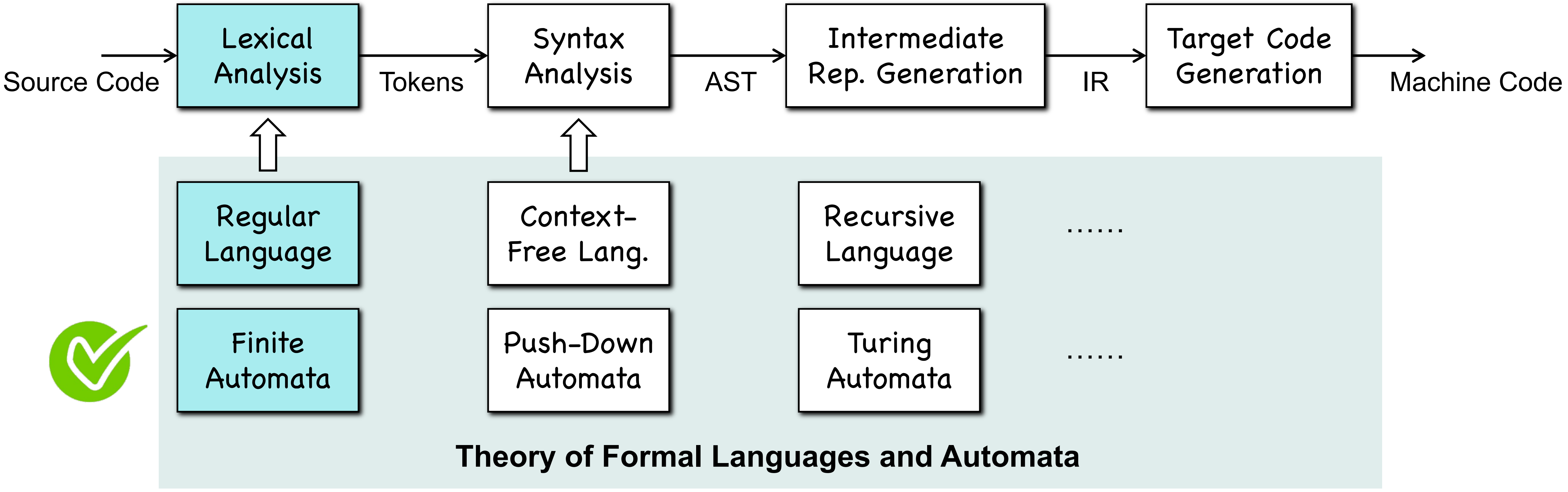
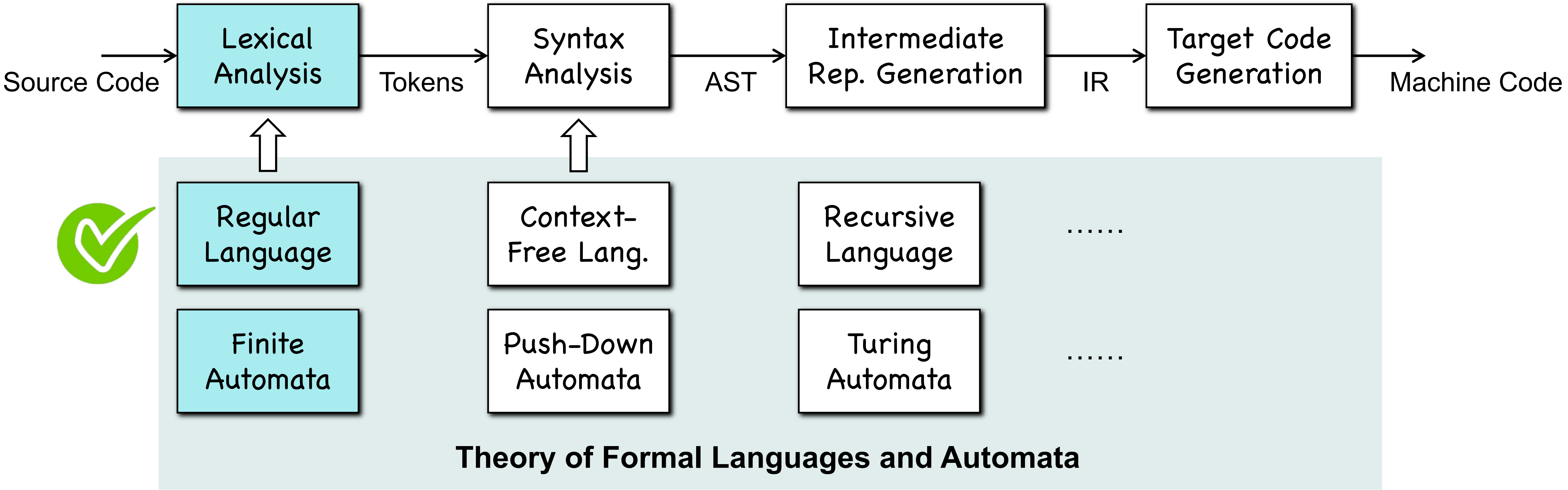
Made by Qingkai Shi
qingkaishi@nju.edu.cn

Chapter 3-2
Lexical Analysis

Lexical Analysis



Lexical Analysis



PART I: Regular Language

Regular Language

- Take a DFA $M = (Q, \Sigma, \delta, q_0, F)$, language can be accepted by the DFA is written as $L(M) = \{w \in \Sigma^*: \delta^*(q_0, w) \subseteq F\}$

Regular Language

- Take a DFA $M = (Q, \Sigma, \delta, q_0, F)$, language can be accepted by the DFA is written as $L(M) = \{w \in \Sigma^*: \delta^*(q_0, w) \subseteq F\}$

- A language L is regular if there is a DFA M such that $L = L(M)$

Regular Language

- Take a DFA $M = (Q, \Sigma, \delta, q_0, F)$, language can be accepted by the DFA is written as $L(M) = \{w \in \Sigma^*: \delta^*(q_0, w) \subseteq F\}$

- A language L is regular if there is a DFA M such that $L = L(M)$

- **Examples**: $\{abba\}$ $\{\varepsilon, ab, abba\}$ $\{a^n b : n \geq 0\}$

  { all strings with prefix ab }

  { all strings with suffix ab }

Regular Language

- Take a DFA $M = (Q, \Sigma, \delta, q_0, F)$, language can be accepted by the DFA is written as $L(M) = \{w \in \Sigma^*: \delta^*(q_0, w) \subseteq F\}$

- Take a NFA $M = (Q, \Sigma \cup \{\epsilon\}, \delta, q_0, F)$, language can be accepted by the NFA is written as $L(M) = \{w \in \Sigma^*: \delta^*(q_0, w) \cap F \neq \emptyset\}$

Regular Language

- Take a DFA $M = (Q, \Sigma, \delta, q_0, F)$, language can be accepted by the DFA is written as $L(M) = \{w \in \Sigma^*: \delta^*(q_0, w) \subseteq F\}$

- Take a NFA $M = (Q, \Sigma \cup \{\epsilon\}, \delta, q_0, F)$, language can be accepted by the NFA is written as $L(M) = \{w \in \Sigma^*: \delta^*(q_0, w) \cap F \neq \emptyset\}$

L is a regular language if there's a DFA/NFA M such that $L = L(M)$

Closure Properties

- Given two regular languages, L_1 and L_2, the following are also regular languages:
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^R
  - L_1^*
  - $\overline{L_1}$
  - $L_1 \cap L_2$

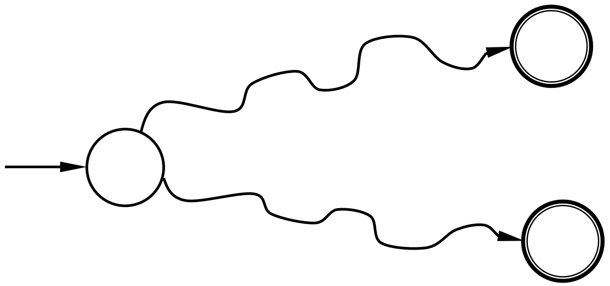
Closure Properties

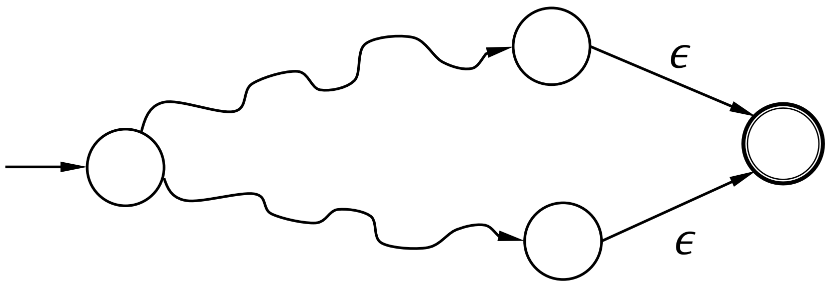
- Given two regular languages, L_1 and L_2, the following are also regular languages:
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^R
  - L_1^*
  - $\overline{L_1}$
  - $L_1 \cap L_2$

Why?

