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

Programming languages

Most programming languages are (deterministic) context-free.

There are tools to automatically build:

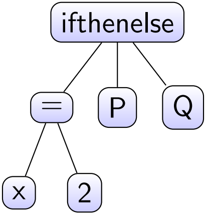
- a lexical analyzer ('lexer') from regular expressions.

  “`if x = 2 then P else Q`”

  $\leadsto$ $\boxed{\text{if}}$ $\boxed{\text{x}}$ $\boxed{=}$ $\boxed{2}$ $\boxed{\text{then}}$ $\boxed{\text{P}}$ $\boxed{\text{else}}$ $\boxed{\text{Q}}$

- a parser from a CFG.

  $\leadsto$



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:

$$\boxed{\begin{array}{lcl} A & \to & AB \\ B & \to & A \end{array}}$$

Start with A, expand once per iteration:

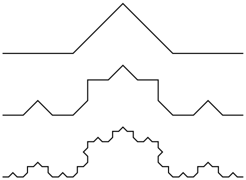
0	A
1	AB
2	ABA
3	$ABAAB$
4	$ABAABABA$
	$\ldots$

Lindenmayer systems

Drawing a Lindenmayer system:

- F: move forward
- $+$: rotate counter clockwise
- $-$: rotate clockwise
- [: push location/angle
-]: pop location/angle

Example: $\boxed{F \quad \rightarrow \quad F + F - - F + F}$



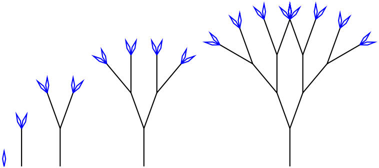
0 F

1 F+F--F+F

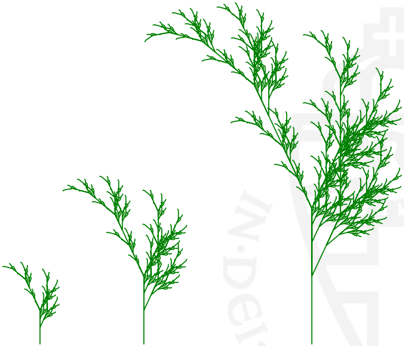
2 F+F--F+F+F+F--F+F--F+F--F+F+F+F--F+F

3 ...

Lindenmayer systems



$$S \quad \rightarrow \quad F[+S][-S]$$

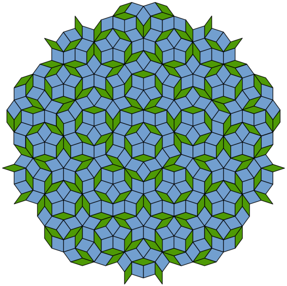


$$\begin{array}{lcl} S & \rightarrow & F-[[S]+S]+F[+FS]-S \\ F & \rightarrow & FF \end{array}$$

Lindenmayer systems (cont'd)

Example: Penrose Tiling (P3)

$$
\begin{array}{lcl}
S & \rightarrow & [N]++[N]++[N]++[N]++[N] \\
M & \rightarrow & OF++PF----NF[-OF----MF]++ \\
N & \rightarrow & +OF--PF[---MF--NF]+ \\
O & \rightarrow & -MF++NF[+++OF++PF]- \\
P & \rightarrow & --OF++++MF[+PF++++NF]--NF
\end{array}
$$



