Logisch Programmeren en Zoektechnieken: Tentamen A **ANSWERS**
25 September 2013

Instructions: This exam paper consists of four questions. You may answer them in Dutch or in English. Only write in the answer booklets provided and mark each booklet with your name and student number. Give short and precise answers. Write clearly. The maximum time allowed is 120 minutes.

Question 1

(a) For each of the following built-in Prolog predicates, give a short explanation of their functionality: 10 marks

- `append/3`
- `select/3`
- `reverse/2`

Support each of your explanations with an example.

Answer:

See Lecture Notes.

(b) Some of the standard predicates for list manipulation can easily be defined in terms of other such predicates. For example, we can define `last/2` in terms of `append/3`: 10 marks

```
last(List, Last) :- append(_, [Last], List).
```

In each of the following cases, provide a solution in terms of a single clause and without the (explicit) use of recursion:

- Define `last/2` in terms of `reverse/2`.

  Answer:

  ```
  last(List, Last) :- reverse(List, [Last|_]).
  ```

- Define `select/3` in terms of `append/3`.

  Answer:

  ```
  select(Elem, List, Rest) :-
    append(Front, [Elem|End], List),
    append(Front, End, Rest).
  ```

- Define `member/2` in terms of `select/3`.

  Answer:

  ```
  member(Elem, List) :- select(Elem, List, _).
  ```

(c) For each of the following queries, give a short explanation for the answer returned by Prolog (as shown below): 5 marks

(*i*)
```
?- [cat, dog, horse, cow] = [H|T].
H = cat,
T = [dog, horse, cow]
Yes
```

(*ii*)
```
?- [cat, dog, horse, cow] == [H|T].
No
```

(*iii*)
```
?- [cat, dog, horse, cow] =:= [H|T].
ERROR
```

(*iv*)
```
?- [cat, dog, horse, cow] =.. [H|T].
H = '.',
T = [cat, [dog, horse, cow]]
Yes
```

Answer:
In the first example, we *match* the list on the left with the head/tail-pattern on the right. In the second, we ask whether the two terms are *identical*, which they are not. In the third, we check whether the two terms are *arithmetically equivalent*, which causes an error as they cannot be evaluated as arithmetic expressions. In the fourth example, finally, we treat the list on the lefthand side as a general term and match `H` with its functor and `T` with its list of arguments. To be able to process this request, Prolog has to resort to the internal notation for lists, under which `[cat,dog,horse,cow]` is the same as a compound term with functor `'.'` and two arguments, the first of which is the head of the list and the second of which is its tail.

Question 2

(a) Write a Prolog predicate `squares/2` that, when provided with a list of numbers in the first argument position, will return the list of the squares of those numbers in the second argument position. Example: [10 marks]

```
?- squares([7, 20, 1, 101, 3], Result).
Result = [49, 400, 1, 10201, 9]
Yes
```

Answer:
```
squares([], []).
squares([Head|Tail], [SqHead|SqTail]) :-
  SqHead is Head * Head,
  squares(Tail, SqTail).
```

(b) Write a Prolog predicate `triangle/1` to print a triangle of a given size, as in the following two examples: [10 marks]

```
?- triangle(5).
x x x x x
  x x x x
    x x x
      x x
        x
Yes
```

```
?- triangle(3).
x x x
  x x
    x
Yes
```

Answer:

```
triangle(N) :- triangle(N, N).
triangle(0, _).
triangle(K, N) :-
  K > 0,
  White is N - K,
  line(White, '  '), line(K, 'x '), nl,
  K1 is K - 1,
  triangle(K1, N).

line(0, _).
line(N, Symbol) :- N > 0, write(Symbol), N1 is N - 1, line(N1, Symbol).
```

(c) For each of the following queries, say whether it would succeed, fail, or result in an error message. In case of success also give the (first) answer to the query. [5 marks]

- `?- X = (Y is 5 + 7).`

  Answer:

  ```
  ?- X = (Y is 5 + 7).
  X = (Y is 5+7)
  Yes
  ```

- `?- X is (Y = 5 + 7).`

  Answer:

  ```
  ?- X is (Y = 5 + 7).
  ERROR
  ```

- `?- length(X, 3), append(X, [Y|_], [1,2,3,4,5]).`

  Answer:

  ```
  ?- length(X, 3), append(X, [Y|_], [1,2,3,4,5]).
  X = [1, 2, 3],
  Y = 4
  Yes
  ```

