Appendix

Helper predicates for ranking and unranking balanced parentheses expressions

The predicate \texttt{cat} computes efficiently the \(n\)-th Catalan number \(\frac{1}{n+1} \binom{2n}{n}\):

\[
\text{cat}(0,1).
\text{cat}(N,R) :- N > 0, 
PN is N+1, SN is N+1, cat(PN,R1), R is 2*(2*N-1)*R1//SN.
\]

The predicate \texttt{binDif} computes the difference of two binomials.

\[
\text{binDif}(N,X,Y,R) :- N1 is 2*N-X, R1 is N - (X + Y)//2, 
R2 is R1-1, binomial(N1,R1,B1), binomial(N1,R2,B2), R is B1-B2.
\]

The predicate \texttt{localRank} computes, by binary search the rank of sequences of a given length.

\[
\text{localRank}(N,As,NewLo) :- X is 1, Y is 0, Lo is 0, 
\text{binDif}(N,0,0,Hi0), Hi is Hi0-1, 
\text{localRankLoop}(As,N,X,Y,Lo,Hi,NewLo,NewHi).
\]

After finding the appropriate range containing the rank with \texttt{binDif}, we delegate the work to the predicate \texttt{localRankLoop}.

\[
\text{localRankLoop}(As,N,X,Y,Lo,Hi,FinalLo,FinalHi) :- N2 is 2*N,X<N2,!,
PY is Y+1, SY is Y+1, nth0(X,As,A), (0=:=A-> binDif(N,X,PY,Hi1),
NewHi is Hi-Hi1, NewLo is Lo, NewY is SY ; binDif(N,X,SY,Lo1),
NewLo is Lo+Lo1, NewHi is Hi, NewY is PY),
NewX is X+1, 
\text{localRankLoop}(As,N,NewX,NewY,NewLo,NewHi,FinalLo,FinalHi).
\]

\[
\text{localRankLoop}(_As,_N,_X,_Y,Lo,Hi,Lo,Hi).
\]

\[
\text{rankLoop}(I,S,FinalS) :- I>=0,!,
cat(I,C), NewS is S+C, PI is I-1, 
\text{rankLoop}(PI,NewS,FinalS).
\]

\[
\text{rankLoop}(_,S,S).
\]

Unranking works in a similar way. The predicate \texttt{localUnrank} builds a sequence of balanced parentheses by doing binary search to locate the sequence in the enumeration of sequences of a given length.

\[
\text{localUnrank}(N,R,As) :- Y is 0, Lo is 0, binDif(N,0,0,Hi0), Hi is Hi0-1, X is 1, 
\text{localUnrankLoop}(X,Y,N,R,Lo,Hi,As).
\]

\[
\text{localUnrankLoop}(X,Y,N,R,Lo,Hi,As) :- N2 is 2*N,X<N2,!,
PY is Y-1, SY is Y+1, binDif(N,X,SY,K), LK is Lo+K, 
( R<LK -> NewHi is LK-1, NewLo is Lo, NewY is SY, Digit=0 
; NewLo is LK, NewHi is Hi, NewY is PY, Digit=1 ),
nth0(X,As,Digit), NewX is X+1, 
\]

\[
\text{localUnrankLoop}(_,Y,_,N,_,_,_,_).
\]
unrankLoop(R, S, I, FinalS, FinalI) :- cat(I, C), NewS is S + C, NewS <= R, !, NewI is I + 1, unrankLoop(R, NewS, NewI, FinalS, FinalI).
unrankLoop(_, S, I, S, I).

The bijection between finite lists and sets

The bijection list2set together with its inverse set2list are defined as follows:

list2set(Ns, Xs) :- list2set(Ns, -1, Xs).
list2set([], _, []).
list2set([N | Ns], Y, [X | Xs]) :-
    X is (N + Y) + 1, list2set(Ns, X, Xs).

set2list(Xs, Ns) :- set2list(Xs, -1, Ns).
set2list([], _, []).
set2list([X | Xs], Y, [N | Ns]) :-
    N is (X - Y) - 1, set2list(Xs, X, Ns).

The following examples illustrate this bijection:

?- list2set([2, 0, 1, 4], Set), set2list(Set, List).
Set = [2, 3, 5, 10],
List = [2, 0, 1, 4].

As a side note, this bijection is mentioned in [12] with indications that it might even go back to the early days of the theory of recursive functions.

Binomial Coefficients, efficiently

Binomial coefficients are given by the formula $\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)...(n-(k-1))}{k!}$. By performing divisions as early as possible to avoid generating excessively large intermediate results, one can derive the binomialLoop tail-recursive predicate:

binomialLoop(_, K, I, P, R) :- I >= K, !, R = P.
binomialLoop(N, K, I, P, R) :- I1 is I + 1, P1 is ((N - I) * P) // I1, binomialLoop(N, K, I1, P1, R).

The predicate binomial(N, K, R) computes $\binom{N}{K}$ and unifies the result with R.

binomial(_, N, K, R) :- K < 0, !, R = 0.
binomial(N, K, R) :- K > N, !, R = 0.
binomial(N, K, R) :- K1 is N - K, > K1, !, binomialLoop(N, K1, 0, 1, R).
binomial(N, K, R) := binomialLoop(N, K, 0, 1, R).