DFA to Regular Expressions, Language Transformations

Lecture 7

September 17, 2019

Regular Languages, DFAs, NFAs

Theorem

Languages accepted by DFAs, NFAs, and regular expressions are the same.

Regular Languages, DFAs, NFAs

Theorem

Languages accepted by DFAs, NFAs, and regular expressions are the same.

- DFAs are special cases of NFAs (trivial)
- NFAs accept regular expressions (we saw already, relative easy)
- DFAs accept languages accepted by NFAs (subset construction)
- Regular expressions for languages accepted by DFAs (sketch today, again later in the course)

Part I

DFA to Regular Expressions

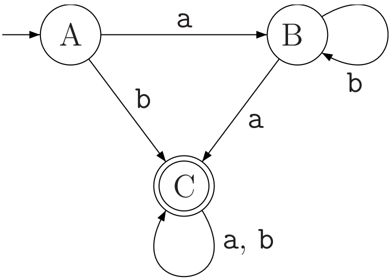
DFA to Regular Expressions

Theorem

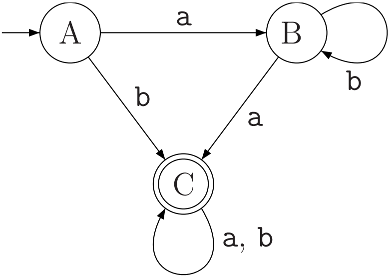
Given a DFA $M = (Q, \Sigma, \delta, s, A)$ *there is a regular expression* r *such that* $L(r) = L(M)$. *That is, regular expressions are as powerful as DFAs (and hence also NFAs).*

- Simple algorithm but formal proof is technical. See notes.
- A different algorithm with an easier formal proof later in the course.

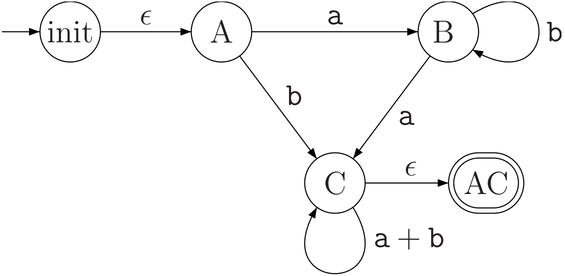
Stage 0: Input



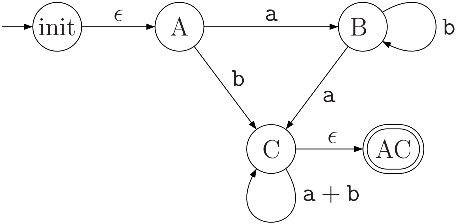
Stage 1: Normalizing

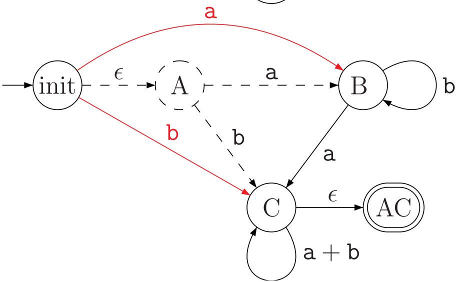


2: Normalizing it.

