02285 AI and MAS, SP19
Exercises for week 4, 26/2-19

When referring to the **POP algorithm** below we refer to the algorithm provided on today's slides and described in detail in Section 11.3 of Russell & Norvig **2nd edition** (relevant section available on CampusNet).

Exercise 1 (Partially ordered plans)

Define the action schemas for the problem of putting on shoes and socks and a hat and coat, assuming that there are no preconditions for putting on the hat and coat. Give a partial-order plan that is a solution, and show that there are 180 different linearisations of this solution. You don't necessarily have to apply the POP algorithm to find the partial-order plan in this exercise, you just have to find the plan and show it as a directed graph (like for the socks-and-shoes problems on today's slides).

Exercise 2 (POP Algorithm)

Consider again the milk-and-bananas domain with the following action schemas:

ACTION: $Buy(x, y)$ (buy item x at location y)
PRECONDITION: $Buyable(x) \land Place(y) \land At(y) \land Sells(y, x)$
EFFECT: $Have(x)$

ACTION: $Go(x, y)$
PRECONDITION: $Place(x) \land Place(y) \land At(x)$
EFFECT: $At(y) \land \neg At(x)$

Assume we are given the following initial state:

$Buyable(Milk) \land Buyable(Bananas) \land Buyable(Drill) \land Place(Home) \land Place(Netto) \land Place(ToolShop) \land At(Home) \land Sells(Netto, Milk) \land Sells(Netto, Bananas) \land Sells(ToolShop, Drill)$.

a) What are the rigid atoms in this problem (cf. the exercise session from last week)?

b) Use the POP algorithm to compute a partial-order plan for the following goal: $At(Home) \wedge Have(Milk) \wedge Have(Bananas)$. Whenever you encounter an open precondition that is a rigid atom, just make sure it holds in the initial state, without drawing an edge (to keep the graph clutter-free). If such an open precondition does *not* hold in the initial state, you are enforced to backtrack.

c) How many linearisations does you solution have? Are those linearisations optimal solutions to the problem? Will the POP algorithm necessarily find an optimal solution to this particular planning problem?

d) Now use the POP algorithm to compute a partial-order plan for the following goal: $At(Home) \wedge Have(Milk) \wedge Have(Bananas) \wedge Have(Drill)$. Note that you can solve this problem by continuing to run the POP algorithm from the graph produced in the previous question with one additional open precondition $Have(Drill)$ of the action $Finish$. You might be forced to backtrack.

Exercise 3 (Delete-Relaxation and POP)

Consider a *delete-relaxed* planning problem P^+, that is, where all negative effects are removed. This exercise concerns whether the relaxed problem P^+ is easier to solve using POP than the original problem P.

a) Show that when running the POP algorithm on P^+ a conflict can never arise.

b) Consider a POP algorithm working in a best-first manner, where a heuristics chooses the next open precondition to be resolved. Can such an algorithm ever be forced to backtrack on a delete-relaxed problem P^+?

Exercise 4 (HTN planning)

Solve exercise 11.2: "You have a number of trucks with which to deliver a set of packages. Each package starts at some location on a grid map, and has a destination somewhere else. Each truck is directly controlled by moving forward or turning. Construct a hierarchy of high-level actions for this problem (you are free to decide on the details of the problem). What knowledge about the solutions does your hierarchy encode?"

Additional question: Solve a simple instance of the problem by refinements of you high-level actions.

Exercise 5 (HTN and attached procedures in Sokoban)

Optional: In Sokoban, there are essentially two types of actions: moves and pushes. Concerning moves, it seems advantageous to introduce a high-level action for "jumping" from one cell to another reachable cell. However, such a high-level action would need a precondition for checking whether the to-cell is reachable from the from-cell by a sequence of moves (no pushes). We can not construct literals for recording such information in pure classical planning. Why not?

A possible solution is to introduce so-called *attached procedures*. These are also sometimes called *code-call conditions*, and in the F.E.A.R. paper by Jeff Orkin they are called *procedural preconditions*. The idea is to allow actions to have preconditions that need to be checked by external computations. An example would be a precondition $Reachable(x, y)$ that checks whether the cell y is reachable from cell x. This can easily be computed from the state description by a procedure external to the planner.

Construct a hierarchy of high-level actions for the Sokoban domain, possibly using attached procedures. How does your high-level actions reduce the search needed in solving Sokoban problems? What knowledge about solving Sokoban puzzles does your hierarchy encode?