Automata and Formal Languages (7)

Email:	christian.urban at kcl.ac.uk
Office:	S1.27 (1st floor Strand Building)
Slides:	KEATS (also home work is there)

Two Weeks Ago: CFGs

A context-free grammar (CFG) G consists of:

- a finite set of nonterminal symbols (upper case)
- a finite terminal symbols or tokens (lower case)
- a start symbol (which must be a nonterminal)
- a set of rules

$$A \rightarrow \text{rhs}_1 | \text{rhs}_2 | \ldots$$

where rhs are sequences involving terminals and nonterminals (can also be empty).

Two Weeks Ago: CFGs

A context-free grammar (CFG) G consists of:

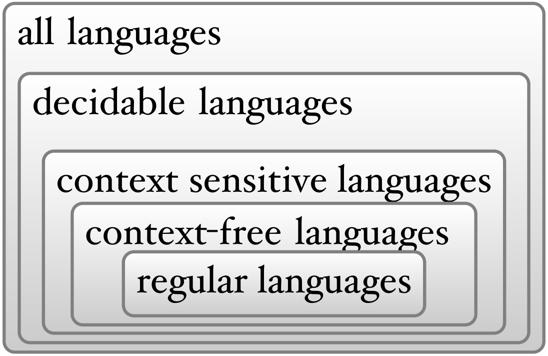
- a finite set of nonterminal symbols (upper case)
- a finite terminal symbols or tokens (lower case)
- a start symbol (which must be a nonterminal)
- a set of rules

$$A \rightarrow \text{rhs}_1 | \text{rhs}_2 | \ldots$$

where rhs are sequences involving terminals and nonterminals (can also be empty).

Hierarchy of Languages

Recall that languages are sets of strings.



Arithmetic Expressions

A grammar for arithmetic expressions and numbers:

$$\begin{array}{lcl} E & \rightarrow & E \cdot + \cdot E \mid E \cdot * \cdot E \mid (\cdot E \cdot) \mid N \\ N & \rightarrow & N \cdot N \mid 0 \mid 1 \mid \ldots \mid 9 \end{array}$$

Unfortunately it is left-recursive (and ambiguous).

A problem for recursive descent parsers (e.g. parser combinators).

Arithmetic Expressions

A grammar for arithmetic expressions and numbers:

$$\begin{array}{lcl} \boldsymbol{E} & \rightarrow & \boldsymbol{E} \cdot + \cdot \boldsymbol{E} \mid \boldsymbol{E} \cdot * \cdot \boldsymbol{E} \mid (\cdot \boldsymbol{E} \cdot) \mid \boldsymbol{N} \\ \boldsymbol{N} & \rightarrow & \boldsymbol{N} \cdot \boldsymbol{N} \mid 0 \mid 1 \mid \ldots \mid 9 \end{array}$$

Unfortunately it is left-recursive (and ambiguous).

A problem for recursive descent parsers
(e.g. parser combinators).

Numbers

$$\boldsymbol{N} \;\rightarrow\; \boldsymbol{N} \cdot \boldsymbol{N} \mid 0 \mid 1 \mid \ldots \mid 9$$

A non-left-recursive, non-ambiguous grammar for numbers:

$$\boldsymbol{N} \;\rightarrow\; 0 \cdot \boldsymbol{N} \mid 1 \cdot \boldsymbol{N} \mid \ldots \mid 0 \mid 1 \mid \ldots \mid 9$$

Operator Precedences

To disambiguate

$$E \;\rightarrow\; E \cdot + \cdot E \mid E \cdot * \cdot E \mid (\cdot E \cdot) \mid N$$

Decide on how many precedence levels, say

highest for $()$, medium for $*$, lowest for $+$

$$\begin{array}{lcl} E_{low} & \rightarrow & E_{med} \cdot + \cdot E_{low} \mid E_{med} \\ E_{med} & \rightarrow & E_{hi} \cdot * \cdot E_{med} \mid E_{hi} \\ E_{hi} & \rightarrow & (\cdot E_{low} \cdot) \mid N \end{array}$$

Operator Precedences

To disambiguate

$$E \;\rightarrow\; E \cdot + \cdot E \mid E \cdot * \cdot E \mid (\cdot E \cdot) \mid N$$

Decide on how many precedence levels, say

highest for $()$, medium for $*$, lowest for $+$

$$\begin{array}{lcl} E_{low} & \rightarrow & E_{med} \cdot + \cdot E_{low} \mid E_{med} \\ E_{med} & \rightarrow & E_{hi} \cdot * \cdot E_{med} \mid E_{hi} \\ E_{hi} & \rightarrow & (\cdot E_{low} \cdot) \mid N \end{array}$$

What happens with $1 + 3 + 4$?

