

## Homework for Chapters 9 and 11, CMSC 421, Fall 2008

- There are 5 problems worth 10 points each, for a total of 50 points.

**Chapter 9:** Do problems 9.12 and 9.19 from Russell & Norvig.

**Planning homework:** Consider the following *coffee-making domain*. This is a planning domain in which you have a coffee maker and several different kinds of coffee beans, and the coffee maker can be used to make pots of coffee. There are three operators: load coffee into the machine, start the machine running, and end the machine's coffee-making cycle. Here is a classical representation of them:

load(x)

precond: coffee(x), loaded(nil)

effects: loaded(x), ¬loaded(nil)

brew(x)

precond: loaded(x), ¬loaded(nil), ¬loaded(waste)

effects: ¬loaded(x), loaded(waste), pot(x)

unload(x)

precond: loaded(x), ¬loaded(nil)

effects: ¬loaded(x), loaded(nil)

Most of the questions in this homework assignment involve the following planning problem in the coffee-making domain. We will call this problem  $P_1$ . In  $P_1$ , there are two kinds of coffee: *caf* and *decaf*. The goal is to make one pot of each. Here are the initial state and the goal formula:

**Initial state:** {coffee(caf), coffee(decaf), loaded(nil)}

**Goal formula:** {pot(caf), pot(decaf)}

**Problem 1** Suppose we do a backward state-space search on  $P_1$ , without any loop-checking.

- How many execution traces lead to solutions?
- A solution is *shortest* if there is no other solution that contains fewer actions. What are all of the shortest solutions?
- How many execution traces don't lead to solutions? If any of these execution traces is finite, then give an example—and if none of them is finite, then explain why not.

**Problem 2** Suppose we run Graphplan on  $P_1$ .

- Draw the planning graph after the first graph-expansion. Include all maintenance actions and mutexes. For each mutex, tell what kind of mutex it is.
- At what level does Graphplan first call Extract?

**Problem 3** To encode the coffee-making domain as a task-list planning domain, we can use the operators on the first page, and a single method:

`make( $x, y, z$ )`

Task: `pot( $x$ )`

Precond: `none`

Subtasks: `(load( $x$ ), brew( $y$ ), unload( $z$ ))`

- (a) With the domain description that's given above and the initial state that's shown on the first page, what solutions can TFD find if the initial task network is `{pot(caf),pot( $y$ )}`?
- (b) Add an additional method to enable TFD to find optimal (i.e., shortest) solutions to coffee-making problems.