Towards more complex grammar systems Some basic formal language theory

- Grammars, or: how to specify linguistic knowledge
- Automata, or: how to process with linguistic knowledge
- Levels of complexity in grammars and automata: The Chomsky hierarchy

Grammars

A grammar is a 4-tuple (N, Σ, S, P) where

- N is a finite set of **non-terminals**
- Σ is a finite set of **terminal symbols**, with $N \cap \Sigma = \emptyset$
- S is a distinguished **start symbol**, with $S \in N$
- P is a finite set of **rewrite rules** of the form $\alpha \rightarrow \beta$, with $\alpha, \beta \in (N \cup \Sigma)*$ and α including at least one non-terminal symbol.

A simple example

$N = \{$S, NP, VP, V$_i$, V$_t$, V$_s\}$

$\Sigma = \{$John, Mary, laughs, loves, thinks$\}$

$S =$ S

$$P = \left\{ \begin{array}{lll} \text{S} & \rightarrow & \text{NP VP} \\ & & \\ \text{NP} & \rightarrow & \text{John} \\ \text{NP} & \rightarrow & \text{Mary} \\ & & \\ \text{VP} & \rightarrow & \text{V}_i \\ \text{VP} & \rightarrow & \text{V}_t \text{ NP} \\ \text{VP} & \rightarrow & \text{V}_s \text{ S} \\ \text{p} & & \\ \text{V}_i & \rightarrow & \text{laughs} \\ \text{V}_t & \rightarrow & \text{loves} \\ \text{V}_s & \rightarrow & \text{thinks} \end{array} \right\}$$

How does a grammar define a language?

Assume $\alpha, \beta \in (N \cup \Sigma)*$, with α containing at least one non-terminal.

- A **sentential form** for a grammar G is defined as:
  - The start symbol S of G is a sentential form.
  - If $\alpha\beta\gamma$ is a sentential form and there is a rewrite rule $\beta \rightarrow \delta$ then $\alpha\delta\gamma$ is a sentential form.
- α (directly or immediately) **derives** β if $\alpha \rightarrow \beta \in P$. One writes:
  - $\alpha \Rightarrow {}^*\beta$ if β is derived from α in zero or more steps
  - $\alpha \Rightarrow {}^+\beta$ if β is derived from α in one or more steps
- A **sentence** is a sentential form consisting only of terminal symbols.
- The **language** $L(G)$ generated by the grammar G is the set of all sentences which can be derived from the start symbol S, i.e., $L(G) = \{\gamma | S \Rightarrow {}^*\gamma\}$

Processing with grammars: automata

An **automaton** in general has three components:

- an **input tape**, divided into squares with a read-write head positioned over one of the squares
- an **auxiliary memory** characterized by two functions
  - fetch: memory configuration $\rightarrow$ symbols
  - store: memory configuration $\times$ symbol $\rightarrow$ memory configuration
- and a **finite-state control** relating the two components.

Different levels of complexity in grammars and automata

Let $A, B \in N$, $x \in \Sigma$, $\alpha, \beta, \gamma \in (\Sigma \cup T)*$, and $\delta \in (\Sigma \cup T)+$, then:

Type	Automaton		Grammar	
	Memory	Name	Rule	Name
0	Unbounded	TM	$\alpha \rightarrow \beta$	General rewrite
1	Bounded	LBA	$\beta\ A\ \gamma \rightarrow \beta\ \delta\ \gamma$	Context-sensitive
2	Stack	PDA	$A \rightarrow \beta$	Context-free
3	None	FSA	$A \rightarrow xB$, $A \rightarrow x$	Right linear

Abbreviations:
- TM: Turing Machine
- LBA: Linear-Bounded Automaton
- PDA: Push-Down Automaton
- FSA: Finite-State Automaton

Type 3: Right-Linear Grammars and Finite-State Automata

A **right-linear grammar** is a 4-tuple (N, Σ, S, P) with

P a finite set of rewrite rules of the form $\alpha \rightarrow \beta$, with $\alpha \in N$ and $\beta \in \{\gamma\delta | \gamma \in \Sigma*, \delta \in N \cup \{\epsilon\}\}$, i.e.:

- left-hand side of rule: a single non-terminal, and
- right-hand side of rule: a string containing at most one non-terminal, as the rightmost symbol

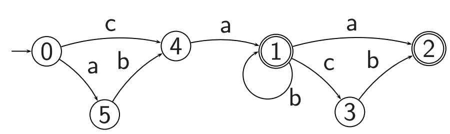
Right-linear grammars are formally equivalent to left-linear grammars (at most one, leftmost non-terminal).

