Implementing incremental and parallel parsing

Tobias Olausson

University of Gothenburg

May 22, 2014

In this talk

- Parsing using the CYK algorithm
- Implementing an incremental parser
- Writing dependently typed Haskell code
- ...and a lot of itemized lists

Motivation

- Parallelism everywhere
  → divide-and-conquer algorithms suitable
- Syntax highlighting sucks
- Parsing is usually linear-ish

We can do better!

Context-free grammars

- 4-tuple: $G = (V, \Sigma, P, S)$
  - V set of variables
  - Σ set of terminal symbols
  - P productions, recursive rules
  - S start symbol, entrypoint
- Backus-Naur Form

  $A ::= a$

  $A ::= BC$

CYK algorithm

- Cocke, Younger and Kasami (60s)
- Grammar in Chomsky Normal Form
  $\rightarrow$ Only binary or unary rules
- Recognition matrix
- Not very efficient, $O(n^3)$

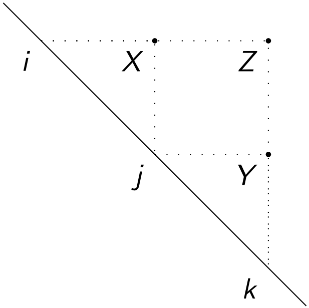
How does it work?

$$W_{i,i+1} = \{A | A ::= S_t[i] \in P\}$$

$$W_{ij} = \sum_{k=i+1}^{j} W_{ik} \cdot W_{kj}$$

$$x \cdot y = \{A | A_0 \in x, A_1 \in y, A ::= A_0 A_1 \in P\}$$

Anything below the diagonal is 0



Valiant's algorithm (1975)

- Improvement of the CYK algorithm
- Context-free recognition is the same as transitive closure...
  ...which is the same as matrix multiplication
  ...which in turn we only need to do with boolean matrices.

Matrix multiplication can be done faster than $O(n^3)$ (but not *that* much faster)

Recent improvement

- Bernardy and Claessen (2013)
- For a lot of inputs, large parts of the matrices will be empty, in fact so empty that we can optimise based on that.
- New time complexity: $O(log^3\ n)$
- Took care of linear behaviour by using an oracle
  → tagged rules as left or right

Implementation

- Lexing
  - Input is FingerTree of characters
  - Output is FingerTree of tokens
- Parsing
  - Input is FingerTree of Tokens
  - Output is Matrix of AST

Finger trees

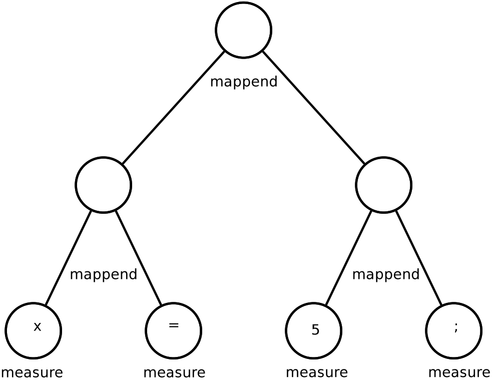
- Balanced trees
- Notion of *measuring*

```
class Monoid v ⇒ Measured v a | a → v where
    measure :: a → v
```

Measures are cached at each node in the tree. That, plus the tree structure makes the FingerTree suitable for an incremental approach - that can be easily parallelizable. Also note the Monoid constraint!

measure and mappend

```
class Monoid a where
    mempty :: a,
    mappend :: a → a → a
```



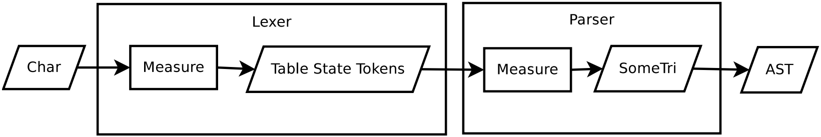
Using a lexer

- MSc thesis by Hansson and Hugo
- Input source code as a FingerTree
- Lexing by measuring `Char` → `FingerTree` of tokens

Idea of parsing

- Use the same approach as the lexer!
- Lexer measure `Char` → `FingerTree` of tokens
- Have the parser measure tokens → CYK entries

Pipeline of measures

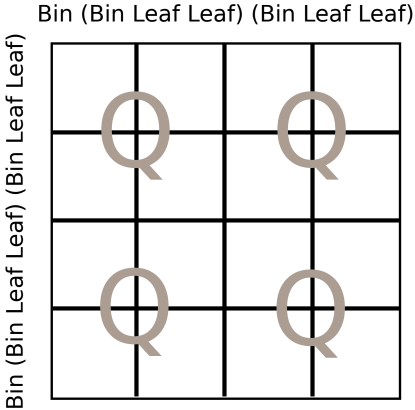


This is done for every `Char`. Results are combined using `mappend`.

