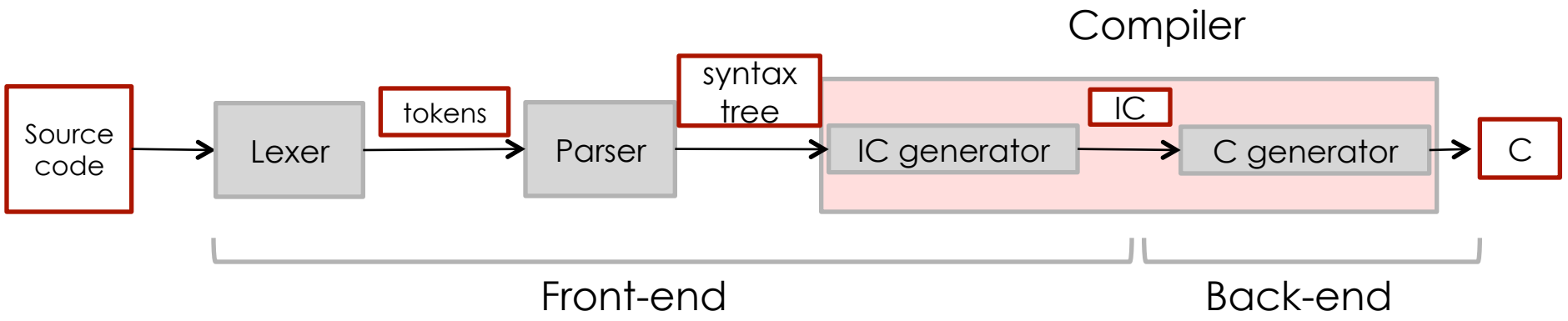
Intermediate language, array and subprograms

Lecture 11

Formal Languages and Compilers 2011

Nataliia Bielova

Compiler for crème CAraMeL



Intermediate language

ADD	val_1	val_2	dest	-	sum
CPY	src	NULL	dest	-	copy
CGE	val_1	val_2	dest	-	copy greater or equal
GOTO	label	NULL	NULL	-	unconditional jump
JNE	val_1	val_2	label	-	conditional jump
OUT	val	NULL	NUL	-	print
AGET	addr	idx	dest	-	read array
ASET	addr	idx	src	-	write array
PARAM	val	NULL	NULL	-	add a parameter on the stack
CALL	id	NULL	NULL	-	call a procedure
CALL	id	NULL	dest	-	call a function

Implementation details

- Memory cells: union of int and float
- Two different vectors: stack and "registers"
- Allocation of variables: assignment of offset in the stack
- Allocation of temporal values: assignment of a new register

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
```

Intermediate.ml

- Define the instructions of intermediate code and all types of operands:
  - inst_type: `ADD, MUL, CPY,...`
  - label, offset for variables, register for temporal values
- `class intermediateCode`
- `dec_table` - declaration table binds `ide` with `(int, int, element)`

Exercise

- Note: we do not allow arithmetical operations of mixed types
- But so far `write(…)` command can contain any expression:

```
write(2 + 5.2)
```

  which should not be allowed
- How to fix it?

Vectors and matrices in crème CAraMeL: compilation


```
var m : array[5] of int;
var v : array[3,2] of int

...

for i := 0 to 2 do begin
    for j := 0 to 1 do begin
            v[i,j] := i + j
    end
end
```

Vectors and matrices in the compiler

- Declaration in style of C:

```
var v : array[4,2] of int
```

- Access like before:

```
v[2,1] := 45;
```

- No V.O. (or better, V.O.= α)
- Simplifies the multiplies

Vectors and matrices in the compiler

- **Declaration:** add dimensions to the declaration table
- **Semantic control**: `v[i,j]` is OK ⇔ `i` and `j` are integers and within the bounds
- **Evaluate expression**: calculate the position + `AGET`
- **Assignment**: calculate the position + `ASET`

Subprograms in crème CAraMeL: compilation


```
program
   var x : int

   function fact(a: int): int
      var b : int
   begin
      if (a = 0) then
         fact := 1
      else begin
         b := call fact(a - 1);
         fact := a * b
        end
   end

begin
   x := call fact(12);
   write(x)
end
```

⇒

479001600

Subprograms in the compiler

- Syntax: the same as in the interpreter
- Table of subprograms
- Managing stack pointer and base pointer
- Call: push on the stack (param) + call
- Using one register for the return of the functions

Subprograms in the compiler

- Declaration: Building and Subroutine (return type of the functions)
- Generation of the code: subroutines.ml
- Parameters and local variables: stack!
- Call: commands.ml and expressions.ml