

Compilers and Formal Languages

Email: christian.urban at kcl.ac.uk

Office Hour: Thursdays 15 – 16

Location: N7.07 (North Wing, Bush House)

Slides & Progs: KEATS

Pollev: <https://pollev.com/cfltutoratki576>

1 Introduction, Languages	6 While-Language
2 Regular Expressions, Derivatives	7 Compilation, JVM
3 Automata, Regular Languages	8 Compiling Functional Languages
4 Lexing, Tokenising	9 Optimisations
5 Grammars, Parsing	10 LLVM

Coursework 1: Submissions

- Scala (162)
- Ocaml (1)
- Java (1) ...uses new features of Java 21
- Rust (6)

Parser



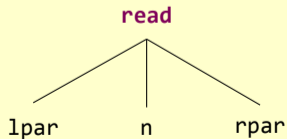
Parser



parser input: a sequence of tokens

key(**read**) lpar id(n) rpar semi

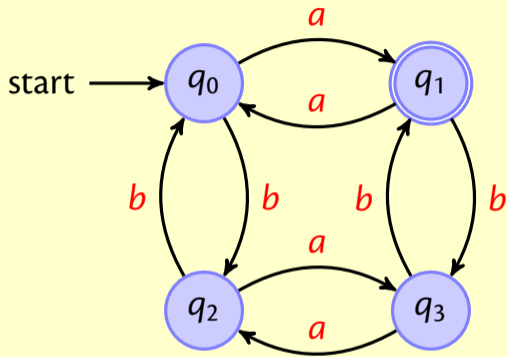
parser output: an abstract syntax tree



What Parsing is Not

Usually parsing does not check semantic correctness, e.g.

- whether a function is not used before it is defined
- whether a function has the correct number of arguments or are of correct type
- whether a variable can be declared twice in a scope



Which language?

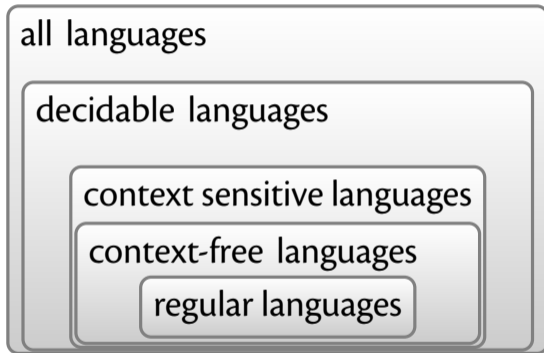
Regular Languages

While regular expressions are very useful for lexing, there is no regular expression that can recognise the language $a^n b^n$.

$((((()))))$ vs. $((((())) ()))$

So we cannot find out with regular expressions whether parentheses are matched or unmatched. Also regular expressions are not recursive, e.g. $(1 + 2) + 3$.

Hierarchy of Languages



Time flies like an arrow.
Fruit flies like bananas.

CFGs

A **context-free grammar** G consists of

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

$$A ::= rhs$$

where rhs are sequences involving terminals and nonterminals, including the empty sequence ϵ .

CFGs

A **context-free grammar** G consists of

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

$$A ::= rhs$$

where rhs are sequences involving terminals and nonterminals, including the empty sequence ϵ .

We also allow rules

$$A ::= rhs_1 | rhs_2 | \dots$$

Palindromes

A grammar for palindromes over the alphabet $\{a, b\}$:

$$S ::= a \cdot S \cdot a$$

$$S ::= b \cdot S \cdot b$$

$$S ::= a$$

$$S ::= b$$

$$S ::= \epsilon$$

Palindromes

A grammar for palindromes over the alphabet $\{a, b\}$:

$$S ::= a \cdot S \cdot a \mid b \cdot S \cdot b \mid a \mid b \mid \epsilon$$

Arithmetic Expressions

$$\begin{aligned} E ::= & 0 \mid 1 \mid 2 \mid \dots \mid 9 \\ & \mid E \cdot + \cdot E \\ & \mid E \cdot - \cdot E \\ & \mid E \cdot * \cdot E \\ & \mid (\cdot E \cdot) \end{aligned}$$

