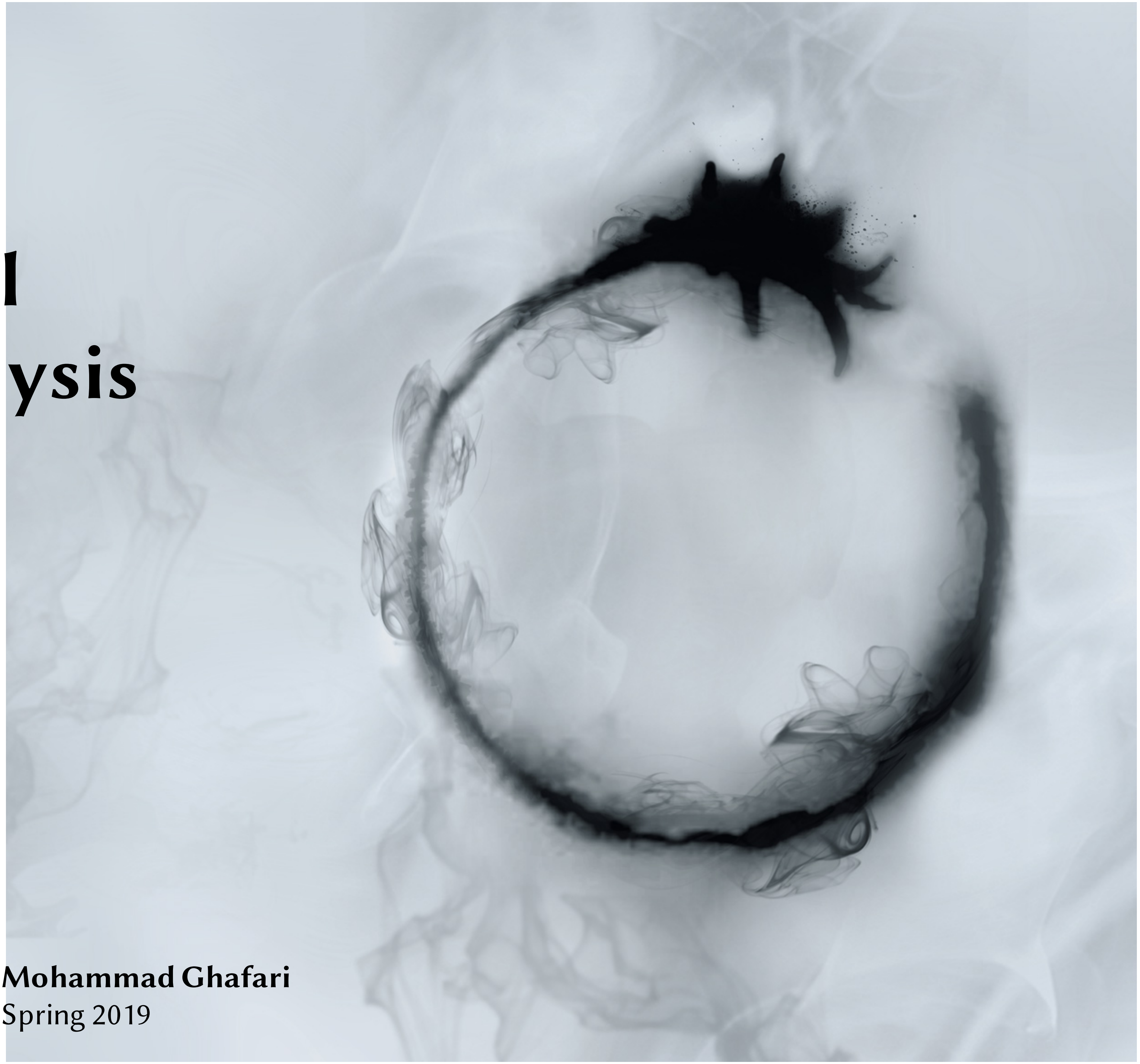


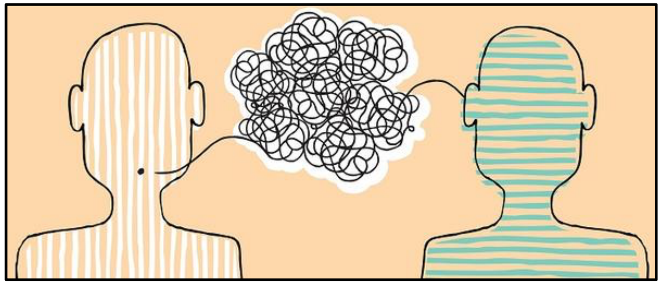
Lexical Analysis

Mohammad Ghafari
Spring 2019

UNIVERSITÄT BERN

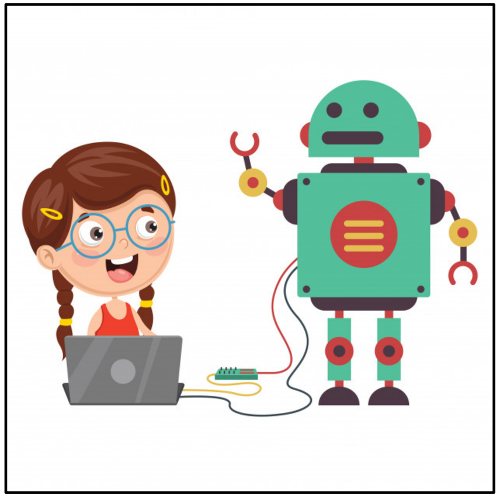
What is a language?

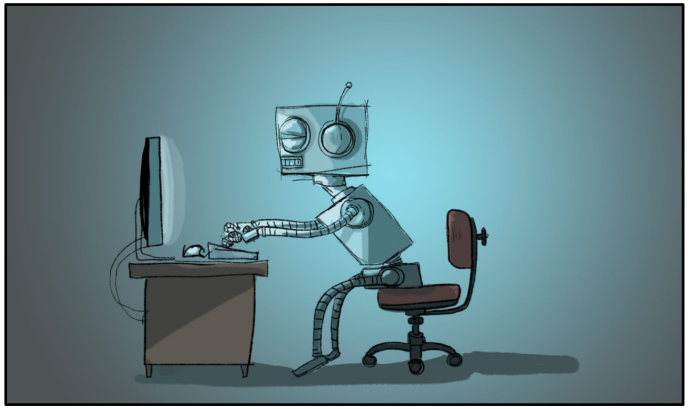
The method of human communication, either spoken or written, consisting of the use of words in a structured and conventional way.



What is a programming language?

The means of communication with machines often written in ASCII characters.





