Recall that a non-negative integer $n$ can be encoded (in unary) as $n$ succ’s applied to 0

\[
\text{numeral}(0).
\]

\[
\text{numeral}(\text{succ}(X)) :- \text{numeral}(X).
\]

In general, Prolog terms constructed from a constant \texttt{null} and $k + 1$ unary functors $f_0, \ldots, f_k$ amount to strings over an alphabet \{a_0, \ldots, a_k\}, with \texttt{null} encoding the empty string, and $f_i(\texttt{null})$ encoding the string $a_i$. For strings of length $> 1$, it will prove useful (for some purposes such as the arithmetic encoding below) to encode the string in reverse, representing, for example, $a_2 a_3$ as $f_3(f_2(\texttt{null}))$, rather than $f_2(f_3(\texttt{null}))$.

To simplify notation, let us work with the alphabet \{0, 1\} (with $k = 1, a_0 = 0, a_1 = 1$) and unary functors $f_0, f_1$. Let “pterm” abbreviate “Prolog term built from \texttt{null}, $f_0$, $f_1$” as described by the clauses

\[
\text{pterm}(\texttt{null}).
\]

\[
\text{pterm}(f_0(X)) :- \text{pterm}(X).
\]

\[
\text{pterm}(f_1(X)) :- \text{pterm}(X).
\]

Putting numbers in binary form,

$\dagger$ 0 becomes the bitstring 0 and pterm $f_0(\texttt{null})$,

1 becomes the bitstring 1 and pterm $f_1(\texttt{null})$,

2 becomes the bitstring 10 and pterm $f_0(f_1(\texttt{null}))$,

3 becomes the bitstring 11 and pterm $f_1(f_1(\texttt{null}))$,

\vdots

Note that pterms such as \texttt{null} and $f_1(f_1(f_0(\texttt{null})))$ are excluded from $\dagger$, even though we can associate non-negative integers (0 and 3) with them.

\textbf{N.B.} In solving the problems below, you are banned from using built-in arithmetic predicates or lists in Prolog.

\footnote{Due Oct 16 (Tuesday): demonstrate during lab (Mon 4-5, Tues 2-3) or, failing that, submit to Blackboard. For any extensions beyond Oct 16, email your demonstrator, David Woods (dwoods@tcd.ie). E-mail submissions to Tim Fernando will receive an F3.}
Problem 1. Define a predicate \(\text{incr}(P_1, P_2)\) over pterms \(P_1\) and \(P_2\) such that under \(\dagger\), \(P_2\) is the successor of \(P_1\). For example, as 3 is 2+1,

\[
| \neg \text{incr}(f0(f1(null)), X). \\
X = f1(f1(null)) ; \\
no
\]

You are free to define \(\text{incr}\) such that \(\text{incr}(null, X)\) holds for no \(X\) or else only for \(X=f1(null)\). Likewise for \(\text{incr}(P, X)\) where \(P\) is some other pterm not in \(\dagger\).

Problem 2. Define a predicate \(\text{legal}(P)\) true exactly of pterms \(P\) mentioned by \(\dagger\). Hence,

\[
| \neg \text{legal}(X). \\
X = f0(null) ; \\
X = f1(null) ; \\
X = f0(f1(null)) ; \\
X = f1(f1(null)) ; \\
X = f0(f0(f1(null))) ; \\
X = f1(f0(f1(null))) ; \\
... \\
\]

Using \(\text{legal}\), revise your predicate \(\text{incr}\) to \(\text{incrR}\) such that

\[
| \neg \text{incrR}(X, Y). \\
X = f0(null), Y = f1(null) ; \\
X = f1(null), Y = f0(f1(null)) ; \\
X = f0(f1(null)), Y = f1(f1(null)) ; \\
X = f1(f1(null)), Y = f0(f0(f1(null))) ; \\
X = f0(f0(f1(null))), Y = f1(f0(f1(null))) ; \\
X = f1(f0(f1(null))), Y = f0(f1(f1(null))) ; \\
... \\
\]