- `?- findall(17, fail, X).`

  Answer:

  ```
  ?- findall(17, fail, X).
  X = []
  Yes
  ```

- `?- 2 * 3 + 4 =.. X.`

  Answer:

  ```
  ?- 2 * 3 + 4 =.. X.
  X = [+, 2*3, 4]
  Yes
  ```

Question 3

(a) In the context of operator definitions in Prolog, explain the meaning of the `'x'` and `'y'` in associativity patterns such as `yfx`. Also explain why the pattern `yfy` is not available in Prolog. [5 marks]

Answer:

See Lecture Notes.

(b) Provide operator definitions that ensure that the three queries below all succeed: [10 marks]

```
?- 5 alpha 6 alpha 7 = alpha(5, _).
?- 1 alpha 2 alpha 3 beta 4 = (1 alpha 2 alpha 3) beta 4.
?- gamma(_) = gamma gamma 8 beta 9.
```

Briefly explain your solution.

Answer:

```
:- op(300, xfy, alpha),
   op(400, xfx, beta),
   op(500, fy, gamma).
```

Due to the first query, `alpha` has to be a *right-associative* infix operator. Due to the second query, the precedence value of `beta` needs to be *higher* than that of `alpha`. Due to the third query, `gamma` needs to be an *associative* prefix operator with a precedence value that is *higher* than that of `beta`.

(c) Write a Prolog predicate `mainop/2` to extract the principal operator (or functor) of a given term. Do not use the built-in predicate `functor/2`. Examples: 5 marks

```
?- mainop(25 * 3 + 16, F).
F = (+)
Yes

?- mainop(bigger(elephant,tiger), F).
F = bigger
Yes
```

Answer:

```
mainop(Term, Op) :- Term =.. [Op|_].
```

(d) Explain how you can implement an operator for *negation as failure* in terms of a *cut* (`!`). Provide both a suitable operator definition and a definition of the corresponding predicate. 5 marks

Answer:

See Lecture Notes.

Question 4

(a) Explain the term *matching* as used in the context of logic programming. 5 marks

Answer:

See Lecture Notes.

(b) Suppose we use the atom `one` to represent the number 1, the term `succ(one)` to represent the number 2 ("the successor of 1"), the term `succ(succ(one))` to represent the number 3 ("the successor of the successor of 1"), and so forth. Write a Prolog predicate `translate/2` to translate such successor-expressions into the corresponding numbers. Examples: 10 marks

```
?- translate(succ(succ(succ(succ(one)))), Result).
Result = 5
Yes

?- translate(one, Result).
Result = 1
Yes
```

Answer:

```
translate(one, 1).
translate(succ(Expression), Number) :-
  translate(Expression, Predecessor),
  Number is Predecessor + 1.
```

(c) Consider the following Prolog program: 10 marks

```
p([], []).
p(List, [X|Perm]) :- select(X, List, Rest), p(Rest, Perm).

o([]).
o([_]).
o([A,B|Rest]) :- A =< B, o([B|Rest]).

s(List, SList) :- p(List, SList), o(SList).
```

The predicate `s/2` can be used to sort a given list of numbers:

```
?- s([7, 3, 2, 8, 1, 4, 6, 5], Result).
Result = [1, 2, 3, 4, 5, 6, 7, 8]
Yes
```

Explain how this program solves the problem of devising a general method for sorting a given list of numbers. What is the overall idea underlying the solution given? What is the role of the predicates `p/2` and `o/1`? When you explain the program, make sure you include at least a brief comment on each of its clauses.

Answer:

The predicate `p/2` succeeds if the second argument can be instantiated with a permutation of the list given in the first argument position (see Lecture Notes). The predicate `o/1` succeeds if the list of numbers provided is in ascending order (see Bratko). The overall idea is to use backtracking to generate all permutations of the input list, until one of them is found to be ordered. (This should be followed by a brief explanation of how the three predicates are implemented.) Note that this is a very slow method for sorting lists of numbers; it is only usable for lists of length up to about 10.