

Automata & Complexity

Jörg Endrullis

Vrije Universiteit Amsterdam

2018

Looking Back

Previous subjects relevant for this lecture:

- languages as sets of words
- finite automata: deterministic and non-deterministic
- language L is **regular** \iff L accepted by a **DFA**
- language L is **regular** \iff L accepted by a **NFA**

Grammars

A **grammar** defines a **language**.

Applications areas:

- natural language,
- artificial intelligence,
- syntax of programming languages.

Example

⟨sentence⟩ → ⟨article⟩ ⟨noun⟩ ⟨verb⟩ ⟨article⟩ ⟨noun⟩
⟨article⟩ → the
⟨article⟩ → a
⟨noun⟩ → farmer
⟨noun⟩ → cow
⟨verb⟩ → milks

With these **grammar rules** we can construct a ⟨sentence⟩.

Grammars Example

$\langle \text{sentence} \rangle \rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle \langle \text{verb} \rangle \langle \text{article} \rangle \langle \text{noun} \rangle$
 $\langle \text{article} \rangle \rightarrow \text{the}$
 $\langle \text{article} \rangle \rightarrow \text{a}$
 $\langle \text{noun} \rangle \rightarrow \text{farmer}$
 $\langle \text{noun} \rangle \rightarrow \text{cow}$
 $\langle \text{verb} \rangle \rightarrow \text{milks}$

The farmer milks a cow is a sentence in the language.

$\langle \text{sentence} \rangle \Rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle \langle \text{verb} \rangle \langle \text{article} \rangle \langle \text{noun} \rangle$
 $\Rightarrow \text{the} \langle \text{noun} \rangle \langle \text{verb} \rangle \langle \text{article} \rangle \langle \text{noun} \rangle$
 $\Rightarrow \text{the farmer} \langle \text{verb} \rangle \langle \text{article} \rangle \langle \text{noun} \rangle$
 $\Rightarrow \text{the farmer milks} \langle \text{article} \rangle \langle \text{noun} \rangle$
 $\Rightarrow \text{the farmer milks a} \langle \text{noun} \rangle$
 $\Rightarrow \text{the farmer milks a cow}$



The cow milks the farmer is also a sentence in the language.

Grammars

A **grammar** $G = (V, T, S, P)$ consists of:

- finite set V of **non-terminals** (or **variables**)
- finite set T of **terminals**
- a **start symbol** $S \in V$
- finite set P of **production rules** $x \rightarrow y$ where
 - $x \in (V \cup T)^+$ containing at least one symbol from V
 - $y \in (V \cup T)^*$

(for context-free grammars: $x \in V$)

In the previous example:

- variables: $\langle \text{sentence} \rangle$, $\langle \text{article} \rangle$, $\langle \text{noun} \rangle$, $\langle \text{verb} \rangle$
- terminals: the, a, farmer, cow, milks
- starting symbol: $\langle \text{sentence} \rangle$

Languages Generated by Grammars

If $x \rightarrow y$ is a production rule, then we have a **derivation step**

$$uxv \Rightarrow uyv$$

for every $u, v \in (V \cup T)^*$.

A **derivation** \Rightarrow^* is the reflexive, transitive closure of \Rightarrow .

Thus there is a derivation $u \Rightarrow^* v$ if v can be obtained from u by zero or more derivation steps.

The **language generated** by a grammar $G = (V, T, S, P)$ is

$$L(G) = \{ w \in T^* \mid S \Rightarrow^* w \}$$

The language consists of all words that

- contain only terminal letters (no variables), and
- can be derived from the start symbol.

Exercises

$G = (\{S\}, \{a, b\}, S, P)$, where P consists of

$$S \rightarrow aSb$$

$$S \rightarrow \lambda$$

Example derivations:

$$S \Rightarrow \lambda$$

$$S \Rightarrow aSb \Rightarrow ab$$

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

What is the language generated by G as a set?

$$L(G) = \{a^n b^n \mid n \geq 0\}$$

Notational Conventions for Grammars

When defining $G = (V, T, S, P)$ we use

- capital case letters for non-terminals (variables)
- lower case letters for terminals

Often, we only specify the production rules.

