Two Concrete Challenges in Complexity Analysis of Actual Haskell Code

Johannes Waldmann, HTWK Leipzig

Logic, Compexity, and Automation
Obergurgl, 2016

Why do Research in Complexity?

- better theorems
- more papers
- faster code!
- more realistically: warn about slow code

Why do Research in Complexity?

- better theorems
- more papers
- faster code!
- more realistically: warn about slow code

Why do Research in Complexity?

- better theorems
- more papers
- faster code!
- more realistically: warn about slow code

Why do Research in Complexity?

- better theorems
- more papers
- faster code!
- more realistically: warn about slow code

This Talk: Two Concrete Challenges

- text formatting (*pretty-printing*)
  innermost derivational complexity
  accidentally quadratic?
- binary operations on *balanced search trees*:
  sharp bound that depends on both input sizes (instead of just their sum)
  innermost runtime complexity
- this is actual Haskell code from popular libraries (e.g., GHC uses them), performance is crucial
- code is being developed actively,
  guided by papers, benchmarks, …
  and gut feelings — can we do better?

This Talk: Two Concrete Challenges

- text formatting (*pretty-printing*)
  innermost derivational complexity
  accidentally quadratic?
- binary operations on *balanced search trees*:
  sharp bound that depends on both input sizes (instead of just their sum)
  innermost runtime complexity
- this is actual Haskell code from popular libraries (e.g., GHC uses them), performance is crucial
- code is being developed actively,
  guided by papers, benchmarks, . . .
  and gut feelings — can we do better?

This Talk: Two Concrete Challenges

- text formatting (*pretty-printing*)
  innermost derivational complexity
  accidentally quadratic?
- binary operations on *balanced search trees*:
  sharp bound that depends on both input sizes (instead of just their sum)
  innermost runtime complexity
- this is actual Haskell code from popular libraries (e.g., GHC uses them), performance is crucial
- code is being developed actively,
  guided by papers, benchmarks, . . .
  and gut feelings — can we do better?

This Talk: Two Concrete Challenges

- text formatting (*pretty-printing*)
  innermost derivational complexity
  accidentally quadratic?
- binary operations on *balanced search trees*:
  sharp bound that depends on both input sizes (instead of just their sum)
  innermost runtime complexity
- this is actual Haskell code from popular libraries (e.g., GHC uses them), performance is crucial
- code is being developed actively,
  guided by papers, benchmarks, . . .
  and gut feelings — can we do better?

Text Formatting (Pretty Printing)

- abstract data type *D* (Document)
  D represents set of 2-dimensional layouts
- construction from
  - individual letters (or short strings)
  - composition: atop, beside, choice

```
text "A" <+> vcat [text "B", text "C"] ==>  A B
                                              C
```

- rendering: compute one layout that (best) matches some criteria (e.g., page width)
- typical papers/implementations:
  - John Hughes 1995, *The Design of a Pretty Printing Library*, Simon Peyton Jones 1997, `pretty`
  - Derek Oppen 1980, Phil Wadler 1998, *A Prettier Printer*, Dan Leijen, `wl-pprint`, Edward Kmett, `wl-pprint-extras`

Expected Cost and Cost Model

- practical consideration:
  - (pretty) printing must *never* take noticeable time
  - application: printing termination problems and termination proofs — these can be *huge*
- formal model:
  - cf. Wadler Section 3, page 11: “reasonble to expect ... time $O(s)$, where s is the size of the document”
  - “document” $=$ term of *nested* API calls
  - innermost derivational complexity should be linear
- assumptions:
  - string catenation in (amortized) constant time
  - nested indentation may produce quadratic amount of whitespace — ignore (compress)

Actual Cost of Formatting

- occasional observations (for a long time) pretty printer appears slow, or even hangs
- automated benchmarking:
  `https://github.com/jwaldmann/pretty-test`
  enumeration of families $t_k = C^k[t_0]$
  suggest that cost is super-linear
-
```
import Text.PrettyPrint.HughesPJ --  pretty
length $ render
  $ iterate (\d -> sep [text "l",cat [d],text "l"]
            (text "l") !! 400
```
-
```
import Text.PrettyPrint.Free -- wl-pprint-extras
putDoc $ iterate (\d -> hsep [d, sep []])
                 (text  "l") !! 40
```

What the Implementation Looks Like

- it is really a first-order functional program
- higher order functions for notational convenience only, the compiler inlines them (it should)
- uses algebraic data types (trees) to represent documents
- also uses numbers, e.g., to decide whether something fits on a line, or needs break

Actual Code: Does This Look Risky?

