CS 5600/6600/7890: Intelligent Systems
Assignment 10
Knowledge Engineering for General Problem Solver

Vladimir Kulyukin
Department of Computer Science
Utah State University

November 10, 2018

Learning Objectives

1. General Problem Solver (GPS)
2. Writing GPS Operators
3. Knowledge Engineering

Introduction

In this assignment, you will solve a famous planning problem with the General Problem Solver (GPS) system. GPS was a major benchmark of AI planning. Its knowledge engineering techniques (i.e., designing problem-specific operators) and problem solving techniques (e.g., means-ends analysis) influenced and still keep influencing AI planning systems. This assignment will give you some experience of designing problem-specific operators.

Problem 1

The zip archive of this assignment has two files – `auxfuns.lisp` and `gps.lisp`. The file `auxfuns.lisp` contains some auxiliary functions and the file `gps.lisp` is an Lisp implementation of the GPS planner.

Loading GPS

Change your directory to where you saved `gps.lisp` and `auxfuns.lisp`, fire up the Lisp VM, and load `gps.lisp`.

```
> (load "gps.lisp")
;; Loading file gps.lisp ...
;;  Loading file /home/vladimir/teaching/AI/hw/hw10/auxfuns.lisp ...
;;  Loaded file /home/vladimir/teaching/AI/hw/hw10/auxfuns.lisp
;; Loaded file gps.lisp
T
```

If you're on Windows and using Allegro Common Lisp (ACL) IDE, start it and go to the listener window. The listener window should say "Listener 1" in its top left corner. Remember that you need to change the package from `CG-USER` to `CL-USER` by typing into the prompt the following command.

```
CG-USER(1): :package :user
CL-USER(2):
```

Now load gps.lisp as you did ELIZA, SAM, and CA in the previous assignments.

In gps.lisp, the variable *school-ops* is a list of operators we developed and analyzed in class to solve the son-at-school problem.

```
(defparameter *school-ops*
  (list
    (make-op :action 'drive-son-to-school
         :preconds '(son-at-home car-works)
         :add-list '(son-at-school)
         :del-list '(son-at-home))
    (make-op :action 'shop-installs-battery
         :preconds '(car-needs-battery shop-knows-problem
                     shop-has-money)
         :add-list '(car-works))
    (make-op :action 'tell-shop-problem
         :preconds '(in-communication-with-shop)
         :add-list '(shop-knows-problem))
    (make-op :action 'telephone-shop
         :preconds '(know-phone-number)
         :add-list '(in-communication-with-shop))
    (make-op :action 'look-up-number
         :preconds '(have-phone-book)
         :add-list '(know-phone-number))
    (make-op :action 'give-shop-money
         :preconds '(have-money)
         :add-list '(shop-has-money)
         :del-list '(have-money))))
```

Let's use these operators to solve the son-at-school problem with GPS. We start by defining the initial state of the world for the agent and the agent's goal.

```
> (setf world-state-1 '(son-at-home car-needs-battery have-money
                        have-phone-book))
(SON-AT-HOME CAR-NEEDS-BATTERY HAVE-MONEY HAVE-PHONE-BOOK)
> (setf goal-1 '(son-at-school))
(SON-AT-SCHOOL)
```

Now let's apply GPS to this problem by first telling the system to use *school-ops* and then applying the system to the initial state of the world and the goal.

```
> (use *school-ops*)
6
> (gps world-state-1 goal-1)
((START) (EXECUTE LOOK-UP-NUMBER) (EXECUTE TELEPHONE-SHOP)
 (EXECUTE TELL-SHOP-PROBLEM) (EXECUTE GIVE-SHOP-MONEY)
 (EXECUTE SHOP-INSTALLS-BATTERY) (EXECUTE DRIVE-SON-TO-SCHOOL))
```

The first call (use *school-ops*) returns the number of operators in the list of operators the system is asked to use. As you can see from the second call, GPS returns a plan for the agent to follow. The plan consists of looking up the auto shop's number, phoning the shop, telling the shop about the battery problem, giving the shop the money, having the shop install the battery, and driving the son to school. If you need to save the plan, you can do it by saving it in a variable.

```
> (setf son-at-school-plan (gps world-state-1 goal-1))
((START) (EXECUTE LOOK-UP-NUMBER) (EXECUTE TELEPHONE-SHOP)
 (EXECUTE TELL-SHOP-PROBLEM) (EXECUTE GIVE-SHOP-MONEY)
 (EXECUTE SHOP-INSTALLS-BATTERY) (EXECUTE DRIVE-SON-TO-SCHOOL))
> son-at-school-plan
((START) (EXECUTE LOOK-UP-NUMBER) (EXECUTE TELEPHONE-SHOP)
 (EXECUTE TELL-SHOP-PROBLEM) (EXECUTE GIVE-SHOP-MONEY)
 (EXECUTE SHOP-INSTALLS-BATTERY) (EXECUTE DRIVE-SON-TO-SCHOOL))
```

