Sapienza Università di Roma
Dipartimento di Informatica

FOUNDATIONS OF COMPUTER SCIENCE
LECTURE 8: Chomsky hierarchy

Prof. Daniele Gorla

Beyond context-free languages

We proved that $\{a^n b^n c^n \mid n \geq 0\}$ is not C.F.

→ how can we generate and/or recognize it?

Up-to now:

Regular languages	*DFA/NFA*	*Regular grammars*	*Regular expressions*
CF languages	*(non-det)PDA*	*CF grammars*	

In this class, we want to start completing this table for non-C.F. languages

SAPIENZA
Università di Roma
Dipartimento di Informatica

Context-sensitive and Type 0 Grammars

Recall that a grammar is in general defined as

> **DEFINITION**
>
> A ***grammar*** is a 4-tuple (V, Σ, R, S), where
>
> **1.** V is a finite set called the ***variables***,
>
> **2.** Σ is a finite set, disjoint from V, called the ***terminals***,
>
> **3.** R is a finite set of ***rules***, with $R \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$
>
> **4.** $S \in V$ is the start variable.

Without any requirement on the productions (like in this definition), we have a ***type 0 grammar***

If we require that, for every $(\alpha,\beta) \in R$ we have that $|\alpha| \leq |\beta|$, the grammar is called ***context-sensitive***

→ REMARK: ε cannot belong to any C.S. language!

the string can never shrink

This name originates from the fact that every C.S. grammar can be turned into an equivalent (in the sense that they generate the same language) grammar where all productions have the form $(\alpha_1 A \alpha_2 , \alpha_1 \beta \alpha_2)$ for $A \in V$, $\alpha_1,\alpha_2 \in (\Sigma \cup V)^*$ and $\beta \in (\Sigma \cup V)^+$

In a CFG, $\alpha_1 = \alpha_2 = \varepsilon$ and so A can always be replaced by β; in a CSG, this is possible only within the context $\alpha_1[-]\alpha_2$

A first C.S. grammar (1)

Let us now provide a CSG for $\{a^n b^n c^n \mid n \geq 1\}$

→ REMARK: $\{a^n b^n c^n \mid n \geq 0\}$ is NOT a C.S. language because it includes ε !!

Consider

S ::= abc | $abSc$

ba ::= ab

Intuitively, for generating $a^n b^n c^n$:

- If $n = 1$, apply rule S ::= abc
- Otherwise, apply n-1 times rule S ::= $abSc$ to produce $(ab)^{n-1}Sc^{n-1}$
- Finally use rule S ::= abc to produce $(ab)^{n-1}abc^n = (ab)^n c^n$
- We have now to rearrange the a's and b's
- all a's must be moved at left (rule ba ::= ab)

For example, let's derive $aaabbbccc$:

$$S \Rightarrow abSc \Rightarrow ababScc \Rightarrow ababab\underline{c}cc \Rightarrow aabbabccc \Rightarrow aababbccc \Rightarrow aaabbbccc$$

A first C.S. grammar (2)

Problem: the previous grammar also generates strings that are NOT in the language!

→ for example, $S \Rightarrow^* abababccc \notin \{a^n b^n c^n \mid n > 1\}$

it can generate for example (ab)^n c^n

So, consider

S ::= abc | $aBSc$

Ba ::= aB

Bb ::= bb

we need some new non-terminal symbols to accept only the correct strings

Now, for generating $a^n b^n c^n$ (n > 1):

- Apply n-1 times rule S ::= $abSc$ to produce $(aB)^{n\text{-}1}Sc^{n\text{-}1}$
- Then, use rule S ::= abc to produce $(aB)^{n\text{-}1}abc^n$
- We have now to rearrange the a's and B's, and turn the B's into b's
  - Apply rule Ba ::= aB to the last occurrence of B
  - Apply rule Bb ::= bb to the last occurrence of B
  - Iterate this until no more B is around

Now, to derive $aaabbbccc$ we proceed as follows:

$$S \Rightarrow aBSc \Rightarrow aBaBScc \Rightarrow aBaBabccc \Rightarrow aBaaBbccc \Rightarrow aBaabbccc \Rightarrow$$

$$\Rightarrow aaBabbccc \Rightarrow aaaBbbccc \Rightarrow aaabbbccc$$

A more involved C.S. grammar (1)

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Let us now provide a CSG for $\{ww \mid w \in \{a,b\}^+\}$

The problem is that there isn't any way to directly generate the two *w*'s in the correct order.