(from `wl-pprint-1.2/Text/PrettyPrint/Leijen.hs`)

```
best n k (Cons i d ds) = case d of
   Union x y -> nicest n k (best n k (Cons i x ds))
                           (best n k (Cons i y ds))
nicest n k x y  | fits width x  = x
                | otherwise     = y
    where width = min (w - k) (r - k + n)
fits w x     | w < 0   = False
fits w SEmpty          = True
fits w (SChar c x)     = fits (w - 1) x
fits w (SText l s x)   = fits (w - l) x
fits w (SLine i x)     = True
```

Neil Mitchell (on similar code in `pretty`):
"I'd be surprised if this was *not* quadratic."

Accidentally Quadratic

- claim: in most practical cases, you want linear cost, and anything above that is a *bug*
- nice collection of such bugs:
  `http://accidentallyquadratic.tumblr.com/`
  (Nelson Elhage)
  from various libraries, languages, paradigms
- suggestion:
  develop methods and tools to prove and disprove *linear* upper bounds
- start with: counting symbols, weights, exotic matrix interpretations (incl. matchbounds)

Balanced Search Trees

- standard implementation of sets (and maps)
  - order: ensures that query can be answered by walking one path (so work is bounded by height)
  - balance (AVL, Red-Black, 2/3, weight, ...): ensures that height is logarithmic in size
- typical proof obligations:
  - correctness $\Leftarrow$ maintain order
  - complexity $\Leftarrow$ maintain balance
- do we really need this? — yes, if ...
  - we don't have a good hash function,
  - or we need persistence,
  - or we need bulk operations (see later this talk)

Cost Model and Analysis

- model: innermost runtime complexity
  (cost of evaluating *one* operator application)
- challenge: bound complexity (automatically)
  without referring to balance
- suggestion: instead of “cost is logarithmic in size”, prove
  “cost is linear in height”.
- an analyzer would look at “just the code”,
  and not at the proof (of balance).
  but really, the code *should* contain the proof
  (the language should be dependently typed)

Binary Operations

- intersection, union, difference, ...

  ```
  intersectionWith :: Ord k
    => (a -> b -> c)
    -> Map k a -> Map k b -> Map k c
  ```

- example application:
  (general) multiplication of sparse matrices
  (specific) my recent re-implemetation of matchbound construction
- weighted relation: pair of nested maps
  $(A \to (B \to W), B \to (A \to W))$
  for $R \circ S$, need range(R) $\cap$ domain(S)
- cannot do this efficiently with hashtables (?)

Cost: Claims, Hopes and Proofs

containers:Data.Map.Strict.union

- API doc (version 0.5.6.3) says: $O(m + n)$
  (Adams 1992)
- but really? What if $m \ll n$?
  naive method (m lookups) gives $m \cdot \log n$.
- Adams: “considerably faster when one tree is small or trees have dense regions that do not overlap” (but no formal claim)
- (version 0.5.8.1, August 31) $O(m \cdot \log(n/m + 1))$
  (Blelloch et al., 2016)
  cf. no. of comparisons for merging: $\log_2 \binom{n+m}{m}$