Finite-state transition network: states + arcs

A **finite-state machine** consists of
- a tape
- a finite-state control
- no auxiliary memory

A regular language example:
$(ab|c)ab*(a|cb)?$

Finite-state transition network:



Right-linear grammar:

$N = \{$Expr, X, Y, Z$\}$

$\Sigma = \{$a,b,c$\}$

$S =$ Expr

$$P = \left\{ \begin{array}{lll} \text{Expr} & \rightarrow & \text{ab X} \\ \text{Expr} & \rightarrow & \text{c X} \\ \text{X} & \rightarrow & \text{a Y} \\ \text{Y} & \rightarrow & \text{b Y} \\ \text{Y} & \rightarrow & \text{Z} \\ \text{Z} & \rightarrow & \text{a} \\ \text{Z} & \rightarrow & \text{cb} \\ \text{Z} & \rightarrow & \epsilon \end{array} \right\}$$

Thinking about regular languages

- A language is regular iff one can define a FSM (or regular expression) for it.
- An FSM only has a fixed amount of memory, namely the number of states.
- Strings longer than the number of states, in particular also any infinite ones, must result from a loop in the FSM.
- Pumping Lemma: if for an infinite string there is no such loop, the string cannot be part of a regular language.

Type 2: Context-Free Grammars and Push-Down Automata

A **context-free grammar** is a 4-tuple (N, Σ, S, P) with

P a finite set of rewrite rules of the form $\alpha \rightarrow \beta$, with $\alpha \in N$ and $\beta \in (\Sigma \cup N)*$, i.e.:

- left-hand side of rule: a single non-terminal, and
- right-hand side of rule: a string of terminals and/or non-terminals

A **push-down automaton** is a

- finite state automaton, with a
- stack as auxiliary memory

A context-free language example: $a^n b^n$

Context-free grammar:

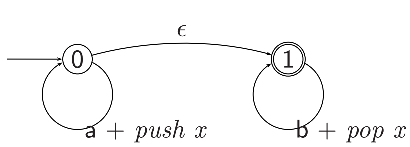
$N = \{\mathsf{S}\}$

$\Sigma = \{\mathsf{a}, \mathsf{b}\}$

$S = \mathsf{S}$

$$P = \left\{ \begin{array}{lll} \mathsf{S} & \rightarrow & \mathsf{a\ S\ b} \\ \mathsf{S} & \rightarrow & \epsilon \end{array} \right\}$$

Push-down automaton:



Type 1: Context-Sensitive Grammars and Linear-Bounded Automata

A rule of a **context-sensitive grammar**

- rewrites at most one non-terminal from the left-hand side.
- Contextual restrictions on the occurrence of this non-terminal may be imposed.
- The non-terminal must not rewrite as the empty string ϵ.

A **linear-bounded automaton** is a

- finite state automaton, with an
- auxiliary memory which cannot exceed the length of the input string.

A context-sensitive language example: $a^n b^n c^n$

Context-sensitive grammar:

$N = \{\text{S, B, C}\}$

$\Sigma = \{\text{a, b}\}$

$S = \text{S}$

$$P = \begin{Bmatrix} \text{S} \rightarrow \text{a S B C,} \\ \text{S} \rightarrow \text{a b C,} \\ \text{b B} \rightarrow \text{b b,} \\ \text{b C} \rightarrow \text{b c,} \\ \text{c C} \rightarrow \text{c c,} \\ \text{C B} \rightarrow \text{B C} \end{Bmatrix}$$

Type 0: General Rewrite Grammar and Turing Machines

- In a **general rewrite grammar** there are no restrictions on the form of a rewrite rule.
- A **turing machine** has an unbounded auxiliary memory.
- Any language for which there is a recognition procedure can be defined, but recognition problem is not decidable.

Properties of different language classes

Reasoning:

- Languages are defined to be sets of strings.
- One can therefore apply set operations to languages and investigate results for particular language classes.