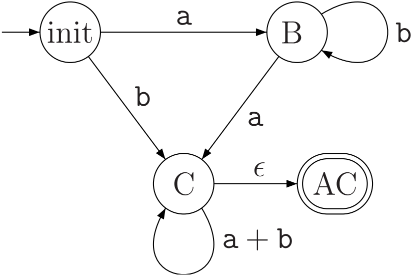


Stage 2: Remove state A

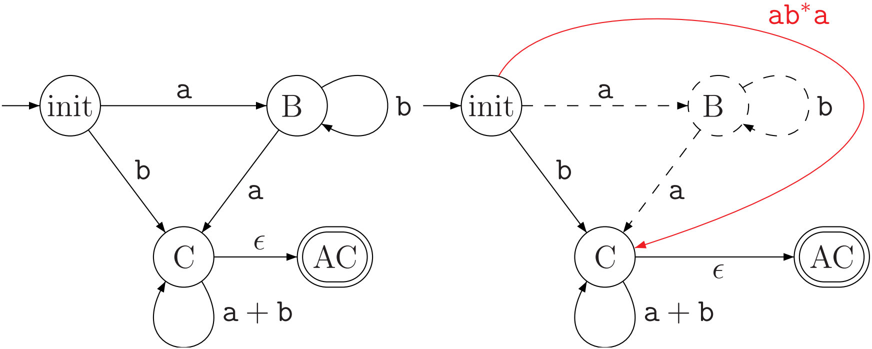




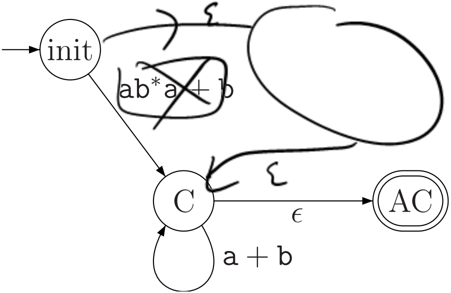
Stage 4: Redrawn without old edges



Stage 4: Removing B



Stage 5: Redraw



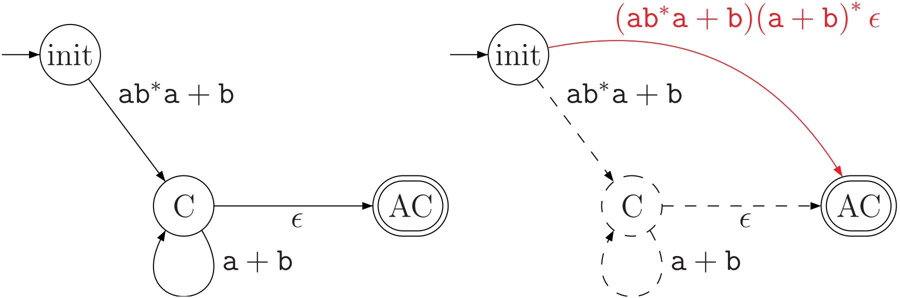
abbba

w

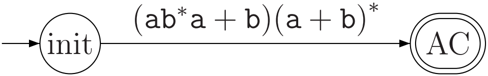
$$s \xrightarrow{\substack{w_1 \\ a_1}} q_1 \xrightarrow{\substack{w_2 \\ a_2}} q_2 \longrightarrow \cdots \xrightarrow{\substack{w_k \\ a_k}} q_f$$

$$w = a_1 a_2 \cdots a_k \qquad q_f \in A$$

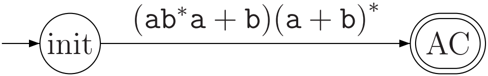
Stage 6: Removing C



Stage 7: Redraw

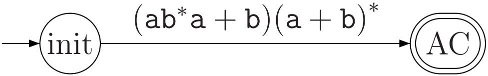


Stage 8: Extract regular expression



Thus, the automata is equivalent to the regular expression $(ab^*a + b)(a + b)^*$.

Stage 8: Extract regular expression



Thus, the automata is equivalent to the regular expression $(ab^*a + b)(a + b)^*$.

- States can be eliminated in any order
- Can start with NFA

Part II

Closure Properties and Language Transformations

Closure propeties

Definition

(Informal) A set A is **closed** under an operation **op** if applying **op** to any elements of A results in an element that also belongs to A.

Closure propeties

Definition

(Informal) A set A is **closed** under an operation **op** if applying **op** to any elements of A results in an element that also belongs to A.

Examples:

- *Integers:* closed under $+$, $-$, $*$, but not division.
- *Positive integers:* closed under $+$ but not under $-$
- *Regular languages:* closed under union, intersection, Kleene star, complement, difference, homomorphism, inverse homomorphism, reverse, $\ldots$

Closure properties of Regular Languages

Question: How do we prove that regular languages are closed under some new operation?

