Prof. Dr. Jürgen Giesl
Jera Hensel

Exercise 1 (Programming in Prolog): (1 + 1 + 2 + 2 + 2 + 4 = 12 points)

Important: In addition to handing in the solution on paper, please also mail your solutions for this exercise to **lp17-hiwis@i2.informatik.rwth-aachen.de**. Make sure to send this email from an address that ends with "`rwth-aachen.de`" (otherwise we will not receive your email). Indicate your immatriculation numbers in the subject of the mail and inside the Prolog file.

In this exercise we investigate algorithms on lists in Prolog.

Lists

Lists in Prolog are represented by terms built over the signature $\Sigma = \Sigma_0 \cup \Sigma_2$ with $\Sigma_0 = \{\texttt{[]}\}$ and $\Sigma_2 = \{\texttt{.}\}$. The symbol `[]` denotes the empty list, while the term `.(X,XS)` denotes the list starting with the element `X` (called the head of the list) and having the list `XS` as the remaining list (called the tail of the list). Thus, a list in Prolog containing the three numbers 2, 3 and 5 would be written as `.(2,.(3,.(5,[])))`. This is the standard representation internally used by Prolog.
However, Prolog also knows a more comfortable way to write lists. The term `.(X,XS)` can also be written as `[X|XS]`. With this representation, the above list is written as `[2|[3|[5|[]]]]`. To save brackets, the representation can be shortened by just enumerating elements in the order they appear in the list: `[2,3,5]`. Equivalently, one can also write `[2,3|[5]]`. While this list representation is easier to use for humans, it is equivalent to the internal representation used in Prolog. You can use both representations and even mix them within one Prolog program.
As an example for an algorithm working on lists, we write a predicate `hasLast/2` (i.e., a predicate `hasLast` of arity 2) in Prolog where `hasLast(XS, X)` is true iff `X` is the last element of the list `XS`.

```
hasLast([X], X).
hasLast([X|XS], Y) :- hasLast(XS, Y).
```

The following solution is also correct, but uses the less readable list representation.

```
hasLast(.(X, []), X).
hasLast(.(X, XS), Y) :- hasLast(XS, Y).
```

To avoid programming with built-in predicates, we use a special representation for natural numbers. The number 0 is just represented as the constant `0/0`, while each other natural number is represented as a successor of another natural number using the symbol `s/1`. Thus, the number 2 is represented as `s(s(0))`.

You may not use any predefined Prolog predicates in this exercise (unless allowed explicitly)! However, you can define your own auxiliary predicates. In all subexercises, you may use the predicates of previous subexercises, even if you have not implemented them.

a) Implement the predicates `smaller/2` and `smallerEqual/2` in Prolog. For two numbers `X` and `Y` of the form `s(...s(0)...)`, `smaller` determines if `X` is smaller than `Y` and `smallerEqual` determines if `X` is smaller than or equal to `Y`.

b) Implement a predicate `isSorted/1` in Prolog which for a list of numbers of the form `s(...s(0)...)` determines whether the numbers are sorted in non-descending order. For example, the query `isSorted([0,s(0),s(0),s(s(s(0)))])` should yield the answer `true`.

c) Implement a predicate `bubble/2` in Prolog. For a call of `bubble(`l`,X)`, where l is a list of numbers, each pair of adjacent numbers is compared beginning from the start of the list. If the first number is greater than the second number, the numbers are swapped. Otherwise, the second number is compared to its successor, etc. For example, the query `bubble([s(0),s(s(0)),s(0),0],X)` should return the only answer `X = [s(0),s(0),0,s(s(0))]`, which results from first swapping `s(s(0))` and `s(0)`, and then swapping `s(s(0))` and `0`.

d) Implement a predicate `bubbleSort/2` which sorts the list given as its first argument using the predicate `bubble`. For example, the query `bubbleSort([s(0),s(s(0)),s(0),0],X)` should return the answer `X = [0,s(0),s(0),s(s(0))]`.

Hints:

- To test if `X` and `Y` are different, you may use the predefined predicate `\=`. Example: `[s(0),0] \= [0,s(0)]` is true. In this way you can restrict the number of answers given by Prolog to one.

e) Implement a predicate `duplicates/1` which for a list of numbers determines whether there is at least one number that occurs more than once. For example, the query `duplicates([s(0),s(s(0)),s(0),0])` should yield the answer `true`.

f) Implement a predicate `countNumbers/2` in Prolog which is true iff the first argument is a list of numbers and the second argument is a list whose ith element states how often the number i occurs in the first list. Here, the head of a list is the 0th element of the list. For example, the query

`countNumbers([s(0),s(s(s(s(0)))),s(0),0],X)`

should yield the answer `X = [s(0),s(s(0)),0,0,s(0)]`, because `0` occurs once in the input list, `s(0)` occurs twice in the input list, `s(s(0))` and `s(s(s(0)))` do not occur at all in the input list, and `s(s(s(s(0))))` occurs once in the input list.