Closure Properties

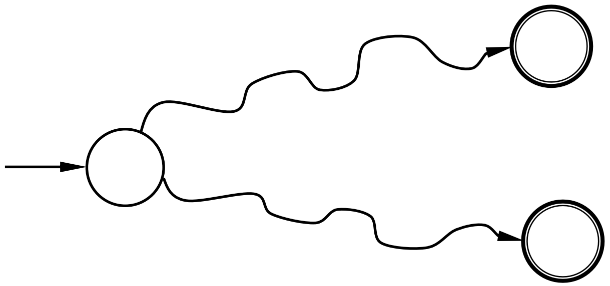
- Any NFA can be converted into an NFA with a single final state

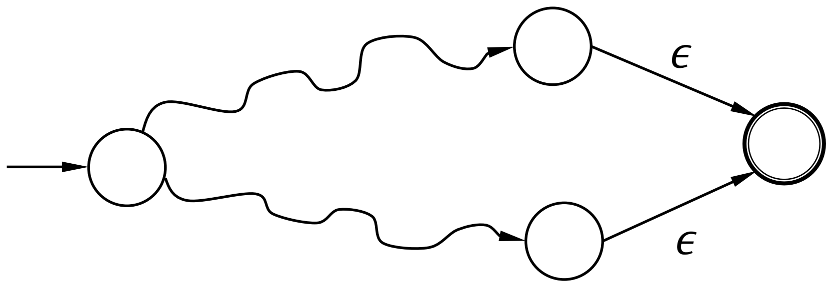




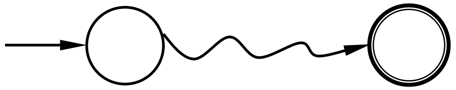
Closure Properties

- Any NFA can be converted into an NFA with a single final state



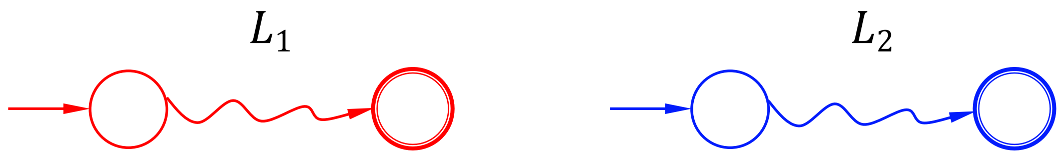


- Every regular language has an equivalent and single-exit NFA



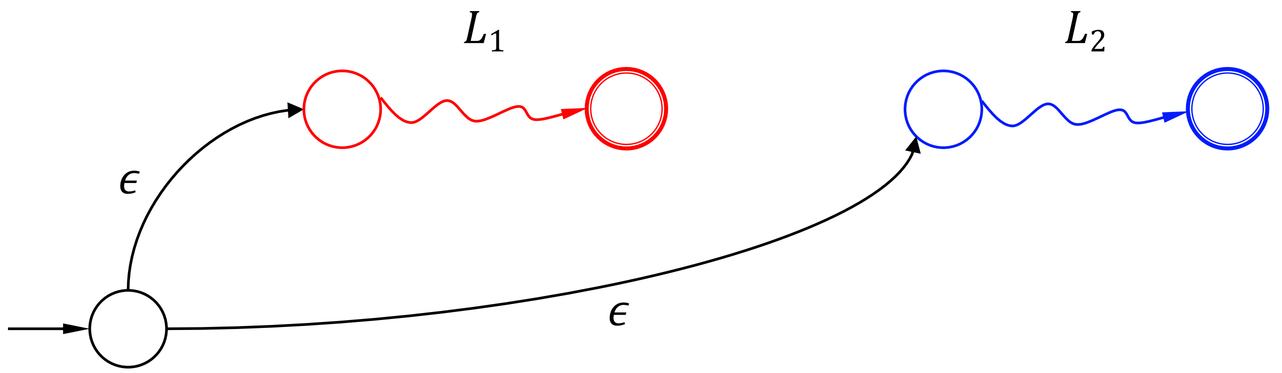
Closure Properties

- Given two regular languages, L_1 and L_2, the following are also regular languages:
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^R
  - L_1^*
  - $\overline{L_1}$
  - $L_1 \cap L_2$



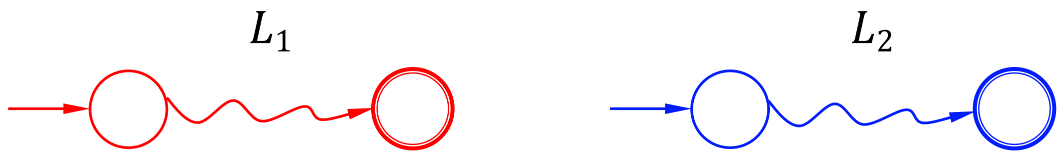
Closure Properties

- Given two regular languages, L_1 and L_2, the following are also regular languages:
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^R
  - L_1^*
  - $\overline{L_1}$
  - $L_1 \cap L_2$



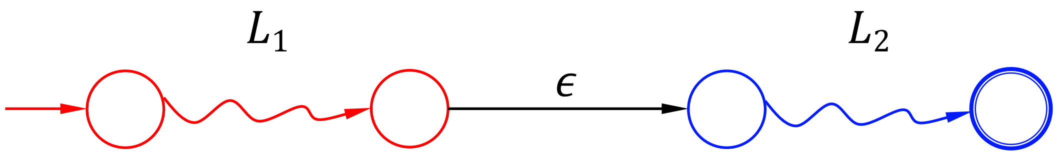
Closure Properties

- Given two regular languages, L_1 and L_2, the following are also regular languages:
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^R
  - L_1^*
  - $\overline{L_1}$
  - $L_1 \cap L_2$



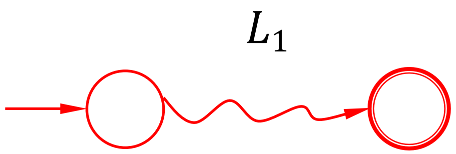
Closure Properties

- Given two regular languages, L_1 and L_2, the following are also regular languages:
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^R
  - L_1^*
  - $\overline{L_1}$
  - $L_1 \cap L_2$



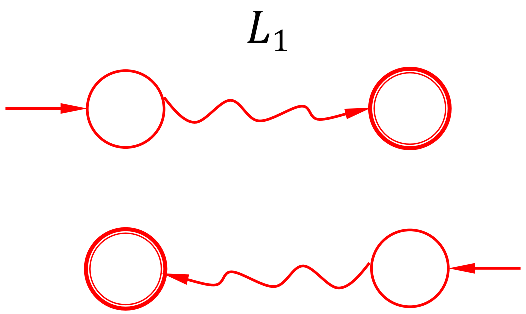
Closure Properties

- Given two regular languages, L_1 and L_2, the following are also regular languages:
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^R
  - L_1^*
  - $\overline{L_1}$
  - $L_1 \cap L_2$



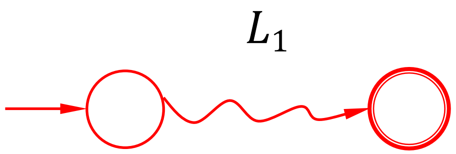
Closure Properties

- Given two regular languages, L_1 and L_2, the following are also regular languages:
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^R
  - L_1^*
  - $\overline{L_1}$
  - $L_1 \cap L_2$



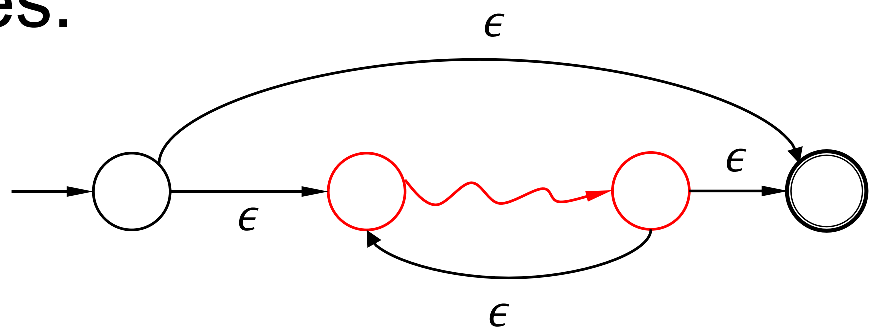
Closure Properties

- Given two regular languages, L_1 and L_2, the following are also regular languages:
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^R
  - L_1^*
  - $\overline{L_1}$
  - $L_1 \cap L_2$



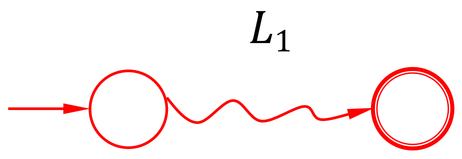
Closure Properties

- Given two regular languages, L_1 and L_2, the following are also regular languages:
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^R
  - L_1^*
  - $\overline{L_1}$
  - $L_1 \cap L_2$



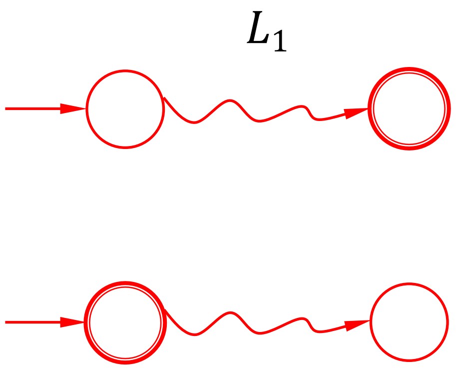
Closure Properties

- Given two regular languages, L_1 and L_2, the following are also regular languages:
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^R
  - L_1^*
  - $\overline{L_1}$
  - $L_1 \cap L_2$



Closure Properties

- Given two regular languages, L_1 and L_2, the following are also regular languages:
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^R
  - L_1^*
  - $\overline{L_1}$
  - $L_1 \cap L_2$



Closure Properties

- Given two regular languages, L_1 and L_2, the following are also regular languages:
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^R
  - L_1^*
  - $\overline{L_1}$
  - $L_1 \cap L_2$

Have a Try!

Closure Properties

- Given two regular languages, L_1 and L_2, the following are also regular languages:
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^R
  - L_1^*
  - $\overline{L_1}$
  - $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

Closure Properties

- Given two regular languages, L_1 and L_2, the following are also regular languages:
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^R
  - L_1^*
  - $\overline{L_1}$
  - $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

Regular language is closed under union, intersection, reversal, complement, concatenation, and Kleene star.

PART II: Regular Expression

Regular Expression (Regex)

- Regex describes a language

Note: We have not proved that it is a regular language!

