Compilers
Lab I

Mo.Agamia

Plan

- Course Rules
- Course overview & motivation
- Introduction & compiler phases
- Lexical analyser
- Regular expression
- NFA

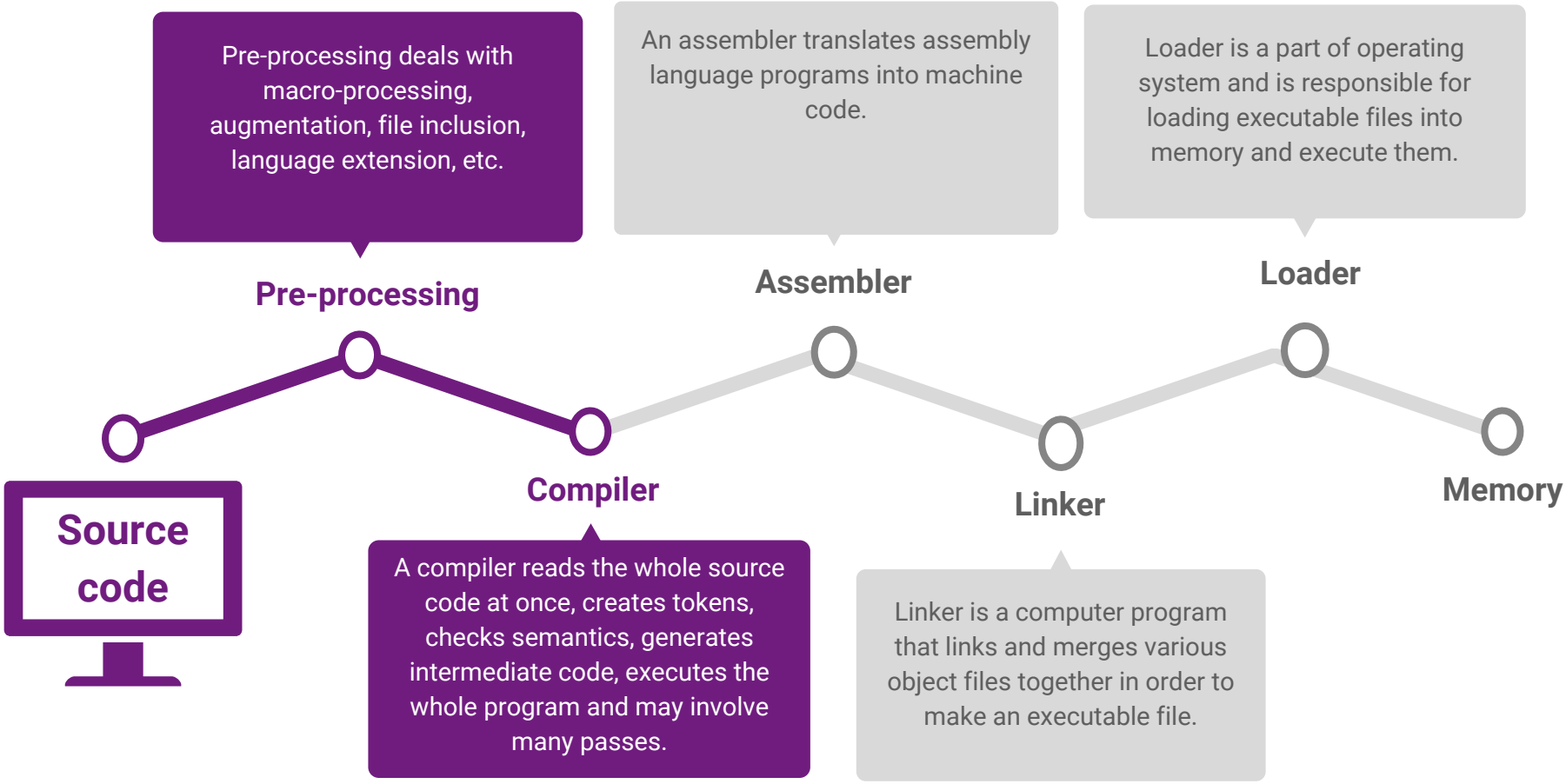
1.
Course rules

Rules:

1. Cheating → zero
2. No cross attendance
3. Adhere to deadline
4. Follow rule 1 & 2 & 3

2.
Course overview & motivation

Language Processing System



Motivation

- Why build compiler?
- Why study compiler construction?
- Isn't it a solved problem?
- Why attend lab?

