Computing with Hereditarily Finite Sequences

Paul Tarau
University of North Texas

CICLOPS 2011
Hereditarily Finite Sequences - are a kind of trees - but a bit less colorful than this one...
Imagine that you are at a place where

- You are given ordered rooted trees with empty leaves.
- You are asked: can you do computations with them?
- Can you do computations with them efficiently?
- Can you make sure that no tree is wasted?
- *And the really hard one:* which movie that hopeless tree is from?
• our game: the “Tracker” provides the challenges...
• ontology: the trees have empty leaves (no bananas!)
Can you compute using trees with empty leaves?

• Yes - but that’s just slow successor arithmetic...

• []
• [ [], [] ]
• [ [], [], [] ]
• .......... 
• [ [], [], [], .... ]
Can you compute as fast as binary arithmetic?

- Yes - but I will waste an infinite number of trees...
- \(0 = []\)
- \(1 = [[]]\)
- \([0, 0, 1, 0, 1]\) would look like this:
- \([[], [], [[]], [], [[]]]\)
Can you compute without wasting any tree?

- yes, but it is quite tricky (see next slides...)
- a *bijection* between trees with empty leaves and natural numbers will be used
- after defining successor and predecessor we can even mimic the additive and multiplicative semigroup structure of N!
A bijection between finite sequences and natural numbers

\[ \text{cons}(X,Y,XY):-X \geq 0, Y \geq 0, XY \text{ is } (1 + (Y << 1)) << X. \]

\[ \text{hd}(XY,X):-XY > 0, P \text{ is } XY \land 1, \text{hd1}(P,XY,X). \]

\[ \text{hd1}(1, _, 0). \]
\[ \text{hd1}(0,XY,X):-Z \text{ is } XY >> 1, \text{hd}(Z,H), X \text{ is } H + 1. \]

\[ \text{tl}(XY,Y):-\text{hd}(XY,X), Y \text{ is } XY >> (X + 1). \]

\[ \text{null}(0). \]

- \( \text{cons}(X,Y,Z), \text{hd}(Z,X), \text{tl}(Z,Y) \iff Z = 2^X \times (2Y + 1) \)
- given \( Z \), the Diophantine eq. has one solution \( X,Y \)
- this gives a bijection between \( \mathbb{N} \) and \([\mathbb{N}]\)
You can do everything when walking over heads (and tails, not shown!)
From N to [N] and back

\[
\text{list2nat([],0).} \\
\text{list2nat([X|Xs],N):-list2nat(Xs,N1),cons(X,N1,N).}
\]

\[
\text{nat2list(0,[]).} \\
\text{nat2list(N,[X|Xs]):-N>0,hd(N,X),tl(N,T),nat2list(T,Xs).}
\]

?- nat2list(2012,Ns),list2nat(Ns,N).  
Ns = [2, 0, 0, 1, 0, 0, 0, 0],  
N = 2012
Recursing over the “\(N \text{ to } [N]\) bijection” gives:

- ranking and unranking bijections between \(N\) and hereditarily finite sequences - seen here as trees with ‘[ ]’ leaves

\[?- \ \text{nat2hfseq}(2012, \text{HFSEQ}), \text{hfseq2nat}(\text{HFSEQ}, N).\]
\[\text{HFSEQ} = [[[[]]], [], [], [[]], [], [], [], [], []], \quad N = 2012\]
Successor ($s$) and predecessor ($p$) on hereditarily finite sequences

\begin{align*}
s(\[] , \[]) & . \\
&s([[K\backslash Ks]\backslash Xs], [[\[] , K1\backslash Xs]]) : -p([K\backslash Ks], K1) . \\
s([[\[] \backslash Xs], [[K1\backslash Ks] \backslash Ys]]) : -s(Xs, [K\backslash Ys]), s(K, [K1\backslash Ks]) . \\
\end{align*}

\begin{align*}
p(\[] , \[]) & . \\
p(\[] , K\backslash Xs], [[K1\backslash Ks] \backslash Xs]) : -s(K, [K1\backslash Ks]) . \\
p([[K\backslash Ks] \backslash Xs], [[\[] \backslash Zs]]) : -p([K\backslash Ks], K1), p([K1\backslash Xs], Zs) . \\
\end{align*}
We do not want to work with these ugly tree-shaped things!

