Final lecture:
Applications, Chomsky hierarchy, and Recap

H. Geuvers and J. Rot

Institute for Computing and Information Sciences
Radboud University Nijmegen

Version: 2016-17
Outline

Applications of CFGs

Beyond CFGs

Recap
Most programming languages are (deterministic) context-free.

There are tools to automatically build:

- a lexical analyzer (‘lexer’) from regular expressions.

  “if x = 2 then P else Q”

\[ \leadsto \text{if} \quad x \quad \leftarrow \quad 2 \quad \text{then} \quad P \quad \text{else} \quad Q \]

- a parser from a CFG.

\[ \leadsto \text{ifthenelse} \quad \begin{array}{c}
\leftarrow \\
P \\
Q \\
x \\
2
\end{array} \]
Lindenmayer systems

“The development of an organism...may be considered as the execution of a ‘developmental program’ present in the fertilized egg... A central task of developmental biology is to discover the underlying algorithm from the course of development.”

– Lindenmayer & Rozenberg (1976)

Example:

\[
\begin{align*}
A & \rightarrow AB \\
B & \rightarrow A
\end{align*}
\]

Start with \(A\), expand once per iteration:

\[
\begin{align*}
0 & \quad A \\
1 & \quad AB \\
2 & \quad ABA \\
3 & \quad ABAAB \\
4 & \quad ABAABABA \\
& \ldots
\end{align*}
\]
Lindenmayer systems

Drawing a Lindenmayer system:
- **F**: move forward
- **+**: rotate counter clockwise
- **−**: rotate clockwise
- **[:** push location/angle
- **]:** pop location/angle

Example: $F \rightarrow F + F -- F + F$

<table>
<thead>
<tr>
<th>Step</th>
<th>Symbol Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F+F--F+F</td>
</tr>
</tbody>
</table>
| 2    | F+F--F+F+F+F--F+F--F+F--F+F+F+F--F+F--F+F--F+F+F+F--F+F--F+F--F+F
| 3    | ...            |
Lindenmayer systems

\[ S \rightarrow F [+S][−S] \]
\[ F \rightarrow FF \]

\[ S \rightarrow F − [S] + S + F [+FS] − S \]
Lindenmayer systems (cont’d)

Example: Penrose Tiling (P3)

\[
\begin{align*}
S & \rightarrow [N] + +[N] + +[N] + +[N] + +[N] \\
M & \rightarrow OF + +PF \quad \quad \quad \quad \quad NF[−OF \quad − \quad − \quad MF \quad + \quad +] \\
N & \rightarrow +OF \quad − \quad PF[− \quad − \quad MF \quad − \quad NF] + \\
O & \rightarrow −MF \quad + \quad NF[+ \quad + \quad OF \quad + \quad PF] − \\
P & \rightarrow −−OF \quad + \quad + \quad + \quad MF[+PF \quad + \quad + \quad +NF] − \quad −NF
\end{align*}
\]
Jill frequently eats a juicy yellow mango. belongs to this language
Natural language

- Many sentences can be modelled using CFGs, e.g.:
  
  ...because I saw Cecilia feed the hippopotamuses.

- But some (particularly crazy 😊) natural languages have non-context-free features like *cross-serial dependencies*:
  
  ...omdat ik Cecilia de nijlpaarden zag voeren.

- To capture these, one needs more power than CFGs
Whereas context-free grammars have rules like this:

\[ X \to w \quad X \in V, \ w \in (\Sigma \cup V)^* \]

...a *context-sensitive grammar* has rules like this:

\[ \alpha X \beta \to \alpha w \beta \]

with \( X \in V, \ \alpha, \beta, \ w \in (\Sigma \cup V)^*, \ w \neq \lambda \).