Removing Left-Recursion

The rule for numbers is directly left-recursive:

$$N \;\rightarrow\; N \cdot N \mid 0 \mid 1 \qquad (\ldots)$$

Translate

$$\begin{array}{rcl} N & \rightarrow & N \cdot \alpha \\ & \mid & \beta \end{array} \quad\Rightarrow\quad \begin{array}{rcl} N & \rightarrow & \beta \cdot N' \\ N' & \rightarrow & \alpha \cdot N' \\ & \mid & \epsilon \end{array}$$

Removing Left-Recursion

The rule for numbers is directly left-recursive:

$$N \;\rightarrow\; N \cdot N \mid 0 \mid 1 \quad (\ldots)$$

Translate

$$\begin{array}{rcl} N & \rightarrow & N \cdot \alpha \\ & \mid & \beta \end{array} \quad \Rightarrow \quad \begin{array}{rcl} N & \rightarrow & \beta \cdot N' \\ N' & \rightarrow & \alpha \cdot N' \\ & \mid & \epsilon \end{array}$$

Which means

$$\begin{array}{rcl} N & \rightarrow & 0 \cdot N' \mid 1 \cdot N' \\ N' & \rightarrow & N \cdot N' \mid \epsilon \end{array}$$

Chomsky Normal Form

All rules must be of the form

$$A \rightarrow a$$

or

$$A \rightarrow B \cdot C$$

No rule can contain ϵ.

ϵ-Removal

1. If $A \rightarrow \alpha \cdot B \cdot \beta$ and $B \rightarrow \epsilon$ are in the grammar, then add $A \rightarrow \alpha \cdot \beta$ (iterate if necessary).
2. Throw out all $B \rightarrow \epsilon$.

$$\begin{array}{ll} N & \rightarrow 0 \cdot N' \mid 1 \cdot N' \\ N' & \rightarrow N \cdot N' \mid \epsilon \end{array}$$

$$\begin{array}{ll} N & \rightarrow 0 \cdot N' \mid 1 \cdot N' \mid 0 \mid 1 \\ N' & \rightarrow N \cdot N' \mid N \mid \epsilon \end{array}$$

$$\begin{array}{ll} N & \rightarrow 0 \cdot N' \mid 1 \cdot N' \mid 0 \mid 1 \\ N' & \rightarrow N \cdot N' \mid N \end{array}$$

ϵ-Removal

1. If $A \rightarrow \alpha \cdot B \cdot \beta$ and $B \rightarrow \epsilon$ are in the grammar, then add $A \rightarrow \alpha \cdot \beta$ (iterate if necessary).
2. Throw out all $B \rightarrow \epsilon$.

$$\begin{array}{ll} N & \rightarrow 0 \cdot N' \mid 1 \cdot N' \\ N' & \rightarrow N \cdot N' \mid \epsilon \end{array}$$

$$\begin{array}{ll} N & \rightarrow 0 \cdot N' \mid 1 \cdot N' \mid 0 \mid 1 \\ N' & \rightarrow N \cdot N' \mid N \mid \epsilon \end{array}$$

$$\begin{array}{ll} N & \rightarrow 0 \cdot N' \mid 1 \cdot N' \mid 0 \mid 1 \\ N' & \rightarrow N \cdot N' \mid N \end{array}$$

$$N \rightarrow 0 \cdot N \mid 1 \cdot N \mid 0 \mid 1$$

CYK Algorithm

If grammar is in Chomsky normalform ...

$$
\begin{array}{lcl}
S & \rightarrow & N \cdot P \\
P & \rightarrow & V \cdot N \\
N & \rightarrow & N \cdot N \\
N & \rightarrow & \texttt{students} \mid \texttt{Jeff} \mid \texttt{geometry} \mid \texttt{trains} \\
V & \rightarrow & \texttt{trains}
\end{array}
$$

