Online appendix for the paper

Compact Serialization of Prolog Terms (with Catalan Skeletons, Cantor Tupling and Gödel Numberings

published in Theory and Practice of Logic Programming

PAUL TARAU

Department of Computer Science and Engineering
(*e-mail:* `tarau@cs.unt.edu`)

submitted April 10, 2013; revised June 23, 2013; accepted July 5, 2013

Binomial Coefficients, efficiently

Computing binomial coefficients efficiently is well-known.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)\ldots(n-(k-1))}{k!} \tag{1}$$

However, we will need to make sure that we avoid unnecessary computations and reduce memory requirements by using a tail-recursive loop. After simplifying the slow formula in the first part of the equation (1) with the faster one based on falling factorial $n(n-1)\ldots(n-(k-1))$, and 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).
```

Note that, as a simple optimization, when $N-K \leq K$, the faster computation of $\binom{N}{N-K}$ is used to reduce the number of steps in `binomial_loop`.

The resulting 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, K>K1, !, binomialLoop(N,K1,0,1,R).
binomial(N,K,R) :- binomialLoop(N,K,0,1,R).
```

***The bijection* `list2set` *together with its inverse* `set2list`**

.

```
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 (**?**) and implicitly in (**?**), with indications that it might even go back to the early days of the theory of recursive functions.

A bijection from balanced parenthesis lists to $\mathbb{N}$

This algorithm follows closely the procedural implementation described in (**?**).

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

```
cat(0,1).
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 `binDif` computes the difference of two binomials.

```
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 `localRank` computes, by binary search the rank of sequences of a given length.

```
localRank(N,As,NewLo):- X is 1, Y is 0, Lo is 0,binDif(N,0,0,Hi0),Hi is Hi0-1,
  localRankLoop(As,N,X,Y,Lo,Hi,NewLo,_NewHi).

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,
  localRankLoop(As,N,NewX,NewY,NewLo,NewHi,FinalLo,FinalHi).
localRankLoop(_As,_N,_X,_Y,Lo,Hi,Lo,Hi).
```

The predicate `rankCatalan` uses the Catalan numbers computed by `cat` in `rankLoop` to shift the ranking over the ranks of smaller sequences, after calling `localRank`.

```
rankCatalan(Xs,R):-length(Xs,XL),XL>=2, L is XL-2, I is L // 2,
  localRank(I,Xs,N),S is 0, PI is I-1,rankLoop(PI,S,NewS),R is NewS+N.

rankLoop(I,S,FinalS):-I>=0,!,cat(I,C),NewS is S+C, PI is I-1,
  rankLoop(PI,NewS,FinalS).
rankLoop(_,S,S).
```

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

```
localUnrank(N,R,As):-Y is 0,Lo is 0,binDif(N,0,0,Hi0),Hi is Hi0-1, X is 1,
  localUnrankLoop(X,Y,N,R,Lo,Hi,As).

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,
   localUnrankLoop(NewX,NewY,N,R,NewLo,NewHi,As).
localUnrankLoop(_X,_Y,_N,_R,_Lo,_Hi,_As).
```

The predicate `unrankCatalan` uses the Catalan numbers computed by `cat` in `unrankLoop` to shift over smaller sequences, before calling `localUnrank`.

```
unrankCatalan(R,Xs):- S is 0, I is 0,unrankLoop(R,S,I,NewS,NewI),
  LR is R-NewS, L is 2*NewI+1,length(As,L),localUnrank(NewI,LR,As),
  As=[_|Bs], append([0|Bs],[1],Xs).

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 following example illustrates the ranking and unranking algorithms:

```
?- unrankCatalan(2014,Ps),rankCatalan(Ps,Rank).
Ps = [0,0,1,0,1,0,1,0,0,0,0,0,1,1,1,1,1,1], Rank = 2014 .
```

An injective-only structure encoding

We sketch here an encoding mechanism that might also be useful to Prolog implementors interested in designing alternative heap representations for new Prolog runtime systems or abstract machine architectures as well as hashing mechanisms for ground terms or variant checking for tabling.

After the encoding provided by `term2bitpars` one can aggregate bitlists into natural numbers with `injEncodeStructure` by converting to $\mathbb{N}$ the resulting bitlists seen as bijective-base 2 digits and then convert them back with `injDecodeStructure`.

```
injEncodeStructure(T,N,As):-term2bitpars(T,Ps,As),fromBBase(2,Ps,N).

injDecodeStructure(N,As,T):-toBBase(2,N,Ps),bitpars2term(Ps,As,T).
```

The following example illustrates their use:

```
?- TA = f(a, g(X, Y), g(Y, X)),injEncodeStructure(TA,N,As),
   injDecodeStructure(N,As,TB).
TA = f(a, g(X, Y), g(Y, X)), N = 125347,
As = [fun(7),leaf(1,2),fun(8),leaf(0,0),leaf(0,1),fun(8),leaf(0,1),leaf(0,0)],
TB = f(a, g(A, B), g(B, A)) .
```

Note that, while simple and efficient, this encoding is *injective* only, i.e., not every natural number is a code of a term.