Automata & Complexity

Jörg Endrullis

Vrije Universiteit Amsterdam

2018

Looking Back

Last Lecture:

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 $L(r) = L$.

Elementary Properties of Regular Languages

Theorem

If L_1, L_2, L are regular languages, then

$$L_1 \cup L_2\,, \quad L_1 \cap L_2\,, \quad L_1 L_2\,, \quad \overline{L}\,, \quad L_1 \backslash L_2\,, \quad L^*\,, \quad L^R$$

are also regular.

Proof.

Let r_1, r_2, r be regular expr. with $L(r_1) = L_1$, $L(r_2) = L_2$, $L(r) = L$.

- $L_1 \cup L_2 = L(r_1 + r_2)$ is regular.
- $L_1 L_2 = L(r_1 \cdot r_2)$ is regular.
- $L^* = L(r^*)$ is regular.
- L is accepted by some DFA $(Q, \Sigma, \delta, q_0, F)$.
  $\overline{L} = L((Q, \Sigma, \delta, q_0, Q \backslash F))$ is regular.
- $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ is regular. □
- $L_1 \backslash L_2 = L_1 \cap \overline{L_2}$ is regular.

L^R is Regular

There are various possible proofs:

Construction 1

- Take an NFA that accepts L.
- Reverse all arrows.
- Swap final and starting states.

The resulting automaton accepts L^R.

Construction 2

- Take a right linear grammar.
- Reverse all rules ($x \to y$ becomes $x^R \to y^R$).

The result is a left linear grammar for L^R.

L^R is Regular

Construction 3

Reverse a regular expression (defined by recursion):

$$
\begin{aligned}
rev(\emptyset) &= \emptyset \\
rev(\lambda) &= \lambda \\
rev(a) &= a \qquad (a \in \Sigma) \\
rev(r_1 + r_2) &= rev(r_1) + rev(r_2) \\
rev(r_1 \cdot r_2) &= rev(r_2) \cdot rev(r_1) \\
rev(r^*) &= rev(r)^*
\end{aligned}
$$

Using induction, it can be shown that

$$L(rev(r)) = L(r)^R$$

Decidability of Membership

Theorem

It is decidable if a word u is member of a regular language L.

Proof.

1. Represent L in the form of a DFA M.
2. Check if u is accepted by M. □

Practical difficulty: state-space explosion

The conversion to DFA might require an exponential number of states. (E.g. when L is given as NFA or regular expression.)

Solution

On-the-fly generation of DFA prevents state-space explosion. We only generate those states visited when reading u.

Decidability of Emptyness

Theorem

It is decidable whether a regular language L is empty.

Proof.

- Construct a DFA (or NFA) M with $L(M) = L$.
- Check if M has a path from starting state to a final state.
- If **yes**, then $L \neq \emptyset$. If **no**, then $L = \emptyset$. □

Decidability of Subsets

Theorem

It is decidable for regular languages L_1 and L_2 if $L_1 \subseteq L_2$.

Proof.

We have

$$L_1 \subseteq L_2 \quad \Longleftrightarrow \quad L_1 \setminus L_2 = \emptyset$$

The language $L_1 \setminus L_2$ is regular.

Finally, emptyness is decidable. □

Decidability of Equivalence

Theorem

It is decidable if two regular languages L_1 and L_2 are equal.

Proof.

We have

$$L_1 = L_2 \quad \Longleftrightarrow \quad (L_1 \subseteq L_2) \wedge (L_2 \subseteq L_1)$$

Both problems on the right are decidable. $\square$

Word (String) Matching (Thompson, 1968)

The input:

- a word u,
- a regular expression r,

Question: Does u contain a subword in $L(r)$?

The following algorithm answers this question.

