September, 18th

Gilles GEERAERTS

gilles.geeraerts@ulb.be

Office: Plaine Campus
No building 8th floor
N side, last office

TA: Matthieu Sassolas
Léonard Brice

⚠ Timetable.

Universite Virtuelle webpage

- lecture notes (PDF)
- Slides
- Project

Evaluation

Project → group 2 people
→ 3 parts 8/20

No second chance!

Exam 12/20

"Teaching style"

Lecture notes ≃ text book
complete
mathematically
precise.

Lectures → higher level
interactions.

What is a language?

Vocabulary
Syntax
} what we can write

Syntax

Way to communicate meaning
semantics

"Colorless green ideas sleep furiously"

Syntax ✓

semantics ✗

"We can write a program that compiles but does not do what we want"

Syntax ✓

Semantics ✗

Formal language

Definition: an **Alphabet** is finite set of symbols

$\{a, b, c, \ldots, z\}$: alphabet

$\{0, 1\}$: alphabet

$\mathbb{N}$: not an alphabet

Definition: a **word** is a finite sequence of symbols from the alphabet

Alphabet: $\{0, 1\}$

00110 → word

110 → word

Empty word: ε

Definition A language is a set of words.

Example: Alphabet: $\{0, 1\}$

Language = set of all binary numbers which are even.

$\{0, 10, 100, 110, \ldots\}$

what about programming languages?

alphabet

a ... z { }
A ... Z ( )
0 ... 9 +, −, %
␣ ...

word

program.

language

set of all
syntactically
correct programs

The membership Problem

$w \overset{?}{\in} L$

word (finite)

language (infinite)

1. The set L_{Cid} of all non-empty words on Σ_C (see example 1.7) that do not begin with a digit, is a language. It contains all valid C identifiers (variable names, function names, etc) and all C keywords (`for`, `while`, etc).

2. The set L_{odd} of all non-empty words on $\{0,1\}$ that end with a 1 is a language. It contains all the binary encodings of odd numbers.

3. Similarly to the previous example, the set $L_{()}$ of all words on $\Sigma = \{(,)\}$ which are well-parenthesised, i.e., s.t. each closing parenthesis matches a previously open and still pending parenthesis, and each open parenthesis is eventually closed. For example $(()()) \in L_{()}$, but neither)((nor (() do.

   This language is also known as the *Dyck language*, named after the German mathematician Walter VON DYCK (1856–† 1934). It is mainly of theoretical interest: we will rely on it several times later to discuss the kind of formalism we need to recognise languages of expressions that contain parenthesis, such as the language L_{alg} defined in the next item:

① $\Sigma_C = \{a, b, \ldots, z, A, B, \ldots, Z, 0, \ldots, 9, \_\}$

all the words on Σ_C that do not start with a digit.

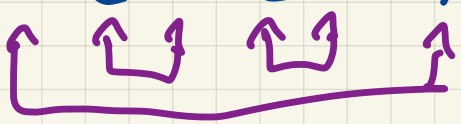
→ easy to check just look at the first symbol

② 0 1 0 1 0 1 easy to check.