We need a “valid” language

Validity breaks down into syntax and semantics. The former is the arrangement of words, while the latter is the meaning of words.

For example:

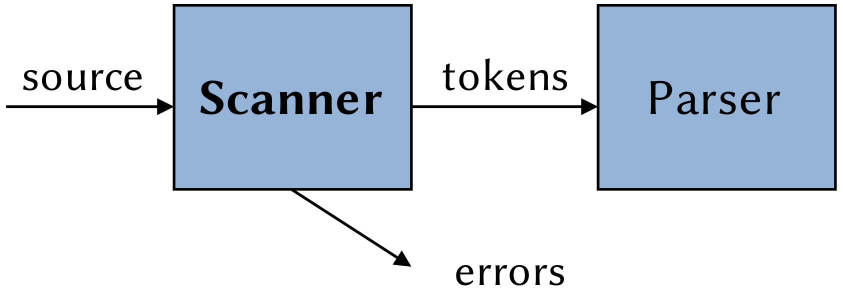
1. The dog the man walks.

2. The dog walks the man.

3. The man walks the dog.

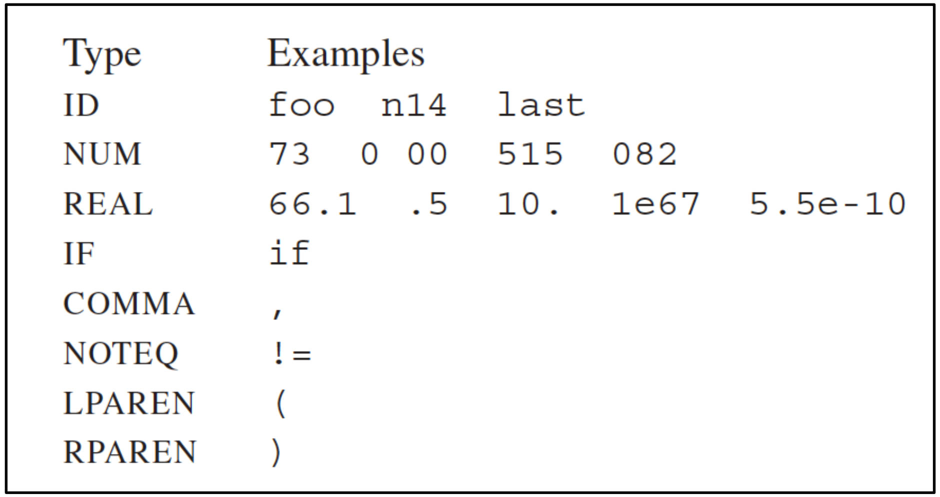
Lexical analysis

The process of mapping sequences of characters to tokens in a particular language.



