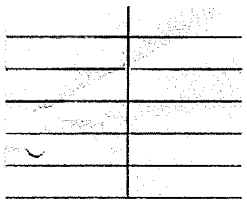
Försättsblad till skriftlig tentamen vid Linköpings universitet

Datum för tentamen	2017-10-28
Sal (1)	TER1(12)
Tid	14-18
Kurskod	TDDD48
Provkod	TEN1
Kursnamn/benämning **Provnamn/benämning**	Automatisk planering Skriftlig tentamen
Institution	IDA
Antal uppgifter som ingår i tentamen	4
Jour/Kursansvarig Ange vem som besöker salen	Jonas Kvarnström
Telefon under skrivtiden	0704-737579
Besöker salen ca klockan	nej (nås bara på telefon)
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, ankn. 2362, anna.grabska.eklund@liu.se
Tillåtna hjälpmedel	inga
Övrigt	
Antal exemplar i påsen	

TDDD48 Automated Planning 2017-10-25

Important Instructions: Read before you begin!

- Though the questions are in English, feel free to answer in Swedish if you prefer! **Det går bra att skriva på svenska!**
- Please, write clearly (block letters if necessary), and use the *lined* side of the paper unless you have a good reason not to! Checked paper works well for math but tends to make text difficult to read...



- *Clear and comprehensible* explanations and motivations are always required. This does not necessarily mean that each answer should be a long essay. What is important is that all the relevant facts are present and clearly explained.
- To see if you have succeeded, turn the tables and **try to misunderstand.** Specifically, after you write an answer, go back and see if you can creatively misinterpret the answer. If so, *clarify*, *clarify*, *clarify*!
- A good way to show your knowledge is to write explanations that can be understood by someone *else* who *does not already know* the correct answer.

  If your answer explains a topic well enough that a fellow student could *learn* something new from the answer and/or apply it in practice, you have probably succeeded.

  Conversely, if we ask how HTNs work, saying that "there are methods and operators and you construct a tree using patterns and there can be constraints too, and there's no goal" does not provide much in terms of *useful* information, and certainly will not give you a full score.
- Concrete examples or counterexamples may be useful as part of a motivation. If so, please make sure you include all relevant information about the example you have chosen to use. What is relevant naturally depends on how you use the example.
- When asked to provide examples or illustrate something through a planning problem instance, save time by *keeping the example as small as possible*.

1 Search, Search Guidance and Heuristics

This question relates to *fact landmarks* in the simple Blocks World domain. As you have seen, states in this domain can be described using five predicates:

- (on-table ?x) – block ?x is on the table.
- (on ?x ?y) – block ?x is on top of block ?y.
- (holding ?x) – the robotic hand is holding block ?x.
- (clear ?x) – you are free to place something on top of block ?x. This implies that there is nothing on top of it, and you are not holding it.
- (handempty) – the single robotic hand is not holding a block.

Assume the following 5-block problem instance with the blocks A,B,C,D,E:

- Current state: {(clear E), (on E D), (on D C), (on C B), (on B A), (ontable A), (handempty)} – All blocks are in a single tower with E on top.
- Goal: {(on A B), (on B C), (on C D), (on D E)} – All blocks are in a single tower in the other order.

This problem instance results in a total of 41 atomic facts that may be true or false, including the initial and goal facts shown above. Therefore there are $2^{41} \approx 2 \cdot 10^{12}$ states. These can be modified using 4 actions:

- (pickup ?x) picks up a block that is clear and on the *table*, given that you are not already holding a block.
- (unstack ?x ?y) picks up a block that is clear and on top of another *block*, given that you are not already holding a block.
- (putdown ?x) puts the block you are holding on the table.
- (stack ?x ?y) puts the block you are holding on top of a clear block.

For the purpose of this question, you should only require a general understanding of these actions. However, the complete action specifications are also available in an appendix.

Question: The problem instance also has more than 10 fact landmarks that can be useful for landmark-based heuristics. Please *specify* four of these. For each one, you should clearly *motivate* why it is a fact landmark. The general definition of a fact landmark is not sufficient as a motivation – you must also explain (at least briefly) why each landmark actually satisfies this definition.