Arithmetic Expressions

$$\begin{aligned} E ::= & 0 \mid 1 \mid 2 \mid \dots \mid 9 \\ & \mid E \cdot + \cdot E \\ & \mid E \cdot - \cdot E \\ & \mid E \cdot * \cdot E \\ & \mid (\cdot E \cdot) \end{aligned}$$

1 + 2 * 3 + 4

A CFG Derivation

1. Begin with a string containing only the start symbol, say S
2. Replace any nonterminal X in the string by the right-hand side of some production $X ::= rhs$
3. Repeat 2 until there are no nonterminals left

$$S \rightarrow \dots \rightarrow \dots \rightarrow \dots \rightarrow \dots$$

Example Derivation

$$S ::= \epsilon \mid a \cdot S \cdot a \mid b \cdot S \cdot b$$

$$\begin{aligned} S &\rightarrow aSa \\ &\rightarrow abSba \\ &\rightarrow abaSaba \\ &\rightarrow abaaba \end{aligned}$$

Example Derivation

$$\begin{aligned} E ::= & 0 \mid 1 \mid 2 \mid \dots \mid 9 \\ & \mid E \cdot + \cdot E \\ & \mid E \cdot - \cdot E \\ & \mid E \cdot * \cdot E \\ & \mid (\cdot E \cdot) \end{aligned}$$

$$\begin{aligned} E &\rightarrow E * E \\ &\rightarrow E + E * E \\ &\rightarrow E + E * E + E \\ &\rightarrow^+ 1 + 2 * 3 + 4 \end{aligned}$$

Example Derivation

$E ::= 0 \mid 1 \mid 2 \mid \dots \mid 9$

$\mid E \cdot + \cdot E$

$\mid E \cdot - \cdot E$

$\mid E \cdot * \cdot E$

$\mid (\cdot E \cdot)$

$E \rightarrow E * E$

$\rightarrow E + E * E$

$\rightarrow E + E * E + E$

$\rightarrow^+ 1 + 2 * 3 + 4$

$E \rightarrow E + E$

$\rightarrow E + E + E$

$\rightarrow E + E * E + E$

$\rightarrow^+ 1 + 2 * 3 + 4$

Language of a CFG

Let G be a context-free grammar with start symbol S .
Then the language $L(G)$ is:

$$\{c_1 \dots c_n \mid \forall i. c_i \in T \wedge S \rightarrow^* c_1 \dots c_n\}$$

Language of a CFG

Let G be a context-free grammar with start symbol S .
Then the language $L(G)$ is:

$$\{c_1 \dots c_n \mid \forall i. c_i \in T \wedge S \rightarrow^* c_1 \dots c_n\}$$

- Terminals, because there are no rules for replacing them.
- Once generated, terminals are “permanent”.
- Terminals ought to be tokens of the language (but can also be strings).

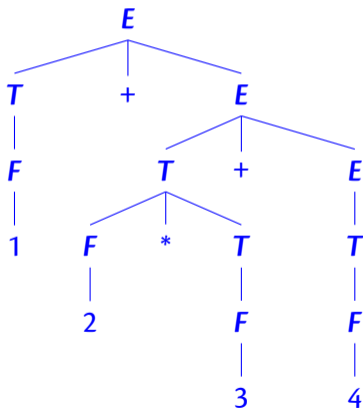
Parse Trees

$E ::= T \mid T \cdot + \cdot E \mid T \cdot - \cdot E$

$T ::= F \mid F \cdot * \cdot T$

$F ::= 0\dots9 \mid (\cdot E \cdot)$

1 + 2 * 3 + 4



Arithmetic Expressions

$E ::= 0..9$

| $E \cdot + \cdot E$

| $E \cdot - \cdot E$

| $E \cdot * \cdot E$

| $(\cdot E \cdot)$

Arithmetic Expressions

$$\begin{aligned} E ::= & 0..9 \\ & | E \cdot + \cdot E \\ & | E \cdot - \cdot E \\ & | E \cdot * \cdot E \\ & | (\cdot E \cdot) \end{aligned}$$

