SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

FOUNDATIONS OF COMPUTER SCIENCE
LECTURE 4: Regular grammars

Prof. Daniele Gorla

Grammars for generating regular languages

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- We now provide a further method to characterize regular languages

- *Generative Grammars* can easily describe certain features that have a recursive structure, which makes them useful in a variety of applications

- They are a powerful tool in computer science (and not only) and their use goes far beyond regular languages

- (Generative) Grammars were first used in the formal study of human languages

- Grammars heavily occur in the specification and compilation of programming languages:
  - A grammar for a programming language is the reference way to learn the language syntax;
  - Designers of compilers and interpreters for programming languages often start by the grammar;
  - Most compilers and interpreters contain a component called a ***parser*** (originating from the grammar) that extracts the meaning of a program before generating the compiled code or performing the interpreted execution.

Algebric Expressions on Natural numbers

recursive way of describing syntactic entities

EXPR	::=	EXPR+EXPR | EXPR×EXPR | (EXPR) | NUMB
NUMB	::=	DIGIT | NONZERODIGIT DIGITSEQ
DIGITSEQ	::=	DIGIT | DIGIT DIGITSEQ
DIGIT	::=	0 | NONZERODIGIT
NONZERODIGIT	::=	1 | 2 | … | 9

Here:

- Capital-letter words are items that have to be replaced for generating a valid entity
- Digits 0,…,9 and symbols +,×,(,) are constants
- Only sequences of constants are elements of the laguage generated by the grammar

EX.: Number 1903 is produced as

NUMB ⇒ NONZERODIGIT DIGITSEQ ⇒ 1 DIGITSEQ ⇒ 1 DIGIT DIGITSEQ ⇒

⇒ 19 DIGITSEQ ⇒ 19 DIGIT DIGITSEQ ⇒ 190 DIGITSEQ ⇒ 190 DIGIT

⇒ 1903

arrow stands for "followed by"

QUESTION: Why don’t we use the following (simpler) description for numbers?

NUMB ::= DIGIT | DIGIT NUMB

Grammar for Algebraic Expressions (cont'd)

EX.: Number 31 is produced as

NUMB ⇒ NONZERODIGIT DIGITSEQ ⇒ 3 DIGITSEQ ⇒ 3 DIGIT ⇒ 31

EX.: Expression (1903+31)×31 is produced as

EXPR ⇒ EXPR×EXPR ⇒ (EXPR)×EXPR ⇒ (EXPR+EXPR)×EXPR

⇒⇒⇒ (NUMB+NUMB)×NUMB

⇒⇒⇒⇒⇒⇒⇒⇒⇒ (1903+NUMB)×NUMB (see previous slide)

⇒⇒⇒⇒ (1903+31)×NUMB (see above)

⇒⇒⇒⇒ (1903+31)×31 (see above)

Formal Definition

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.

Notationally, rules with the same LHS (i.e., first component) are grouped together, i.e.

If (α,β_1) , … , (α,β_n) are all the rules in R with first component α,

we shall write them as $\alpha ::= \beta_1 \mid \ldots \mid \beta_n$

::= reads "rewrites"

If u, v, α , β are strings of variables and terminals, and (α,β) is a rule of the grammar, we say that $u\alpha v$ ***yields*** $u\beta v$, written $u\alpha v \Rightarrow u\beta v$.

yields -> replaces

We say that u ***derives*** v, written $u \Rightarrow^{*} v$, if $u = v$ or there exists a sequence $u_1, u_2, \ldots, u_k$ (for $k \geq 0$) such that $u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v$.

The ***language of the grammar*** G is $L(G) = \{w \in \Sigma^{*} \mid S \Rightarrow^{*} w\}$.

