Formal Languages and Compilers

Quick introduction to lex

About the lab

- Some notes about the course
- This year: short hand form
- Lab site: used to get slides, read last minute communications https://sites.google.com/site/compilerclassunitn/
- Github repository (for code snippets, suggestions are well accepted – use a pull request) https://github.com/LorenzoGramola/LFC2016-2017
- My contact is lorenzom<dot>gramola<at>gmail<dot>com SUBJECT must start with LFCLAB2016:

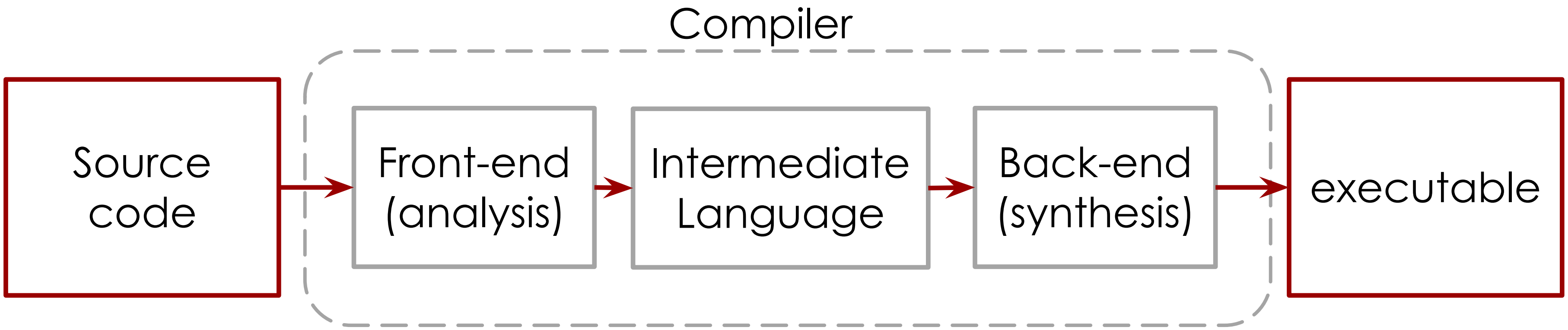
General purpose

- Appling theoretical concept to real problems:
  can this grammar be parsed bottom up?
  is this grammar LARL?
  Does we have conflicts? SR/RR conflicts?
  How to solve them (using powerful tools)
- Knowing Lex
- Knowing Yacc
- Make co-working lex and yacc togheder