`x = x + y` → `<ID, x> <EQ> <ID, x> <Plus> <ID, y>`

Typical token types



Type	Examples
ID	`foo` `n14` `last`
NUM	`73` `0` `00` `515` `082`
REAL	`66.1` `.5` `10.` `1e67` `5.5e-10`
IF	`if`
COMMA	`,`
NOTEQ	`!=`
LPAREN	`(`
RPAREN	`)`

NB. Each reserved world like if, void, return, etc. has a dedicated token.

Nontokens are:

- comment,
- blanks, tabs, and newlines,
- etc.

Regular expressions

We use the regular expressions to specify the grammar of a language.

Symbols

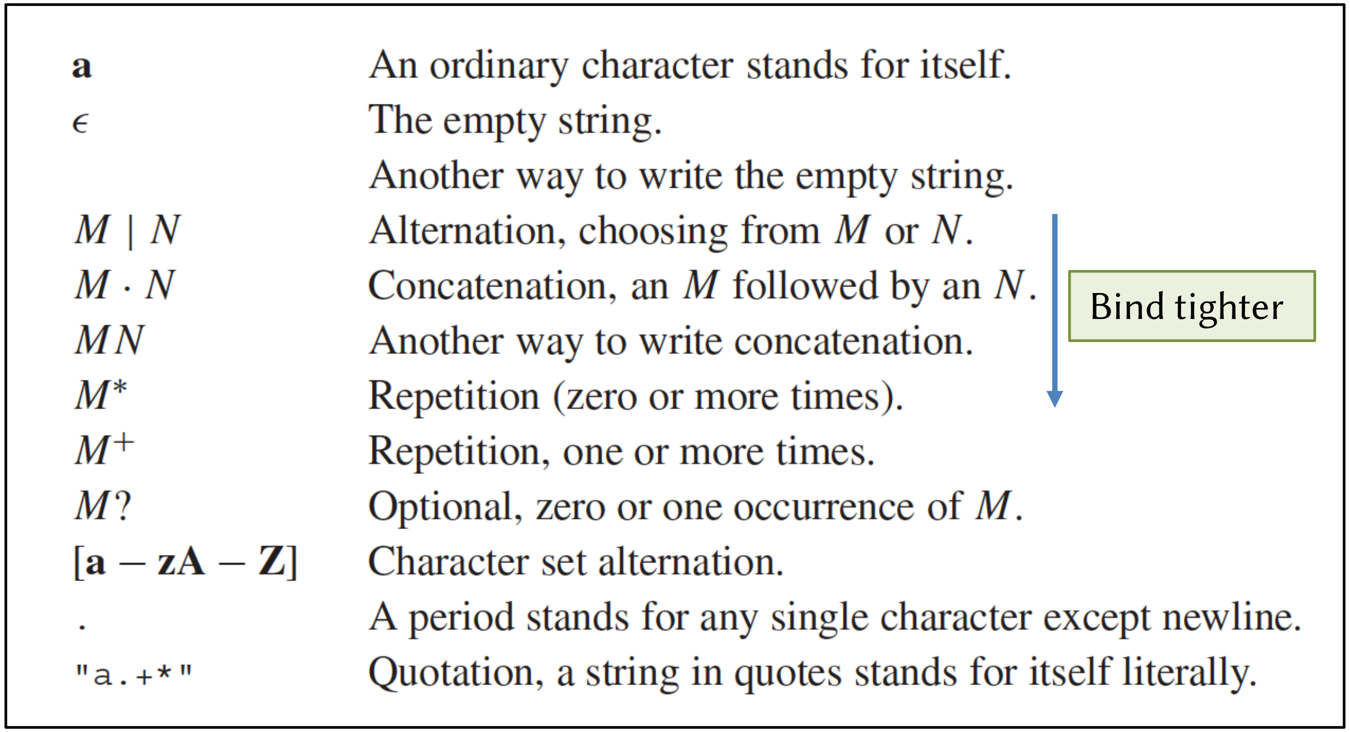
Strings

Language

We can decide whether a string is in the language or not.

