CSE216
Programming Abstraction

Instructor: Zhoulai Fu

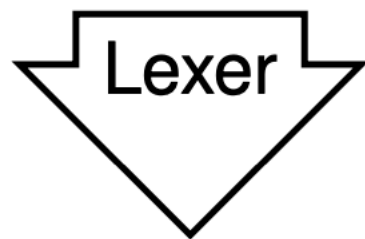## State University of New York, Korea

What is a program?

- Let us consider an expression x + y*10

- Think of this expression as a program in a programming language

- This is actually the calculator's programming language

Two aspects of a programming language

- Syntax: How a program is structured

- Semantics: Meaning of a program

Syntax analysis

Regular expression specification looks like this in Ocaml
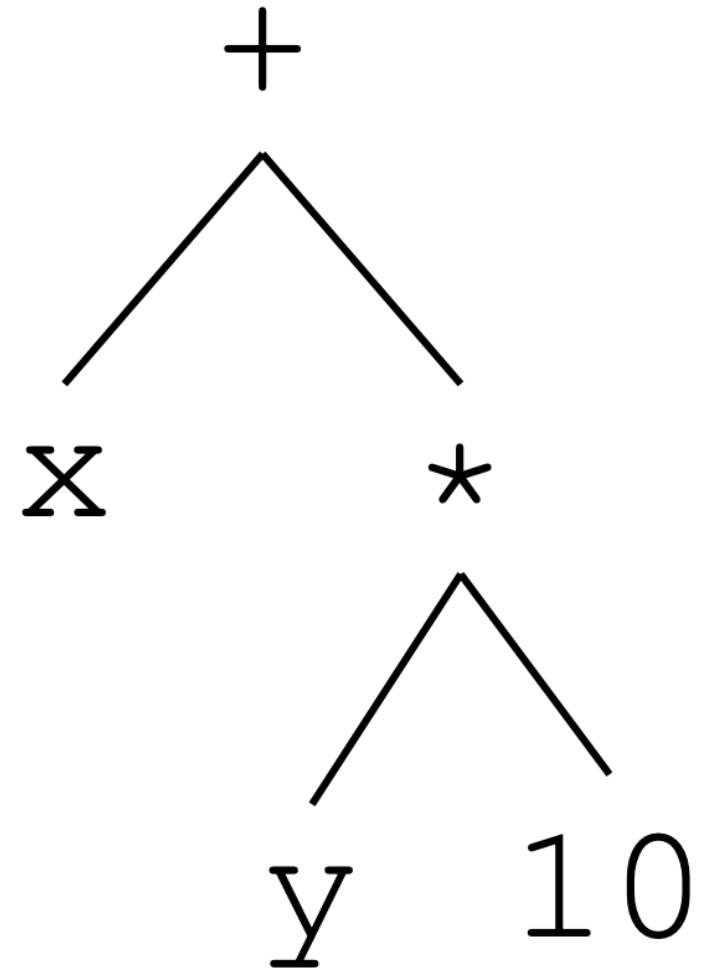
```
rule Token = parse
  | [' ' '\t' '\n' '\r']      { Token lexbuf }
  | ['0'-'9']+                { CSTINT (...) }
  | ['a'-'z''A'-'Z']['a'-'z''A'-'Z''0'-'9']*
                              { keyword (...) }
  | '+'                       { PLUS  }
  | '-'                       { MINUS }
  | '*'                       { TIMES }
  | '('                       { LPAR  }
  | ')'                       { RPAR  }
  | eof                       { EOF   }
  | _                         { lexerError lexbuf "Bad char" }
```

Context-free grammar specification looks like this in Ocaml

```
Main ::= Expr EOF
Expr ::= NAME
       | CSTINT
       | - CSTINT
       | ( Expr )
       | let NAME = Expr in Expr end
       | Expr *  Expr
       | Expr + Expr
       | Expr - Expr
```

Today

- Regular expressions

- Finite State Automata

- Nondeterministic Finite Automaton (NFA)

- Deterministic Finite Automaton (DFA)

- Context-free Grammars

- Derivation and ambiguity

- Python basics for the recitation

Regular expressions

r	Meaning	Language $\mathcal{L}(r)$
a	Character a	$\{"a"\}$
ε	Empty string	$\{""\}$
$r_1 r_2$	r_1 followed by r_2	$\{s_1 s_2 \mid s_1 \in \mathcal{L}(r_1), s_2 \in \mathcal{L}(r_2)\}$
r^*	Zero or more r	$\{s_1 \ldots s_n \mid s_i \in \mathcal{L}(r), n \geq 0\}$
$r_1 \mid r_2$	Either r_1 or r_2	$\mathcal{L}(r_1) \cup \mathcal{L}(r_2)$