Closure properties of Regular Languages

Question: How do we prove that regular languages are closed under some new operation?

Three broad approaches

- Use existing closure properties

Closure properties of Regular Languages

Question: How do we prove that regular languages are closed under some new operation?

Three broad approaches

- Use existing closure properties
  - L_1, L_2, L_3, L_4 regular implies $(L_1 - L_2) \cap (\bar{L_3} \cup L_4)^*$ is regular

Closure properties of Regular Languages

Question: How do we prove that regular languages are closed under some new operation?

Three broad approaches

- Use existing closure properties
  - L_1, L_2, L_3, L_4 regular implies $(L_1 - L_2) \cap (\bar{L_3} \cup L_4)^*$ is regular
- Transform regular expressions

Closure properties of Regular Languages

Question: How do we prove that regular languages are closed under some new operation?

Three broad approaches

- Use existing closure properties
  - L_1, L_2, L_3, L_4 regular implies $(L_1 - L_2) \cap (\bar{L_3} \cup L_4)^*$ is regular
- Transform regular expressions
- Transform DFAs to NFAs — versatile technique and shows the power of nondeterminism

Example: REVERSE

Given string w, w^R is reverse of w.
For a language L define $L^R = \{w^R \mid w \in L\}$ as reverse of L.

Example: REVERSE

Given string w, w^R is reverse of w.
For a language L define $L^R = \{w^R \mid w \in L\}$ as reverse of L.

Theorem

L^R is regular if L is regular.

Example: REVERSE

Given string w, w^R is reverse of w.
For a language L define $L^R = \{w^R \mid w \in L\}$ as reverse of L.

Theorem

L^R is regular if L is regular.

Infinitely many regular languages!

Example: REVERSE

Given string w, w^R is reverse of w.
For a language L define $L^R = \{w^R \mid w \in L\}$ as reverse of L.

Theorem

L^R is regular if L is regular.

Infinitely many regular languages!

Proof technique:

- take some finite representation of L such as regular expression r
- Describe an algorithm A that takes r as input and outputs a regular expression r' such that $L(r') = (L(r))^R$.
- Come up with A and prove its correctness.

REVERSE via regular expressions

Suppose r is a regular expression for L. How do we create a regular expression r' for L^R?

$(010 + 101)^* 110$

$100(11 + 010)$

$(11 + 010)001$

100

$100 + 0100$

$(100)^* \quad (001)^*$

REVERSE via regular expressions

Suppose r is a regular expression for L. How do we create a regular expression r' for L^R? Inductively based on recursive definition of r.

- $r = \emptyset$ or $r = a$ for some $a \in \Sigma$
- $r = r_1 + r_2$
- $r = r_1 \cdot r_2$
- $r = (r_1)^*$

REVERSE via regular expressions

- $r = \emptyset$ or $r = a$ for some $a \in \Sigma$
  $r' = r$

REVERSE via regular expressions

- $r = \emptyset$ or $r = a$ for some $a \in \Sigma$
  $r' = r$
- $r = r_1 + r_2$. If r_1', r_2' are reg expressions for $(L(r_1))^R, (L(r_2))^R$ then $r' =$

REVERSE via regular expressions

- $r = \emptyset$ or $r = a$ for some $a \in \Sigma$
  $r' = r$
- $r = r_1 + r_2$. If r'_1, r'_2 are reg expressions for $(L(r_1))^R, (L(r_2))^R$ then $r' = r'_1 + r'_2$

REVERSE via regular expressions

- $r = \emptyset$ or $r = a$ for some $a \in \Sigma$
  $r' = r$
- $r = r_1 + r_2$. If r_1', r_2' are reg expressions for $(L(r_1))^R, (L(r_2))^R$ then $r' = r_1' + r_2'$
- $r = r_1 \cdot r_2$. If r_1', r_2' are reg expressions for $(L(r_1))^R, (L(r_2))^R$ then $r' =$