Notations

If M and N are the languages, then:



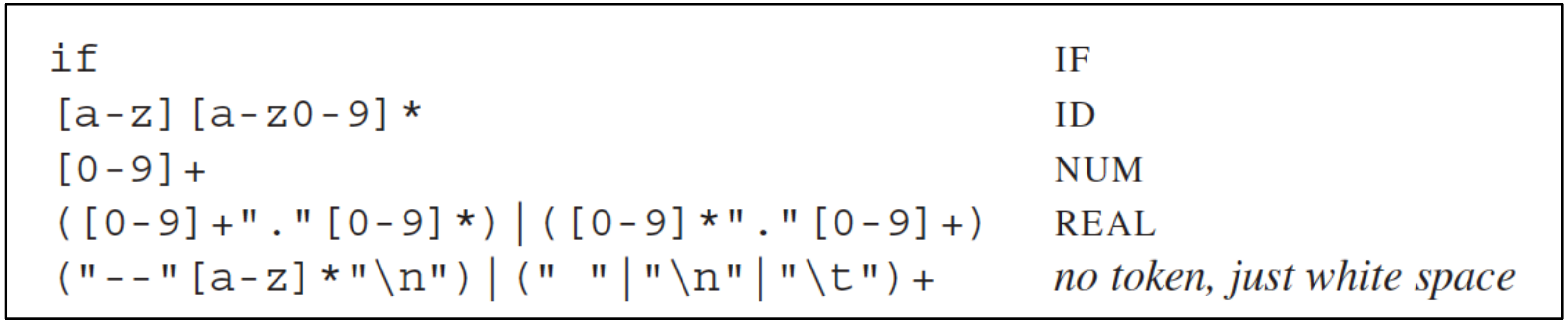
a	An ordinary character stands for itself.
ϵ	The empty string.
	Another way to write the empty string.
$M \mid N$	Alternation, choosing from M or N.
$M \cdot N$	Concatenation, an M followed by an N.
MN	Another way to write concatenation.
M^*	Repetition (zero or more times).
M^+	Repetition, one or more times.
$M?$	Optional, zero or one occurrence of M.
[a − zA − Z]	Character set alternation.
.	A period stands for any single character except newline.
`"a.+*"`	Quotation, a string in quotes stands for itself literally.

Useful extensions:

`[abc]` means `(a|b|c)`

`[d-g]` means `[defg]`

Some examples



`if`	IF
`[a-z][a-z0-9]*`	ID
`[0-9]+`	NUM
`([0-9]+"."[0-9]*)|([0-9]*"."[0-9]+)`	REAL
`("--"[a-z]*"\n")|(" "|"\n"|"\t")+`	*no token, just white space*

How about the followings?

`ab|c`

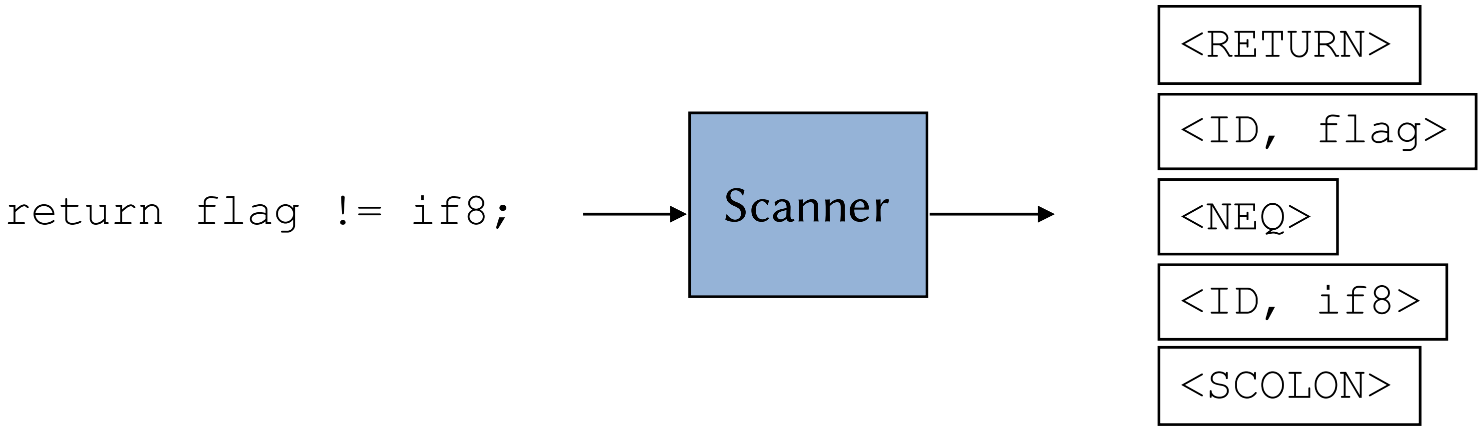
`(a|b)*`

`aa*bb*`

`a*(abb*)*(a|)`

Principle of longest match

Usually, the scanner should pick the longest possible string as the next token.