(Groups of two, 1 minute)

$$G_1 : S \rightarrow Ab$$

$$A \rightarrow aAb$$

$$A \rightarrow \lambda$$

$$G_2 : S \rightarrow aSb$$

$$S \rightarrow b$$

What languages are generated by these grammars?

$$L(G_1) = \{a^n b^{n+1} \mid n \geq 0\} = L(G_2) = \{a^n b^{n+1} \mid n \geq 0\}$$

Exercises (1)

(Individual, 1 minute)

Find a grammar G such that

$$L(G) = \{a, b\}^* \{c\} \{b, c\}^*$$

Exercises (2)

A word $w = a_1 a_2 \cdots a_n$ is called **palindrome** if $w = w^R$, that is

$$a_1 a_2 \cdots a_n = a_n \cdots a_2 a_1$$

(Individual, 2 minutes)

Find a grammar G that generates all palindromes over the alphabet $\Sigma = \{a, b\}$. In other words

$$L(G) = \{w \in \Sigma^* \mid w \text{ is a palindrome}\}$$

(Groups of two, 3 minutes)

Find a grammar G that generates all non-palindromes over the alphabet $\Sigma = \{a, b\}$. In other words

$$L(G) = \{w \in \Sigma^* \mid w \text{ is not a palindrome}\}$$

B(ackus) N(aur) F(orm) is a Context-Free Grammar

Non-terminals (variables) are indicated by \langle and \rangle .

Example

$\langle \text{stm} \rangle \rightarrow \langle \text{var} \rangle := \langle \text{expr} \rangle$

$\langle \text{stm} \rangle \rightarrow \langle \text{stm} \rangle ; \langle \text{stm} \rangle$

$\langle \text{stm} \rangle \rightarrow \mathbf{begin} \langle \text{stm} \rangle \mathbf{end}$

$\langle \text{stm} \rangle \rightarrow \mathbf{if} \langle \text{cond} \rangle \mathbf{then} \langle \text{stm} \rangle \mathbf{else} \langle \text{stm} \rangle$

$\langle \text{stm} \rangle \rightarrow \mathbf{while} \langle \text{cond} \rangle \mathbf{do} \langle \text{stm} \rangle$

$\langle \text{cond} \rangle \rightarrow \dots$

$\langle \text{var} \rangle \rightarrow \dots$

$\langle \text{expr} \rangle \rightarrow \dots$

$\dots \rightarrow \dots$

$\langle \text{var} \rangle \rightarrow x$

Write $\langle \text{var} \rangle ::= x \mid y \mid z$ instead of $\langle \text{var} \rangle \rightarrow y$

$\langle \text{var} \rangle \rightarrow z$

Right Linear Grammars

A grammar $G = (V, T, S, P)$ is **right linear** if all production rules are of the form

$$A \rightarrow uB \quad \text{or} \quad A \rightarrow u$$

with $A, B \in V$ and $u \in T^*$.

Moreover G is **strictly right linear** if $|u| \leq 1$ (i.e. $u \in (T \cup \{\lambda\})$).

(Individual, 1 minute)

Construct a right linear grammar G such that

$$L(G) = \{a, b\}^* \{aa\} \{b\}^*$$

(Groups of two, 2 minutes)

Construct a right linear grammar G such that

$$L(G) = \{ab\} (\{a\}^* \{cb\})^* \{b\}$$

(Strictly) Right Linear Grammars

Theorem

Let G be a right linear grammar G . There exists a **strictly** right linear grammar H such that $L(G) = L(H)$.

Construction

Let $G = (V, T, S, P)$ be a right linear grammar.

Assume that we have a production rule γ of the form $A \rightarrow u(B)$ with $A, B \in V$ and $u \in T^*$ with $|u| > 1$.

Then $u = aw$ for some $a \in T$ and $w \in T^+$.

Let X be a **fresh** variable (that is, $X \notin (V \cup T)$).

We add X to V and split the rule γ into:

$$A \rightarrow aX \qquad X \rightarrow w(B)$$

Then $A \rightarrow aX \rightarrow aw(B) = u(B)$. It follows $L(G) = L(H)$.

Repeat splitting until $|u| \leq 1$ for all rules.

Right Linear Grammars \iff Regular Languages

Theorem

Language L is **regular**

\iff there is a **right linear grammar** G with $L(G) = L$

Proof.

We need to prove two directions:

- (\Rightarrow) Translate NFA's (or DFA's) into right linear grammars.
- (\Leftarrow) Translate right linear grammars to NFA's (or DFA's).



Exercise for (\Rightarrow)

(Groups of two, 2 minutes)

Consider the following NFA M



Construct a right linear grammar G such that:

$$L(M) = L(G)$$

(\Rightarrow) From NFA's to Right Linear Grammars

Construction (\Rightarrow)

Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA.

We construct a right linear grammar $G = (V, T, S, P)$ with

$$L(G) = L(M)$$

We define $V = Q$ and $T = \Sigma$ and $S = q_0$.

The set P consists of the following production rules

$$q \rightarrow aq' \quad \text{for every } q' \in \delta(q, a)$$

$$q \rightarrow q' \quad \text{for every } q' \in \delta(q, \lambda)$$

$$q \rightarrow \lambda \quad \text{for every } q \in F$$

There is a derivation in G from S to $u \in T^* \iff$ there is a path in M from S to a state in F labelled with the letters from u .

