

# Exam for IN1905-I Kennistechnologie

6 April 2010

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*. It is *not* allowed to use materials such as 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 6 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

This assignment concerns Prolog.

15 points

- (a) (10 points) Write a (set of) clause(s) that define the predicate `remove(X,L1,L2)` such that `L2` is a list obtained from the list `L1` by removing all occurrences of `X`. For example, the query `remove(3,[1,3,2,3,4],L2)` should yield `L2 = [1, 2, 4]` and `remove(3,[1,2,4],L2)` should yield `L2 = [1, 2, 4]`.
- (b) (5 points) Modify the (set of) clause(s) that define the predicate `remove(X,L1,L2)`, such that `L2` is a list obtained from the list `L1` by removing *only the first* occurrence of `X`. If `X` does not occur in `L1`, nothing should be removed. For example, the query `remove(3,[1,3,2,3,4],L2)` should yield `L2 = [1, 2, 3, 4]` and `remove(3,[1,2,4],L2)` should yield `L2 = [1, 2, 4]`.

### Question 2

This assignment concerns Prolog. Binary trees are trees where all internal nodes have exactly two children. The smallest binary trees consist of only one leaf node. We will represent leaf nodes as `leaf(Int)`, where `Int` is an integer. That is, leaf nodes have integer labels. For example, `leaf(1)` and `leaf(5)` are leaf nodes, and therefore small binary trees. Given two binary trees `B1` and `B2` we can combine them into one binary tree using a predicate `tree/2` as follows: `tree(B1,B2)`.

15 points

So, from the leafs `leaf(1)` and `leaf(5)` we can build the binary tree `tree(leaf(1),leaf(5))`, from the binary trees `tree(leaf(1),leaf(5))` and `leaf(3)` we can build the binary tree `tree(tree(leaf(1),leaf(5)),leaf(3))`, from the binary trees `tree(leaf(1),leaf(5))` and `tree(leaf(7),leaf(8))` we can build the binary tree `tree(tree(leaf(1),leaf(5)),tree(leaf(7),leaf(8)))`, etc.

- (a) (15 points) Write a (set of) clause(s) that define the predicate `sumTree(Tree,Sum)` such that `Sum` is the sum of the labels of the leafs of the tree `Tree`. For example, the query `sumTree(tree(tree(leaf(1),leaf(5)),leaf(3)),X)` should yield `X = 9`.



10 points

## Question 3

This assignment concerns Prolog. Assume we have a database of results of Unreal Tournament matches played by various teams. The pairings of which team played against which other were not arranged in any systematic way, so each team just played some other team(s). The results are in the program represented as facts like `won(team1,team2)`, `won(team3,team1)`, `won(team4,team2)`, where `won(T1,T2)` represents that team T1 won the match (and team T2 lost). In this assignment, we consider a set of clauses that should define a predicate `class(Team,Category)` that ranks teams into categories. We have just three categories:

- **great**: every team that won all its games
- **ok**: any team that won some games and lost some
- **improvable**: any team that lost all its games

For example, given the results stated above, team3 and team4 are great, team1 is ok, and team2 is improvable.

The following knowledge base is a *faulty* implementation of the predicate `class/2`.

```
won(team1,team2).
won(team3,team1).
won(team4,team2).
```

```
class(X,ok) :- won(X,_), won(_,X).
class(X,great) :- not(won(_,X)), won(X,_).
class(X,improvable) :- won(_,X), not(won(X,_)).
```

The query `class(X,great)` should yield `X = team3` and `X = team4`, but this knowledge base yields `false`. The following trace is produced:

```
[trace] ?- class(X,great).
  Call: (7) class(_G335, great) ? creep
  ~ Call: (8) not(won(_G403, _G335)) ? creep
    Call: (9) won(_G403, _G335) ? creep
    Exit: (9) won(team1, team2) ? creep
  ~ Fail: (8) not(won(_G403, _G335)) ? creep
    Redo: (7) class(_G335, great) ? creep
false.
```

The queries `class(team3,great)` and `class(team4,great)` do correctly yield `true`.

- (5 points) Explain why this program yields the wrong answer when posing the query `class(X,great)`.
- (5 points) Propose a modification of the program to yield the correct solution.

10 points

## Question 4

This question concerns the agent programming language GOAL.

- (5 points) Explain the difference between the goal(`<fact>`), `a-goal(<fact>)`, and `goal-a(<fact>)` operators.
- (5 points) Explain the difference between executing a rule of the form `if bel(p(X)) then insert(r(X))` in the program section and in the perceptrule section.



## Question 5

This question concerns the agent programming language GOAL.

25 points

```

main: deliveryAgent {
  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)).
    packed :- setOf(P,in(P,truck),L), length(L,X), X>=2.
    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,a). loc(truck,a).
    order(c1,[p1,p2]). order(c2,[p3,p4]).
    loc(c1,b). loc(c2,c).
  }
  goals {
    delivered_order(c1). delivered_order(c2).
  }
  program {
    if bel(home(X)) then goto(X).

    module deliverOrder {
      context { goal(delivered_order(C)) }
      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(empty, home(Y)) then goto(Y).
      }
    }
  }
  actionspec{
    load(P){
      pre{ not(packed), loc(truck,X), loc(P,X) }
      post{ in(P,truck), not(loc(P,X)) }
    }
    goto(Y){
      pre{ loc(truck,X), not(X=Y) }
      post{ loc(truck,Y), not(loc(truck,X))}
    }
    unload(P){
      ...
    }
  }
}

```



- (a) (10 points) Explain which actions the GOAL agent may perform *next*, given the agent program listed above.
- (b) (10 points) Complete the action specification for the action `unload(P)` for unloading a package `P`. Only use the predicates that are already available in the given agent program.
- (c) (5 points) Explain which goals will be part of the attention set if the module is activated given the mental state that is part of the agent program above.

25 points

## Question 6

This question concerns the agent programming language GOAL.

```

main: coffeeMaker {
  knowledge {
    requiredFor(coffee, water).
    requiredFor(coffee, grounds).
    requiredFor(espresso, coffee).
    requiredFor(grinds, beans).

    canMakeIt(M, P) :- canMake(M, Prods), member(P, Prods).
  }
  beliefs {
    have(water). have(beans).
    canMake(maker, [coffee, espresso]).
  }
  goals {
    have(coffee).
  }
  program {
    % if we need to make something, then make it (the action's precondition
    % checks if we have what it takes, literally)
    if goal(have(P)) then make(P).
  }
  actionspec {
    make(Prod) {
      pre { forall(requiredFor(Prod, Req), have(Req)) }
      post { have(Prod) }
    }
  }
  perceptrules {
    ...
  }
}

```

- (a) (15 points) Provide a rule that will make the agent above ask (send a question) any other agent what it can make, but only asks this if the agent does not already know what the other agent can make. Only use the predicates that are already available in the given agent program.
- (b) (5 points) Explain in which section you would insert the rule you provided.
- (c) (5 points) Explain the difference between the three possible moods of a message. That is, explain the difference between an agent `agent1` that performs `send(agent2, :<fact>)`, `send(agent2, ?<fact>)`, and `send(agent2, !<fact>)`.

End of exam