`return flag != if8;`

Scanner

`<RETURN>`

`<ID, flag>`

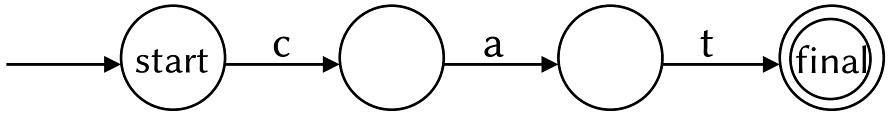
`<NEQ>`

`<ID, if8>`

`<SCOLON>`

Finite state automata

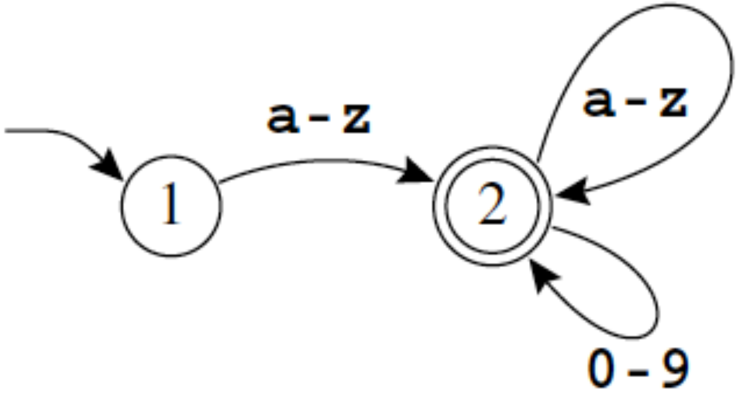
- A finite automaton has a finite set of states; edges lead from one state to another, and each edge is labeled with a symbol. One state is the start state, and certain of the states are distinguished as final states.
- Finite automata are recognizers; they simply say "yes" or "no" about each possible input string.



- They come in two flavors:
  - Nondeterministic finite automata (NFA)
  - Deterministic finite automata (DFA)

Example

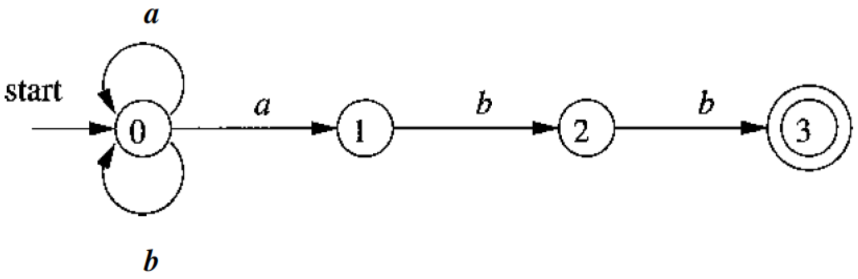
The regular expressions [a-z][a-z0-9]* specifies an identifier.



NFA

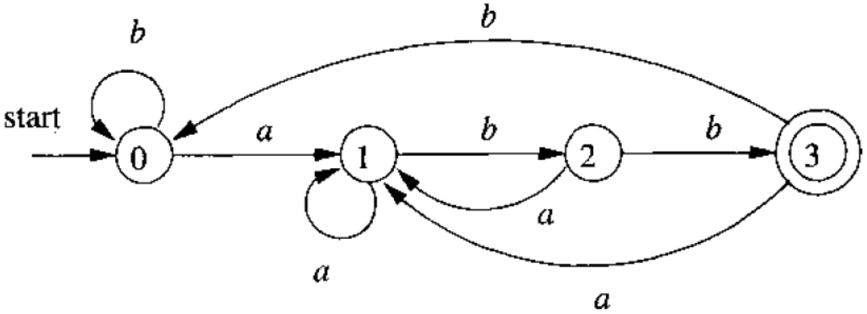
It is an automaton that has a choice of edges – labeled with the same symbol – to follow out of a state. Or it may have special edges labeled with epsilon that can be followed without eating any symbol from the input.

`(a|b)*abb`

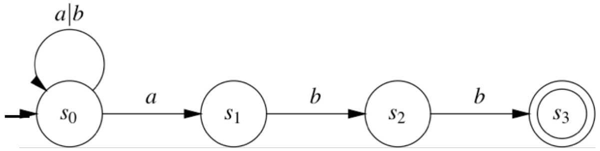


DFA

In this automaton no two edges leaving from the same state are labeled with the same symbol.

