Computing with Catalan Families, Generically

Paul Tarau

Department of Computer Science and Engineering
University of North Texas

PADL'2016

– research supported by NSF grant 1423324 –

Overview

- we describe an arithmetic system that, instead of bitstrings, works with members of the Catalan family of combinatorial objects (e.g., trees)
- tractability of computations is only limited by the tree-representation size rather than the bitsize of their operands
- we describe arithmetic algorithms generically in terms of a Haskell type class that are
  1. within constant factors from their traditional counterparts for their average case behavior
  2. super-exponentially faster on some “interesting” giant numbers
- $\Rightarrow$ we make tractable important computations that are impossible with traditional number representations

Related work

- a tree-based number system occurs in the proof of Goodstein's theorem (1947) , where replacement of finite numbers on a tree's branches by the ordinal ω allows him to prove that a "hailstone sequence" visiting arbitrarily large numbers eventually turns around and terminates
- notations vs. computations
  - notations for very large numbers have been invented in the past ex: Knuth's up-arrow
  - in contrast to our tree-based natural numbers, such notations are not closed under successor, addition and multiplication
- other tree-representations: Knuth's TCALC, Vuillemin's Trichotomy: not a bijection to tree domains, but handling giant numbers as well
- arithmetic-like computations: J.L. Loday's - non-commutative addition
- Paulson's mechanized proof of Gödel's theorems using hereditarily finite sets (also a tree-representation) instead of $\mathbb{N}$

The Catalan family of combinatorial objects

- one of the most prolific families of combinatorial objects
- binary trees (rooted, ordered, with empty leaves)

```
data T = E | C T T deriving (Eq,Show,Read)
```

- multiway trees (rooted, ordered, with empty leaves)

```
data M = F [M] deriving (Eq,Show,Read)
```

- language of balanced parentheses
- mountain ranges
- non-crossing partitions
- handshakes over a round a table
- ...
- 58 counted in Stanley's book

A quick look at some members of the Catalan family

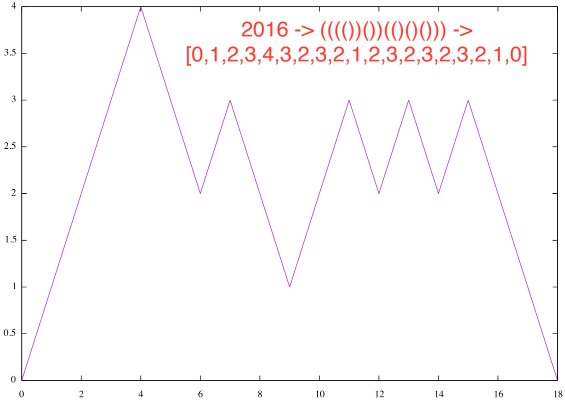


Figure: In a nutshell: we exploit a bijection between $\mathbb{N}$ and Catalan objects

A generic view of Catalan families as a Haskell type class

- built as an abstraction of the binary tree view

```
class (Show a,Read a,Eq a) => Cat a where
  e :: a

  c :: (a,a) -> a
  c' :: a -> (a,a)

  e_ ,c_ :: a -> Bool
  e_ a = a == e
  c_ a = a /= e
```

- `c` and `c'` are inverses on their domains `Cat` $\times$ `Cat` and `Cat - {e}`
- `e` is distinct from objects constructed with `c`

$$\forall x.\, c'(c\, x) = x \wedge \forall y.\, (c\_\, y \Rightarrow c\,(c'\, y) = y) \tag{1}$$

$$\forall x.\, (e\_\, x \vee c\_\, x) \wedge \neg(e\_\, x \wedge c\_\, x) \tag{2}$$

Two obvious instances of `Cat`

- rooted ordered binary trees with empty leaves

```haskell
instance Cat T where
  e = E

  c (x,y) = C x y
  c' (C x y) = (x,y)
```

- rooted ordered multiway trees with empty leaves

```haskell
instance Cat M where
  e = F []

  c (x,F xs) = F (x:xs)
  c' (F (x:xs)) = (x,F xs)
```

Blocks of digits in the binary representation of natural numbers

The (big-endian) binary representation of a natural number can be written as a concatenation of binary digits of the form

$$n = b_0^{k_0} b_1^{k_1} \dots b_i^{k_i} \dots b_m^{k_m} \tag{3}$$

with $b_i \in \{0,1\}$, $b_i \neq b_{i+1}$ and the highest digit $b_m = 1$.