Solution:

```
% a)
smaller(0,s(_)).
smaller(s(X),s(Y)) :- smaller(X,Y).

smallerEqual(X,X).
smallerEqual(X,Y) :- smaller(X,Y).

% alternatively:
% smallerEqual(0,_).
% smallerEqual(s(X),s(Y)) :- smallerEqual(X,Y).

% b)
isSorted([]).
isSorted([_]).
isSorted([X,Y|XS]) :- smallerEqual(X,Y), isSorted([Y|XS]).

% c)
bubble([],[]).
bubble([X],[X]).
bubble([X,Y|XS],[X|YS]) :- smallerEqual(X,Y), bubble([Y|XS],YS).
bubble([X,Y|XS],[Y|YS]) :- smaller(Y,X), bubble([X|XS],YS).

% d)
bubbleSort(XS,XS) :- isSorted(XS).
bubbleSort(XS,ZS) :- bubble(XS,YS), XS \= YS, bubbleSort(YS,ZS).

% alternatively:
% bubbleSort(XS,XS) :- bubble(XS,XS).
% bubbleSort(XS,ZS) :- bubble(XS,YS), XS \= YS, bubbleSort(YS,ZS).

% e)
duplicates(XS) :- bubbleSort(XS,YS), dupl(YS).

dupl([X,X|_]).
dupl([_,Y|XS]) :- dupl([Y|XS]).
```

```
% f)
countNumbers(XS,AS) :- bubbleSort(XS,YS), count(YS,AS,0).

count([],[],_).
count([I],[s(0)],I).
count([I|XS],[s(A)|AS],I) :- count(XS,[A|AS],I).
count([X|XS],[0|AS],I) :- smaller(I,X), count([X|XS],AS,s(I)).
```

Exercise 2 (Skolem Normal Form): (2 + 2 + 2 = 6 points)

Let

$$\begin{aligned}\varphi = &\forall X\ \mathsf{p}(X,X)\\ &\wedge (\forall X,Y\ \mathsf{p}(X,Y) \rightarrow \mathsf{p}(X,\mathsf{f}(Y)))\\ &\wedge \neg\forall X\ \mathsf{p}(\mathsf{a},X)\end{aligned}$$

be a formula over the signature (Σ,Δ) with $\Sigma = \Sigma_0 \cup \Sigma_1$, $\Sigma_0 = \{\mathsf{a}\}, \Sigma_1 = \{\mathsf{f}\}$, $\Delta = \Delta_2 = \{\mathsf{p}\}$.

a) Prove that φ is satisfiable. (Hint: It suffices to choose a carrier with at most two elements.)

b) Transform φ into a satisfiability-equivalent formula ψ in Skolem normal form.

c) Are φ and ψ equivalent (cf. Definition 2.2.1)? Explain your answer.

Solution:

a) $S = (\mathcal{A},\alpha)$ with $\mathcal{A} = \{0,\bigstar\}$, $\alpha_{\mathsf{a}} = 0, \alpha_{\mathsf{f}}(0) = 0, \alpha_{\mathsf{f}}(\bigstar) = \bigstar$, and $\alpha_{\mathsf{p}} = \{(0,0),(\bigstar,\bigstar)\}$.

b) We first transform φ into Prenex normal form:

$$\varphi' = \forall X_1,X_2,Y\ \exists X_3\ \mathsf{p}(X_1,X_1) \wedge (\neg\mathsf{p}(X_2,Y) \vee \mathsf{p}(X_2,\mathsf{f}(Y))) \wedge \neg\mathsf{p}(\mathsf{a},X_3)$$

Now we transform φ' to Skolem normal form (where g is a new function symbol with arity 3):

$$\psi = \forall X_1,X_2,Y\ \mathsf{p}(X_1,X_1) \wedge (\neg\mathsf{p}(X_2,Y) \vee \mathsf{p}(X_2,\mathsf{f}(Y))) \wedge \neg\mathsf{p}(\mathsf{a},\mathsf{g}(X_1,X_2,Y))$$

By reordering the subformulas of φ, the last conjunct could also be $\neg\mathsf{p}(\mathsf{a},\mathsf{g})$, where g is a function symbol with arity 0.

c) Consider the structure $S' = (\mathcal{A}',\alpha')$ with $\mathcal{A}' = \{0,\bigstar\}$ and

- $\alpha'_{\mathsf{a}} = 0$,
- $\alpha'_{\mathsf{f}}(0) = 0, \alpha'_{\mathsf{f}}(\bigstar) = \bigstar$,
- $\alpha'_{\mathsf{g}}(X,Y,Z) = 0$,
- $\alpha'_{\mathsf{p}} = \{(0,0),(\bigstar,\bigstar)\}$,

which we obtain by a small modification of S. Again, we have $S' \models \varphi$. However, we also have $S' \not\models \psi$ since $S' \not\models \neg\mathsf{p}(\mathsf{a},\mathsf{g}(0,0,0))$. Therefore, by Definition 2.2.1 the formulas φ and ψ are not equivalent.