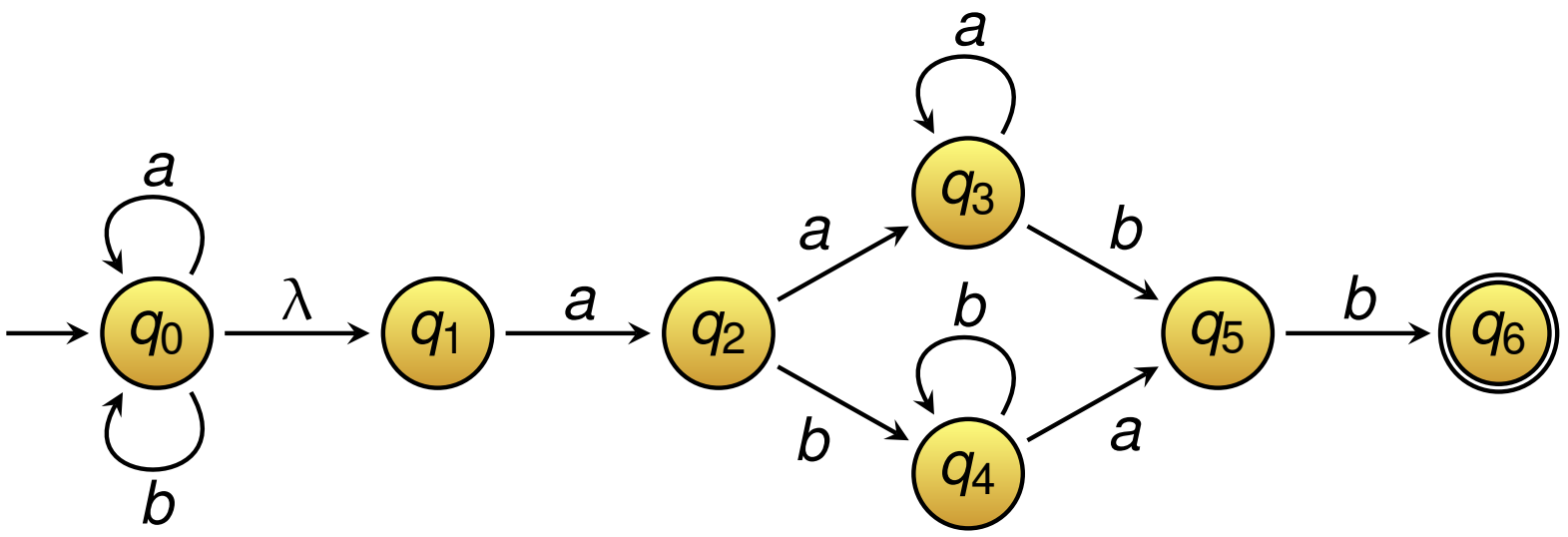
1. Transform the regular expression $\Sigma^* \cdot r$ into an NFA.
2. Compute '**on-the-fly**' path of u in the corresponding DFA.
3. Terminate as soon as a final state is reached.

The algorithm is used for example in grep in Unix.

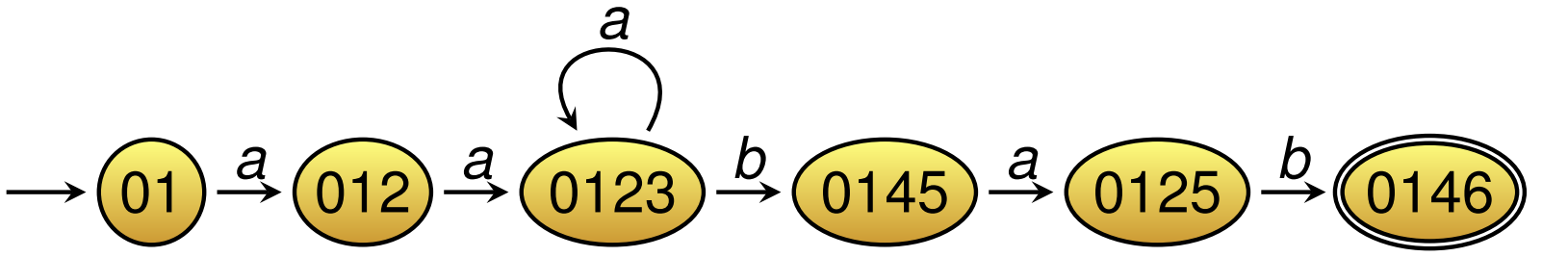
Worst-case time complexity: $O(|r| \cdot |u|)$

Word Matching Example

Regular expression $r = a(aa^*b + bb^*a)b$ gives rise to the NFA:



We match r with $u = aaababbb$.



q_0 q_0 $q_0\,q_1$ q_2 q_4 q_5 q_6

Word (String) Matching (Thompson, 1968)

The standard regular expression libraries of

- Java,
- Perl,
- PHP,
- Python

do not use the efficient algorithm from the last slide.