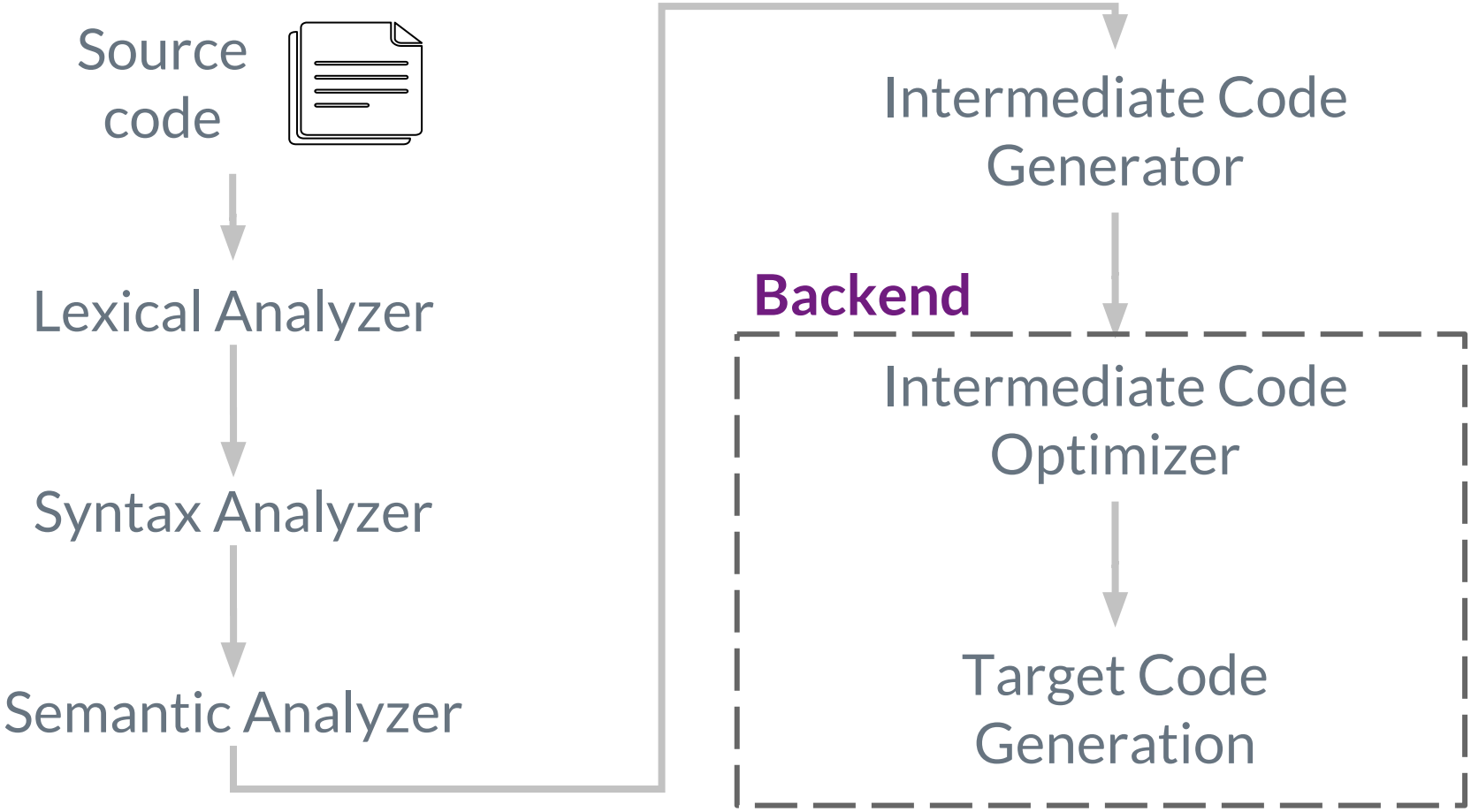
Motivation

- Compilers provide an essential interface between applications and architectures
- Compilers embody a wide range of theoretical techniques
  - AI, Algorithms, Theory, systems etc.
- Compiler construction teaches programming and software engineering skills
- Machines have continued to change since they have been invented

3.
Introduction & compiler phases

A compiler is a program that converts high-level language to assembly language.