`Jeff trains geometry students`

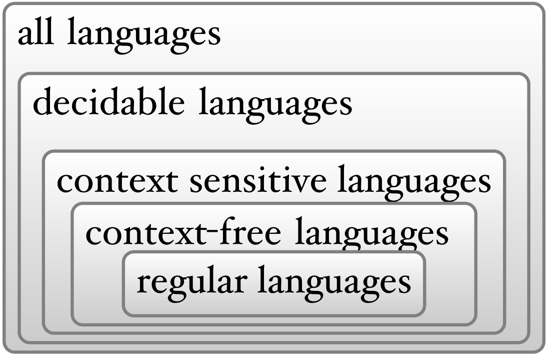
CYK Algorithm

- fastest possible algorithm for recognition problem
- runtime is $O(n^3)$

- grammars need to be transferred into CNF

Hierarchy of Languages

Recall that languages are sets of strings.



Context Sensitive Grms

$$\begin{array}{lcl} S & \Rightarrow & bSAA \mid \epsilon \\ A & \Rightarrow & a \\ bA & \Rightarrow & Ab \end{array}$$

Context Sensitive Grms

$$\begin{array}{lcl} S & \Rightarrow & bSAA \mid \epsilon \\ A & \Rightarrow & a \\ bA & \Rightarrow & Ab \end{array}$$

$$S \Rightarrow \ldots \Rightarrow^{?} \text{"}ababaa\text{"}$$

$$
\begin{array}{lcl}
\mathit{Stmt} & \rightarrow & \texttt{skip} \\
 & | & \mathit{Id}\ \texttt{:=}\ \mathit{AExp} \\
 & | & \texttt{if}\ \mathit{BExp}\ \texttt{then}\ \mathit{Block}\ \texttt{else}\ \mathit{Block} \\
 & | & \texttt{while}\ \mathit{BExp}\ \texttt{do}\ \mathit{Block} \\
 & | & \texttt{read}\ \mathit{Id} \\
 & | & \texttt{write}\ \mathit{Id} \\
 & | & \texttt{write}\ \mathit{String} \\
\mathit{Stmts} & \rightarrow & \mathit{Stmt}\ \texttt{;}\ \mathit{Stmts} \\
 & | & \mathit{Stmt} \\
\mathit{Block} & \rightarrow & \{\ \mathit{Stmts}\ \} \\
 & | & \mathit{Stmt} \\
\mathit{AExp} & \rightarrow & \ldots \\
\mathit{BExp} & \rightarrow & \ldots
\end{array}
$$

```
/* Fibonacci Program
   input: n */

write ”Fib”;
read n;    // n := 19;
minus1 := 0;
minus2 := 1;
while n > 0 do {
       temp := minus2;
       minus2 := minus1 + minus2;
       minus1 := temp;
       n := n - 1
};
write ”Result”;
write minus2
```

An Interpreter

$$
\begin{array}{l}
\{ \\
\quad x := 5; \\
\quad y := x * 3; \\
\quad y := x * 4; \\
\quad x := u * 3 \\
\}
\end{array}
$$

- the interpreter has to record the value of x before assigning a value to y

An Interpreter

$$
\begin{array}{l}
\{ \\
\quad x := 5; \\
\quad y := x * 3; \\
\quad y := x * 4; \\
\quad x := u * 3 \\
\}
\end{array}
$$

- the interpreter has to record the value of x before assigning a value to y
- eval(stmt, env)

Interpreter

$$
\begin{array}{lcl}
\text{eval}(n, E) & \stackrel{\text{def}}{=} & n \\
\text{eval}(x, E) & \stackrel{\text{def}}{=} & E(x) \quad \text{lookup } x \text{ in } E \\
\text{eval}(a_1 + a_2, E) & \stackrel{\text{def}}{=} & \text{eval}(a_1, E) + \text{eval}(a_2, E) \\
\text{eval}(a_1 - a_2, E) & \stackrel{\text{def}}{=} & \text{eval}(a_1, E) - \text{eval}(a_2, E) \\
\text{eval}(a_1 * a_2, E) & \stackrel{\text{def}}{=} & \text{eval}(a_1, E) * \text{eval}(a_2, E) \\
\\
\text{eval}(a_1 = a_2, E) & \stackrel{\text{def}}{=} & \text{eval}(a_1, E) = \text{eval}(a_2, E) \\
\text{eval}(a_1\, != a_2, E) & \stackrel{\text{def}}{=} & \neg(\text{eval}(a_1, E) = \text{eval}(a_2, E)) \\
\text{eval}(a_1 < a_2, E) & \stackrel{\text{def}}{=} & \text{eval}(a_1, E) < \text{eval}(a_2, E)
\end{array}
$$

