

Final lecture: Recap, L-Systems, Chomsky hierarchy



Outline

Recap

Lindenmayer systems

Beyond CFGs



Organisational business

- No more homework!
- Last exercise class (Friday): practice session
- Exam: Monday January 22, 8:30 – 11:30, location: check schedule
- Homework is obligatory for a grade! Missing homework, last chance: hand in on Monday 22; you will get 0 points for those, but at least be able to pass the course.
- The final grade is composed of
 - the grade of your half-way test, **h**,
 - the grade of your final test, **f**,
 - the average grade of your exercises, **a**,
- Your final grade is $\min(10, \frac{f+h}{2} + \frac{a}{10})$
 - Additional requirements (everything at least 5) in study guide won't be applied
 - The re-exam is a full 3hrs exam about the whole course. You keep the (average) grade of the exercises.



Exam topics

- **Lecture 4,5,6**
- Lecture 1,2,3: regular languages/automata etc, not the various constructions for turning one into the other (see next slides)
- Nothing about Lecture 7



Necessary background knowledge first part

You should know:

- the definition of regular expression and regular language
- how to specify a given regular language with a 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')

I won't ask for the stuff between parentheses, but you need to know the constructions exist.



Necessary background knowledge first part

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)
- Kleene's theorem

I won't ask for the stuff between parentheses, but you need to know the constructions exist.



Exam topic: regular languages

You should know:

- typical non-regular languages ($\{a^n b^n \mid n \in \mathbb{N}\}$ and palindromes)
- 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)
- 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
- when a grammar is ambiguous
- 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 to find the language accepted by a given PDA
- how to construct a PDA from a CFG
- how to construct a CFG from a PDA
- when a PDA/CFL is deterministic
- 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.
- \bar{L} : complement, the words not in L , $\bar{L} = \Sigma^* - L$.

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 $L^0 \cup L^1 \cup L^2 \cup \dots$.



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 $P(aw)$.



Remarks on the Pumping Lemma

Be careful what you should choose, and what is given by the pumping lemma!

The recipe:

1. Assume that L is regular.
2. By the Pumping Lemma, there is a 'pumping number' k .
you can not choose k
3. **Give a word w such that $|w| \geq k$.**
4. The Pumping Lemma gives u, v, z such that
 - a. $w = uvz$, $|uv| \leq k$ and $|v| \geq 1$, and
 - b. for all $n \in \mathbb{N}$: $uv^n z \in L$.you can not choose u, v, z
5. **Try to show $uv^n z \notin L$ for some n .**
6. Conclude that there is a contradiction: hence L can not be regular.



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 if we ask for it. 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 .
- 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 $L_1 \cap L_2$?)



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:

$$\begin{array}{l} A \rightarrow AB \\ B \rightarrow A \end{array}$$

Start with A , expand once per iteration:

0 A
1 AB
2 ABA
3 $ABAAB$
4 $ABAABABA$
...



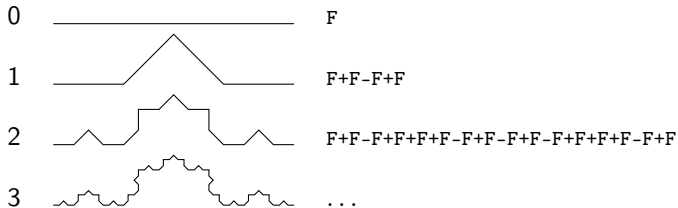
Lindenmayer systems

Drawing a Lindenmayer system:

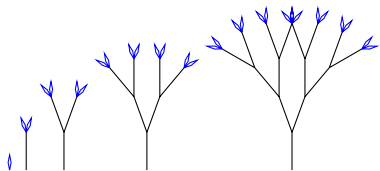
- F : move forward
- $+$: rotate counter clockwise
- $-$: rotate clockwise
- $[$: push location/angle
- $]$: pop location/angle

Example:

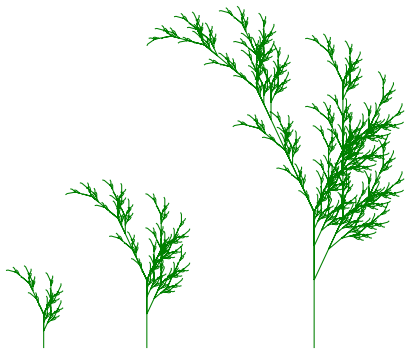
$F \rightarrow F + F - -F + F$



Lindenmayer systems



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

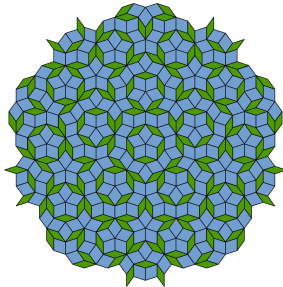


$$\begin{array}{l} S \rightarrow F - [[S] + S] + F[+FS] - S \\ F \rightarrow FF \end{array}$$