They use a **backtracking** algorithm with worst-case complexity

exponential time (in $|u|$)

Ken Thompson

Matching algorithm has been developed by **Ken Thompson**.

Won the Turing award in 1983 together with **Dennis Ritchie** for the operating system Unix. Dennis Ritchie (1941-2011) has also invented the programming language C.

Minimal DFA's (Hopcroft, 1971)

Construction

Given a DFA $M = (Q, \Sigma, \delta, q_0, F)$.

We construct the (unique) **minimal DFA** $\widehat{M}$ with $L(M) = L(\widehat{M})$. (Here minimal is with respect to the number of states.)

Step 1: Remove all unreachable states from M.

Step 2: Partition Q in indistinguishable states.

Step 3: Read off the minimal DFA.

Step 1: Remove all unreachable states from M.

That is, remove all states q for which there exists no path from the starting state q_0 to q.

Minimal DFA's (2)

States $q_1, q_2 \in Q$ are **distinguishable** if there exists $w \in \Sigma^*$ s.t.

$$q_1 \xrightarrow{w} q_1' \in F \qquad\qquad q_2 \xrightarrow{w} q_2' \notin F \ ,$$

or vice versa.

Step 2: Partition Q in indistinguishable states.

We construct the partition stepwise:

- Initial partitioning is $Q \setminus F$, F.
- If there are R and S in the partition of Q such that for some $a \in \Sigma$ and $q, q' \in R$:

  $$\delta(q, a) \in S \qquad \text{and} \qquad \delta(q', a) \notin S,$$

  then we split R in

  $$\{\, q \in R \mid \delta(q, a) \in S \,\} \qquad \{\, q \in R \mid \delta(q, a) \notin S \,\}$$

We keep splitting until no more split is possible.

Minimal DFA's (3)

Step 3: Read off the minimal DFA.

Let $Q_1, \ldots, Q_n$ be the final partition of Q.

These are the states of the minimal DFA $\widehat{M}$.

The transitions (arrows) of $\widehat{M}$ are:

$$Q_i \xrightarrow{a} Q_j \quad \Longleftrightarrow \quad \delta(q, a) \in Q_j \text{ for some } q \in Q_i$$

The starting state is the set that contains q_0.

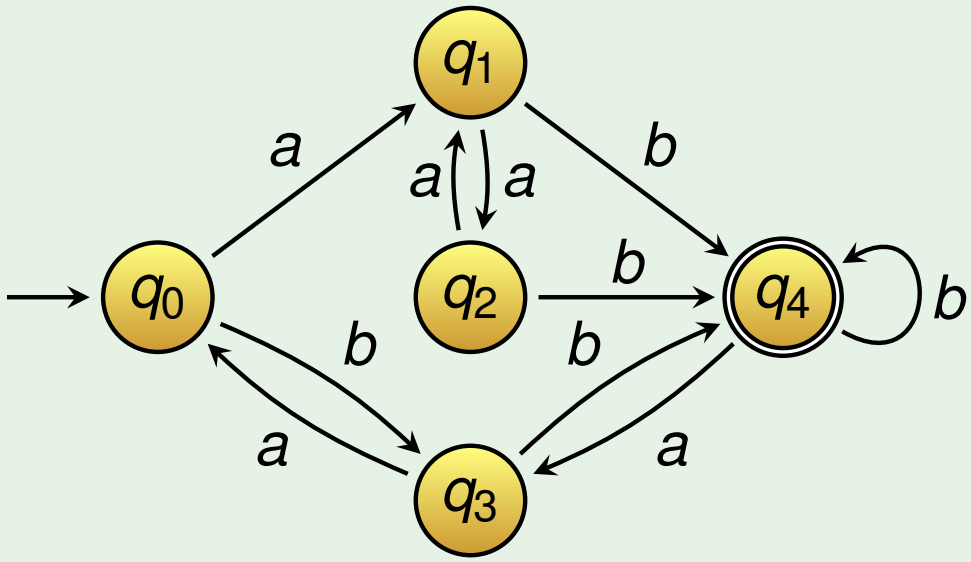
The final states are the subsets of F.