If you want to trace how GPS solves the problem step by step, you can call the function `trace-gps` and then calling the function `gps` on the world's state and the goal. As you can see below, GPS works by recursively satisfying the preconditions of each operator so that it can be applied to reduce the differences between the current state of the world and the desired state of the world.

```
> (trace-gps)
(:GPS)
> (gps world-state-1 goal-1)
Goal: SON-AT-SCHOOL
Consider: DRIVE-SON-TO-SCHOOL
  Goal: SON-AT-HOME
  Goal: CAR-WORKS
  Consider: SHOP-INSTALLS-BATTERY
    Goal: CAR-NEEDS-BATTERY
    Goal: SHOP-KNOWS-PROBLEM
    Consider: TELL-SHOP-PROBLEM
      Goal: IN-COMMUNICATION-WITH-SHOP
      Consider: TELEPHONE-SHOP
        Goal: KNOW-PHONE-NUMBER
        Consider: LOOK-UP-NUMBER
          Goal: HAVE-PHONE-BOOK
        Action: LOOK-UP-NUMBER
      Action: TELEPHONE-SHOP
    Action: TELL-SHOP-PROBLEM
    Goal: SHOP-HAS-MONEY
    Consider: GIVE-SHOP-MONEY
      Goal: HAVE-MONEY
    Action: GIVE-SHOP-MONEY
  Action: SHOP-INSTALLS-BATTERY
Action: DRIVE-SON-TO-SCHOOL
((START) (EXECUTE LOOK-UP-NUMBER) (EXECUTE TELEPHONE-SHOP)
 (EXECUTE TELL-SHOP-PROBLEM) (EXECUTE GIVE-SHOP-MONEY)
 (EXECUTE SHOP-INSTALLS-BATTERY) (EXECUTE DRIVE-SON-TO-SCHOOL))
```

Monkey and Bananas

The monkey and bananas problem is a classic problem, as classic as the Sussman Anomaly, that many AI planners solve as they are being developed. Imagine the following situation. A hungry monkey is standing at the doorway to a room. In the middle of the room there is a bunch of bananas suspended from the ceiling by a rope, well out of the monkey's reach. There is a chair near the door, which the monkey can push. The chair is tall enough for the monkey to get the bananas after he climbs on it.

Develop a set of operators for the monkey agent to solve quench his hunger and save them in the variable `*banana-ops*` in `gps.lisp`. Below is one possible solution to the problem. Of course, your solution may be different in that your knowledge representation may be different.

```
> (use *banana-ops*)
6
> (setf world-state-2 '(at-door on-floor has-ball hungry chair-at-door))
(AT-DOOR ON-FLOOR HAS-BALL HUNGRY CHAIR-AT-DOOR)
> (setf goal-2 '(not-hungry))
(NOT-HUNGRY)
> (gps world-state-2 goal-2)
Goal: NOT-HUNGRY
Consider: EAT-BANANAS
  Goal: HAS-BANANAS
  Consider: GRASP-BANANAS
    Goal: AT-BANANAS
    Consider: CLIMB-ON-CHAIR
      Goal: CHAIR-AT-MIDDLE-ROOM
      Consider: PUSH-CHAIR-FROM-DOOR-TO-MIDDLE-ROOM
        Goal: CHAIR-AT-DOOR
        Goal: AT-DOOR
      Action: PUSH-CHAIR-FROM-DOOR-TO-MIDDLE-ROOM
      Goal: AT-MIDDLE-ROOM
      Goal: ON-FLOOR
    Action: CLIMB-ON-CHAIR
    Goal: EMPTY-HANDED
    Consider: DROP-BALL
      Goal: HAS-BALL
    Action: DROP-BALL
  Action: GRASP-BANANAS
Action: EAT-BANANAS
((START) (EXECUTE PUSH-CHAIR-FROM-DOOR-TO-MIDDLE-ROOM)
 (EXECUTE CLIMB-ON-CHAIR) (EXECUTE DROP-BALL)
 (EXECUTE GRASP-BANANAS) (EXECUTE EAT-BANANAS))
```

What to Submit

Save your operators in `*banana-ops*` and submit `gps.lisp` through Canvas.

Happy Hacking and Knowledge Engineering!