Information page for written examinations at Linköping University

Examination date	2012-01-13
Room (1) If the exam is given in different rooms you have to attach an information paper for each room and mark intended place	U3
Time	8-12
Course code	TDDD48
Exam code	TEN1
Course name **Exam name**	Automatisk planering Skriftlig tentamen
Department	IDA
Number of questions in the examination	4
Teacher responsible/contact person during the exam time	Jona Kvarnström
Contact number during the exam time	0704-737579
Visit to the examination room approx.	kl. 09
Name and contact details to the course administrator (name + phone nr + mail)	Anna Grabska Eklund, ankn. 2362, anna.grabska.eklund@liu.se
Equipment permitted	inga/none
Other important information	
Which type of paper should be used, cross-ruled or lined	
Number of exams in the bag	

Exam: TDDD48 Automated Planning 2012-01-10

Please note that you may **answer in Swedish** if you prefer!
Some pages have clarifications at the end.

1 Classical Planning

Recall that in standard classical planning problems, no function symbols are allowed, goals constrain only the final state reached by a plan, and preconditions, effects and goals can be represented as simple sets (conjunctions) of positive and negative literals, without disjunction, quantification or conditional effects.

Let $P_1 = (O, s_0, g_1)$ and $P_2 = (O, s_0, g_2)$ be two classical planning problems that share the same operators and initial state. Let $\pi_1 = [a_1, \ldots, a_n]$ be a solution of length n for P_1, and let $\pi_2 = [b_1, \ldots, b_n]$ be a solution of the same length for P_2.

a) Suppose that no operator in O has negative effects. Can we then be certain that the interleaved action sequence $[a_1, b_1, a_2, b_2, \ldots, a_n, b_n]$ is a solution for $(O, s_0, g_1 \cup g_2)$? Motivate clearly why this is or is not the case. **(1 point)**

b) Let $\Pi = \{\pi \mid \pi \text{ is applicable to } s_0\}$ be the set of all action sequences that are executable starting at s_0. Is Π *always* infinite, *sometimes* infinite or *never* infinite? Explain why. **(1 point)**

c) Let $S = \{\gamma(s_0, \pi) \mid \pi \text{ is applicable to } s_0\}$ be the set of all states that are reachable through executable action sequences starting at s_0. Is S *always* infinite, *sometimes* infinite or *never* infinite? Explain why. **(1 point)**

Note that clear and comprehensible explanations and motivations are required. In some cases, examples or counterexamples may be useful as part of a motivation.

2 Planning Graphs (6 points)

We will now take a closer look at **planning graphs** using the standard **logistics domain**, where a set of packages should be transported to their destinations. This domain is defined in Appendix A. We will use a simple problem instance containing the packages $\{p_1, p_2\}$, the trucks $\{t_1, t_2\}$ and the locations $\{l_1, l_2\}$, none of which are airports.

In the initial state, we know that $\mathsf{at}(p_1, l_1)$, $\mathsf{at}(p_2, l_2)$, $\mathsf{at}(t_1, l_2)$, and $\mathsf{at}(t_2, l_2)$. All locations are in the same city: $\{\mathsf{same\text{-}city}(l, l') \mid l, l' \in \{l_1, l_2\}\}$. No packages are loaded into vehicles.

The goal is that $\mathsf{at}(p_1, l_1)$, $\mathsf{at}(p_2, l_1)$, $\mathsf{at}(t_1, l_2)$, and $\mathsf{at}(t_2, l_2)$.

You may assume a typed representation, where "nonsense" propositions such as $\mathsf{at}(t_1, t_2)$ and actions such as $\mathsf{drive\text{-}truck}(p_1, p_2, p_1)$ cannot occur. Type *predicates* such as $\mathsf{truck}(t)$ are not used by the operators defined in the appendix and therefore do not have to be present in the planning graph. You may also omit the same-city predicate from the graph, as it may otherwise be too complex to keep track of the graph "manually".

You should do the following:

a) Generate a planning graph for the problem instance defined above, containing a total of three proposition levels including the initial level. That is, you can stop expanding the graph after reaching three proposition levels, without having to continue until a fixpoint is reached.

