Prolog exercises

Natural numbers and trees

1. Like in Haskell (cf. the warm-up exercise for Assignment 0), we can work with a unary representation of natural numbers in Prolog. Specifically, we will represent a number *n* as the term `s(...(s(z)))` with a total of *n* nested applications of `s` to `z`; for example, the number 0 would correspond to just `z`, while 3 would be `s(s(s(z)))`. With this representation, implement the following Prolog predicates:
   - **add(N_1, N_2, N)**, which expresses that N is the sum of N_1 and N_2. For example, the query `?- add(s(s(z)), s(z), R).` should succeed (exactly once) with R instantiated to `s(s(s(z)))`. It is desirable, but not required, that the query `?- add(X, Y, s(s(s(z)))).` would enumerate all the possible ways of obtaining 3 as a sum of two natural numbers X and Y.
   - **mult(N_1, N_2, N)**, which analogously expresses that N is the product of N_1 and N_2. Do not worry about running this predicate in reverse.
   - **comp(N_1, N_2, A)**, which compares the natural numbers N_1 and N_2, and succeeds where A is one of the atoms `lt`, `eq`, or `gt`, depending on whether N_1 is less than, equal to, or greater than N_2, respectively. For example, `?- comp(s(z), z, R).` should succeed (exactly once) with `R = gt`.
2. We can also encode binary search trees in Prolog, with a search tree being either the atom `leaf`, or the term `node(N, T1, T2)`, where N is a number (represented as above), and T_1 and T_2 are themselves search trees, such that all the numbers in T_1 are strictly less than N, and those in T_2 are strictly greater than N. Implement the following two predicates:
   - **insert(N, T_I, T_O)**, which says that T_O is the result of inserting the number N into the tree T_I, at the appropriate leaf position. For example, the query `?- insert(s(s(z)), node(s(z), leaf, leaf), T).` should succeed with `T = node(s(z), leaf, node(s(s(z)), leaf, leaf)).` If N is already present in T_I, T_O should be the same as T_I.
   - **insertlist(Ns, T_I, T_O)**, which says that T_O is the result of successively inserting all the numbers from the Prolog list Ns into T_I. For example, `?- insertlist([s(z), s(s(z))], leaf, T).` should succeed with the same T as above.

For all your predicate definitions, use **only pure Prolog**, i.e., no built-in predicates or control operators – including in particular, but not limited to, `is`, `\+`, or `!`. You may assume that all input arguments to the predicates are fully instantiated and of the appropriate type (i.e., using only the allowed term constructors), which – unlike in Haskell – we cannot ensure statically in Prolog.

Further exercises

You will find a collection of additional exercises here (introductory) and here (more advanced). Where possible, try to also stick to pure Prolog in your solutions for those.