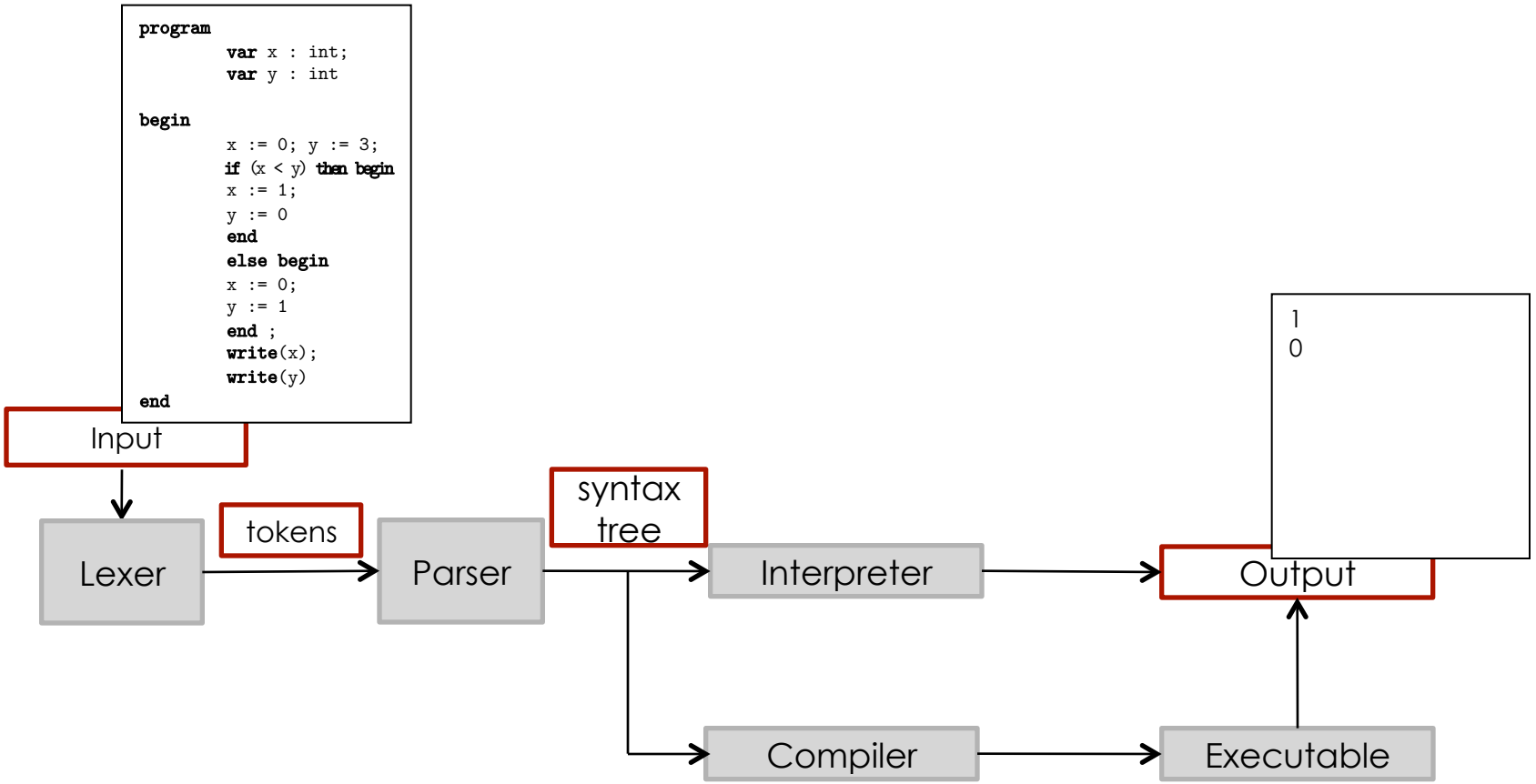
Compiler

Lecture 10

Formal Languages and Compilers 2011

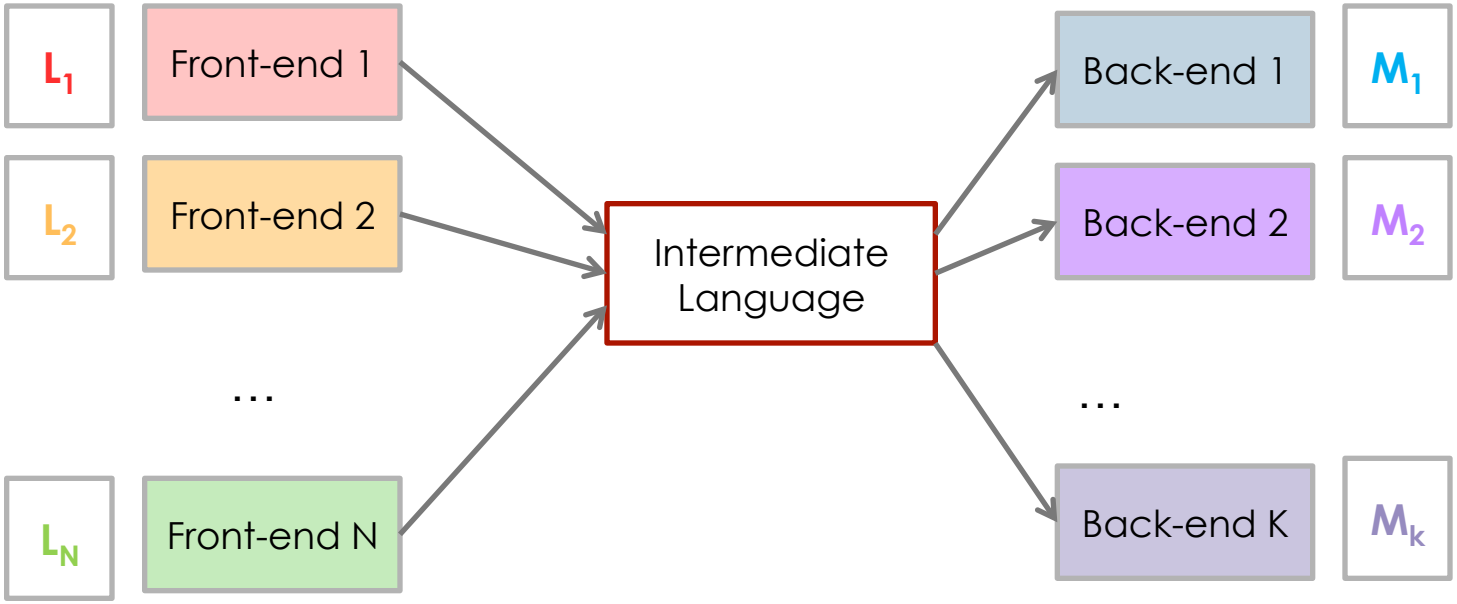
Nataliia Bielova

Compiler: from syntax tree to target code



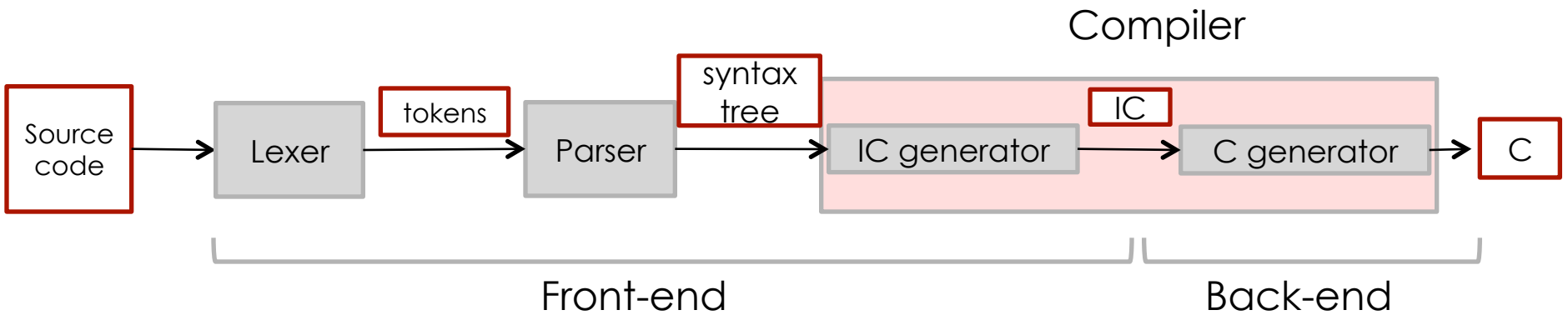
Structure for a compiler

Front-end and back-end



- Intermediate language is *platform-independent*
- Reuse the same front-end for different machines
- Reuse the same back-end for different source languages