Matrix implementation

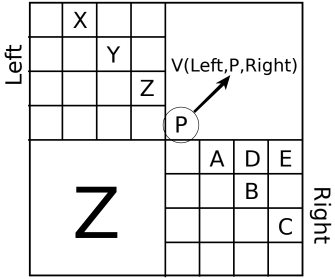
```
data Shape = Bin Shape Shape | Leaf
data Mat :: Shape → Shape → ∗ → ∗ where
  Zero :: Mat x y a
  Row :: Mat x1 Leaf a → Mat x2 Leaf a → Mat (Bin x1 x2) Leaf
  Col :: Mat Leaf y1 a → Mat Leaf y2 a → Mat Leaf (Bin y1 y2)

data SomeTri a where
  T :: Shape’ s → Pair (Mat s s a) → SomeTri a
```

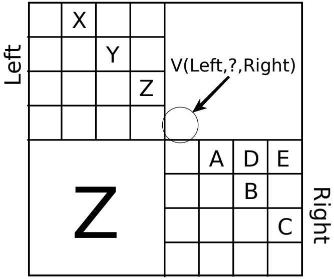
Matrix representation


Bin (Bin Leaf Leaf) (Bin Leaf Leaf)
Bin (Bin Leaf Leaf) (Bin Leaf Leaf)
Q
Q
Q
Q

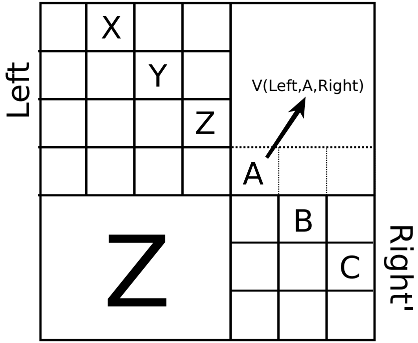
Parsing as matrix multiplication

Existing implementation using middle element. Insufficient when such an element is missing.





Chopping



Oracle

How to select random bit?

```
instance Monoid (SomeTri a) where
    t0 ‘mappend‘ t1 = unsafePerformIO $ do
        b ← randomIO
        return $ merge b t0 t1
```

Complete parser

- Lexer and parser generated from BNFC
- Successfully parsing correct input
- Has no error handling

Error handling

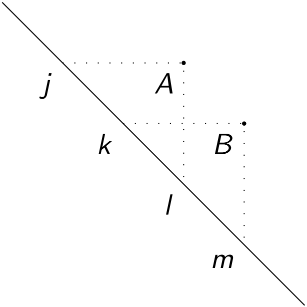
- No information about the position of tokens
  → hard to pin-point errors.

Error handling

- No information about the position of tokens
  → hard to pin-point errors.
- CYK may fail completely or just for some sections.
  → has to be handled!
- CYK may overlap in some sections, what to do?

Overlaps

Overlaps may occur in the parser, depending on the input. In this example we can parse $j - l$ and $k - m$, but not $k - l$ and not $j - m$.



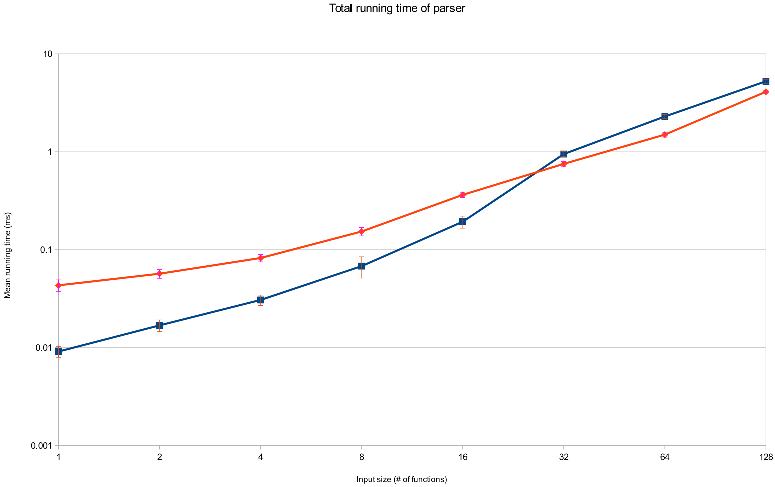
Running time

- Hard to measure, large memory requirements!
- Needs to force evaluation
- But seems to behave

Running time

- Hard to measure, large memory requirements!
- Needs to force evaluation
- But seems to behave
- Calls for further investigation

Graph of running time


Total running time of parser
Mean running time (ms)
Input size (# of functions)