Proposition

An even number of the form $0^i j$ corresponds to the operation $2^i j$ and an odd number of the form $1^i j$ corresponds to the operation $2^i (j+1) - 1$.

Proposition

A number n is even if and only if it contains an even number of blocks of the form $b_i^{k_i}$ in equation (3). A number n is odd if and only if it contains an odd number of blocks of the form $b_i^{k_i}$ in equation (3).

The constructor c: prepending a new block of digits

$$c(i,j) = \begin{cases} 2^{i+1}j & \text{if } j \text{ is odd,} \\ 2^{i+1}(j+1)-1 & \text{if } j \text{ is even.} \end{cases} \tag{4}$$

- the exponents are $i+1$ instead of i as we start counting at 0
- $c(i,j)$ will be even when j is odd and odd when j is even

Proposition

The equation (4) defines a bijection $c : \mathbb{N} \times \mathbb{N} \to \mathbb{N}^+ = \mathbb{N} - \{0\}$.

An unusual member of the Catalan family: the set of natural numbers $\mathbb{N}$

```haskell
type N = Integer
instance Cat N where
  e = 0

  c (i,j) | i>=0 && j>=0 = 2^(i+1)*(j+b)-b where b = mod (j+1) 2
```

the inverse `c'` based on *dyadic valuation* of a number *n*: i.e., the largest exponent of 2 dividing *n*

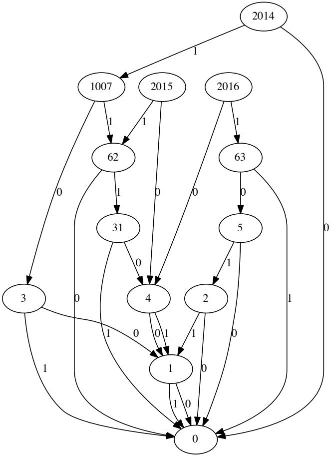
```haskell
  c' k | k>0 = (max 0 (i-1),j-b) where
    b = mod k 2
    (i,j) = dyadicVal (k+b)

    dyadicVal k | even k = (1+i,j) where
      (i,j) = dyadicVal (div k 2)
    dyadicVal k = (0,k)
```

Examples illustrating c and c' on $\mathbb{N}$

```
*GCat> c (100,200)
509595541291748219401674688561151

*GCat> c' it
(100,200)

*GCat> map c' [1..10]
[(0,0),(0,1),(1,0),(1,1),(0,2),(0,3),(2,0),(2,1),(0,4),(0,5)]

*GCat> map c it
[1,2,3,4,5,6,7,8,9,10]
```

The DAG representation of 2014,2015 and 2016

- a more compact representation is obtained by folding together shared nodes in one or more trees
- integers labeling the edges are used to indicate their order



The transformers: morphing between instances of the Catalan family

a generic transformer

```
view :: (Cat a, Cat b) => a -> b
view z | e_ z = e
view z | c_ z = c (view x,view y) where (x,y) = c' z
```

transformers defining bijections between instances of `Cat`

```
n :: Cat a => a->N
n = view
```

...

```
t :: Cat a => a->T
t = view

*GCat> t 42
C E (C E (C E (C E (C E (C E E)))))
*GCat> n it
42
```

A list view

- a list view of an instance of type class `Cat`: by iterating the constructor `c` and its inverse `c'`

```
to_list :: Cat a => a -> [a]
to_list x | e_ x = []
to_list x | c_ x  = h:hs where
    (h,t) = c' x
    hs = to_list t

from_list :: Cat a => [a] -> a
from_list [] = e
from_list (x:xs) = c (x,from_list xs)
```