Regular Expression (Regex)

- Regex describes a language
- Primitive regex
  - $\emptyset, \epsilon, a$

Regular Expression (Regex)

- Regex describes a language
- Primitive regex
  - $\emptyset, \epsilon, a$
- Given two regex: r_1, r_2, the following are regex
  - $r_1|r_2$
  - $r_1 r_2$
  - r_1^*
  - (r_1)

Regular Expression (Regex)

- Regex describes a language
- Primitive regex
  - $\emptyset, \epsilon, a$
- Given two regex: r_1, r_2, the following are regex
  - $r_1 | r_2$
  - $r_1 r_2$
  - r_1^*
  - (r_1)
- **Example:** $(a|b)^* c$

Language by Regex

- The language represented by regex are defined below
- Primitive regex
  - $L(\emptyset) = \emptyset;\ \ L(\epsilon) = \{\epsilon\};\ \ L(a) = \{a\}$

Language by Regex

- The language represented by regex are defined below
- Primitive regex
  - $L(\emptyset) = \emptyset$; $L(\epsilon) = \{\epsilon\}$; $L(a) = \{a\}$
- Given two regex: r_1, r_2, the following are regex
  - $L(r_1|r_2) = L(r_1) \cup L(r_2)$
  - $L(r_1 r_2) = L(r_1)L(r_2)$
  - $L(r_1^*) = (L(r_1))^*$
  - $L((r_1)) = L(r_1)$

Language by Regex

- The language represented by regex are defined below
- Primitive regex
  - $L(\emptyset) = \emptyset$; $L(\epsilon) = \{\epsilon\}$; $L(a) = \{a\}$
- Given two regex: r_1, r_2, the following are regex
  - $L(r_1|r_2) = L(r_1) \cup L(r_2)$
  - $L(r_1 r_2) = L(r_1)L(r_2)$
  - $L(r_1^*) = (L(r_1))^*$
  - $L((r_1)) = L(r_1)$
- Two regex are equivalent if they represent the same language

Laws of Regex

- Commutativity and Associativity
  - $r_1|r_2 = r_2|r_1$
  - $(r_1|r_2)|r_3 = r_1|(r_2|r_3)$
  - $(r_1 r_2) r_3 = r_1 (r_2 r_3)$

Laws of Regex

- Commutativity and Associativity
  - $r_1|r_2 = r_2|r_1$
  - $(r_1|r_2)|r_3 = r_1|(r_2|r_3)$
  - $(r_1r_2)r_3 = r_1(r_2r_3)$

- Identities and Annihilators
  - $r_1|\emptyset = \emptyset|r_1 = r_1$
  - $r_1\epsilon = \epsilon r_1 = r_1$
  - $r_1\emptyset = \emptyset r_1 = r_1$

Laws of Regex

- Distributive and Idempotent Law
  - $(r_1|r_2)r_3 = r_1r_3|r_2r_3$
  - $r_3(r_1|r_2) = r_3r_1|r_3r_2$
  - $r_1|r_1 = r_1$

Laws of Regex

- Distributive and Idempotent Law
  - $(r_1|r_2)r_3 = r_1r_3|r_2r_3$
  - $r_3(r_1|r_2) = r_3r_1|r_3r_2$
  - $r_1|r_1 = r_1$
- Closure
  - $(r_1^*)^* = r_1^*$
  - $\emptyset^* = \epsilon^* = \epsilon$
  - $r_1^+ = r_1(r_1^*) = (r_1^*)r_1$
  - $r_1? = \epsilon|r_1$

Regex = Regular Language

- (1) Any regex represents a **regular** language
- (2) Any regular language can be expressed by a regex

Regex = Regular Language

- (1) Any regex represents a **regular** language

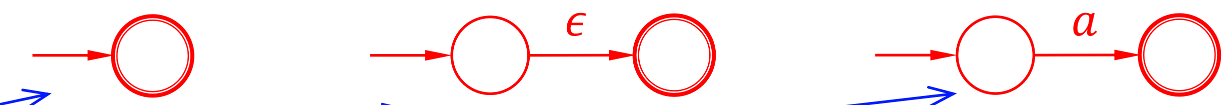
Regex to NFA (DFA)

- (1) Any regex represents a ~~regular language~~NFA/DFA

Regex to NFA (DFA)

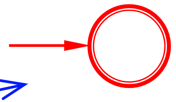
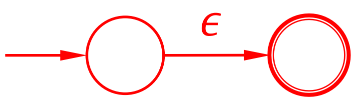
- Primitive regex
  - $L(\emptyset) = \emptyset;\ \ L(\epsilon) = \{\epsilon\};\ \ L(a) = \{a\}$

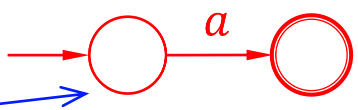
Regex to NFA (DFA)



- Primitive regex
  - $L(\emptyset) = \emptyset$; $L(\epsilon) = \{\epsilon\}$; $L(a) = \{a\}$

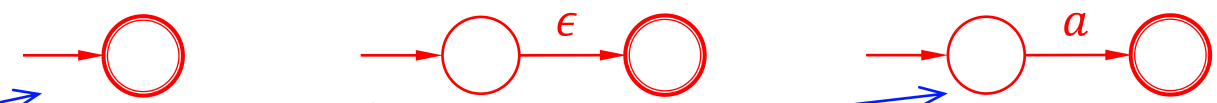
Regex to NFA (DFA)



- Primitive regex
  - $L(\emptyset) = \emptyset$; $L(\epsilon) = \{\epsilon\}$; $L(a) = \{a\}$
- Given two regex: r_1, r_2, the following are regex
  - $L(r_1|r_2) = L(r_1) \cup L(r_2)$
  - $L(r_1 r_2) = L(r_1)L(r_2)$
  - $L(r_1^*) = (L(r_1))^*$
  - $L((r_1)) = L(r_1)$

Regex to NFA (DFA)



- Primitive regex
  - $L(\emptyset) = \emptyset$; $L(\epsilon) = \{\epsilon\}$; $L(a) = \{a\}$
- Given two regex: r_1, r_2, the following are regex
  - $L(r_1|r_2) = L(r_1) \cup L(r_2)$
  - $L(r_1 r_2) = L(r_1)L(r_2)$
  - $L(r_1^*) = (L(r_1))^*$
  - $L((r_1)) = L(r_1)$

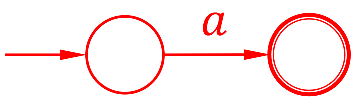
- *<u>Recall how we prove regular language closed under union, concatenation, and Kleene star</u>*

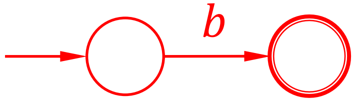
Example

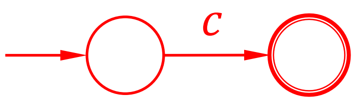
- Build the NFA for the regex:$(a|b)^*c$
- $L\big((a|b)^*c\big) = \big(L(a|b)\big)^* L(c) = \big(L(a) \cup L(b)\big)^* L(c)$

Example

- Build the NFA for the regex:$(a|b)^*c$
- $L\left((a|b)^*c\right) = \left(L(a|b)\right)^*L(c) = \left(L(a) \cup L(b)\right)^*L(c)$

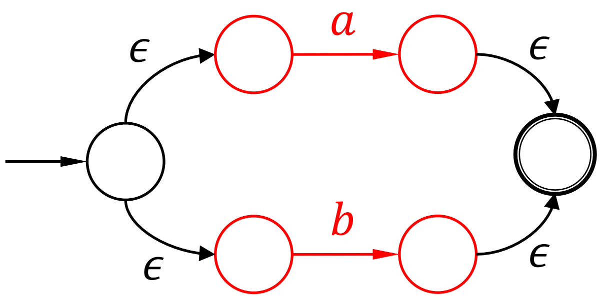


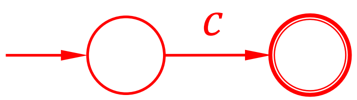




Example

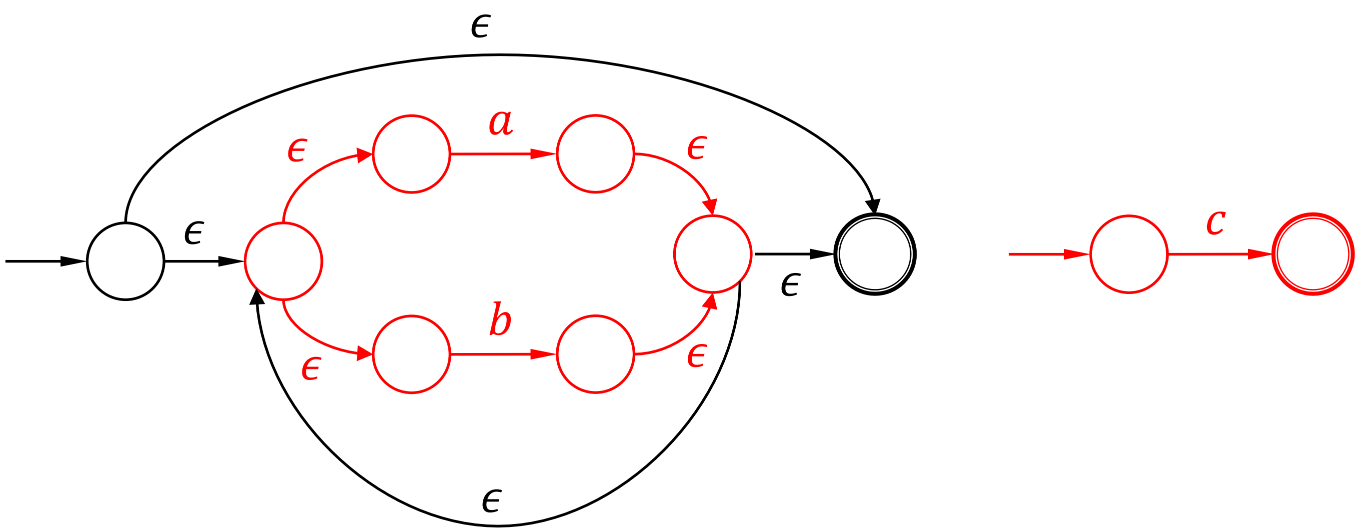
- Build the NFA for the regex:$(a|b)^*c$
- $L\big((a|b)^*c\big) = \big(L(a|b)\big)^* L(c) = \big(L(a) \cup L(b)\big)^* L(c)$