Examples

ab^* represents $\{"a", "ab", "abb", \ldots\}$

$(ab)^*$ represents $\{"", "ab", "abab", \ldots\}$

$(a\mid b)^*$ represents $\{"", "a", "b", "aa", "ab", "ba", \ldots\}$

Exercise

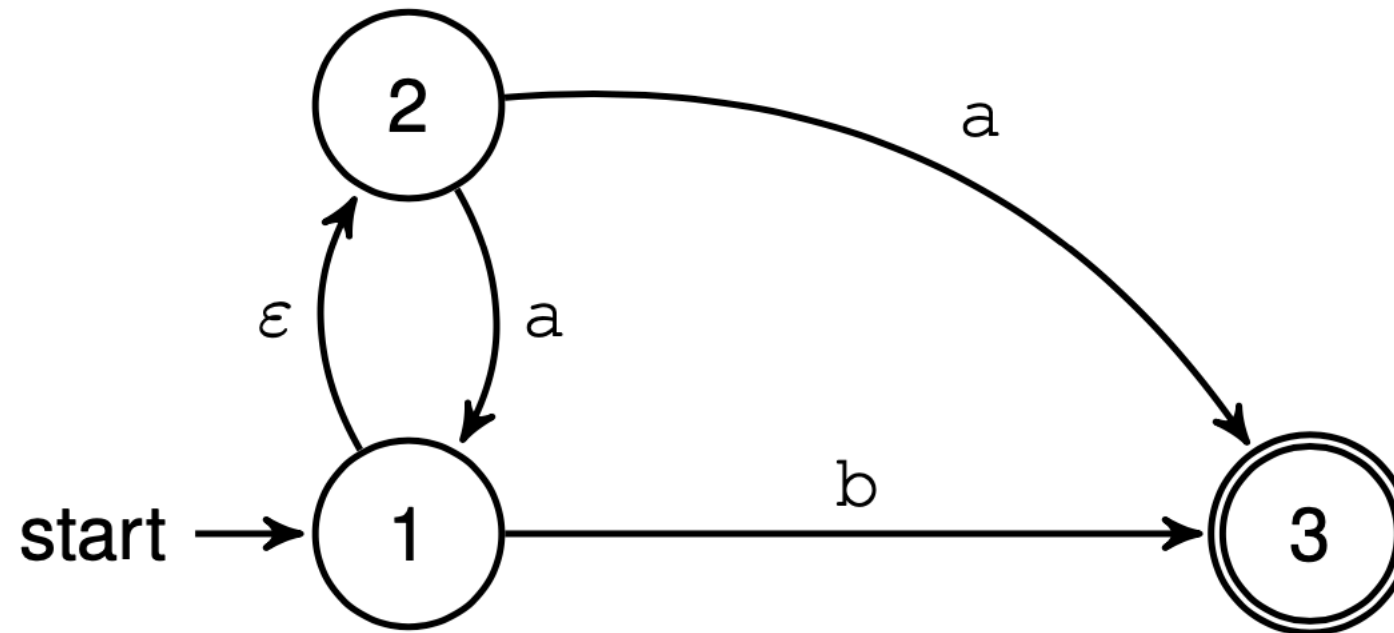What does $(a\mid b)c^*$ represent?

Regular expression abbreviations

| Abbrev.	Meaning	Expansion
`[aeiuo]`	Set	`a|e|i|o|u`
`[0-9]`	Range	`0|1|`...`|8|9`
`[0-9a-z]`	Ranges	`0|1|`...`|8|9|a|b|`...`|y|z`
$r?$	Zero or one r	$r|\varepsilon$
r^+	One or more r	$r\,r^*$

Exercises

Write regular expressions for:

- Non-negative integer constants

- Integer constants
- Floating-point constants:
  - 3.14
  - 3E8
  - +6.02E23
- Java variable names:
  - xy
  - x12
  - _x
  - $x12

Finite State Automata

- A finite automaton, FA, is a graph of states (nodes) and labelled transitions (edges)

- An FA accepts string s if there is a path from start to an accept state such that the labels make up s

- Epsilon (ε) does not contribute to the string

- This automaton is nondeterministic (NFA)

- It accepts string b

- Does it accept a or aa or ab or aba?

Regular expression = finite automata
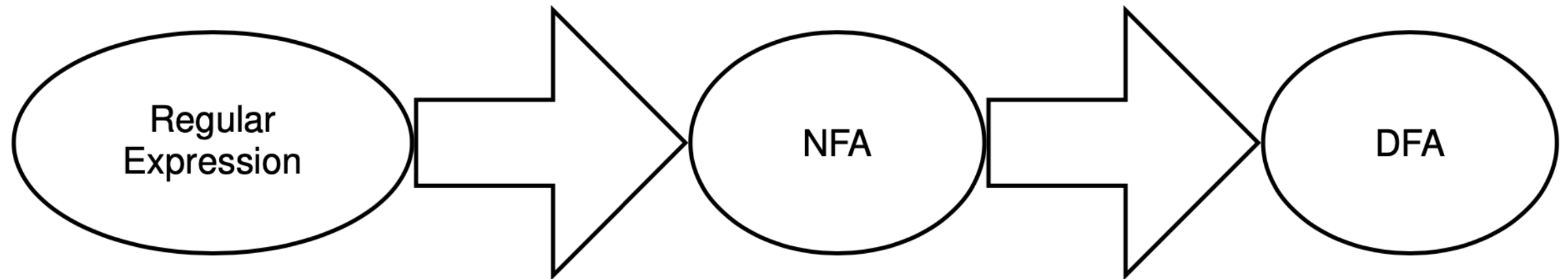
- For every regular expression r, there exists a deterministic finite automaton that recognizes precisely the strings described by r.

- The converse is also true.

- Construction: Regular expression $\Rightarrow$ Nondeterministic finite automaton (NFA) $\Rightarrow$ Deterministic finite automaton (DFA)

