Towards more complex grammar systems
Some basic formal language theory

L445 / L545
Spring 2017
(With thanks to Detmar Meurers)

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 ...

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 \( \rightarrow \) symbol
  - store: memory configuration \( \times \) symbol \( \rightarrow \) memory configuration
- and a finite-state control relating the two components.

Towards more complex grammar systems
Some basic formal language theory
Grammars
Automata
Complexity
Type 3
Type ...

A simple example

\[
N = \{S, NP, VP, V_i, V_s\} \\
\Sigma = \{\text{John}, \text{Mary}, \text{laughs}, \text{loves}, \text{thinks}\} \\
S = S \\
P = \begin{cases} 
S &\rightarrow NP \ VP \\
V_P &\rightarrow V_i \\
V_P &\rightarrow V_i NP \\
V_P &\rightarrow V_s S \\
NP &\rightarrow \text{John} \\
V_i &\rightarrow \text{laughs} \\
V_i &\rightarrow \text{loves} \\
V_s &\rightarrow \text{thinks} 
\end{cases}
\]

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 \( S \) is a sentential form.
  - If \( \alpha \beta \delta \) is a sentential form and there is a rewrite rule \( \beta \rightarrow \delta \), then \( \alpha \beta \delta \) is a sentential form.
  - \( \alpha \) (directly or indirectly) 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., \( L(G) = \{ \gamma | S \Rightarrow^* \gamma \} \)

Towards more complex grammar systems
Some basic formal language theory
Grammars
Automata
Complexity
Type 3
Type ...

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 \( \rightarrow \) symbol
  - store: memory configuration \( \times \) symbol \( \rightarrow \) memory configuration
- and a finite-state control relating the two components.
Pumping Lemma:

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

- $\text{Type 1: Unbounded memory}$
  - $\text{Memory: TM}$
  - $\text{Name: } \alpha \rightarrow \beta$
  - $\text{Rule Name: General rewrite}$

- $\text{Type 2: Bounded memory}$
  - $\text{Memory: LBA}$
  - $\text{Name: } \beta A \gamma \rightarrow \beta \delta \gamma$
  - $\text{Rule Name: Context-sensitive}$

- $\text{Type 3: None}$
  - $\text{Memory: FSA}$
  - $\text{Name: } A \rightarrow xB, A \rightarrow x$
  - $\text{Rule Name: Right linear}$

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

A regular language example: $(ab)cab \ast (a|cb)$?

Right-linear grammar:

$$
N = \{\text{Expr, X, Y, Z}\} \\
\Sigma = \{a,b,c\} \\
S = \text{Expr} \\
P = \\
\begin{align*}
\text{Expr} & \rightarrow \text{ab X} & X & \rightarrow a Y \\
\text{Expr} & \rightarrow c \text{ X} & Z & \rightarrow a \\
Y & \rightarrow b \text{ Y} & Z & \rightarrow cb \\
Y & \rightarrow Z & Z & \rightarrow \epsilon \\
\end{align*}
$$

Finite-state transition network:

```
0 -- a → 1
      b → 4
   1 -- c → 2
   |
   v 5
2 -- a → 3
     b → 0
   3 -- c → 1
```

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 $\beta \in (\gamma \delta \gamma') \subseteq (\Sigma \cup N)^*$, 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

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 states 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).

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

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
All language classes are closed under grammar mathematical power the stronger the claim made about possible languages linguistic naturalness computational effectiveness and efficiency the greater the potential efficiency of the parsing procedure 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.

Example for weakly equivalent grammars

**Example string:**

if x then y then a else b

**Grammar 1:**

\[
\begin{align*}
S & \rightarrow \text{if } T \text{ then } S \text{ else } S \\
S & \rightarrow \text{if } T \text{ then } S \\
S & \rightarrow a \\
S & \rightarrow b \\
T & \rightarrow x \\
T & \rightarrow y
\end{align*}
\]

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

\[
\begin{align*}
S1 & \rightarrow \text{if } T \text{ then } S1 \text{ else } S1 \\
S1 & \rightarrow a \\
S1 & \rightarrow b \\
S2 & \rightarrow \text{if } T \text{ then } S2 \text{ else } S2 \\
S2 & \rightarrow a \\
S2 & \rightarrow b \\
T & \rightarrow x \\
T & \rightarrow y
\end{align*}
\]