

# Towards more complex grammar systems

## Some basic formal language theory

L545

Spring 2013

(With thanks to Detmar Meurers)

# Overview

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

Automata

Complexity

Type 3

Type 2

Type 1

Type 0

Properties

A grammar is a 4-tuple  $(N, \Sigma, S, P)$  where

- ▶  $N$  is a finite set of **non-terminals**
- ▶  $\Sigma$  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  $\alpha$  including at least one non-terminal symbol.

## Grammars

### Automata

### Complexity

Type 3

Type 2

Type 1

Type 0

### Properties

# A simple example

Towards more  
complex grammar  
systems  
Some basic formal  
language theory

$$N = \{S, NP, VP, V_i, V_t, V_s\}$$
$$\Sigma = \{\text{John, Mary, laughs, loves, thinks}\}$$
$$S = S$$
$$P = \left\{ \begin{array}{ll} S \rightarrow NP VP & NP \rightarrow \text{John} \\ & NP \rightarrow \text{Mary} \\ VP \rightarrow V_i & V_i \rightarrow \text{laughs} \\ VP \rightarrow V_t NP & V_t \rightarrow \text{loves} \\ VP \rightarrow V_s S & V_s \rightarrow \text{thinks} \end{array} \right\}$$

Grammars

Automata

Complexity

Type 3

Type 2

Type 1

Type 0

Properties

# How does a grammar define a language?

Assume  $\alpha, \beta \in (N \cup \Sigma)^*$ , with  $\alpha$  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.
- ▶  $\alpha$  (directly or immediately) **derives**  $\beta$  if  $\alpha \rightarrow \beta \in P$ .
  - ▶  $\alpha \Rightarrow^* \beta$  if  $\beta$  is derived from  $\alpha$  in zero or more steps
  - ▶  $\alpha \Rightarrow^+ \beta$  if  $\beta$  is derived from  $\alpha$  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.,  $L(G) = \{\gamma | S \Rightarrow^* \gamma\}$

## Grammars

### Automata

### Complexity

Type 3

Type 2

Type 1

Type 0

### Properties

# Processing with grammars: automata

Towards more  
complex grammar  
systems  
Some basic formal  
language theory

Grammars

Automata

Complexity

Type 3

Type 2

Type 1

Type 0

Properties

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 & automata

Towards more complex grammar systems  
Some basic formal language theory

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

Grammars

Automata

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

Complexity

Type 3

Type 2

Type 1

Type 0

Properties

Abbreviations:

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

## Type 3: Right-Linear Grammars and FSAs

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

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

A **finite-state automaton** consists of

- ▶ a tape
- ▶ a finite-state control
- ▶ no auxiliary memory



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

Towards more complex grammar systems  
Some basic formal language theory

## Right-linear grammar:

$$\begin{array}{l} N = \{\text{Expr}, X, Y, Z\} \\ \Sigma = \{a, b, c\} \\ S = \text{Expr} \end{array} \quad P = \left\{ \begin{array}{ll} \text{Expr} & \rightarrow ab X \\ \text{Expr} & \rightarrow c X \\ Y & \rightarrow b Y \\ Y & \rightarrow Z \end{array} \quad \left. \begin{array}{ll} X & \rightarrow a Y \\ Z & \rightarrow a \\ Z & \rightarrow cb \\ Z & \rightarrow \epsilon \end{array} \right\}$$

Grammars

Automata

Complexity

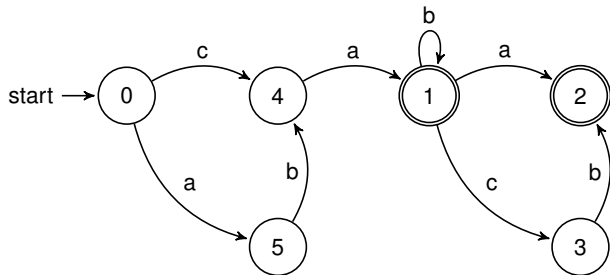
Type 3

Type 2

Type 1

Type 0

## Finite-state transition network:



Properties

# Thinking about regular languages

Towards more complex grammar systems  
Some basic formal language theory

- ▶ A language is regular iff one can define a FSM (or regular expression) for it.
  - ▶ Note the rough correspondence between state 0 & Expr, state 4 & X, and state 1 & Y
  - ▶ Think about why we need the rule  $Y \rightarrow Z$  (Could we write an FSM to more directly match the rules?)
- ▶ An FSM only has a fixed amount of memory, namely the number of states.
- ▶ Strings longer than the number of states (in particular, 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 (e.g.,  $a^n b^n$  is not regular).

Grammars

Automata

Complexity

Type 3

Type 2

Type 1

Type 0

Properties

# Pumping Lemma

**Pumping Lemma:** Let  $L$  be an infinite regular language. Then there are strings  $x$ ,  $y$ , and  $z$ , s.t.  $y \neq \epsilon$  and  $xy^n z \in L$  for  $n \geq 0$ .