Worst-case time complexity: $O(|\Sigma| \cdot |Q|^2)$, since

- There are maximal $|Q| - 1$ splits.
- Every split costs maximal $O(|\Sigma| \cdot |Q|)$.

Exercise

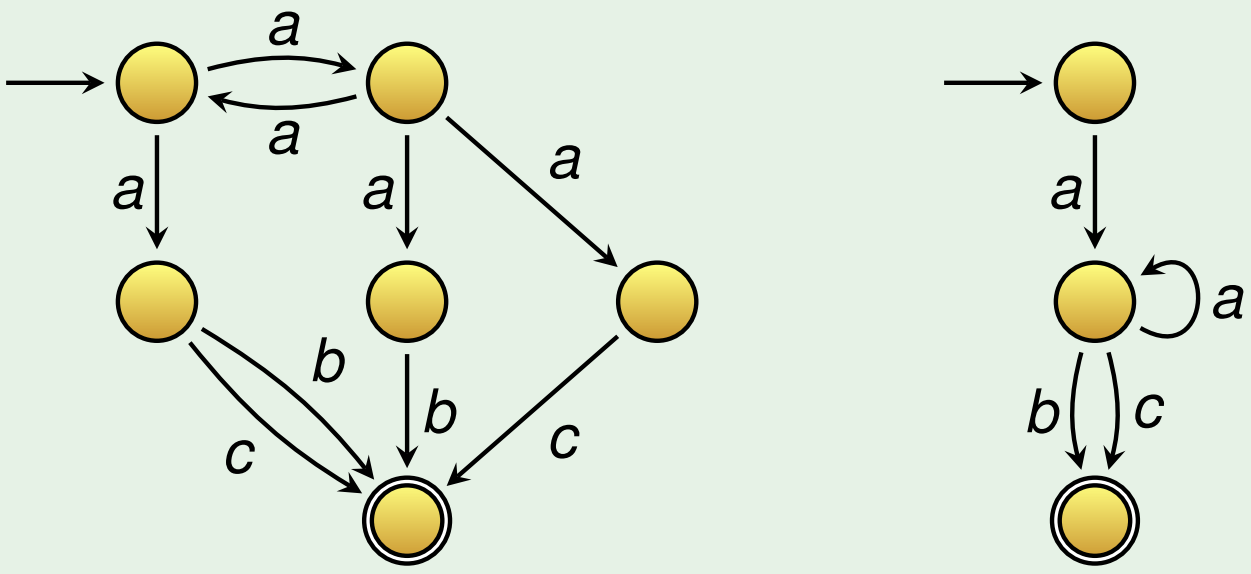
Minimise the following DFA



Minimising of NFA's

Minimising of NFA's is very difficult.

Example



Theorem

Minimising of NFA's is PSpace-complete.

The definition of PSpace-complete follows later.

Lexical Analysis

Lexical analysis converts a sequence of **characters** into a sequence of **tokens**.

Programs that do lexical analysis are **lexers** or **tokenizers**.

For example the expression

```
sum = 15 + 2;
```

could be converted to the sequence of tokens

token	token category
`sum`	identifier
`=`	assignment
`15`	integer literal
`+`	operator
`2`	integer literal

Allows to write parsers on the more abstract level of tokens.

Lexical Analysis

How do we get from characters to tokens?

- Regular expressions $r_1, \ldots, r_n$ express the pattern.
- **Every regular expression corresponds to a token.**
- Lexical analysis repeatedly searches the **longest prefix** of the input that is matched by one of the regular expressions. This prefix is transformed into a token.

  When no prefix matches, the result is an error message.

  When there are multiple longest prefixes, one is chosen.

For **improved performance**:

- Regular expression $r_1 + \cdots + r_n$ is translated to an NFA.
- This NFA is transformed to a **minimal DFA**.

Example of Lexical Analysis

Parser/lexer generators like

- JavaCC
- LEX

generate the lexer automatically. Thereby regular expressions or grammars are converted to minimal DFA's.