REVERSE via regular expressions

- $r = \emptyset$ or $r = a$ for some $a \in \Sigma$
  $r' = r$
- $r = r_1 + r_2$. If r_1', r_2' are reg expressions for $(L(r_1))^R, (L(r_2))^R$ then $r' = r_1' + r_2'$
- $r = r_1 \cdot r_2$. If r_1', r_2' are reg expressions for $(L(r_1))^R, (L(r_2))^R$ then $r' = r_2' \cdot r_1'$

If u, v are strings
then $(uv)^R = v^R u^R$

REVERSE via regular expressions

- $r = \emptyset$ or $r = a$ for some $a \in \Sigma$
  $r' = r$
- $r = r_1 + r_2$. If r_1', r_2' are reg expressions for $(L(r_1))^R, (L(r_2))^R$ then $r' = r_1' + r_2'$
- $r = r_1 \cdot r_2$. If r_1', r_2' are reg expressions for $(L(r_1))^R, (L(r_2))^R$ then $r' = r_2' \cdot r_1'$
- $r = (r_1)^*$. If r_1' is reg expressions for $(L(r_1))^R$ then $r' = (r_1')^*$

$$r_1^* = (\epsilon + r_1 + r_1 r_1 + r_1 r_1 r_1 + \cdots)$$
$$\epsilon + r_1' + r_1' r_1' + r_1' r_1' r_1' + \cdots$$
$$= (r_1')^*$$

REVERSE via regular expressions

- $r = \emptyset$ or $r = a$ for some $a \in \Sigma$
  $r' = r$
- $r = r_1 + r_2$. If r_1', r_2' are reg expressions for $(L(r_1))^R, (L(r_2))^R$ then $r' = r_1' + r_2'$
- $r = r_1 \cdot r_2$. If r_1', r_2' are reg expressions for $(L(r_1))^R, (L(r_2))^R$ then $r' = r_2' \cdot r_1'$
- $r = (r_1)^*$. If r_1' is reg expressions for $(L(r_1))^R$ then $r' = (r_1')^*$

$r = (0 + 10)^*(001 + 01)1$ then $r' =$

REVERSE via regular expressions

- $r = \emptyset$ or $r = a$ for some $a \in \Sigma$
  $r' = r$
- $r = r_1 + r_2$. If r_1', r_2' are reg expressions for $(L(r_1))^R, (L(r_2))^R$ then $r' = r_1' + r_2'$
- $r = r_1 \cdot r_2$. If r_1', r_2' are reg expressions for $(L(r_1))^R, (L(r_2))^R$ then $r' = r_2' \cdot r_1'$
- $r = (r_1)^*$. If r_1' is reg expressions for $(L(r_1))^R$ then $r' = (r_1')^*$

$r = (0 + 10)^*(001 + 01)1$ then $r' = 1(100 + 10)(0 + 01)^*$

REVERSE via regular expressions

- $r = \emptyset$ or $r = a$ for some $a \in \Sigma$
  $r' = r$
- $r = r_1 + r_2$. If r'_1, r'_2 are reg expressions for $(L(r_1))^R, (L(r_2))^R$ then $r' = r'_1 + r'_2$
- $r = r_1 \cdot r_2$. If r'_1, r'_2 are reg expressions for $(L(r_1))^R, (L(r_2))^R$ then $r' = r'_2 \cdot r'_1$
- $r = (r_1)^*$. If r'_1 is reg expressions for $(L(r_1))^R$ then $r' = (r'_1)^*$

$r = (0 + 10)^*(001 + 01)1$ then $r' = 1(100 + 10)(0 + 01)^*$

Proof for each identity: tedious case analysis based on definitions of union, concatenation, Kleene star and reverse.

REVERSE via machine transformation

Given DFA $M = (Q, \Sigma, \delta, s, A)$ want NFA N such that $L(N) = (L(M))^R$.

N should accept w^R **iff** M accepts w

M accepts w iff $\delta^*_M(s, w) \in A$

Idea: N reverses transitions of M and starts at a final state of M.

