CSU34011 Symbolic Programming

First assessed assignment

Due 27 Oct (Tuesday): submit to Blackboard

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

```
numeral(0).
numeral(succ(X)) :- numeral(X).
```

In general, Prolog terms constructed from a constant `null` and $k+1$ unary functors $\texttt{f0}, \ldots, \texttt{f}k$ amount to strings over an alphabet $\{a_0, \ldots, a_k\}$, with `null` encoding the empty string, and $\texttt{f}i\texttt{(null)}$ 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 `f3(f2(null))`, rather than `f2(f3(null))`.

To simplify notation, let us work with the alphabet $\{0, 1\}$ (with $k = 1, a_0 = 0, a_1 = 1$) and unary functors `f0, f1`. Let "pterm" abbreviate "Prolog term built from `null, f0, f1`" as described by the clauses

```
pterm(null).
pterm(f0(X)) :- pterm(X).
pterm(f1(X)) :- pterm(X).
```

Putting numbers in binary form,

(†) 0 becomes the bitstring 0 and pterm `f0(null)`,

1 becomes the bitstring 1 and pterm `f1(null)`,

2 becomes the bitstring 10 and pterm `f0(f1(null))`,

3 becomes the bitstring 11 and pterm `f1(f1(null))`,

$\vdots$

Note that pterms such as `null` and `f1(f1(f0(f0(null))))` are excluded from (†), even though we can associate non-negative integers (0 and 3) with them.

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

Problem 1. Define a predicate `incr(P1,P2)` over pterms `P1` and P2 such that under (†), `P2` is the successor of `P1`. For example, as 3 is 2+1,

```
| ?- incr(f0(f1(null)),X).
X = f1(f1(null)) ;
no
```

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

Problem 2. Define a predicate `legal(P)` true exactly of pterms `P` mentioned by (†). Hence,

```
| ?- 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 `legal`, revise your predicate `incr` to `incrR` such that

```
?- 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 `add(P1,P2,P3)` over pterms `P1,P2` and P3 such that under (†), `P3` is `P1` plus `P2`. For example, as 3 is 1+2,

```
| ?- add(f1(null),f0(f1(null)),X).
X = f1(f1(null)) ;
no
```

Problem 4. Define a predicate `mult(P1,P2,P3)` over pterms `P1`,`P2` and `P3` such that under (†), `P3` is `P1` times `P2`. For example, as 2 is 1×2,

```
| ?- mult(f1(null),f0(f1(null)),X).
X = f0(f1(null)) ;
no
```

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

```
| ?- revers(f0(f1(null)),X).
X = f1(f0(null)) ;
no
```

Problem 6. Define a predicate `normalize(P, Pn)` true of pterms `P` and `Pn` such that `legal(Pn)` and `P` and `Pn` encode the same number, enc(`P`) = enc(`Pn`), where

$$\begin{aligned} \mathrm{enc}(\mathtt{null}) &:= 0 \\ \mathrm{enc}(\mathtt{f0}(\mathtt{X})) &:= 2 \times \mathrm{enc}(\mathtt{X}) \\ \mathrm{enc}(\mathtt{f1}(\mathtt{X})) &:= 2 \times \mathrm{enc}(\mathtt{X}) + 1. \end{aligned}$$

For example,

```
| ?- normalize(null, X).
X = f0(null) ;
no
| ?- normalize(f1(f0(f0(null))), X).
X = f1(null) ;
no
```

Feel free to use Prolog's built-in binary predicate `\=` for inequality (e.g. `null \= f0(null)`).

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

```
% test add inputting numbers N1 and N2
testAdd(N1,N2,T1,T2,Sum,SumT) :- numb2pterm(N1,T1), numb2pterm(N2,T2),
                                 add(T1,T2,SumT), pterm2numb(SumT,Sum).
```

```
% 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.