Example

Fortran describes whole numbers by the BNF grammar:

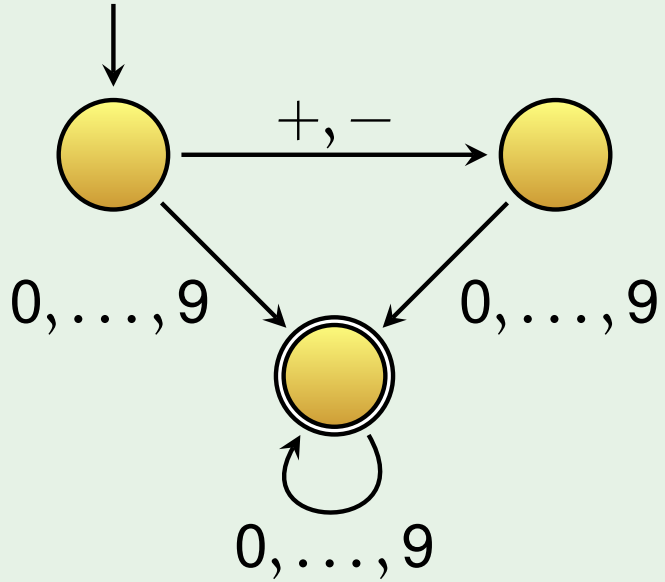
$$
\begin{array}{lcl}
\langle \text{digit} \rangle & ::= & 0 \mid 1 \mid \cdots \mid 9 \\
\langle \text{integer} \rangle & ::= & \langle \text{digit} \rangle^{+} \\
\langle \text{sign} \rangle & ::= & + \mid - \mid \lambda \\
\langle \text{signed-integer} \rangle & ::= & \langle \text{sign} \rangle \langle \text{integer} \rangle
\end{array}
$$

Note that this grammar gives rise to a regular language.

Example of Lexical Analysis (2)

Fortran Example Continued

The corresponding minimal DFA is



As always the longest matching prefix is taken,

`+52`

is **not** interpreted as single token `+5` and `2`, but as one token.

Non-Regular Languages

Theorem

$L = \{ a^n b^n \mid n \geq 0 \}$ is **not** regular.

Proof.

For a contradiction, assume that L was regular.

Then there exists a DFA $M = (Q, \{a, b\}, \delta, q_0, F)$ with $L(M) = L$.

Because Q is finite, we have

$$q_0 \xrightarrow{a^k} q \qquad\qquad q_0 \xrightarrow{a^\ell} q$$

for some $k < \ell$ and $q \in Q$. Then for some $q' \in Q$

$$q_0 \xrightarrow{a^k b^k} q' \qquad\qquad q_0 \xrightarrow{a^\ell b^k} q'$$

We have $a^k b^k \in L$; so $q' \in F$. However, $a^\ell b^k \notin L$; so $q' \notin F$.

Contradiction, thus L is not regular. □

We generalise the idea of the proof...

Pumping Lemma for Regular Languages (1959)

Pumping Lemma

Let L be a regular language. There exists $m > 0$ such that every $w \in L$ with $|w| \geq m$ can be written in the form

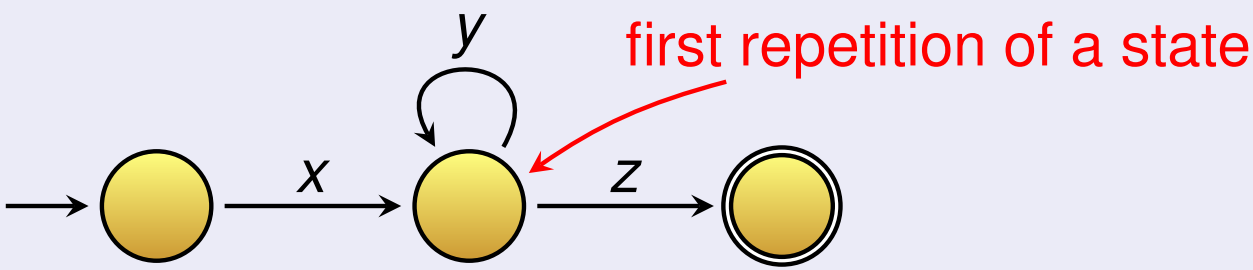
$$w = xyz$$

with $|xy| \leq m$ and $|y| \geq 1$, and $xy^i z \in L$ for every $i \geq 0$.