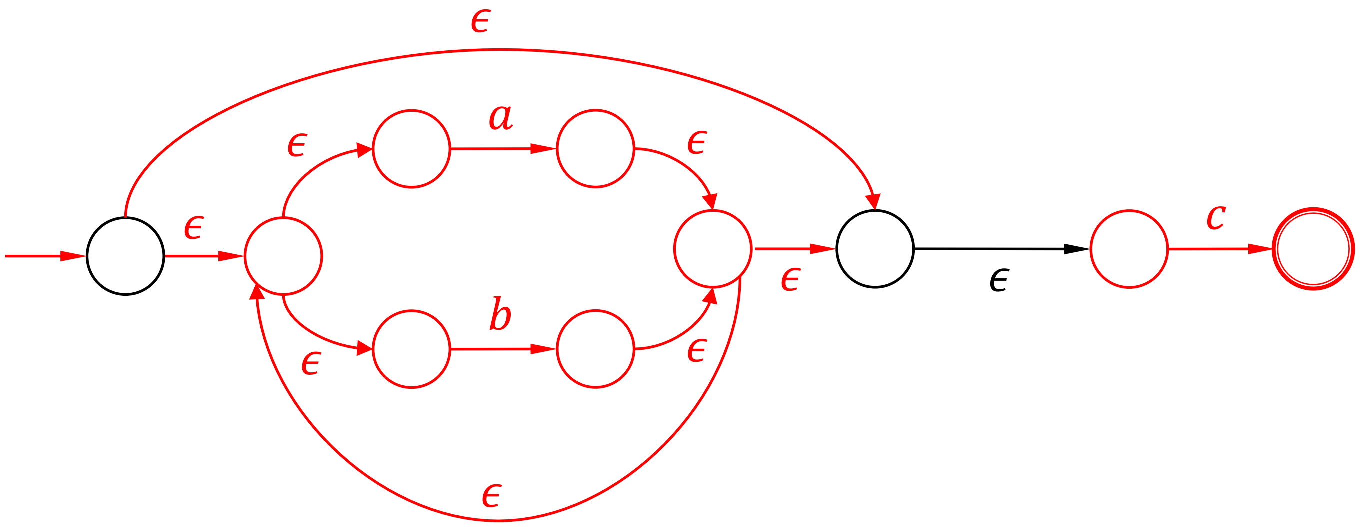
Example

- Build the NFA for the regex: $(a|b)^*c$
- $L\left((a|b)^*c\right) = \left(L(a|b)\right)^* L(c) = \left(L(a) \cup L(b)\right)^* L(c)$



Example

- Build the NFA for the regex:$(a|b)^*c$
- $L\big((a|b)^*c\big) = \big(L(a|b)\big)^* L(c) = \big(L(a) \cup L(b)\big)^* L(c)$



Equivalence of RL & Regex

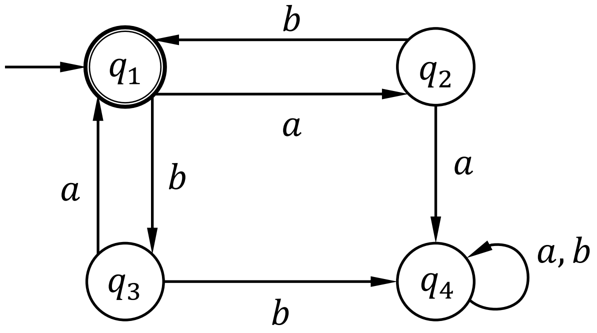
- (1) Any regex represents a regular language ✔
- (2) Any regular language can be expressed by a regex

DFA to Regex

- (2) Any ~~regular language~~DFA can be expressed by a regex

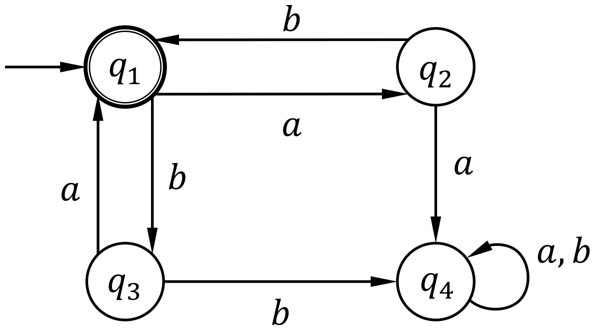
DFA to Regex

- (2) Any ~~regular language~~DFA can be expressed by a regex



DFA to Regex

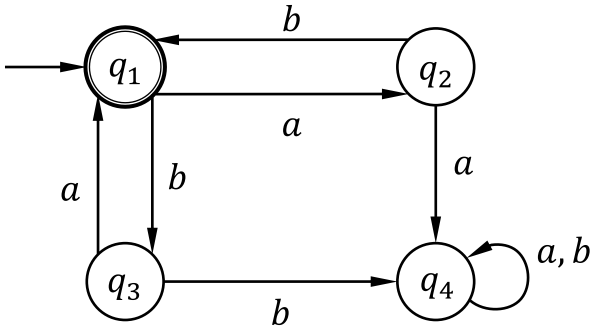
- (2) Any ~~regular language~~DFA can be expressed by a regex



$$q_1 = \epsilon \mid q_2 b \mid q_3 a$$

DFA to Regex

- (2) Any ~~regular language~~DFA can be expressed by a regex

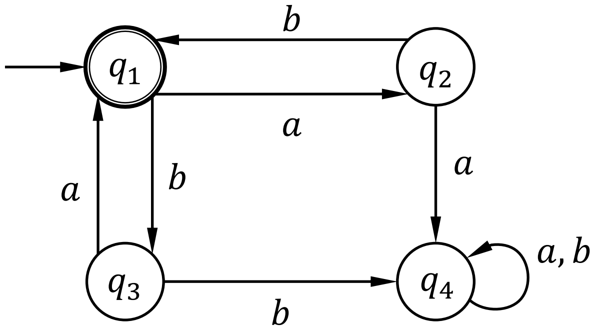


$q_1 = \epsilon \mid q_2 b \mid q_3 a$

$q_2 = q_1 a$

DFA to Regex

- (2) Any ~~regular language~~DFA can be expressed by a regex



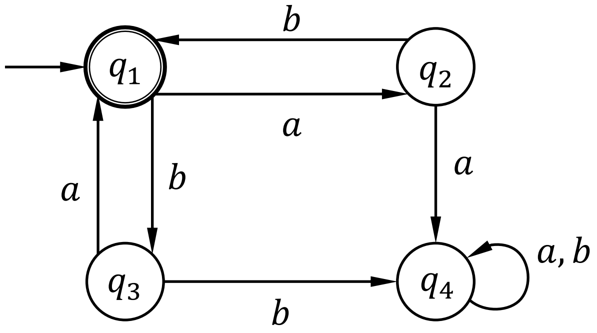
$q_1 = \epsilon \mid q_2 b \mid q_3 a$

$q_2 = q_1 a$

$q_3 = q_1 b$

DFA to Regex

- (2) Any ~~regular language~~DFA can be expressed by a regex



$$q_1 = \epsilon \mid q_2 b \mid q_3 a$$

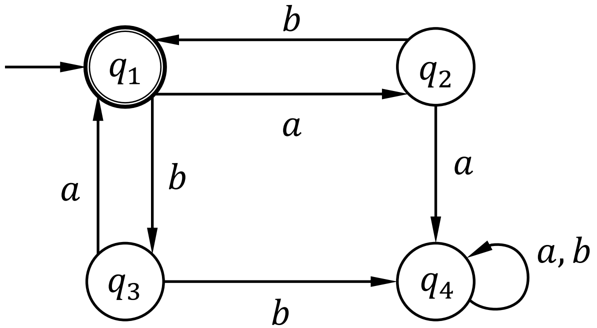
$$q_2 = q_1 a$$

$$q_3 = q_1 b$$

$$q_4 = q_2 a \mid q_3 b \mid q_4 a \mid q_4 b$$

DFA to Regex

- (2) Any ~~regular language~~DFA can be expressed by a regex



$$q_1 = \epsilon \mid q_2 b \mid q_3 a$$

$$q_2 = q_1 a$$

$$q_3 = q_1 b$$

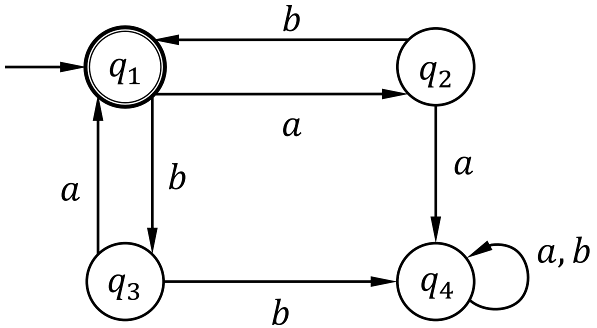
$$q_4 = q_2 a \mid q_3 b \mid q_4 a \mid q_4 b$$

Variable Elimination

$$q_1 = \epsilon \mid q_1 ab \mid q_1 ba$$

DFA to Regex

- (2) Any ~~regular language~~DFA can be expressed by a regex



$$q_1 = \epsilon \mid q_2 b \mid q_3 a$$

$$q_2 = q_1 a$$

$$q_3 = q_1 b$$

$$q_4 = q_2 a \mid q_3 b \mid q_4 a \mid q_4 b$$

Variable Elimination

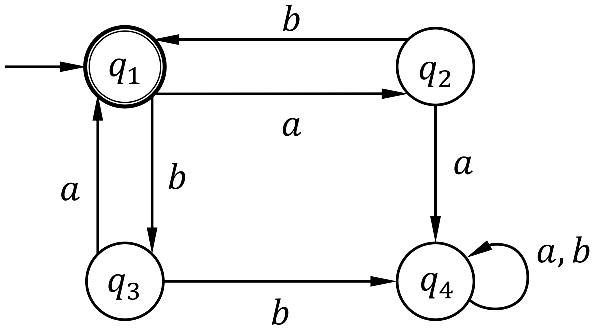
$$q_1 = \epsilon \mid q_1 ab \mid q_1 ba$$

