15-381 Problem Set 4: Planning

Out: Tuesday, October 17, 2000
Due: Tuesday, October 31, 2000

This time we ask you to form groups of up to two people. Send email to Rune (runej@cs) to form a group. All groups must be made by Friday Oct. 20. As usual group directories will be created in `/afs/andrew.cmu.edu/scs/cs/15-381/handin`.

Part I: GPS and POP Implementation

You are to implement a version of the General Problem Solver (GPS) [Newell and Simon 60] and the Partial Order Planner (POP). GPS is explained in the lecture notes of Sep. 28 and the POP algorithm is described on page 355-356 in the text book.

The input to your two programs should be two text files containing a STRIPS domain and problem description. The STRIPS language is described on page 343-345 in the text book. Note that STRIPS operators only allow a conjunction of atoms as precondition and a conjunction of literals as effect. Both files are written in the PDDL language which is a superset of STRIPS. A description of PDDL is found in `/afs/cs/academic/class/15381-f00/public/hw4/pddl-subset.ps`. We require that your programs also take a numeric input stating a search depth limit or the max number of time steps or search nodes allowed.

The output from your program should be either 1) The plan solving the problem given as a sequence of plan steps, or 2) 'Failure' if the planner does not succeed to solve the problem.

To simplify your task we provide functions for parsing a PDDL domain and problem description into a propositional version with no variables. In the propositional version a complete set of grounded actions is obtained by instantiating each STRIPS operator with all possible combinations of objects in the problem. To make this more clear consider the Blocks-World domain and problem description shown below:

domain.pddl

```
(define (domain BLOCKS)
  (:requirements :strips)
  (:predicates (on ?x ?y)
```

```
      (ontable ?x)
      (clear ?x)
      (handempty)
      (holding ?x))

  (:action pick-up
     :parameters (?x)
     :precondition (and (clear ?x) (ontable ?x) (handempty))
     :effect
     (and (not (ontable ?x))
    (not (clear ?x))
    (not (handempty))
    (holding ?x)))

  (:action put-down
     :parameters (?x)
     :precondition (holding ?x)
     :effect
     (and (not (holding ?x))
    (clear ?x)
    (handempty)
    (ontable ?x)))

  (:action stack
     :parameters (?x ?y)
     :precondition (and (holding ?x) (clear ?y))
     :effect
     (and (not (holding ?x))
    (not (clear ?y))
    (clear ?x)
    (handempty)
    (on ?x ?y)))

  (:action unstack
     :parameters (?x ?y)
     :precondition (and (on ?x ?y) (clear ?x) (handempty))
     :effect
     (and (holding ?x)
    (clear ?y)
    (not (clear ?x))
    (not (handempty))
    (not (on ?x ?y)))))
```

prob.pddl

```
(define (problem BLOCKS-2)
(:domain BLOCKS)
(:objects A B)
```

```
(:INIT (CLEAR A) (ON A B) (ONTABLE B) (HANDEMPTY))
(:goal (CLEAR B)))
```

When instantiating all the operators for this problem we get the set of grounded actions: { pick-up(A), pick-up(B), put-down(A), put-down(B), stack(A,A), stack(A,B), stack(B,B), stack(B,A), unstack(A,A), unstack(A,B), unstack(B,B), unstack(B,A)}. The predicates in the precondition and effect of these actions are fully grounded (no free variables). Thus, when given the initial and goal state as a set of grounded predicates, we can regard the planning problem as stated in propositional logic since a grounded predicate corresponds to a proposition.

The parser and translator is written C/C++ and returns an internal propositional representation that should be used as input to your POP and GPS algorithms (the files are in `/afs/cs/academic/class/15381-f00/public/hw4/translator`).

For each program we ask you to:

1. Write a *one page* specification of the major data structures and the planning algorithm. Either explain the algorithm in free text or use a pseudo-code language of your choice. It should be clear which kind of search the planning algorithm is performing and where and when it backtracks.

2. Show the search tree performed by your planning algorithm for the small Blocks-World example above.

3. Implement the program in C/C++ using the parser and translator we provide. It is required that you make comments in your code that makes clear the connection between the code and the specification.

Part II: Comparison Analysis

1. Run GPS and POP on each of the problems of the Rocket, Blocksworld and Gripper domain in `/afs/cs/academic/class/15381-f00/public/hw4/domains/`. Make three subdirectories in your hand in directory called rocket, blocksworld and gripper and store a file with the result for each problem using the following naming convention: `sol_i.gps` and `sol_i.pop` for the results of problem i for GPS and POP, respectively.

2. For each domain make graphs showing the solution length of POP and GPS as a function of the problem number.

3. For each domain explain in up to three lines the result of the planners.

Extra Credit

1. Write a domain in which POP does much better in terms of CPU time than GPS or vice versa. You can e.g. use the unix command 'time' to measure the CPU time of your programs.

2. Down-load IPP Ver. 4.1 (`/afs/cs/academic/class/15381-f00/public/hw4/planners/IPPcode4.1.tar.gz`). IPP is an optimized GraphPlan planner by J. Koehler, B. Nebel and J. Hoffmann (1997). It is written in C and solves PDDL planning problems. Unpack the source files and compile them with `make`. Run the planner on a problem by executing `ipp -o <domain>.pddl -f <problem>.pddl`. Add the results obtained with IPP to the plan length graphs of GPS and POP. Explain the resulting graphs.

Submission Instructions

After completing the assignment, place one copy of your source code and executables in

`/afs/andrew/scs/cs/15-381/handin/`*userid1_userid2*

Please create a separate directory for each planner. Also place a copy of any domain and problem files you may have created and the output files for the test domains. Submission directories for each group will be made on your request, but no later than Friday Oct. 20. They will be closed at 11:59pm on October 31.

Please contact either Laurie (hyaku@cs) or Rune (runej@cs) if you have any questions.