Compilers and Formal Languages (7)

Email: christian.urban at kcl.ac.uk
Office: N7.07 (North Wing, Bush House)
Slides: KEATS (also home work is there)

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).

CFGs

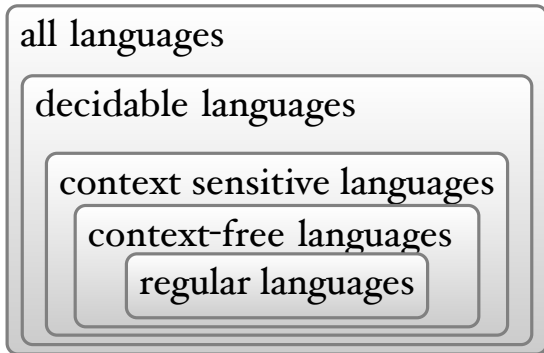A **context-free** grammar (CFG) G consists of:

- 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.

all languages
- decidable languages
  - context sensitive languages
    - context-free languages
      - regular languages

Arithmetic Expressions

A grammar for arithmetic expressions and numbers:

$$E \;\rightarrow\; E \cdot + \cdot E \mid E \cdot * \cdot E \mid (\cdot E \cdot) \mid N$$
$$N \;\rightarrow\; N \cdot N \mid 0 \mid 1 \mid \ldots \mid 9$$

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:

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

Unfortunately it is left-recursive (and ambiguous).

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

Numbers

$$N \;\rightarrow\; N \cdot N \mid 0 \mid 1 \mid \ldots \mid 9$$

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

$$N \;\rightarrow\; 0 \cdot N \mid 1 \cdot N \mid \ldots \mid 0 \mid 1 \mid \ldots \mid 9$$

Operator Precedences

To disambiguate

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

Decide on how many precedence levels, say

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

$$E_{low} \quad \rightarrow \quad E_{med} \cdot + \cdot E_{low} \mid E_{med}$$
$$E_{med} \quad \rightarrow \quad E_{hi} \cdot * \cdot E_{med} \mid E_{hi}$$
$$E_{hi} \quad \rightarrow \quad (\cdot E_{low} \cdot) \mid N$$

Operator Precedences

To disambiguate

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

Decide on how many precedence levels, say highest for (), medium for *, lowest for +

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

What happens with $1 + 3 + 4$?

Removing Left-Recursion

The rule for numbers is directly left-recursive:

$$N \;\to\; N \cdot N \mid 0 \mid 1 \quad (\dots)$$

Translate

$$
\begin{aligned}
N \;&\to\; N \cdot \alpha \\
&\mid\; \beta
\end{aligned}
\quad\Longrightarrow\quad
\begin{aligned}
N \;&\to\; \beta \cdot N' \\
N' \;&\to\; \alpha \cdot N' \\
&\mid\; \epsilon
\end{aligned}
$$

Removing Left-Recursion

The rule for numbers is directly left-recursive:

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

Translate

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

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 $B \rightarrow \epsilon$.

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

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

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

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 $B \rightarrow \epsilon$.

$$N \rightarrow \mathsf{o} \cdot N' \mid \mathsf{I} \cdot N'$$
$$N' \rightarrow N \cdot N' \mid \epsilon$$

$$N \rightarrow \mathsf{o} \cdot N' \mid \mathsf{I} \cdot N' \mid \mathsf{o} \mid \mathsf{I}$$
$$N' \rightarrow N \cdot N' \mid N \mid \epsilon$$

$$N \rightarrow \mathsf{o} \cdot N' \mid \mathsf{I} \cdot N' \mid \mathsf{o} \mid \mathsf{I}$$
$$N' \rightarrow N \cdot N' \mid N$$

$$N \rightarrow \mathsf{o} \cdot N \mid \mathsf{I} \cdot N \mid \mathsf{o} \mid \mathsf{I}$$

CYK Algorithm

If grammar is in Chomsky normalform ...

$$S \rightarrow N \cdot P$$
$$P \rightarrow V \cdot N$$
$$N \rightarrow N \cdot N$$
$$N \rightarrow \text{students} \mid \text{Jeff} \mid \text{geometry} \mid \text{trains}$$
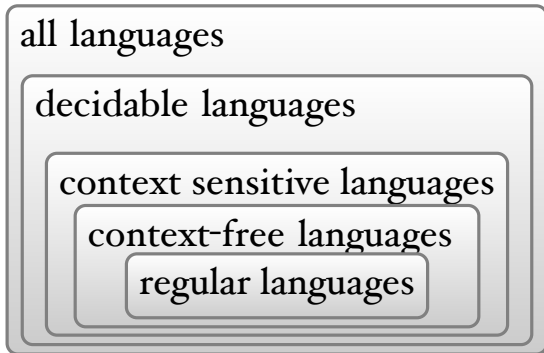$$V \rightarrow \text{trains}$$