The previous example, formally

In the grammar for algebraic expressions, we have that

- V = {EXPR , NUMB , DIGIT , NONZERODIGIT , DIGITSEQ}
- Σ = {0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , + , × , (,) }
- S = EXPR
- R has been given in the first slide of the example, viz.

$$R = \left\{ \begin{array}{l} \text{(EXPR , EXPR+EXPR) , (EXPR , EXPR×EXPR) , (EXPR , (EXPR)) , (EXPR , NUMB) ,} \\ \text{(NUMB , DIGIT) , (NUMB , NONZERODIGIT DIGITSEQ) ,} \\ \text{(DIGITSEQ , DIGIT) , (DIGITSEQ , DIGIT DIGITSEQ) ,} \\ \text{(DIGIT , 0) , (DIGIT , NONZERODIGIT) ,} \\ \text{(NONZERODIGIT , 1) , (NONZERODIGIT , 2) , ... (NONZERODIGIT , 9)} \end{array} \right\}$$

Grammars are usually given just by their rules (grouped by the LHSs):

- The variables are denoted in capital letters (or in square parenthesis, e.g. <EXPR>)
- Terminals are usually well-identifiable symbols (or simply the remaining symbols)
- The starting variable is usually the left part of the first production given

the "-" is NOT a terminal symbol

As we saw before, (1903+31)×31 ∈ $L(G)$, whereas -2×31 ∉ $L(G)$.

Regular Grammars

According to the restrictions that we pose on the shape of the rules, we have different kinds of grammars

both L & R lineas:
every variable rewrites in a string of terminals

A grammar $G = (V, \Sigma, R, S)$ is said

- ***left-linear*** if, for every $(\alpha, \beta) \in R$, we have that $\alpha \in V$ and $\beta \in (\Sigma^* \cup V\Sigma^*)$;
- ***right-linear*** if, for every $(\alpha, \beta) \in R$, we have that $\alpha \in V$ and $\beta \in (\Sigma^* \cup \Sigma^* V)$;
- ***regular***, if it is either left- or right-linear.

EXAMPLE: The following grammar for natural numbers (derived from the previous one for algebraic expressions on naturals) is regular (actually, right-linear):

NUMB ::= 0 | 1 DIGITSEQ | ... | 9 DIGITSEQ

DIGITSEQ ::= ε | 0 DIGITSEQ | ... | 9 DIGITSEQ

this is right linear as we have a terminal symbol followed by a variable (see def above)

EXERCISE: define a left-linear grammar for natural numbers.

REMARK: For algebraic expressions, regular grammars are not enough

(We will see other kinds of grammars later on in this course!)

Another example for regular grammars

The (regular) language $0(10)^*$ is generated by the right-linear grammar

$S ::= 0A$

$A ::= \varepsilon \mid 10A$

or, equivalently, by the left-linear grammar

$S ::= 0 \mid S10$

With the first grammar, we can derive 0101010 as follows:

$S \Rightarrow 0A \Rightarrow 010A \Rightarrow 01010A \Rightarrow 0101010A \Rightarrow 0101010$

With the second grammar, the derivation is instead:

$S \Rightarrow S10 \Rightarrow S1010 \Rightarrow S101010 \Rightarrow 0101010$

Regular Languages vs Right-linear Grammars

Thm1: If L is regular, then it is generated by a right-linear grammar.

Proof:

Let $L = L(M)$ for a DFA $M = (Q, \Sigma, \delta, q_0, F)$.

the variables are essentially the states

First suppose that q_0 is not a final state.

Then, $L = L(G)$, where G is the right-linear grammar (Q, Σ, R, q_0), where R includes

- $q ::= aq'$, whenever $\delta(q, a) = q'$
- $q ::= a$, whenever $\delta(q, a) \in F$.

Then, by induction on $|w|$, we can prove that $\delta(q, w) = q'$ if and only if $q \Rightarrow^* wq'$.

If q_0 is final, we consider the grammar $G' = (Q \cup \{S\}, \Sigma, R', S)$ with $S \notin Q$ and

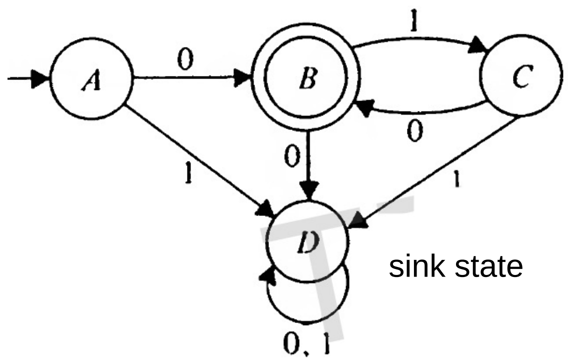
$$R' = R \cup \{ S ::= \varepsilon \mid q_0 \}$$

where R is built as above.

