;  (vars (<ob> <new-loc> <RMG2-loc> <loc-size> <ob-size>))
 (preconds
    (exists (<new-lo>) (vacant-loc <new-loc>)
     (exists (<ob> <ob-size>) (object <ob> <ob-size>)
```

```
      (exists (<RMG2-loc>) (next-to <new-loc> <RMG2-loc>)
       (and
            (exists (<loc-size>) (location-size <new-loc> <loc-size>)
            (equal-p <loc-size> <ob-size>))
            (supported-loc <new-loc>)
            (supported-loc <RMG2-loc>)
            (holding <ob>)
            (at RMG2 <RMG2-loc>)
            (within-height <new-loc> <RMG2-loc> *RMG2-reach*))))))
 (effects (
   (del (rmg2-holding <ob>))
   (add (at <ob> <new-loc>)))))

;add robot2-----------------------------------------------------------------
(MOVE-II
 (params (<adj-loc> <to-loc>))
;  (vars (<adj-loc> <to-loc>))
 (preconds
        (exists (<to-loc>) (is-type <to-loc> location)
          (and (supported-loc <to-loc>)
               (vacant-loc <to-loc>)
               (exists (<adj-loc>) (t-adjacent-locs <to-loc> <adj-loc>)
                  (at RMG2 <adj-loc>)))))
 (effects (
   (del (at RMG2 <adj-loc>))
   (add (at RMG2 <to-loc>)))))



))

(setq *INFERENCE-RULES* '(

(INFER-VACANT-LOC
 (params (<loc>))
;  (vars (<loc> <obj> <obj-loc>))
 (preconds
     (forall (<loc>) (is-type <loc> location)
      (forall (<obj> <obj-loc>) (at <obj> <obj-loc>)
        (disjoint <loc> <obj-loc>))))
 (effects (
   (add (vacant-loc <loc>)))))

;Dont need this rule because the INFER-SUPPORTED-LOC handles this
;just fine.

;(INFER-SUPPORTED-LOC-BASE
; (params (<loc>))
; (vars (<loc>))
; (preconds
;  (forall (<loc>)(is-type <loc> location)
;   (ground-loc <loc>)))
; (effects (
;   (add (supported-loc <loc>)))))


(INFER-SUPPORTED-LOC
 (params (<loc>))
;  (vars (<loc> <sqloc>))
 (preconds
        (forall (<sqloc>) (supporting-sqlocs <loc> <sqloc>)
            (occupied-sqloc <sqloc>)))
 (effects (
   (add (supported-loc <loc>)))))
```

```
; This rule is hacked to deal with the 'at-in' problem (might be solved by
; the ability to set up a failed exists generator as a goal)
;
(INFER-OCCUPIED-SQLOC
 (params (<sqloc> <obj>))
;  (vars (<sqloc> <obj> <obj-size> <loc>))
 (preconds
         (exists (<obj> <obj-size>) (object <obj> <obj-size>)
             (exists (<loc>) (covers <loc> <obj-size> <sqloc>)
                 (at <obj> <loc>))))
 (effects (
   (add (occupied-sqloc <sqloc>)))))


(INFER-CLEAR
 (params (<ob>))
;  (vars (<ob> <loc> <above-loc>))
 (preconds
    (exists (<loc>) (at <ob> <loc>)
     (exists (<above-loc>) (above-loc <above-loc> <loc>)
      (vacant-loc <above-loc>))))
 (effects (
   (add (clear <ob>)))))


(INFER-ARM-EMPTY
 (params nil)
;  (vars (<ob>))
 (preconds
     (FORALL (<obj>) (OBJECT <obj>)(~(Holding <obj>)))        )
 (effects (
   (add (arm-empty)))))


;add robot2-----------------------------------------------------------------
(INFER-ARM-EMPTY-II
 (params nil)
;  (vars (<ob>))
 (preconds
     (FORALL (<obj>) (OBJECT <obj>)(~(rmg2-Holding <obj>)))        )
 (effects (
   (add (rmg2-arm-empty)))))
))
```

```
;******* VERSION 1-2 ******* Locked by nobody *******
;
; This problem has the RMG2 take one step from 1,2 to 1,1.
; And it has two robots

(load-goal '(at RMG2 (location 1 1 0 1 1 0)))

(load-start-state
    '((at RMG (location 1 2 0 1 2 0))
      (at RMG2 (location 3 3 3 3 3 3)))
);stored as r2g-one-step-var.lisp
```

6.4

2-Robot EBL Output

The attached run was made using the single step problem of robot2 in the two-robot domain of gridworld. The learning module was activated. The listing shows first the solution of the problem and then the actions taken by the learning module to deduce any valid and "apparently" productive search rules. In subsequent runs, the rules which survived were hardcoded into the SC-rules to make them a permanent feature of the the two-robot domain. Thus the OBSERVER learned, implemented, and then learned again on the modified database.

```
PRODIGY 2.01   (The Prodigy Project, May 15, 1989)
Mail questions and remarks to prodigy@CS.CMU.EDU


Welcome to   P R O D I G Y ...


Current Domain:   gridworld.
Current Problem:  r2g-1step.
Active Switches:  EBL  EBL-TEXT  TEXT.

Prodigy: run

OBSERVER reset

Number of rules learned: 0
Number of learned rules still in: 0


* * * * * P R O D I G Y 2.01 * * * * *

Goal State: (AT RMG2 (LOCATION 1 1 0 1 1 0))
Start State: (AT RMG (LOCATION 1 2 0 1 2 0))
             (AT RMG2 (LOCATION 1 0 0 1 0 0))

N1 (DONE)
Alts: *FINISH*

   N2 (AT RMG2 (LOCATION 1 1 0 1 1 0))
   Alts: PUT-DOWN RMG2 (LOCATION 1 1 0 1 1 0) <RMG-LOC-PUT-DOWN> [2]

      N3 (VACANT-LOC (LOCATION 1 1 0 1 1 0))
      Alts: INFER-VACANT-LOC (LOCATION 1 1 0 1 1 0)

   ...Done: INFER-VACANT-LOC (LOCATION 1 1 0 1 1 0)

   N4 (AT RMG2 (LOCATION 1 1 0 1 1 0))
   Alts: PUT-DOWN RMG2 (LOCATION 1 1 0 1 1 0) <RMG-LOC-PUT-DOWN>

      N5 (OBJECT RMG2 <v637>)
      There are no relevant alts for this node!
-------------------------------------------------

   N2 (AT RMG2 (LOCATION 1 1 0 1 1 0))
   Alts: [1] MOVE-II <ADJ-LOC-MOVE-II> (LOCATION 1 1 0 1 1 0)

