Towards more complex grammar systems
Some basic formal language theory

L445 / L545

Spring 2017

(With thanks to Detmar Meurers)

Towards more complex grammar systems
Some basic formal language theory
Grammars
Automata
Complexity
Type 3
Type 2
Type 1
Type 0
Properties

Overview

- 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

Towards more complex grammar systems
Some basic formal language theory
Grammars
Automata
Complexity
Type 3
Type 2
Type 1
Type 0
Properties

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 $\alpha \to \beta$, with $\alpha, \beta \in (N \cup \Sigma)*$ and α including at least one non-terminal symbol.

Towards more complex grammar systems
Some basic formal language theory
Grammars
Automata
Complexity
Type 3
Type 2
Type 1
Type 0
Properties

A simple example

N = {S, NP, VP, V_i, V_t, V_s}
Σ = {John, Mary, laughs, loves, thinks}
S = S

$$P = \left\{ \begin{array}{lllllll} S & \to & NP\ VP & & NP & \to & John \\ & & & & NP & \to & Mary \\ VP & \to & V_i & & & & \\ VP & \to & V_t\ NP & & V_i & \to & laughs \\ VP & \to & V_s\ S & & V_t & \to & loves \\ & & & & V_s & \to & thinks \end{array} \right\}$$

Towards more complex grammar systems
Some basic formal language theory
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 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 $\beta \to \delta$, then $\alpha\delta\gamma$ is a sentential form.
- α (directly or immediately) **derives** β if $\alpha \to \beta \in P$.
  - $\alpha \Rightarrow^* \beta$ 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., $L(G) = \{\gamma | S \Rightarrow^* \gamma\}$

Towards more complex grammar systems
Some basic formal language theory
Grammars
Automata
Complexity
Type 3
Type 2
Type 1
Type 0
Properties

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 $\to$ symbols
  - store: memory configuration $\times$ symbol $\to$ memory configuration
- and a **finite-state control** relating the two components.

Different levels of complexity in grammars & automata

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

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 FSAs

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.

A **finite-state automaton** consists of

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

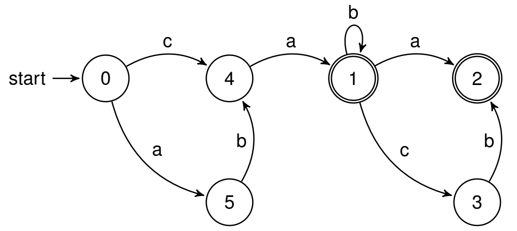
A regular language example: $(ab|c)ab*(a|cb)?$

Right-linear grammar:

N = {Expr, X, Y, Z}
Σ = {a,b,c}
S = Expr

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

Finite-state transition network:



Thinking about regular languages

- 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 → 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).

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 → more a's than b's
2. y is composed of all b's → more b's than a's
3. y is composed of a's & b's → some b's precede some a's

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

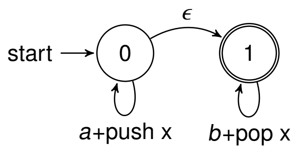
A context-free language example: $a^n b^n$

Context-free grammar:

N = {S}
Σ = {a, b}
S = S

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

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 ($\beta\ A\ \gamma \rightarrow \beta\ \delta\ \gamma$).
- 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 β.

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).

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

Context-sensitive grammar:

N = {S, B, C}
Σ = {a, b}
S = S

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

Weakly equivalent way to derive C B → B C:
https://en.wikipedia.org/wiki/Context-sensitive_grammar

Type 0: General Rewrite Grammar & 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

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

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 potential efficiency of the parsing procedure

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

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.

Towards more complex grammar systems
Some basic formal language theory
Grammars
Automata
Complexity
Type 3
Type 2
Type 1
Type 0
Properties

Example for weakly equivalent grammars

Example string:

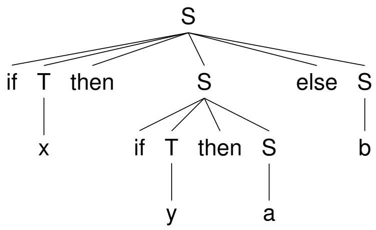
if x then if y then a else b

Grammar 1:

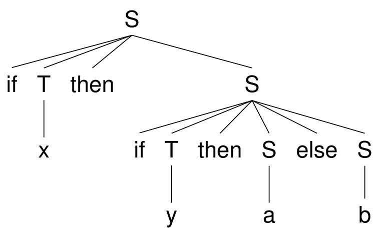
$$\left\{\begin{array}{l} S \rightarrow \text{if T then S else S,} \\ S \rightarrow \text{if T then S,} \\ S \rightarrow a \\ S \rightarrow b \\ T \rightarrow x \\ T \rightarrow y \end{array}\right\}$$

Towards more complex grammar systems
Some basic formal language theory
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).

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