③ $\Sigma = \{ (,) \}$

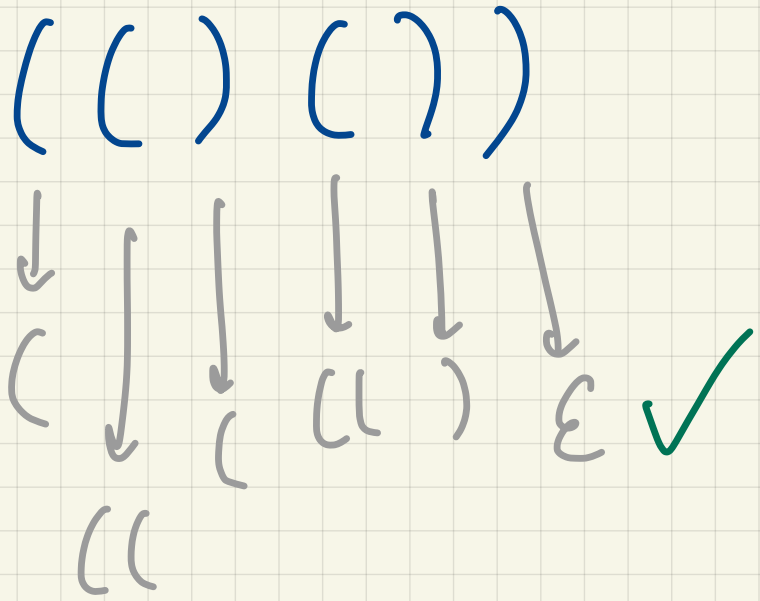
( ) ✓

(( ) ( )) ✓



)(✗

(((✗

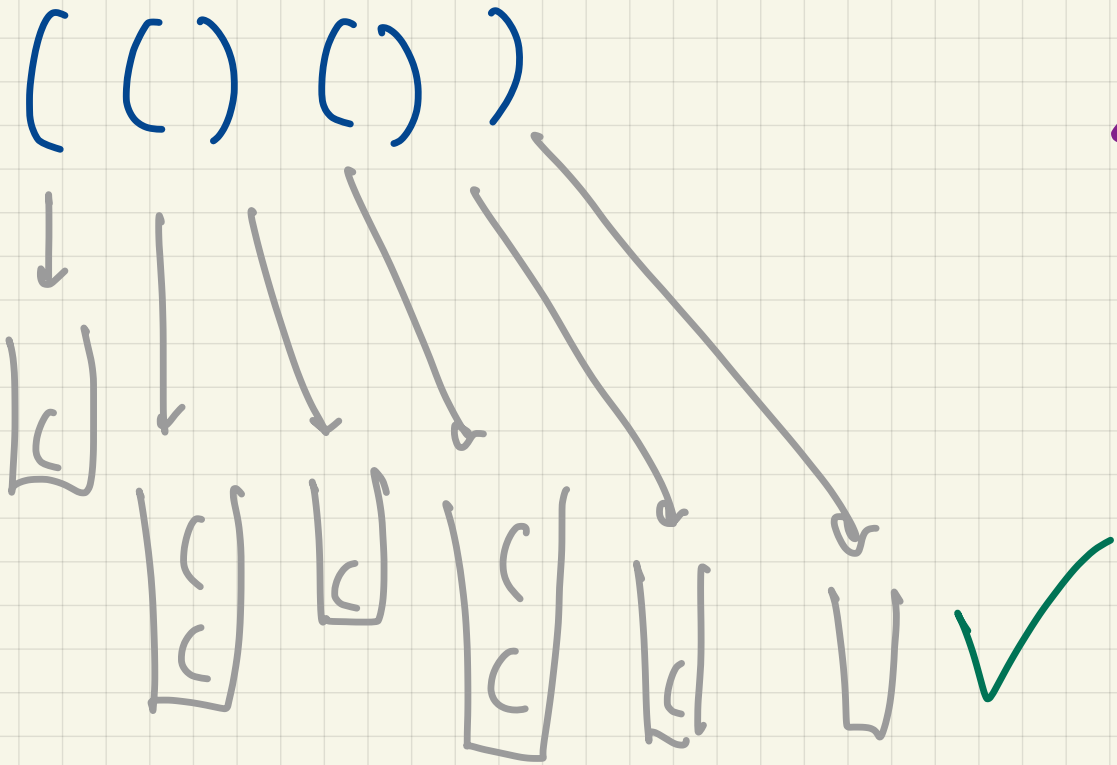


queue version.

(: add it to the queue

) : remove from the queue (if possible) otherwise reject

In the end: empty queue.

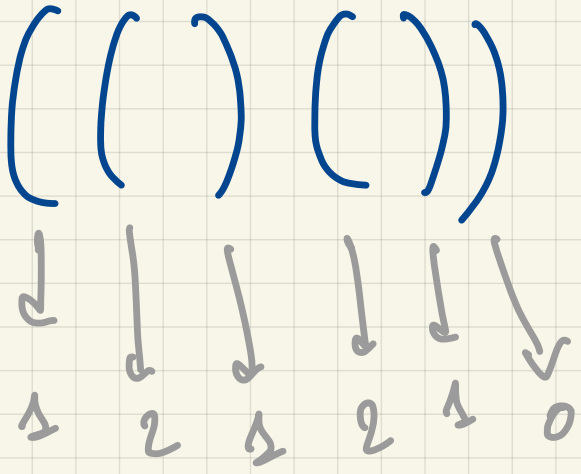


Stack version

(→ Push

) → Pop

Empty at the end.