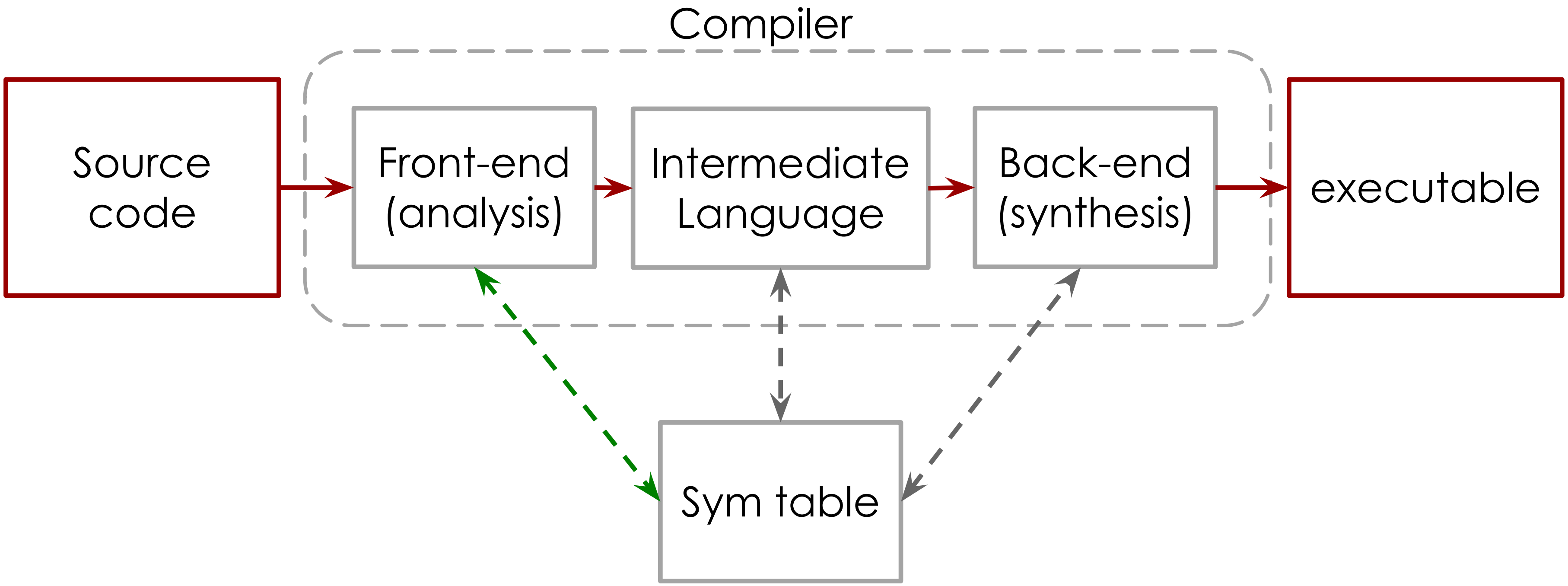
Before starting…

- We shall recap, briefly, how does a compiler/interpreter work…
- Everything starts having a language
  - Grammar
  - Syntax
  - Sematic
  - …
- Which allow us to write source files – hence programs

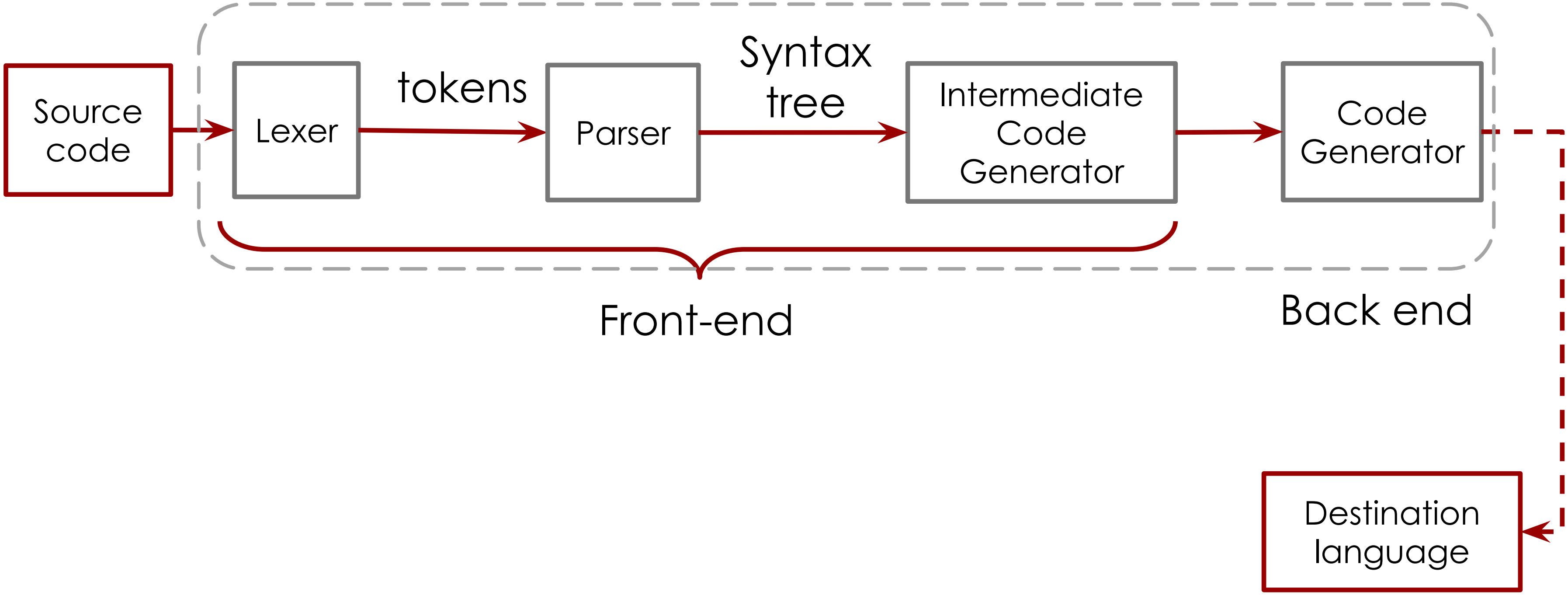
Structure of a compiler



Structure of a compiler



Front-end structure

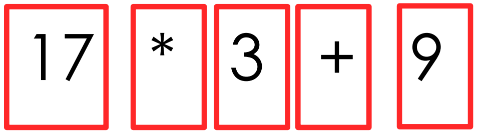


Front-end (cont.)

