Prodigy 4.0

System for Planning

Means-End Analysis

The Set of Legal Actions
Operators

Example
BlocksWorld
A Simulation given:
A Simple Robot Arm
A Set of Blocks

Domain

Operators:

PICK-UP
Picks up a Block From the Table
PUT-DOWN
Puts down a Block on the table that the robot was holding
STACK
Put a block that the robot is holding on another block
UNSTACK
Picks up a block that is on top of another block

Inference Rule
Not an operator
Infers situations from the given situation
(For example, infering that the *arm-empty*
because it is not holding any blocks)

Operator

Precondition Expression
This expression must be satisfied in order to perform the operation

Effects List
This describes how the environment is changed as a result of the operation

Objects
Each object has a specific type
These are the variables which are operated on

STACK

```
(OPERATOR
 STACK
 (params <ob> <underob>)
 (preconds
   ((<ob> Object)
    (<underob> (and OBJECT (diff <ob> <underob>))))
    (and (clear <underob>)
         (holding <ob>)))
 (effects
      ()
      ((del (holding <ob>))
       (del (clear <underob>))
       (add (arm-empty))
       (add (clear <ob> ))
       (add (on <ob> <underob>)))))
```

STACK

STACK
operates on two objects
<ob> and <underob>

Preconditions
Both are OBJECTS
They are different
<underob> has nothing on top of it
The robot is holding <ob>

effects
The robot is no longer holding <ob>
<underob> is no longer clear on top
The robot is not holding anything
<ob> has nothing on top of it
<ob> is on top of <underob>

```
(OPERATOR
 UNSTACK
 (params <ob> <underob>)
 (preconds
  ((<ob> Object)
   (<underob> Object))
  (and (on <ob> <underob>)
       (clear <ob>)
       (arm-empty)))
 (effects
        ()
        ((del (on <ob> <underob>))
         (del (clear <ob>))
         (del (arm-empty))
         (add (holding <ob>))
         (add (clear <underob>)))))
```

```
(OPERATOR
  PICK-UP
  (params <ob1>)
  (preconds
   ((<ob1> OBJECT))
   (and (clear <ob1>)
        (on-table <ob1>)
        (arm-empty)))
  (effects
      () ; no vars need genenerated in effects list
      ((del (on-table <ob1>))
       (del (clear <ob1>))
       (del (arm-empty))
       (add (holding <ob1>))))))

(OPERATOR
  PUT-DOWN
  (params <ob>)
  (preconds
   ((<ob> OBJECT))
    (holding <ob>))
  (effects
        ()
        ((del (holding <ob>))
         (add (clear <ob>))
         (add (arm-empty))
         (add (on-table <ob>)))))
```

Functions

```
(defun diff (x y) (not (eq x y)))
```

LISP format

Yield
t for true
nil for false

Specification of Problem

LISP function
create-problem

Name The name of the problem

Objects
The objects to be manipulated during the problem
(These are the variables to be instantiated)

State
This is the initial conditions

iGoal
This is the state one is trying to accomplish

```
(setf (current-problem)
      (create-problem
       (name problem-1)
       (objects
        (A B C object))
       (state
        (and (holding A)
             (on-table B)
             (clear B)
             (on-table C)
             (clear C)))
       (igoal (arm-empty))))
```

Name

problem-1

Objects

A B C each of type OBJECT

Initial State

The robot is *holding* A
Object B is on the table
Object B has nothing on top of it
Object C is on the table
Object C has nothing on top of it

Goal

To put down A so the robot is not holding anything

Control

Given a Goal Predicate
the planner selects one operator that adds that goal to the world
relevant
The given operator is *relevant* to the goal

If
the preconditions of the operator
are true in the state
Then
The operator is applied

Applying
an operator changes the state of the world within the problem

Subgoals
If not all of the preconditions of the operator are true
Then they become **subgoals**