- Results in an efficient way of determining whether a given string is described by a regular expression
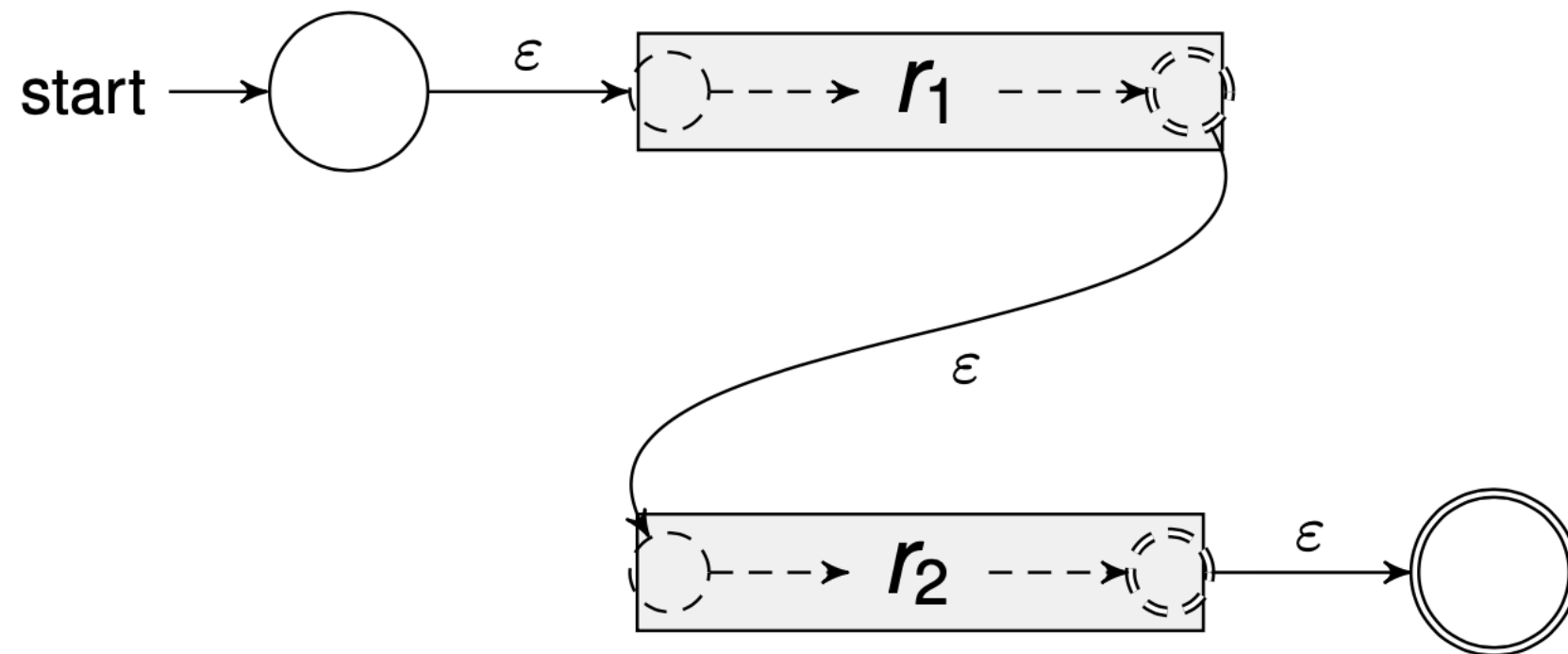
From regular expression to NFA