Q.E.D.

Example

Consider the following DFA for language 0(10)* :



The right-linear grammar from this DFA is

A ::= 0B | 1D | 0

B ::= 0D | 1C

C ::= 0B | 1D | 0

D ::= 0D | 1D

remember that ::= is rewrites
basically it's a transition fcn (e.g. A with 0 goes to B, so A ::= 0B)

construction is easier with right-linear grammar (more natural, as we first give 0/1 and then transition)

Notice that variable *D* is not really needed (in the automaton we need state *D* because the automaton is deterministic, and so we have to handle, e.g., strings that start with a 1) in the grammar it's not needed as D only transitions to D (sink state)

Hence, a more compact grammar (equivalent to the previous one) is

A ::= 0B | 0

B ::= 1C

C ::= 0B | 0

Regular Languages vs Left-linear Grammars

Cor1: If L is regular, then it is generated by a left-linear grammar.

Proof:

Since L is regular, also L^R is regular (by closure properties of reg.lang's).

Let $L^R = L(M)$ for a DFA M.

By Thm1, there exists a right-linear grammar G s.t. $L^R = L(G)$.

Now, consider G^R, the grammar obtained from G by reversing the RHSs of all its productions.

G^R is left-linear and $L(G^R) = (L(G))^R = (L^R)^R = L$.

Q.E.D.

reverse language and then reverse grammar

Right-linear Grammars vs Regular Languages

Thm2: If L is generated by a right-linear grammar, then L is regular.

Proof:

Let $L=L(G)$, for some right-linear grammar $G = (V,\Sigma,R,S)$.

We construct an NFA with ε-moves, $M = (Q, \Sigma, \delta, q_0, F)$ that simulates derivations in G:

- Q consists of the symbols $[\alpha]$ such that $\alpha = S$ or α is a (not necessarily proper) suffix of some right-hand side of a rule in R all the suffixes of all the LHS
- $q_0 = [S]$
- $F = \{ [\varepsilon] \}$ a string always finishes with epsilon, so it's the only final state
- δ is defined as follows:
  - If $A \in V$, then $\delta([A], \varepsilon) = \{ [\alpha] \mid A ::= \alpha \in R \}$
  - If $a \in \Sigma$ and $\alpha \in (\Sigma^* \cup \Sigma^* V)$, then $\delta([a\alpha], a) = \{ [\alpha] \}$. suffix evolution

By induction on $|w|$, we can show that $[\alpha] \in \delta([S], w)$ if and only if $S \Rightarrow^* w\alpha$.

As $[\varepsilon]$ is the unique final state, M accepts w (i.e., $[\varepsilon] \in \delta([S], w)$) if and only if $S \Rightarrow^* w$.

Q.E.D.