Please constrain your answer to the list of landmarks and your motivations, and do not explain other aspects of landmark heuristics. (2p)

The following is a standard formulation of the Blocks World in PDDL.

```
(define (domain blocksworld)
  (:requirements :strips)

(:predicates (clear ?x)
             (on-table ?x)
             (handempty)
             (holding ?x)
             (on ?x ?y))

(:action pickup  :parameters (?ob)
  :precondition (and (clear ?ob) (on-table ?ob) (handempty))
  :effect (and (holding ?ob) (not (clear ?ob)) (not (on-table ?ob))
               (not (handempty))))

(:action putdown  :parameters  (?ob)
  :precondition (holding ?ob)
  :effect (and (clear ?ob) (handempty) (on-table ?ob)
               (not (holding ?ob))))

(:action stack  :parameters  (?ob ?underob)
  :precondition (and (clear ?underob) (holding ?ob))
  :effect (and (handempty) (clear ?ob) (on ?ob ?underob)
               (not (clear ?underob)) (not (holding ?ob))))

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

2 Partial-Order Planning

We now consider partial-order planning. We use the standard **logistics domain** as defined in Appendix A; please familiarize yourself with it before continuing.

We will use 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 $\mathsf{at}(p_1, l_1)$, $\mathsf{at}(p_2, l_2)$, $\mathsf{at}(t_1, l_2)$, and $\mathsf{at}(t_2, l_2)$. All locations are in the same city: $\{\mathsf{same\text{-}city}(l, l') \mid l, l' \in \{l_1, l_2, l_3\}\}$. No packages are loaded into vehicles.

The goal is that $\mathsf{at}(p_1, l_3)$, $\mathsf{at}(p_2, l_3)$, $\mathsf{at}(t_1, l_2)$, and $\mathsf{at}(t_2, l_2)$.

Below you will be required to **demonstrate a number of partially ordered plans** that could be encountered during the planning process. Unless we explicitly say otherwise, you must clearly show the following in each such plan:

- For each action, all preconditions *above* the action and all effects *below* the action. (Or, if you write them horizontally: Preconditions to the left, effects to the right.)
- All other relevant structural features in the plan: precedence constraints (solid arrows), causal links (dashed arrows) and threats.

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 or the planning procedure illustrated during the course lectures. 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* additions that would be made relative to the initial partial plan, including new constraints and relations as discussed above! **(2 points)**

c) Show a partial plan for this problem in which one or more threats appear. Indicate all threats clearly, if there is more than one, and explain why they are threats. (Note that you might be able to extend this plan into a solution in the next subquestion.) **(2 points)**

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)**

3 SATisfiability-based Planning

We will now consider planning based on propositional satisfiability ("SAT planning").

a) SAT-based *planners* must be able to find solution plans of arbitrary length, where the length is not known in advance. In contrast, the general SAT *solvers* that they are based on require an input containing a fixed and finite number of boolean propositions. How is this discrepancy handled by a SAT planner? **(2 points)**

b) SAT planners require *frame axioms* (for example, *explanatory* frame axioms) to ensure that the answer from the SAT solver corresponds to a valid plan.

Explain what the frame axioms "say" and what would happen if they were not included. You do not need to be able to write a frame axiom formula, but should still be able to explain the concept sufficiently clearly that someone with experience in SAT solving could define axioms based on your explanation.

Hint: If this question had been about *exclusion axioms*, you could have answered that *exclusion axioms "say" that if one action occurs at a particular time step, then no other action can occur at the same time step; if this axiom was omitted, then plans could contain concurrent actions.*

Demonstrate / illustrate your answer using a concrete example. The example does not have to involve an entire SAT assignment and SAT formula processed from beginning to end, as this would take considerable time to generate by hand. It is sufficient to illustrate the most central aspects of how the planning process could return invalid plans given a lack of frame axioms.

(2 points)

4 Markov Decision Processes

a) The states and transitions involved in a classical planning problem can be described using a specific type of state transition system, which is specified as a tuple $(S, \ldots)$ that includes a finite set of states and some additional information that you should already be aware of. This is also the case for the states and transitions involved in a fully observable Markov Decision Process (MDP).

However, there is a key difference in expressivity between classical planning problems and MDP problems, which is also reflected in the information provided by the respective state transition systems. What is this difference, and how is it represented in the state transition system $(S, \ldots)$ of an MDP? **(1 point)**

b) Markov Decision Processes are characterized by the *Markov property*. What is the Markov property? **(1 point)**

c) Markov Decision Processes often use a *discount factor*. Which value related to MDPs is calculated using this discount factor, and how is the discount factor used in the calculations? The exact formula is not required – an intuitive description is sufficient, as long as it gives a *clear* picture of how discount factors are used.

Also explain why the use of a discount factor is not only a mathematical trick but in many cases results in a better model of our preferences/goals/desires than if the discount factor had not been used. **(2 points)**

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*)