`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

$$S \quad \Rightarrow \quad bSAA \mid \epsilon$$
$$A \quad \Rightarrow \quad a$$
$$bA \quad \Rightarrow \quad Ab$$

Context Sensitive Grms

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

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

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

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

An Interpreter

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

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

An Interpreter

$$\{$$
$$x := 5;$$
$$y := x * 3;$$
$$y := x * 4;$$
$$x := u * 3$$
$$\}$$

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

Interpreter

$$\text{eval}(n, E) \quad \stackrel{\text{def}}{=} \quad n$$

$$\text{eval}(x, E) \quad \stackrel{\text{def}}{=} \quad E(x) \quad \text{lookup } x \text{ in } E$$

$$\text{eval}(a_1 + a_2, E) \quad \stackrel{\text{def}}{=} \quad \text{eval}(a_1, E) + \text{eval}(a_2, E)$$

$$\text{eval}(a_1 - a_2, E) \quad \stackrel{\text{def}}{=} \quad \text{eval}(a_1, E) - \text{eval}(a_2, E)$$

$$\text{eval}(a_1 * a_2, E) \quad \stackrel{\text{def}}{=} \quad \text{eval}(a_1, E) * \text{eval}(a_2, E)$$

$$\text{eval}(a_1 = a_2, E) \quad \stackrel{\text{def}}{=} \quad \text{eval}(a_1, E) = \text{eval}(a_2, E)$$

$$\text{eval}(a_1 \,!= a_2, E) \quad \stackrel{\text{def}}{=} \quad \neg(\text{eval}(a_1, E) = \text{eval}(a_2, E))$$

$$\text{eval}(a_1 < a_2, E) \quad \stackrel{\text{def}}{=} \quad \text{eval}(a_1, E) < \text{eval}(a_2, E)$$

Interpreter (2)

$$\text{eval}(\text{skip}, E) \quad \overset{\text{def}}{=} \quad E$$

$$\text{eval}(x := a, E) \quad \overset{\text{def}}{=} \quad E(x \mapsto \text{eval}(a, E))$$

$$\text{eval}(\text{if } b \text{ then } cs_1 \text{ else } cs_2, E) \overset{\text{def}}{=}$$
$$\quad\quad \text{if eval}(b, E) \text{ then eval}(cs_1, E)$$
$$\quad\quad\quad\quad \text{else eval}(cs_2, E)$$

$$\text{eval}(\text{while } b \text{ do } cs, E) \overset{\text{def}}{=}$$
$$\quad\quad \text{if eval}(b, E)$$
$$\quad\quad \text{then eval}(\text{while } b \text{ do } cs, \text{eval}(cs, E))$$
$$\quad\quad \text{else } E$$

$$\text{eval}(\text{write } x, E) \quad \overset{\text{def}}{=} \quad \{ \text{ println}(E(x)) \text{ ; } E \}$$

Test Program

```
start := 1000;
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
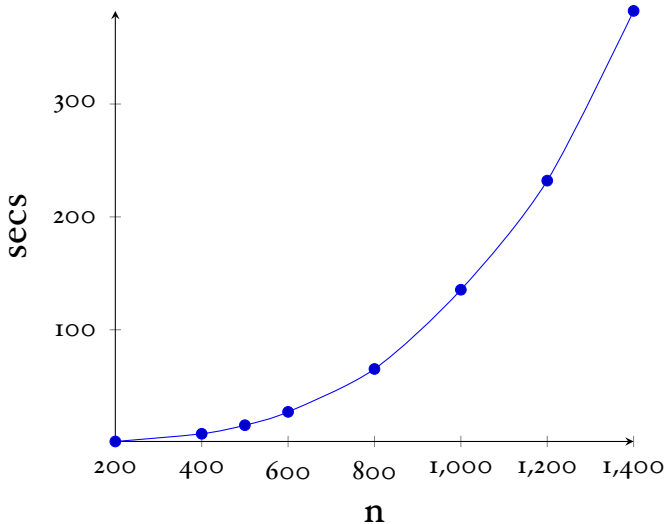
Java Virtual Machine

- 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 ⇒ no buffer overflows
- some languages compile to the JVM: Scala, Clojure...