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?

What Dreams May Come -1998 movie -

- 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

```
cons(X,Y,XY):-X>=0,Y>=0,XY is (1+(Y<<1))<<X.

hd(XY,X):-XY>0,P is XY /\ 1,hd1(P,XY,X).

  hd1(1,_,0).
  hd1(0,XY,X):-Z is XY>>1,hd(Z,H),X is H+1.

tl(XY,Y):-hd(XY,X),Y is XY>>(X+1).

null(0).
```

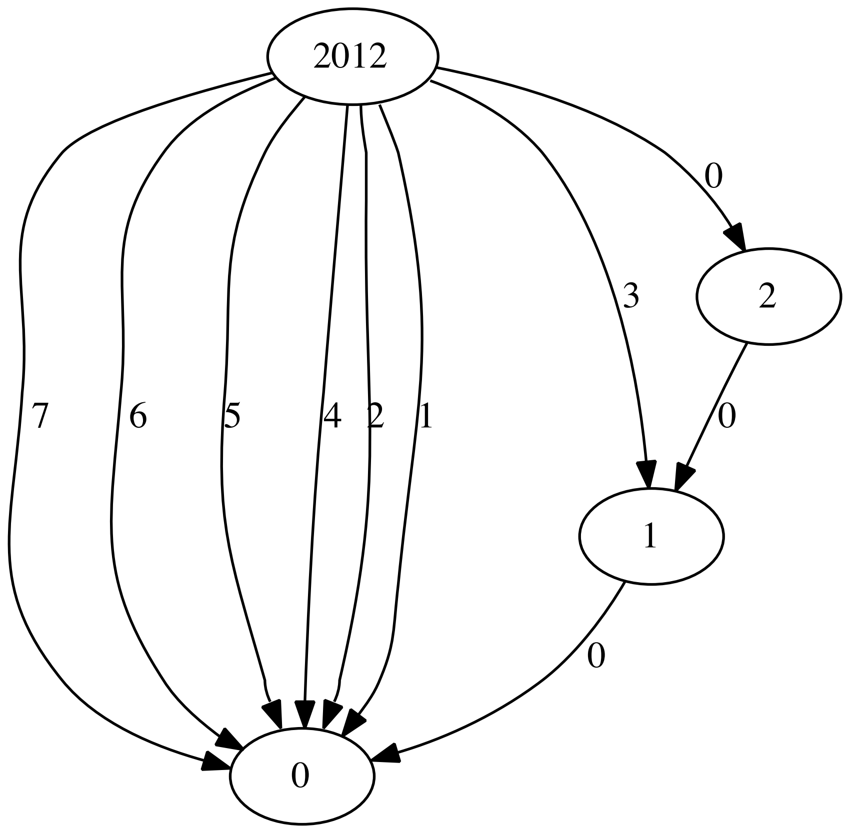
- cons(X,Y,Z), hd(Z,X), tl(Z,Y) <=> Z = 2^X*(2*Y+1)
- given Z, the Diophantine eq. has one solution X,Y
- this gives a bijection between N and [N]

You can do everything when walking over heads (and tails, not shown!)

From N to [N] and back

```
list2nat([],0).
list2nat([X|Xs],N):-list2nat(Xs,N1),cons(X,N1,N).

nat2list(0,[]).
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 to [N]* bijection" gives:



- *ranking* and *unranking* bijections between N and hereditarily finite sequences - seen here as trees with '`[]`' leaves

```
?- nat2hfseq(2012,HFSEQ),hfseq2nat(HFSEQ,N).
HFSEQ = [[[[]]], [], [], [[]], [], [], [], []],
N = 2012
```

Successor ($\texttt{s}$) and predecessor ($\texttt{p}$) on hereditarily finite sequences

```
s([],[]).
s([[K|Ks]|Xs],[[],K1|Xs]):-p([K|Ks],K1).
s([[]|Xs],[[K1|Ks]|Ys]):-s(Xs,[K|Ys]),s(K,[K1|Ks]).

p([[]],[]).
p([[],K|Xs],[[K1|Ks]|Xs]):-s(K,[K1|Ks]).
p([[K|Ks]|Xs],[[]|Zs]):-p([K|Ks],K1),p([K1|Xs],Zs).
```

We do not want to work with these ugly tree-shaped things!

Let's build an API 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_([]). % is 0

o(X,[[]|X]). % X->2*X+1
i(X,Y):-s([[]|X],Y). % X->2*X+2

% destructor: undo the effect of o,i
r([[]|Xs],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

```
a([],Y,Y).
a([X|Xs],[],[X|Xs]).
a(X,Y,Z):-o_(X),o_(Y),a1(X,Y,R),  i(R,Z).
a(X,Y,Z):-o_(X),i_(Y),a1(X,Y,R), a2(R,Z).
a(X,Y,Z):-i_(X),o_(Y),a1(X,Y,R), a2(R,Z).
a(X,Y,Z):-i_(X),i_(Y),a1(X,Y,R), s(R,S),i(S,Z).

  a1(X,Y,R):-r(X,RX),r(Y,RY),a(RX,RY,R).
  a2(R,Z):-s(R,S),o(S,Z).
```

Adding some large numbers (in tree form)

```
?-n2s(12345678901234567890,A),
  n2s(100000000000000000000,B),
  a(A,B,S),
  s2n(S,N).

