Towards more complex grammar systems
Some basic formal language theory

Grammars

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

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

A simple example

\(N = \{S, NP, VP, V_1, V_2\}\)
\(\Sigma = \{John, Mary, laughs, loves, thinks\}\)
\(S = S\)
\(P = \{\)
\(S \to NP \quad VP \quad V_1\)
\(NP \to John \quad VP \to V_1 NP \quad V_1 \to loves\)
\(VP \to V_2 S \quad V_2 \to thinks\)\}

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Memory</th>
<th>Name</th>
<th>Rule</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unbounded</td>
<td>TM</td>
<td>$\alpha \rightarrow \beta$</td>
<td>General rewrite</td>
</tr>
<tr>
<td>1</td>
<td>Bounded</td>
<td>LBA</td>
<td>$\beta \ A \gamma \rightarrow \beta \ \delta \ \gamma$</td>
<td>Context-sensitive</td>
</tr>
<tr>
<td>2</td>
<td>Stack</td>
<td>PDA</td>
<td>$A \rightarrow \beta$</td>
<td>Context-free</td>
</tr>
<tr>
<td>3</td>
<td>None</td>
<td>FSA</td>
<td>$A \rightarrow xB, A \rightarrow x$</td>
<td>Right linear</td>
</tr>
</tbody>
</table>

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

A regular language example: $(ab)cab * (ac|cb)$?

**Right-linear grammar:**

$$N = \{\text{Expr, X, Y, Z}\}, \quad \Sigma = \{a,b,c\}, \quad S = \text{Expr}$$

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

Finite-state transition network:

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

**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 $\beta \in (\{\gamma | \gamma \in \Sigma^* \delta \in \mathbb{N} \cup \{\epsilon\}\})$, 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

**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 $\beta \in (\Sigma \cup \mathbb{N})^*$, i.e.:

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

A push-down automaton is a
- finite state automaton, with a
- stack as auxiliary memory

**Pumping Lemma**

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

**Thinking about regular languages**

- A language is regular if 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 $Y \rightarrow Z$, i.e.: $\gamma \delta \gamma$ (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).
A context-free language example: $a^n b^n$

**Context-free grammar:**

\[
N = \{S\} \\
\Sigma = \{a, b\} \\
S = S \\
P = \{ \\
S \rightarrow a S b \\
S \rightarrow \epsilon \\
\} \\
\]

**Push-down automaton:**

```
Start 0
epsilon
a+push x
b+pop x
1
3
```

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

**Context-sensitive grammar:**

\[
N = \{S, B, C\} \\
\Sigma = \{a, b\} \\
S = S \\
P = \{ \\
S \rightarrow a S B C, \\
b B \rightarrow b b, \\
b C \rightarrow b c, \\
c C \rightarrow c c, \\
C B \rightarrow B C \\
\} \\
\]

**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:

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

**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)

**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.
- 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 $\alpha \rightarrow \beta$ 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).
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 if y then a else b
```

**Grammar 1:**

```
S  →  if T then S else S,
S  →  if T then S,
S  →  a
S  →  b
T  →  x
T  →  y
```

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

```
S1 → if T then S1,
S1 → if T then S2 else S1,
S1 → a,
S1 → b,
S2 → if T then S2 else S2,
S2 → a
S2 → b
T  →  x
T  →  y
```