Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2013-01-09
Sal (1) Om tentan går i flera salar ska du bifoga ett försättsblad till varje sal och ringa in vilken sal som avses	TER2
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	ankn. 23 05 eller 0704-737579
Besöker salen ca kl.	ja
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, ankn. 2362, 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 2013-01-09

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.

1 Classical Planning and State Transition Systems

A *state transition system* $\Sigma = (S, A, E, \gamma)$ consists of the following components:

- $S = \{s_0, s_1, \ldots\}$ – a finite, or infinite but enumerable, set of world states
- $A = \{a_0, a_1, \ldots\}$ – a finite, or infinite but enumerable, set of actions
- $E = \{e_0, e_1, \ldots\}$ – a finite, or infinite but enumerable, set of exogenous events
- $\gamma : S \times (A \cup E) \rightarrow 2^S$: A state transition function, specifying the set of possible states we might end up in after a given action or event occurs in a particular state

a) Classical planning does not support the full expressivity of a state transition system. For example, in classical planning the sets above must be finite. Name two other distinct restrictions that are made in classical planning relative to the full expressivity of a state transition system. **(2 points)**

b) In contrast, many modern planners have a level of expressivity that in some ways goes beyond that of a state transition system. For example, probabilistic planners specify not only which states you may end up in after executing an action but also the *probability* of ending up in each of the possible outcome states. Name two other distinct expressivity extensions that exist in some modern "extended classical" planners and allow us to model different types of planning domains.

Note that language extensions such as conditional effects and disjunctive preconditions do not count, as they are merely convenient "syntactic sugar" and can easily be compiled down into (a larger number of) standard classical actions. We are also not considering control formulas or hierarchical task networks, which can be seen more as guidance to the planner rather than as support for different types of domains. **(2 points)**

c) Explain why we generally use planning domain definition languages such as PDDL to specify planning problem instances, rather than directly specifying each component of the corresponding state transition system. **(1 point)**

2 Planning and Satisfiability

Recall that SAT planning is based on translations of planning domains and problem instances into *propositional satisfiability* problems consisting of propositional formulas to be satisfied by assignments of values to propositional variables.

Assume an extremely simple planning problem instance with:

- Three ground fact atoms: {at1, at2, at3}
- Four ground actions:
  move12 with precondition at1 and effect {at2, ¬at1},
  move23 with precondition at2 and effect {at3, ¬at2},
  move32 with precondition at3 and effect {at2, ¬at3},
  move21 with precondition at2 and effect {at1, ¬at2}.
- A simple initial state: {at1}
- A simple goal: {at3}

How would a standard SAT-based planner translate and process this planning problem, up to the point where a solution is found? Your answer should be in the shape of a step-by-step demonstration where one clearly sees the *process* of planning, together with explicit answers to the questions below. Clearly indicate where each question is answered, and please provide both general descriptions and concrete examples whenever possible. **(5 points)**

a) How can we represent the fact that a particular action is part of a plan, given that a SAT solver only knows about propositions and has no built-in concept of actions?

b) How can we represent preconditions and effects of an action? Give a concrete example demonstrating how to

c) How can facts and action executions at different timepoints be represented, given that a SAT solver has no built-in concept of time?

d) How can a SAT-based planner test plans of different and unbounded length, given that a SAT solver only takes a fixed set of propositions as input?

e) How do we ensure that the plans that are generated are sequential and do not execute two actions at the same time?

3 Search Guidance

Given the size of a typical search space, a planner almost always needs some form of search guidance as opposed to doing blind search. Often such guidance takes the shape of a *heuristic function* that evaluates the "quality" of a certain search node, corresponding to one specific choice that can be made at a particular point in the search space.

a) Heuristic functions can yield *plateaus* in the search space. What is a plateau? Visualize one using part of a search space: Show a set of search nodes, a number of possible transitions between nodes, and the type of heuristic values that characterize a plateau. Show which nodes are part of the plateau and explain in words *why* they form a plateau. **(2 points)**

b) Is the $h_{\max}$ heuristic, also called h_1, admissible? If it is, motivate clearly why. If it is not, demonstrate using a counterexample: Show a state and goal for which h_{add} is *not* admissible, using a planning domain of your choice. **(1 point)**

c) What is a *landmark* (not an *action landmark*)? Explain the general definition and give an example of a concrete landmark from a simple planning domain such as for example Logistics, Blocks World or Towers of Hanoi.

