The PRODIGY User Interface

Jim Blythe, Manuela Veloso, and Luiz Edival de Souza[1]

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213 USA
{*blythe,veloso,ledival*}@*cs.cmu.edu*

Abstract

The PRODIGY user interface supports the process of both building and running a planning domain in PRODIGY. It was designed to be highly modular, requiring no changes to the code of the PRODIGY planner to run, and extensible, so that interfaces to other modules built on PRODIGY could easily be integrated into the interface. In this paper we describe how these goals were achieved. We demonstrate building a domain and animating the planning process. We describe extensions to the user interface to support planning by analogical reasoning and probabilistic planning with PRODIGY.

[1] On leave from Escola Federal de Engenharia de Itajubá, MG, Brazil. Visiting researcher at CMU with financial support provided by CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico, Brasília, DF.

1 Introduction

PRODIGY is a domain-independent planning and learning system that provides a base planning module and many capabilities implemented as integrated modules [Veloso *et al.*, 1995]. We recently developed a graphical user interface to PRODIGY that provides support for several of the tasks involved in developing and using a planning domain in PRODIGY. The PRODIGY user interface has a number of functions. First, it provides a visual animation of the planning procedure and visual representations of the output. Second, it improves the process of creating and debugging planning domains in PRODIGY. Third, it provides uniform access to the modules built on top of PRODIGY, supporting a variety of visualization strategies.

Since PRODIGY is an on-going research project, it was important to develop an interface that could be integrated easily with any variant of the system. This requirement led us to seek an architecture that is modular in two ways. First, the code for the planner should not need to be modified for the user interface to run. Second, the user interface itself should make very few assumptions about the planner's implementation. At the same time the interface should have a tight integration with the planner so that the planning process can be traced graphically and the interface can be used to interrupt and direct the planner while it is in operation.

The architecture used for the PRODIGY user interface accomplishes these goals. In addition, our use of existing off-the-shelf software components allowed the interface to be implemented relatively quickly. Figure 1 shows a typical view of the interface.

In this paper we describe the architecture, implementation and the current capabilities of the interface. We hope that the information will be of use to developers of other planning systems, as well as other users of PRODIGY. In the remainder of this section we describe the architecture of the interface, and the systems used in its implementation. Section 2 describes the capabilities of the core part of the interface, namely the visualization of the planning algorithm. The user can dynamically follow an animation of the algorithm. If desired, the user may also interrupt and step through the planner to analyze in greater detail a particular episode. Finally, the user can control directly the planner, as well as freely change different planning search strategies. Section 3 presents the framework that we are currently developing to support the user with building a planning domain. In Section 4 we briefly present the interfaces for two modules built on top of PRODIGY: planning by analogical reasoning and a probabilistic planner.