- ▶ If  $L$  is regular, then  $y$  can be “pumped”
- ▶ Used to show that a particular language isn't regular if no string can be pumped that way

**Example:** Trying to map  $a^n b^n$  to  $xy^n z$  leads to a contradiction

1.  $y$  is composed of all  $a$ 's  $\rightarrow$  more  $a$ 's than  $b$ 's
2.  $y$  is composed of all  $b$ 's  $\rightarrow$  more  $b$ 's than  $a$ 's
3.  $y$  is composed of  $a$ 's &  $b$ 's  $\rightarrow$  some  $b$ 's precede some  $a$ 's

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

Towards more complex grammar systems  
Some basic formal language theory

A **context-free grammar** is a 4-tuple  $(N, \Sigma, S, P)$  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

Grammars

Automata

Complexity

Type 3

Type 2

Type 1

Type 0

Properties

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

## Context-free grammar:

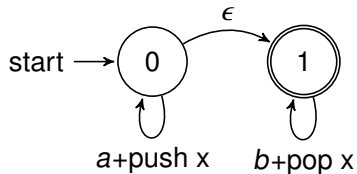
$$N = \{S\}$$

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

$$S = S$$

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

## Push-down automaton:



Grammars

Automata

Complexity

Type 3

Type 2

Type 1

Type 0

Properties

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

Towards more complex grammar systems  
Some basic formal language theory

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

- ▶ rewrites at most one non-terminal from the left-hand side.
- ▶ right-hand side of a rule required to be at least as long as the left-hand side, i.e. only contains rules of the form

$$\alpha \rightarrow \beta \text{ with } |\alpha| \leq |\beta|$$

and optionally  $S \rightarrow \epsilon$  with the start symbol  $S$  not occurring in any  $\beta$ .

## A **linear-bounded automaton** is a

- ▶ finite state automaton, with an
- ▶ auxiliary memory which cannot exceed the length of the input string (but is not as restrictive as a stack).

Grammars

Automata

Complexity

Type 3

Type 2

Type 1

Type 0

Properties

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

Towards more complex grammar systems  
Some basic formal language theory

## Context-sensitive grammar:

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

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

$$S = S$$

$$P = \left\{ \begin{array}{l} S \rightarrow a S B C, \\ S \rightarrow a b C, \\ b B \rightarrow b b, \\ b C \rightarrow b c, \\ c C \rightarrow c c, \\ C B \rightarrow B C \end{array} \right\}$$

Grammars

Automata

Complexity

Type 3

Type 2

Type 1

Type 0

Properties

# Type 0: General Rewrite Grammar & Turing Machines

Towards more complex grammar systems  
Some basic formal language theory

Grammars

Automata

Complexity

Type 3

Type 2

Type 1

Type 0

Properties

- ▶ 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

Languages are sets of strings, so that one can apply set operations to languages and investigate the 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: The intersection of the two context-free languages  $L_1$  and  $L_2$  is not context free:

- ▶  $L_1 = \{a^n b^n c^i \mid n \geq 1 \text{ and } i \geq 0\}$
- ▶  $L_2 = \{a^j b^n c^n \mid n \geq 1 \text{ and } j \geq 0\}$
- ▶  $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 1\}$

# Criteria under which to evaluate grammar formalisms

Towards more complex grammar systems  
Some basic formal language theory

There are three kinds of criteria:

- ▶ linguistic naturalness
- ▶ mathematical power
- ▶ computational effectiveness and efficiency

Grammars

Automata

Complexity

Type 3

Type 2

Type 1

Type 0

The weaker the type of grammar:

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

Properties

Reasons for choosing a stronger grammar class:

- ▶ to capture the empirical reality of actual languages
- ▶ to provide for elegant analyses capturing more generalizations (→ more “compact” grammars)

# Accounting for the facts vs. linguistically sensible analyses

Towards more  
complex grammar  
systems  
Some basic formal  
language theory

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.

Grammars

Automata

Complexity

Type 3

Type 2

Type 1

Type 0

Properties

# Example for weakly equivalent grammars

Towards more  
complex grammar  
systems  
Some basic formal  
language theory

## Example string:

if x then if y then a else b

## Grammar 1:

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

Grammars

Automata

Complexity

Type 3

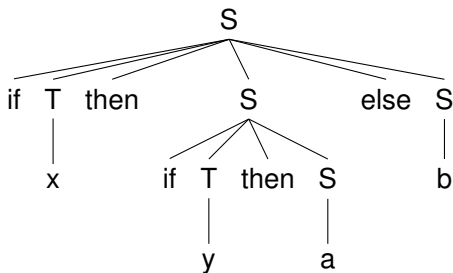
Type 2

Type 1

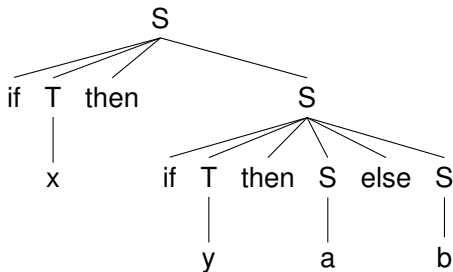
Type 0

Properties

## First analysis:



## Second analysis:



Towards more complex grammar systems  
Some basic formal language theory

Grammars

Automata

Complexity

Type 3

Type 2

Type 1

Type 0

Properties

**Grammar 2 rules:** A weakly equivalent grammar eliminating the ambiguity (only licenses second structure).

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

Grammars

Automata

Complexity

Type 3

Type 2

Type 1

Type 0

Properties