Give an example of how landmarks can be used in the definition of a heuristic function for state-space planning. Keep the description general as opposed to applying it to a specific state or problem instance! **(2 points)**

4 Classical Planning vs. Hierarchical Task Networks

In the Towers of Hanoi puzzle, there are three *rods* and a number of *disks* of distinct sizes which can be placed on these rods. Initially, all the disks are on a single rod and are sorted by size, with the largest disk at the bottom:

The objective is to move the entire pile of disks to another rod so that the disks remain in the same order, while following these rules:

1. Each action consists of picking up the topmost disk from one of the rods and placing it on another rod. If other disks are already placed on that rod, the new disk is placed on top of them. (These are physical constraints on what you are able to do. For example, it isn't physically possible to pick up the third disk from the top without moving the other disks first, and you are only able to carry one disk at a time.)
2. A disk must never be placed on top of a smaller disk. (This additional rule is what makes the problem an interesting puzzle.)

You should now create two distinct models of the Towers of Hanoi domain as planning problems.

a) Create a **planning domain** for the Towers of Hanoi puzzle using **classical planning extended with numbers**. That is, you must base your domain on standard classical planning constructs but may extend this with integer numbers and numeric operators such as addition (+), comparison (<) and so on in order to simplify the model. You may also use disjunctions, and quantifications in preconditions, as well as quantified and conditional effects. Depending on what you feel is more convenient, you may choose to use a *classical representation* with predicates whose instances are deleted and added by actions, or a *state-variable representation* where variables are reassigned new values by actions.

   In the planning domain, specify clearly which object types you use, which predicates or state variables you use (with typed arguments), and so on. Also specify a *move* operator that can be used to move a disk from one rod to another. Its preconditions and effects must ensure that the rules shown above are followed! The number of disks

and their initial placement must not be hardcoded in the planning domain. Instead, this information should be specified in the problem instance.

You should also specify **one problem instance** where there are 7 disks, all initially placed on a single rod in order of size. The goal of this problem instance should be for all disks to be on another specific rod.

Clarity matters more than exact syntax. Thus there is no need to precisely follow PDDL or any other standard syntax, but it *must* be very clear how your definitions map into a standard planning domain and problem instance! If in doubt, add more explanations. **(3 points)**

b) Create a **planning domain** for the Towers of Hanoi puzzle using **hierarchical task networks**. Specifically, define an HTN formulation of the elevator domain for a **TFD-like** procedure (Total-order Forward Decomposition) with totally ordered methods. You may use the move action you defined above as a primitive task.

In this formulation the "goal" will be represented through an initial task network consisting only of the non-primitive task move-all(*from*,*to*, *intermediate*), representing the intention that all disks currently on rod *from* should be moved to rod *to* using rod *intermediate* for intermediate storage. The assumption is that both *intermediate* and *to* are initially empty, which means that the following algorithm can be used (given n disks on rod *from*):

1. Move $n-1$ disks from rod *from* to rod *intermediate* (with *to* as temporary storage)
2. Move the last (bottom) disk from rod *from* to rod *to*
3. Move $n-1$ disks from rod *intermediate* to rod *to* (with *from* as temporary storage)

Represent this solution method as an HTN planning domain. Note that JSHOP2 uses a somewhat different structure for tasks and methods compared to the book. You may choose to use either the JSHOP2 structure or the book structure, as long as it is indicated clearly which one you use.

As above, clarity matters more than exact syntax. It *must* be very clear how your definitions map into a standard HTN planning domain and problem instance! For example, for every method you create, you must specify which task it corresponds to, which preconditions it has, and which parameterized subtasks it is decomposed into. If in doubt, add more explanations. **(2 points)**