Proof.

We have $L = L(M)$ for some DFA M with m states.

When M reads $w \in L$ with $|w| \geq m$, there must be a cycle



with $|xy| \leq m$ and $|y| \geq 1$. Then $xy^i z \in L$ for every $i \geq 0$. □

Pumping Lemma Example

The **pumping lemma** can be used to prove that a language is **not regular**.

Assume that $L = \{ w \in \{a, b\}^* \mid w = w^R \}$ is regular.

By the pumping lemma there exists $m > 0$ such that

$$a^m b a^m = xyz$$

with $|xy| \leq m$, $|y| \geq 1$, and $xy^i z \in L$ for every $i \geq 0$.

Since $|xy| \leq m$ and $|y| \geq 1$, it follows that

$$x = a^j \qquad \text{and} \qquad y = a^k$$

with $j \geq 0$ and $k \geq 1$.

However $xyyz = a^{m+k} b a^m \notin L$. Contradiction!

Thus L is not regular. □

Using the Pumping Lemma

Attention

A contradiction of the pumping lemma for specific values of m, or of x, y, z, is not sufficient!

Pumping lemma as formula (note the quantifiers):

$$\begin{array}{l} \exists m > 0. \\ \quad \forall w \in L \text{ with } |w| \geq m. \\ \quad\quad \exists x, y, z \text{ with } w = xyz,\ |xy| \leq m,\ |y| \geq 1. \\ \quad\quad\quad \forall i \geq 0.\, xy^i z \in L \end{array}$$

To **contradict the pumping lemma**, we prove the negation:

$$\begin{array}{l} \forall m > 0. \\ \quad \exists w \in L \text{ with } |w| \geq m. \\ \quad\quad \forall x, y, z \text{ with } w = xyz,\ |xy| \leq m,\ |y| \geq 1. \\ \quad\quad\quad \exists i \geq 0.\, xy^i z \notin L \end{array}$$

Pumping Lemma as a Game

To **contradict the pumping lemma**, we prove the negation:

$$\forall m > 0.$$
$$\exists w \in L \text{ with } |w| \geq m.$$
$$\forall x, y, z \text{ with } w = xyz,\ |xy| \leq m,\ |y| \geq 1.$$
$$\exists i \geq 0.\ xy^i z \notin L$$

Pumping Lemma as a Game

Given is a language L. We want to prove that L is not regular.

1. Opponent picks m.
2. We choose a word $w \in L$ with $|w| \geq m$.
3. Opponent picks x, y, z with $w = xyz$, $|xy| \leq m$ and $|y| \geq 1$.
4. If we can find $i \geq 0$ such that $xy^i z \notin L$, then **we win**.

If we can always win, then L does not fulfil the pumping lemma!

Who wins the game when L is finite?

Exercise

Use the pumping lemma to show that

$$\{ a^n b^n \mid n \geq 0 \}$$

is not regular.

Is $\{ a^{2^k} \mid k \geq 0 \}$ regular?

Summary

- If L_1, L_2, L are regular, then

  $$L_1 \cup L_2\ , \quad L_1 \cap L_2\ , \quad L_1 L_2\ , \quad \overline{L}\ , \quad L_1 \setminus L_2\ , \quad L^*\ , \quad L^R$$

  are also regular languages.
- **Word (string) matching** (on-the-fly construction)
- **Minimal DFA's** (Hopcroft's algorithm)
- **Pumping lemma** for proving that a language is not regular.

Looking Forward

Read:

- Linz 2.4, 4.1–4.3

Do the following exercises:

- Linz 4.1: 17, 20, 26 (‘regular’ must be “right linear”, twice)
- Linz 4.2: 5, 12
- Linz 2.4: 1, 4, 6
- Linz 4.3: 1, 3, 4ef

Following lecture:

- Context-free languages:
  - Definition
  - Transformations and normal forms
  - Parsing algorithms