Counter

(→ +1

) → −1

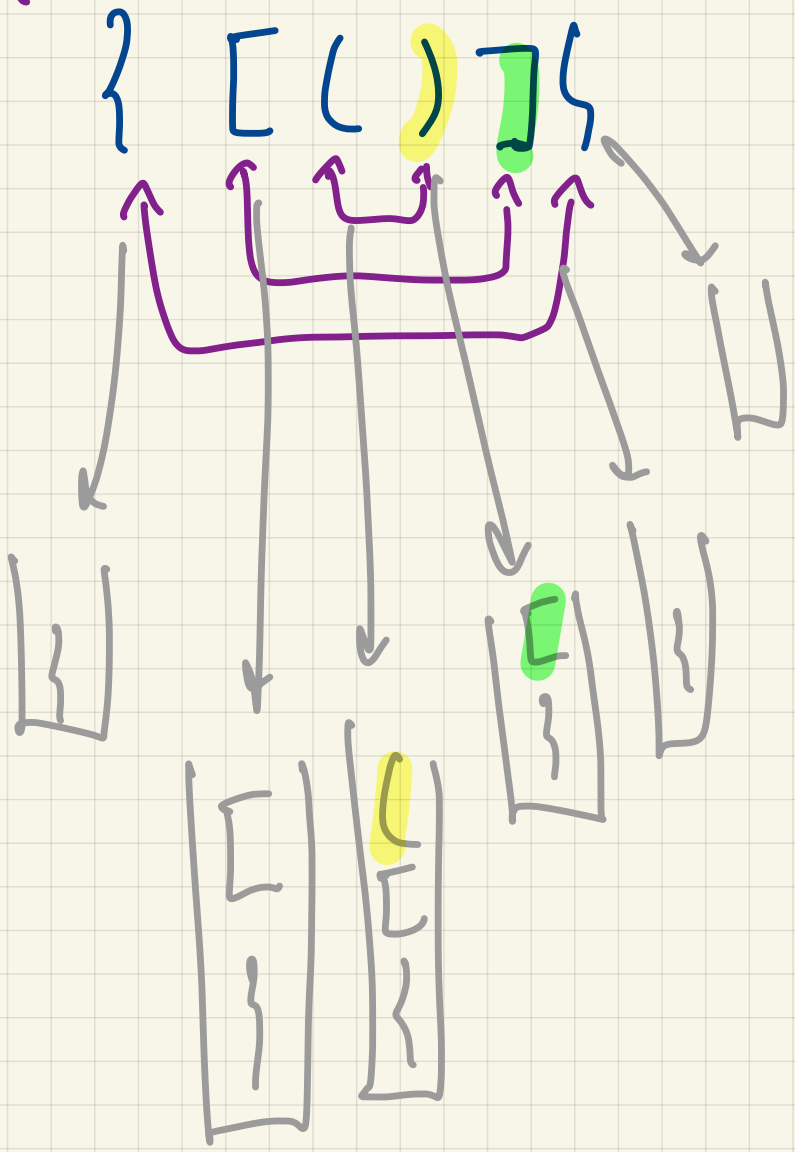
≠ 0 at all times

= 0 in the end

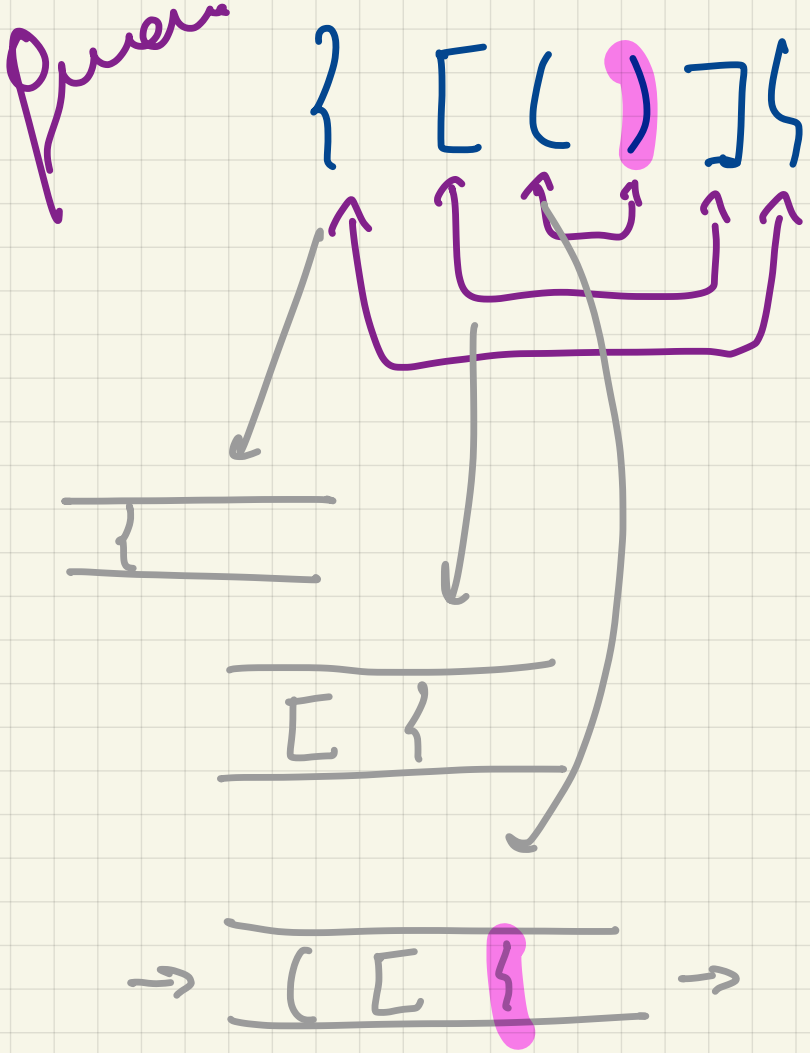
⚠ We need unbounded memory.