- `to_list`: the children of a node in the multiway tree view
- one can use `to_list` and `from_list` to define size-proportionate bijective encodings of sets, multisets and data types built from them
- ⇒ next talk: size-proportionate encodings of lambda terms

Helpers for successor and predecessor

The operations `even_` and `odd_` implement the observation following from of Prop. 2 that parity (staring with 1 at the highest block) alternates with each block of distinct 0 or 1 digits.

```
even_ :: Cat a => a -> Bool
even_ x | e_ x = True
even_ z | c_ z = odd_ y where (_,y)=c' z

odd_ :: Cat a => a -> Bool
odd_ x | e_ x = False
odd_ z | c_ z = even_ y where (_,y)=c' z
```

We also provide a constant `u` and a recognizer predicate `u_` for 1.

```
u :: Cat a => a
u = c (e,e)

u_ :: Cat a => a-> Bool
u_ z = c_ z && e_ x && e_ y where (x,y) = c' z
```

The successor s

```
s :: Cat a => a -> a
s x | e_ x = u -- 1
s z | c_ z && e_ y = c (x,u) where -- 2
   (x,y) = c' z
s a | c_ a = if even_ a then f a else g a where
   f k | c_ w && e_ v = c (s x,y) where -- 3
    (v,w) = c' k
    (x,y) = c' w
   f k = c (e, c (s' x,y)) where -- 4
     (x,y) = c' k

   g k | c_ w && c_ n && e_ m = c (x, c (s y,z)) where -- 5
    (x,w) = c' k
    (m,n) = c' w
    (y,z) = c' n
   g k | c_ v = c (x, c (e, c (s' y, z))) where -- 6
    (x,v) = c' k
    (y,z) = c' v
```

The predecessor s'

```
s' :: Cat a => a -> a
s' k | u_ k = e where -- 1
    (x,y) = c' k
s' k | c_ k && u_ v = c (x,e) where -- 2
    (x,v) = c' k
s' a | c_ a = if even_ a then g' a else f' a where
     g' k | c_ v && c_ w && e_ r = c (x, c (s y,z)) where -- 6
       (x,v) = c' k
       (r,w) = c' v
       (y,z) = c' w
     g' k  | c_ v = c (x,c (e, c (s' y, z))) where -- 5
       (x,v) = c' k
       (y,z) = c' v

     f' k | c_ v && e_ r = c (s x,z) where -- 4
        (r,v) = c' k
        (x,z) = c' v
     f' k =  c (e, c (s' x,y)) where -- 3
        (x,y) = c' k
```

Effect of successor: a mountain range view

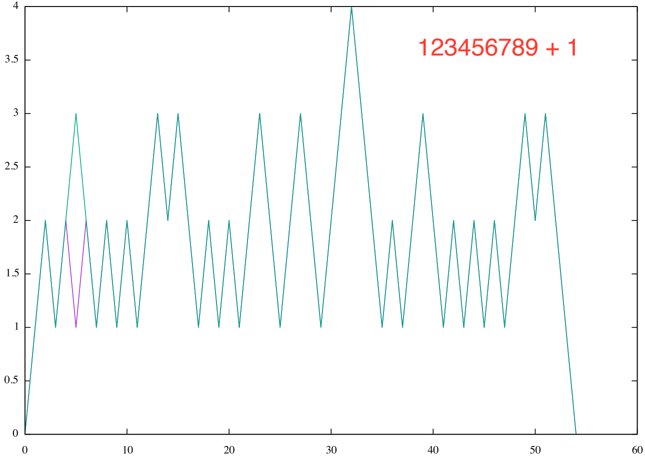


Figure: The change induced by an application of s

`s` and `s'` are inverses

Proposition

Denote $Cat^{+} = Cat - \{e\}$ The functions $s : Cat \rightarrow Cat^{+}$ and $s' : Cat^{+} \rightarrow Cat$ are inverses.

