# 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
- Syntax highlighting sucks
- Parsing is costly

We can address all three!

# Context-free grammars

- 4-tuple: $G = (V, \Sigma, P, S)$
  - $V$ set of variables
  - $\Sigma$ set of terminal symbols
  - $P$ productions, recursive rules
  - $S$ start symbol, entrypoint
- Language recognized denoted $L(G)$
- $L(G) = \{w \in \Sigma^* \mid S \underset{G}{\overset{*}{\Rightarrow}} w\}$

# CYK algorithm

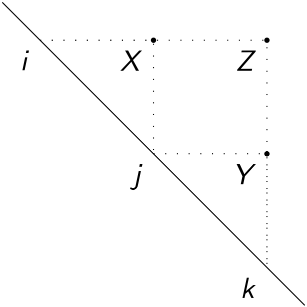
- Cocke, Younger and Kasami (60s)
- Recognition matrix for context-free languages
- Not very efficient, $O(n^3)$

# How does it work?

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

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

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



# Valiant's algorithm

- 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 input, 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

# Using a lexer

- MSc thesis by Hansson and Hugo
- Input source code as a FingerTree
- Lexing by measuring

# Finger trees

- Balanced trees
- Notion of *measuring*

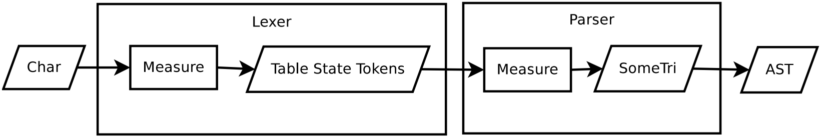
```
class Monoid v ⇒ Measured v a | a → v where
    measure :: a → v
```

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.

# Idea of parsing

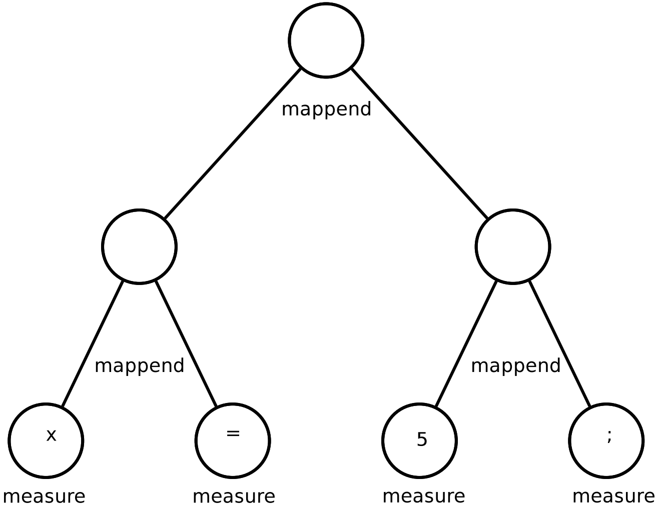
- Use the same approach as the lexer!
- Lexer measure Char → FingerTree of tokens
- Have the parser measure tokens → CYK entries

# Pipeline of measures

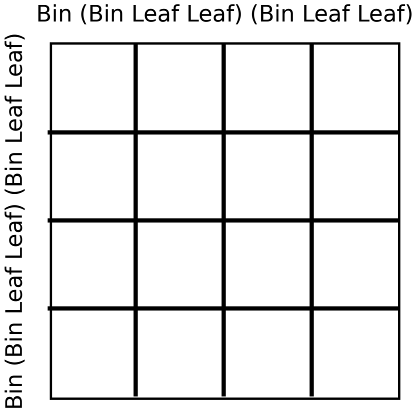


This is done for every `Char`. Results are combined using `mappend`.

# measure and mappend



# Matrix representation


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

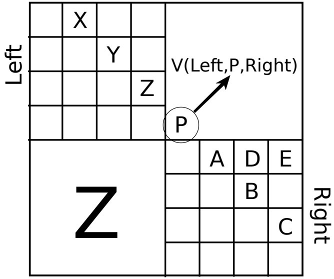
# Matrix implementation

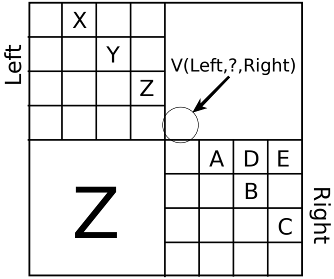
```
data Shape = Bin Shape Shape | Leaf
data Mat :: Shape → Shape → * → * where
  Zero :: Mat x y a
  Row :: Mat x1 Leaf a → Mat x2 Leaf a → Mat (Bin x1 x2) Leaf
  Col :: Mat Leaf y1 a → Mat Leaf y2 a → Mat Leaf (Bin y1 y2)

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

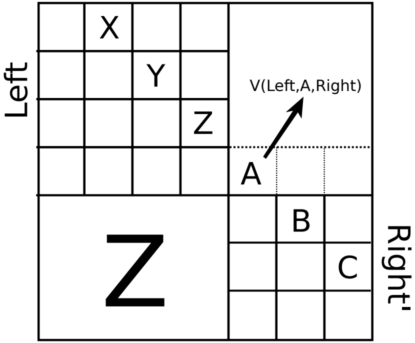
# Parsing as matrix multiplication

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





# Chopping



# Complete parser

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

Something about the error handling