Some closure properties:

- All language classes are closed under **union with themselves**.
- All language classes are closed under **intersection with regular languages**.
- The class of **context-free languages is not closed under intersection with itself**. Proof:

  Assume the two context-free languages L_1 and L_2:

  - $L_1 = \left\{a^n b^n c^i | n \geq 1 \text{ and } i \geq 0\right\}$
  - $L_2 = \left\{a^j b^n c^n | n \geq 1 \text{ and } j \geq 0\right\}$

  Their intersection is not context-free:

  - $L_1 \cap L_2 = \{a^n b^n c^n | n \geq 1\}$

Criteria under which to evaluate grammar formalisms

There are three kinds of criteria:

- linguistic naturalness
- mathematical power
- computational effectiveness and efficiency

The weaker the type of grammar:

- the stronger the claim made about possible languages
- the greater the efficiency of the parsing procedure

Reasons for choosing a stronger grammar class:

- to capture the bare facts about actual languages
- to provide for elegant analyses capturing more generalizations (→ more “compact” grammars)

Language classes and natural languages
Natural languages are not regular

Example grammar:

$$\left\{\begin{array}{l} \text{S} \rightarrow \text{If S then S} \\ \text{S} \rightarrow \text{Either S or S} \\ \text{S} \rightarrow \text{I laugh}|\text{I have to sneeze}|\text{Tim screams} \end{array}\right\}$$

Example analyses:

- [If [I laugh] then [I have to sneeze]]
- [If [either [I laugh] or [I have to sneeze]] then [Tim screams]]
- . . .

A more abstract version of the grammar rules:

$$\left\{\begin{array}{l} \text{S} \rightarrow \text{aSaS} \\ \text{S} \rightarrow \text{bSbS} \\ \text{S} \rightarrow \epsilon \end{array}\right\}$$

which accepts $a^n b^m b^m a^n$ which is not a regular language.

Accounting for bare facts vs. linguistically sensible analyses

Looking at grammars from a linguistic perspective, one can distinguish their

- **weak generative capacity**, considering only the set of strings generated by a grammar
- **strong generative capacity**, considering the set of strings and their syntactic analyses generated by a grammar

Two grammars can be strongly or weakly equivalent.

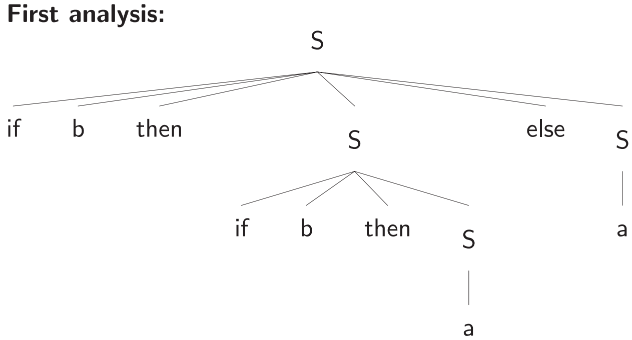
Example for weakly equivalent grammars

Example string:
if b then if b then a else a p

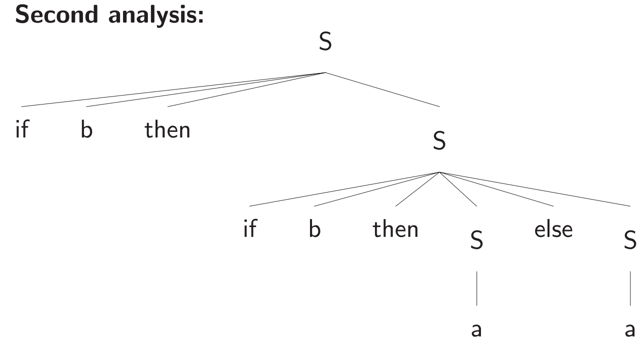
Grammar 1 rules:

$$\left\{\begin{array}{l} \text{S} \rightarrow \text{if b then S else S,} \\ \text{S} \rightarrow \text{if b then S,} \\ \text{S} \rightarrow \text{a} \end{array}\right\}$$

First analysis:



Second analysis:



Grammar 2 rules: A weekly equivalent grammar eliminating the ambiguity (only licenses second analysis).

$$\left\{\begin{array}{l} \text{S1} \rightarrow \text{if b then S1,} \\ \text{S1} \rightarrow \text{if b then S2 else S1,} \\ \text{S1} \rightarrow \text{a,} \\ \text{S2} \rightarrow \text{if b then S2 else S2,} \\ \text{S2} \rightarrow \text{a} \end{array}\right\}$$