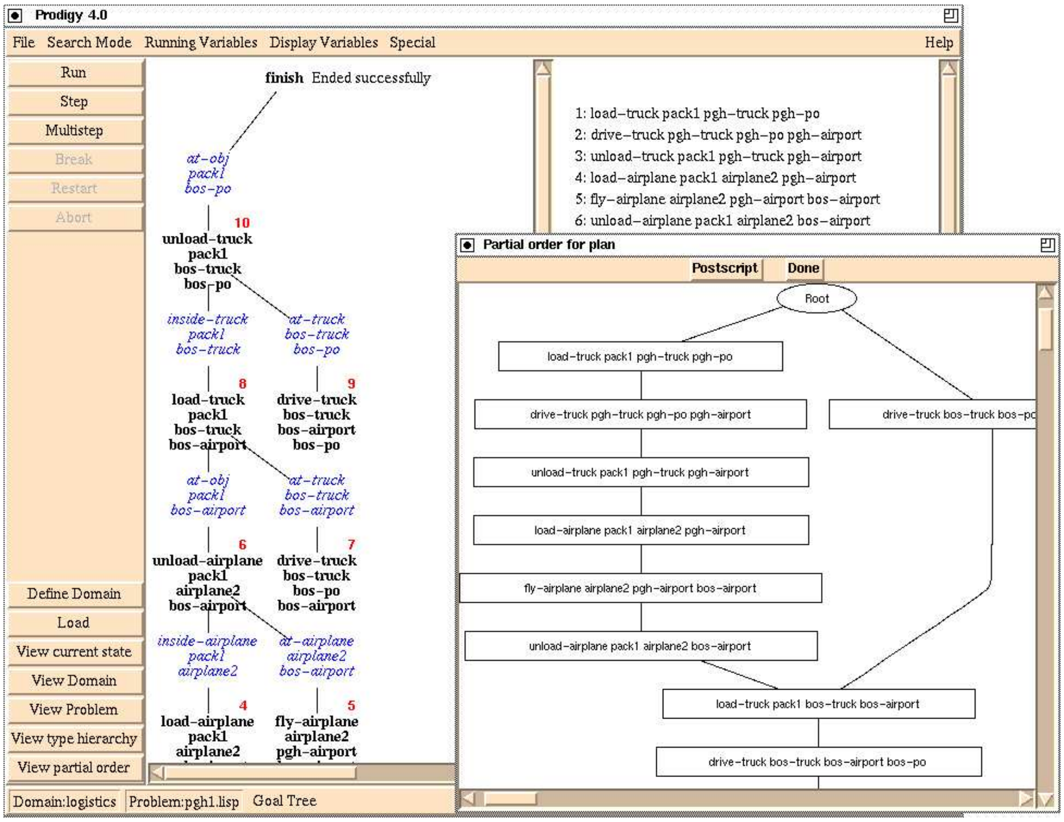

Figure 1: A snapshot from the user interface shows two windows. The window at the back contains the basic controls, a drawing of the goal tree and of the plan tail. The window in front displays the partial order for the plan that was produced.

1.1 Architecture and implementation details

The user interface runs in a separate process from the planner, and the two communicate through sockets. This architecture makes the systems highly modular. As long as the planner and the interface agree on a set of messages to be passed between them, each can be implemented in any language and use a number of different algorithms. It is possible for the interface to run on a different machine from the planner, but this is not usually done in practice. The planner currently runs in a Common Lisp process that listens for commands from two sources: the terminal running lisp and a socket connected to the user interface. This allows problems to be loaded or planning to be initiated from either the terminal or the interface.

The user interface is implemented in Tcl/Tk, a scripting language which includes a set of Motif widgets, and makes use of "Dag", a freely available preprocessor for drawing directed graphs [Gansner *et al.*, 1988]. For example, when the user clicks on "view partial order", the UI sends a message to the planner to print the plan's partial order to a file, and runs Dag on that file when the planner replies that the task is completed. The output from Dag is a file containing placements for nodes and spline commands for edges that the UI reads to produce a window containing the graph. It attaches actions to the nodes so that if the user clicks on them it will pass another message to the planner to display appropriate information.

The simple communication described is enough to start the planner from the interface and interpret its output graphically. A tighter communication is required for the user to be able to interrupt and direct the planner from the interface during its operation. This is achieved without altering the code of the planner using PRODIGY's "interrupt system". The PRODIGY planner searches for a plan by repeatedly selecting an open node in its search tree, selecting one way to expand the node, and placing the new node on the search tree. On each pass through this inner cycle, it looks for code that is specified by the user and runs it. This can be arbitrary lisp code that can cause the planner to terminate if desired. In the normal operation of PRODIGY, resource bounds such as the size of the search space or cpu limits are implemented using this system. It is also used by the UI.

When the planner is run from the UI, the UI attaches code to the interrupt system that causes the planner to pass a message to the UI on each iteration through its search process and wait for a response before continuing. The response can lead the interrupt system to terminate planning. This scheme provides real-time information to the UI during planning and allows the UI to interrupt or halt the planner. The UI uses this to implement a plan stepper that is described in the next section.

The requirement that the planner halt and wait for a message from the UI on each iteration of its inner loop could in principle slow the planner down significantly. However in practice we have found that even when the UI computes the position for the node and draws the node in a window containing the goal tree, the process is faster than the alternative tracing method of printing a line of text to the lisp terminal.

2 Visualizing the planning algorithm

One of the main goals underlying the design of the graphical user interface for PRODIGY4.0 was to provide a clear animation of the planning algorithm [Carbonell *et al.*, 1992]. In this section we first overview briefly the PRODIGY4.0's planning algorithm.[2] We then discuss several features in the user interface that enable the visualization of the running of algorithm. The design of our user interface benefitted from and was inspired by other planning user interfaces, such as the ones from SIPE [Wilkins *et al.*, 1995], CAPLAN [Paulokat and Wess, 1994], and O-PLAN [Tate and Drabble, 1995].

