TI1600-A Multi-Agent Systemen

28 June 2013

This exam will test your knowledge and understanding of the material provided to you and presented in the lectures, the book of Blackburn, Bos and Striegnitz *Learn Prolog Now!* (Chapters 1 to 5, Section 10.3, Section 11.2), as well as *The* GOAL *Programming Guide*. It is *not* allowed to use materials such as books, papers or slides during the exam. You will have 3 hours (from 9 till 12) to complete the exam. You may provide your answers in Dutch as well as in English. It has 8 questions, for a total of 80 points. Please don't include irrelevant information: you will be marked down for this. Before you hand in your answers, please check that you have put your name and student number on top of every sheet you hand in.

Questions

Question 1 *10 points*

This assignment concerns Prolog.

(a) (10 points) Which of the following queries succeed, which fail, and which give rise to an error? If a query succeeds, provide the unifier that Prolog computes (if it is non-empty). For example, the query `f(X) = f(a).` succeeds, and Prolog computes the unifier `X=a`; the query `5 = 5` succeeds, and the unifier is empty (i.e., Prolog returns `true`).

```
1) p(a(A),f(a,Y),g(b)) = p(X, f(a,c), g(Z)).
2) [[1,2,3]|[]] = [1,2,3].
3) [10, f(a), [], [3]] = [_,X,Y|Z].
4) X > 6, X is 3+4.
5) 5 = X + 4.
```

Question 2 *5 points*

This assignment concerns Prolog.

(a) (5 points) Write a set of clauses that define the predicate `factorial(N, F)` which succeeds if `N` is a non-negative integer and `F` is the factorial of `N`, i.e., `F` = `N`!. ("`N` faculteit" in Dutch). The factorial of 0 is 1. For example, the query `factorial(3,F)` should yield `F = 6`, as `3 * 2 * 1 * 1 = 6`, and `factorial(5,F)` should yield `F = 120`, as `5 * 4 * 3 * 2 * 1 * 1 = 120`. *Do not define any auxiliary (=additional) predicates!*

Question 3 *10 points*

This assignment concerns Prolog.

(a) (10 points) Write a set of clauses that define the predicate `removeNil(L,R)` which succeeds if `L` is a list and `R` is a list that is the same as `L` except that all instances of the item `[]` (the empty list) have been removed *at all levels* of `L`. That is, if elements of `L` are lists, the item `[]` should also be removed from them, etc. For example, the query `removeNil([1,2,[],3],R)` should yield `R = [1,2,3]`, and the query `removeNil([1,2,[],[a,[],b]],R)` should yield `R = [1,2,[a,b]]`.

Define the predicate `removeNil(L,R)` by defining a clause for each of the following cases:

1. base case;
2. the first element of `L` is the empty list;
3. the first element of `L` is not a list; use the Prolog built-in predicate `is_list(X)` which succeeds if `X` is a list, and fails otherwise;
4. the first element of `L` is a list with at least one element.

Do not define any other clauses!

Question 4 *9 points*

This assignment concerns Prolog. Consider the following program (the numbers indicate clause numbers, for ease of reference):

```
1. invented(edison,lightbulb).
2. invented(colmeraurer,prolog).

3. iq(einstein,210).
4. iq(edison,160).
5. iq(waldorf,90).

6. genius(Person):-
       iq(Person,IQ),
       IQ > 150.
```

```
7. genius(Person):-
       invented(Person,_).
```

(a) (2 points) When we pose the query `genius(G).` what is the first answer that Prolog produces?

(a) `Person = edison`
(b) `Person = einstein`
(c) `G = edison`
(d) `G = einstein`
(e) `false`

(b) (3 points) Explain your answer to the previous question in a few sentences, referring to the clauses in the program using the clause numbers.

(c) (4 points) Give all answers that Prolog produces if we pose the query `genius(G).` and cycle through all answers by pressing ';' after each answer, *in the right order.*

Question 5 *6 points*

(a) (3 points) Agents can be applied naturally for controlling non-player characters such as bots in a capture-the-flag scenario in the game UNREAL TOURNAMENT. In this scenario multiple bots form a team that has the objective of capturing the flag of a competing team. Each team needs to defend its own flag and can kill bots of the other team by shooting them. Explain why you think software agents can be naturally applied for controlling such bots (use 5 sentences at most).

