Deriving a Fast Inverse of the Generalized Cantor N-tupling Bijection

Paul Tarau[1]

[1] Department of Computer Science and Engineering
University of North Texas

ICLP 2012

Motivation

- curious about the following open problem:
  - Cantor's a pairing function: a bijection $f_2 : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$
  - its inverse has a well known, simple closed formula
  - it has a generalization to k-tuples, a bijection $f_k : \mathbb{N}^k \rightarrow \mathbb{N}$
  - its inverse, $f_k^{-1} : \mathbb{N} \rightarrow \mathbb{N}^k$ can be computed inefficiently by enumerating all possibilities
  - the problem: can we find an efficient way to compute it ?
  - why is this important: it is conjectured that, up to a permutation, it is the only such function that is expressed by a polynomial formula
- logic programming is an ideal paradigm for solving combinatorial search (and generation!) problems
  - backtracking and unification naturally automate search algorithms
  - well-understood program transformation techniques
  - an interactive environment, ideal for incremental development
- ⇒ derive a solution by refining a declarative specification

Outline

- the direct formula
- the case k=2 and some geometric intuitions
- the specification of the inverse
- the refinement process
- the final algorithm
- applications and conclusion

The Direct Formula for the Generalized Cantor n-tupling bijection, K_n

$$K_n(x_1, \ldots, x_n) = \binom{n-1+x_1+\ldots+x_n}{n} + \ldots + \binom{1+x_1+x_2}{2} + \binom{x_1}{1}$$

- $\binom{n}{k}$: binomial coefficient, “n choose k”
- $\binom{n}{k}$ is easy to compute, but care is taken to use a tail recursive predicate (see paper)
- EXAMPLE: $K_3(x_1, x_2, x_3) = \binom{2+x_1+x_2+x_3}{3} + \binom{1+x_1+x_2}{2} + \binom{x_1}{1}$
- $K_3(2,0,3) = \binom{2+2+0+3}{3} + \binom{1+2+0}{2} + \binom{2}{1} = \binom{7}{3} + \binom{3}{2} + \binom{2}{1}$
- $K_3(2,0,3) = 35 + 3 + 2 = 40$

To try it out, one can use the Prolog code at `http://logic.cse.unt.edu/tarau/research/2012/pcantor.pl`

```
?- from_cantor_tuple1([2,0,3],N).
N = 40.
```

The Problem: Computing the Inverse

The problem, in general terms: find a solution of the *Diophantine equation*

$$\binom{x_1}{1} + \binom{1 + x_1 + x_2}{2} + \ldots + \binom{n - 1 + x_1 + \ldots + x_n}{n} = z \qquad (1)$$

and prove that it is unique.

- unfortunately, solving an arbitrary Diophantine equation is Turing-equivalent (this is a consequence of the negative answer to Hilbert's 10-th problem, proven by Matiyasevich)
- fortunately, an inductive proof that K_n is a bijection is quite easy (see ref. in the paper) $\Rightarrow$
- we know we have exactly one solution $\Rightarrow$
- let's find it!

Cantor's Pairing Function (when $n = 2$)

- $K_2(x_1, x_2) = x_1 + \frac{(x_1+x_2+1)(x_1+x_2)}{2}$
- $K_2'(x_1, x_2) = x_2 + \frac{(x_1+x_2+1)(x_1+x_2)}{2}$ (symmetric alternative)
- K_2 and K_2' are the only ones known to be polynomials in $x_1,\ x_2$
- the inverse of $K_2(x_1, x_2)$ has simple closed formula - involving an integer square root operation (see paper)

The Inverse of Cantor’s Pairing Function: a Geometric View

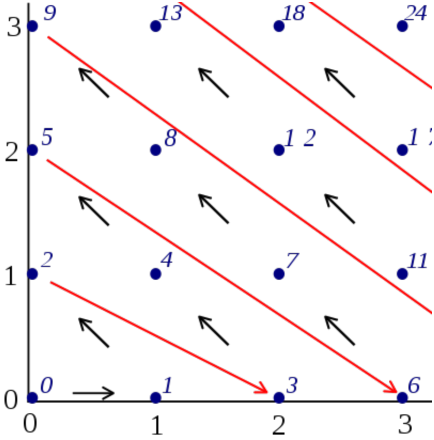

Figure : Path connecting pairs associated to successive natural numbers by the inverse of Cantor’s Pairing Function (credit: Wikipedia)

