TI1600-B Multi-Agent Systemen

1 July 2011

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, Sections 9.1 & 9.2, Chapter 10, Section 11.2), as well as *The* GOAL *Programming Guide* and the papers "Rules Capturing Events and Reactivity" (Paschke and Boley) and "Common ground and coordination in joint activity" (Klein et al.). 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 7 questions, for a total of 100 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 rule-based programming.

(a) (5 points) Explain the difference between *real-time* and *any-time* rule systems.

Solution: Real-time reaction rules systems respond to events within stringent timing constraints, where any-time rule systems impose no constraints on the reaction time wrt to the processed events.

(b) (5 points) What is the difference between forward-chaining and backward-chaining style execution of rules in rule-based languages? Provide an example system for each style of execution.

Solution: Forward-chaining execution starts from facts and reasons towards conclusions that may be derived from those facts (data-driven). Expert systems use forward-chaining. Backward-chaining execution starts with a goal (goal-driven) and aims to reason in reverse from that goal back to the facts that support the goal. Prolog is an example that uses backward-chaining.

Question 2 *10 points*

This assignment concerns Prolog.

(a) (10 points) Which of the following queries succeed? If a query succeeds, provide the unifier that Prolog computes. For example, the query `f(X) = f(a).` succeeds, and Prolog computes the unifier `X=a`.

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

Solution:

```
1) Y = 5.
2) false.
3) X = [3, 4], A = [1, 2], B = [[d]].
4) X = [1, 2], Y = a.
5) X = g(c), Z = A, Y = a.
```

Question 3 *10 points*

This assignment concerns Prolog.

(a) (7 points) Write a (set of) clause(s) that define the predicate `removeFirst(PairL,SecondL).` such that `SecondL` is a list obtained from the list of pairs `PairL`, by removing the first element of each pair. For example, the query `removeFirst([(1,2),(3,4),(5,6)],L).` should yield `L = [2, 4, 6].` *Do not use any auxiliary (=additional) predicates!*

(b) (3 points) Write a (set of) clause(s) that define the predicate `removeFirstandAdd(PairL,SecondL,Sum).` such that `SecondL` is a list obtained from the list of integer pairs `PairL`, by removing the first element of each pair, and `Sum` is the summation of the elements in `SecondL`. For example, the query `removeFirstandAdd([(1,2),(3,4),(5,6)],L,Sum).` should yield `L = [2, 4, 6], Sum = 12.` *Do not use any auxiliary (=additional) predicates!*

Solution:

```
removeFirst([],[]).
removeFirst([(X,Y)|T1],[Y|T2]) :- removeFirst(T1,T2).

removeFirstandAdd([],[],0).
removeFirstandAdd([(_,Y)|T1],[Y|T2],Sum) :- removeFirstandAdd(T1, T2, SumSub),
                                            Sum is Y + SumSub.
```

Question 4 *10 points*

This assignment concerns Prolog. Consider the following program:

```
myMember(X,[X|_]) :- f(X).
myMember(X,[_|T]) :- myMember(X,T).

a(a).
a(b).
a(3).
b(b).
f(X):- a(X), not(b(X)).
```

When posing the query `myMember(X,[a,b,3]).` to Prolog, the following trace is produced (only the first part is displayed here):

```
[trace]  ?- myMember(X,[a,b,3]).
   Call: (7) myMember(_G392, [a, b, 3]) ? creep
   Call: (8) f(a) ? creep
   Call: (9) a(a) ? creep
   Exit: (9) a(a) ? creep
^  Call: (9) not(b(a)) ? creep
   Call: (10) b(a) ? creep
   Fail: (10) b(a) ? creep
^  Exit: (9) not(user:b(a)) ? creep
   Exit: (8) f(a) ? creep
   Exit: (7) myMember(a, [a, b, 3]) ? creep
X = a ;
   Redo: (7) myMember(_G392, [a, b, 3]) ? creep
   Call: (8) myMember(_G392, [b, 3]) ? creep
   Call: (9) f(b) ? creep
   Call: (10) a(b) ? creep
   Exit: (10) a(b) ? creep
^  Call: (10) not(b(b)) ? creep
   Call: (11) b(b) ? creep
   Exit: (11) b(b) ? creep
^  Fail: (10) not(user:b(b)) ? creep
   Fail: (9) f(b) ? creep
   Redo: (8) myMember(_G392, [b, 3]) ? creep
   Call: (9) myMember(_G392, [3]) ? creep
   Call: (10) f(3) ? creep
....
```

