Bijective Term Encodings

Paul Tarau

Department of Computer Science and Engineering
Univ of North Texas

CICLOPS 2011

Motivation

Gödel's incompleteness results (relying on Gödel numberings i.e. mappings between formulas+proofs and natural numbers) had a huge impact on logic, foundations of mathematics, number theory, computer science and quite a few other fields

- some infelicities of the original Gödel numberings:
  - encoding individual symbols rather then expression trees $\Rightarrow$ encodings of syntactically ill-formed terms are possible
  - using exponents of distinct prime numbers $\Rightarrow$ computing the inverse is intractable (based on factoring)
- none of those shortcomings matter when focus is on computability
- but they do, when one cares about computational complexity!

Revisiting Gödel numberings - with “efficiency" in mind

- we design Gödel numberings with the following properties:
  - they are bijections between terms and $\mathbb{N}$
  - natural numbers always decode to syntactically valid terms
  - work in low polynomial time in the bitsize of the representations
  - the bitsize of the encoding is within constant factor of the syntactic representation of the input
  - encodings of Term Algebras $\Rightarrow$ good for both code and data!
- bijections ensure that we can *uniquely* encode and decode something as something else
- if such bijections are isomorphisms (i.e. also transport algebraic structure) – we can even compute with them (see next paper!)

Why bijections - a simpler reason!

Figure: Bijections are popular :-)

Our Toolbox

- each of the following operations are at most linear in the bitsize of their operand `N`
- some can be considered constant time when operands fit in a machine word as well as when algorithms using mutable implementations of arbitrary length integers is used

```
first_bit(N,Bit):- Bit is 1 ∧ N.
times_exp2(N,K,R):-R is N « K.
div_by_exp2(N,K,R):-R is N » K.
predecessor(N,R):-R is N-1.
successor(N,R):-R is N+1.
```

Two More Tools

- `k_deflate` collects each k-th bit from a number's binary representation and aggregates the result into a new natural number
- `k_inflate` builds a new natural number by inserting `0`s in every position except in each k-th position where the bits of its argument `X` are placed

```
?- k_inflate(3,42,X),k_deflate(3,X,Y).
X = 33288,   % [0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1]
Y = 42 .     % [0,    1,    0,    1,    0,    1]
```

From Natural Numbers to Tuples of Natural Numbers

`to_tuple`: $Nat \rightarrow Nat^k$ converts a natural number to a k-tuple by splitting its bit representation into k groups, from which the k members in the tuple are finally rebuilt

```
to_tuple(K,N,Ns):-K>0,K1 is K-1,
  numlist(0,K1,Ks),
  maplist(div_by_exp2(N),Ks,Ys),
  maplist(k_deflate(K),Ys,Ns).
```

equivalent to the transposition of a bit matrix obtained from the tuple, seen as a number in base 2^k

From Tuples of Natural Numbers to Natural Numbers

```
from_tuple(Ns,N):-
  length(Ns,K),K>0,K1 is K-1,
  maplist(k_inflate(K),Ns,Xs),
  numlist(0,K1,Ks),
  maplist(times_exp2,Xs,Ks,Ys),
  sumlist(Ys,N).
```

- merging back the bits of the numbers `Ns` into `N`
- note the use of `k_inflate` (and `k_deflate` on the other side of the conversion) working directly on bitvectors

```
?- to_tuple(3,42,T), from_tuple(T,N).
T = [2, 1, 2],
N = 42 .
```

Gödel Numberings of a Term Algebra with Finite Signature

- a term algebra is defined over a finite set of function symbols of given arities
- constants can be singled out as a special set or considered function symbols of arity 0
- in various logic formalisms a term algebra is called a Herbrand Universe
- term algebras can be see as *free magmas* induced by a set of variables and a set of function symbols of various arities (0 included), called `signature`, that are closed under the operation of inserting terms as arguments of function symbols

Uses and Requirements for Bijective Term Encodings

- possible uses:
  - bijective encodings over a signature and a finite set of variables (seen as input “wires") for synthesizing code over a fixed set of function symbols
  - a library of logic gates, in the case of circuit synthesis
  - generating random terms of a given signature for testing purposes
  - generating candidate terms representing code in ILP or GP
  - compact ground representations of terms for SAT and ASP
  - enumerating progressively larger terms of known signature in generator based logic variable implementations (e.g. Curry)
  - compact subterm sharing in tabling
- requirements:
  - the mapping should relate terms to numbers of comparable representation size
  - it should work in linear or low polynomial time to be useful for practical applications

