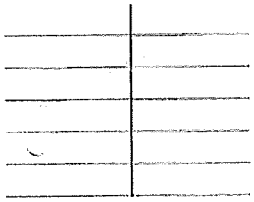
Försättsblad till skriftlig tentamen vid Linköpings universitet

Datum för tentamen	2018-11-02
Sal (1)	G36(7)
Tid	8-12
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	013-28 23 05 eller 0704-737579
Besöker salen ca klockan	ca 09:00
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 2018-11-02

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 except for diagrams! 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.*

Notation and Terminology

To avoid unnecessary confusion, we will hint at some of the terminology that you have hopefully learned during the course. **Please be careful to use the correct words!**

- **Predicate** (name/symbol) – at, on, raining
- **Object** (name/symbol) – A, B, red, truck1, helicopter8, location4
- **Variable** – *truck*, *location* – in italics
- **Atom** – at(mytruck, overthere), at(*truck*, location4), raining
- **Ground atom** – at(mytruck, overthere), raining
- **Fact** – typically a ground atom
- **Literal** – atom or negated atom
- **Ground literal**
- **Formula** – combination of atoms using connectives
- **State** – a full specification of the "configuration" of the world (to the extent it is modeled in the problem specification)
- **Goal formula, goal state, set of goal states, goal fact, ...**
- $h^*(s)$ – the cost of an optimal plan reaching a goal state from state s

1 General Concepts in Automated Planning

a) **State transition systems:** Please define the three components of a standard state transition system, as used in classical planning. You should also clearly (but briefly) explain each of those three components. **(2 points)**

b) **Reachability and physically possible states:** Let us define a *physically possible state* as a state that corresponds to a situation that may actually occur in the real world. This can be contrasted with physically *impossible* states, which may include facts such as on(A,A) (representing that an entity is on top of itself), or contain both at(entity,loc1) and at(entity,loc2) (representing that an entity is at two distinct locations).

Consider the standard Logistics domain, defined in Appendix A. Can the state achieved by executing an instance of the drive-truck operator ever be a physically impossible state? If so, provide an example. If not, provide a clear motivation. **(2 points)**

c) **Domain transition graphs:** Explain clearly what a *domain transition graph* represents (what information it provides) and how to construct one given a planning problem instance. Exemplify and illustrate such a graph, based on a problem instance that you define for a domain of your choice. Hint: You will have to use a state-variable representation for the DTG to make sense. **(3 points)**

2 Search Techniques

a) In *backward search*, what does a *search node* contain / correspond to? In other words, what *information* is stored there, that the search algorithm can use to determine whether the goal is satisfied and what successors exist? **(1 point)**

b) Given a search node in backward search (according to the previous question), how do we define the set of *successors*? For full points, your answer should be sufficiently detailed that it can be used to determine/compute the set of successor search nodes of a given search node given an arbitrary planning problem instance. **(3 points)**

3 Heuristics

This question relates to **fact landmarks** in the simple Blocks World domain. As you have seen, states in this domain can be described with atoms related to 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*, which requires handempty.
- **(unstack ?x ?y)** picks up a block that is clear and on top of another *block*, which requires handempty.
- **(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 on the next page.

Question: The problem instance has more than 10 positive (non-negated) 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. **(3 points)**

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)))))
```

4 Planning with Incomplete Information

a) Explain the meaning of three types of planning: *Fully observable, partially observable* and *non-observable*. Include sufficient detail for the reader to understand the differences between these types of planning and the consequences for the planner and plan executor.

Hints: This is about observations, but *who* would observe something? *What* can or can't this entity observe? *When* would (or wouldn't) such observations be made? When could the observations be *useful* and what would they be used for? What information do you have if you *don't* have observations? **(3 points)**

b) *Policy iteration* is one of several algorithms that can be applied to Markov Decision Processes. Please describe the policy iteration algorithm. You do not need to include formulas, but there should be sufficient details to understand the steps in the algorithm at a higher level. **(3 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 every location must be in the same city 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 $\text{at}(t_1, t_1)$ 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 follows:

- load-truck(package *pkg*, truck *trk*, location *loc*)
  Precondition: at(*pkg*, *loc*) ∧ at(*trk*, *loc*)
  Effects: ¬at(*pkg*, *loc*), in(*pkg*, *trk*)

- load-airplane(package *pkg*, airplane *plane*, location *loc*)
  Precondition: at(*pkg*, *loc*) ∧ at(*plane*, *loc*)
  Effects: ¬at(*pkg*, *loc*), in(*pkg*, *plane*)

- unload-truck(package *pkg*, truck *trk*, location *loc*)
  Precondition: in(*pkg*, *trk*) ∧ at(*trk*, *loc*)
  Effects: ¬in(*pkg*, *trk*), at(*pkg*, *loc*)

- unload-airplane(package *pkg*, airplane *plane*, location *loc*)
  Precondition: in(*pkg*, *plane*) ∧ at(*plane*, *loc*)
  Effects: ¬in(*pkg*, *plane*), at(*pkg*, *loc*)

- drive-truck(truck *trk*, location *from*, location *to*)
  Precondition: at(*trk*, *from*) ∧ same-city(*from*, *to*)
  Effects: ¬at(*trk*, *from*), at(*trk*, *to*)

- fly-airplane(airplane *plane*, airport *from*, airport *to*)
  Precondition: at(*plane*, *from*)
  Effects: ¬at(*plane*, *from*), at(*plane*, *to*)

Total points: 20