What if we have ≠ brackets?

Stack



Queue

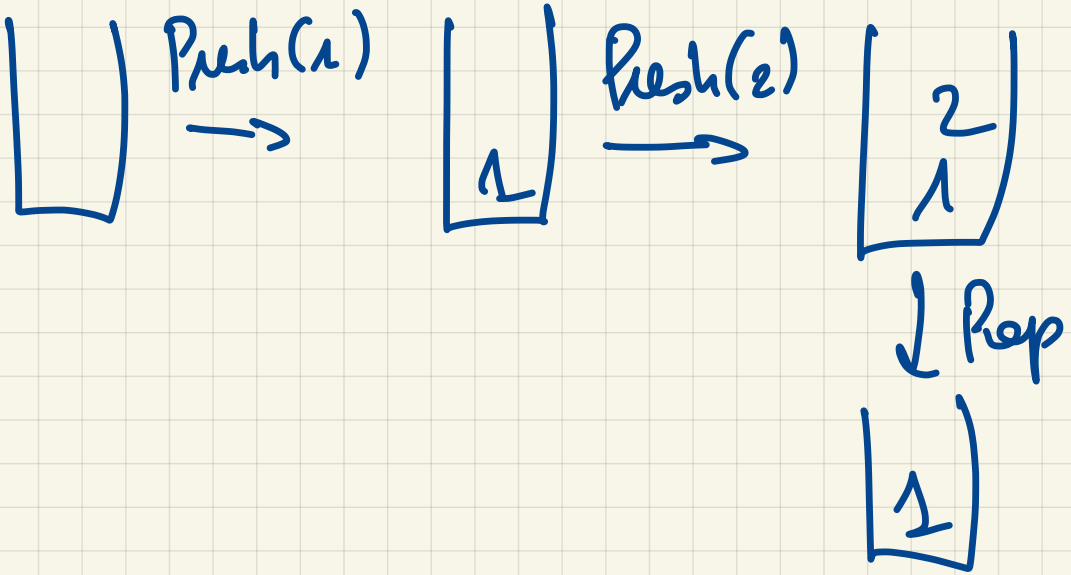


LIFO

Stack Push Pop

Manipulate only the "Top"

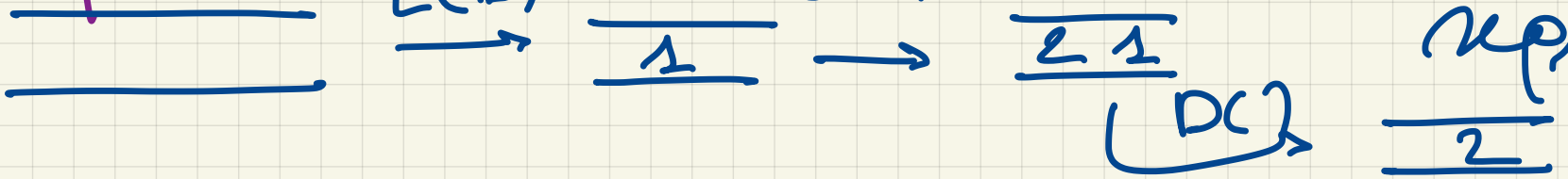
One end of the sequence



FIFO

Queue Enqueue Dequeue

Manipulate ≠ ends of the sequence



More examples ! Yeah !

