Compiler Construction

Chapter 2: Grammars

OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

Formal Language

Formal Language L:

- Set of words over alphabet Σ
- $L \subset \Sigma^*$
- Words follow rules called **grammar**

aaaaaaa

aaaabb

abbbbbbbb

bbbb

ab

Formal Grammar

Grammar defined by: A set of production rules

Example:

$$S \rightarrow A$$
$$A \rightarrow aA$$
$$A \rightarrow B$$
$$B \rightarrow bB$$
$$B \rightarrow \epsilon$$

Non-Terminals N: { S, A, B }

Non-terminals are to be replaced by rules

Terminals Σ: { a, b }

A final word is only constituted from terminal symbols. No non-terminals may be part of a final word.

Special terminal: ϵ

ϵ denotes the empty word

Formal Grammar

Quadrupel: G := (Σ, N, P, S)

- Σ: Set of terminal symbols
- N: Set of non-terminal symbols
- P: Set of production rules
- S: Start symbol ($S \in N$)

$$
\begin{aligned}
S &\rightarrow A \\
A &\rightarrow aA \\
A &\rightarrow B \\
B &\rightarrow bB \\
B &\rightarrow \epsilon
\end{aligned}
$$

$$(\ \{a, b\}\ ,\ \{A, B, S\}\ ,\ \{S \rightarrow A, \ldots\}\ ,\ S\ )$$

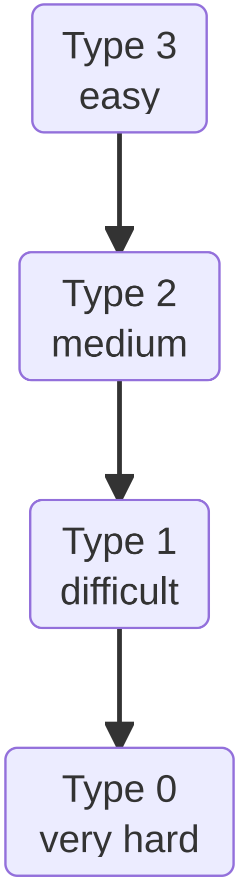
Formal Grammar

Chomsky hierarchy of grammars:
Varying difficulty to parse

Depends on: Format of production rules

LeftHandSide → RightHandSide

Different restrictions for what is allowed on Lefthand-Side and Righthand-Side for grammar types 0 - 3



Formal Grammar

General form of production rules:

$$\alpha \to \beta \text{ with } \alpha, \beta \in (N \cup \Sigma)^* \land \alpha \neq \epsilon$$

(ϵ is the empty word)

Expressed differently:

$$\alpha \to \beta \text{ with } \alpha \in (N \cup \Sigma)^+, \beta \in (N \cup \Sigma)^*$$

(+/* is Kleene plus/star)

ℹ A Type 0 (unrestricted) grammar allows all rules of that form

Type 0: Unrestricted

Rule Format: $(N \cup \Sigma)^+ \rightarrow (N \cup \Sigma)^*$

Example: apbqc-system

$$G := (\{a, b, c, p, q\}, \{X, Y\}, P, X)$$

1. $X \rightarrow aXc$
2. $X \rightarrow pY$
3. $Y \rightarrow bYc$
4. $Y \rightarrow q$

X

(1) aXc

(1) aaXcc

(2) aapYcc

(3) aapbYccc

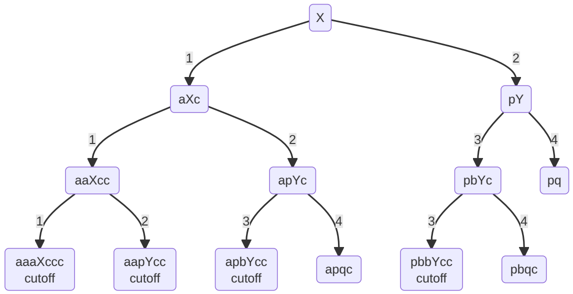
(3) aapbbYcccc

(4) aapbbqcccc

...