      N6 (SUPPORTED-LOC (LOCATION 1 1 0 1 1 0))
      Alts: INFER-SUPPORTED-LOC (LOCATION 1 1 0 1 1 0)

   ...Done: INFER-SUPPORTED-LOC (LOCATION 1 1 0 1 1 0)

   N7 (AT RMG2 (LOCATION 1 1 0 1 1 0))
   Alts: MOVE-II <ADJ-LOC-MOVE-II> (LOCATION 1 1 0 1 1 0)

      N8 (VACANT-LOC (LOCATION 1 1 0 1 1 0))
      Alts: INFER-VACANT-LOC (LOCATION 1 1 0 1 1 0)
```

```
   ...Done: INFER-VACANT-LOC (LOCATION 1 1 0 1 1 0)

   N9 (AT RMG2 (LOCATION 1 1 0 1 1 0))
   Alts: MOVE-II (LOCATION 1 0 0 1 0 0) (LOCATION 1 1 0 1 1 0) [2-8]

...Done: MOVE-II (LOCATION 1 0 0 1 0 0) (LOCATION 1 1 0 1 1 0)

N10 (DONE)
Alts: *FINISH*

...Done: *FINISH*

Completed Success

CPU time: 13.8 seconds
Number of Nodes: 11
Solution Length: 3
Operator Sequence:

      MOVE-II (LOCATION 1 0 0 1 0 0) (LOCATION 1 1 0 1 1 0)

REMOVED NIL
Type:            Name:                          Cum Match Time:      Cum Savings:
-----            -----                          ---------------      ------------
GOAL-SELECT      SELECT-FIRST-GOAL  0.0         0.16   + 0.08        0.0  + 0.0
BINDINGS-SE      PUT-DOWN-ANYWHERE  0.0         0.72   + 0.36        0.0  + 0.0
BINDINGS-SE      MOVE-DELETE-RMG  0.0           0.92   + 0.46        0.0  + 0.0
BINDINGS-SE      MOVE-DELETE-RMG2  0.0           0.8   +  0.4        0.0  + 0.0
BINDINGS-SE      PUT-DOWN-ANYWHERE-II  0.0       0.8   + 0.44      0.0   + 0.0
NODE-REJECT      BLOCK-SUPPORT-RULE-1  0.0      1.72   + 0.8       0.0   + 0.0
OP-REJECT-R      DONT-PUTDOWN-RMG  0.0          0.72   + 0.36        0.0  + 0.0
OP-REJECT-R      DONT-PUTDOWN-RMG-II  0.0        0.7   + 0.36      0.0  + 0.0
BINDINGS-RE      SUPPORTING-OBJ  0.0            1.2   + 0.64        0.0  + 0.0
BINDINGS-PR      TOWARDS  0.0                  0.12   + 0.06        0.0  + 0.0
BINDINGS-PR      TOWARDS-II  0.0                6.4   +  3.2        0.0  + 0.0
BINDINGS-PR      CLOSEST-SIDE-FOR-PICK-UP  0.0     0.12   + 0.06      0.0   + 0.0
BINDINGS-PR      CLOSEST-SIDE-FOR-PICK-UP-II  0.0   0.12   + 0.06      0.0   + 0.0
BINDINGS-PR      CLOSEST-SIDE-FOR-PUT-DOWN  0.0     0.12   + 0.06      0.0   + 0.0
BINDINGS-PR      CLOSEST-SIDE-FOR-PUT-DOWN-II  0.0   0.12   + 0.06      0.0   + 0.0
BINDINGS-PR      CLOSEST-SIDE-FOR-OCCUPIED  0.0     0.12   + 0.06      0.0   + 0.0
BINDINGS-PR      CLOSEST-SIDE-FOR-OCCUPIED-II  0.0   0.14   + 0.06      0.0   + 0.0
BINDINGS-PR      TOWARD-SUPERGOAL  0.0          0.12   + 0.06        0.0  + 0.0
BINDINGS-PR      TOWARD-SUPERGOAL-II  0.0       0.12   + 0.06      0.0  + 0.0
BINDINGS-PR      NEXT-TO-OBJ-BEFORE-VACATING  0.   0.14  + 0.08     0.0   + 0.
BINDINGS-PR      NEXT-TO-OBJ-BEFORE-VACATING-II  0.0   0.22  +  0.1   0.0   + 0.0
BINDINGS-PR      UNOCCUPIED-SQLOC-FOR-PICK-UP  0.0   0.12  + 0.06      0.0   + 0.0
BINDINGS-PR      UNOCCUPIED-SQLOC-FOR-PICK-UP-II  0.0   0.1  + 0.06     0.0  + 0.0

Observer 2 invoked
--------------------------
EBS Processing Node: N5


TRAINING EXAMPLE: (FAILS N5)      HIST: HST1

INITIAL-RESULT:
(FAILS <@!NODE>) if
(AND (IS-EQUAL <@2> (OBJECT <@3> <@4>))
     (AND (PRIMARY-CANDIDATE-GOAL <@!NODE> (OBJECT <@3> <@4>))
          (AND T (AND T T))))

SIMPLIFIED-RESULT:
(FAILS <@!NODE>) if
(PRIMARY-CANDIDATE-GOAL <@!NODE>
                        (OBJECT <@3> <@4>))
Turning on TP
```

```
New Learned Rule: R-1-1
------------------------
EBS Processing Node: N4


TRAINING EXAMPLE: (FAILS N4)      HIST: HST2

INITIAL-RESULT:
(FAILS <@!NODE>) if
(AND (IS-EQUAL <@3> (AT <@4> <@5>))
     (AND (ALT-ON-DECK <@!NODE> (AT <@4> <@5>) PUT-DOWN)
          (AND T
               (AND T
                    (AND
                      (HAS-BOUND-VARS <@!NODE>
                                      (AT <@4> <@5>)
                                      PUT-DOWN
                                      (NIL NIL NIL NIL NIL))
                      (OR
                        (~ (KNOWN <@!NODE>
                                  (OBJECT <@4>
                                          <OB-SIZE-PUT-DOWN>)))
                        (FORALL (<OB-SIZE-PUT-DOWN>)
                                (KNOWN <@!NODE>
                                       (OBJECT <@4> <OB-SIZE-PUT-DOWN>))
                                (OR NIL)))))))))

SIMPLIFIED-RESULT:
(FAILS <@!NODE>) if
(AND (ALT-ON-DECK <@!NODE>
                  (AT <@4> <@5>)
                  PUT-DOWN)
     (KNOWN <@!NODE>
            (~ (OBJECT <@4> <OB-SIZE-PUT-DOWN>))))
No rule: Dont-make-rule-flag -- IS-RESET
Extending (FAILS N4) to N3
------------------------
EBS Processing Node: N3