- The overall process should be able to spot errors and give a feedback to the programmer
- Syntactically incorrect constructs
- Constructs that does not have any semantic meaning
- Signal those errors to the user with the purpose of helping him correcting the code
- So called compilation errors

The lexer

- Is the first step
- Is the program that does the lexical analysis
- Takes in input the source code and identifies the tokens
- Input: source language program
- Output: sequence of token or errors (if chars not recognized as tokens)
- EG: 17 * 3 + 9



17 * 3 + 9

Lexer – tokens

- What are – intuitively – the tokens in the following input?

```
public static void main(String [] args){
        System.out.println(“LFC lexer example”);
}
```

Lexer – errors: java sample

```
class SomeTest{
    public static void main(String[] args) {
        int level = 0;
        System.out.println(”init level = " +level );
        level = randomStuff(level );
        System.out.println(“randomStuff = " + level );
    }

    int randomStuff (int y) {
        y = 5;
        return y;
    }
}
```

At some point during the compilation phase you will get:

- Cannot make a static reference to the non-static method

LEX

- Lex is the tool we will use for our purposes
- Lex is a program generator designed for lexical processing of character input streams.
- Nowadays substituted by Flex
- Reg exp are used for lesseme matching

Structure of the Lex input file

```
Declarations

%%

Patterns

%%

Functions
```

Structure of the Lex file

word count example

```
%{
int charCount = 0, wordCount = 0, lineCount = 0;
%}
word [^ \t\n]+

%%

{word}          {wordCount++; charCount += yyleng; }
[\n]            {charCount++; lineCount++; }
.               {charCount++; }

%%
main() {
  yylex();
  printf("Characters: %d Words: %d Lines %d\n",
                                    charCount, wordCount, lineCount);
}
```

Reg exp for describing patterns

- How are patterns described?

‘a’
“string”
.
[a-z]
Expr*
Expr+
Expr1 | expr2
Expr1 expr2

Any guess or intuition about the meaning of these regular expression?

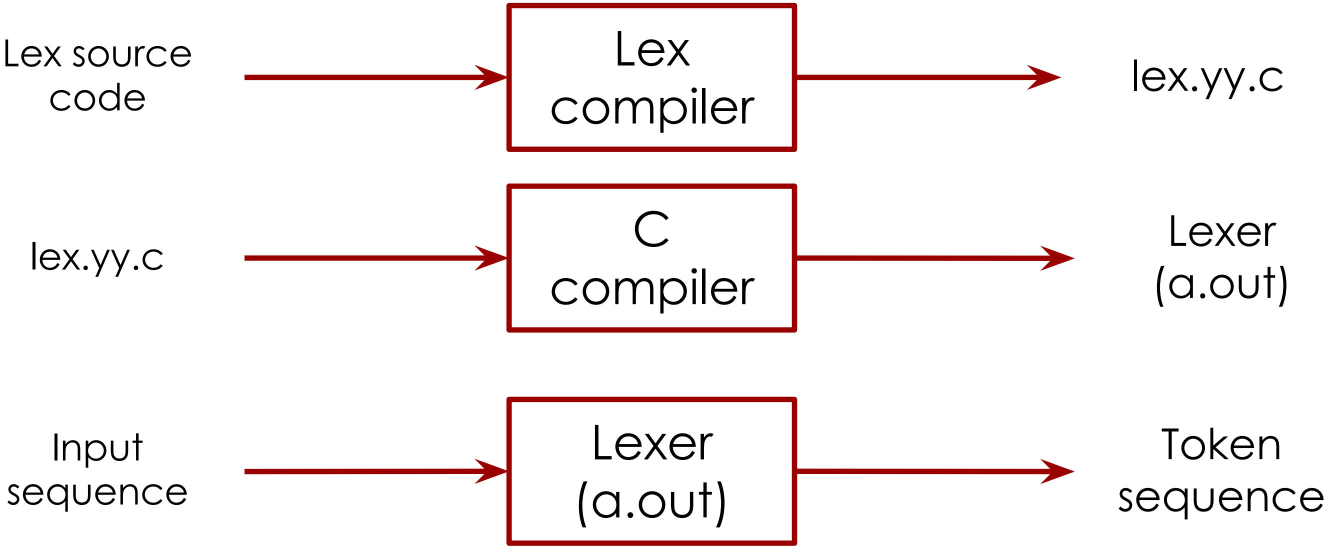
Reg exp for describing patterns

- How are patterns described?

‘a’	simple char
“string”	string
.	Any char
[a-z]	set of chars from a to z
Expr*	kleene star – kleene closure
Expr+	= (expr)expr* - one or more time expr (but not zero)
Expr1 | expr2	either expr1 or expr2
Expr1 expr2	expr1 followed by expr2

Generating a lexer with LEX

We can now generate a simple lexer, just using (f)lex



Let’s practice

- Proceed with the two simple example
- One that spots the number of chars and lines
- One that count the words
- At home: write a lex file that read and sum all the number in a file
- To begin write the file.l then use lex
- Lex –o file.yy.cc file.l (alternative way involves the usage of -t)
- Cc –c –o count.o count.c (mandatory to put -c)
- Cc –o counter count.o –ll (mandatory to link yet)

Bibliography

- Compilers 2nd edition – Aho, Lam, Sethi, Ullman
- Lex − A Lexical Analyzer Generator *M. E. Lesk and E. Schmidt*