Type 0: Unrestricted

How to find out if the word apbqc is derivable in G?



1. $X \rightarrow aXc$
2. $X \rightarrow pY$
3. $Y \rightarrow bYc$
4. $Y \rightarrow q$

Enumerate all candidate words by doing a breadth-first search on the grammar tree. Stop on nodes, where the number of terminals exceeds the word length.

Type 0: Unrestricted

Rule Format: $(N \cup \Sigma)^+ \to (N \cup \Sigma)^*$

Example: apbqc-system

$$G := (\{a, b, c, p, q\}, \{X, Y\}, P, X)$$

1. $X \to aXc$
2. $X \to pY$
3. $Y \to bYc$
4. $Y \to q$
5. $apb \to pbb$
6. $apb \to aap$
7. $bqc \to Y$

X

(1) aXc

(1) aaXcc

(2) aapYcc

(3) aapbYccc

(3) aapbbYcccc

(4) aapbbqcccc

(5) apbbbqcccc

(7) apbbYccc

(4) apbbqccc

(7) apbYcc

(6) aapYcc

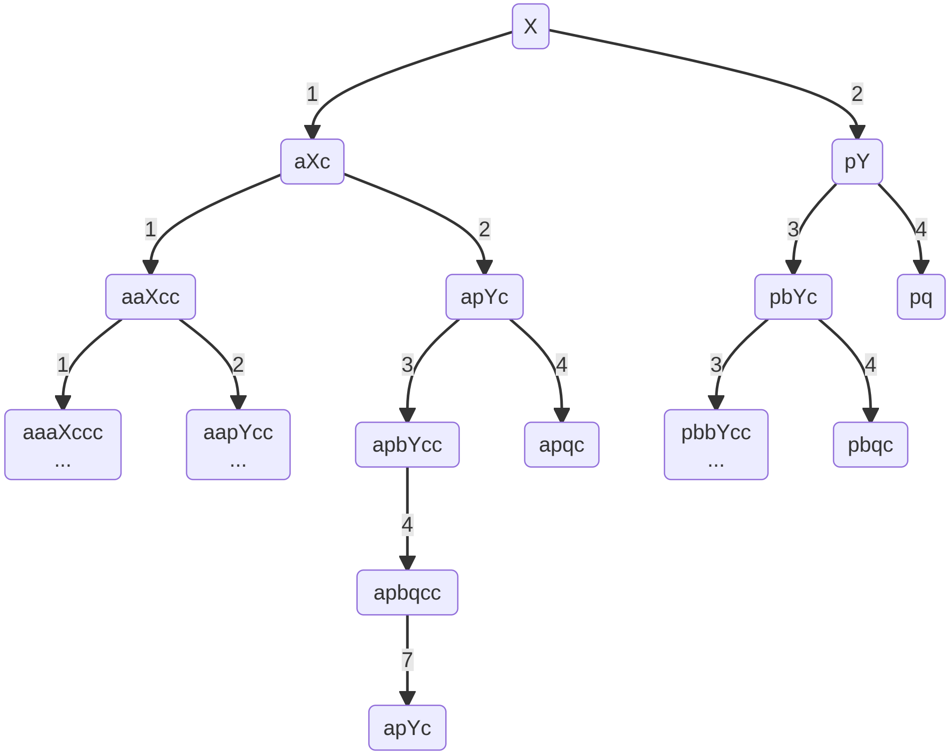
(4) aapqcc

…

Type 0: Unrestricted

Why is it hard to decide, if a given word can be produced with G?