Compiler phases



Compiler phases

Lexical Analysis

It scans the source code as a stream of characters and converts it into meaningful lexemes.

Syntax Analysis

It takes the token produced by lexical analysis as input and generates a parse tree.

Semantic Analysis

It checks whether the parse tree constructed follows the rules of language

Intermediate Code Generation

It generates an intermediate code of the source code for the target machine.

Intermediate Code Optimization

It does code optimization of the intermediate code.

Target Code Generation

It takes the optimized representation of the intermediate code and maps it to the target machine language.

Compiler phases

- x = a + b * c //statement
- Id = id + id * id
- Parse tree

$S \rightarrow id = E$

$E \rightarrow E+T/T$

$T \rightarrow T*F/F$

$F \rightarrow id$

- Verify parse tree semantically

Source code

Lexical Analyzer

Syntax Analyzer

Semantic Analyzer

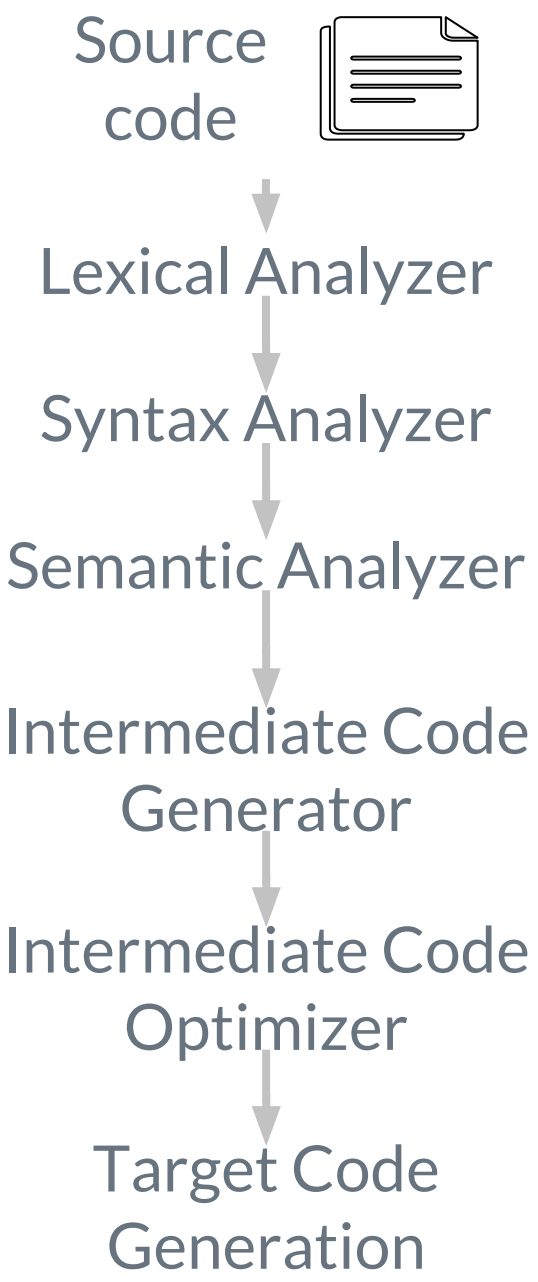
Intermediate Code Generator

Intermediate Code Optimizer

Target Code Generation

Compiler phases

- t1 = b * c
  t2 = a + t1
  x = t2
- t1 = b * c
  x = a + t1
- Mul R1 R2
  Add R0 R2
  Mov R2 x



4.
Lexical analyser

Lexical Analyser scans the source code as a stream of characters and converts it into meaningful lexemes.

LA

Ex. 1

```
System.out.println(“max x= ”
+ x);
```

Ex. 2

```
public void max(int x, int y) {
    if (x > y)
    System.out.println(x);
    System.out.println(y);
}
```

LA

Ex. 1

System.out.println(“max x= ”
+ x);

Token #: 11

Lexeme----------Token

System	[Key_Word]
.	[Object_Accessor]
out	[Key_Word]
.	[Object_Accessor]
println	[Key_Word]
(	[left_Parenthesis]
"max x= "	[String_Literal]
etc.	