------------------------
EBS Processing Node: N11

------------------------
EBS Processing Node: N10

------------------------
EBS Processing Node: N9


TRAINING EXAMPLE: (OP-SUCCEEDS N9 (AT RMG2 (LOCATION 1 1 0 1 1 0))
MOVE-II ((LOCATION 1 0 0 1 0 0) (LOCATION 1 1 0 1 1 0)))      HIST: HST3

INITIAL-RESULT:
(OP-SUCCEEDS <@!NODE> <@!GOAL> <@!OP> <@!B>) if
(AND (MATCHES (AT RMG2 <TO-LOC-MOVE-II>)
              <@!GOAL>)
     (IS-EQUAL <@!B>
               (<ADJ-LOC-MOVE-II> <TO-LOC-MOVE-II>))
     (IS-EQUAL <@!OP> MOVE-II)
     (KNOWN <@!NODE>
            (EXISTS (<TO-LOC-MOVE-II>)
                    (IS-TYPE <TO-LOC-MOVE-II> LOCATION)
                    (AND (SUPPORTED-LOC <TO-LOC-MOVE-II>)
                         (VACANT-LOC <TO-LOC-MOVE-II>)
                         (EXISTS (<ADJ-LOC-MOVE-II>)
                                 (T-ADJACENT-LOCS <TO-LOC-MOVE-II>
```

```
                                                    <ADJ-LOC-MOVE-II>)
                                    (AT RMG2 <ADJ-LOC-MOVE-II>))))))

SIMPLIFIED-RESULT:
(OP-SUCCEEDS <@!NODE> <@!GOAL> <@!OP> <@!B>) if
(AND (IS-EQUAL (AT RMG2 <TO-LOC-MOVE-II>)
               <@!GOAL>)
     (IS-EQUAL <@!B>
               (<ADJ-LOC-MOVE-II> <TO-LOC-MOVE-II>))
     (IS-EQUAL <@!OP> MOVE-II)
     (KNOWN <@!NODE>
            (IS-TYPE <TO-LOC-MOVE-II> LOCATION))
     (KNOWN <@!NODE>
            (SUPPORTED-LOC <TO-LOC-MOVE-II>))
     (KNOWN <@!NODE>
            (VACANT-LOC <TO-LOC-MOVE-II>))
     (KNOWN <@!NODE>
            (T-ADJACENT-LOCS <TO-LOC-MOVE-II> <ADJ-LOC-MOVE-II>))
     (KNOWN <@!NODE> (AT RMG2 <ADJ-LOC-MOVE-II>)))
No rule: Dont-make-rule-flag -- INTERMED-RESULT
-------------------------
EBS Processing Node: N8


TRAINING EXAMPLE: (OP-SUCCEEDS N8 (AT RMG2 (LOCATION 1 1 0 1 1 0))
 MOVE-II ((LOCATION 1 0 0 1 0 0) (LOCATION 1 1 0 1 1 0)))     HIST: HST5

INITIAL-RESULT:
(OP-SUCCEEDS <@!NODE> <@!GOAL> <@!OP> <@!B>) if
(AND
  (KNOWN <@!NODE>
         (FORALL (<LOC-INFER-VACANT-LOC>)
                 (IS-TYPE <LOC-INFER-VACANT-LOC> LOCATION)
                 (FORALL (<OBJ-INFER-VACANT-LOC> <OBJ-LOC>)
                         (AT <OBJ-INFER-VACANT-LOC> <OBJ-LOC>)
                         (DISJOINT <LOC-INFER-VACANT-LOC> <OBJ-LOC>))))
  (AND
    (AND (IS-EQUAL (AT RMG2 <@20>) <@19>)
         (IS-EQUAL <@18> (<@17> <@20>))
         (IS-EQUAL <@16> MOVE-II)
         (KNOWN <@!NODE> (IS-TYPE <@20> LOCATION))
         (KNOWN <@!NODE>
                (SUPPORTED-LOC <@20>))
         (KNOWN <@!NODE>
                (AND (IS-EQUAL VACANT-LOC VACANT-LOC)
                     (IS-EQUAL <@20> <LOC-INFER-VACANT-LOC>)))
         (KNOWN <@!NODE>
                (T-ADJACENT-LOCS <@20> <@17>))
         (KNOWN <@!NODE> (AT RMG2 <@17>)))
    (IS-EQUAL <@15> <@7>)
    (IS-EQUAL <@19> <@!GOAL>)
    (IS-EQUAL <@16> <@!OP>)
    (IS-EQUAL <@18> <@!B>)))

SIMPLIFIED-RESULT:
(OP-SUCCEEDS <@!NODE> <@!GOAL> <@!OP> <@!B>) if
(AND
  (FORALL (<OBJ-LOC>)
          (KNOWN <@!NODE>
                 (AT <OBJ-INFER-VACANT-LOC> <OBJ-LOC>))
          (KNOWN <@!NODE>
                 (DISJOINT <LOC-INFER-VACANT-LOC>
                           <OBJ-LOC>)))
  (KNOWN <@!NODE>
         (IS-TYPE <LOC-INFER-VACANT-LOC> LOCATION))
  (KNOWN <@!NODE>
```

```
          (SUPPORTED-LOC <LOC-INFER-VACANT-LOC>))
   (KNOWN <@!NODE>
          (T-ADJACENT-LOCS <LOC-INFER-VACANT-LOC> <@17>))
   (KNOWN <@!NODE> (AT RMG2 <@17>))
   (IS-EQUAL (AT RMG2 <LOC-INFER-VACANT-LOC>)
             <@!GOAL>)
   (IS-EQUAL MOVE-II <@!OP>)
   (IS-EQUAL (<@17> <LOC-INFER-VACANT-LOC>)
             <@!B>))
No rule: Dont-make-rule-flag -- INTERMED-RESULT
-------------------------
EBS Processing Node: N7


TRAINING EXAMPLE: (OP-SUCCEEDS N7 (AT RMG2 (LOCATION 1 1 0 1 1 0))
MOVE-II ((LOCATION 1 0 0 1 0 0) (LOCATION 1 1 0 1 1 0)))     HIST: HST6