- Words can shrink
- Because: Rule 7
- No clear decision at which point a word definitively cannot be produced any more

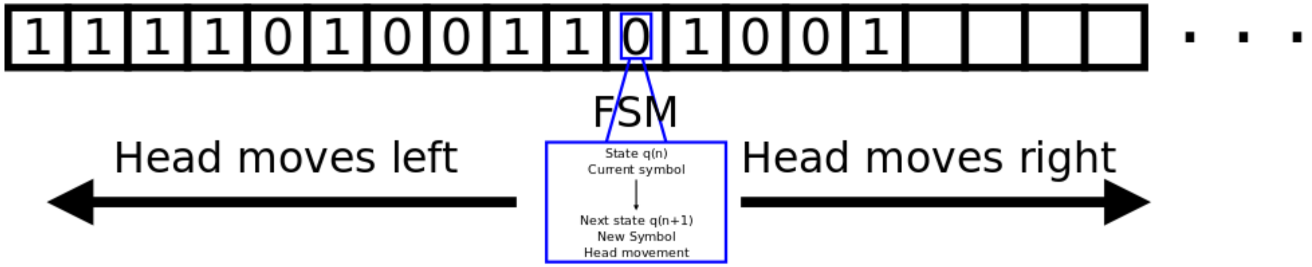


1. $X \rightarrow aXc$
2. $X \rightarrow pY$
3. $Y \rightarrow bYc$
4. $Y \rightarrow q$
5. $apb \rightarrow pbb$
6. $apb \rightarrow aap$
7. $bqc \rightarrow Y$

Type 0: Unrestricted

What is needed for parsing type 0 grammars:

- Our most powerful machine model: Turing Machine
- A grammar may be undecidable:
  - Turing machine halts after indefinite time if the word can be parsed
  - Turing machine runs infinitely if not
    (note the connection to the parse tree)



Type 0 grammars are very hard to parse and sometimes undecidable!

Type 1: Context-sensitive

Conclusion: Parsing unrestricted grammars is problematic

Solution: restrict the form of the rules

Type 0: $(\Sigma \cup N)^{+} \rightarrow (\Sigma \cup N)^{*}$

Type 1: $\alpha A \beta \rightarrow \alpha \gamma \beta$
with $\alpha, \beta, \gamma \in (\Sigma \cup N)^{*}$ and $A \in N$

Type 1 is called **Context-sensitive Grammar (CSG)**

Reason: A non-terminal A is replaced by a string of terminals and non-terminals γ depending on the context α and β, which remains constant.

Type 1: Context-sensitive

Type 1: $\alpha A \beta \rightarrow \alpha \gamma \beta$

with $\alpha, \beta, \gamma \in (\Sigma \cup N)^*$ and $A \in N$

Less powerful machine type required to parse:

Linear bounded automaton

Similar to a Turing machine, but length-restricted tape

Still a very powerful machine type: parsing is difficult and inefficient

Type 2: Context-free

Type 1: $\alpha A \beta \rightarrow \alpha \gamma \beta$
with $\alpha, \beta, \gamma \in (\Sigma \cup N)^*$ and $A \in N$

Type 2: $A \rightarrow \gamma$
with $\gamma \in (\Sigma \cup N)^*$ and $A \in N$

Type 2 is called **Context-free Grammar (CFG)**

Reason: A non-terminal A is replaced by a string of terminals and non-terminals γ independent of its context where it appears

Type 2: Context-free

Type 2: $A \rightarrow \gamma$ with $\gamma \in (\Sigma \cup N)^*$ and $A \in N$

1. $X \rightarrow aXc$
2. $X \rightarrow pY$
3. $Y \rightarrow bYc$
4. $Y \rightarrow q$

- pq
- apqc
- pbqc
- apbbqccc
- aapbqccc
- aaapbbqccccc

❓ How could p and q be pronounced?
(try to find an isomorphism to another concept)

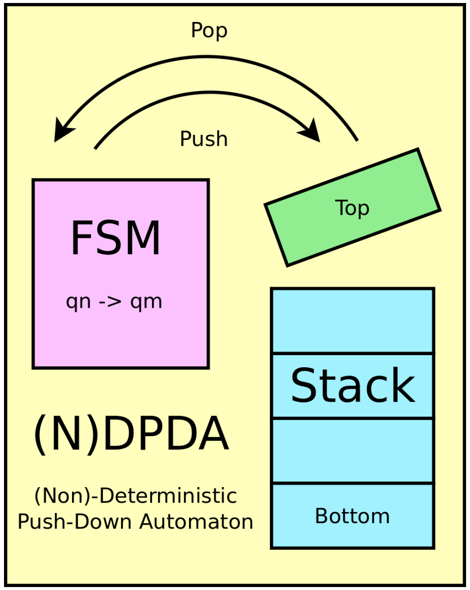
Type 2: Context-free

Machine needed: (Non-)Deterministic Pushdown Automaton ((N)DPDA)