Associativity

$$q_1 = \epsilon \mid q_1(ab|ba)$$

DFA to Regex

- (2) Any ~~regular language~~DFA can be expressed by a regex



$$q_1 = \epsilon \mid q_2 b \mid q_3 a$$

$$q_2 = q_1 a$$

$$q_3 = q_1 b$$

$$q_4 = q_2 a \mid q_3 b \mid q_4 a \mid q_4 b$$

Variable Elimination

$$q_1 = \epsilon \mid q_1 ab \mid q_1 ba$$

Associativity

$$q_1 = \epsilon \mid q_1 (ab|ba)$$

Arden's Theorem:
R = QP is the solution of R = Q | RP*

$$q_1 = \epsilon\ (ab|ba)* = (ab|ba)*$$

Equivalence of RL & Regex

- (1) Any regex represents a regular language
- (2) Any regular language can be expressed by a regex

Elementary Questions

- Given a regular language L by a regex, check if a string $w \in L$

Elementary Questions

- Given a regular language L by a regex, check if a string $w \in L$
- Build a DFA and check if the DFA accepts the string

Elementary Questions

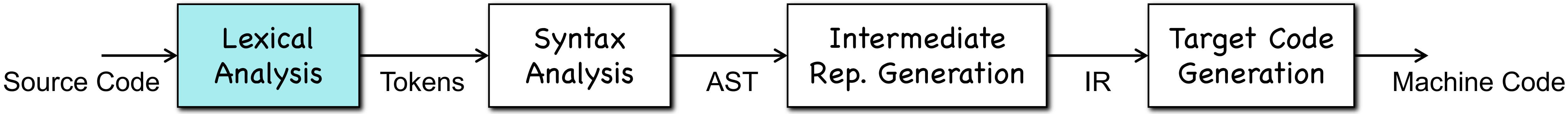
- Given a regular language L by a regex, check if a string $w \in L$
- Build a DFA and check if the DFA accepts the string
- This is the foundation of the compilers' lexical analysis

Elementary Questions

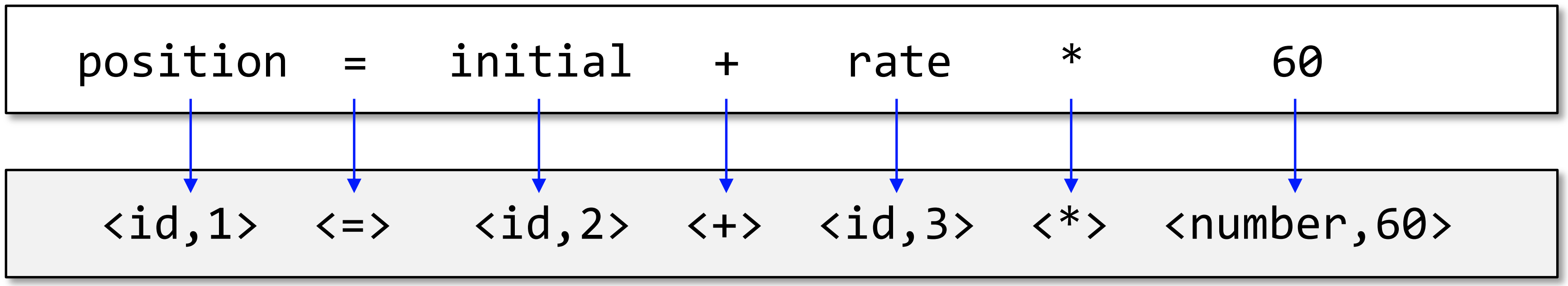
- Given a regular language L by a regex, check if a string $w \in L$
- Build a DFA and check if the DFA accepts the string
- This is the foundation of the compilers' lexical analysis
  - **Step 1**: Write regex for lexemes (e.g., numbers, variables), build DFAs
  - **Step 2**: Regard the source code as an input string
  - **Step 3**: Check the input string against the DFAs to recognize lexemes

PART III: Lexical Analysis

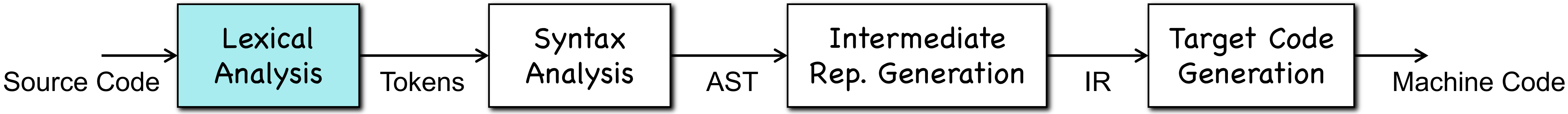
Lexical Analysis



- Find **lexemes** according to **patterns**, and create **tokens**
  - Lexeme – a character string
  - Pattern – regular expression (lexical errors if no patterns matched)
  - Token – <token-class-name, attribute>



Lexical Analysis



- Find **lexemes** according to **patterns**, and create **tokens**
  - Lexeme – a character string
  - Pattern – regular expression (lexical errors if no patterns matched)
  - Token – <token-class-name, attribute>

Regexp → NFA → DFA → min DFA

position	=	initial	+	rate	*	60
<id,1>	<=>	<id,2>	<+>	<id,3>	<*>	<number,60>

Lexical Analysis

Source Code → **Lexical Analysis** → Tokens → Syntax Analysis → AST → Intermediate Rep. Generation → IR → Target Code Generation → Machine Code

- Find **lexemes** according to **patterns**, and create **tokens**
  - Lexeme – a character string
  - Pattern – regular expression (lexical errors if no patterns matched)
    - Regexp → NFA → DFA → min DFA
  - Token – <token-class-name, attribute>

key	name	…
1	position	…
2	initial	…
…		…

symbol table

```
position  =  initial  +  rate  *  60
<id,1>  <=>  <id,2>  <+>  <id,3>  <*>  <number,60>
```

Lexical Analysis

- Input the source code (as an input string) into a lexical analyzer
- Check against the DFAs of keywords/operators/identifiers/…

Lexical Analysis

- Input the source code (as an input string) into a lexical analyzer
- Check against the DFAs of keywords/operators/identifiers/…

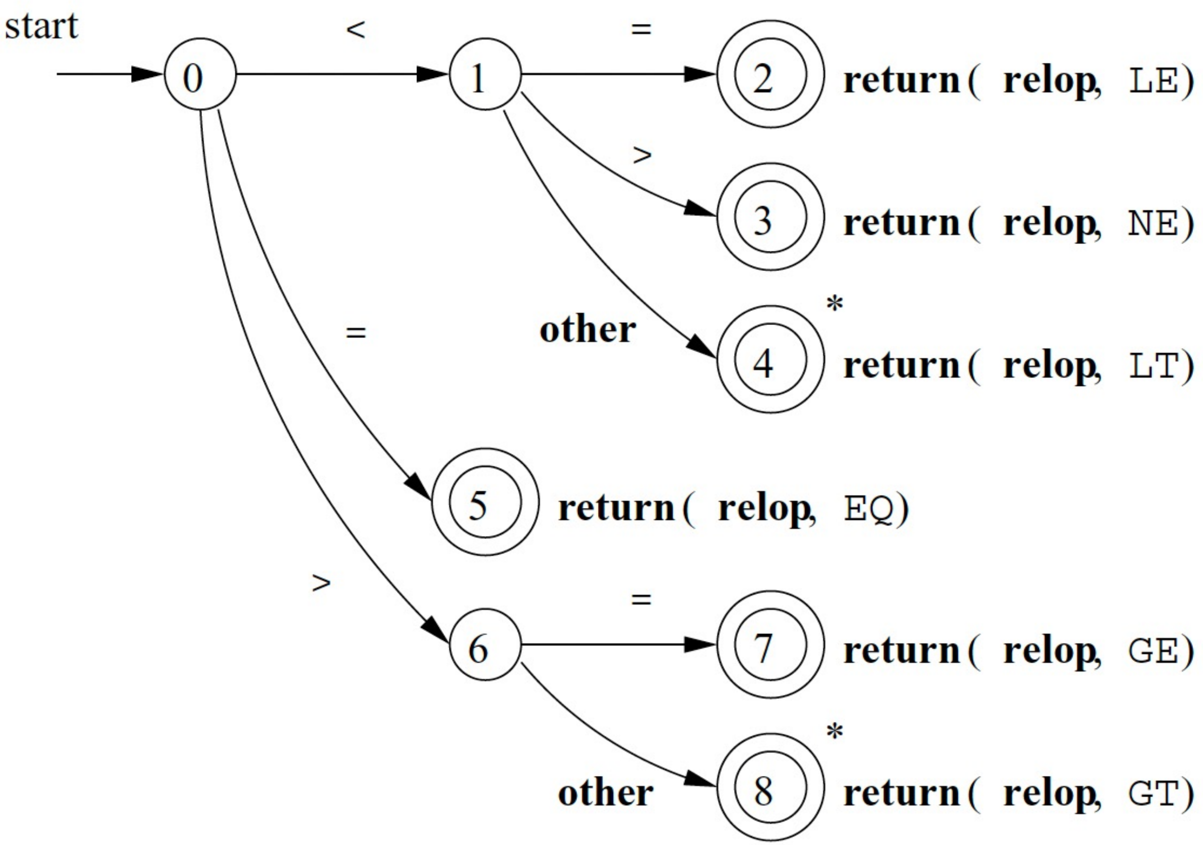
```
void check(w, M) {
    q = q0;
    while (true) {
        c = read(w); if (c == None) { print(q ∈ F ? “accept” : “reject”); }
        switch(q) {
        case q0: if (c == ‘a’) { q = q1; } break; // δ(q0,a)=q1; δ(q0,!a)=q0
        case q1: … break;
        case q2: … break;
        case ……
        }
    }}
```