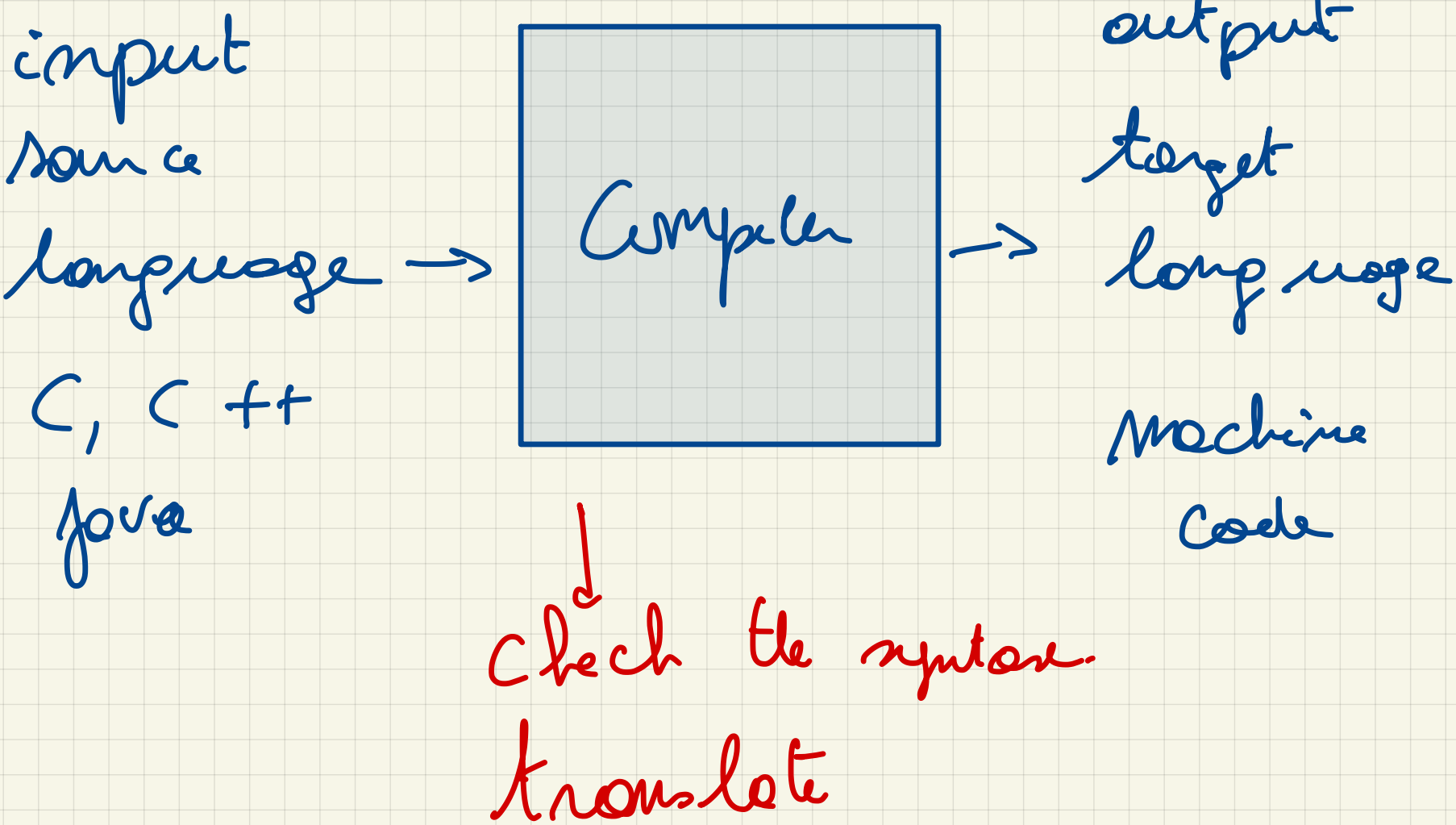
4. The set L_{alg} of all algebraic expressions that use only the x variable, the $+$ and $*$ operators and parenthesis, and which are well-parenthesised, is a language on the alphabet $\Sigma = \{(,), x, +, *\}$. For instance $((x+x)*x)+x$ belongs to this language, while $)(x + x$ does not, although it is a word on Σ.

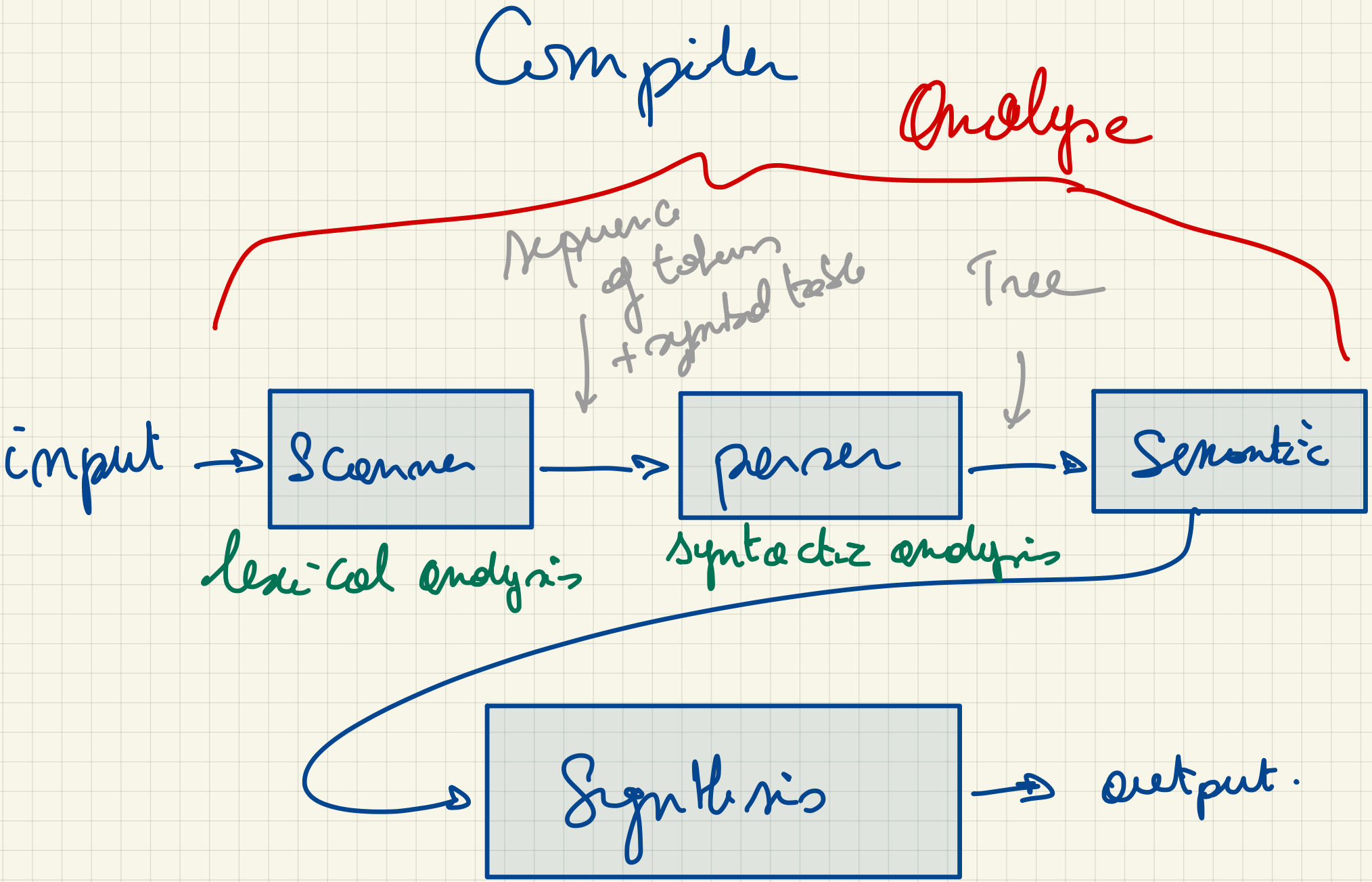
5. The set L_C of all syntactically correct C programs is a language.

