Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2013-08-22
Sal (1) Om tentan går i flera salar ska du bifoga ett försättsblad till varje sal och ringa in vilken sal som avses	TER3
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
Besöker salen ca kl.	ja
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, ankn. 23062, anna.grabska.eklund@liu.se
Tillåtna hjälpmedel	inga
Övrigt	
Vilken typ av papper ska användas, rutigt eller linjerat	Valfritt
Antal exemplar i påsen	

Exam: TDDD48 Automated Planning 2013-08-22

Important Notes

Read the following before you begin!

- Though the questions are in English, you may **answer in Swedish** if you prefer!
- *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.
- 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.

1 Fundamental Concepts

a) Explain the difference between *domain-independent* and *domain-specific* planning. Also, for each of these two types of planning, describe one advantage it has over the other. **(2 points)**

b) We know the following:

   - A (sequential) solution plan π for a classical problem instance P is *redundant* iff it is possible to remove one or more actions from π in such a way that the remaining actions, *in their original order*, still form a solution that achieves the goal. For example, the solution $[a_1, a_2, a_3, a_4, a_5, a_6]$ is redundant if a subsequence such as $[a_1, a_2, a_3, a_6]$ is also a solution.
   - A solution plan π for a given problem instance is *minimal* iff there is no other solution for P that contains strictly fewer actions.

   The standard **logistics domain**, where a set of packages should be transported to their destinations, is defined in Appendix A. Create a classical planning problem instance for this domain, including a clearly specified initial state and goal. Show a *redundant* solution for this problem instance. Show why the solution is redundant – in other words, indicate which actions can be removed from the solution. **(1 point)**

c) Create a classical planning problem instance for the same domain, including a clearly specified initial state and goal. Show a solution for this problem instance that is *not* redundant but still not minimal. Explain / motivate clearly why the solution is not redundant and why it is not minimal. **(1 point)**

Clarifications and hints:

- Don't add too many objects or goals – even quite small problem instances will allow you to generate redundant plans, and using large instances will take you more time than necessary.

2 Hierarchical Task Networks

Recall that Simple Task Networks are a simple form of Hierarchical Task Networks and are used by the book as well by the lecture slides.

a) Explain the difference between *total-order* and *partial-order* Simple Task Networks. Also describe an HTN planning problem that can be modeled more easily or more naturally as a partial-order STN than as a total-order STN. **(2 points)**

3 SAT-based Planning

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

a) SAT planners translate planning problems into a set of boolean propositions and a set of formulas over these propositions. Typically, there are two specific types of information encoded as boolean propositions. In other words, once a SAT solver has found a solution, two specific types of information can be inferred from the values of the boolean propositions. Which ones? **(2 points)**

b) SAT *planners* must be able to find solution plans of arbitrary length, where the length is not known in advance. In contrast, the 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)**

c) SAT planners require *frame axioms* (for example, *explanatory* frame axioms) to ensure that the SAT solver gives correct results that correspond to valid plans.

   Explain how planning could go wrong if frame axioms were not included in the SAT translation of a planning problem. Also demonstrate 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. 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 definition of their respective state transition systems. What is this difference, and how is it represented in the state transition system 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)**

d) What is the main difference between *policy iteration* and *value iteration*? What consequences can this difference have in terms of convergence towards an optimal policy for an MDP? **(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**.

- 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 any location is 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 at(t_1, t_1) is not merely false but syntactically incorrect. Should you need **type predicates** at some point, you can assume that thing(x), package(x), vehicle(x), truck(x), airplane(x), location(x) and airport(x) are inferred automatically by the planner given the typed values you define in the problem specification.

We can then define the operators more formally as shown below:

- 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*)