REVERSE via machine transformation

Given DFA $M = (Q, \Sigma, \delta, s, A)$ want NFA N such that $L(N) = (L(M))^R$.

N should accept w^R **iff** M accepts w

M accepts w iff $\delta^*_M(s, w) \in A$

Idea: N reverses transitions of M and starts at a final state of M. Which one?

REVERSE via machine transformation

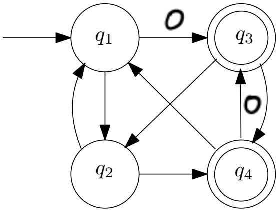
Given DFA $M = (Q, \Sigma, \delta, s, A)$ want NFA N such that $L(N) = (L(M))^R$.

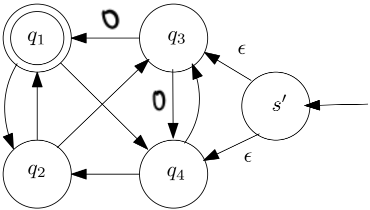
N should accept w^R **iff** M accepts w

M accepts w iff $\delta^*_M(s, w) \in A$

Idea: N reverses transitions of M and starts at a final state of M. Which one? Non-deterministically guesses and accepts if it reaches s.

REVERSE via machine transformation





Caveat: Reversing transitions may create an NFA.

REVERSE via machine transformation

Proof (DFA to NFA): Let $M = (\Sigma, Q, s, A, \delta)$ be an arbitrary DFA that accepts L. We construct an NFA $M^R = (\Sigma, Q^R, s^R, A^R, \delta^R)$ with ε-transitions that accepts L^R, intuitively by reversing every transition in M, and swapping the roles of the start state and the accepting states. Because M does not have a unique accepting state, we need to introduce a special start state s^R, with ε-transitions to each accepting state in M. These are the only ε-transitions in M^R.

$$
\begin{aligned}
Q^R &= Q \cup \{s^R\} \\
A^R &= \{s\} \\
\delta^R(s^R, \varepsilon) &= A \\
\delta^R(s^R, a) &= \varnothing && \text{for all } a \in \Sigma \\
\delta^R(q, \varepsilon) &= \varnothing && \text{for all } q \in Q \\
\delta^R(q, a) &= \{p \mid q \in \delta(p, a)\} && \text{for all } q \in Q \text{ and } a \in \Sigma
\end{aligned}
$$

Routine inductive definition-chasing now implies that the reversal of any sequence $q_0 \to q_1 \to \cdots \to q_\ell$ of transitions in M is a valid sequence $q_\ell \to q_{\ell-1} \to \cdots \to q_0$ of transitions in M^R. Because the transitions retain their labels (but reverse directions), it follows that M accepts any string w if and only if M^R accepts w^R.

We conclude that the NFA M^R accepts L^R, so L^R must be regular. $\square$

A more complicated example: CYCLE

$CYCLE(L) = \{yx \mid x, y \in \Sigma^*, xy \in L\}$

Theorem

$CYCLE(L)$ *is regular if* L *is regular.*

A more complicated example: CYCLE

$$CYCLE(L) = \{yx \mid x, y \in \Sigma^*, xy \in L\}$$

Theorem

$CYCLE(L)$ is regular if L is regular.

Example: $L = \{abc, 374a\}$

$CYCLE(L) = \{abc, bca, cab, a374, 4a37, 74a3, 374a\}$

A more complicated example: CYCLE

$$CYCLE(L) = \{yx \mid x, y \in \Sigma^*, xy \in L\}$$

Theorem

$CYCLE(L)$ is regular if L is regular.

A more complicated example: CYCLE

$CYCLE(L) = \{yx \mid x, y \in \Sigma^*, xy \in L\}$

Theorem

$CYCLE(L)$ *is regular if* L *is regular.*

Given DFA M for L create NFA N that accepts $CYCLE(L)$.