2.1 Overview of PRODIGY4.0's planning algorithm

PRODIGY4.0 is a nonlinear planner that follows a means-ends analysis backward chaining search procedure reasoning about multiple goals and multiple alternative operators relevant to achieving the goals. PRODIGY4.0's nonlinear character stems from its dynamic goal selection which enables the planner to fully interleave sub-plans, exploiting common subgoals and addressing issues of resource contention. Operators can be organized in different levels of abstractions that are used by PRODIGY4.0 to plan hierarchically. Table 1 shows a top-level view of the PRODIGY planning algorithm.

1. Initialize.
2. Terminate if the goal statement has been satisfied.
3. Determine which goals are pending, i.e. still need to be achieved.
4. Determine if there are any selected operators that have their preconditions satisfied in the current state, and hence could be applied to the state as the next step in the plan.
5. Choose to subgoal on a goal or to apply an operator: *(backtrack point)*
    - To subgoal, go to step 6.
    - To apply, go to step 7.
6. Select one of the pending goals *(no backtrack point)*, an instantiated operator that can achieve it *(backtrack point)*; go to step 3.
7. Change the state according to an applicable operator *(backtrack point)*; go to step 2.

Table 1: A top-level view of PRODIGY4.0's planning algorithm.

[2] The reader familiar with the algorithm may skip this brief overview.

As shown in Table 1, PRODIGY4.0 follows a sequence of decision choices, selecting a goal, an operator, and an instantiation for the operator to achieve the goal. PRODIGY4.0 has an additional decision point, namely where it decides whether to "apply" an operator to the current state or continue "subgoaling" on a pending goal. "Subgoaling" can be best understood as regressing one goal, or backward chaining, using means-ends analysis. It includes the choices of a goal to plan for and an operator to achieve this goal. "Applying" an operator to the state means a commitment (not necessarily definite since backtracking is possible) in the ordering of the final plan. On the other hand, updating the state through this possible commitment allows PRODIGY4.0 to use its state to more informed and efficient future decisons. Hence, PRODIGY4.0's planning algorithm is a combination of state-space search corresponding to a simulation of plan execution of the plan and backward-chaining responsible for goal-directed reasoning.

2.2 The head-plan and the tail-plan windows

From a plain search point of view all of the four decision points presented above are equivalent. However it showed beneficial to capture PRODIGY4.0's behavior by separating the part of the plan resulting from applying operators, i.e. the *head plan* and the subgoaling structure, i.e. the *tail plan* [Fink and Veloso, 1995]. A search node in PRODIGY4.0 can be seen then as being composed by the head and tail plans. The PRODIGY user interface captures this distinction as shown in Figure 2. It divides the planning window into the tail-plan and head-plan windows. Operators, when applied, are represented in the head-plan window. The figure shows an additional window where the current state can be viewed at any time during planning.

The head-plan is represented simply as an ordered list of the operators applied to the state. The tail-plan is represented as a tree with alternating goal and instantiated operator nodes. We currently use a layout routine that minimizes the space taken by the tree.

2.3 Run, step, break, restart, abort

PRODIGY4.0, as most planners, involves the generation and exploration of alternatives. Choices are made, explored, and possibly abandoned if found not feasible or not suitable. The tail-plan and the head-plan change dynamically during the planning search episode. It is clearly a challenging task to provide an effective