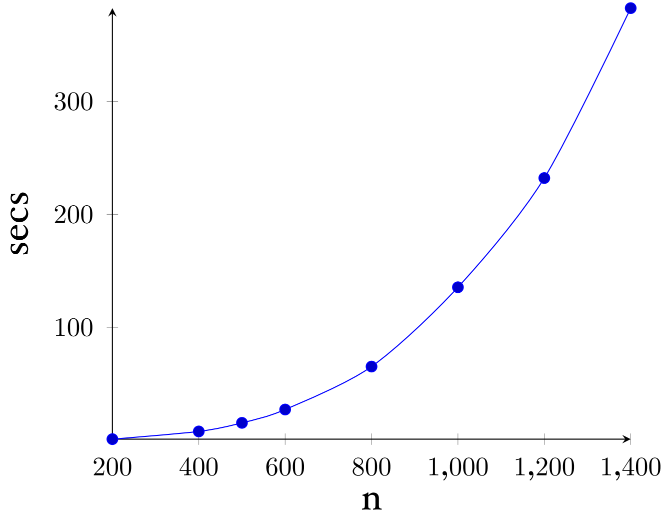
Interpreter (2)

$$
\begin{array}{l}
\mathrm{eval}(\mathrm{skip}, \boldsymbol{E}) \overset{\mathrm{def}}{=} \boldsymbol{E} \\
\mathrm{eval}(\boldsymbol{x} := \boldsymbol{a}, \boldsymbol{E}) \overset{\mathrm{def}}{=} \boldsymbol{E}(\boldsymbol{x} \mapsto \mathrm{eval}(\boldsymbol{a}, \boldsymbol{E})) \\
\mathrm{eval}(\mathrm{if}\ \boldsymbol{b}\ \mathrm{then}\ \boldsymbol{cs}_1\ \mathrm{else}\ \boldsymbol{cs}_2, \boldsymbol{E}) \overset{\mathrm{def}}{=} \\
\qquad \mathrm{if}\ \mathrm{eval}(\boldsymbol{b}, \boldsymbol{E})\ \mathrm{then}\ \mathrm{eval}(\boldsymbol{cs}_1, \boldsymbol{E}) \\
\qquad \qquad \qquad \qquad \mathrm{else}\ \mathrm{eval}(\boldsymbol{cs}_2, \boldsymbol{E}) \\
\mathrm{eval}(\mathrm{while}\ \boldsymbol{b}\ \mathrm{do}\ \boldsymbol{cs}, \boldsymbol{E}) \overset{\mathrm{def}}{=} \\
\qquad \mathrm{if}\ \mathrm{eval}(\boldsymbol{b}, \boldsymbol{E}) \\
\qquad \mathrm{then}\ \mathrm{eval}(\mathrm{while}\ \boldsymbol{b}\ \mathrm{do}\ \boldsymbol{cs}, \mathrm{eval}(\boldsymbol{cs}, \boldsymbol{E})) \\
\qquad \mathrm{else}\ \boldsymbol{E} \\
\mathrm{eval}(\mathrm{write}\ \boldsymbol{x}, \boldsymbol{E}) \overset{\mathrm{def}}{=} \{\ \mathrm{println}(\boldsymbol{E}(\boldsymbol{x}))\ ;\ \boldsymbol{E}\ \}
\end{array}
$$

Test Program

```
start := 1000;    // start value
x := start;
y := start;
z := start;
while 0 < x do {
 while 0 < y do {
  while 0 < z do { z := z - 1 };
  z := start;
  y := y - 1
 };
 y := start;
 x := x - 1
}
```

Interpreted Code



Java Virtual Machine

- introduced in 1995
- is a stack-based VM (like Postscript, CLR of .Net)
- contains a JIT compiler
- many languages take advantage of JVM's infrastructure (JRE)
- is garbage collected $\Rightarrow$ no buffer overflows
- some languages compile to the JVM: Scala, Clojure...