Specifying the Inverse of the Generalized Cantor Bijection

- enumerate naively until the direct function hits the value we are looking for
- a general method, good as a specification, but very slow

```
% split N in a list of K elements Ns
to_cantor_tuple1(K,N,Ns) :-
  numlist(0,N,Is), % build Is=[0,1,..N]
  cartesian_power(K,Is,Ns), % generate candidates
  from_cantor_tuple1(Ns,N). % test candidates

% generates lists of K members of input list Is
cartesian_power(0,_,[]).
cartesian_power(K,Is,[X|Xs]) :-
  K>0, K1 is K-1,
  member(X,Is), % this is where we backtrack
  cartesian_power(K1,Is,Xs).
```

Toward a Better Algorithm, using a Tighter Upper Limit

- the geometric analogy extends from 2D to N-dimensions
- `from_cantor_tuple(K,Ns,N)` runs through successive hyperplanes $X_1 + \ldots + X_k = M$ as we increment M
- for each of them the sum maxes out when $X_1 = M$ and $X_J = 0$ for $2 \leq J \leq K$.
- we can compute directly (and efficiently) this maximum value with the predicate `largest_binomial_sum` (see paper)
- we use this to derive `sum_bounded_cartesian_power`

The Algorithm, with Search Restricted to a Hyperplane

- still generate and test, but significantly faster
- search space is narrowed down to the relevant hyperplane

```
to_cantor_tuple2(K,N,Ns) :-
  % find (quickly) the relevant hyperplane associated to M
  find_hyper_plane(K,N,M),
  % generate only tuples located on it
  sum_bounded_cartesian_power(K,M,Xs),
  from_cantor_tuple1(Xs,N), % test candidates
  !, % as we know that the solution is unique
  Ns=Xs.
```

- this is “as good as it gets” – no obvious next step
- should we give up hope to find a deterministic algorithm?

The Missing Link: from Lists to Sets and Back

- lists and sets of natural numbers (represented canonically) can be morphed into each other using a simple bijection
- see PPDP'2009: An Embedded Declarative Data Transformation Language for various such morphings – specified as Prolog code

```
?- list2set([2,0,1,2],Set).
Set = [2, 3, 5, 8].

?- set2list([2, 3, 5, 8],List).
List = [2, 0, 1, 2].
```

There's Hope: the Eureka Step!

we can transform the direct function by observing that it can be decomposed into:

- a `list2set` transformation
- a simple tail recursive predicate summing up binomials

```
from_cantor_tuple(Ns,N) :-
  list2set(Ns,Xs),
  untupling_loop(Xs,0,0,N).

untupling_loop([],_L,B,B).
untupling_loop([X|Xs],L1,B1,Bn) :-
  L2 is L1+1,
  binomial(X,L2,B),
  B2 is B1+B,
  untupling_loop(Xs,L2,B2,Bn).
```

We (luckily!) bump into Combinatorial Number Systems

- the“Eureka step”: `untupling_loop` implements the sum of the combinations
- this is the representation of N in the *combinatorial number system of degree K* (also called “combinadics”)
- efficient conversion algorithms between the conventional and the combinatorial number system are well known

Theorem (Knuth)

The combination $[c_k, \ldots c_2, c_1]$ *is visited after exactly* $\binom{c_k}{k} + \ldots + \binom{c_2}{2} + \binom{c_1}{1}$ *other combinations have been visited.*

The Efficient Inverse

- the final code is remarkably simple
- it combines `set2list` and conversion from combinadics

```
to_cantor_tuple(K,N,Ns) :-
  tupling_loop(K,N,Xs),
  reverse(Xs,Rs),
  set2list(Rs,Ns).

tupling_loop(0,_,[]).
tupling_loop(K,N,[D|Ns]) :- K>0, NewK is K-1, I is K+N,
  between(NewK,I,M), binomial(M,K,B), B>N,
  !, % no more search is needed
  D is M-1, % the previous binomial gives the "digit" D
  binomial(D,K,BM), NewN is N-BM,
  tupling_loop(NewK,NewN,Ns).
```

Conclusion

- we have derived through iterative refinements a solution to an open problem for which we had no a priori idea if it is solvable
- Prolog's support for backtracking and program transformations did most of the magic
- from a software engineering perspective, this recommends (once more!) logic programming as an ideal problem solving tool
- an interesting application, in the paper: fair search
- other applications
  - dynamic n-dimensional arrays
  - polynomial Gödel numberings for Term Algebras (see Scala-based open source project) at: `http://code.google.com/p/bijective-goedel-numberings/`
- work supported by NSF research grant 1018172