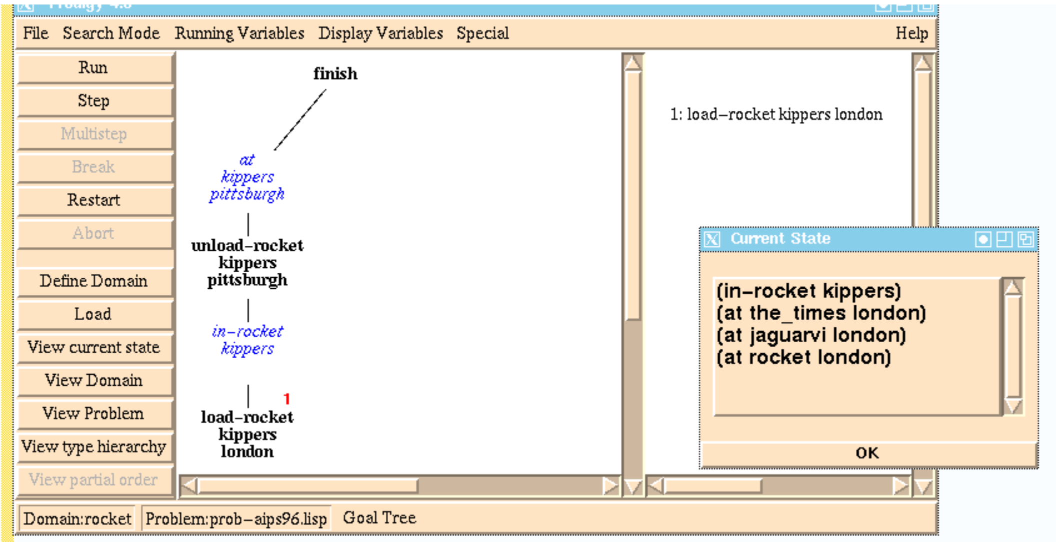

Figure 2: The left window contains the tail-plan and the right window the head-plan. The operator *load-rocket kippers london* was applied and *in kippers rocket* was added to the current state. The user asked to visualize the state by selecting *View Current State*. The tail-plan shows the goals and operators that have been selected to achieve these operators.

visualization of this dynamic process. The PRODIGY user interface currently includes the following features to facilitate the user's understanding of the planning search procedure:

- Run: The planning process is started autonomously.
- Step: The user can *step* through the planning process. PRODIGY4.0 cycles through and breaks at each of its choices at each `Step` command.
- Break: At any time, the user can interrupt the planning procedure by selecting a `Break`. The interface sends a break command to the running Lisp process and the planning algorithm is interrupted. This allows the user to analyze in detail a particular planning situation.
- Restart: There are two situations in which a `Restart` command may be selected: After a break command and as a termination of the stepping mode. In both cases, the planning process is reactived autonomusly, i.e. as in run mode.
- Abort: The user can abort the planning procedure at any time. This may

happen more often while debugging a planning domain, in which case the user may notice that the planning process is diverging from its intended performance. The process is aborted, the user can engage in further refinements, and restart new planning experiments.

The commands described above can occur in a variety of different sequences. These are some examples of possible sequences of commands: `Step`, `Restart`, `Break`, `Step`, `Restart`; or `Run`, `Break`, `Step`, `Restart`.

2.4 Control of choices

PRODIGY is a completely *open-controllable* architecture. This means that all the decision points are open to be controlled explicitly usually through control rules which can automatically dictate particular choices based on the planning scenario. Figure 3 illustrates a particular use of control rules that interrupt the planner to ask the user for guidance.

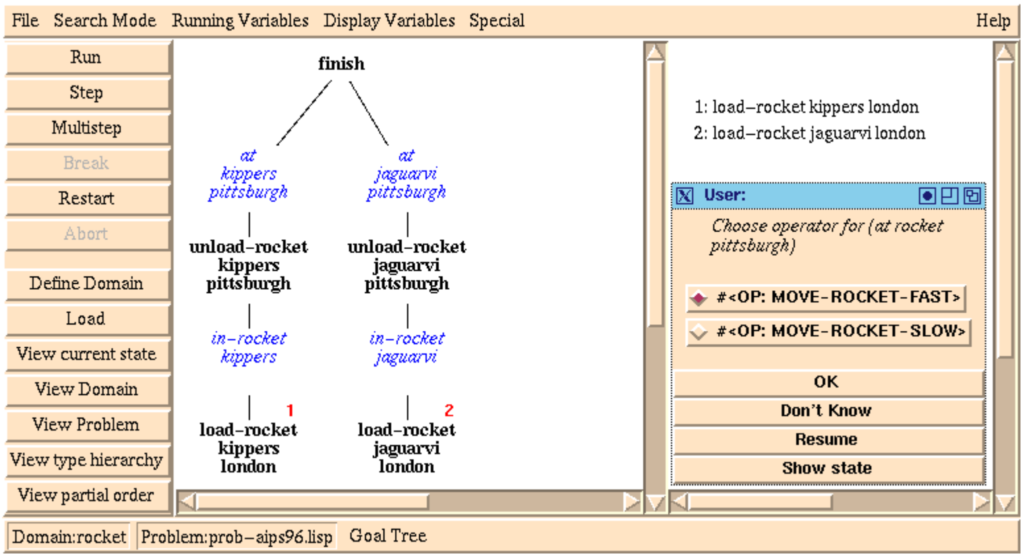

Figure 3: The user can control choices. In the right window, the user is shown the operator choices available to achieve the goal PRODIGY is planning for.