(a) (5 points) In the text below that explains the trace, choose from the following terms to complete the text. Terms should be inserted in the appropriate slots, where each term can be used zero or more times:

- backtracking
- breadth-first search
- depth-first search
- linear search
- negation as failure

Prolog applies __________ (1) and thus unifies `myMember(X,[a,b,3])` with the head of the first clause in the program, resulting in the unification of `_G392` with a. In order to prove f(a), Prolog unifies this goal with the last rule in the program. Prolog applies __________ (2) and thus tries to prove a(a), which succeeds, before trying to prove not(b(a)). The goal not(b(a)) succeeds because Prolog uses __________ (3), which means that not(b(a)) succeeds if b(a) fails. Prolog returns the unifier X=a. The user presses `;`, which makes Prolog apply __________ (4), and thus Prolog unifies `myMember(X,[a,b,3])` with the head of the second clause. Prolog calls myMember recursively, and unifies the goal `myMember(_G392, [b, 3])` with the first clause in the program. Prolog tries to prove f(b) which fails, which makes Prolog apply __________ (5), resulting in unification of `myMember(_G392, [b, 3])` with the second clause in the program.

Solution:

1. linear search
2. depth-first search
3. negation as failure
4. backtracking
5. backtracking

(b) (5 points) Assume now that the first clause of the program is changed to:
`myMember(X,[X|_]) :- f(X),!.`
Is the cut (!) in this modified program a green cut or a red cut? Explain your answer by explaining the meaning of cut and explaining the difference between a green and red cut.

Solution: The cut prevents backtracking once Prolog passes it when proving the goals in the body of a rule. This is a red cut, because the behavior of the program changes if the cut is removed. For example, if the cut is removed and the query `myMember(X,[a,b,3])` is posed, Prolog returns `X=a` and `X=3`, while Prolog returns only `X=a` when the cut is present.

Question 5 *10 points*

This question concerns agent-oriented programming.

(a) (5 points) Explain the difference between the `a-goal` and the `goal` operator.

Solution: The `a-goal` operator is defined in terms of the `goal` and the `bel` operator as follows: `a-goal(`φ`)` = `goal(`φ`), not(bel(`φ`))`. Informally, the `goal` operator only checks whether φ follows from the goals in the agent's goal base whereas the `a-goal` operator also checks that the agent does not believe that φ is the case. Because of this additional condition `a-goal` defines achievement goals, goals that are still to be achieved.

(b) (5 points) Consider the action rule: `if bel(safe(X,Y)) then turn(right) + forward`. The `turn` action updates a predicate `facing(Dir)` and the `forward` action updates a predicate `at(X,Y)` in the agent's belief base. Assume that in the current state the agent is in the action forward will fail. What will happen if the condition `bel(safe(X,Y))` holds for some instantiation and the rule is fired?

Solution: The action `turn(right)` will be executed and the predicate `facing` will be updated in the agent's belief base. Even though the forward action will fail the predicate `at` is also updated (as this is an internal update action performed by the agent).

Question 6 *40 points*

This question concerns the agent programming language GOAL. Consider the agent program below.

(a) (10 points) Explain which action(s) that the GOAL agent may perform *next*, given the agent program listed below.

> **Solution:** The GOAL agent can perform the actions `load(p1)`, `load(p2)`, and `load(p3)`.

(b) (10 points) Complete the action specification for the action `unload(P)` for unloading a package `P` from the truck. A package, of course, can only be unloaded if it is in the truck and will be located where the truck is after unloading. Only use the predicates that are already available in the given agent program. Motivate the pre- and postcondition that you have given.