LA

Ex. 1

```
System.out.println(“max x= ”
+ x);
```

Ex. 2

```
public void max(int x, int y) {
    if (x > y)
    System.out.println(x);
    System.out.println(y);
}
```

1. *Write regular definition*
2. *Compile corresponding regular expression*
3. *Convert expression to NFA*
4. *Convert NFA to DFA*

5.
Regular expression

Regular expressions are patterns used to match character combinations in strings.

They are equivalent to finite automata.

Regex

- Regular definitions:
  - letter → A|B|...|Z|a|b|...|z
  - digit → 0|1|...|9
  - id → letter(letter|digit)*
- One or more instances: r+ = rr*
- Zero or one instance: r? = r|ε
- Character classes:
  - [a-z]=a|b|...|z
  - [0-9]=0|1|...|9

Regex

^ → start of string	\w → any word char	\W → any non-word char
$ → end of string	\s → any whitespace char	\S → any non-whitespace char
\n → new line	\d → any digit	\D → any non-digit
\t → tab	? → zero or one	+→ one or more

Regex

. → any single char	*→ zero or more	[^a] → anything not a
a{3} → exactly 3 a	a{3,} → 3 or more of a	a{3, 6} → between 3 & 6 a
(a|b) → a or b	(?=...) → +ve lookahead	(?!...) → -ve lookahead
(?<=...) → +ve lookbehind	(?<!...) → -ve lookbehind	[a-z] → any characters between a and z

6.
NFA

For any string w, you can NOT determine the exact sequence of states the machine will enter as it scans w.

NFA Examples

- L1 = { starts with an ‘a’ }
- L2 = { contains an ‘a’ }
- L3 = { ends with an ‘a’ }
- L4 = { starts with ‘ab’ }
- L5 = { contains ‘ab’ }
- L6 = { ends with ‘ab’ }