Lexical Analysis

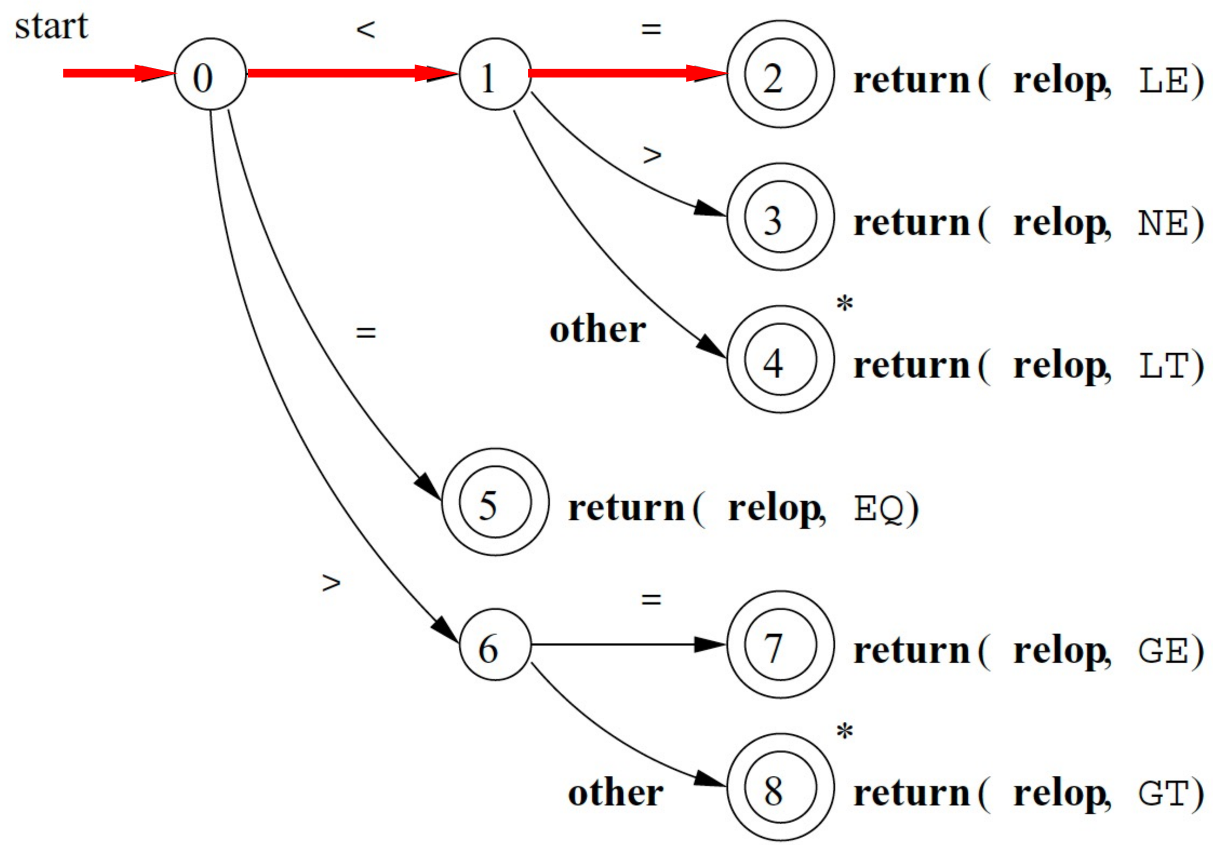
- Input the source code (as an input string) into a lexical analyzer
- Check against the DFAs of keywords/operators/identifiers/…
- Whenever reaching a final states of a DFA, create a token