From Terms to Natural Numbers

- given that `Vs`, `CSyms`, `FSyms` are finite, we map them bijectively to distinct ranges in an initial segment of $\mathbb{N}$
- `term2nat` precomputes these ranges and then calls the recursive converter `t2n`

```
term2nat(Vs,CSyms,FSyms,Term, Code):-
    ...
    t2n(LV,LC,LF,LVC,Vs,CSyms,FSyms,Term, Code).
```

- `t2n` uses `lookup_var` and the built-in `nth0/3` to look-up indices associated to variable, constant and function symbols
- for compound terms, these values are combined with values computed recursively on their arguments
- values are merged using `from_tuple` into natural numbers

From Natural Numbers to Terms

- `nat2term` reverses the encoding process, using the same lists `Vs, CSyms, FSyms` to map variables, constants and function symbols to natural number codes
- it calls the recursive converter `n2t/8`

```
nat2term(Vs,CSyms,FSyms,Code, Term):-
  ...
  n2t(LV,LC,LF,LVC,Vs,CSyms,FSyms,Code, Term).
```

- `n2t` uses `to_tuple` to split natural numbers into codes for of symbols and variables, arities and structure information for arguments of compound terms to be passed to them recursively

Example: from Terms to Natural Numbers and back

```
?- T=f(a,f(X,g(Y))), Vs=[X,Y], Cs=[a], Fs=[f/2,g/1],
   term2nat(Vs,Cs,Fs,T,N), nat2term(Vs,Cs,Fs,N,T_again).

T = f(a, f(X, g(Y))),
Vs = [X, Y],
Cs = [a],
Fs = [f/2, g/1],
N = 17439,
T_again = f(a, f(X, g(Y))) .
```

Example: from Natural Numbers to Terms and back

```
?- N=2012, Vs=[X,Y], Cs=[a,b], Fs=[f/2,g/1],
   nat2term(Vs,Cs,Fs,N,T), term2nat(Vs,Cs,Fs,T,N_again).

N = 2012,
Vs = [X, Y],
Cs = [a, b],
Fs = [f/2, g/1],
T = f(f(Y, b), f(b, a)),
N_again = 2012 .
```

Encoding Boolean Formulas / Circuits

```
?- N=2012, Vs=[A,B], Cs=[0], Fs=['→'/2],
   nat2term(Vs,Cs,Fs,N,T), term2nat(Vs,Cs,Fs,T,N_again).

N = 2012,
Vs = [A, B],
Cs = [0],
Fs = [ (→)/2],
T = (((B→A)→0→A)→ (0→A)→B),
N_again = 2012 .
```

- application: exact circuit synthesis
- in combination with a fast bitvector-based boolean evaluator

Generating Random Terms of a Fixed Signature

```
ranterm(Bits,Vs,Cs,Fs, T):- N is random(2^Bits) ...

?- Vs=[A,B,C],ranterm(100,Vs,[],['+'/2,'*'/2],T).

Vs = [A, B, C],
T = (B+ (C+A))* ((A+B)*A* ((A+A)*C))+ (A+B+A*A+
    (B+ (A+A))*(B+A))+ (B*A*B* (B+B)* (C+ (C+B))+
    (A* (A+A)+C*A)* ((B+A)* ((A+A)*C))) .

?- Vs=[A,B,C,D],ranterm(50,Vs,[0,1],[and/2,or/2,not/1],T).

Vs = [A, B, C, D],
T = and(not(not(or(or(not(0), A), or(and(B, B), A)))),
    or(or(and(A, A), or(D, A)), not(or(C, not(B))))) .
```