Finite State Machine with a stack

Subtype: Deterministic context-free languages/grammars

DCFG are efficient to parse with a DPDA



Type 2: Context-free

1. $X \rightarrow aXc$
2. $X \rightarrow pY$
3. $Y \rightarrow bYc$
4. $Y \rightarrow q$
5. $apb \rightarrow pbb$
6. $apb \rightarrow aap$
7. $bqc \rightarrow Y$

This grammar is not context-free!

But: Generates the same (context-free) language as rules 1-4 alone

Languages and Grammars

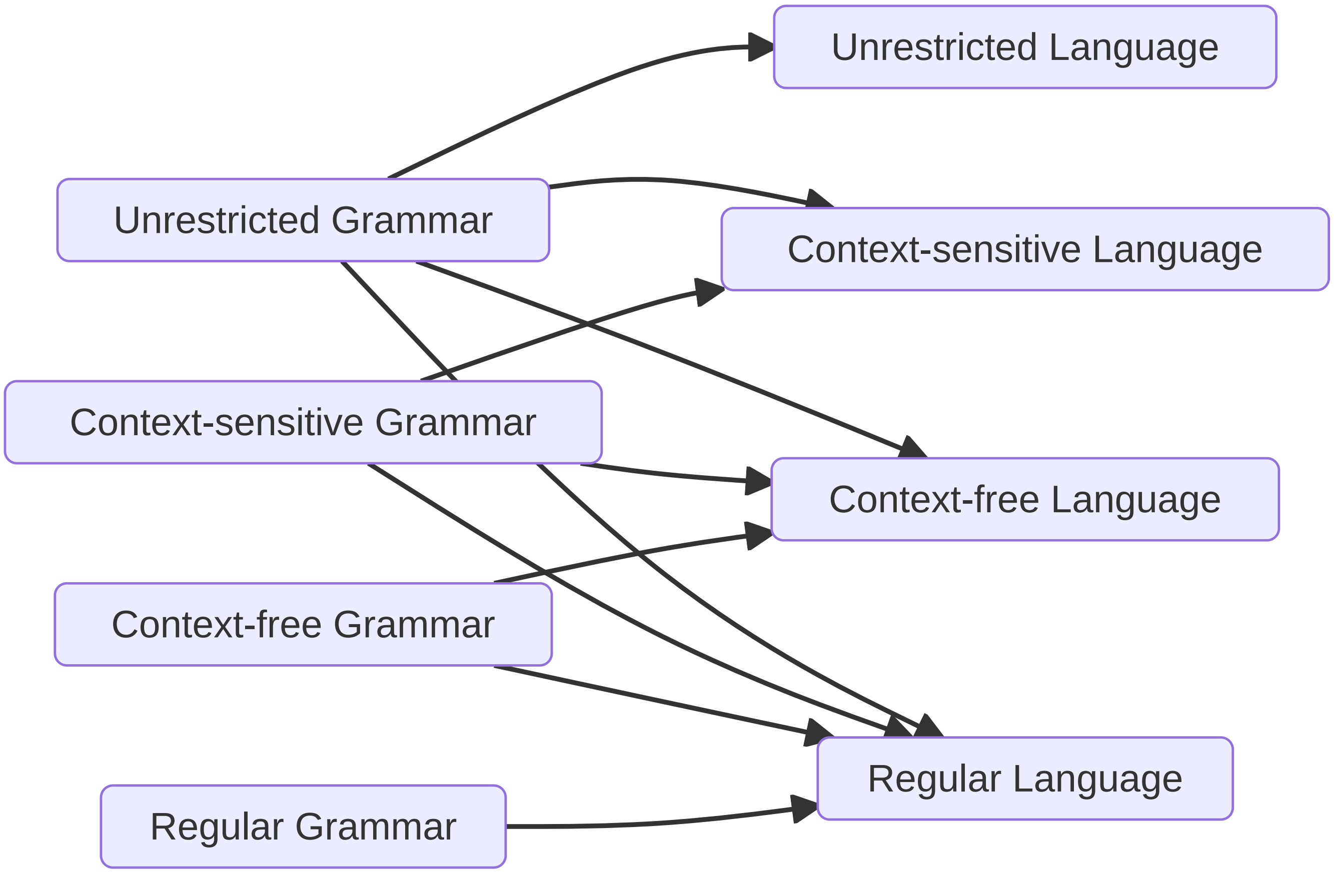
→ A Type n language can be produced by a Type n grammar