Let’s build an API emulating bijective base-2 arithmetic!

% e-->0
% o(X)-->2X+1
% i(X)-->2X+2

s(e, o(e)).
s(o(X), i(X)).
s(i(X), o(Y)):- s(X, Y).

a(e, e, e).
a(e, o(X), o(X)).
a(e, i(X), i(X)).
a(o(X), e, o(X)).
a(i(X), e, i(X)).
a(o(X), o(Y), i(R)):- a(X, Y, R).
a(o(X), i(Y), o(S)):- a1(X, Y, S).
a(i(X), o(Y), o(S)):- a1(X, Y, S).
a(i(X), i(Y), i(S)):- a1(X, Y, S).

a1(X, Y, Z):- a(X, Y, T), s(T, Z).
An API emulating bijective base-2 arithmetic

- recognizers
- constructors + destructor

\[ o([-]) \] is odd
\[ i([-]) \] is even \( <> 0 \)
\[ e(0) \] is 0

\[ o(X, [X]) \] \( X \rightarrow 2 \times X + 1 \)
\[ i(X, Y) :- s([X], Y) \] \( X \rightarrow 2 \times X + 2 \)

% destructor: undo the effect of \( o, i \)
\[ r([-], Xs) \]
\[ r([X|Xs], Ys), Rs) :- p([X|Xs], Ys), [Rs] \].
Using the API: fast conversion from/to ordinary numbers

- n2s(42, S), s2n(S, N).
  S = [[[[]]], [[[]]], [[[]]]],
  N = 42

- n(X), s2n(X, N).
  X = [], N = 0 ;
  X = [[[]]], N = 1 ;
  X = [[[[]]]], N = 2 ;
  X = [[[]], [[]]], N = 3 ;
  .......

- it converts in time/space proportional to the binary representation
- we can enumerate the infinite stream of trees
It’s time to do some real work now!

**ADDITION - efficiently**

\[
\begin{align*}
    \text{a}([\text{[]}, Y, Y].
    \text{a}([X|Xs], [\text{[]}, [X|Xs]].
    \text{a}(X, Y, Z):= \text{o}(X), \text{o}(Y), \text{a}1(X, Y, R), \text{~i}(R, Z). \\
    \text{a}(X, Y, Z):= \text{o}(X), \text{i}(Y), \text{a}1(X, Y, R), \text{a}2(R, Z). \\
    \text{a}(X, Y, Z):= \text{i}(X), \text{o}(Y), \text{a}1(X, Y, R), \text{a}2(R, Z). \\
    \text{a}(X, Y, Z):= \text{i}(X), \text{i}(Y), \text{a}1(X, Y, R), \text{s}(R, S), \text{i}(S, Z).
\end{align*}
\]

\[
\begin{align*}
    \text{a}1(X, Y, R):= \text{r}(X, RX), \text{r}(Y, RY), \text{a}(RX, RY, R). \\
    \text{a}2(R, Z):= \text{s}(R, S), \text{o}(S, Z).
\end{align*}
\]
Adding some large numbers (in tree form)

?-n2s(12345678901234567890,A),
   n2s(10000000000000000000,B),
   a(A,B,S),
   s2n(S,N).

A = [[[[]]]], [[[[]]]], [[[[]]]], [[[]]], [ [] ], [[[]]], [[[...]]], [ [] ], [ [] ! ... ],

B = [[[[]]], [ [] ], [[[[]]]], [[[[]]]], [ [] ], [ [] ], [ [] ], [ [] ], [[[]]], [[[]]], [[[]]], [[...]]... | ... ],

S = [[[[]]]], [[[[]]]], [[[[]]]], [[[]]], [ [] ], [[[]]], [[[]]], [[[...]]], [ [] ], [ [] ! ... ],

N = 22345678901234567890 .
Multiplication

\[ m([], -, []). \]
\[ m(_, [], []). \]
\[ m(X, Y, Z) :- \]
\[ p(X, X1), \]
\[ p(Y, Y1), \]
\[ m(X1, Y1, Z1), \]
\[ s(Z1, Z). \]

\[ m0([], _, _). \]
\[ m0(_|X, Y, _|Z) :- \]
\[ m0(X, Y, Z). \]
\[ m0(X, Y, Z) :- \]
\[ i_(X), r(X, X1), \]
\[ m0(X1, Y, Z1), \]
\[ a(Y, _|Z1|, Y1), \]
\[ s(Y1, Z). \]