- N is a finite state machine, cannot know split of w into xy and yet has to simulate M on x and y.
- Exploit fact that M is itself a finite state machine. N only needs to "know" the state $\delta_M^*(s, x)$ and there are only finite number of states in M

Construction for CYCLE

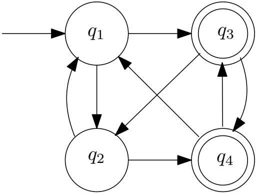
Let $w = xy$ and $w' = yx$.

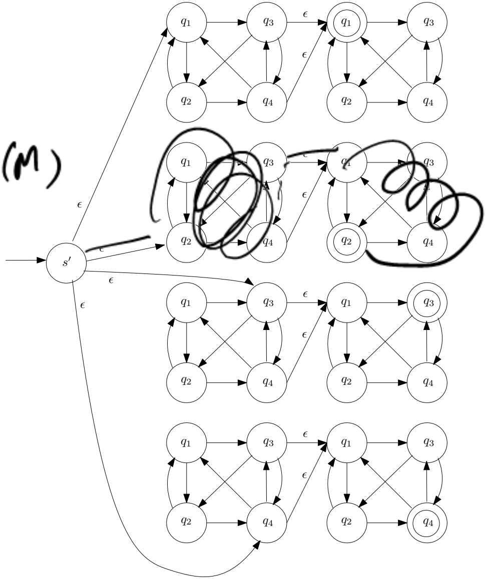
- N *guesses* state $q = \delta^*_M(s, x)$ and simulates M on w' with start state q.
- N *guesses* when y ends (at that point M must be in an accept state) and transitions to a copy of M to simulate M on remaining part of w' (which is x)
- N accepts w' if after second copy of M on x it ends up in the guessed state q

Construction for CYCLE

$w = xy$

$yx \in L(M)$





Proving correctness

Exercise: Write down formal description of N in tuple notation starting with $M = (Q, \Sigma, \delta, s, A)$.

Need to argue that $L(N) = CYCLE(L(M))$

- If $w = xy$ accepted by M then argue that yx is accepted by N
- If N accepts w' then argue that $w' = yx$ such that xy accepted by M.

Application of closure properties to non-regularity

$L_1 = \{0^n 1^n \mid n \geq 0\}$
$L_2 = \{w \in \{0,1\}^* \mid \#_0(w) = \#_1(w)\}$
$L_3 = \{0^i 1^j \mid i \neq j\}$

Application of closure properties to non-regularity

$L_1 = \{0^n 1^n \mid n \geq 0\}$
$L_2 = \{w \in \{0, 1\}^* \mid \#_0(w) = \#_1(w)\}$
$L_3 = \{0^i 1^j \mid i \neq j\}$

L_1 is not regular. Can we use that fact to prove L_2 and L_2 are not regular without going through the fooling set argument?

Application of closure properties to non-regularity

$L_1 = \{0^n 1^n \mid n \geq 0\}$
$L_2 = \{w \in \{0,1\}^* \mid \#_0(w) = \#_1(w)\}$
$L_3 = \{0^i 1^j \mid i \neq j\}$

L_1 is not regular. Can we use that fact to prove L_2 and L_2 are not regular without going through the fooling set argument?

$L_1 = L_2 \cap 0^* 1^*$ hence if L_2 is regular then L_1 is regular, a contradiction.

Application of closure properties to non-regularity

$L_1 = \{0^n 1^n \mid n \geq 0\}$
$L_2 = \{w \in \{0,1\}^* \mid \#_0(w) = \#_1(w)\}$
$L_3 = \{0^i 1^j \mid i \neq j\}$

L_1 is not regular. Can we use that fact to prove L_2 and L_2 are not regular without going through the fooling set argument?

$L_1 = L_2 \cap 0^*1^*$ hence if L_2 is regular then L_1 is regular, a contradiction.

$L_1 = \bar{L_3} \cap 0^*1^*$ hence if L_3 is regular then L_1 is regular, a contradiction

Jeff's reminder about exam

Following topics *not* on the upcoming midterm exam

- Transforming DFA/NFA into regular expressions (covered today)
- Minimizing DFA