Number Sets in Prolog

Course: Advanced Programming, Block 1, 2012

Deadline: 16:00, October 9, 2012

For the purpose of this exam, the natural numbers are defined to be the non-negative integers, i.e., including zero. A natural number n can be (somewhat inefficiently) represented in pure Prolog as the term $\underbrace{\texttt{s}(...(\texttt{s}(\texttt{z})))}_{n}$; for example, the number 3 is represented as the term `s(s(s(z)))`.

A finite set of natural numbers $s = \{n_1, \ldots, n_k\}$, where $k \geq 0$ and $n_1 < n_2 < \cdots < n_k$, can be represented as the Prolog list `[`t_1`, ..., `t_k`]`, where each t_i represents n_i. (The requirement that the list is sorted and without duplicates ensures that every finite set of naturals has exactly one representation.) For example the set $\{3, 1\}$ (which is the same as $\{1, 3\}$) is uniquely represented as `[s(z), s(s(s(z)))]`.

1. Write a predicate `less/2`, such that if t_1 represents n_1 and t_2 represents n_2, then `less(`t_1`, `t_2`)` succeeds iff $n_1 < n_2$. For example, `less(s(z), s(s(s(z))))` should succeed, and `less(s(z), s(z))` should fail. If t_1 is an uninstantiated logic variable, while t_2 is a proper representation of a natural number n_2, then the predicate should enumerate all numbers less than n_2. For example, the query "`?- less(X, s(s(z)))`." should succeed with `X = z` and `X = s(z)` (not necessarily in that order).

2. Write a predicate `checkset/1`, such that `checkset(`t`)` for a fully instantiated term t succeeds iff t is a correct representation of some natural-number set. For example `checkset([z, s(s(z))])` should succeed, while `checkset([z,z])` or `checkset(z)` should fail. The behavior of `checkset` on arguments containing unbound logic variables doesn't matter.

3. Write a predicate `ismember/3`, such that if t_1 represents a number n, and t_2 represents a set s, then `ismember(`t_1`, `t_2`, `t_3`)` succeeds with $t_3 = $ `yes` iff $n \in s$, and with $t_3 = $ `no` iff $n \notin s$. If t_1 does not represent a number, and/or t_2 does not represent a set, the behavior if `ismember` is not constrained.

   Additionally, document and *explain* the behavior of your code on the specific query, "`?- ismember(N, [s(z), s(s(s(z)))], A)`.".

4. Write a predicate `union/3`, such that if t_1 represents s_1, t_2 represents s_2, and t_3 represents s_3, then `union(`t_1`, `t_2`, `t_3`)` succeeds iff $s_3 = s_1 \cup s_2$. If t_1 and t_2 are instantiated but t_3 is a logic variable, the predicate should compute t_3; for example, the query "`?- union([z], [s(z)], A)`." should succeed with `A = [z, s(z)]`. If

t_1 and/or t_2 are logic variables, but t_3 is instantiated, the predicate should enumerate all possible solutions; for example, "?- `union(X, [s(z)], [z,s(z)])`." should succeed with `X = [z]` and `X = [z,s(z)]` (not necessarily in that order).

5. Write a predicate `intersection/3`, such that if t_1 represents s_1, t_2 represents s_2, and t_3 represents s_3, then `intersection($t_1$, $t_2$, $t_3$)` succeeds iff $s_3 = s_1 \cap s_2$. If t_1 and t_2 are instantiated but t_3 is a logic variable, the predicate should compute t_3; for example "?- `intersection([z], [s(z)], A)`." should succeed with `A = []`. (But unlike for `union`, there are no constraints on what the predicate should do if t_1 and/or t_2 are uninstantiated variables.)

Your program must consist of pure facts and rules only; do not use any built-in Prolog predicates or control operators (including in particular, but not limited to, cuts "`!`" and negation-as-failure "`\+`"). If you need any auxiliary predicates like `append`, include their definitions in the program, but be sure to name them something different from the built-in ones.

Though not a strict requirement for this assignment, it is desirable that your predicates succeed only once with each possible answer, and fail explicitly (rather than going into an infinite loop) after enumerating all answers. This both simplifies testing and makes it easier to argue that the code is correct.

Remember to hand in both your Prolog code and a report explaining your solutions and assessing their quality.