Lindenmayer systems (cont'd)

Example: Penrose Tiling (P3)

<i>S</i>	→	[<i>M</i>] ++[<i>M</i>] ++[<i>M</i>] ++[<i>N</i>] ++[<i>N</i>]
<i>M</i>	→	<i>OF</i> ++ <i>PF</i> - - - <i>NF</i> [- <i>OF</i> - - - <i>MF</i>] ++
<i>N</i>	→	+ <i>OF</i> - - <i>PF</i> [- - - <i>MF</i> - <i>NF</i>] +
<i>O</i>	→	- <i>MF</i> ++ <i>NF</i> [++ <i>OF</i> ++ <i>PF</i>]-
<i>P</i>	→	- - <i>OF</i> ++++ <i>MF</i> [+ <i>PF</i> ++++ <i>NF</i>]- - <i>NF</i>



Natural language

$S = \langle \text{sentence} \rangle$	\rightarrow	$\langle \text{noun-phrase} \rangle \langle \text{verb-phrase} \rangle .$
$\langle \text{sentence} \rangle$	\rightarrow	$\langle \text{noun-phrase} \rangle \langle \text{verb-phrase} \rangle \langle \text{object-phrase} \rangle .$
$\langle \text{noun-phrase} \rangle$	\rightarrow	$\langle \text{name} \rangle \langle \text{article} \rangle \langle \text{noun} \rangle$
$\langle \text{name} \rangle$	\rightarrow	<i>John Jill</i>
$\langle \text{noun} \rangle$	\rightarrow	<i>bicycle mango</i>
$\langle \text{article} \rangle$	\rightarrow	<i>a the</i>
$\langle \text{verb-phrase} \rangle$	\rightarrow	$\langle \text{verb} \rangle \langle \text{adverb} \rangle \langle \text{verb} \rangle$
$\langle \text{verb} \rangle$	\rightarrow	<i>eats rides</i>
$\langle \text{adverb} \rangle$	\rightarrow	<i>slowly frequently</i>
$\langle \text{adjective-list} \rangle$	\rightarrow	$\langle \text{adjective} \rangle \langle \text{adjective-list} \rangle \lambda$
$\langle \text{adjective} \rangle$	\rightarrow	<i>big juicy yellow</i>
$\langle \text{object-phrase} \rangle$	\rightarrow	$\langle \text{adjective-list} \rangle \langle \text{name} \rangle$
$\langle \text{object-phrase} \rangle$	\rightarrow	$\langle \text{article} \rangle \langle \text{adjective-list} \rangle \langle \text{noun} \rangle$


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 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:

$$X \rightarrow w \quad X \in V, w \in (\Sigma \cup V)^*$$

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

$$\alpha X \beta \rightarrow \alpha w \beta$$

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

$S \rightarrow aBC \mid aSBC$

$CB \rightarrow XB$

$XB \rightarrow XC$

$XC \rightarrow BC$

$aB \rightarrow ab$

$bB \rightarrow bb$

$bC \rightarrow bc$

$cC \rightarrow cc$



Turing machines

- An *unrestricted grammar* has rules like:

$$u \rightarrow v \quad u, v \in (\Sigma \cup V)^*$$

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



- Transitions look like this:



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

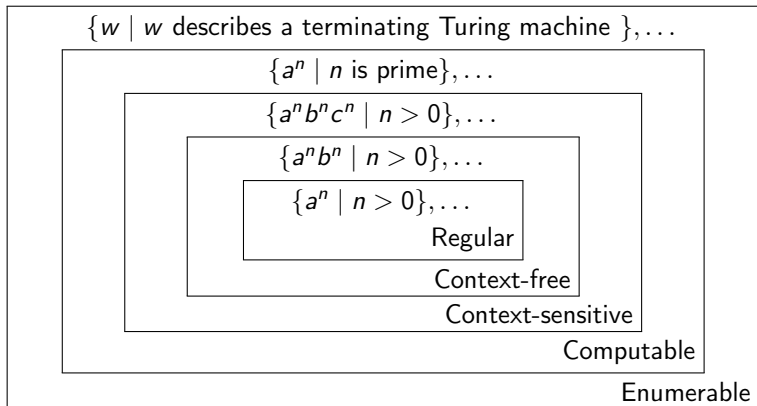
- Languages recognisable by Turing machine are called **enumerable languages**
- A language is **computable** if both L and $\bar{L} = \Sigma^* - L$ are enumerable.
- In other words, there is a Turing machine that terminates telling us $w \in L$ and a (possibly different) one that terminates telling 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



Trade-offs

Bigger classes of languages:

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

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



End

Thanks for your attention!