Natural language

$$
\begin{array}{rcl}
S = \langle\text{sentence}\rangle & \to & \langle\text{noun-phrase}\rangle\langle\text{verb-phrase}\rangle\texttt{.}\\
\langle\text{sentence}\rangle & \to & \langle\text{noun-phrase}\rangle\langle\text{verb-phrase}\rangle\langle\text{object-phrase}\rangle\texttt{.}\\
\langle\text{noun-phrase}\rangle & \to & \langle\text{name}\rangle\langle\text{article}\rangle\langle\text{noun}\rangle\\
\langle\text{name}\rangle & \to & \texttt{John} \mid \texttt{Jill}\\
\langle\text{noun}\rangle & \to & \texttt{bicycle} \mid \texttt{mango}\\
\langle\text{article}\rangle & \to & \texttt{a} \mid \texttt{the}\\
\langle\text{verb-phrase}\rangle & \to & \langle\text{verb}\rangle \mid \langle\text{adverb}\rangle\langle\text{verb}\rangle\\
\langle\text{verb}\rangle & \to & \texttt{eats} \mid \texttt{rides}\\
\langle\text{adverb}\rangle & \to & \texttt{slowly} \mid \texttt{frequently}\\
\langle\text{adjective-list}\rangle & \to & \langle\text{adjective}\rangle\langle\text{adjective-list}\rangle \mid \lambda\\
\langle\text{adjective}\rangle & \to & \texttt{big} \mid \texttt{juicy} \mid \texttt{yellow}\\
\langle\text{object-phrase}\rangle & \to & \langle\text{adjective-list}\rangle\langle\text{name}\rangle\\
\langle\text{object-phrase}\rangle & \to & \langle\text{article}\rangle\langle\text{adjective-list}\rangle\langle\text{noun}\rangle
\end{array}
$$

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

Context-sensitive languages

- Whereas context-free grammars have rules like this:

$$X \to w \qquad\qquad X \in V, w \in (\Sigma \cup V)^*$$

- ...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.

Context-sensitive languages

Example: $\{a^n b^n c^n \mid n > 0\}$

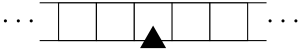
$$
\begin{array}{lcl}
S & \to & aBC \mid aSBC \\
CB & \to & XB \\
XB & \to & XC \\
XC & \to & BC \\
aB & \to & ab \\
bB & \to & bb \\
bC & \to & bc \\
cC & \to & cc
\end{array}
$$

Turing machines

- An *unrestricted grammar* has rules like:

$$u \to v \qquad\qquad u, v \in (\Sigma \cup V)^*$$

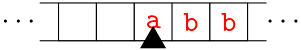
- Recognisable by Turing machines
- The stack is replaced by an infinite *tape*:



- Transitions look like this:

$$p \xrightarrow{\mathtt{a}/\mathtt{b},\leftarrow} q \qquad\qquad p \xrightarrow{\mathtt{a}/\mathtt{b},\rightarrow} q$$

  which *read* a, *write* b, and move left or right on the tape
- We no longer need a separate input. Just use the tape:



Enumerable and computable languages

- Languages recognisable by Turing machine are called enumerable languages
- A languages 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: $\{\mathtt{a}^n \mid n \text{ is not prime}\}$.
- Church-Turing thesis:

$$\text{“computable”} \iff \text{“computable by Turing machine”}$$

- (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

Trade-offs

Bigger classes of languages:

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

	$w \in L$?	time	memory	$L_1 = L_2$?
Regular	yes	$\lvert w \rvert$	const.	yes
Deterministic context-free	yes	$\lvert w \rvert$	$\lvert w \rvert$	no
Context-free	yes	$\lvert w \rvert^3$	$\lvert w \rvert^2$	no
Context-sensitive	yes	$2^{\lvert w \rvert}$	$\lvert w \rvert^k$	no
Computable	yes	∞	∞	no
Enumerable	if $w \in L$	∞	∞	no

Exam topic: regular expressions

You should know:

- the definition of regular expression
- how to compute a regular expression from an NFA_λ using the elimination-of-states method
- how to build an NFA_λ 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 $\emptyset \neq \{\emptyset\}$ $\quad \emptyset \neq \lambda$ $\quad \emptyset \neq \{\lambda\}$.

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:

$$\{w \in \{a,b\}^* \mid |w|_a \text{ is even}\}, \quad \{a^n b^m \mid n < m\}, \quad \emptyset$$

Remarks on words and languages

Languages

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

The language L^*

- always contains λ.
- is *not* the same as $\{w^n \mid w \in L, n \geq 0\}$.
- is $L^0 \cup L^1 \cup L^2 \cup \cdots$.

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-λ that accepts w, after λ-elimination and subset constructon, 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!