Problem 3. Define a predicate \(\text{add}(P_1, P_2, P_3)\) over pterms \(P_1, P_2\) and \(P_3\) such that under \(\dagger\), \(P_3\) is \(P_1\) plus \(P_2\). For example, as 3 is 1+2,

\[
| \neg \text{add}(f1(null), f0(f1(null)), X). \\
X = f1(f1(null)) ; \\
no
\]
Problem 4. Define a predicate \( \text{mult}(P_1, P_2, P_3) \) over pterms \( P_1, P_2 \) and \( P_3 \) such that under \( (†) \), \( P_3 \) is \( P_1 \) times \( P_2 \). For example, as 2 is \( 1 \times 2 \),

\[
\begin{align*}
\text{?- mult}(f1(null), f0(f1(null)), X). \\
X &= f0(f1(null)), \\
\text{no}
\end{align*}
\]

Problem 5. Define a predicate \( \text{revers}(P, \text{RevP}) \) that takes a pterm \( P \) and reverses it to \( \text{RevP} \) so that, for example,

\[
\begin{align*}
\text{?- revers}(f0(f1(null)), X). \\
X &= f1(f0(null)), \\
\text{no}
\end{align*}
\]

Problem 6. Define a predicate \( \text{normalize}(P, P_n) \) true of pterms \( P \) and \( P_n \) such that legal\( (P_n) \) and \( P \) and \( P_n \) encode the same number, \( \text{enc}(P) = \text{enc}(P_n) \), where

\[
\begin{align*}
\text{enc}(&\text{null}) := 0 \\
\text{enc}(f0(X)) := 2 \times \text{enc}(X) \\
\text{enc}(f1(X)) := 2 \times \text{enc}(X) + 1
\end{align*}
\]

For example,

\[
\begin{align*}
\text{?- normalize}(\text{null}, X). \\
X &= f0(\text{null}), \\
\text{no} \\
\text{?- normalize}(f1(f0(\text{null}))), X). \\
X &= f1(\text{null}), \\
\text{no}
\end{align*}
\]

Feel free to use Prolog’s built-in binary predicate \( \neq \) for inequality (e.g. \( \text{null} \neq f0(null) \)).

Final note. To help your demonstrator (and yourself) test your solutions, please add the following clauses to your Prolog code.

\[
\begin{align*}
\text{%- test add inputting numbers N1 and N2} \\
\text{testAdd}(N1, N2, T1, T2, Sum, SumT) :- \text{numb2pterm}(N1, T1), \text{numb2pterm}(N2, T2), \\
\text{add}(T1, T2, SumT), \text{pterm2numb}(SumT, Sum).
\end{align*}
\]
% test mult inputting numbers N1 and N2
testMult(N1,N2,T1,T2,N1N2,T1T2) :- numb2pterm(N1,T1), numb2pterm(N2,T2),
    mult(T1,T2,T1T2), pterm2numb(T1T2,N1N2).

% test revers inputting list L
testRev(L,Lr,T,Tr) :- ptermlist(T,L), revers(T,Tr), ptermlist(Tr,Lr).

% test normalize inputting list L
testNorm(L,T,Tn,Ln) :- ptermlist(T,L), normalize(T,Tn), ptermlist(Tn,Ln).

% make a pterm T from a number N    numb2term(+N,?T)
numb2pterm(0,f0(null)).
numb2pterm(N,T) :- N>0, M is N-1, numb2pterm(M,Temp), incr(Temp,T).

% make a number N from a pterm T   pterm2numb(+T,?N)
pterm2numb(null,0).
pterm2numb(f0(X),N) :- pterm2numb(X,M), N is 2*M.
pterm2numb(f1(X),N) :- pterm2numb(X,M), N is 2*M +1.

% reversible    ptermlist(T,L)
ptermlist(null,[],).
ptermlist(f0(X),[0|L]) :- ptermlist(X,L).
ptermlist(f1(X),[1|L]) :- ptermlist(X,L).

Apart from these clauses, your program should make no use of Prolog’s built-in arithmetic and list predicates.