It follows that, $L(G) = L(M)$.

(\Leftarrow) From Right Linear Grammars to NFA's

Construction (\Leftarrow)

Let $G = (V, T, S, P)$ be a right linear grammar.

Transform G to **strictly** right linear grammar generating the same language. Then all rules are of the form

$$A \rightarrow uB \qquad \text{or} \qquad A \rightarrow u$$

with $A, B \in V$ and $u \in (T \cup \{\lambda\})$.

We construct an NFA $M = (Q, \Sigma, \delta, q_0, F)$ with $L(M) = L(G)$:

$$\Sigma = T \qquad Q = V \cup \{A_f\} \qquad q_0 = S \qquad F = \{A_f\}$$

where $A_f \notin V$. We introduce the following arrows in M :

- $A \xrightarrow{u} B$ for every production rule $A \rightarrow uB$, and
- $A \xrightarrow{u} A_f$ for every production rule $A \rightarrow u$.

Then $S \Rightarrow^* w \in T^* \iff M$ accepts w . Hence $L(G) = L(M)$.

Exercise

(Groups of two, 2 minutes)

Construct an NFA that accepts the language generated by

$$S \rightarrow aT$$

$$T \rightarrow abcS \mid b$$

Note that $T \rightarrow abcS \mid b$ is short for two rules:

$$T \rightarrow abcS$$

$$T \rightarrow b$$

Left Linear Grammars

A grammar $G = (V, T, S, P)$ is **left linear** if all production rules are of the form

$$A \rightarrow Bu \quad \text{or} \quad A \rightarrow u$$

with $A, B \in V$ and $u \in T^*$.

(Difference with right linear grammars highlighted in red.)

Theorem

Language L is **regular**

\iff there is a **left linear grammar** G with $L(G) = L$

Proof.

L is regular $\iff L^R$ is regular

\iff right linear grammar for L^R

\iff left linear grammar for L

(For the last step, reverse all production rules.)



Mixing Right and Left Linear Rules

Mixing right **and** left linear rules, the generated language is **not** always regular.

Example

Let G be the grammar

$$S \rightarrow aA$$

$$A \rightarrow Sb$$

$$S \rightarrow \lambda$$

Every rule of G is either right or left linear.

However, the language $L(G) = \{a^n b^n \mid n \geq 0\}$ is **not** regular.

Regular Expressions

We define the **regular expressions** over an alphabet Σ :

- \emptyset is a regular expression
- λ is a regular expression
- a is a regular expression for every $a \in \Sigma$
- $r_1 + r_2$ is a regular expression for all regular expr. r_1 and r_2
- $r_1 \cdot r_2$ is a regular expression for all regular expr. r_1 and r_2
- r^* is a regular expression for all regular expressions r

A regular expression is syntax, describing a language.

Every **regular expression** r defines a **language** $L(r)$:

$$L(\emptyset) = \emptyset$$

$$L(\lambda) = \{\lambda\}$$

$$L(a) = \{a\} \text{ for } a \in \Sigma$$

$$L(r_1 + r_2) = L(r_1) \cup L(r_2)$$

$$L(r_1 \cdot r_2) = L(r_1)L(r_2)$$

$$L(r^*) = L(r)^*$$

Example

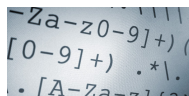
Example

$$L((a + b) \cdot c^*) = \{a, b\}c^*$$

Regular expressions are used to search and manipulate text.

For example:

- grep in Linux
- script languages such as Perl



Every major programming language has regular expressions.

(Groups of two, 1 minute)

Exercise

Find a regular expression r over $\Sigma = \{a, b\}$ such that $L(r)$ consists of all words that contain the pattern *bab*.

Regular Expressions \iff Regular Languages

Theorem

A language L is **regular**

\iff there is a **regular expression** r with $L(r) = L$.

Proof.

We need to prove two directions:

- (\Leftarrow) Translate regular expressions into NFA's.
- (\Rightarrow) Translate NFA's into regular expressions.



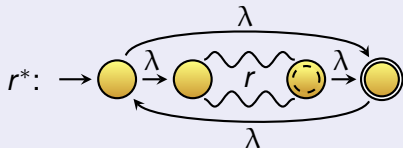
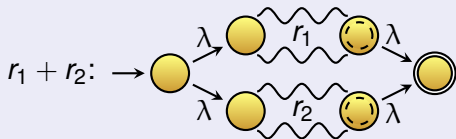
(\Leftarrow) From Regular Expressions to NFA's

Construction (\Leftarrow)

For every regular expression r , we build an NFA M such that

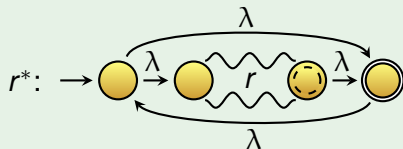
- $L(M) = L(r)$,
- M has precisely one final state (different from starting state)

We construct M by induction (recursion) on r .