INITIAL-RESULT:
(OP-SUCCEEDS <@!NODE> <@!GOAL> <@!OP> <@!B>) if
(AND
   (AND (IS-EQUAL (AT RMG2 <@!17>) <@!19>)
        (KNOWN <@!NODE>
               (IS-TYPE <@!17> LOCATION))
        (KNOWN <@!NODE>
               (SUPPORTED-LOC <@!17>))
        (IS-EQUAL MOVE-II <@!20>)
        (IS-EQUAL (<@!16> <@!17>) <@!21>)
        (KNOWN <@!NODE>
               (T-ADJACENT-LOCS <@!17> <@!16>))
        (KNOWN <@!NODE> (AT RMG2 <@!16>))
        (FORALL (<@!15>)
                (KNOWN <@!NODE> (AT <@!14> <@!15>))
                (KNOWN <@!NODE>
                       (DISJOINT <@!17> <@!15>))))
   (IS-EQUAL <@!18> <@5>)
   (IS-EQUAL <@!19> <@!GOAL>)
   (IS-EQUAL <@!20> <@!OP>)
   (IS-EQUAL <@!21> <@!B>))

SIMPLIFIED-RESULT:
(OP-SUCCEEDS <@!NODE> <@!GOAL> <@!OP> <@!B>) if
(AND (KNOWN <@!NODE>
            (IS-TYPE <@!17> LOCATION))
     (KNOWN <@!NODE> (SUPPORTED-LOC <@!17>))
     (KNOWN <@!NODE>
            (T-ADJACENT-LOCS <@!17> <@!16>))
     (KNOWN <@!NODE> (AT RMG2 <@!16>))
     (FORALL (<@!15>)
             (KNOWN <@!NODE> (AT <@!14> <@!15>))
             (KNOWN <@!NODE>
                    (DISJOINT <@!17> <@!15>)))
     (IS-EQUAL (AT RMG2 <@!17>) <@!GOAL>)
     (IS-EQUAL MOVE-II <@!OP>)
     (IS-EQUAL (<@!16> <@!17>) <@!B>))
No rule: Dont-make-rule-flag -- INTERMED-RESULT
-------------------------
EBS Processing Node: N6


TRAINING EXAMPLE: (OP-SUCCEEDS N6 (AT RMG2 (LOCATION 1 1 0 1 1 0))
 MOVE-II ((LOCATION 1 0 0 1 0 0) (LOCATION 1 1 0 1 1 0)))     HIST: HST8

INITIAL-RESULT:
(OP-SUCCEEDS <@!NODE> <@!GOAL> <@!OP> <@!B>) if
(AND
   (KNOWN <@!NODE>
```

```
          (FORALL (<SQLOC-INFER-SUPPORTED-LOC>)
                  (SUPPORTING-SQLOCS <LOC-INFER-SUPPORTED-LOC>
                                     <SQLOC-INFER-SUPPORTED-LOC>)
                  (OCCUPIED-SQLOC <SQLOC-INFER-SUPPORTED-LOC>)))
   (AND
     (AND (IS-EQUAL (AT RMG2 <@22>) <@21>)
          (KNOWN <@!NODE>
                 (IS-TYPE <@22> LOCATION))
          (KNOWN <@!NODE>
                 (AND (IS-EQUAL SUPPORTED-LOC SUPPORTED-LOC)
                      (IS-EQUAL <@22> <LOC-INFER-SUPPORTED-LOC>)))
          (IS-EQUAL MOVE-II <@19>)
          (IS-EQUAL (<@18> <@22>) <@17>)
          (KNOWN <@!NODE>
                 (T-ADJACENT-LOCS <@22> <@18>))
          (KNOWN <@!NODE> (AT RMG2 <@18>))
          (FORALL (<@16>)
                  (KNOWN <@!NODE> (AT <@15> <@16>))
                  (KNOWN <@!NODE>
                         (DISJOINT <@22> <@16>))))
     (IS-EQUAL <@20> <@7>)
     (IS-EQUAL <@21> <@!GOAL>)
     (IS-EQUAL <@19> <@!OP>)
     (IS-EQUAL <@17> <@!B>)))

SIMPLIFIED-RESULT:
(OP-SUCCEEDS <@!NODE> <@!GOAL> <@!OP> <@!B>) if
(AND
  (FORALL (<SQLOC-INFER-SUPPORTED-LOC>)
          (KNOWN <@!NODE>
                 (SUPPORTING-SQLOCS <LOC-INFER-SUPPORTED-LOC>
                                    <SQLOC-INFER-SUPPORTED-LOC>))
          (KNOWN <@!NODE>
                 (OCCUPIED-SQLOC <SQLOC-INFER-SUPPORTED-LOC>)))
  (KNOWN <@!NODE>
         (IS-TYPE <LOC-INFER-SUPPORTED-LOC> LOCATION))
  (KNOWN <@!NODE>
         (T-ADJACENT-LOCS <LOC-INFER-SUPPORTED-LOC> <@18>))
  (KNOWN <@!NODE> (AT RMG2 <@18>))
  (FORALL (<@16>)
          (KNOWN <@!NODE> (AT <@15> <@16>))
          (KNOWN <@!NODE>
                 (DISJOINT <LOC-INFER-SUPPORTED-LOC> <@16>)))
  (IS-EQUAL (AT RMG2 <LOC-INFER-SUPPORTED-LOC>)
            <@!GOAL>)
  (IS-EQUAL MOVE-II <@!OP>)
  (IS-EQUAL (<@18> <LOC-INFER-SUPPORTED-LOC>)
            <@!B>))
No rule: Dont-make-rule-flag -- INTERMED-RESULT
-------------------------
EBS Processing Node: N2


TRAINING EXAMPLE: (OP-FAILS N2 (AT RMG2 (LOCATION 1 1 0 1 1 0))
 PUT-DOWN)      HIST: HST9

INITIAL-RESULT:
(OP-FAILS <@!NODE> <@!GOAL> <@!OP>) if
(AND (IS-EQUAL <@!GOAL> (AT <@5> <@6>))
     (AND (IS-EQUAL <@!OP> PUT-DOWN)
          (AND
            (AND
              (AND (IS-EQUAL (AT <@5> <@6>)
                             (AT <@!6> <@!5>))
                   (IS-EQUAL PUT-DOWN PUT-DOWN))
              (KNOWN <@!NODE>
```

```
                    (~ (OBJECT <@!6> <@!4>))))
          (IS-EQUAL <@!7> <@2>))))

SIMPLIFIED-RESULT:
(OP-FAILS <@!NODE> <@!GOAL> <@!OP>) if
(AND (IS-EQUAL <@!GOAL> (AT <@!6> <@!5>))
     (IS-EQUAL <@!OP> PUT-DOWN)
     (KNOWN <@!NODE>
            (~ (OBJECT <@!6> <@!4>))))
Turning on TP
New Learned Rule: R-1-9