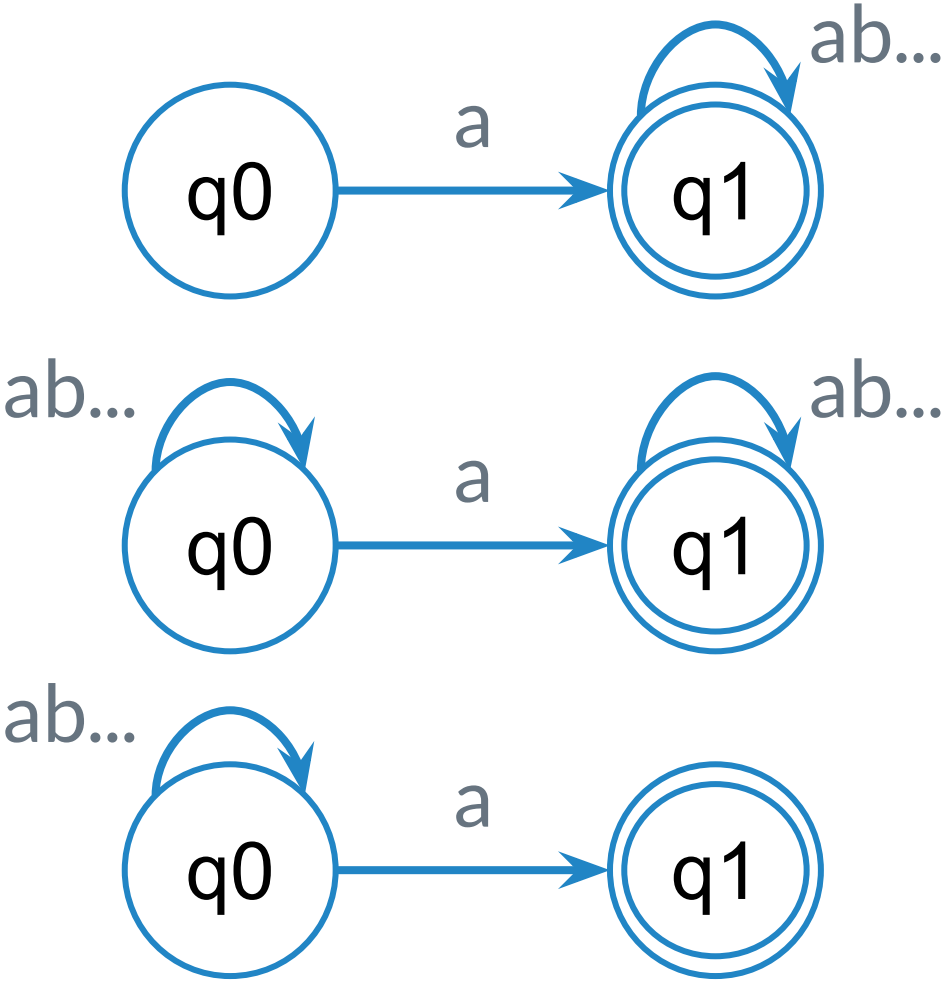
NFA Examples

- L1 = { starts with an ‘a’ }
- L2 = { contains an ‘a’ }
- L3 = { ends with an ‘a’ }
- L4 = { starts with ‘ab’ }
- L5 = { contains ‘ab’ }
- L6 = { ends with ‘ab’ }

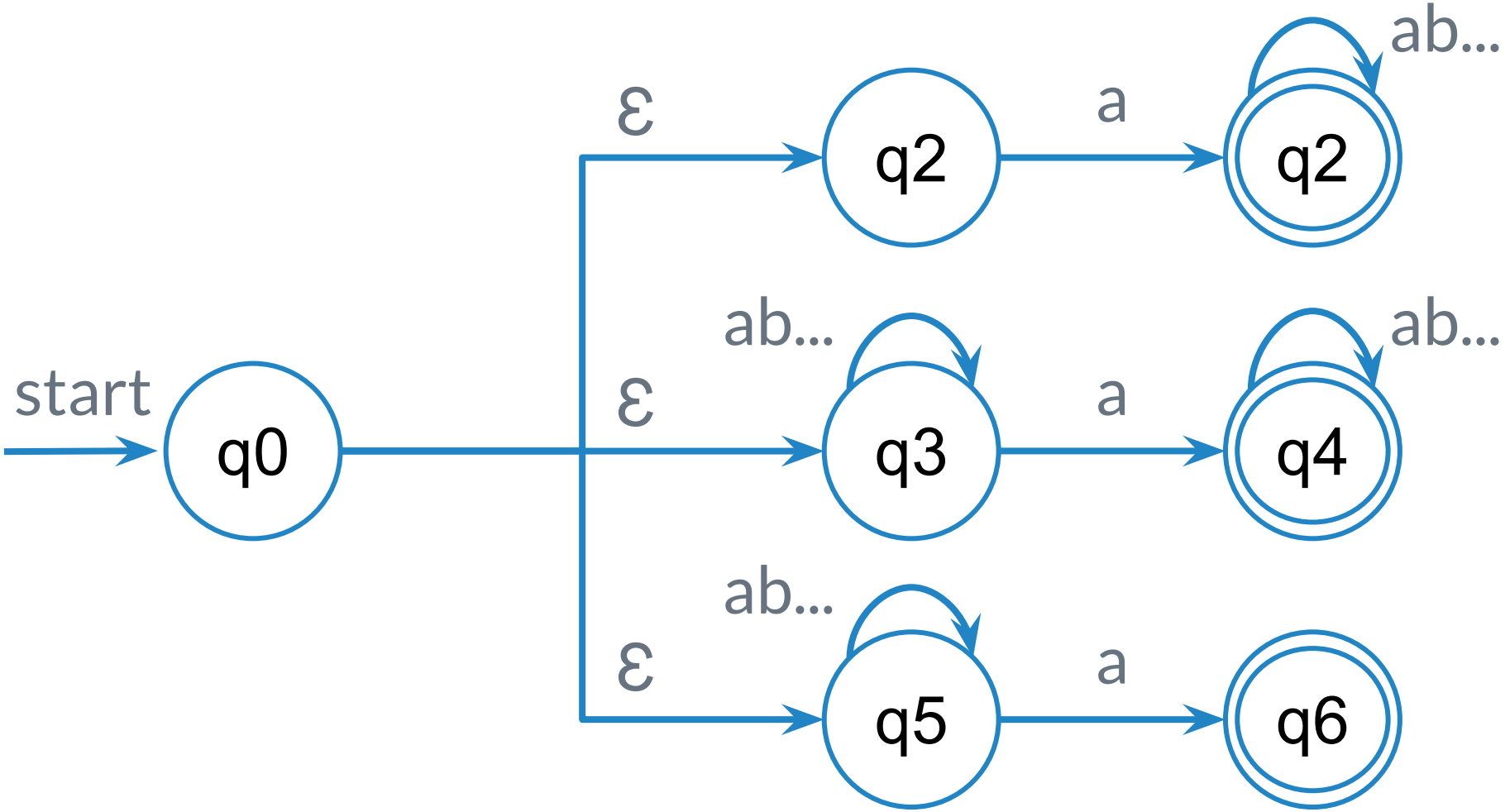
- ^a\w*
- \w*a\w*
- ^\w*a$
- ^ab\w*
- \w*ab\w*
- \w*ab$