A CFG is **left-recursive** if it has a nonterminal E such that $E \rightarrow^+ E \cdot \dots$

Ambiguous Grammars

A grammar is **ambiguous** if there is a string that has at least two different parse trees.

$$\begin{aligned} E ::= & 0\dots9 \\ & | E \cdot + \cdot E \\ & | E \cdot - \cdot E \\ & | E \cdot * \cdot E \\ & | (\cdot E \cdot) \end{aligned}$$

1 + 2 * 3 + 4

'Dangling' Else

Another ambiguous grammar:

$$\begin{array}{l} E \rightarrow \text{if } E \text{ then } E \\ \quad | \quad \text{if } E \text{ then } E \text{ else } E \\ \quad | \quad \dots \end{array}$$

`if a then if x then y else c`

CYK Algorithm

Suppose the grammar:

$S ::= N \cdot P$

$P ::= V \cdot N$

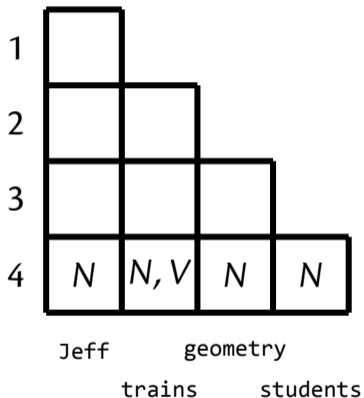
$N ::= N \cdot N$

$N ::= \text{students} \mid \text{Jeff} \mid \text{geometry} \mid \text{trains}$

$V ::= \text{trains}$

Jeff trains geometry students

CYK Algorithm



$S ::= N \cdot P$
 $P ::= V \cdot N$
 $N ::= N \cdot N$
 $N ::= \text{students} \mid \text{Jeff}$
 $\quad \quad \quad \mid \text{geometry} \mid \text{trains}$
 $V ::= \text{trains}$

Chomsky Normal Form

A grammar for palindromes over the alphabet $\{a, b\}$:

$$S ::= a \cdot S \cdot a \mid b \cdot S \cdot b \mid a \cdot a \mid b \cdot b \mid a \mid b$$

CYK Algorithm

- fastest possible algorithm for recognition problem
- runtime is $O(n^3)$
- grammars need to be transformed into CNF

"The C++ grammar is ambiguous, context-dependent and potentially requires infinite lookahead to resolve some ambiguities."

from the [PhD thesis](#) by Willink (2001)

```
int(x), y, *const z;  
int(x), y, new int;
```

Context Sensitive Grammars

It is much harder to find out whether a string is parsed by a context sensitive grammar:

$$S ::= bSAA \mid \epsilon$$

$$A ::= a$$

$$bA ::= Ab$$

Context Sensitive Grammars

It is much harder to find out whether a string is parsed by a context sensitive grammar:

$$S ::= bSAA \mid \epsilon$$

$$A ::= a$$

$$bA ::= Ab$$

$$S \rightarrow \dots \rightarrow? ababaa$$

For CW2, please include `'\'` as a symbol in strings, because the collatz program contains

```
write "\n";
```

$\text{val } (r1s, f1s) = \text{simp}(r1)$

$\text{val } (r2s, f2s) = \text{simp}(r2)$

how are the first rectification functions $f1s$ and $f2s$ made? could you maybe show an example?

Questions regarding CFL CW1

Dear Dr Urban

Regarding CW1, I am stuck on finding the nullable and derivative rules for some important regexes.

The NOT Regex nullable rule: I am not sure how to approach this, I am inclined to simply put this as the negation of the nullable function on the input regex (e.g $!\text{nullable}(r)$). However I have found instances where negating a nullable does not make it un-nullable. For example the negation of r^* can still match regex ab (which is not nullable). So I would like some actual clarification, pointers and help in this area.

The NOT Regex derivation rule: again I am dumbfounded here, I am inclined to think that I should derive the regex and then negate that derivation. But none of this ever works. Please provide some helpful information so I can solve this.