TRAINING EXAMPLE: (SOLE-OP N2 (AT RMG2 (LOCATION 1 1 0 1 1 0)) MOVE-II)
   HIST: HST10

INITIAL-RESULT:
(SOLE-OP <@!NODE> <@!GOAL> <@!OP>) if
(AND (IS-EQUAL <@8> (AT <@9> <@10>))
     (MATCHES <@!GOAL> <@8>)
     (AND (IS-EQUAL <@8> (AT <@15> <@16>))
          (AND T T))
     (IS-EQUAL <@22> (AT <@23> <@24>))
     (MATCHES <@!GOAL> <@22>)
     (AND (IS-EQUAL <@22> (AT <@29> <@30>))
          (AND T T))
     (IS-EQUAL <@36> (AT <@37> <@38>))
     (MATCHES <@!GOAL> <@36>)
     (AND (IS-EQUAL <@36> (AT <@43> <@44>))
          (AND T T))
     (IS-EQUAL <@50> (AT <@51> <@52>))
     (MATCHES <@!GOAL> <@50>)
     (AND (IS-EQUAL <@50> (AT <@57> <@58>))
          (AND T T))
     (IS-EQUAL <@64> (AT <@65> <@66>))
     (MATCHES <@!GOAL> <@64>)
     (AND (IS-EQUAL <@64> (AT <@71> <@72>))
          (AND T T))
     (IS-EQUAL <@78> (AT <@79> <@80>))
     (MATCHES <@!GOAL> <@78>)
     (AND (IS-EQUAL <@78> (AT <@85> <@86>))
          (AND T T))
     (IS-EQUAL <@92> (AT <@93> <@94>))
     (MATCHES <@!GOAL> <@92>)
     (AND (IS-EQUAL <@92> (AT <@99> <@100>))
          (AND T T))
     (IS-EQUAL <@106> (AT <@107> <@108>))
     (MATCHES <@!GOAL> <@106>)
     (AND (IS-EQUAL <@106> (AT <@113> <@114>))
          (AND T T))
     (IS-EQUAL <@!GOAL> (AT <@!6> <@!5>))
     (IS-EQUAL PUT-DOWN PUT-DOWN)
     (KNOWN <@!NODE>
            (~ (OBJECT <@!6> <@!4>)))
     (IS-EQUAL <@123> (AT <@124> <@125>))
     (MATCHES <@!GOAL> <@123>)
     (AND (IS-EQUAL <@123> (AT <@130> <@131>))
          (AND T (NOT-EQUAL RMG <@130>)))
     (IS-EQUAL <@137> (AT <@138> <@139>))
     (MATCHES <@!GOAL> <@137>)
     (AND (IS-EQUAL <@!NODE> <@144>)
          (IS-EQUAL <@137> (AT RMG2 <@143>))
          (IS-EQUAL PUT-DOWN-II PUT-DOWN-II)
          (AND))
     (IS-EQUAL MOVE-II <@!OP>))

SIMPLIFIED-RESULT:
```

```
(SOLE-OP <@!NODE> <@!GOAL> <@!OP>) if
(AND (IS-EQUAL <@!GOAL> (AT RMG2 <@143>))
     (KNOWN <@!NODE>
            (~ (OBJECT RMG2 <@!4>)))
     (IS-EQUAL MOVE-II <@!OP>))
Turning on TP
New Learned Rule: R-1-10

TRAINING EXAMPLE: (OP-SUCCEEDS N2 (AT RMG2 (LOCATION 1 1 0 1 1 0))
MOVE-II ((LOCATION 1 0 0 1 0 0) (LOCATION 1 1 0 1 1 0)))     HIST: HST11

INITIAL-RESULT:
(OP-SUCCEEDS <@!NODE> <@!GOAL> <@!OP> <@!B>) if
(AND
  (AND (IS-EQUAL (AT RMG2 <@!30>) <@!32>)
       (KNOWN <@!NODE>
              (IS-TYPE <@!30> LOCATION))
       (IS-EQUAL MOVE-II <@!33>)
       (IS-EQUAL (<@!29> <@!30>) <@!34>)
       (KNOWN <@!NODE>
              (T-ADJACENT-LOCS <@!30> <@!29>))
       (KNOWN <@!NODE> (AT RMG2 <@!29>))
       (FORALL (<@!28>)
               (KNOWN <@!NODE>
                      (SUPPORTING-SQLOCS <@!30> <@!28>))
               (KNOWN <@!NODE>
                      (OCCUPIED-SQLOC <@!28>)))
       (FORALL (<@!27>)
               (KNOWN <@!NODE> (AT <@!26> <@!27>))
               (KNOWN <@!NODE>
                      (DISJOINT <@!30> <@!27>))))
  (IS-EQUAL <@!31> <@5>)
  (IS-EQUAL <@!32> <@!GOAL>)
  (IS-EQUAL <@!33> <@!OP>)
  (IS-EQUAL <@!34> <@!B>))

SIMPLIFIED-RESULT:
(OP-SUCCEEDS <@!NODE> <@!GOAL> <@!OP> <@!B>) if
(AND (KNOWN <@!NODE>
            (IS-TYPE <@!30> LOCATION))
     (KNOWN <@!NODE>
            (T-ADJACENT-LOCS <@!30> <@!29>))
     (KNOWN <@!NODE> (AT RMG2 <@!29>))
     (FORALL (<@!28>)
             (KNOWN <@!NODE>
                    (SUPPORTING-SQLOCS <@!30> <@!28>))
             (KNOWN <@!NODE>
                    (OCCUPIED-SQLOC <@!28>)))
     (FORALL (<@!27>)
             (KNOWN <@!NODE> (AT <@!26> <@!27>))
             (KNOWN <@!NODE>
                    (DISJOINT <@!30> <@!27>)))
     (IS-EQUAL (AT RMG2 <@!30>) <@!GOAL>)
     (IS-EQUAL MOVE-II <@!OP>)
     (IS-EQUAL (<@!29> <@!30>) <@!B>))
Turning on TP
New Learned Rule: R-1-11

4 new rules added.