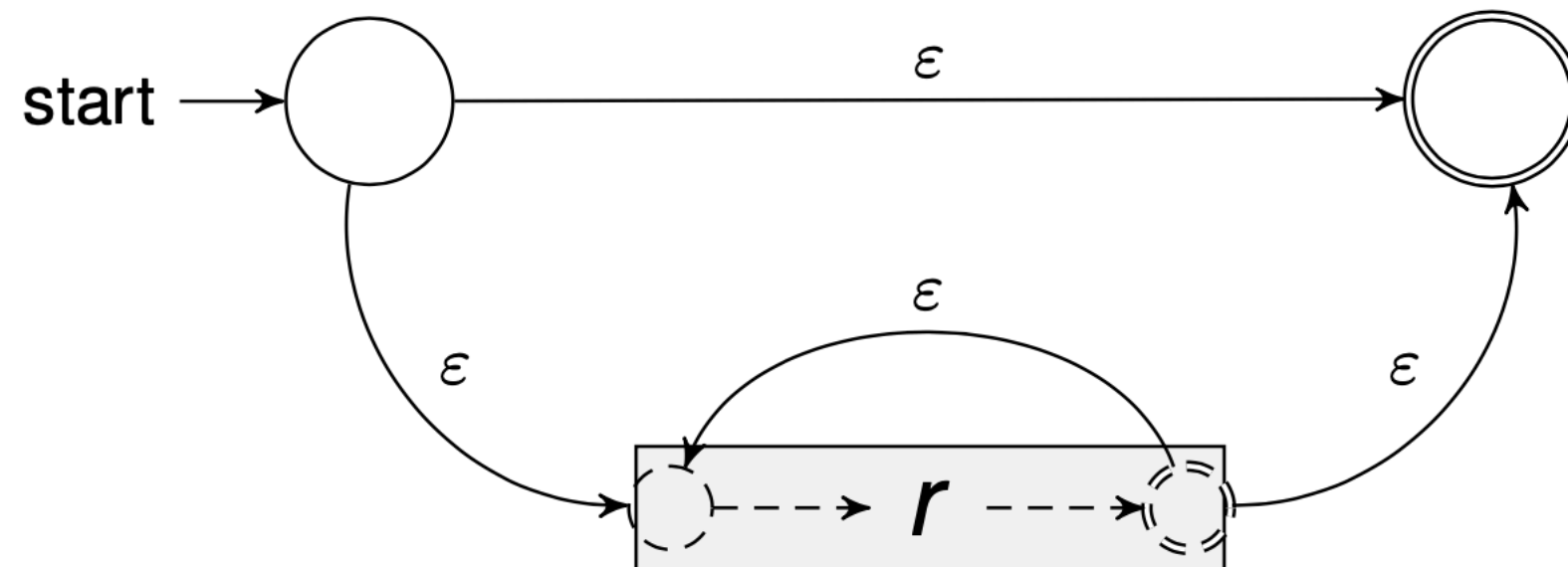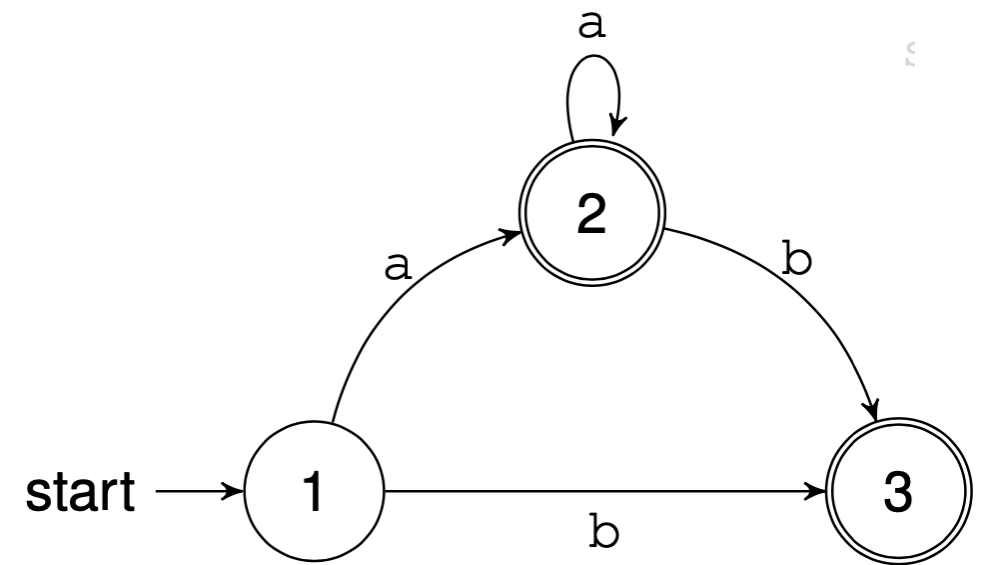Build NFA recursively by the case of the regular expression.
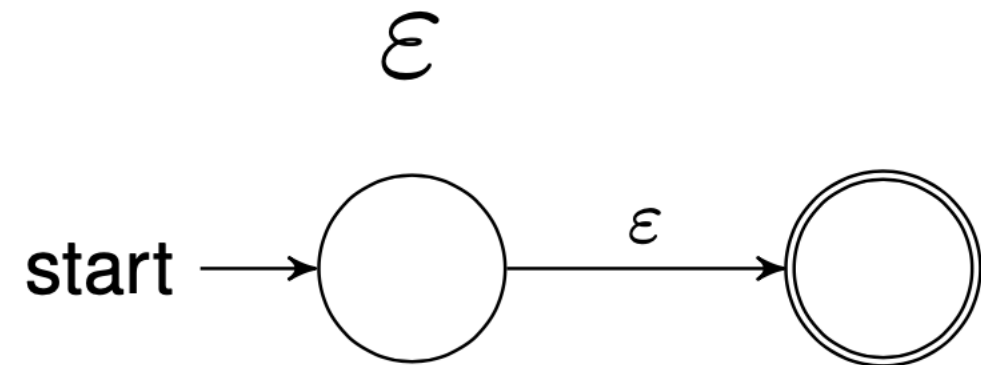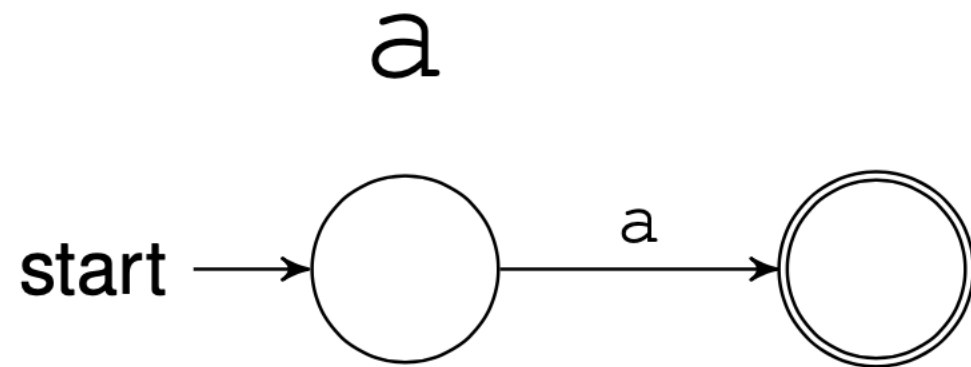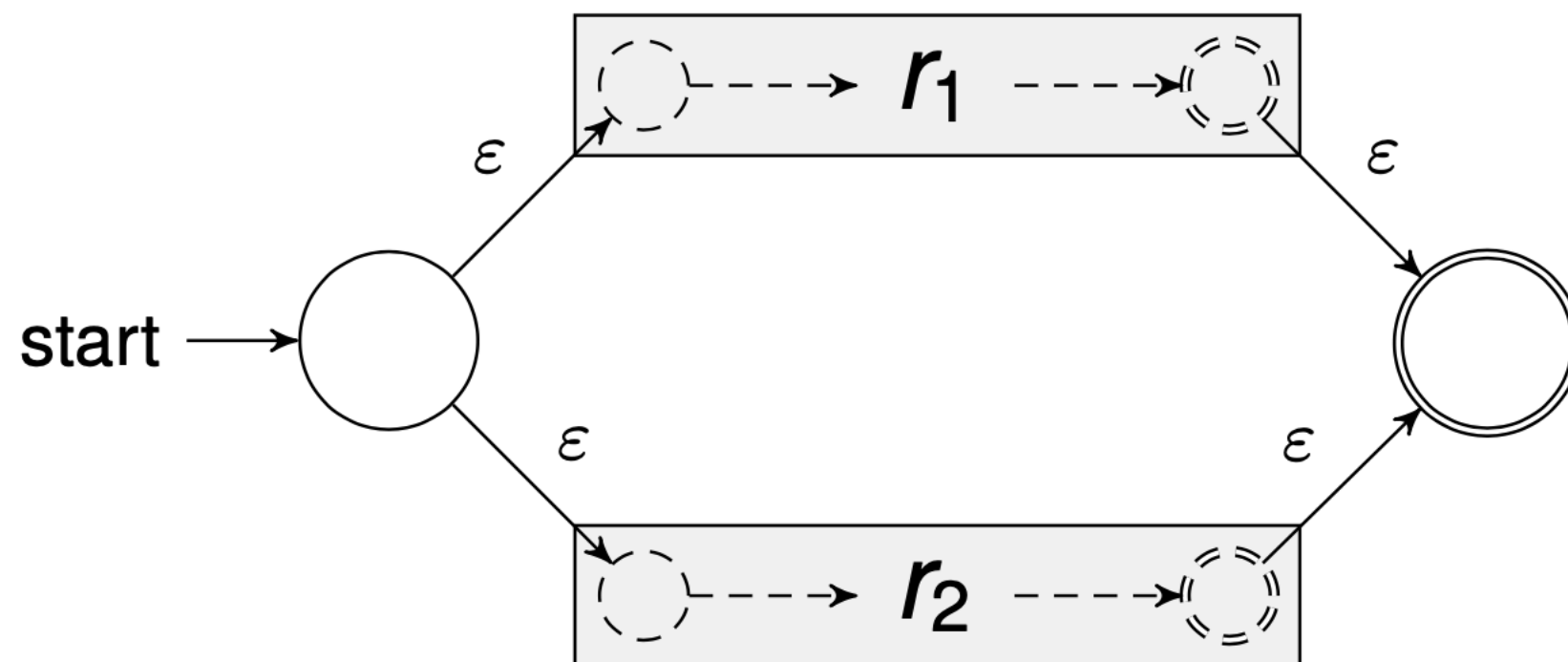
$r*$