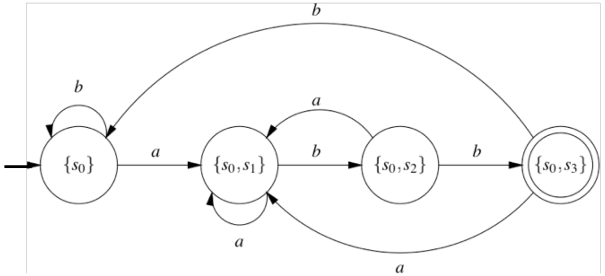


Converting an NFA to a DFA



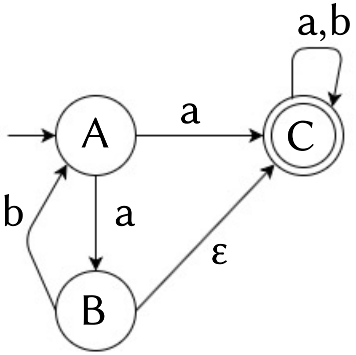
states	a	b
s_0	s_0, s_1	s_0
s_1	0	s_2
s_2	0	s_3
s_3	0	0

states	a	b
s_0	$\{s_0, s_1\}$	s_0
$\{s_0, s_1\}$	$\{s_0, s_1\}$	$\{s_0, s_2\}$
$\{s_0, s_2\}$	$\{s_0, s_1\}$	$\{s_0, s_3\}$
$\{s_0, s_3\}$	$\{s_0, s_1\}$	s_0



Example

- Find the corresponding DFA of the following automaton.



- Draw a DFA that accepts the `aa*bb*` expression.

Compute e-closure

Lets define e-closure (T) as the states reachable from every state in set T on e-transitions.

```
push all sates of T onto stack;
initialize e-closure(T) to T;
while(stack is not empty){
    pop t from the stack;
    for(each state u with an edge from t to u labeled e)
        if(u is not in e-closure(T)){
            add u to e-closure(T);
            push u onto stack;
        }
}
```

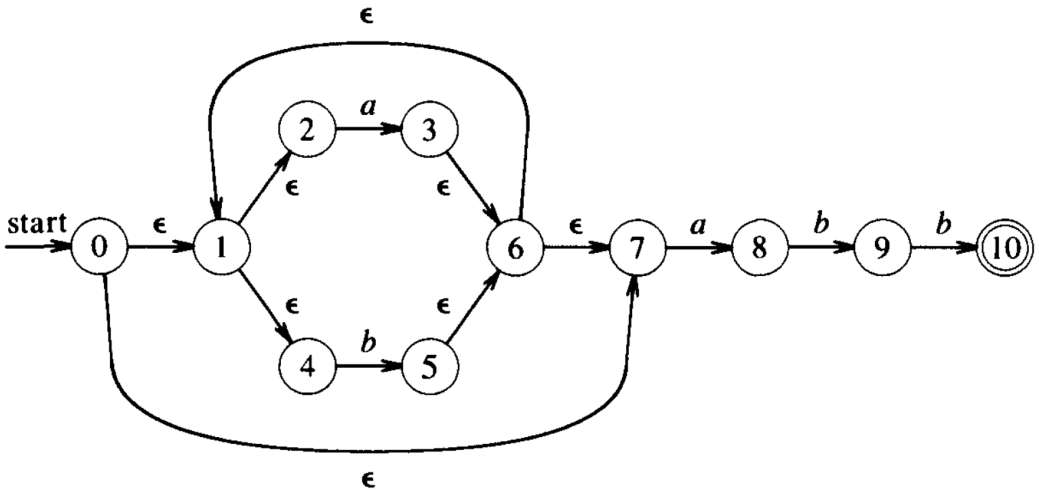
The subset construction

Lets define move(T, a) as set of NFA states to which there is a transition on input symbol “a” from some state s in T.

```
while(there is an unmarked state T in Dstates){
    mark T;
    for(each input symbol a){
        U = e-closure(move(T,a));
        if (U is not in Dstates)
            add U as an unmarked state to Dstates;
        Dtran[T,a] = U;
    }
}
```

Example

Apply the subset construction to the following NFA.