OBSERVER reset

Number of rules learned: 4
Number of learned rules still in: 4
```

```
Prodigy: write-rules "rules-1step.ebl
"


Prodigy: show-rules


Rule: R-1-11
(LHS
  (AND (CURRENT-NODE <@!R23>)
       (CURRENT-GOAL <@!R23> (AT RMG2 <&@!R19>))
       (CANDIDATE-OP <@!R23> MOVE-II)
       (KNOWN <@!R23>
              (IS-TYPE <&@!R19> LOCATION))
       (KNOWN <@!R23>
              (T-ADJACENT-LOCS <&@!R19> <&@!R18>))
       (KNOWN <@!R23> (AT RMG2 <&@!R18>))
       (FORALL (<@!R17>)
               (KNOWN <@!R23>
                      (SUPPORTING-SQLOCS <&@!R19> <@!R17>))
               (KNOWN <@!R23>
                      (OCCUPIED-SQLOC <@!R17>)))
       (FORALL (<@!R16>)
               (KNOWN <@!R23> (AT <@!R15> <@!R16>))
               (KNOWN <@!R23>
                      (DISJOINT <&@!R19> <@!R16>)))
       (CANDIDATE-OP <@!R23> <@!OTHER-OP>)
       (NOT-EQUAL MOVE-II <@!OTHER-OP>)))
(RHS (PREFER OPERATOR MOVE-II <@!OTHER-OP>))

Rule: R-1-10
(LHS
  (AND (CURRENT-NODE <@!R14>)
       (CURRENT-GOAL <@!R14> (AT RMG2 <@!R11>))
       (CANDIDATE-OP <@!R14> MOVE-II)
       (KNOWN <@!R14>
              (~ (OBJECT RMG2 <@!R10>)))))
(RHS (SELECT OPERATOR MOVE-II))

Rule: R-1-9
(LHS
  (AND (CURRENT-NODE <@!R9>)
       (CURRENT-GOAL <@!R9> (AT <@!R6> <@!R5>))
       (CANDIDATE-OP <@!R9> PUT-DOWN)
       (KNOWN <@!R9>
              (~ (OBJECT <@!R6> <@!R4>)))))
(RHS (REJECT OPERATOR PUT-DOWN))

Rule: R-1-1
(LHS
  (AND (CANDIDATE-NODE <@!R3>)
       (PRIMARY-CANDIDATE-GOAL <@!R3>
                               (OBJECT <@!R2> <@!R1>))))
(RHS (REJECT NODE <@!R3>))



Prodigy: exit

> ;;; End of dribble transcript to /home/mlis3a/freeman/prodigy/domains/extended-s
```

6.5

2-Robot Sample Solution

```
      ...Done: INFER-SUPPORTED-LOC (LOCATION 0 1 0 0 1 0)

      N22 (AT RMG (LOCATION 0 1 0 0 1 0))
      Alts: MOVE <ADJ-LOC-MOVE> (LOCATION 0 1 0 0 1 0)

         N23 (VACANT-LOC (LOCATION 0 1 0 0 1 0))
         Alts: INFER-VACANT-LOC (LOCATION 0 1 0 0 1 0)

      ...Done: INFER-VACANT-LOC (LOCATION 0 1 0 0 1 0)

      N24 (AT RMG (LOCATION 0 1 0 0 1 0))
      Alts: MOVE (LOCATION 0 0 0 0 0 0) (LOCATION 0 1 0 0 1 0)

   ...Done: MOVE (LOCATION 0 0 0 0 0 0) (LOCATION 0 1 0 0 1 0)

   N25 (HOLDING BLOCK-C)
   Alts: PICK-UP BLOCK-C (LOCATION 0 2 0 0 2 0)

      N26 (CLEAR BLOCK-C)
      Alts: INFER-CLEAR BLOCK-C

         N27 (VACANT-LOC (LOCATION 0 2 1 0 2 1))
         Alts: INFER-VACANT-LOC (LOCATION 0 2 1 0 2 1)

      ...Done: INFER-VACANT-LOC (LOCATION 0 2 1 0 2 1)

      N28 (CLEAR BLOCK-C)
      Alts: INFER-CLEAR BLOCK-C

   ...Done: INFER-CLEAR BLOCK-C

   N29 (HOLDING BLOCK-C)
   Alts: PICK-UP BLOCK-C (LOCATION 0 2 0 0 2 0)

      N30 (ARM-EMPTY)
      Alts: INFER-ARM-EMPTY

   ...Done: INFER-ARM-EMPTY

   N31 (HOLDING BLOCK-C)
   Alts: PICK-UP BLOCK-C (LOCATION 0 2 0 0 2 0)

...Done: PICK-UP BLOCK-C (LOCATION 0 2 0 0 2 0)

N32 (AT BLOCK-C (LOCATION 2 0 0 2 0 0))
Alts: PUT-DOWN BLOCK-C (LOCATION 2 0 0 2 0 0)

   N33 (VACANT-LOC (LOCATION 2 0 0 2 0 0))
   Alts: INFER-VACANT-LOC (LOCATION 2 0 0 2 0 0)

...Done: INFER-VACANT-LOC (LOCATION 2 0 0 2 0 0)

N34 (AT BLOCK-C (LOCATION 2 0 0 2 0 0))
Alts: PUT-DOWN BLOCK-C (LOCATION 2 0 0 2 0 0)

   N35 (SUPPORTED-LOC (LOCATION 2 0 0 2 0 0))
   Alts: INFER-SUPPORTED-LOC (LOCATION 2 0 0 2 0 0)

...Done: INFER-SUPPORTED-LOC (LOCATION 2 0 0 2 0 0)

N36 (AT BLOCK-C (LOCATION 2 0 0 2 0 0))
Alts: PUT-DOWN BLOCK-C (LOCATION 2 0 0 2 0 0)

   N37 (SUPPORTED-LOC (LOCATION 1 0 0 1 0 0))
   Alts: INFER-SUPPORTED-LOC (LOCATION 1 0 0 1 0 0)
```

```
...Done: INFER-SUPPORTED-LOC (LOCATION 1 0 0 1 0 0)

N38 (AT BLOCK-C (LOCATION 2 0 0 2 0 0))
Alts: PUT-DOWN BLOCK-C (LOCATION 2 0 0 2 0 0)

   N39 (AT RMG (LOCATION 1 0 0 1 0 0))
   Alts: MOVE <ADJ-LOC-MOVE> (LOCATION 1 0 0 1 0 0) [2]

      N40 (VACANT-LOC (LOCATION 1 0 0 1 0 0))
      Alts: INFER-VACANT-LOC (LOCATION 1 0 0 1 0 0)

   ...Done: INFER-VACANT-LOC (LOCATION 1 0 0 1 0 0)

   N41 (AT RMG (LOCATION 1 0 0 1 0 0))
   Alts: MOVE (LOCATION 1 1 0 1 1 0) (LOCATION 1 0 0 1 0 0) [2-6]

      N42 (AT RMG (LOCATION 1 1 0 1 1 0))
      Alts: MOVE <ADJ-LOC-MOVE> (LOCATION 1 1 0 1 1 0) [2]

         N43 (SUPPORTED-LOC (LOCATION 1 1 0 1 1 0))
         Alts: INFER-SUPPORTED-LOC (LOCATION 1 1 0 1 1 0)

      ...Done: INFER-SUPPORTED-LOC (LOCATION 1 1 0 1 1 0)

      N44 (AT RMG (LOCATION 1 1 0 1 1 0))
      Alts: MOVE <ADJ-LOC-MOVE> (LOCATION 1 1 0 1 1 0)