   You must explicitly show all information that is normally part of a planning graph, including for example mutual exclusion (mutex) relations. This information may be easier to provide as a table after the actual graph! Make sure to leave sufficient room for all required arrows. **(2 points)**

b) Explain when entities in a planning graph are mutually exclusive. Also explain how mutual exclusion at one level affects which entities are present in the next level of the graph and how this in turn affects the size of the complete planning graph. Give a concrete example from the planning graph you created above, showing one specific way in which the graph would have been different if mutual exclusion had not been considered. **(1 point)**

c) Above, you have shown how a planning graph can be created and expanded. You should now explain how planners such as GraphPlan actually use this graph during planning. For example, assuming that a graph with n layers has been created, how does GraphPlan use this graph to try to find a plan? When does GraphPlan terminate (give up)? **(1 point)**

We strongly recommend that you make a quick sketch of the planning graph on a separate paper before constructing a final version, since it can be difficult to determine in advance how large the graph will become.

3 Partial-Order Planning

We now consider partial-order planning. We use the standard **logistics domain** as defined in Appendix A, with a problem instance containing the packages $\{p_1, p_2\}$, the trucks $\{t_1, t_2\}$ and the *three* locations $\{l_1, l_2, l_3\}$, none of which are airports.

In the initial state, we know that at(p_1, l_1), at(p_2, l_2), at(t_1, l_2), and at(t_2, l_2). All locations are in the same city: $\{\text{same-city}(l, l') \mid l, l' \in \{l_1, l_2, l_3\}\}$. No packages are loaded into vehicles.

The goal is that at(p_1, l_3), at(p_2, l_3), at(t_1, l_2), and at(t_2, l_2).

You should do the following:

a) Show the *initial partial plan* π_0 generated for this problem instance by a typical partial-order planner, such as the PSP planner in the course book. This is the partial plan that corresponds to the first node ("root node") generated in the search space. **(1 point)**

b) Show all the immediate successors of the initial partial plan. That is, demonstrate all the different ways in which a standard PSP-like partially ordered planner might extend π_0 in a single step.

   For this particular task you do not have to illustrate each successor plan graphically. Instead, you can explain clearly in writing how each successor would extend π_0. However, you still need to indicate *all* changes that would be made, including new constraints and relations! **(1 point)**

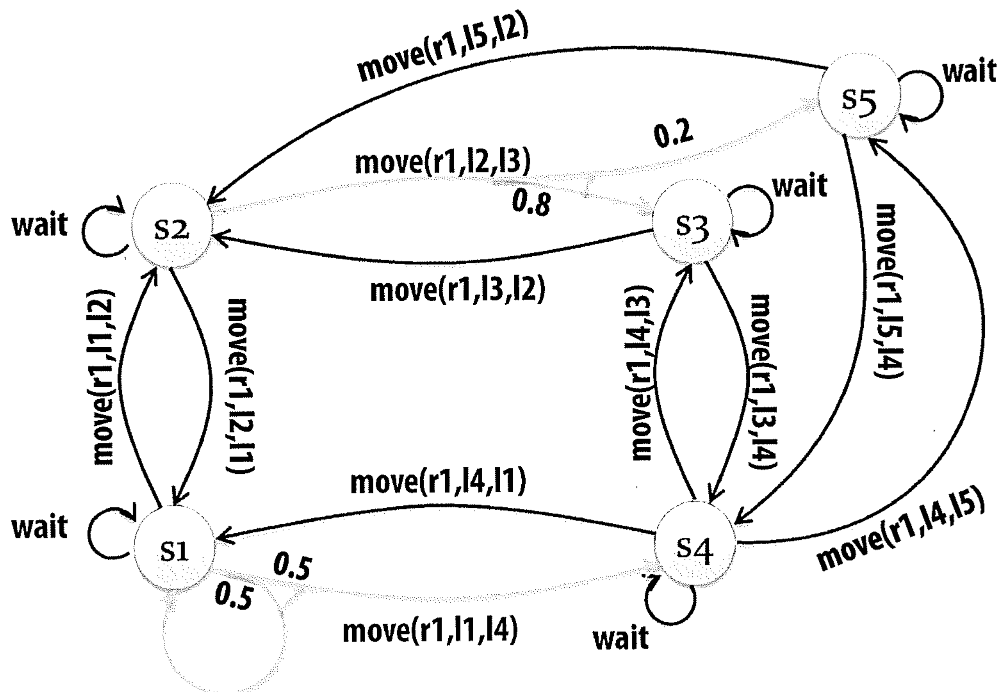
c) Show a partial plan for this problem in which one or more threats appear. Indicate all threats clearly. (Note that you might be able to extend this plan into a solution below.) **(1 point)**

d) Show a complete solution plan for this problem instance, making good use of both trucks to efficiently resolve all goals. Make sure that actions are not temporally constrained relative to each other unless this is necessary in order to achieve the goal. **(1 point)**

Note: For each action in a partially ordered plan, you must clearly show each precondition *above* the action and each *effect* below the action. You must also indicate all other relevant structural features in the plan: precedence constraints (solid arrows), causal links (dashed arrows) and threats.