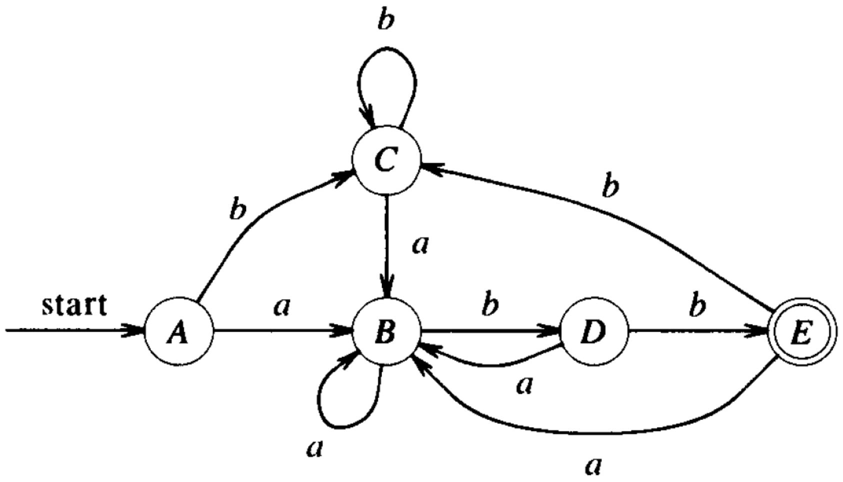


`(a|b)*abb`

Example (answer)

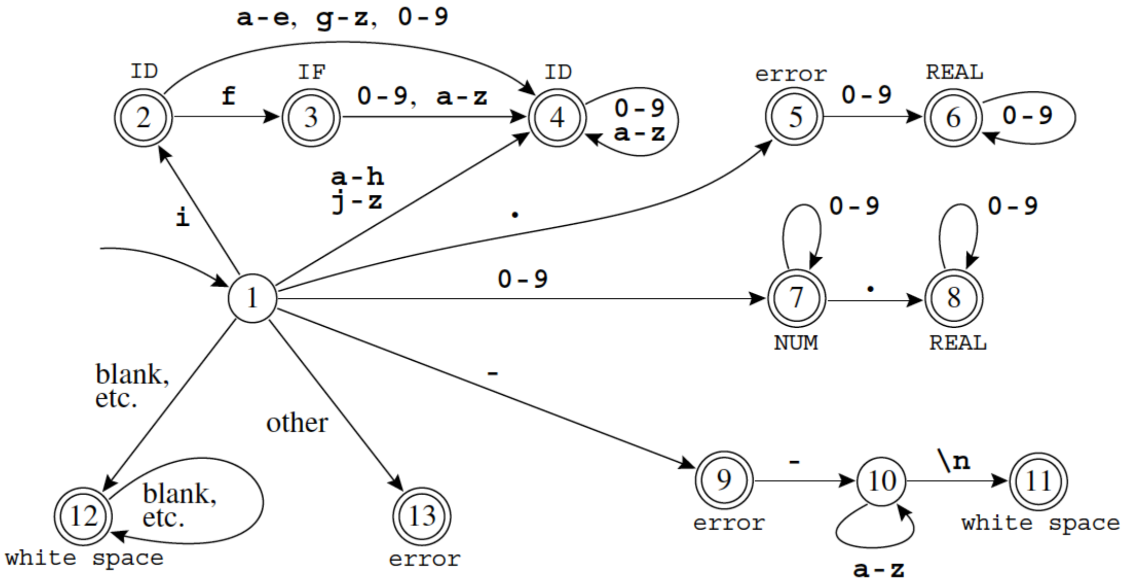
$A = \{0, 1, 2, 4, 7\}$
$B = \{1, 2, 3, 4, 6, 7, 8\}$
$C = \{1, 2, 4, 5, 6, 7\}$
$D = \{1, 2, 4, 5, 6, 7, 9\}$
$E = \{1, 2, 4, 5, 6, 7, 10\}$