The user is prompted with the choices available and can either select one of the alternatives, decide that doesn't know which one to select, or tell PRODIGY not to ask for any more guidance and resume its autonomous behavior.

2.5 Additional features

In PRODIGY4.0 the head-plan is the solution plan encountered. Although this final solution is the sequence of operators applied to the state, PRODIGY4.0 knows the actual dependencies between the plan steps in terms of the corresponding preconditions and effects.

The user can view the partially ordered final plan by selecting the choice to `View Partial Order`. The head-plan is one of the possibly many linearizations of the partial order generated. Figure 1 shows the partial order window.

Finally, the user can change the specific planning search mode, e.g. from eager to delayed step ordering commitments [Stone *et al.*, 1994], can set a variety of different running parameters, and can select different display setups.

3 Building the Application Domain in PRODIGY

To develop an application domain in PRODIGY, the user must specify an object type hierarchy, a set of predicates and a set of operators, all of which have a specific Lisp syntax. The development can be done in three different ways. The most obvious way to build the domain is to create the Lisp structures directly. This requires a complete knowledge of the system representation language. Another approach, embodied in a system called APPRENTICE [Joseph, 1992], was developed to produce the domain from a graphical specification. This method was shown to be successful for visually-oriented domains.

We have implemented a form-based tool called *Domain Builder* in the PRODIGY user interface that allows interactive domain development within the planning system. Like APPRENTICE, the tool generates a syntactically correct domain definition. The interface is linked directly to the planning system so that the user can develop, test, and run a domain interactively.

Domain Builder provides a menu-based interface for all the steps needed to specify the domain. To illustrate the tool, we first show an example of adding a type to the type hierarchy and then an example of specifying an operator.

In Figure 4 the user has selected “Type Hierarchy” from the main menu to bring up the type hierarchy window. This window allows the user to select a type in order to add a subtype. The user has chosen the type “Machine-Tool” to bring up a window in which a subtype can be specified. At any time, a graphical view of the current type hierarchy can be displayed, and this is shown to the right of the

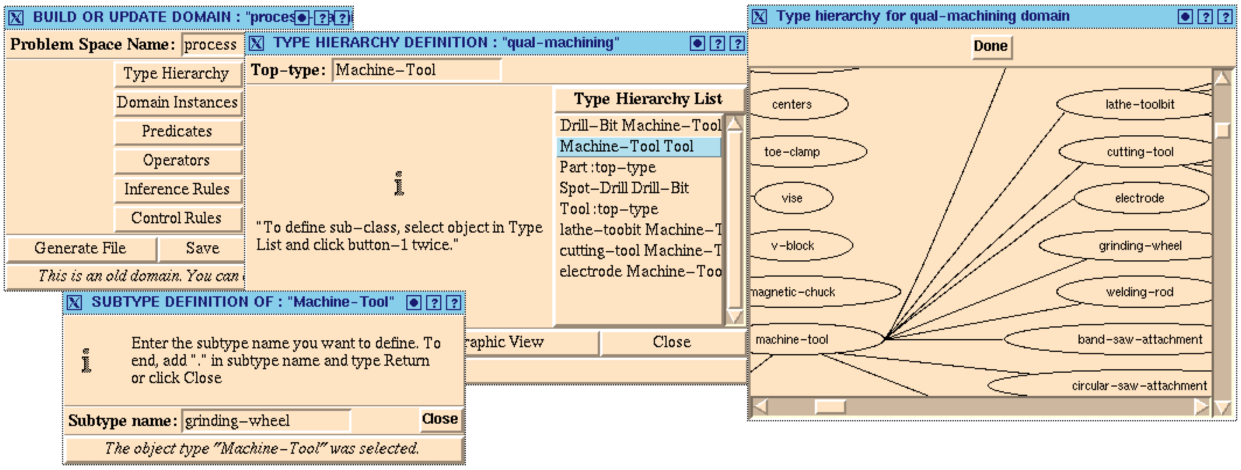

Figure 4: Several windows to support the user defining the object type hierarchy.

figure.