Cost: Claims, Hopes and Proofs

containers:Data.Map.Strict.union

- API doc (version 0.5.6.3) says: $O(m + n)$
  (Adams 1992)
- but really? What if $m \ll n$?
  naive method (m lookups) gives $m \cdot \log n$.
- Adams: “considerably faster when one tree is small or trees have dense regions that do not overlap” (but no formal claim)
- (version 0.5.8.1, August 31) $O(m \cdot \log(n/m + 1))$
  (Blelloch et al., 2016)
  cf. no. of comparisons for merging: $\log_2 \binom{n+m}{m}$

Cost: Claims, Hopes and Proofs

containers:Data.Map.Strict.union

- API doc (version 0.5.6.3) says: $O(m+n)$ (Adams 1992)
- but really? What if $m \ll n$? naive method (m lookups) gives $m \cdot \log n$.
- Adams: “considerably faster when one tree is small or trees have dense regions that do not overlap” (but no formal claim)
- (version 0.5.8.1, August 31) $O(m \cdot \log(n/m+1))$ (Blelloch et al., 2016) cf. no. of comparisons for merging: $\log_2 \binom{n+m}{m}$

Cost: Claims, Hopes and Proofs

containers:Data.Map.Strict.union

- API doc (version 0.5.6.3) says: $O(m+n)$ (Adams 1992)
- but really? What if $m \ll n$? naive method (m lookups) gives $m \cdot \log n$.
- Adams: “considerably faster when one tree is small or trees have dense regions that do not overlap” (but no formal claim)
- (version 0.5.8.1, August 31) $O(m \cdot \log(n/m+1))$ (Blelloch et al., 2016) cf. no. of comparisons for merging: $\log_2 \binom{n+m}{m}$

Formal Analysis of Cost

- again, independent of correctness,
- express bound as function of two input sizes and heights (i.e., four parameters in total)
- the bound we want is product of linear functions, $\text{height}(t_1) \cdot \text{size}(t_2)$
- this is actually the "total work" bound, Blelloch et al. also compute the "span" (longest data dependency) as $\text{height}(t_1) \cdot \text{height}(t_2)$.
- suggestion: methods to bound cost for multi-ary functions by product of linear unary functions (of size and height)

Actual Code

- uses algebraic data types (binary trees) — good
- and numbers (for balancing) — bad?

```
balance k x l r
  | sizeL + sizeR <= 1    = Bin sizeX k x l r
  | sizeR > delta*sizeL   = rotateL k x l r
  | sizeL > delta*sizeR   = rotateR k x l r
  | otherwise             = Bin sizeX k x l r
  where sizeL = size l ; sizeR = size r ; sizeX =
rotateR k x l@(Bin _ _ _ ly ry) r
  | size ry < ratio*size ly = singleR k x l r
  | otherwise               = doubleR k x l r
singleL k1 x1 t1 (Bin _ k2 x2 t2 t3)
  = bin k2 x2 (bin k1 x1 t1 t2) t3
```

- well, not bad, but complicated?

But It's Already Proven In The Papers

- papers can be wrong (Adams 92 was)

```
Note that according to the Adam's paper:
- [delta] should be larger than 4.646 with a [rati
- [delta] should be larger than 3.745 with a [rati
But the Adam's paper is erroneous:
- It can be proved that for delta=2 and delta>=5
  there does not exist any ratio that would work.
- Delta=4.5 and ratio=2 does not work.
```

- even when they're right — code is not paper

```
-- The balance function is equivalent to the follo
[what you saw on previous slide]
-- It is only written in such a way that
-- every node is pattern-matched only once.
```

But It's Already Proven In The Papers

- papers can be wrong (Adams 92 was)

```
Note that according to the Adam's paper:
- [delta] should be larger than 4.646 with a [rati
- [delta] should be larger than 3.745 with a [rati
But the Adam's paper is erroneous:
- It can be proved that for delta=2 and delta>=5
  there does not exist any ratio that would work.
- Delta=4.5 and ratio=2 does not work.
```

- even when they're right — code is not paper

```
-- The balance function is equivalent to the follo
[what you saw on previous slide]
-- It is only written in such a way that
-- every node is pattern-matched only once.
```

Conclusion

- height (in addition to size)
- linear functions
- products of linear functions
- real world examples