> **Solution:**
> ```
>  unload(P){
>   pre{ in(P,truck), loc(truck,X) }
>   post{ loc(P,X), not(in(P,truck)) }
> }
> ```
> The precondition contains `loc(truck,X)` to retrieve the current position of the truck in order to be able to insert the location of the unloaded package in the post-condition. The package no longer is in the truck after unloading so this fact is removed (negated) in the post-condition.

(c) (5 points) Explain why the agent will not be able to achieve either of its goals.

> **Solution:** After loading all packages into the truck, the agent will continuously fire the second rule and perform the `goto` action. The reason is that the `goto` action is also performed if the truck has already gone to a location because the precondition of `goto` does not check whether the truck is already at the location it is supposed to go to.

(d) (10 points) Explain how you would modify the **main module** of the agent program to ensure that the agent will achieve its goals. Provide a modification that is as minimal as possible.

> **Solution:** Simply exchange the order of the second and the third rule. (Another solution would be to add the condition `not(X=Y)` to the belief condition of the second rule but this condition would better be added to the precondition of the action `goto`.)

(e) (5 points) Assume that moving the truck with more packages is more costly than moving the truck with less packages and the truck driver wants to minimize costs. How would you change the agent program to optimize the behaviour of the `transportAgent` agent? Explain which GOAL constructs, if any, you will use to modify the agent code and why.

> **Solution:** One way to optimize the agent's behavior is to deal with the delivery order goal of one customer at a time. A module can be added to select one the goals and focus on that goal only. To create this focus, a `focus` option can be used in combination with the module.

```
main: transportAgent{
  knowledge{
    ordered(C,P) :- order(C,Y), member(P,Y).
    loaded_order(C) :- order(C,O), loaded(O).
    loaded([P]) :- in(P, truck).
    loaded([P|L]) :- in(P,truck), loaded(L).
```

```
    empty :- not(in(P,truck)).
    delivered_order(C) :- order(C,O),loc(C,X),orderloc(O,X).
    orderloc([H|T], X) :- loc(H,X), orderloc(T,X).
    orderloc([], X).
  }
  beliefs{
    home(a).
    loc(p1,a). loc(p2,a). loc(p3,a). loc(p4,c).
    loc(truck,a).
    order(c1,[p1,p2]). order(c2,[p3,p4]).
    loc(c1,b). loc(c2,c).
  }
  goals{
    delivered_order(c1). delivered_order(c2).
  }
  main module{
    program{
      if goal(delivered_order(C)), bel(ordered(C, P), loc(C, X)),
        not(bel(in(P, truck))), not(bel(loc(truck, X))) then load(P).
      if goal(delivered_order(C)), bel(loc(truck, X), loaded_order(C), loc(C, Y))
        then goto(Y).
      if goal(delivered_order(C)), bel(loc(truck, X), loc(C, X), in(P, truck), ordered(C, P))
         then unload(P).
      if bel(home(X)) then goto(X).
    }
  }
  actionspec{
    load(P){
      pre{ loc(truck,X), loc(P,X) }
      post{ in(P,truck), not(loc(P,X)) }
    }
    goto(Y){
      pre{ loc(truck,X) }
      post{ loc(truck,Y), not(loc(truck,X))}
    }
    unload(P){
      pre{ ... }
      post{ ... }
    }
  }
}
```

Question 7 *10 points*

This question concerns joint activity. In the paper *Common Ground and Coordination in Joint Activity* by Klein et al., three dimensions of joint activity are presented: criteria for joint activity, requirements for joint activity, and choreography of joint activity.

(a) (6 points) Which two criteria for joint activity are defined in the paper? Explain in a few sentences what they mean.

(b) (2 points) A requirement for joint activity is common ground. Explain in a few sentences what is meant by common ground and why it is a requirement for joint activity.

(c) (2 points) Give two examples of types of organizational structures.

Solution: (a) intention and interdependence (b) Common ground refers to the pertinent mutual knowledge, mutual beliefs and mutual assumptions that support interdependent actions in some joint activity. Common ground permits people to use abbreviated forms of communication and still be reasonably confident that potentially ambiguous messages and signals will be understood. (c) hierarchy and network/peer-to-peer

End of exam