6. The set L_{Cterm} of syntactically C programs that terminate whatever the input given by the user is a language.

↳ "bug free"

Compilers


input
source
language
C, C++
Java
Compiler
output
target
language
Machine
Code
check the syntax.
translate


Compiler
Analyse
sequence of token + symbol table
Tree
input
Scanner
parser
Semantic
lexical analysis
syntactic analysis
Synthesis
output.

① Scanning.

```
int i = 5 ;

int f ( int j ) {
    int i = j ;
    return i + 1 ;
}

int main ( ) {
    printf ( "Hello␣World␣!" ) ;
    printf ( "%d␣%d" , i , f ( i + 1 ) ) ;
    return 0 ;
}
```

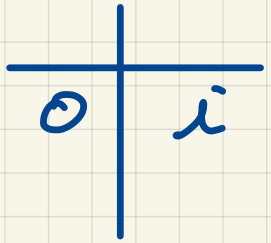
Split the input into tokens.

the scanner can build the symbol table
= list of all identifiers

Example 1.13. Let us consider the simple code excerpt:

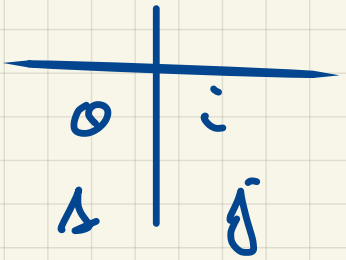
```
1    int i = 5;
2    int j = 3 ;
3    i = 9 ;
```

after line 1



→ return (var, 0)

after line 2



→ return (var, 1)

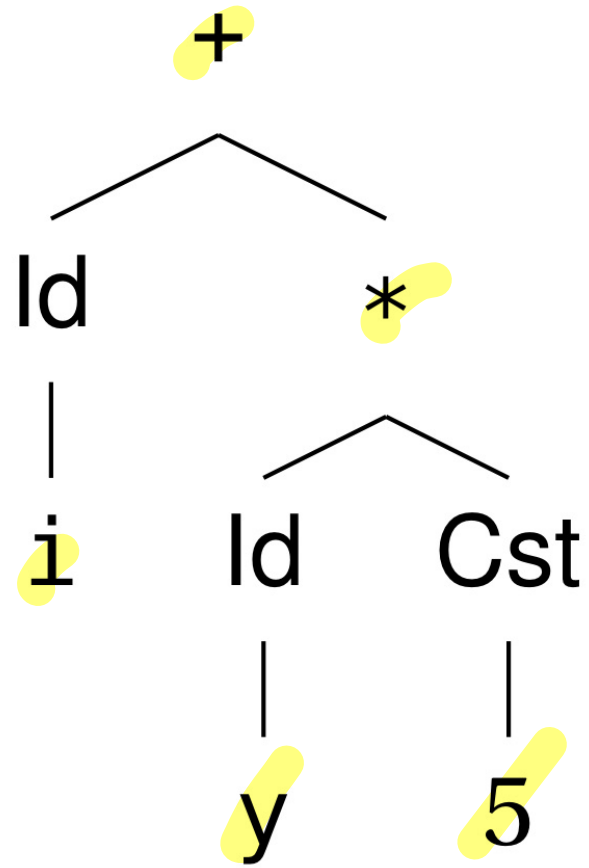
after line 3

// → return (var, 0)

② Parsing

Analysing the global syntax of the program.

d be:



$i + y * 5$

③ Semantic analysis.