Context-sensitive grammars generate *context-sensitive languages.*
Example: \( \{ a^n b^n c^n \mid n > 0 \} \)

\[
\begin{align*}
S & \rightarrow aBC \mid aSBC \\
CB & \rightarrow XB \\
XB & \rightarrow XC \\
XC & \rightarrow BC \\
aB & \rightarrow ab \\
bB & \rightarrow bb \\
bC & \rightarrow bc \\
cC & \rightarrow cc
\end{align*}
\]
Turing machines

- An *unrestricted grammar* has rules like:

\[ u \rightarrow v \quad u, v \in (\Sigma \cup \Gamma)^* \]

- Recognisable by **Turing machines**
- The stack is replaced by an infinite *tape*:

\[ \cdots \quad \cdots \]

- Transitions look like this:

\[ p \quad \xrightarrow{a/b, \leftarrow} \quad q \quad \text{and} \quad p \quad \xrightarrow{a/b, \rightarrow} \quad q \]

which *read* \( a \), *write* \( b \), and move left or right on the tape

- We no longer need a separate input. Just use the tape:

\[ \cdots \quad a \quad b \quad b \quad \cdots \]
Enumerable and computable languages

- Languages recognisable by Turing machine are called enumerable languages.
- A language is computable if both $L$ and $\overline{L} = \Sigma^* - L$ are enumerable.
- In other words, there is a Turing machine with terminates telling us $w \in L$ and a (possibly different) one that terminates telling us $w \notin L$.
- Example: $\{a^n \mid n \text{ is not prime}\}$.
- Church-Turing thesis:
  
  "computable" $\iff$ "computable by Turing machine"

- (To be continued: Berekenbaarheid, 2nd year)
Chomsky hierarchy

\{ w \mid w \text{ describes a terminating Turing machine} \}, \ldots

\{ a^n \mid n \text{ is prime} \}, \ldots

\{ a^n b^n c^n \mid n > 0 \}, \ldots

\{ a^n b^n \mid n > 0 \}, \ldots

\{ a^n \mid n > 0 \}, \ldots

Regular

Context-free

Context-sensitive

Computable

Enumerable
Bigger classes of languages:

- More languages can be described.
- But, you can say less about them.

<table>
<thead>
<tr>
<th></th>
<th>( w \in L? )</th>
<th>time</th>
<th>memory</th>
<th>( L_1 = L_2? )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>yes</td>
<td>(</td>
<td>w</td>
<td>)</td>
</tr>
<tr>
<td>Deterministic context-free</td>
<td>yes</td>
<td>(</td>
<td>w</td>
<td>)</td>
</tr>
<tr>
<td>Context-free</td>
<td>yes</td>
<td>(</td>
<td>w</td>
<td>^3 )</td>
</tr>
<tr>
<td>Context-sensitive</td>
<td>yes</td>
<td>( 2</td>
<td>w</td>
<td>)</td>
</tr>
<tr>
<td>Computable</td>
<td>yes</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>no</td>
</tr>
<tr>
<td>Enumerable</td>
<td>if ( w \in L )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>no</td>
</tr>
</tbody>
</table>
Exam topic: regular expressions

You should know:

- the definition of regular expression
- how to compute a regular expression from an $\text{NFA}_\lambda$ using the elimination-of-states method
- how to build an $\text{NFA}_\lambda$ from a regular expression using the ‘toolkit’
Exam topic: finite automata

You should know:

- the definition of DFA, NFA, NFA_λ
- how to construct a DFA, NFA or NFA_λ for a given language
- how to construct a DFA from an NFA (the subset construction)
- the constructions on DFAs for complement and intersection (product construction) of languages
Exam Topic: regular languages

You should know:

- the closure properties of regular languages (union, intersection, complement), and how to use them to prove that a language is regular (or is not regular)
- typical non-regular languages ($\{a^n b^n \mid n \in \mathbb{N}\}$ and palindromes)
- Kleene’s theorem
- how to apply the pumping lemma to show that a language is not regular
Exam topic: grammars

You should know:

- the definition of context-free grammar (CFG)
- **how to generate strings in a CFG (with leftmost derivations)**
- how to find the language generated by a (simple) CFG
- **how to construct a CFG that generates a given (context-free) language**
- the definition of regular grammar
- how to construct a regular grammar from a NFA
- how to construct an NFA_λ from a regular grammar
Exam topic: pushdown automata and CFLs

You should know:

- the definition of pushdown automata (PDAs)
- how to give a PDA for a given (simple context-free) language
- how find the language accepted by a given PDA
- how to construct a PDA from a CFG
- how to construct a CFG from a PDA
- the closure properties of context-free languages
Remarks on sets

Beware that \( \emptyset \neq \{ \emptyset \} \), \( \emptyset \neq \lambda \), \( \emptyset \neq \{ \lambda \} \).

and \( \emptyset \cdot L = L \cdot \emptyset = \emptyset \)

and \( \{ \lambda \} \cdot L = L \cdot \{ \lambda \} = L \).

Symbols:

- \( w \in L \): “is in”.
- \( L_1 \subseteq L_2 \): is a subset of, i.e. everything in \( L_1 \) is also in \( L_2 \).
- \( L_1 \cup L_2 \): union, the things in either of the two sets.
- \( L_1 \cap L_2 \): intersection, only the things in both sets.
- \( \overline{L} \): complement, the words not in \( L \), \( \overline{L} = \Sigma^* - L \).

Terminology: Language described using set-notation, examples:

\[
\{ w \in \{ a, b \}^* \mid \text{\(|w|_a \) is even} \}, \quad \{ a^n b^m \mid n < m \}, \quad \emptyset
\]
Remarks on words and languages

Languages

- contain words.
- can be infinite, but words are finite.

The language $L^*$

- always contains $\lambda$.
- is not the same as $\{w^n \mid w \in L, n \geq 0\}$.
- is $L^0 \cup L^1 \cup L^2 \cup \ldots$. 
Remarks on proofs

To prove that $L_1 = L_2$, show that

- $L_1 \subseteq L_2$ (for all $w$, $w \in L_1$ implies $w \in L_2$), and
- $L_1 \supseteq L_2$ (for all $w$, $w \in L_2$ implies $w \in L_1$).

Proof by contradiction

- If regular languages have property $X$, and $L_1$ does not have property $X$, then $L_1$ is not regular.
- If from $L_1$ being context-free you can deduce that $\{a^n b^n c^n \mid n \geq 0\}$ is context-free, then $L_1$ is not context-free.

Proof by induction: to prove $P(w)$ for all $w$,

- Show that $P(\lambda)$
- And that $P(w)$ implies $P(aw)$. 

Exam tips

• You may use results and examples we treated during the lectures and exercises. For example, you may use that the language \( \{ a^n b^n \mid n \geq 0 \} \) is not regular without re-proving it.

• Always give an explanation. For example, when asked to give a CFG for a language, explain why your CFG is correct.

• Check your results. For example,
  - check that a DFA that you give indeed is a DFA.
  - given NFA-\( \lambda \) that accepts \( w \), after \( \lambda \)-elimination and subset construction, check that resulting DFA accepts \( w \).

• Connect your knowledge, think further. An exam question may not directly tell you what you need to do. For example,
  - Q: Is \( L \) regular? (Which techniques can you apply?)
  - Q: Give a DFA for \( L \) (Is \( L \) of the form \( \overline{L}_1 \) or \( L_1 \cap L_2 \) ?)
Finally

- Feedback test1: today, ground floor Mercator, 15:30
- Vragenuurtje: Tuesday Jan 17, 13:45 – 14:30 (time and place to be confirmed)
- Last homework assignment: hand in on Monday Jan 16
- Other missing homework: hand in on Tuesday Jan 17
- Exam: Wednesday January 18, 8:30 – 11:30
- Veel succes!