         N45 (VACANT-LOC (LOCATION 1 1 0 1 1 0))
         Alts: INFER-VACANT-LOC (LOCATION 1 1 0 1 1 0)

      ...Done: INFER-VACANT-LOC (LOCATION 1 1 0 1 1 0)

      N46 (AT RMG (LOCATION 1 1 0 1 1 0))
      Alts: MOVE (LOCATION 0 1 0 0 1 0) (LOCATION 1 1 0 1 1 0)

   ...Done: MOVE (LOCATION 0 1 0 0 1 0) (LOCATION 1 1 0 1 1 0)

   N47 (AT RMG (LOCATION 1 0 0 1 0 0))
   Alts: MOVE (LOCATION 1 1 0 1 1 0) (LOCATION 1 0 0 1 0 0)

      N48 (SUPPORTED-LOC (LOCATION 1 0 0 1 0 0))
      Alts: INFER-SUPPORTED-LOC (LOCATION 1 0 0 1 0 0)

   ...Done: INFER-SUPPORTED-LOC (LOCATION 1 0 0 1 0 0)

   N49 (AT RMG (LOCATION 1 0 0 1 0 0))
   Alts: MOVE (LOCATION 1 1 0 1 1 0) (LOCATION 1 0 0 1 0 0)

      N50 (VACANT-LOC (LOCATION 1 0 0 1 0 0))
      Alts: INFER-VACANT-LOC (LOCATION 1 0 0 1 0 0)

   ...Done: INFER-VACANT-LOC (LOCATION 1 0 0 1 0 0)

   N51 (AT RMG (LOCATION 1 0 0 1 0 0))
   Alts: MOVE (LOCATION 1 1 0 1 1 0) (LOCATION 1 0 0 1 0 0)

...Done: MOVE (LOCATION 1 1 0 1 1 0) (LOCATION 1 0 0 1 0 0)

N52 (AT BLOCK-C (LOCATION 2 0 0 2 0 0))
Alts: PUT-DOWN BLOCK-C (LOCATION 2 0 0 2 0 0)

   N53 (VACANT-LOC (LOCATION 2 0 0 2 0 0))
   Alts: INFER-VACANT-LOC (LOCATION 2 0 0 2 0 0)

...Done: INFER-VACANT-LOC (LOCATION 2 0 0 2 0 0)
```

```
            N54 (AT BLOCK-C (LOCATION 2 0 0 2 0 0))
            Alts: PUT-DOWN BLOCK-C (LOCATION 2 0 0 2 0 0)

               N55 (SUPPORTED-LOC (LOCATION 2 0 0 2 0 0))
               Alts: INFER-SUPPORTED-LOC (LOCATION 2 0 0 2 0 0)

            ...Done: INFER-SUPPORTED-LOC (LOCATION 2 0 0 2 0 0)

            N56 (AT BLOCK-C (LOCATION 2 0 0 2 0 0))
            Alts: PUT-DOWN BLOCK-C (LOCATION 2 0 0 2 0 0)

               N57 (SUPPORTED-LOC (LOCATION 1 0 0 1 0 0))
               Alts: INFER-SUPPORTED-LOC (LOCATION 1 0 0 1 0 0)

            ...Done: INFER-SUPPORTED-LOC (LOCATION 1 0 0 1 0 0)

            N58 (AT BLOCK-C (LOCATION 2 0 0 2 0 0))
            Alts: PUT-DOWN BLOCK-C (LOCATION 2 0 0 2 0 0)

         ...Done: PUT-DOWN BLOCK-C (LOCATION 2 0 0 2 0 0)

         N59 (OCCUPIED-SQLOC (LOCATION 2 0 0 2 0 0))
         Alts: INFER-OCCUPIED-SQLOC (LOCATION 2 0 0 2 0 0) BLOCK-C

      ...Done: INFER-OCCUPIED-SQLOC (LOCATION 2 0 0 2 0 0) BLOCK-C

      N60 (SUPPORTED-LOC (LOCATION 2 0 1 2 0 1))
      Alts: INFER-SUPPORTED-LOC (LOCATION 2 0 1 2 0 1)

   ...Done: INFER-SUPPORTED-LOC (LOCATION 2 0 1 2 0 1)

   N61 (AT RMG (LOCATION 2 0 1 2 0 1))
   Alts: MOVE <ADJ-LOC-MOVE> (LOCATION 2 0 1 2 0 1)

      N62 (VACANT-LOC (LOCATION 2 0 1 2 0 1))
      Alts: INFER-VACANT-LOC (LOCATION 2 0 1 2 0 1)

   ...Done: INFER-VACANT-LOC (LOCATION 2 0 1 2 0 1)

   N63 (AT RMG (LOCATION 2 0 1 2 0 1))
   Alts: MOVE (LOCATION 1 0 0 1 0 0) (LOCATION 2 0 1 2 0 1) [2-9]

...Done: MOVE (LOCATION 1 0 0 1 0 0) (LOCATION 2 0 1 2 0 1)

N64 (AT RMG (LOCATION 3 0 2 3 0 2))
Alts: MOVE (LOCATION 2 0 1 2 0 1) (LOCATION 3 0 2 3 0 2)

   N65 (SUPPORTED-LOC (LOCATION 3 0 2 3 0 2))
   Alts: INFER-SUPPORTED-LOC (LOCATION 3 0 2 3 0 2)

      N66 (OCCUPIED-SQLOC (LOCATION 3 0 1 3 0 1))
      Alts: INFER-OCCUPIED-SQLOC (LOCATION 3 0 1 3 0 1) BLOCK-A [2]

   ...Done: INFER-OCCUPIED-SQLOC (LOCATION 3 0 1 3 0 1) BLOCK-A

   N67 (SUPPORTED-LOC (LOCATION 3 0 2 3 0 2))
   Alts: INFER-SUPPORTED-LOC (LOCATION 3 0 2 3 0 2)

...Done: INFER-SUPPORTED-LOC (LOCATION 3 0 2 3 0 2)

N68 (AT RMG (LOCATION 3 0 2 3 0 2))
Alts: MOVE (LOCATION 2 0 1 2 0 1) (LOCATION 3 0 2 3 0 2)

   N69 (VACANT-LOC (LOCATION 3 0 2 3 0 2))
   Alts: INFER-VACANT-LOC (LOCATION 3 0 2 3 0 2)
```

```
   ...Done: INFER-VACANT-LOC (LOCATION 3 0 2 3 0 2)

   N70 (AT RMG (LOCATION 3 0 2 3 0 2))
   Alts: MOVE (LOCATION 2 0 1 2 0 1) (LOCATION 3 0 2 3 0 2)