→ Scoping / Types

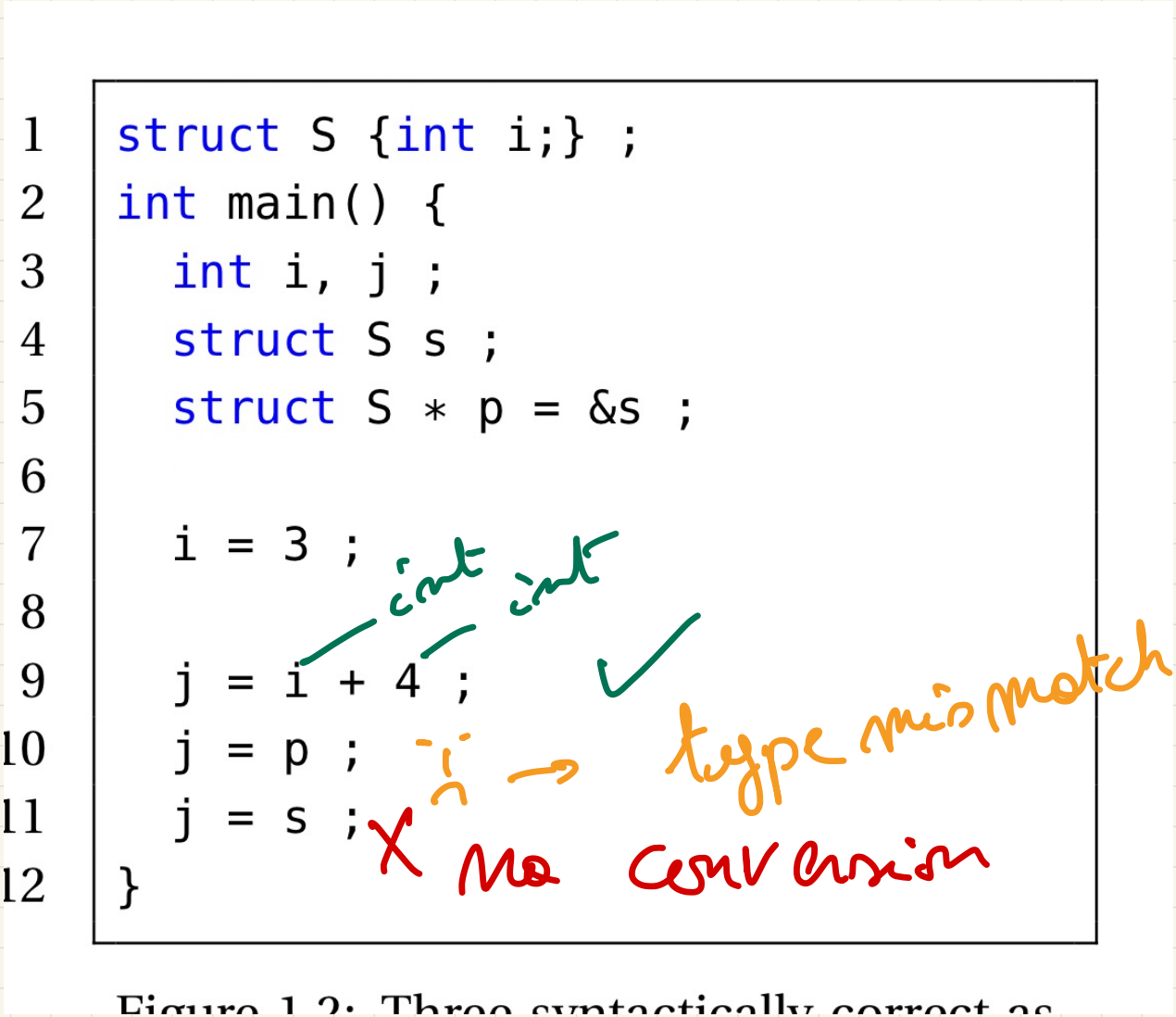


Figure 1.2: Three syntactically correct as

→ Control flow

```
1   #include <iostream>
2
3   int main () {
4     for(int i=1;i<10;++i) {
5       infor: std::cout << i ;
6         std::cout << std::endl ;
7     }
8
9     goto infor ;
10  }
```