Exercise:
- *Make NFA for* `(ab)*`
- *Make NFA for* `(a|b)*`

Deterministic Finite Automata

- No ε-transitions

- Distinct transitions from each state

- Multiple accepting states OK

Why DFA
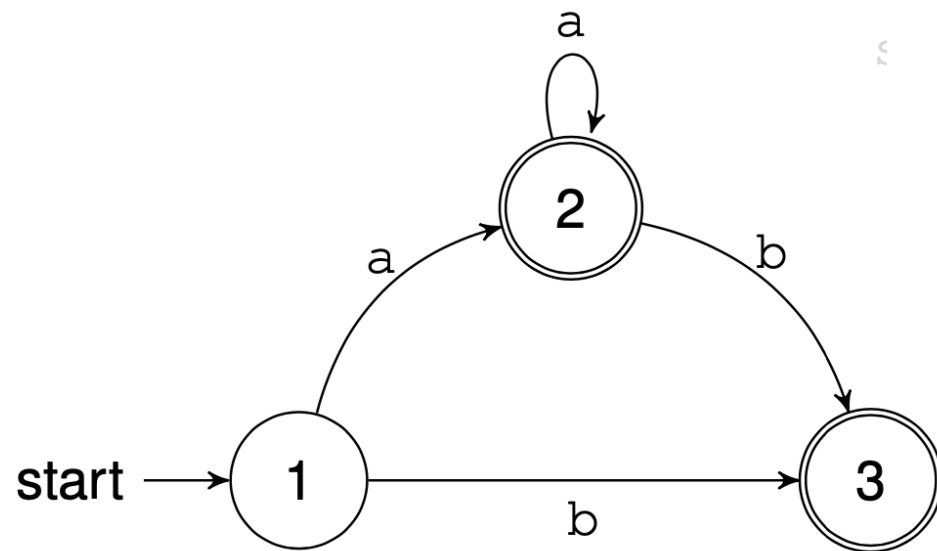
State	Input	Go to
1	a	2
1	b	3
2	a	2
2	b	3
3	a	fail
3	b	fail

- A DFA is easy to implement with table lookup:

  - next_state = table[current_state][next_symbol]

- Decides in linear time whether it accepts a string s

- For every NFA there is a corresponding DFA that accepts the same set of strings.

Summary so far
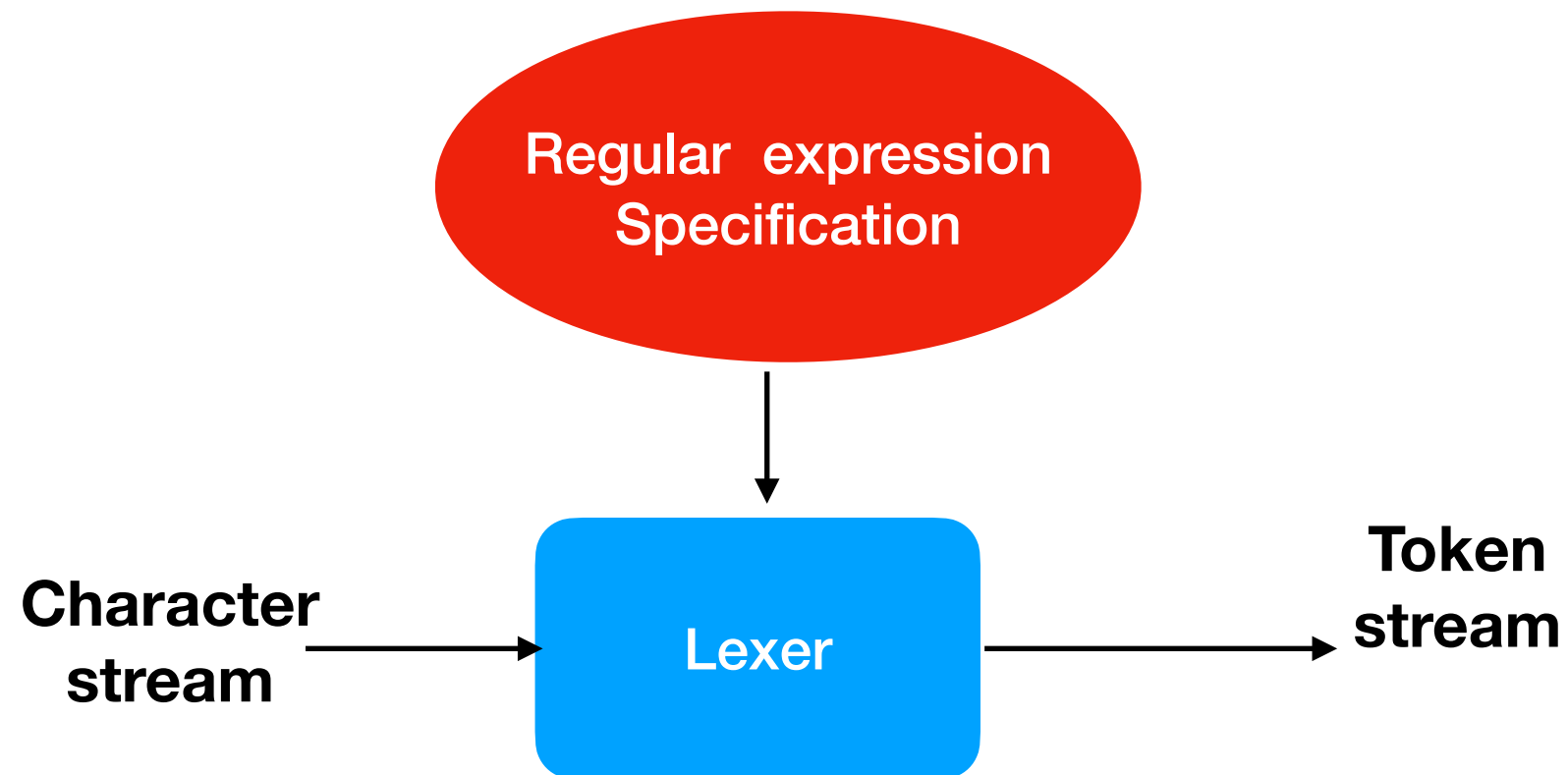
Formal grammar

- Formal grammar is a set of rules **for generating the structure of a language** or a formal language. It is a mathematical framework used to describe the syntax of a language, where syntax refers to the set of rules governing how words and phrases are combined to form sentences.