Once the types are created, the user can specify operators. Figure 5 shows the operator definition window. The left side consists of three sections, in which the user can specify the operator's parameters, preconditions and effects. The right side shows the operator as currently entered. The user can click on any of the fields in this operator definition when additional editing is needed. The figure shows the operator "Drill-Hole-Drill-Press", from the process planning domain defined in [Gil, 1991].

The tool supports the full operator syntax of PRODIGY [Carbonell *et al.*, 1992], including constraint functions for variables, typed first-order logic expressions for preconditions including universal quantification, and conditional effects. Figure 5 illustrates the selection of a function to constrain the variable *drill-bit-diameter*. If a variable has a simple type, the tool checks that the type has been defined. While the operator is being developed, the tool checks the types, variables and predicates being used. For example, if a variable is specified in the preconditions that was not previously defined, the tool will warn the user (but will allow new variables to be entered freely). This helps to detect many typing errors.

Building a planning domain is a difficult task, and this tool is a first step, that shields the user from the raw domain syntax and makes it easier to build a domain incrementally.

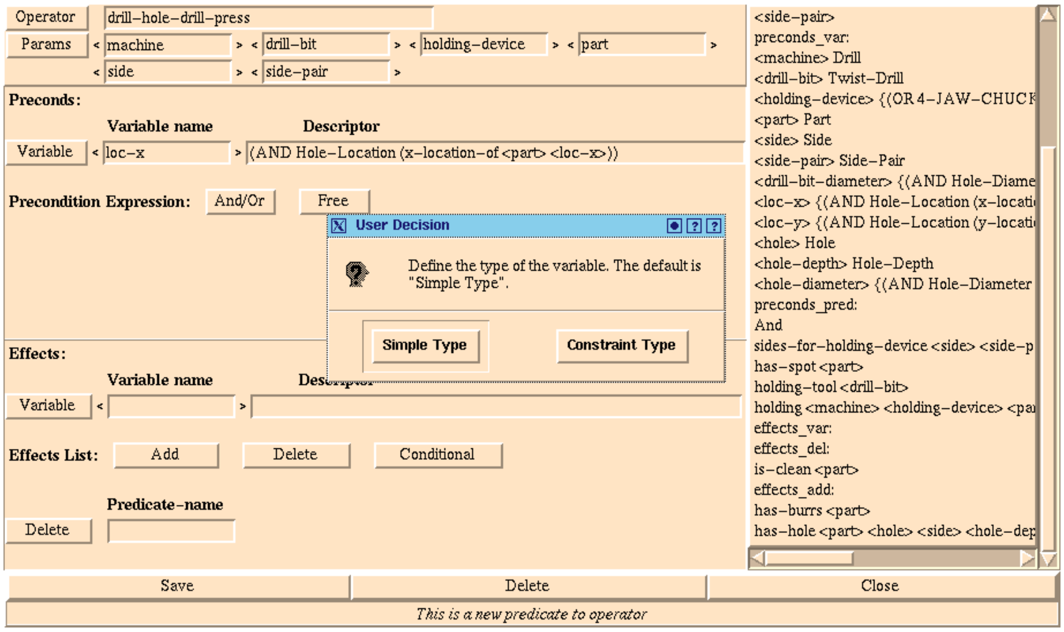

Figure 5: The Operator Definition Window

4 Additions to the User Interface

The PRODIGY user interface is easily extensible, and in this section we briefly describe two optional parts of the UI that provide additional functionality. The first provides an intuitive interface to PRODIGY's analogy component, allowing the user to select cases and watch how they are followed as PRODIGY builds a new plan. The second deals with PRODIGY's probabilistic reasoning capabilities, allowing the user to inspect a Bayesian net that summarises the plan's probability of succeeding in an uncertain environment. A third that is worthy of mention is a facility for domain-dependent graphics, whereby graphics code can be dynamically loaded in the UI along with a planning domain and used to display the current state while stepping through plan creation. The user interface provides a window in which the current state can be viewed, and a default tcl function that lists the true literals in the current state. The domain-dependent file overrides that function with routines that read from the state and draw in the window.