Example: Relational Operator

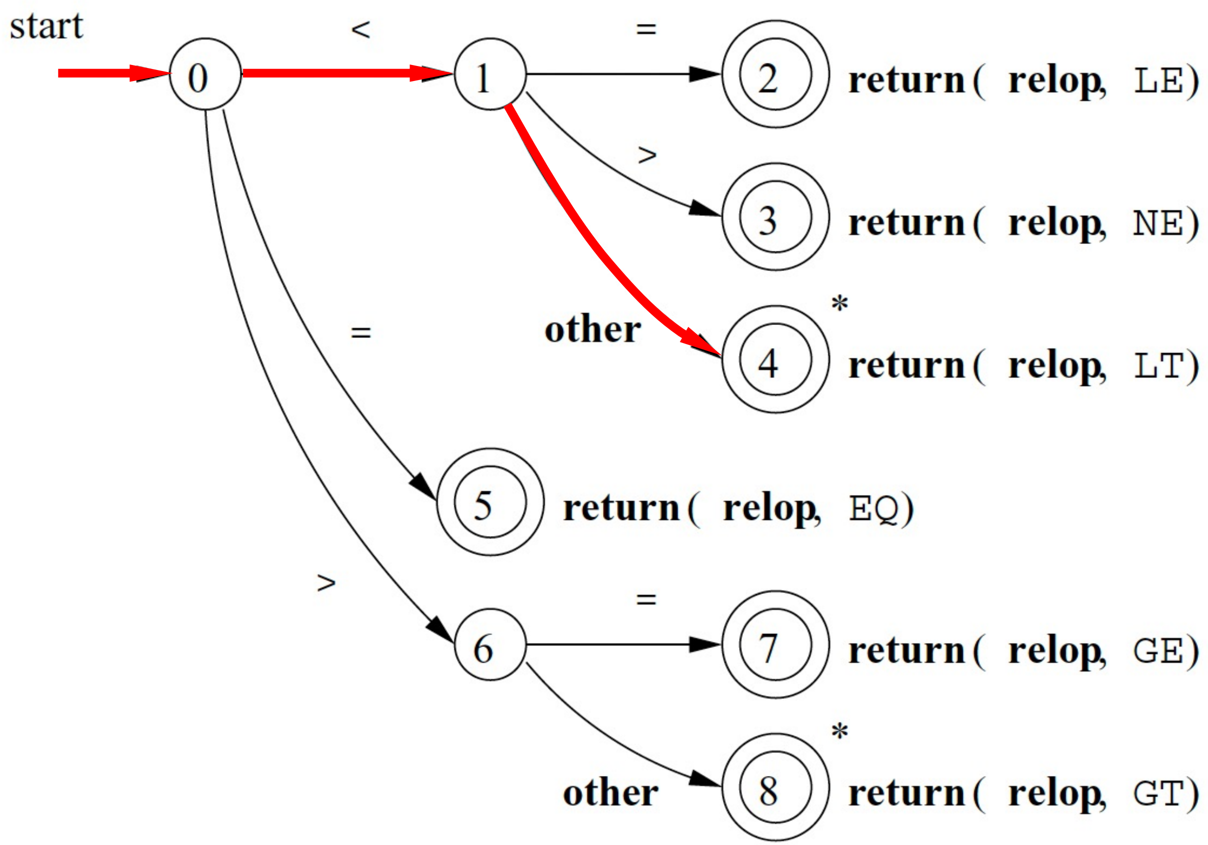
Example: Relational Operator



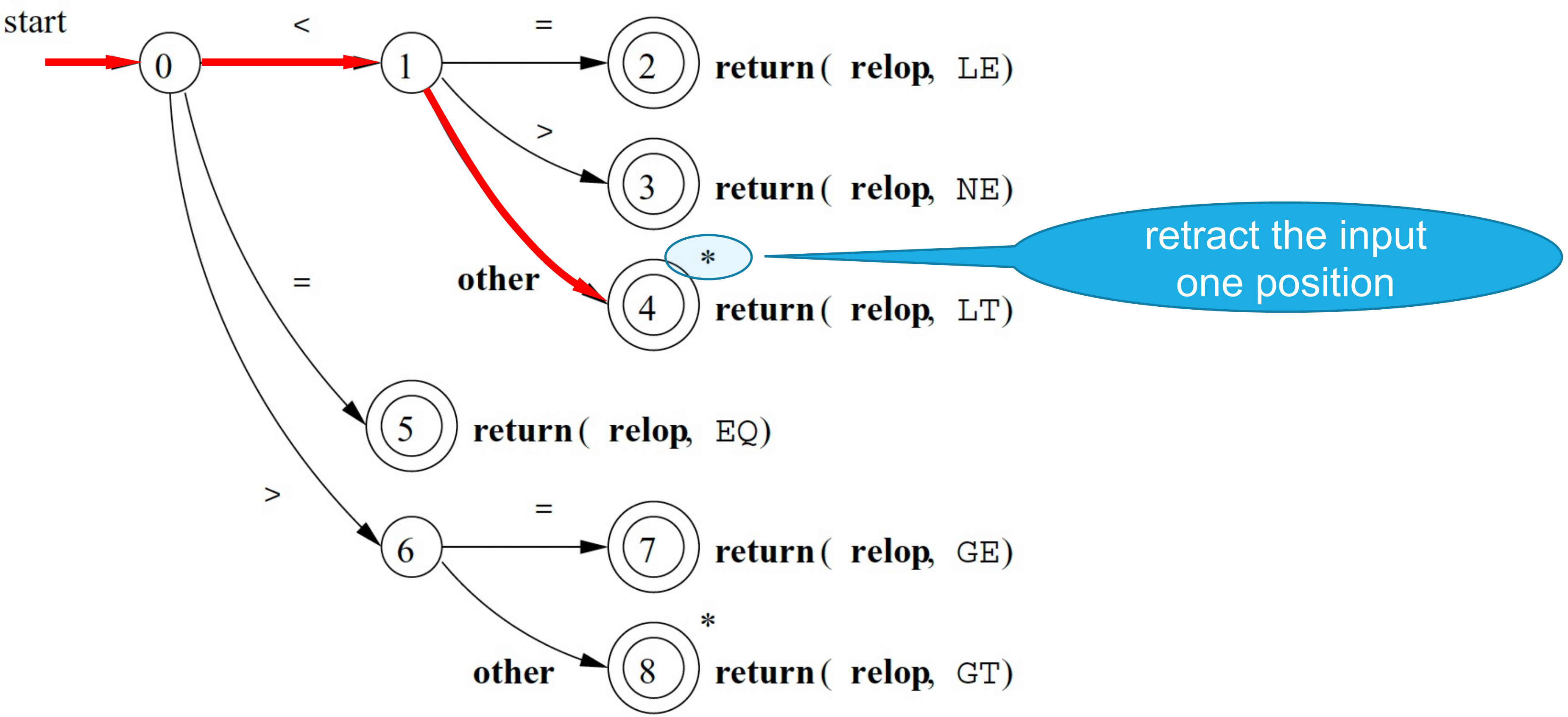
Example: Relational Operator



Example: Relational Operator

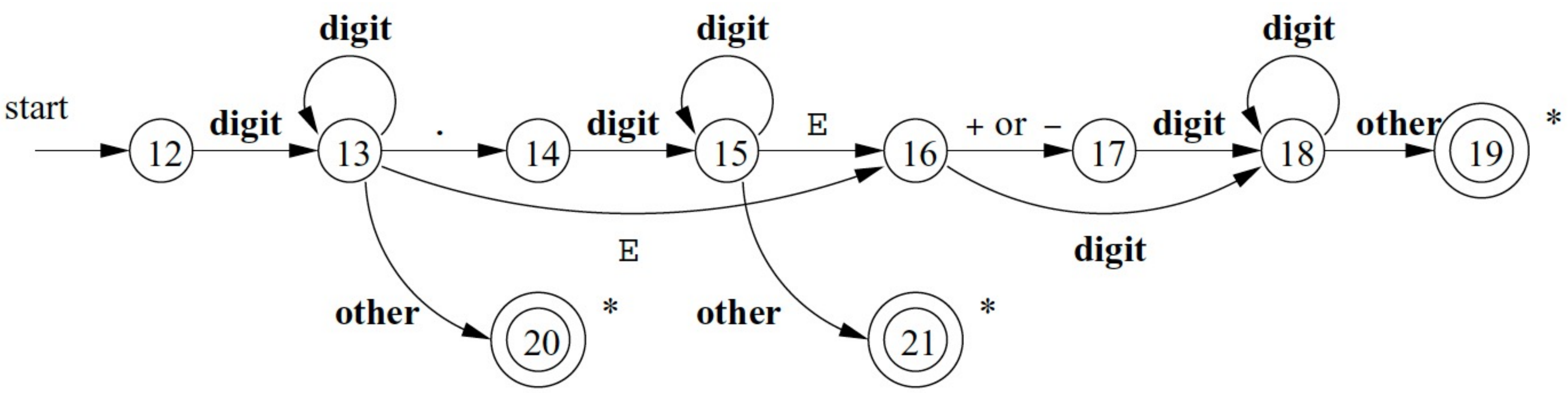


Example: Relational Operator

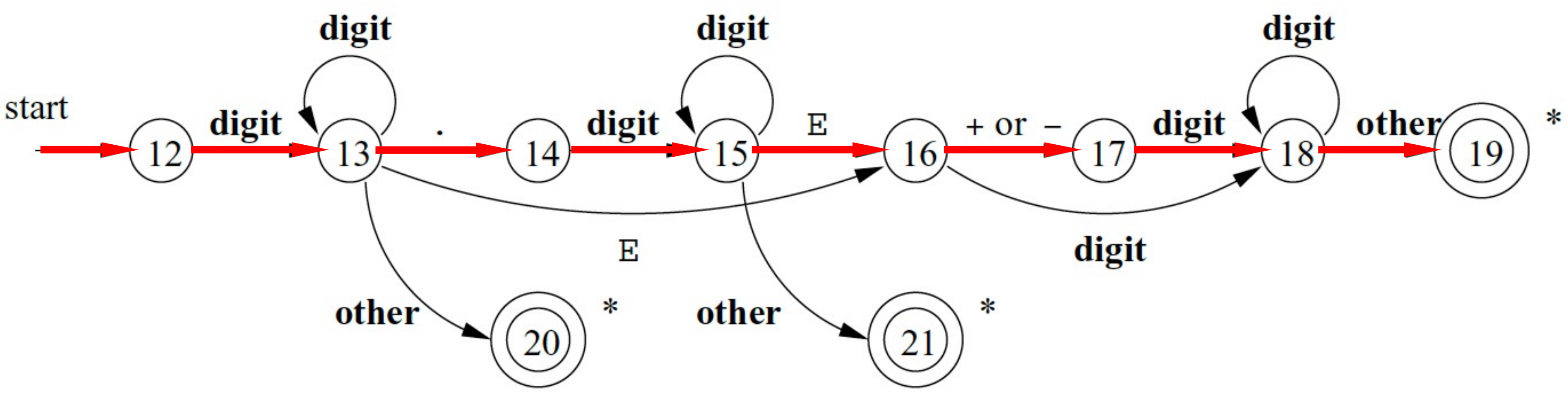


Example: Unsigned Numbers

Example: Unsigned Numbers



Example: Unsigned Numbers



Example: Identifiers

- Try to write the regex of identifiers for a C-like language and build the NFA

What if Multiple DFAs Work

- Priority of DFAs
- Matching the longest string
- …

PART IV: Pumping Lemma

Question

- Is the language represented by $a^n b^m$ regular?

Question

- Is the language represented by $a^n b^m$ regular?
- Yes!
- Regex: $a^* b^*$

Question

- Is the language represented by $a^n b^n$ regular?

Question

- Is the language represented by $a^n b^n$ regular?
- No!

Question

- Is the language represented by $a^n b^n$ regular?
- No!
- Why? Prove it!

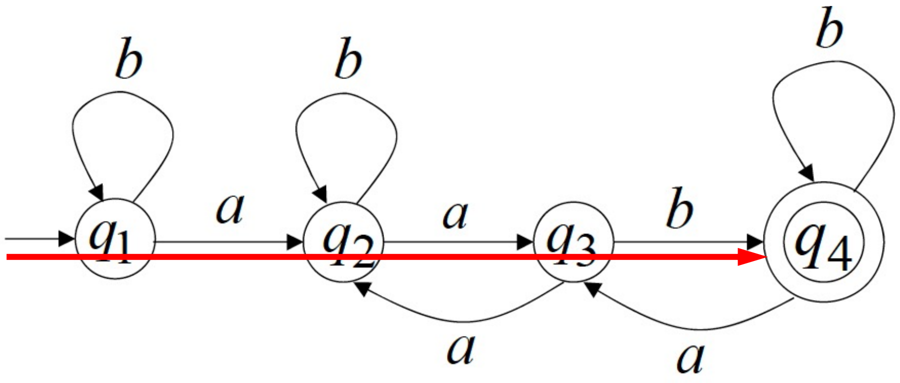
Observation

- For any DFA and string w, $|w| \geq$ # states,
- There must be a state q that repeats in the walk of w

Observation

- For any DFA and string w, $|w| \geq$ # states,
- There must be a state q that repeats in the walk of w

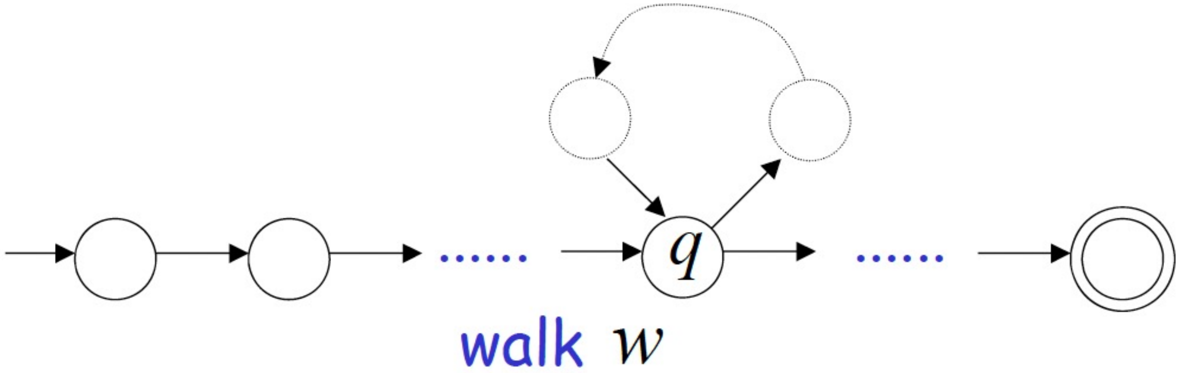
- Example:



A walk without repeating any state yields a string of length ≤ 3

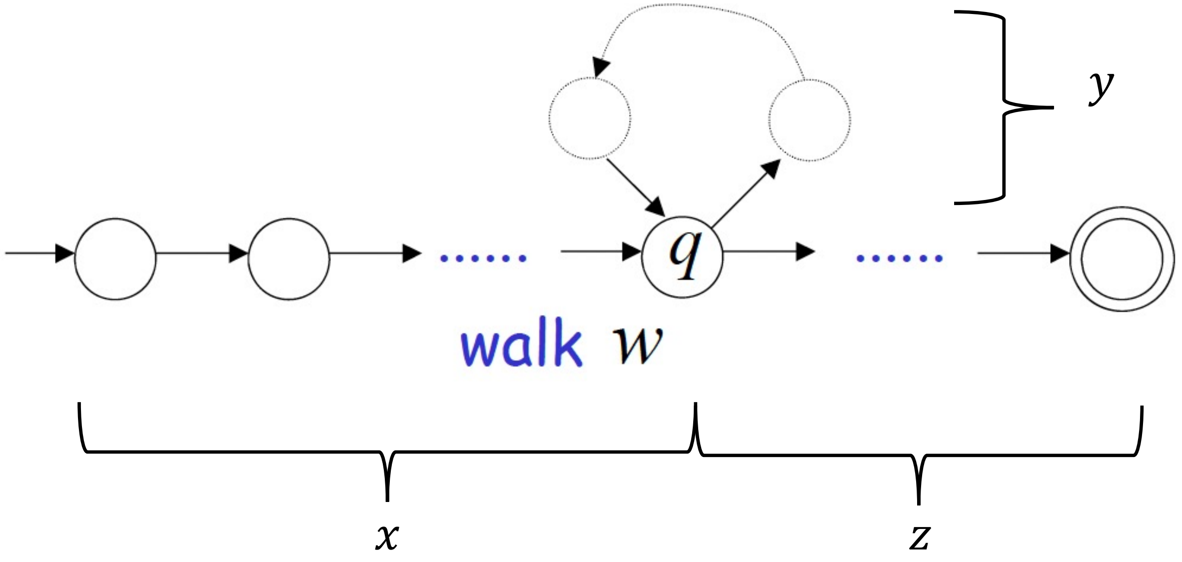
Observation

- $|w| \geq m$ (assume m is the number of states)



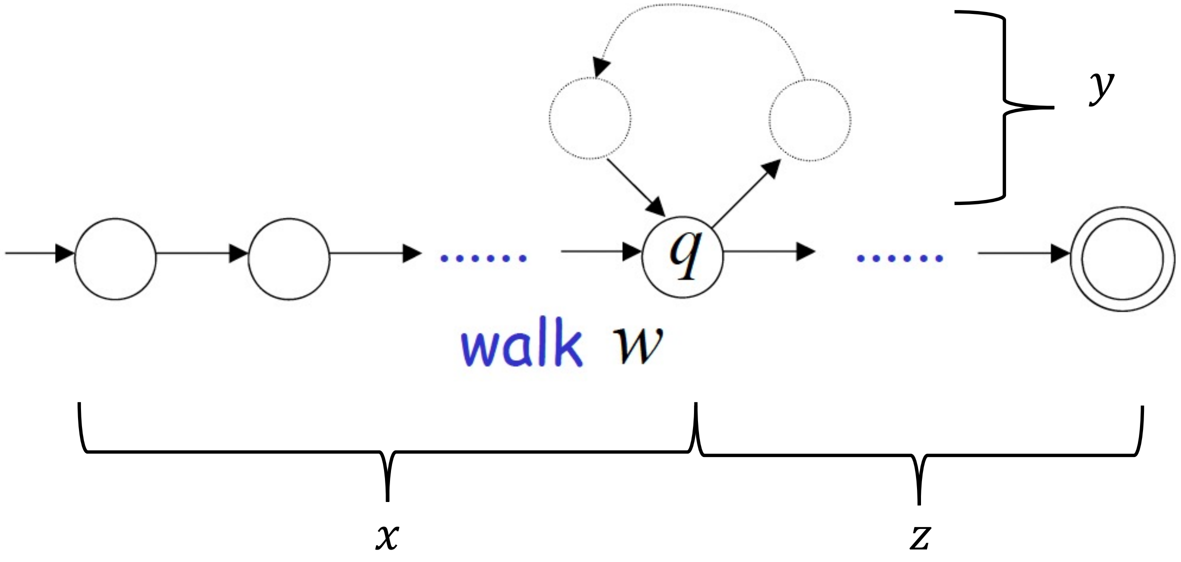
Observation

- $|w| \geq m$ (assume m is the number of states)
- $w_i = xy^iz$ where $|xy| \leq m$; $|y| \geq 1$



Observation

- $|w| \geq m$ (assume m is the number of states)
- $w_i = xy^iz$ where $|xy| \leq m$; $|y| \geq 1$
- For any i, w_i is in the language of the automata



The Pumping Lemma

- If L is an infinite regular language (the set L is infinite)

The Pumping Lemma

- If L is an infinite regular language (the set L is infinite)
- there must exist an integer m

The Pumping Lemma

- If L is an infinite regular language (the set L is infinite)
- there must exist an integer m
- for any string $w \in L$ with length $|w| \geq m$

The Pumping Lemma

- If L is an infinite regular language (the set L is infinite)
- there must exist an integer m
- for any string $w \in L$ with length $|w| \geq m$
- we can write $w = xyz$ with $|xy| \leq m$ and $|y| \geq 1$

The Pumping Lemma

- If L is an infinite regular language (the set L is infinite)
- there must exist an integer m
- for any string $w \in L$ with length $|w| \geq m$
- we can write $w = xyz$ with $|xy| \leq m$ and $|y| \geq 1$
- such that $w_i = xy^iz \in L$

Contrapositive Proportion

- For any integer m

Contraprositive Proportion

- For any integer m
- if there exists $w \in L$ such that $|w| \geq m$

Contrapositive Proportion

- For any integer m
- if there exists $w \in L$ such that $|w| \geq m$
- choose x, y, and z such that $w = xyz$, $y \neq \epsilon$, and $|xy| \leq m$

Contrapositive Proportion

- For any integer m
- if there exists $w \in L$ such that $|w| \geq m$
- choose x, y, and z such that $w = xyz$, $y \neq \epsilon$, and $|xy| \leq m$
- if there exists k, such that $xy^k z \notin L$

Contrapositive Proportion

- For any integer m
- if there exists $w \in L$ such that $|w| \geq m$
- choose x, y, and z such that $w = xyz$, $y \neq \epsilon$, and $|xy| \leq m$
- if there exists k, such that $xy^k z \notin L$
- ➔
- L is not regular

Proving $L = \{a^n b^n\}$ is not Regular

- For any integer m
- if there exists $w \in L$ such that $|w| \geq m$
- choose x, y, and z such that $w = xyz$, $y \neq \epsilon$, and $|xy| \leq m$
- if there exists k, such that $xy^k z \notin L$
- ➔
- L is not regular

Proving $L = \{a^n b^n\}$ is not Regular

- For any integer m Let m = 3
- if there exists $w \in L$ such that $|w| \geq m$
- choose x, y, and z such that $w = xyz$, $y \neq \epsilon$, and $|xy| \leq m$
- if there exists k, such that $xy^k z \notin L$
- ➔
- L is not regular

Proving $L = \{a^n b^n\}$ is not Regular

- For any integer m Let m = 3
- if there exists $w \in L$ such that $|w| \geq m$ Let w = aaabbb
- choose x, y, and z such that $w = xyz$, $y \neq \epsilon$, and $|xy| \leq m$
- if there exists k, such that $xy^k z \notin L$
- ➔
- L is not regular

Proving $L = \{a^n b^n\}$ is not Regular

- For any integer m Let m = 3
- if there exists $w \in L$ such that $|w| \geq m$ Let w = aaabbb
- choose x, y, and z such that $w = xyz$, $y \neq \epsilon$, and $|xy| \leq m$
- if there exists k, such that $xy^kz \notin L$ Let w = xyz, where x = a; y = a; z = abbb
- ➔
- L is not regular

Proving $L = \{a^n b^n\}$ is not Regular

- For any integer m — Let m = 3
- if there exists $w \in L$ such that $|w| \geq m$ — Let w = aaabbb
- choose x, y, and z such that $w = xyz$, $y \neq \epsilon$, and $|xy| \leq m$ — Let w = xyz, where x = a; y = a; z = abbb
- if there exists k, such that $xy^kz \notin L$ — Let k = 2, xy^kz = aaaabbb = $a^4b^3 \notin L$
- ➔
- L is not regular

Proving $L = \{a^n b^n\}$ is not Regular

- For any integer m Let m = 3
- if there exists $w \in L$ such that $|w| \geq m$ Let w = aaabbb
- choose x, y, and z such that $w = xyz$, $y \neq \epsilon$, and $|xy| \leq m$ Let w = xyz, where x = a; y = a; z = abbb
- if there exists k, such that $xy^kz \notin L$
- ➔ Let k = 2, xy^kz = aaaabbb = $a^4b^3 \notin$ L
- L is not regular a^nb^n is not regular

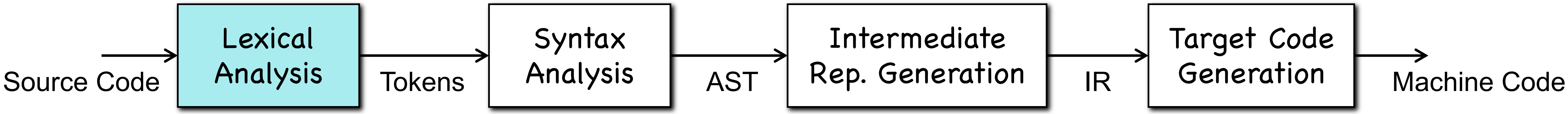
Summary

Source Code → **Lexical Analysis** → Tokens → Syntax Analysis → AST → Intermediate Rep. Generation → IR → Target Code Generation → Machine Code

- Find **lexemes** according to **patterns**, and create **tokens**
  - Lexeme – a character string
  - Pattern – regular expression (lexical errors if no patterns matched)
    - Regexp → NFA → DFA → min DFA
  - Token – <token-class-name, attribute>

```
position   =   initial   +   rate   *   60
   ↓       ↓      ↓      ↓    ↓     ↓    ↓
 <id,1>   <=>   <id,2>  <+>  <id,3> <*>  <number,60>
```

Summary



- Find **lexemes** according to **patterns**, and create **tokens**
  - Lexeme – a character string
  - Pattern – regular expression (lexical errors if no patterns matched)
    - Regexp → NFA → DFA → min DFA
  - Token – <token-class-name, attribute>

key	name	…
1	position	…
2	initial	…
…		…

symbol table

```
position  =  initial  +  rate  *  60
<id,1> <=> <id,2> <+> <id,3> <*> <number,60>
```

THANKS!