A first attempt:

- Start with rule $S ::= WW$, and then let W generate a string of *a*'s and *b*'s
- But this won't work, since we have no way to force the two *W*'s to produce the same string !!

A second attempt:

- Let's start with $S ::= aSa$ and that we want *b* next
- We need $Sa ::= bSab$, since the new *b* has to come after the *a* already present
- Now we have *abSab* and let's say that we want *a* next
- Hence, we need $Sab ::= aSaba$
- The problem is that, as the length of the string grows, so does the number of rules we'll need to cope with all the patterns we could have to replace
- No finite number of rules can deal with replacing *S* and adding a new character that is arbitrarily far away from *S* !!

The right way:

- First produce ww^R (we know how to do this, since this is a C.F. language)
- Then, reverse w^R without touching the initial *w*, so to obtain *ww*

A more involved C.S. grammar (2)

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Idea:

- we generate wW^R, where $W \in \{A,B\}^+$ (so it's a string of variables);
- from wW^R, we use "swapping" productions xX ::= Xx to reverse the variable part, where A and B are turned into a and b, respectively.

Problem: terminal symbols from the second half of the word should not be mixed with ones from the first half by the swapping productions!

Solution: a *marker* \$ is usually added to the variables (to generate $w\$W^R$) so that swapping happens just to the right of it

→ Such marker is eventually disposed with a ***ε-rule***

So, for now, we use a type 0 grammar (then turned into a C.S. one)

S ::= aSA | bSB | \$	aA ::= Aa
\$ ::= ε	aB ::= Ba
$\$A$ ::= $\$a$	bA ::= Ab
$\$B$ ::= $\$b$	bB ::= Bb

Let's derive *abbabb*:

$S \Rightarrow aSA \Rightarrow abSBA \Rightarrow abbSBBA \Rightarrow abb\$BBA \Rightarrow abb\$bBA \Rightarrow abb\$BbA \Rightarrow abb\$BAb \Rightarrow$

$\Rightarrow abb\$bAb \Rightarrow abb\$Abb \Rightarrow abb\$abb \Rightarrow abbabb$

A more involved C.S. grammar (3)

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

For having a CSG, the idea is to turn the ε-rule to $\$ ::= a \mid b$

But the final word must be of even length, so one can use two markers (and work with $w\$W^R\$$)

However, we then should turn *both* the markers to the *same* terminal → HOW???

Solution:

- If we replace $S ::= aSA \mid bSB \mid \$$ and $\$ ::= \varepsilon$ with

  $S ::= S_a\$_a \qquad S_a ::= aS_aA \mid bS_aB \mid \$_a \qquad \$_a ::= a$

  we obtain a grammar that generates $\{wawa \mid w \in \{a,b\}^*\}$. in these cases, w can be epsilon, hence {a,b}*

- Similarly, with

  $S ::= S_b\$_b \qquad S_b ::= aS_bA \mid bS_bB \mid \$_b \qquad \$_b ::= b$

  we obtain a grammar that generates $\{wbwb \mid w \in \{a,b\}^*\}$. in these cases, w can be epsilon, hence {a,b}*

- Hence, this is a CSG for $\{ww \mid w \in \{a,b\}^+\}$: here, we don't have the empty string, as it's context-sensitive

  $S ::= S_a\$_a \mid S_b\$_b \qquad S_a ::= aS_aA \mid bS_aB \mid \$_a \qquad \$_a ::= a$

  $S_b ::= aS_bA \mid bS_bB \mid \$_b \qquad \$_b ::= b$

  $\$_aA ::= \$_aa \qquad \$_aB ::= \$_ab \qquad \$_bA ::= \$_ba \qquad \$_bB ::= \$_bb$