STATE	INPUT SYMBOL	
	a	*b*
A	*B*	*C*
B	*B*	*D*
C	*B*	*C*
D	*B*	*E*
E	*B*	*C*

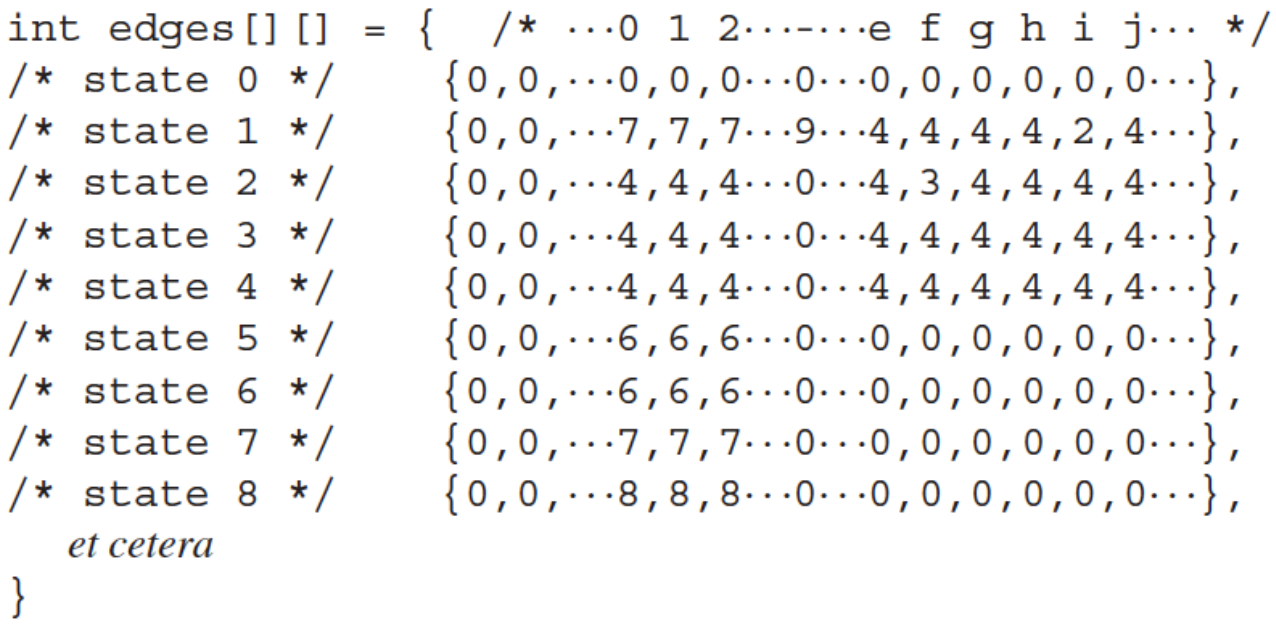


Lexical analyzer

Each automaton accepts a certain token and the combination of several automata can serve as a lexical analyzer (also know as lexer or scanner).



Lexer in practice



```
int edges[][] = {  /* ···0 1 2···-···e f g h i j··· */
/* state 0 */    {0,0,···0,0,0···0···0,0,0,0,0,0···},
/* state 1 */    {0,0,···7,7,7···9···4,4,4,4,2,4···},
/* state 2 */    {0,0,···4,4,4···0···4,3,4,4,4,4···},
/* state 3 */    {0,0,···4,4,4···0···4,4,4,4,4,4···},
/* state 4 */    {0,0,···4,4,4···0···4,4,4,4,4,4···},
/* state 5 */    {0,0,···6,6,6···0···0,0,0,0,0,0···},
/* state 6 */    {0,0,···6,6,6···0···0,0,0,0,0,0···},
/* state 7 */    {0,0,···7,7,7···0···0,0,0,0,0,0···},
/* state 8 */    {0,0,···8,8,8···0···0,0,0,0,0,0···},
    et cetera
}
```

The lexer must keep track of the longest match seen so far, and the input position of that match.

Example

Last Final	Current State	Current Input	Accept Action
0	1	`⌶if --not-a-com`	
2	2	`|i⌶f --not-a-com`	
3	3	`|if⌶ --not-a-com`	
3	0	`|if⊤ ⊥--not-a-com`	*return* IF
0	1	`if⌶ --not-a-com`	
12	12	`if| ⌶--not-a-com`	
12	0	`if| ⊤-⊥-not-a-com`	*found white space; resume*
0	1	`if ⌶--not-a-com`	
9	9	`if |-⌶-not-a-com`	
9	10	`if |-⊤-⊥not-a-com`	
9	10	`if |-⊤-n⊥ot-a-com`	
9	10	`if |-⊤-no⊥t-a-com`	
9	10	`if |-⊤-not⊥-a-com`	
9	0	`if |-⊤-not-⊥a-com`	*error, illegal token '-'; resume*
0	1	`if -⌶-not-a-com`	
9	9	`if -|-⌶not-a-com`	
9	0	`if -|-⊤n⊥ot-a-com`	*error, illegal token '-'; resume*

- | the input position at each call to the lexer.
- ⊥ the current position.
- ⊤ the last final state.

Acknowledgement

- Compilers: Principles, Techniques, and Tools by Alfred V.Aho, Monica S. Lam, Ravi Sethi and Jeffrey D. Ullman.
- Modern Compiler Implementation in Java by Andrew W. Appel and Jens Palsberg.