4 Planning with Markov Decision Processes

The following example of a stochastic process has been used during the lectures:



a) Suppose that our objective is to visit states s2 and s4 *repeatedly* (over and over again). For example, we might be generating a policy for a robot that wants to see regularly what happens at each of those two locations.

   Specify costs (for each action) and rewards (for each state) that ensure that this will happen regardless of which state we start in. Assume a discount factor of 0.9. **(1 point)**

b) Begin with an initial policy π_0 such that for any state s, $\pi_0(s) = \text{wait}$. Then perform two full steps of *policy iteration* given your own costs and rewards, creating the policies π_1 and π_2. Show clearly how each step in the policy iteration is calculated, not just the final result. **(2 points)**

We suggest that you abbreviate the move actions so that move(r1,l5,l2) becomes m52, etc.

A The Logistics Domain

The standard **logistics domain** contains a set of *packages* that should be transported to their destinations. A package can be transported by *truck* between any two *locations* in the same city, and by *airplane* between special *airport* locations in different cities. Since trucks cannot deliver packages directly to other cities, and airplanes cannot visit arbitrary locations, delivering a package might require using a truck to move it to an airport, using an airplane to move it to another city and then once again using a truck to get the package to its final destination.

We assume the following types of objects:

- thing, with subtypes package and vehicle
- vehicle, with subtypes truck and airplane
- location, with subtype airport

A model of this domain may include the following **operators**. For Simple Task Network planning, these operators correspond directly to **primitive tasks**.

- load-truck(package, truck, location) loads a package into a truck, given that they are both at the same location. The truck can hold an arbitrary number of packages.
- unload-truck(package, truck, location) unloads a package from a truck in the current location.
- load-plane(package, airplane, location) loads a package into a plane, given that they are both at the same location. The plane can hold an arbitrary number of packages.
- unload-plane(package, airplane, location) unloads a package from an airplane in the current location.
- drive-truck(truck, location, location) drives a truck between locations in the same city.
- fly-plane(plane, airport, airport) flies a plane between two airports in different cities.

We assume the following **predicates** are available:

- at(thing,loc) – the package or vehicle thing is at the location loc. Note that a package "at" a certain location cannot be "in" a vehicle in the same state.
- in(x,vehicle) – the package x is in the given vehicle.
- same-city(loc1,loc2) – the given locations are in the same city. Note that a location must be in the same location as itself.

We assume a **typed** domain model where each operator parameter and predicate parameter is given a specific type and cannot take on values outside that type. For example, if t_1 is a truck, an expression such as at(t_1, t_1) is not merely false but incorrect. Nevertheless, we provide the following **type predicates** that may be useful in some situations: thing(x), package(x), vehicle(x), truck(x), airplane(x), location(x) and airport(x).

We can then define the operators more formally as shown on the following page:

- load-truck(package *pkg*, truck *trk*, location *loc*)
  Precondition: at(*pkg*, *loc*) $\wedge$ at(*trk*, *loc*)
  Effects: $\neg$at(*pkg*, *loc*), in(*pkg*, *trk*)

- load-airplane(package *pkg*, airplane *plane*, location *loc*)
  Precondition: at(*pkg*, *loc*) $\wedge$ at(*plane*, *loc*)
  Effects: $\neg$at(*pkg*, *loc*), in(*pkg*, *plane*)

- unload-truck(package *pkg*, truck *trk*, location *loc*)
  Precondition: in(*pkg*, *trk*) $\wedge$ at(*trk*, *loc*)
  Effects: $\neg$in(*pkg*, *trk*), at(*pkg*, *loc*)

- unload-airplane(package *pkg*, airplane *plane*, location *loc*)
  Precondition: in(*pkg*, *plane*) $\wedge$ at(*plane*, *loc*)
  Effects: $\neg$in(*pkg*, *plane*), at(*pkg*, *loc*)

- drive-truck(truck *trk*, location *from*, location *to*)
  Precondition: at(*trk*, *from*) $\wedge$ same-city(*from*, *to*)
  Effects: $\neg$at(*trk*, *from*), at(*trk*, *to*)

- fly-airplane(airplane *plane*, airport *from*, airport *to*)
  Precondition: at(*plane*, *from*)
  Effects: $\neg$at(*plane*, *from*), at(*plane*, *to*)