4.1 The interface for planning by analogical reasoning

PRODIGY can plan by analogical reasoning following the derivational analogy approach [Carbonell, 1986]. PRODIGY/ANALOGY is the module in PRODIGYthat retrieves and replays planning episodes from a library of planning cases also accumulated and maintained by the system [Veloso, 1994]. This running mode can be selected from the interface. The user can solve problems and request that the planning episodes be stored. A new problem can then be solved by replaying and and merging possibly multiple past planning cases. Figure 6 shows a snapshot of the interface where one case is instantiated to guide two different goals.

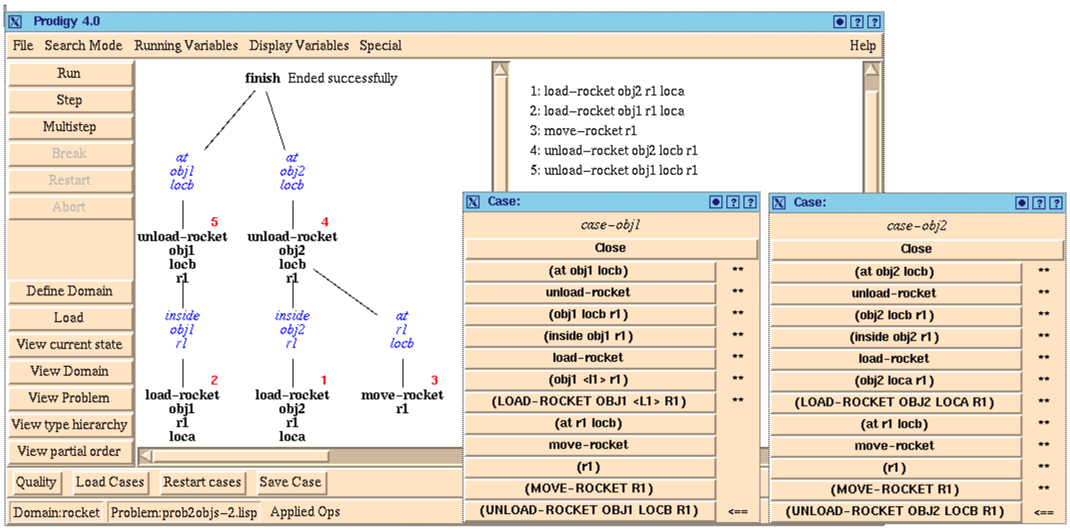

Figure 6: A snapshot of the setup used by the analogical reasoning module. Guiding cases are displayed as shown in the two windows at the right. The steps reused are marked while the steps not needed in the new situation are skipped.

The user can visualize the merging procedure, as it interleaves the multiple cases, marks the steps that are used after successful validation, and skips the ones that are no longer necessary or are invalid.

4.2 The probabilistic planner's user interface

One extension of PRODIGY, discussed in [Blythe, 1995], reasons probabilistically about plans using Bayesian nets and Markov processes. Figure 7 shows part of

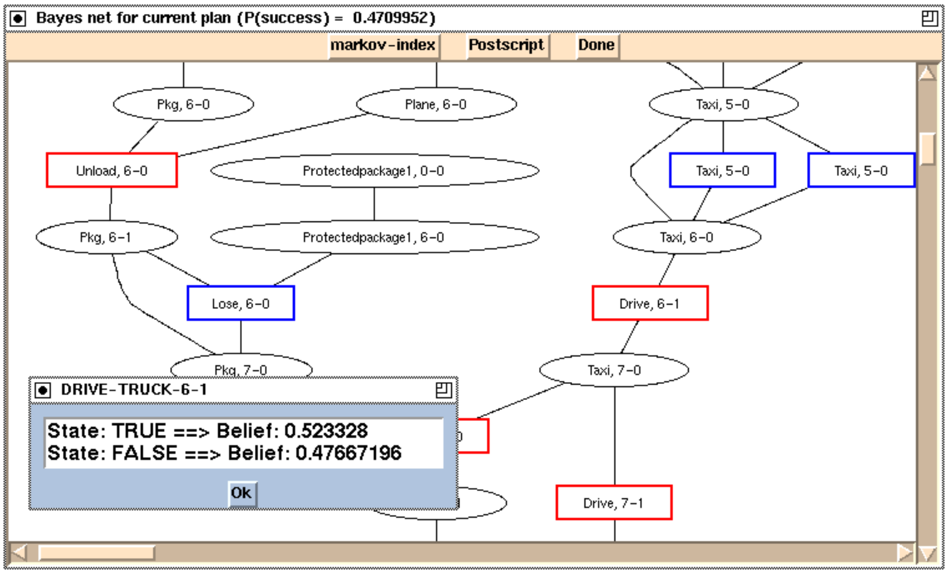

Figure 7: A window showing a fragment of the Bayesian net corresponding to a plan in a logistics domain with uncertainty. Square nodes are actions or events and round nodes are time-stamped propositions about the domain. The window in the lower left of the figure shows the probability distribution for a node.