  $aA ::= Aa \qquad aB ::= Ba \qquad bA ::= Ab \qquad bB ::= Bb$

Let's derive *abbabb*:

$$S \Rightarrow S_b\$_b \Rightarrow a\underline{S_b}A\$_b \Rightarrow ab\underline{S_b}BA\$_b \Rightarrow ab\underline{\$_bB}A\$_b \Rightarrow ab\underline{\$_b}bA\$_b \Rightarrow ab\underline{\$_b}Ab\$_b \Rightarrow$$

$$\Rightarrow ab\underline{\$_ba}b\$_b \Rightarrow abbab\$_b \Rightarrow abbabb$$

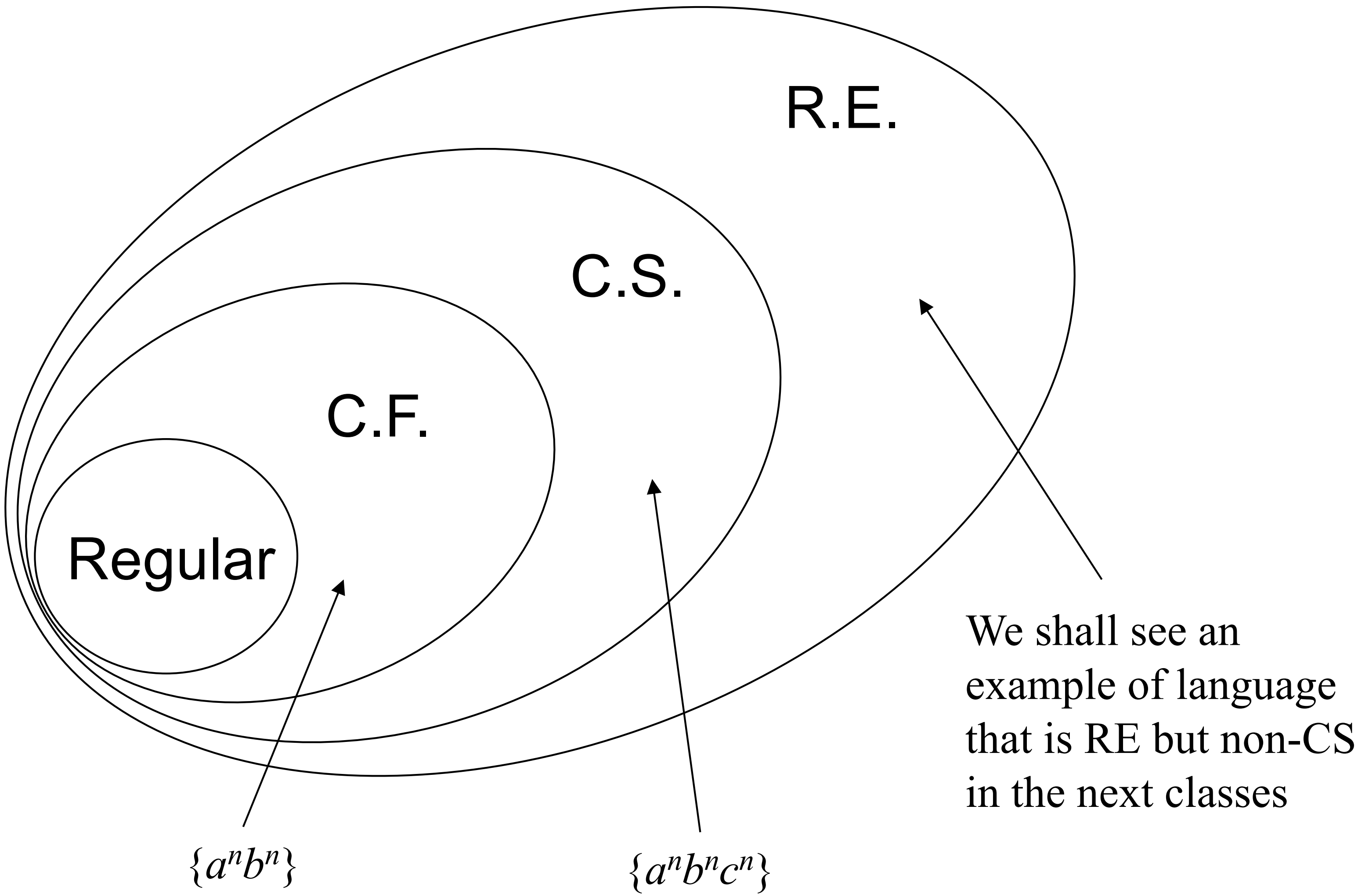
Chomsky hierarchy

Type of the grammar	***Shape of the Rules***	***Language produced***	
Type 0	$\alpha ::= \beta$ for $\alpha \neq \varepsilon$	Recursively enumarable (or semi-decidable)	Turing Machine