But: a Type n grammar can produce a Type m language with $n \leq m$

Example: An unrestricted grammar can produce a context-free language

→ Sometimes a grammar can be modified to increase its type (e.g. from unrestricted to context-free) without changing the language it generates

Languages and Grammars



Type 3: Regular grammar

Type 2: $A \rightarrow \gamma$ with $\gamma \in (\Sigma \cup N)^*$ and $A \in N$

Type 3 (L): $\boldsymbol{A \rightarrow w(B)}$ $\boldsymbol{A}, \boldsymbol{B} \in N$ and $w \in \Sigma^*$

Type 3 (R): $\boldsymbol{A \rightarrow (B)w}$ $\boldsymbol{A}, \boldsymbol{B} \in N$ and $w \in \Sigma^*$

The non-terminal B on the right side is optional.
Left- and right-regular grammars are equivalent, but rules may not be mixed.

Type 3: Regular grammar

Type 3 (L): $A \rightarrow w(B)$ $A, B \in N$ and $w \in \Sigma^*$

Example:

1. $S \rightarrow 1B$
2. $B \rightarrow 0B$
3. $B \rightarrow 1B$
4. $B \rightarrow +S$
5. $B \rightarrow \epsilon$

- 10010
- 111+101
- 1+1+1+1
- 1000001
- 100+101+102

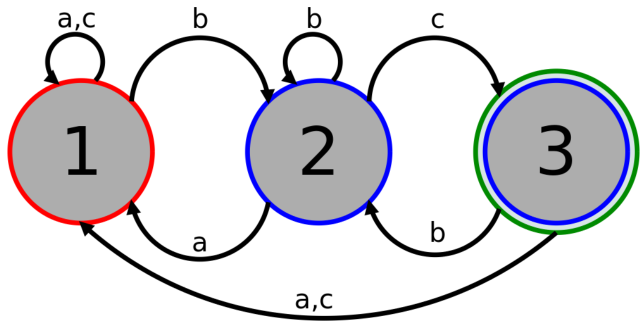
Type 3: Regular grammar

Type 3 (L): $A \rightarrow w(B)$ $A, B \in N$ and $w \in \Sigma^*$

Regular grammars: very easy to parse

Required machine model:

- Finite State Machine (FSM)
- Also called: (Non-)Deterministic Finite Automaton (NFA/DFA)



In contrast to Context-Free Languages:
No difference between deterministic and non-deterministic Regular Languages

Type 3: Regular grammar

Popular notation: **Regular Expressions** (Regex)

Example:

1. $S \rightarrow 1B$
2. $B \rightarrow 0B$
3. $B \rightarrow 1B$
4. $B \rightarrow +S$
5. $B \rightarrow \epsilon$

```
(1(0|1)*+)*1(0|1)*
```

```
(1[01]*+)*1[01]*
```
(UN*X style)

Summary:

- 4 different grammar types, depending on rule restrictions
- Type 0 and 1: Very hard to recognize
- Higher types are easier to parse
- Regular Grammars: Lexer
- (Deterministic) Context-Free Grammars: Parser