(b) (3 points) A software agent is usually connected to an environment and can receive percepts from its environment. Explain the difference between a "send always" and a "send-on-change" percept and provide an example of a "send-on-change" type of percept (use 5 sentences at most).

Question 6 *10 points*

This question concerns the agent programming language GOAL.

(a) (3 points) List three mental state operators that can be used in the conditions of a rule in a GOAL agent program to inspect the mental state of that agent.

(b) (4 points) Provide rules for processing the "send-on-change-with-negation" percept `in(RoomID)`.

(c) (3 points) Explain the difference between the **linear**, **linearall**, and **random** rule evaluation orders.

Question 7 *18 points*

This question concerns the agent programming language GOAL. Consider the agent program below. This program is written for a robot that gets beers for his owner from the fridge. Its main goal in life is to get and serve beers. The robot can open a closed fridge if it is at the fridge and not holding a beer. It can close an open fridge if it is at the fridge. It can grab a beer from the fridge by the `get` action if it is not holding a beer already, the fridge is open, and the robot is at the fridge. The robot can hand over a beer to its owner if it is holding beer and close to (at) its owner. Finally, it can move towards a place. Whenever the robot is at its owner, it can see whether its owner has a beer or not. In that case it receives a corresponding percept, otherwise the robot does not get any percepts. You may assume that there are always beers in the fridge ready for the robot to serve. In order for the robot to perform its job correctly and safe energy, it must always ensure that the fridge is closed if the robot is not at the fridge.

(a) (4 points) Explain which action(s) selected by rules from the **main** module the GOAL agent may perform *next*, given the agent program listed below. Only provide actions that it can perform in the mental state that results from processing once the percepts the agent receives by means of the **event** module. You may assume that the owner does not have beer and the agent will receive the percept `percept(not(ownerhasbeer))`. Note that actions refer to built-in GOAL actions such as `insert` as well as environment actions specified in the program's action specification section.

(b) (4 points) If the robot is not holding a beer and perceives that its owner already has a beer, will the robot still get a beer from the fridge? Explain your answer.

(c) (6 points) What is the default exit option of a module? If we would remove the option `exit=noaction` from the `getbeer` module, would the robot still serve beers to its owner? Would the robot still perform correctly? Explain your answer.

(d) (4 points) Complete the action specification for the action `close` for closing the fridge. Only use predicates that are already available in the given agent program. Motivate the pre- and postcondition that you have given.

```
init module {
  beliefs{
    fridgeclosed. place(owner). place(fridge). at(owner).
  }
  actionspec{
    open{
      pre{ not(holdingbeer), at(fridge), fridgeclosed }
      post{ not(fridgeclosed) }
    }
    close{
      pre{ ... }
      post{ ... }
    }
    get{
      pre{ not(holdingbeer), at(fridge), not(fridgeclosed) }
      post{ holdingbeer }
    }
    hand_over{
      pre{ holdingbeer, at(owner) }
      post{ not(holdingbeer) }
    }
    move_towards(Place) {
      pre{ place(Place), at(OldPlace) }
      post{ not(at(OldPlace)), at(Place) }
    }
}
main module {
  program[order=random] {
    if bel( not(holdingbeer) ) then adopt( holdingbeer ) + adopt( servebeer ).
    if a-goal( holdingbeer ) then getbeer.
    if bel( holdingbeer ), a-goal( servebeer ) then servebeer.
  }
}
module getbeer [exit=noaction] {
  program {
    if bel( not(at(fridge)) ) then move_towards(fridge).
    if bel( fridgeclosed ) then open.
    if bel( not(holdingbeer) ) then get.
    if bel( not(fridgeclosed) ) then close.
  }
}
module servebeer [exit=noaction]{
  program {
    if bel( not(at(owner)) ) then move_towards(owner).
    if bel( not(ownerhasbeer) ) then hand_over.
  }
}
event module {
  program {
    if bel( percept(not(ownerhasbeer)) ) then delete( ownerhasbeer ).
    if bel( percept( ownerhasbeer)) ) then insert( ownerhasbeer ).
  }
}
```