- Formal grammars are typically expressed using a set of **symbols and production rules** that define how those symbols can be combined to form valid sentences or expressions.

- There are **different types of formal grammars, including context-free grammars, regular grammars, and context-sensitive grammars**, each with its own set of rules and restrictions. Formal grammar is an essential tool for analyzing and understanding the structure of languages, both natural and artificial, and it has applications in fields such as computational linguistics, natural language processing, and artificial intelligence.

Context-free grammar, a first example

1. $S \rightarrow aSb$
2. $S \rightarrow ba$

- We start with S, and can choose a rule to apply to it.

- If we choose rule 1, we obtain the string aSb. If we then choose rule 1 again, we replace S with aSb and obtain the string aaSbb. If we now choose rule 2, we replace S with ba and obtain the string aababb, and are done.

- We can write this series of choices more briefly, using symbols: S => aSb => aaSbb => aababb

- The language of the grammar is the infinite set {a^n ba b^n | n>0} where n is is repeated times.

- This grammar is *context-free* (only single nonterminals appear as left-hand sides) and *unambiguous*.

Definition of Formal Grammar

$G = < V, \Sigma, P, \sigma >$

$V -$ set of *terminal* symbols

$\Sigma -$ set of *nonterminal* symbols with the restriction
that V and Σ are disjoint

$\sigma -$ start symbol

$P -$ set of production rules in a form:

$A -> B$

where:

A – is a sequence of symbols having at least one
nonterminal,

B – is the result of replacing some *nonterminal* symbol
A with a sequence of symbols (possibly empty)
from V and Σ

Example 2: A small set of English

$V = \{$"the", "a", "cat", "dog", "saw", "chased"$\}$

$\Sigma = \{S, NP, VP, D, N, V\}$

    S – sentence   D – determiner

    NP – noun phrase   N – noun

    VP – verb phrase   V – verb

$\sigma = S$

$P = \quad \{$

            S –> NP VP,

            NP –> D N,

            VP –> V NP,

            D –> "the",   D –> "a",

            N –> "cat",   N –> "dog",

            V –> "saw",   V –> "chased"

    $\}$

Cont.

S –> NP VP

  –> D N VP

  –> "the" N VP

  –> "the" "cat" VP

  –> "the" "cat" V NP

  –> "the" "cat" "chased" NP

  –> "the" "cat" "chased" D N

  –> "the" "cat" "chased" "a" N

  –> "the" "cat" "chased" "a" "dog"

Abstract syntax tree
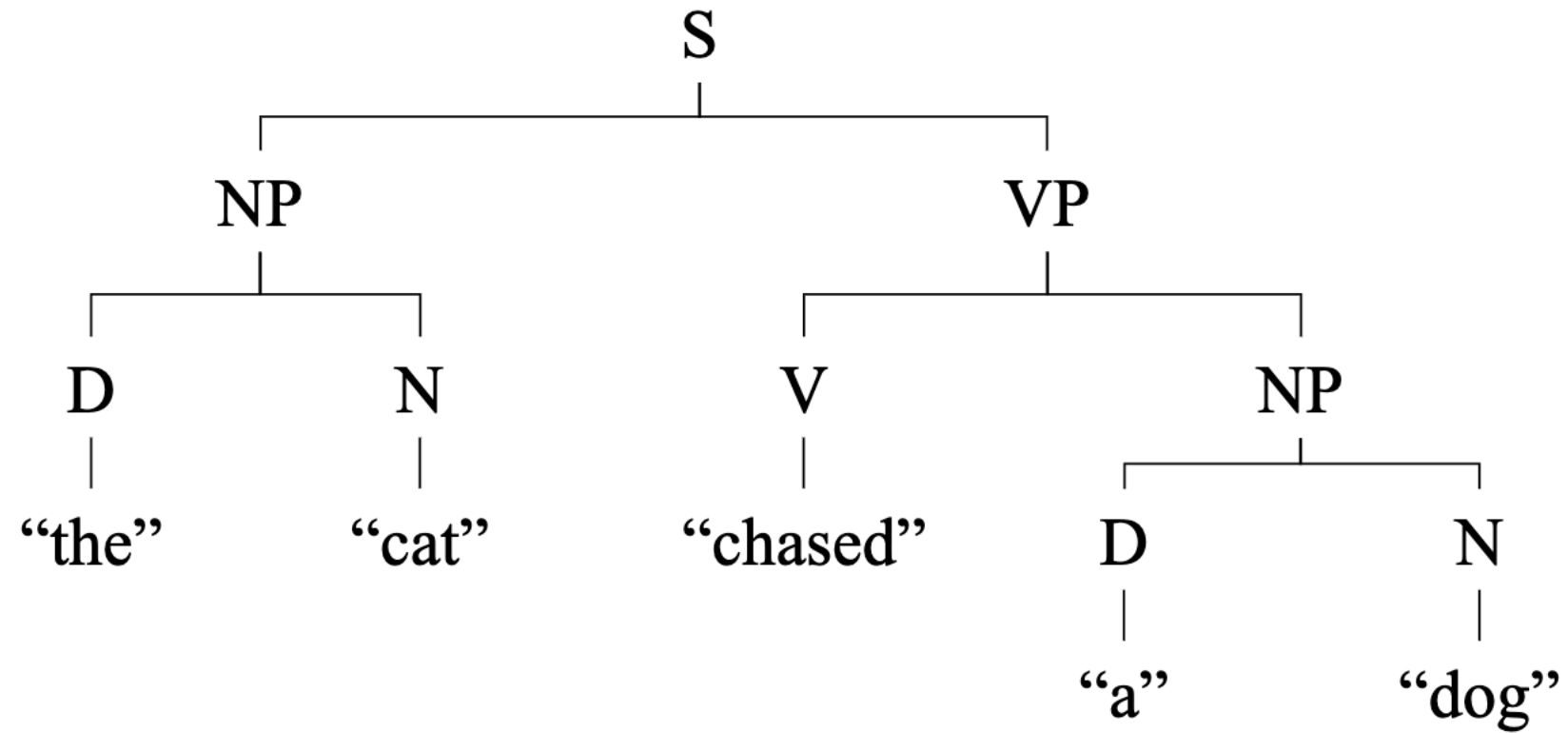
Example 3: An Ocaml expression (to be parsed together with a lexer)

```
Main ::= Expr EOF                          (rule A)
Expr ::= NAME                              (rule B)
       | CSTINT                            (rule C)
       | - CSTINT                          (rule D)
       | ( Expr )                          (rule E)
       | let NAME = Expr in Expr end       (rule F)
       | Expr *  Expr                      (rule G)
       | Expr + Expr                       (rule H)
       | Expr - Expr                       (rule I)
```

