Notes:

- To solve the programming exercises you can use the Prolog interpreter SWI-Prolog, available for free at http://www.swi-prolog.org. For Debian and Ubuntu it suffices to install the swi-prolog package. You can use the command "swipl" to start it and use "[exercise1]." to load the facts from the file exercise1.pl in the current directory.

- Please solve these exercises in groups of two or three!

- The solutions must be handed in directly before (very latest: at the beginning of) the exercise course on 24.04.2015, in lecture hall AH 1. Alternatively you can drop your solutions into a box which is located right next to Prof. Giesl’s office (this box is emptied a few minutes before the exercise course starts).

- Please write the names and immatriculation numbers of all students on your solution. Also please staple the individual sheets!

- Please register at https://aprove.informatik.rwth-aachen.de/lp15/ (https, not http!).

Exercise 1 (Programming in Prolog): \( 0.5 + 1 + 2 + 1.5 + 2 + 1 = 8 \) points

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 \( \Sigma_0 = \{ [] \} \) and \( \Sigma_2 = \{ . \} \). 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 \( XS \).

\[
\text{hasLast}([X], X). \\
\text{hasLast}([X|XS], Y) :- \text{hasLast}(XS, Y).
\]

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

\[
\text{hasLast}(.(X, []), X). \\
\text{hasLast}(.(X, XS), Y) :- \text{hasLast}(XS, Y).
\]

You may not use any predefined Prolog predicates in this exercise (unless allowed explicitly)! However, you can define your own auxiliary predicates.

a) Implement a predicate isNumber/1 in Prolog which determines whether its only argument is of the form \( s(...s(0)...) \), where 0 represents the number Zero and \( s(X) \) is the successor of \( X \).
b) Implement a predicate \texttt{remove/3} in Prolog. Here, \texttt{remove(X,XS,YS)} is true iff \texttt{XS} is a list containing \texttt{X} as an element and \texttt{YS} is the list resulting by removing the first occurrence of \texttt{X} from \texttt{XS}. For example, the query \texttt{remove(s(0),[s(0),0,s(0)],ZS)} should return the only answer \texttt{ZS = [0,s(0)]. The query remove(s(0),[0],ZS)} should fail.

Hints:

- To test if \texttt{X} and \texttt{Y} are different, you may use the predefined predicate \texttt{\neq}. Example: \texttt{a \neq b} is true.

c) Implement a predicate \texttt{minimum/2} which computes the minimum of a list (specified in the first argument) as its second argument. If the list in the first argument is empty or contains any element not being a number as specified in \texttt{a}), the predicate should fail. For example, the query \texttt{minimum([s(0),s(s(0)),s(0)],X)} should return the answer \texttt{X = s(0)}.

Hints:

- Make use of predicates you already defined, for example the predicate \texttt{isNumber} to test if an argument is a number.
- You may use the predicate \texttt{smaller/2}, which you can find in Exercise 3.

d) Use the predicates \texttt{remove} and \texttt{minimum} to implement a predicate \texttt{selectionSort/2} which sorts the list specified in the first argument. Here, \texttt{selectionSort(XS,YS)} is true iff \texttt{YS} contains exactly the elements of \texttt{XS} in ascending order. For example, the query \texttt{selectionSort([s(0),s(s(0)),0,s(0)],ZS)} should return the only answer \texttt{ZS = [0,s(0),s(0),s(s(0))].}

Hints:

- Make use of predicates you already defined, for example the predicate \texttt{isNumber} to test if an argument is a number.
- You may use the predicate \texttt{smaller/2}, which you can find in Exercise 3.

e) Implement a predicate \texttt{insertSorted/3} in Prolog which inserts a natural number specified in the first argument into the list specified in the second argument. Here, for a sorted lists \texttt{XS, insertSorted(X, XS, YS)} is true iff \texttt{YS} is the list \texttt{XS} where \texttt{X} has been inserted at the correct position such that \texttt{YS} is sorted again. For example, the query \texttt{insertSorted(s(0),[0,s(0),s(s(0))],X)} should return the only answer \texttt{X = [0,s(0),s(0),s(s(0))]. If insertSorted’s first argument is not a natural number, the query should fail.}

Hints:

- Make use of predicates you already defined, for example the predicate \texttt{isNumber} to test if an argument is a number.
- You may use the predicate \texttt{smaller/2}, which you can find in Exercise 3.

f) Use the predicate \texttt{insertSorted} to implement a predicate \texttt{insertionSort/2} in Prolog which sorts the list specified in the first argument. Here, \texttt{insertionSort(XS,YS)} is true iff \texttt{YS} contains exactly the elements of \texttt{XS} in ascending order. For example, the query \texttt{insertionSort([s(0),s(s(0)),0,s(0)],X)} should return the only answer \texttt{X = [0,s(0),s(0),s(s(0))].}

Exercise 2 (Herbrand model): \hspace{1cm} (2 + 2 + 1.5 + 1.5 + 2 = 9 points)

Let

\[
\varphi = p(a) \\
\land \forall X \ p(X) \rightarrow q(X, f(X)) \\
\land \forall X, Y \ q(X, Y) \rightarrow p(Y) \\
\land \exists X \ \neg p(X)
\]

be a formula over the signature \((\Sigma, \Delta)\) with \(\Sigma = \Sigma_0 \cup \Sigma_1, \Sigma_0 = \{a\}, \Sigma_1 = \{f\}, \Delta = \Delta_1 \cup \Delta_2, \Delta_1 = \{p\} \text{ and } \Delta_2 = \{q\}.

Hints:

- You can use \(f^t(a)\) as an abbreviation for \(f(\ldots(f(a))\ldots).\)

\[
\text{t times}
\]
a) Prove that \( \varphi \) is satisfiable. (Hint: It suffices to choose a carrier with at most two elements.)
b) Give a Herbrand model for \( \varphi \) or show why no such model exists.
c) Transform \( \varphi \) into a satisfiability-equivalent formula \( \psi \) in Skolem normal form.
d) Give a Herbrand model for \( \psi \) or show why no such model exists.
e) Are \( \varphi \) and \( \psi \) equivalent (cf. Definition 2.2.1)? Explain your answer.

Exercise 3 (Gilmore's algorithm): (5 points)

Consider the following logic program

\[
\text{smaller}(0, s(X)). \\
\text{smaller}(s(X), s(Y)) :\neg \text{smaller}(X, Y).
\]

and the query

\[ ? - \text{smaller}(s(0), s(s(s(0)))) \]

Use Gilmore’s algorithm to show that the formulas \( \varphi_1 \) and \( \varphi_2 \) corresponding to the logic program entail the formula \( \varphi \) corresponding to the query (i.e., \( \{\varphi_1, \varphi_2\} \models \varphi \)).