Question 8 *12 points*

This question concerns cooperative teams of robots in the BlocksWorld for Teams (BW4T) environment (see figure) that want to perform the BW4T task. BW4T is a virtual office-like environment that consists of rooms in which colored blocks are hidden, and a drop zone where blocks can be delivered. The BW4T task is to deliver a sequence of colored blocks in a particular order (an example task is illustrated at the bottom of the figure). The basic actions that agents in the BW4T world can perform are to move, to pick up blocks, and to drop blocks. An agent can only carry one block at a time. Agents know which rooms there are and which colors need to be delivered in what order, but they can only see blocks and their colors when they are inside the room where these blocks are. At most one agent (robot) can be in a room at any time (entrance to a room is blocked as soon as one robot enters a room). Agents cannot see each other. To deliver a block successfully, an agent has to find a block of the right color, go to the block, pick it up, go to the drop zone and drop the block there. Performance on the BW4T task is measured by the time needed to complete the task. The task is completed at the moment that all blocks needed have been delivered (dropped) in the right order in the drop zone. You may assume that there are always sufficiently many blocks present in the environment to achieve the BW4T task.

Suppose there are two robots trying to solve the BW4T task together. Also suppose that both robots execute *exactly the same* strategy (program); the programs do not depend on any information that differentiates the robots such as names. This program makes a robot adopt a (single!) goal to go to and pick up a block of the right color if it is not yet holding a block and it knows where it can pick up such a block. A block is of the right color if it matches the color of the block that is needed next according to the BW4T task. If a robot has such a goal, it will perform the right actions to achieve it (and drop the goal only if it knows that it cannot achieve its goal anymore or has achieved the goal). Only if a robot does not know where to find a block of the right color, it will randomly visit a room that it has not visited before. Otherwise it will do nothing. Finally, each robot always immediately communicates its beliefs about the location of blocks to the other robot when it receives new information about the location of blocks. They do not share any other information.

Robots need to coordinate to avoid duplicating effort. Here we mean by duplication of effort two things: (i) both robots duplicate effort if both carry a block where only one would have been sufficient, and (ii) both robots visit one and the same room whereas only one robot visiting that room would have been sufficient. We do not count under any circumstances two robots walking one and the same path as duplicate effort. In this question we are looking for conditions where coordination can be minimal.

(a) (6 points) Specify a sufficient and necessary constraint on the number of rooms present in the BW4T environment which ensures that the two robots will never duplicate effort. That is, provide a constraint on the number of rooms such that the robots never will try to pick up the same block and waste time in doing so. Argue that your constraint is (i) sufficient and (ii) necessary. A constraint is sufficient in case the robots in all situations that satisfy the constraint will never duplicate effort; it is necessary if the robots in some situations would duplicate effort if (part of) the condition you specify does not hold.

(b) (6 points) Specify a sufficient and necessary constraint on how blocks are distributed over rooms and on the BW4T task which ensures that the two robots will never duplicate effort. That is, provide a constraint on blocks and the BW4T task such that the robots will avoid ever going to a room with the goal to pick up the same block. Argue that your constraint is (i) sufficient and (ii) necessary.

End of exam

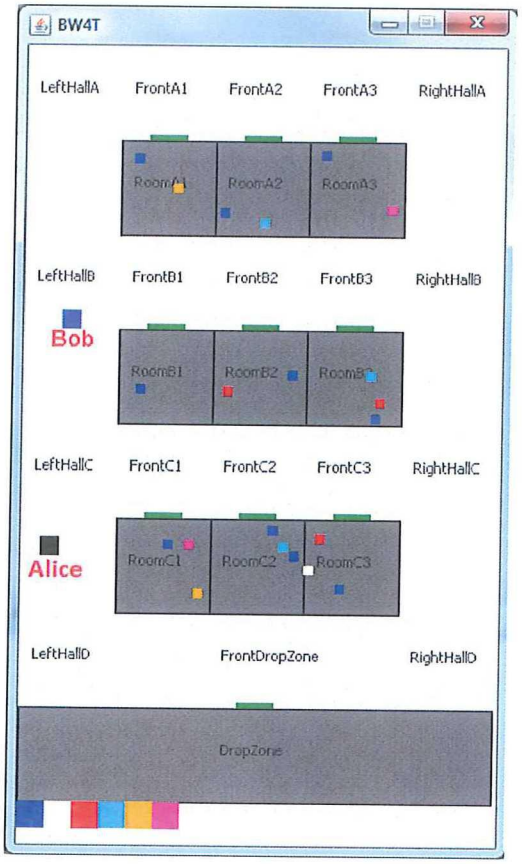

Figure 1: BW4T world