Proof.

It follows by structural induction after observing that patterns marked with the same label in `s` correspond one by one to the same patterns in `s'` and vice versa. □

More generally, it can be proved by structural induction that Peano's axioms hold and, as a result, $< Cat, e, s >$ is a Peano algebra.

The *log** worst case complexity of `s` and `s'`

NOTE: our statements about complexity apply to instances like `T` and `M` (for which `c` and `c'` are constant time)

Proposition

The worst case time complexity of the `s` and `s'` operations on n is given by the iterated logarithm $O(\log_2^*(n))$.

Proof.

Note that calls to `s,s'` in `s` or `s'` happen on terms at most logarithmic in the bitsize of their operands. The recurrence relation counting the worst case number of calls to `s` or `s'` is: $T(n) = T(\log_2(n)) + O(1)$, which solves to $T(n) = O(\log_2^*(n))$. $\square$

The constant average complexity of `s` and `s'`

Proposition

The functions `s` and `s'` work in constant time, on the average.

Proof.

Observe that the average size of a contiguous block of 0s or 1s in a number of bitsize n is asymptotically 2 as $\sum_{k=0}^{n} \frac{1}{2^k} = 2 - \frac{1}{2^n} < 2$. $\square$

While the same average case complexity applies to successor and predecessor operations on ordinary binary numbers, their worst case complexity is $O(\log_2(n))$ rather than the asymptotically much smaller $O(\log_2^*(n))$.

Other $O(log^*)$ worst case and $O(1)$ average operations

At most one call to `s`, `s'` are made in each function.

```
db :: Cat a => a -> a
db x | e_ x  = e
db x | odd_ x = c (e,x)
db z = c (s x,y) where (x,y) = c' z

hf :: Cat a => a -> a
hf x | e_ x = e
hf z | e_ x = y where (x,y) = c' z
hf z  = c (s' x,y) where (x,y) = c' z

exp2 :: Cat a => a -> a
exp2 x | e_ x = u
exp2 x = c (s' x, u)

log2 :: Cat a => a -> a
log2 x | u_ x = e
log2 x | u_ z = s y where (y,z) = c' x
```

Addition and subtraction - details in the paper

- a (fairly long) chain of mutually recursive functions defines addition and subtraction.
- we want to take advantage of large contiguous blocks of o^n and i^m applications
- we will rely simple equations governing applications and “un-applications” of such blocks

1. leftshift, rightshift operations
2. detaching and fusing blocks of similar digits
3. addition and subtraction:
4. comparison operation
5. bitsize operation

Conclusions

- we have described through a type class mechanism an arithmetic system working on members of the Catalan family of combinatorial objects
- the resulting numbering system is *canonical* - each natural number is represented as a unique object
- it is also *generic* – no commitment is made to a particular member of the Catalan family
- efficient arithmetic operations with any of the 58 known instances of the Catalan family described in Stanley's book
- instances with geometric, combinatorial, algebraic, formal languages, number and set theoretical or physical flavor
- this generalization opens the door to new and possibly unexpected applications
- the paper is a literate program, our Haskell code is at `http://www.cse.unt.edu/~tarau/research/2015/GCat.hs`

Next: the geometry behind such arithmetic operations

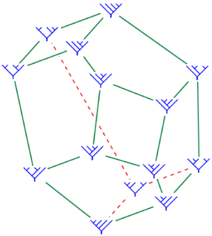


Figure: The K5 associahedron: [8,9,10,11,12,13,14,16,30,31,63,127,255,65535]

- famous polytopes (in dim *n*): hypercubes, associahedrons, permutahedrons
- geometry behind the arithmetic on the hypercubes - the usual binary computation - well known
- geometry behind the arithmetic associahedrons - follow-up to this paper
- open problem: can this be done for permutahedrons? (that would bring us reversible arithmetic operations – important for Quantum Computing)