?- n2s((10^100), Googol),
   m(Googol, Googol, S),
   s2n(S, N).

\[ \text{Googol} = [[[[]], [[]]], []], \\
   [[], []], [[], [], []], \\
   [[], []], [[][ ]|...]], \]

\[ S = [[[[]], []], [[[]]], [[]]], \\
   [[[[]]], []], [], [], [[]], \\
   [[[]]], [] |...]], \]

\[ N = 100000000............... \\
   .... 00000000000000000000 \]
Why are these operations really *cooler* than they seem at a first sight?

- These are not just *an* addition and *a* multiplication on a trees - they are *the* addition and *the* multiplication, i.e.

- The addition and multiplication operations *a/3* and *m/3* induce an isomorphism between the semirings with commutative multiplication \(<\mathbb{N},+,\ast>\) and \(<\mathbb{T},a,m>\).
Next: a fly over a few other tree-like objects
Binary Trees - seen as Goedel’s System T types

% successor
s_(e, (e->e)).
s_(((K->Ks)->Xs), (e->(K1->Xs))) :-
   p_((K->Ks), K1).
s_((e->Xs), ((K1->Ks)->Ys)) :-
   s_(Xs, (K->Ys)),
   s_(K, (K1->Ks)).

% predecessor
p_((e->e), e).
p_((e->(K->Xs)), ((K1->Ks)->Xs)) :-
   s_(K, (K1->Ks)).
p_(((K->Ks)->Xs), (e->Zs)) :-
   p_((K->Ks), K1),
Types can act as natural numbers and we can compute with them.

\[
\text{% the stream of types}
\]

? - n(T), t2n(T, N).

\[
T = e, \ N = 0 \ ;
\]
\[
T = (e \to e), \ N = 1 \ ;
\]
\[
T = ((e \to e) \to e), \ N = 2 \ ;
\]
\[
T = (e \to e \to e), \ N = 3 \ ;
\]
\[
T = (((e \to e) \to e) \to e), \ N = 4 \ ;
\]
\[
\ldots
\]

- see a derivation of a bidirectional variant in the paper

- arithmetization of types is interesting - for instance one can do type-level arithmetic in Haskell or in languages with dependent types
We can also compute with parenthesis languages!

- 0,1 strings can represent our trees succinctly ~ 2 bits/node
- they are uniquely decodable - see Kraft’s inequality in the paper
- and we can also compute with any of the members of the **Catalan family** - dozens of interesting combinatorial objects -

```
pars_hfseq(Xs,T):-pars2term(0,1,T,Xs,[]).

pars2term(L,R,Xs) --> [L],pars2args(L,R,Xs).

pars2args(_,R,[]) --> [R].
pars2args(L,R,[X|Xs])-->pars2term(L,R,X),pars2args(L,R,Xs).

?- pars_hfseq([0,0,1,0,1,1],T),pars_hfseq(Ps,T).
T = [[], []],
Ps = [0, 0, 1, 0, 1, 1]
```
And what about correctness?

- some proofs using Coq at: http://logic.csci.unt.edu/tarau/research/2011/Bij2.v.txt

- a Mathematica script with visualizations at: http://logic.csci.unt.edu/tarau/research/2010/iso.nb

Future work

• This can turn out to be practical - the representation handles huge numbers - towers of exponents that overflow binary representations

• Java and C prototypes for an arbitrary length integer package using binary trees at http://logic.csci.unt.edu/tarau/research/bijectiveNSF
Conclusion

• logic programming provides a flexible framework for modeling mathematical concepts from fields as diverse as combinatorics, formal languages, type theory and coding theory

• we have shown algorithms expressing arithmetic computations symbolically, in terms of hereditarily finite sequences, System T types, parenthesis languages

• literate Prolog program, code at: http://logic.cse.unt.edu/tarau/research/2011/pPAR.pl

• grateful for NSF support (research grant 1018172)
Questions?

• (image from Kurosawa - Dreams - 1990)