- • Nonterminal symbols

- Terminal symbols (from lexer): EOF, CSTINT, NAME, 'let', 'in'

- Grammar rules, or Productions (called A–I)

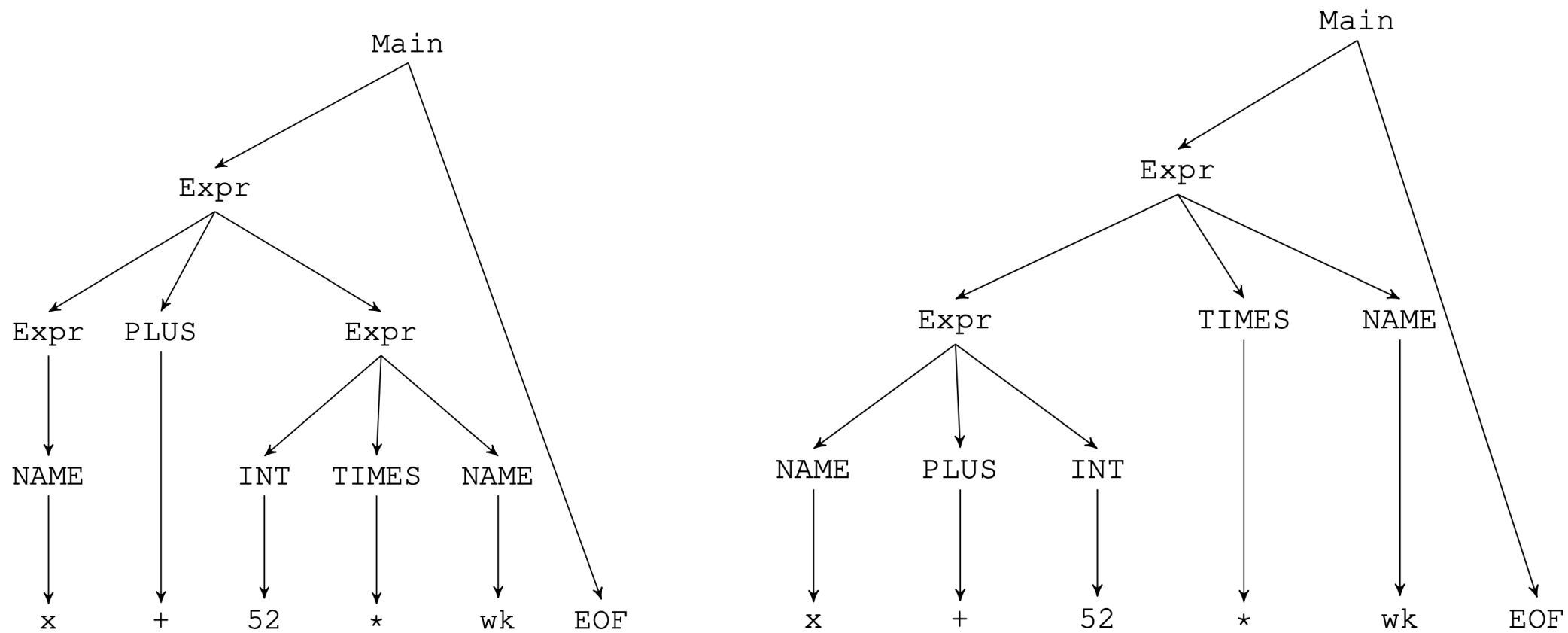- Start symbol (the nonterminal Main)