Exercise

Understanding the start case



Note that:

- $((a^*) \cdot b)^*$ shows that the new starting state is needed
- $(a \cdot (b^*))^*$ shows that the new final state is needed

What goes wrong without introducing the new start/final state?

(\Rightarrow) From NFA's to Regular Expressions (1)

Construction (\Rightarrow)

For every NFA M , we construct a regular expression r with

$$L(r) = L(M)$$

Step 1:

We transform M such that there is only one final state.

If M has multiple final states:

- Add a fresh state q_f to M .
- For every final state q in M , add an arrow $q \xrightarrow{\lambda} q_f$ to M .
- Define q_f as the only final state.

Now M has precisely one final state ($\#F = 1$).

(\Rightarrow) From NFA's to Regular Expressions (2)

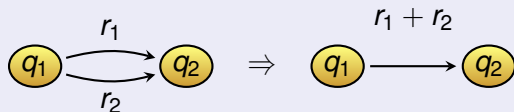
Construction (\Rightarrow continued)

Step 2:

We remove all double arrows.

We use transition graphs with **regular expressions as labels**.

If there are 2 arrows from a state q_1 to q_2 with labels r_1 and r_2 replace them by one arrow with label $r_1 + r_2$:



Note that q_1 can be equal to q_2 . Then the arrows are loops!

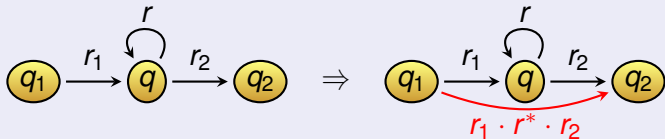
We remove all double arrows before continuing with Step 3.

(\Rightarrow) From NFA's to Regular Expressions (3)

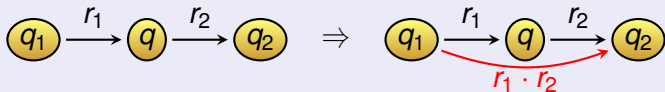
Construction (\Rightarrow continued)

Step 3: As long as there are states other than the starting and final state, remove **one** other state q as follows.

For all states q_1 and q_2 and arrows $q_1 \xrightarrow{r_1} q$ and $q \xrightarrow{r_2} q_2$, we add an arrow from q_1 to q_2 as follows:



for the case that there is an arrow $q \xrightarrow{r} q$, and otherwise:



Note that q_1 can be equal to q_2 .

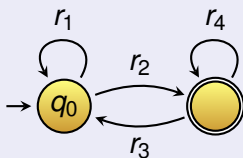
We **repeat** Step 2 and Step 3 until there is nothing to be done.

(\Rightarrow) From NFA's to Regular Expressions (4)

Construction (\Rightarrow continued)

Step 4:

If $F \neq \{q_0\}$, then the transition graph is finally of the form:



Possibly r_1 , r_2 , r_3 or r_4 are equal to \emptyset .

Then the regular expression is:

$$L(r_1^* \cdot r_2 \cdot (r_4 + r_3 \cdot r_1^* \cdot r_2)^*) = L(M)$$

Question

What is the form of the transition graph and regular expression for the case that $F = \{q_0\}$?

Exercise

(Groups of two, 3 minutes)

Find a regular expression r such that

$$L(r) = \{ w \in \{a, b\}^* \mid n_a(w) \text{ even and } n_b(w) \text{ is odd} \}$$

where

- $n_a(w)$ is the number of a 's in w , and
- $n_b(w)$ is the number of b 's in w .

(Groups of two, 4 minutes)

Find a regular expression r over $\{a, b\}$ such that

$L(r)$ consists of all words that do **not** contain the pattern *bab*.

Alternative Descriptions of Regular Languages

The following statements are equivalent:

- There is a **DFA** M with $L(M) = L$.
- There is an **NFA** M with $L(M) = L$.
- There is a **right linear grammar** G with $L(G) = L$.
- There is a **left linear grammar** G with $L(G) = L$.
- There is a **regular expression** r with $L(r) = L$.

Looking Forward

Read:

- Linz 1.2, 3.1–3.3

Do the following exercises:

- Linz 1.2: 11a,b,c, 13, 14a,b,e,f,h
- Linz 3.1: 5, 7, 13, 16,a,b
- Linz 3.2: 1, 2, 4b,d, 9, 10a,c, 13a
- Linz 3.3: 1, 3, 6, 12

Following lecture:

- Properties of regular languages