a Bayesian net used to reason about the probability of success in a plan. Nodes represent actions (light squares), events (dark squares) or time-stamped literals (ellipses). The user can inspect the net by clicking on a node, displaying the probability distribution of the associated random variable as shown in the lower left part of the figure.

5 Conclusions

We report on the user interface that we developed for the PRODIGY architecture. Planning is a complex process and developing user planning interfaces is an important contribution for making implemented systems available to researchers, students, and practioners. The Prodigy user interface is designed as a modular interface built using Tcl/Tk which provides a user-friendly interface and makes it easy to incrementally add new features and modules. The user interface has been in use for almost one year and has already proven very useful both for research and educational purposes.

Acknowledgements

The distributed architecture for the user interface was based on a scheme built by Sean Slattery. Karen Haigh wrote much of the Tcl/Tk code for the look and feel of the interface.

References

[Blythe, 1995] Jim Blythe. Planning under uncertainty. Technical report, School of Computer Science, Carnegie Mellon University, 1995.

[Carbonell *et al.*, 1992] Jaime G. Carbonell, Jim Blythe, Oren Etzioni, Yolanda Gil, Robert Joseph, Dan Kahn, Craig Knoblock, Steven Minton, Alicia Pérez, Scott Reilly, Manuela Veloso, and Xuemei Wang. PRODIGY4.0: The manual and tutorial. Technical Report CMU-CS-92-150, Department of Computer Science, Carnegie Mellon University, June 1992.

[Carbonell, 1986] Jaime G. Carbonell. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning, An Artificial Intelligence Approach, Volume II*, pages 371–392. Morgan Kaufman, 1986.

[Fink and Veloso, 1995] Eugene Fink and Manuela Veloso. Formalizing the PRODIGY planning algorithm. In *Proceedings of the European Workshop on Planning*, September 1995. An earlier extended version is available as technical report CMU-CS-94-123, 1994.

[Gansner *et al.*, 1988] E. R. Gansner, S. C. North, and K. P. Vo. Dag - a program to draw directed graphs. *Software Practice and Experience*, 17(1):1047–1062, 1988.

[Gil, 1991] Yolanda Gil. A specification of process planning for PRODIGY. Technical Report CMU-CS-91-179, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, August 1991.

[Joseph, 1992] Robert L. Joseph. *Graphical Knowledge Acquisition for Visually-Oriented Planning Domains*. PhD thesis, School of Computer Science, Carnegie Mellon University, August 1992. Available as technical report CMU-CS-92-188.

[Paulokat and Wess, 1994] Juergen Paulokat and Stefan Wess. Planning for machining workpieces with a partial-order, nonlinear planner. In *Working notes of the AAAI Fall Symposium on Planning and Learning: On to Real Applications*, November 1994.

[Stone *et al.*, 1994] Peter Stone, Manuela Veloso, and Jim Blythe. The need for different domain-independent heuristics. In *Proceedings of the Second International Conference on AI Planning Systems*, pages 164–169, June 1994.

[Tate and Drabble, 1995] Austin Tate and Brian Drabble. O-plan's planworld viewers. In *Proceedings of the Fourteenth UK Special Interest Group on Planning and Scheduling*, Wivenhoe House Conference Centre, Essex University, November 1995.

[Veloso *et al.*, 1995] Manuela Veloso, Jaime Carbonell, M. Alicia Pérez, Daniel Borrajo, Eugene Fink, and Jim Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1):81–120, 1995.

[Veloso, 1994] Manuela M. Veloso. *Planning and Learning by Analogical Reasoning*. Springer Verlag, December 1994.

[Wilkins *et al.*, 1995] David E. Wilkins, Karen L. Myers, John D. Lowrance, and Leonard P. Wesley. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical AI*, 7(1):197–227, 1995.