A = [[[]], [[[]]], [[]], [], [[]], [[]], [[[...]]], [],
[]|...],

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

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

N = 22345678901234567890 .
```

Multiplication

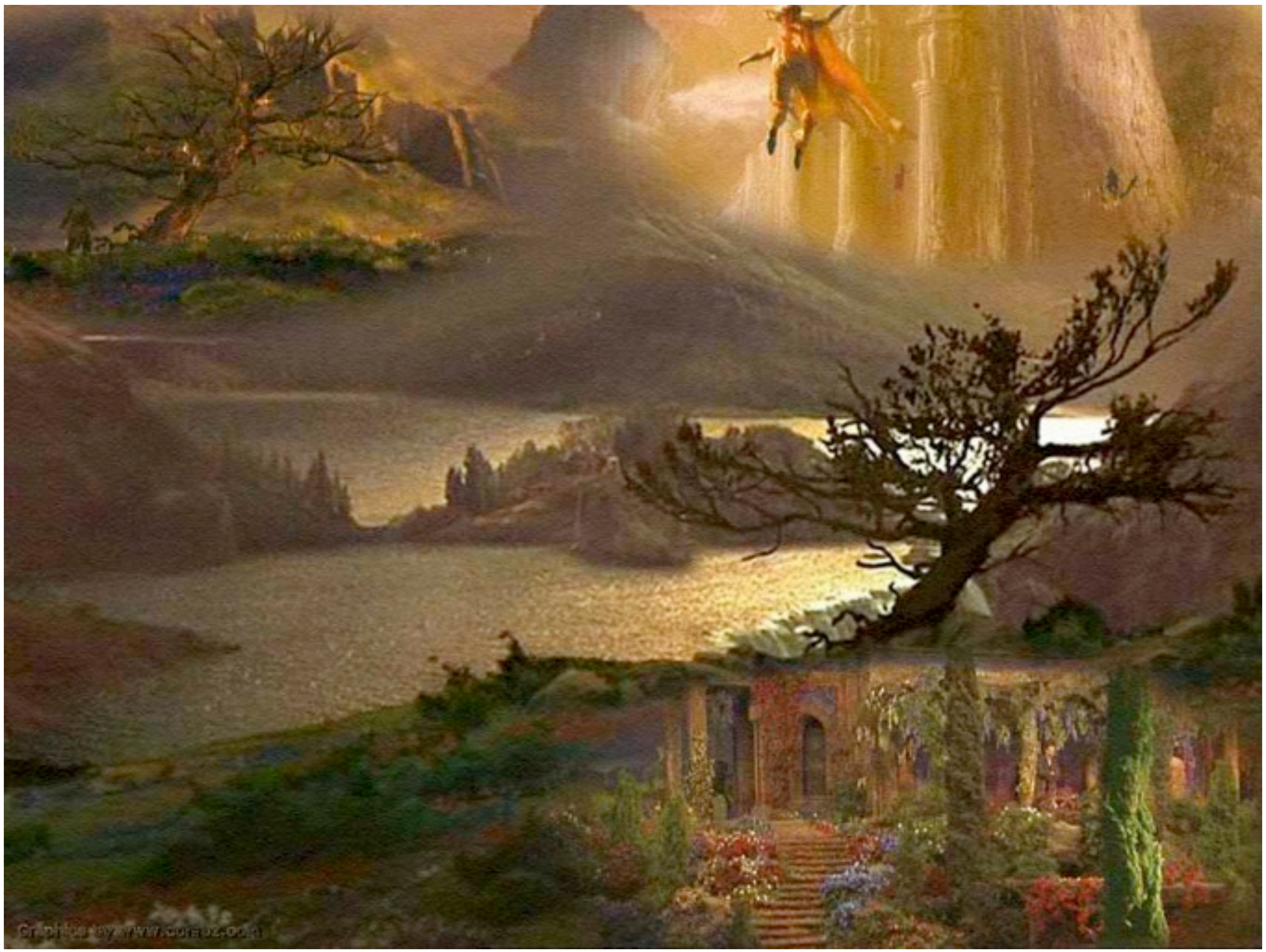
```
m([],_,[]).
m(_,[],[]).
m(X,Y,Z):-
  p(X,X1),
  p(Y,Y1),
  m0(X1,Y1,Z1),
  s(Z1,Z).

m0([],Y,Y).
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).

Googol = [[[[[]]], [[[]]], []],
         [[], []], [], [], [],
         [[], []], [[]] |...],

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

N = 100000000................
    .... 000000000000000000000
```

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 **<N,+,*>** and **<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.

```
% the stream of types

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

T = e, N = 0 ;
T = (e->e), N = 1 ;
T = ((e->e)->e), N = 2 ;
T = (e->e->e), N = 3 ;
T = (((e->e)->e)->e), N = 4 ;
...
```

- 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!

```
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]
```

- 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 -

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
- Haskell code of PPDP'2010 paper at: http://logic.csci.unt.edu/tarau/research/2010/shared.hs

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)