...Done: MOVE (LOCATION 2 0 1 2 0 1) (LOCATION 3 0 2 3 0 2)

N71 (DONE)
Alts: *FINISH*

...Done: *FINISH*

Completed Success

CPU time: 105.7 seconds
Number of Nodes: 72
Solution Length: 35
Operator Sequence:

      MOVE (LOCATION 0 0 0 0 0 0) (LOCATION 0 1 0 0 1 0)
      PICK-UP BLOCK-C (LOCATION 0 2 0 0 2 0) (LOCATION 0 1 0 0 1 0)
      MOVE (LOCATION 0 1 0 0 1 0) (LOCATION 1 1 0 1 1 0)
      MOVE (LOCATION 1 1 0 1 1 0) (LOCATION 1 0 0 1 0 0)
      PUT-DOWN BLOCK-C (LOCATION 2 0 0 2 0 0) (LOCATION 1 0 0 1 0 0)
      MOVE (LOCATION 1 0 0 1 0 0) (LOCATION 2 0 1 2 0 1)
      MOVE (LOCATION 2 0 1 2 0 1) (LOCATION 3 0 2 3 0 2)

Prodigy: (exit)

Type ``(prodigy)'' to begin another session.

BYE
> (quit)
;;; End of dribble transcript to /home/mlis3a/freeman/prodigy/domains/gridworld/no
```

6.6

Index of Primitives

This listing includes the names of the LISP predicates defined in two of the more useful files in the PLANNER file structure. Other files containing commonly used predicates are misc.lisp, data-types.lisp and g-map.lisp. The ones listed below explain a large portion of the syntax and semantics of the PDL used in describing the search rules and operators. of any domain. This is an incomplete list of ALL predicates used. In addition, all Common Lisp predicates are allowable and they are not included below.

Gridworld/functions.lisp static predicates

```
(defun covers (loc loc-size sqloc)
(defun cover-gen (size sqloc)
(defun in-location (sqloc loc)
(defun next-to (loc sqloc)
(defun next-to-gen (loc)
(defun gen-sqlocs (min-x min-y min-z max-x max-y max-z)
(defun t-adjacent-locs (loc-a loc-b)
(defun t-adj-gen (loc)
(defun within-height (location RMG-location reach)
(defun location-size (location location-size)
(defun size-calc (loc)
(defun equal-p (first-arg second-arg)
(defun equalp-bind (var value)
(defun ground-loc (loc)
(defun is-type (thing thing-type)
(defun towards (f-loc a-loc t-loc)
(defun distance-calc (a-loc b-loc)
(defun distance-sq-to-loc (loc-sq loc)
(defun closest-coor (coor min-coor max-coor)
(defun distance (a-loc b-loc dist)
(defun less-than (a b)
(defun above-loc (a-loc b-loc)
(defun supporting-sqlocs (location sqloc)
(defun gen-corner-loc (location)
(defun disjoint (loc-a loc-b)
(defun under-loc (loc-a loc-b)
(defun binding-list (var val-list)
(defun type-error (function)
(defun is-location (location)
(defun is-square-location (location)
(defun is-size (size)
(defun make-location (x1 y1 z1 x2 y2 z2)
(defun min-x-coor (location)
(defun min-y-coor (location)
(defun min-z-coor (location)
(defun max-x-coor (location)
(defun max-y-coor (location)
(defun max-z-coor (location)
(defun x-dim (size)
(defun y-dim (size)
(defun z-dim (size)
```

System/Planner/Meta-fns.lisp

```
(defun not-equal (a b)
(defun is-bound (v)
(defun is-equal (a b)
(defun candidate-node (n)
(defun primary-candidate-node (n)
(defun node-pref-not-cached (n1 n2)
(defun latest-node-pref (n)
```

```
(defun created-later-than (n1 n2)
(defun current-node (n)
(defun not-top-level-node (n)
(defun is-top-level-node (n)
(defun list-of-candidate-goals (node goals)
(defun is-first-goal (goal goals)
(defun which-is-parent (possible-parents n)
(defun protected-goal (n g)
(defun was-added (node lit)
(defun was-added-above (n lit)
(defun was-deleted (node lit)
(defun find-where-lit-absent (node lit)
(defun was-deleted-above (n lit)
(defun current-op (node op)
(defun candidate-op (node op)
(defun candidate-goal (node g)
(defun primary-candidate-goal (node g)
(defun high-on-goal-stack (n g)
(defun on-goal-stack (n super)
(defun direct-supergoal-of (node sub super)
(defun is-top-level-goal (node goal)
(defun in-goal-exp (node goal)
(defun adjunct-goal (node goal)
(defun current-goal (node goal)
(defun candidate-bindings (node vals-list)
(defun constant-tst (bindings)
(defun lst-match (slst olst)
(defun previous-state-diff (n diffs)
(defun are-in-state (dels s)
(defun diff-match (diffs actuals)
(defun r-diff-match (diffs orig-actuals orig-bindings)
(defun get-actual-diffs (cs ns)
(defun get-adds-from-diffs (diffs)
 (defun get-dels-from-diffs (diffs)
(defun alt-on-deck (n g op)
(defun r-alt-on-deck (n g op)
(defun has-bound-vars (n g op bvars)
(defun r-has-bound-vars (n g op bvars)
(defun new-bvar-bindings (bvars alt bindings)
(defun known (node exp)
(defun initialize-recur-start-node (current-node)
(defun achievable (current-node exp)
(defun has-any-vars (exp)
(defun provable (current-node exp)
(defun recur-run (inferences-only-flag current-node goal-exp)
(defun applied-operator (node &optional (oper nil))
(defun applied-inference (node &optional (infer nil))
(defun was-deleted-by (node lit op vars)
```

REFERENCES

1. Greiner, Russel & Likuski, Joseph, "Incorporating Redundant Learned Rules: A Preliminary Formal Analysis of EBL", Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Vol II, Morgan Kaufmann Publishers, Inc., 1989, pp. 744-749.

2. Minton, Steve, Documentation for Prodigy Version 2.0 EBL Module, published as a part of the Prodigy Planner. See CMU technical report CMU-CS-89-146.

3. Minton, Steve, Learning Search Control Knowledge, An Explanation-Based Approach, Kluwer Academic Publishers, 1988, pp. 1-184.

4. Minton, Steven, et. al., Prodigy 2.0 : The Manual and Tutorial, CMU-CS-89-146, Carnegie Mellon University, Pittsburgh, Pa. 15213, May 1989, pp. 1-68.

5. Nilsson, Nils J., Principles of Artificial Intelligence, Morgan Kaufmann Publishers, 1980, pp.19-28.