Bijective Encodings of Prolog Atoms

- conventional numbering systems do not provide a bijection between arbitrary combinations of digits (or character codes) and natural numbers, given that leading `0`s are irrelevant
- $\Rightarrow$ we encode numbers in bijective base-k
- $\Rightarrow$ we extend the encoding to the character codes in an atom

```
?- to_bbase(7,2012,Ds), from_bbase(7,Ds,N).
Ds = [2, 6, 4, 4], N = 2012 .

?- Cs = "hello", string2nat(Cs,N), nat2string(N,CsAgain).
Cs = [104, 101, 108, 108, 111], N = 7073802,
CsAgain = [104, 101, 108, 108, 111] .
```

“Catalan skeletons” of Prolog terms

- encodings focusing on the separation of the structure and the content of Prolog terms
- we use the connections between balanced parenthesis languages and a number of different data types (among which ordered rooted multi-way and binary trees) known to combinatorialists as the *Catalan family*
- this time no fixed signature is assumed – we are just abstracting away the *structure* of a term

A skeleton: no content - just style :-)

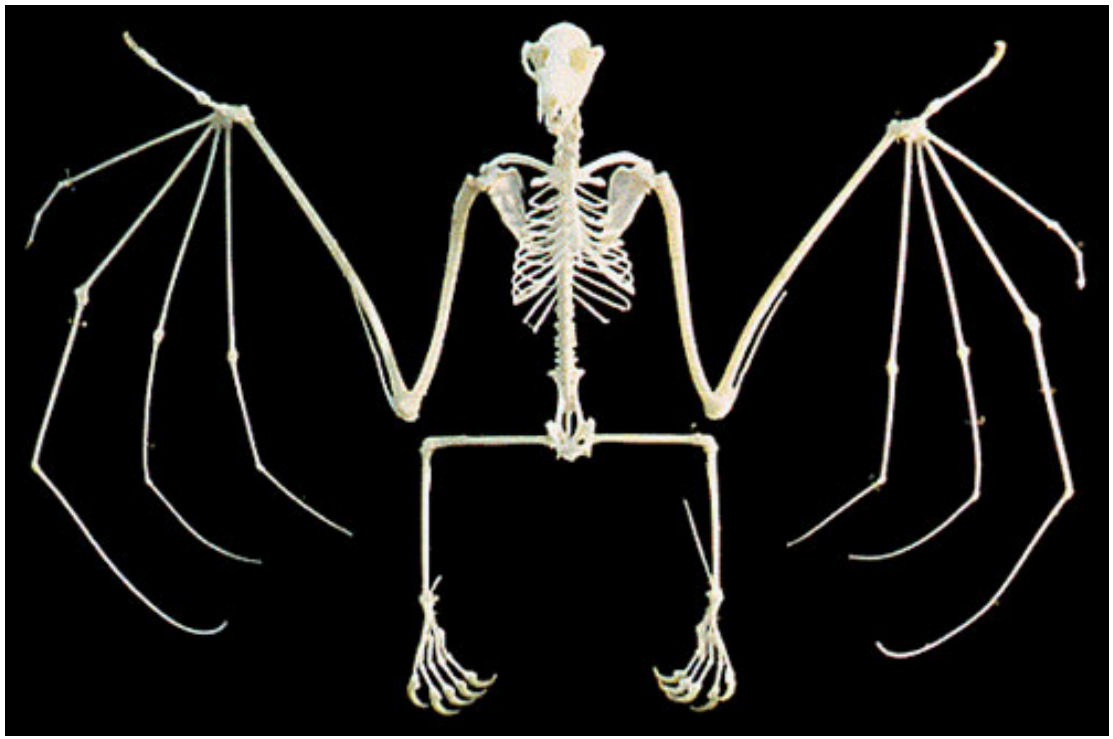

Figure: Guess to whom the content if this one used to belong!

From Terms to Parenthesis Languages: Injective Encoding

```
?- term2bitpars(f(g(a,X),X,42), Ps,As),
   bitpars2term(Ps,As, Term).

Ps = [0,0,1,0,0,0,1,0,1,0,1,1,1,0,1,0,1,1],
As = [f,g,a,X,X,42],
Term = f(g(a,X),X,42) .
```

- left parenthesis represented as bit `0`
- right parenthesis represented as bit `1`

```
?- term2inj_code(f(a,g(X,Y),g(Y,X)),N,As),
  inj_code2term(N,As,T).

N = 131364115, As = [f, a, g, X, Y, g, Y, X],
T = f(a, g(X, Y), g(Y, X)) .
```

Uncovering the Implicit List Structure of a Natural Number

Proposition

$\forall z \in \mathbb{N} - \{0\}$ *the diophantic equation*

$$2^x(2y+1) = z \tag{1}$$

has exactly one solution $x, y \in \mathbb{N}$.

```
cons(X,Y, Z) :- Z is ((Y«1)+1)«X.
```

```
decons(Z, X,Y) :- Z>0, X is lsb(Z), Y is Z»(X+1).
```

asymmetric impact of `x` (an exponent of 2) and `y` (a multiple of 2)!