Type 1	$\alpha ::= \beta$ for $0 < \lvert\alpha\rvert \leq \lvert\beta\rvert$	Context-sensitive	Linear-bounded non-deterministic Turing machine
Type 2	$A ::= \beta$ for $A \in V$, $\beta \in (\Sigma \cup V)^*$	Context-free	Non-deterministic pushdown automaton
Type 3	$A ::= \beta$ for $A \in V$, $\beta \in (\Sigma^* \cup \Sigma^* V \cup V\Sigma^*)$	Regular	Finite state automaton

For languages that do not contain ε, we can easily see that every grammar of type i is also a grammar of type j, for every $j < i$.

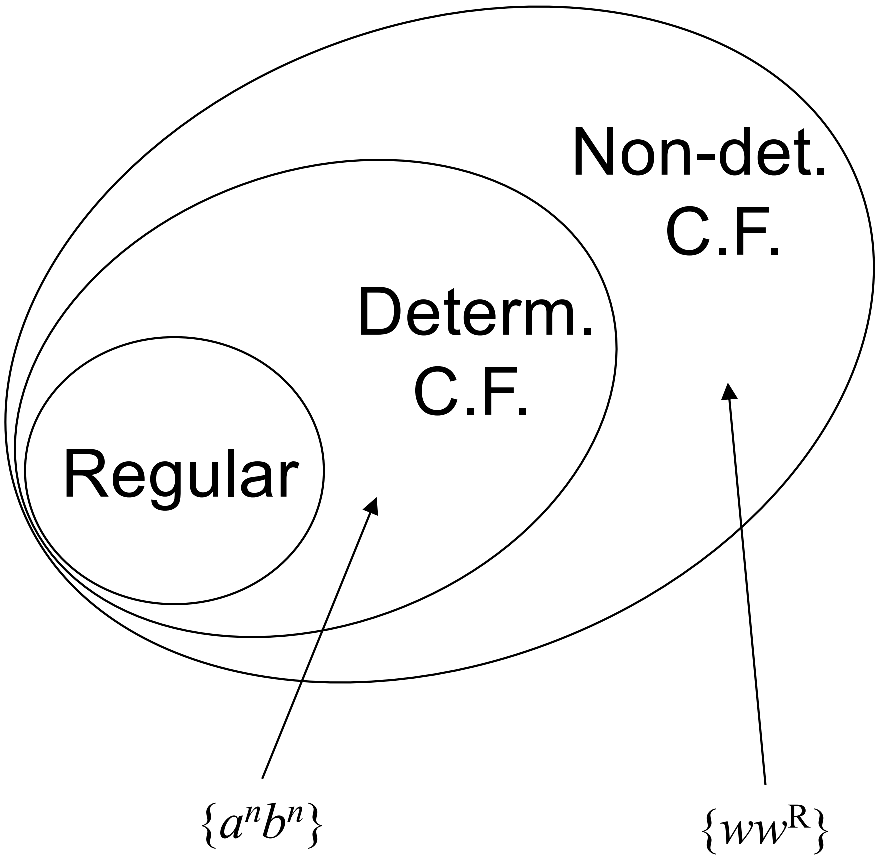
A hierarchy of (ε-free) languages

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA



Refining the hierarchy

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA



11