Compiler for crème CAraMeL



Compiler for crème CAraMeL

Compiler:

http://disi.unitn.it/~bielova/flc/exercises/10-Compiler.zip

- `make`
- `./compiler < input/test_0.cre`
- [your preferred editor] `output/syntax.ast`
- [the same editor] `output/code.ic`
- [C compiler] `output/code.c`

How the compiler is made

- `input/` file of input
- `output/` file produced by compilation
- `Documentation/index.html` documentation of the compiler
- `script.sh` script for automatic testing
- `main.ml` main program
- `lexer.mll` definition of lexical analyzer
- `parser.mly` definition of syntactic analyzer
- `syntaxtree.ml` definition of the syntax tree
- `print_syntaxtree.ml` functions for printing the syntax tree

How the compiler is made (2)

- `exceptions.ml` exceptions used at the global level
- `declaration.ml` generalization of the table of variables ("environment")
- `semantic.ml` type control
- `expressions.ml` translation of the evaluation of expressions
- `commands.ml` translation of the commands
- `subroutines.ml` translation of the subprograms
- `intermediate.ml` definitions of the intermediate code
- `print.ml` printing intermediate code
- `target.ml` generalization (and output) of target code

Fazes of the compiler

Syntax tree

Type control

Generation of intermediate code

Generation of target code

Source code ⇒ abstract syntax tree


```
program
    var x : int

begin
    x := 3;
    write(x)
 end
```

⇒

```
Program(
    Dec(Ide(x), Basic(Int))

Blk(
    Ass(LVar(Ide(x)), N(3))
    Write(Var(Ide(x)))
    )
)
```

Abstract syntax tree⇒ intermediate code


```
Program(
    Dec(Ide(x), Basic(Int))

Blk(
    Ass(LVar(Ide(x)), N(3))
    Write(Var(Ide(x)))
    )
)
```

⇒

```
CPY     Val INT: 3  NULL    offset 0
OUT     offset 0    NULL    NULL
NOP     NULL        NULL    NULL
HALT    NULL        NULL    NULL
```

Intermediate language

- Is independent from the target language
- Is easy to translate to the target language (effectiveness)
- Code with 3 components:

$$\underbrace{\texttt{NAME}}_{\text{operator}} \quad \underbrace{\texttt{ind}_1}_{\text{operand}_1}, \quad \underbrace{\texttt{ind}_2}_{\text{operand}_2}, \quad \underbrace{\texttt{ind}_3}_{\text{operand}_3}$$

- Examples:

```
ADD     val1    val2    dest
CPY     src     NULL    dest
GOTO    label   NULL    NULL
JNE     val1    val2    label   - Jump if Not Equal : if val1 != val2 goto label
CGE     val1    val2    dest    - Copy Greater or Equal: dest = (val1 >= val2 )
OUT     val     NULL    NULL
```

Temporal symbols

- One simple operation at a time → "cut" the expressions
- Every intermediate result should be saved in the register

```
x := (3 + 7) * 11;   ⇒   ADD   Val INT: 3   Val INT: 7     reg[2].i
                         MUL   reg[2].i     Val INT: 11    reg[1].i
                         CPY   reg[1].i     NULL           offset 0
```

Labels

- Label ≅ "destination of the jump"
- Example:

```
        CPY     Val INT: 1      NULL        offset 0
        CPY     Val INT: 5      NULL        offset 2
        CPY     Val INT: 1      NULL        offset 1
Label2: CGE     offset 2        offset 1    reg[1].i
        JNE     reg[1].i        Val INT: 1  Label nr. 1
        OUT     offset 1        NULL        NULL
        MUL     offset 0        offset 1    reg[2].i
        CPY     reg[2].i        NULL        offset 0
        ADD     offset 1        Val INT: 1  reg[3].i
        CPY     reg[3].i        NULL        offset 1
        NOP     NULL            NULL        NULL
        GOTO    Label nr. 2     NULL        NULL
Label1: OUT     offset 0        NULL        NULL
        NOP     NULL            NULL        NULL
        HALT    NULL            NULL        NULL
```