Pending
A goal is *pending* if it is a precondition of the chosen operation is not true in the state

Search

A search tree is expanded by the following series

Compute the set of
pending goals
and
the possibly applicable operators

Decide between

To apply the operator

Choose a goal
Choose an operator to achieve that goal
choose bindings of the variables within the goal

Types of Nodes

Goal Nodes
Holds the goal node at that point

Operator (or inference rule) Nodes
Always a child of a goal node
Holds a non-instantiated operator

Binding Nodes
Child of an operator node
Holds instantiations of that operator

Applied Operator Nodes
Child of either a Binding node or another Applied operator node

Choosing a Node

Chooses those nodes which have not been fully expanded

Default:
Depth First
The last node created will be the first node expanded (Breadth First can also be chosen)

Control Rules
Rules by which the order can be altered

The order in which the nodes are expanded
corresponds to normal decision making
$Goal \rightarrow Operator \rightarrow Bindings \rightarrow$
either one or more operator applications or a goal

Choosing a Goal

Candidate Goals
Unmatched Preconditions of instantiated operators

The solver focuses on one goal at a time
(even though all must be eventually satisfied)

Change of Focus
(caused by a control rule)
Under a given set of conditions
can try to solve another goal before the original is satisfied

Non-linear Planning
Interleaving of goals

Choosing an Operator

Default:
All operators which are relevant to the goal (both operators and inference rules)

The operator are considered in the order they are stated

Control Rules

The planner must make decisions as to which operator to use and which subgoal to pursue

Purpose

Increase Efficiency
Down the correct path for fast solution

Improve Quality of Solutions
More than one solution
Can direct to a particular type of solution

Syntax

SELECT
To select a set of candidates
(nodes, goals or operators)

REJECT
To reject or filter out candidate decisions

PREFER
To choose the best alternative from a set of candidates

Meta-Predicates

(Candidate Node <node>)
(Newest-Candidate-Node <node>)
(Oldest-Candidate-Node <node>)
(Expanded-Goal <goal>)
(On-Goal-Stack <goal>)
(Current-Goal <goal>)
(Candidate-Operator <op>)
(Current-Operator <op>)
(Current-Ops <ops>)
(True-in-state <expr>)
(Applicable-Operator <inst-op>)
(Type-Of-Object <obj> <type>)

Prefer

```
(Control-rule Prefer-right-goal
              (IF (and (candidate-goal (on <x> <y>))
                       (candidate-goal (on <y> <z>))
                       (true-in-state (clear <x>))))
              (THEN prefer goal (on <y> <z>) (on <x> <y>)))

(control-rule prefer-bindings-unstack-clear
              (IF (and (current-goal (clear <y>))
                       (current-ops (unstack))
                       (true-in-state (on <x> <y>))))
              (THEN prefer bindings
                    ((<ob> . <x>) (<underob> . <y>))
                    ((<ob> . <anything>) (<underob> . <anyth
```

Select

```
(control-rule DEPTH-FIRST-SEARCH
              (IF (newest-candidate-node <node>))
              (THEN select node <node>))

(CONTROL-RULE SELECT-BINDINGS-UNSTACK-CLEAR
              (IF (and (current-goal (clear <y>))
                       (current-ops (UNSTACK))
                       (true-in-state (on <x> <y>))))
              (THEN select bindings ((<ob> . <x>) (<underob>
```

Reject

```
(CONTROL-RULE REJECT-WRONG-GOAL
              (IF (and (candidate-goal (on <x> <y>))
                       (candidate-goal (on <y> <z>))
                       (true-in-state (clear <x>))))
              (THEN reject goal (on <x> <y>)))
```

Sub-Goal

```
(Control-Rule avoid-apply-for-wrong-goal
              (IF (and (on-goal-stack (on <x> <y>))
                       (candidate-goal (on <y> <z>))
                       (applicable-operator (pick-up <x>))))
              (THEN sub-goal))
```