A Bijection between Natural Numbers and Lists

- we combine cons/3 and decons/3 with `to_tuple` and `from_tuple` to obtain an bijection between $\mathbb{N}$ and $[\mathbb{N}]$

```
nat2nats(0,[]).
nat2nats(N,Ns) :- N>0, decons(N,L1,N1), L is L1+1,
  to_tuple(L,N1,Ns).

nats2nat([],0).
nats2nat(Ns,N) :- length(Ns,L), L1 is L-1,
  from_tuple(Ns,N1),
  cons(L1,N1,N).

?- nat2nats(2012,Ns), nats2nat(Ns,N).
Ns = [7, 7, 2],
N = 2012 .
```

A Bijection between Lists of Balanced Parenthesis and $\mathbb{N}$

```
pars2nat(Xs,T) :- pars2nat(0,1,T,Xs,[]).

pars2nat(L,R,N) --> [L],pars2nats(L,R,Xs),{nats2nat(Xs,N)}.

pars2nats(_,R,[]) --> [R].
pars2nats(L,R,[X|Xs]) --> pars2nat(L,R,X),pars2nats(L,R,Xs).
```

the inverse mapping works in a similar way, using `nat2nats` to recursively generate the lists of balanced parenthesis using a DCG

```
nat2pars(N,Xs) :- nat2pars(0,1,N,Xs,[]).

nat2pars(L,R,N) --> {nat2nats(N,Xs)},[L],nats2pars(L,R,Xs).

nats2pars(_,R,[]) --> [R].
nats2pars(L,R,[X|Xs]) --> nat2pars(L,R,X),nats2pars(L,R,Xs).
```

Bijective Catalan skeletons of Prolog terms

By combing the converters between terms to lists of parenthesis with a bijection provided by `pars2nat` and nat2pars we obtain:

```
term2code(T,N,As) :- term2bitpars(T,Ps,As),pars2nat(Ps,N).

code2term(N,As,T) :- nat2pars(N,Ps),bitpars2term(Ps,As,T).

?- term2code(f(a,g(X,Y),g(Y,X)),N,As),code2term(N,As,T).

N = 786632,
As = [f, a, g, X, Y, g, Y, X],
T = f(a, g(X, Y), g(Y, X)) .
```

- a succinct representation of the structure of a term
- could be used as a good hashkey
- an application: perfect hashing for small terms

Related Work and Links

- this paper can be seen as an application to our bijective data transformation framework which helps gluing together the pieces needed for our encodings
- PPDP'09 (Prolog-based), PPDP10 (Haskell-based) papers and large 158p draft + Haskell code at:
  - http://logic.csci.unt.edu/tarau/research/2009
  - http://logic.csci.unt.edu/tarau/research/2010
- main new contributions
  - algebras with finite signatures
  - bijective catalan skeletons

- SWI-compatible Prolog code at: http://logic.cse.unt.edu/tarau/research/2011/bijenc.pl
- we thank NSF (research grant 1018172) for support

Conclusion

- literate Prolog – for “executable” theoretical computer science –
- in contrast with cold fusion :-), executable papers start to be taken seriously, see: http://www.executablepapers.com/
- the original field for Gödel numberings is computability theory
- our Gödel numberings are “complexity aware" - possible uses in encodings relevant for complexity theory
  - encodings work in space and time proportional to the bitsize of the representations
  - natural numbers always decode to syntactically valid terms
- a possible more practical application: generate random terms - useful for QuickCheck-style testing
- also - natural numbers represent terms succinctly $\Rightarrow$ serialization of data and code, compression of terms sent over a network etc.

Questions?