Generating a syntax tree using a context-free grammar

```
Main ::= Expr EOF                          (rule A)
Expr ::= NAME                              (rule B)
       | CSTINT                            (rule C)
       | - CSTINT                          (rule D)
       | ( Expr )                          (rule E)
       | let NAME = Expr in Expr end       (rule F)
       | Expr *  Expr                      (rule G)
       | Expr + Expr                       (rule H)
       | Expr - Expr                       (rule I)
```
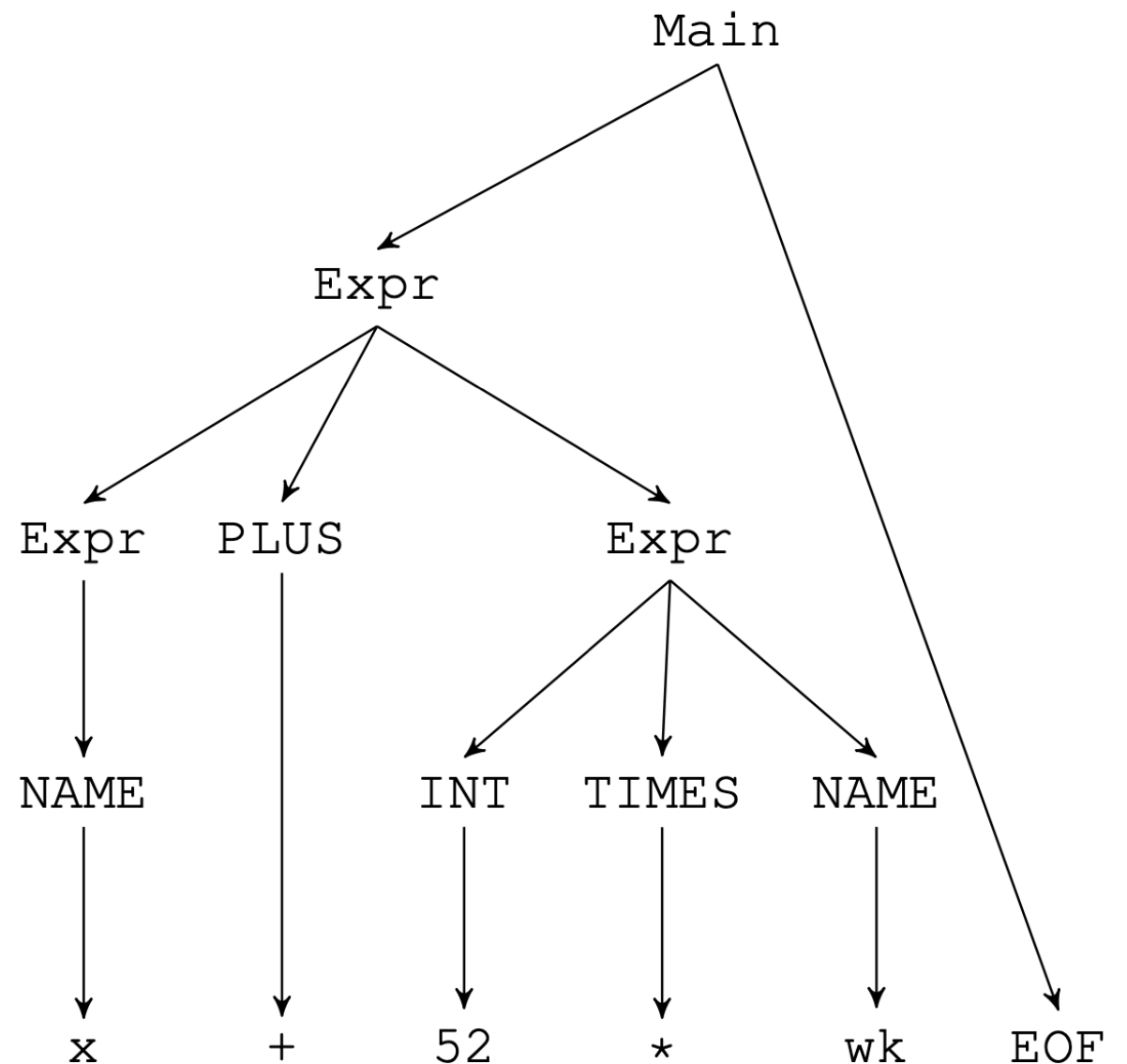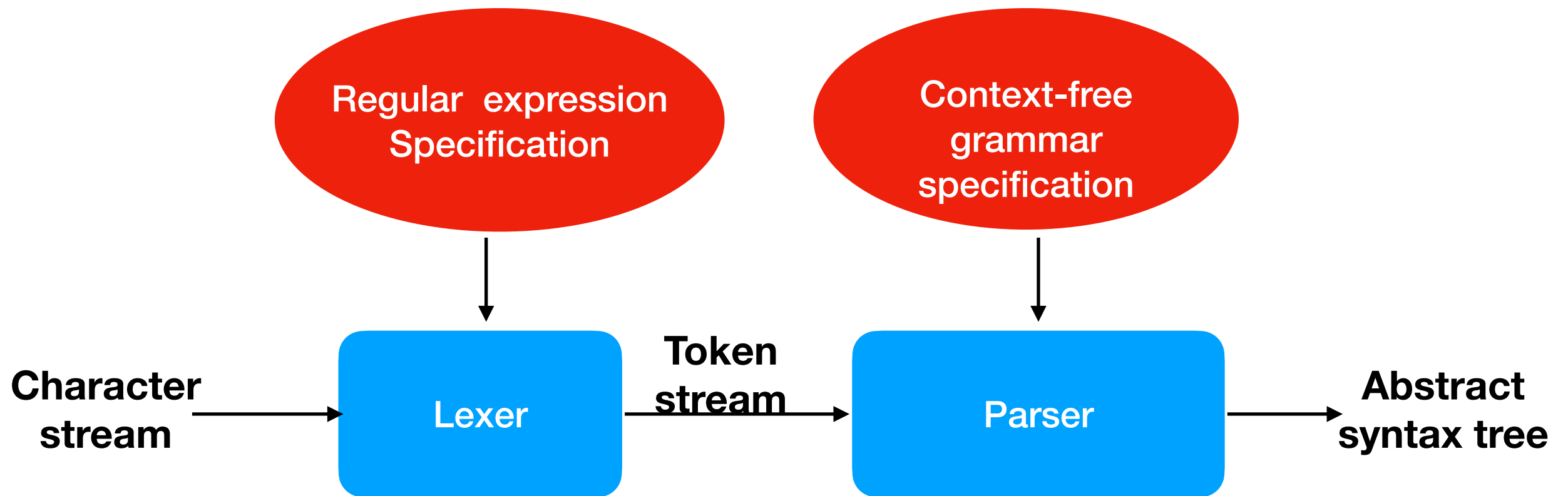
Grammar ambiguity

- A grammar is ambiguous if there exists a string with more than one derivation tree.

Summary

- A lexer converts a character stream to a token stream, using specification of regular expressions

- A parser converts a token stream to an abstract syntax tree, using specification of context-free grammar

Basic Python (for recitation 1)

- Basic Python:

- https://colab.research.google.com/drive/15eilquB2QVacfZWadm_jIw5Xv2ihl60J#scrollTo=UhcbBQUiStHG