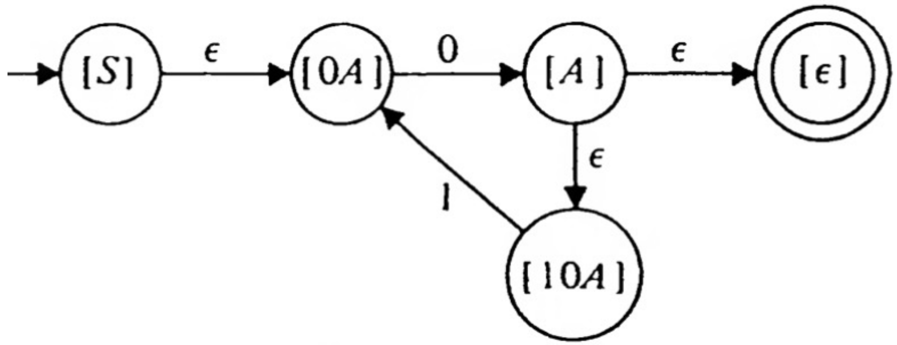
Example

Consider the right-linear grammar

$S ::= 0A$ 0(10)*

$A ::= \varepsilon \mid 10A$

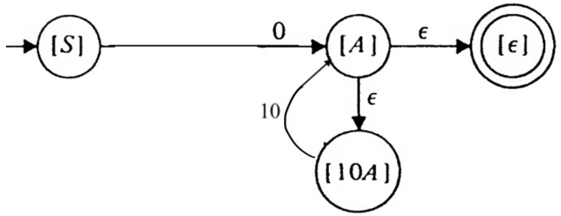
Its associated automaton is:



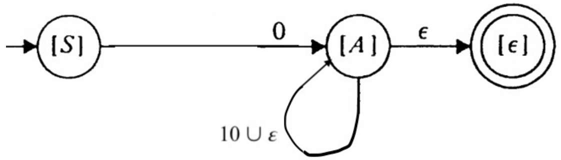
vedi a casa, importante

By deriving its associated regular expression (as we saw in the previous class), we obtain

- Consider q_{rip} to be $[0A]$
  - $q_i = [S]$, $q_j = [A]$, R1 = R2 = ε, R3 = 0 and R4 = $\emptyset$ yield $[S] \xrightarrow{\varepsilon\varepsilon^*0 \cup \emptyset} [A]$, with $\varepsilon\varepsilon^*0 \cup \emptyset = 0$
  - $q_i = [10A]$, $q_j = [A]$, R1 = 1, R2 = ε, R3 = 0 and R4 = $\emptyset$ yield $[10A] \xrightarrow{1\varepsilon^*0 \cup \emptyset} [A]$, with $1\varepsilon^*0 \cup \emptyset = 10$



- Consider q_{rip} to be $[10A]$
  - $q_i = q_j = [A]$, R1 = R2 = ε, R3 = 10 and R4 = ε yield $[A] \xrightarrow{\varepsilon\varepsilon^*10 \cup \varepsilon} [A]$, with $\varepsilon\varepsilon^*10 \cup \varepsilon = 10 \cup \varepsilon$



- Consider q_{rip} to be $[A]$
  - $q_i = [S]$, $q_j = [\varepsilon]$, R1 = 0, R2 = $10 \cup \varepsilon$, R3 = ε and R4 = $\emptyset$ yield $[S] \xrightarrow{0(10 \cup \varepsilon)^*\varepsilon \cup \emptyset} [\varepsilon]$, with $0(10 \cup \varepsilon)^*\varepsilon \cup \emptyset = 0(10)^*$

Left-linear Grammars vs Regular Languages

Cor2: If L is generated by a left-linear grammar, then L is regular.

Proof:

Let $L=L(G)$, for some left-linear grammar G.

Consider G^R, the right-linear grammar obtained by G by reversing the RHSs of its productions.

Trivially, $L(G^R) = (L(G))^R$.

By Thm2, since G^R is right-linear, $L(G^R)$ is a regular language.

Since regular languages are closed by reversion, $(L(G^R))^R$ is regular.

But $(L(G^R))^R = ((L(G))^R)^R = L(G)$, and so $L = L(G)$ is regular.

Q.E.D.

Left- vs Right-linear Grammars

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Cor: *L* has a right-linear grammar if and only if it has a left-linear grammar.

Proof:

$L=L(G)$, for some right-linear grammar

IFF L is regular (Thm1+Thm2)

IFF $L=L(G)$, for some left-linear grammar (Cor1+Cor2)

Q.E.D.

REMARK 1: this Corollary also give an algorithm for passing from a left- to a right-linear grammar, and vice versa (by passing through automata)

REMARK 2: usually, right-linear grammars are much easier to invent and handle

→ there are algorithms that directly (and more efficiently) transform a left- into a right-linear grammar (out of the scope of this course)

però vedi comunque se trovi qualcosa