NFA Examples

- ^a\w*
- \w*a\w*
- \w*a$

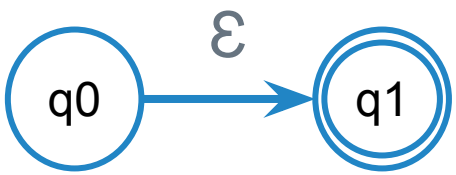


NFA Examples

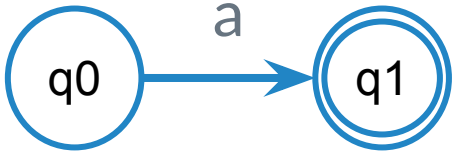


Thompson's Construction

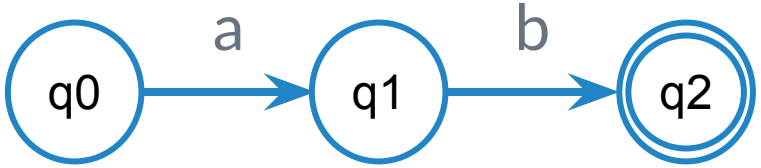
Empty-expression



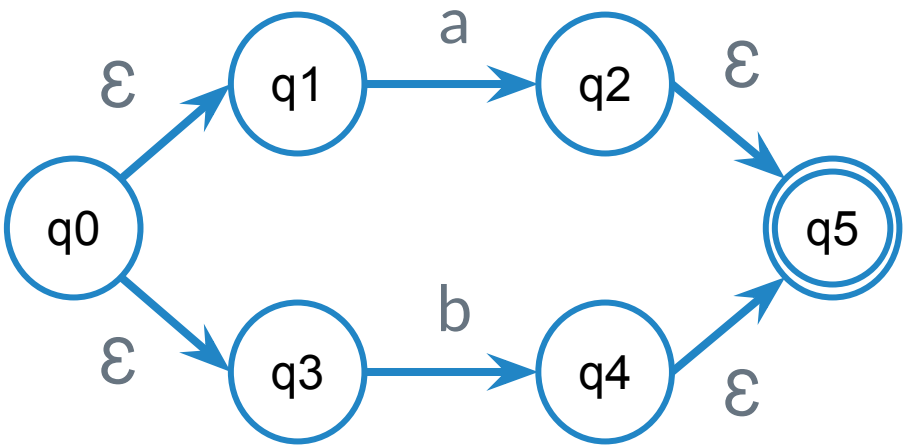
A symbol a



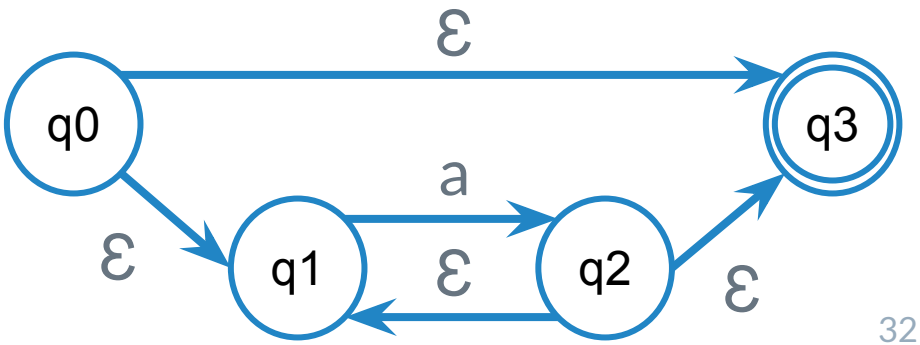
Concatenation expression



Union expression



Kleene star expression



NFA Examples

- xy*
- (x|y)*
- (ϵx)|y*

Thanks!

Any questions?

You can find me at:

@piazza

mohammed.agamia@guc.edu.eg