i exists outside the scope of the for

Operations on words

$w = w_1 \dots w_n \qquad v = v_1 v_2 \dots v_e$

* Concatenation of w and v

$$w \cdot v = w_1 \dots w_n v_1 v_2 \dots v_e$$

$$abc \cdot de = abcde$$

* Repeated concatenation

$$w^n = \underbrace{w \cdot w \cdot w \dots w}_{n \text{ times}}$$

$$(ab)^3 = ab\,ab\,ab$$

Operations on languages

$$L_1 \cdot L_2 = \{ w_1 \cdot w_2 \mid w_1 \in L_1, \; w_2 \; L_2 \}$$

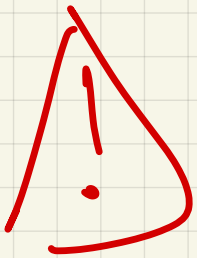
$$\{ab, cd\} \cdot \{e, f\}$$

$$= \{abe, abf, cde, cdf\}$$

?

$$L \cdot \{\varepsilon\} = L = \{\varepsilon\} \cdot L$$

$$L \cdot \phi = \phi$$



ε ↓ empty word

$\{\varepsilon\}$ ↓ non empty language that contains the empty word

ϕ ↓ empty language

$$L^n = \underbrace{L.\ L.\ L \ldots L}_{n \text{ times}}$$

Kleene closure.

$$L^* = \{ w_1 . w_2 \ldots w_n \mid n \geq 0, \forall i : w_i \in L \}$$

$$\{a\}^* = \{ \varepsilon, a, aa, aaa, \ldots \}$$

$\{0,1\}^*$ = all binary words (including ε)

the set of all representations of natural numbers

(in a given base, say 10)

$$\{0, \dots, 9\}^* \setminus \{\varepsilon\}$$

Regular languages

= family of languages

2 kinds of tools

Specification	Machine ≈ code
Regular expressions	Automata (finite)

Definition 2.1 (Regular languages)**.** Let us fix an alphabet Σ. Then, a language L is regular iff:

Base

1. either $L = \emptyset$;
2. or $L = \{\varepsilon\}$;
3. or $L = \{a\}$ for some $a \in \Sigma$;

Inductive

4. or $L = L_1 \cup L_2$;
5. or $L = L_1 \cdot L_2$;
6. or $L = L_1^*$

where L_1 and L_2 are regular languages on Σ.

$\Sigma = \{a, b, c, d, e, f\}$

$L = \{abc, def\}$

(3) $\{a\}$ $\{b\}$ $\{c\}$ $\{d\}$ $\{e\}$ $\{f\}$: R. L.

(5) $\{a\} \cdot \{b\} \cdot \{c\}$ $\{d\} \cdot \{e\} \cdot \{f\}$

(4) $\{abc\} \cup \{def\}$

Corollary: any finite language is regular.

$L = \{w_1 \dots w_n\}$

$= \{w_1\} \cup \{w_2\} \cup \dots \{w_n\}$

↓

can be obtained by concatenation

so is regular

Σ^* = regular language which is infinite

⚠ $L \subseteq \Sigma^* \not\Rightarrow$ L is regular.

Remark $L_{()}$ = long mog. of well-parenthesized word

is not regular!

$$\{(,)\}^* \neq L_{()}$$

$$)(\in \{(,)\}^*$$

New operation on languages!!

$$L^+ = L \cdot L^*$$

$$= \{ w_1 \dots w_n \mid \forall i\ w_i \in L,\ n \geq 1 \}$$

So if L is regular, then L^+ is regular.

Definition 2.3 (Regular expressions)**.** Given a finite alphabet Σ, the following are regular expressions on Σ:

1. The constant $\emptyset$. It denotes the language $L(\emptyset) = \emptyset$.
2. The constant ε. It denotes the language $L(\varepsilon) = \{\varepsilon\}$.
3. All constants $a \in \Sigma$. Each constant $a \in \Sigma$ denotes the language $L(a) = \{a\}$.
4. All expressions of the form $r_1 + r_2$, where r_1 and r_2 are regular expressions on Σ. Each expression $r_1 + r_2$ denotes the language $L(r_1 + r_2) = L(r_1) \cup L(r_2)$.
5. All expressions of the form $r_1 \cdot r_2$, where r_1 and r_2 are regular expressions on Σ. Each expression $r_1 \cdot r_2$ denotes the language $L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$.
6. All expressions of the form r^*, where r is a regular expression on Σ. Each expression r^* denotes the language $L(r^*) = \big(L(r)\big)^*$.

In addition, parenthesis are allowed in regular expressions to group subexpressions (with their usual semantics).

Sometimes we will also use the r^+ notation as a shorthand for $r \cdot r^*$

Base

Inductive

Example

$$l = a + b + c + \dots + z + A + B + \dots + Z$$

$$d = 0 + 1 + 2 + \dots + 9$$

C identifiers

$$l \cdot (l + d)^*$$

Each regular expression represents a regular language
Each regular language can be represented by a regular expression