Formal Languages and Compilers - Exercises
Lecture 5
crème CAraMeL Interpreter

28/03/2012

Outline

1 Definition

2 Elements of interpreter

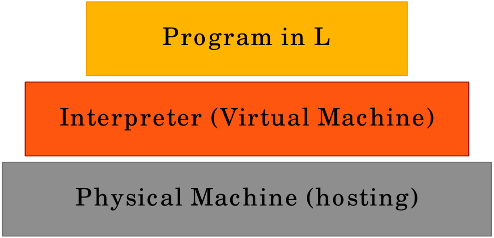
3 Semantic analysis

4 Example

5 Exercise

Definition

Intepreter for a language L



Program in L

Interpreter (Virtual Machine)

Physical Machine (hosting)

crème CAraMeL

- Basic types: `int` and `float`
- Flow control: `if-then-else`, `while`, `while-do`, `for`
- Arithmetic operators: `+`, `-`, `*`, `/`
- Assignment: `:=`
- Relational operators: `=`, `<`, `<=`
- Boolean operators: `&`, `|`, `!`
- Utility: `write( val )`

Objective

Construct an interpreter for the language crème CAraMeL

```
program
  var x : int;
  var y : int
begin
  x := 0;
  y := 3;
  if (x < y) then begin
      x := 1;
      y := 0
    end
  else begin
      x := 0;
      y := 1
    end;
  write(x);
  write(y)
end
```

Result

```
1
0
```

Outline

1 Definition

2 Elements of interpreter

3 Semantic analysis

4 Example

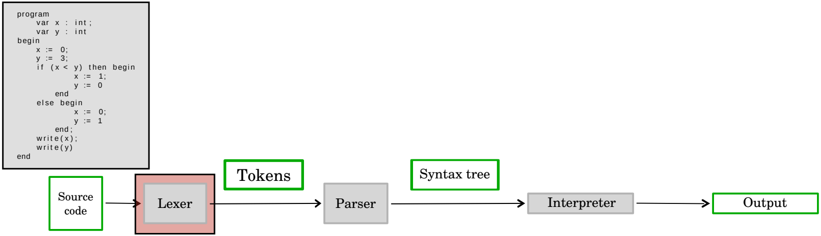
5 Exercise

Elements of interpreter

Lexer

in: source code

out: token



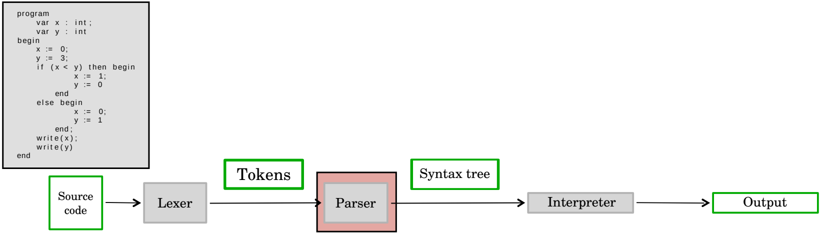
```
program
    var x : int;
    var y : int
begin
    x := 0;
    y := 3;
    if (x < y) then begin
            x := 1;
            y := 0
        end
    else begin
            x := 0;
            y := 1
        end;
    write(x);
    write(y)
end
```

Elements of interpreter

Parser

in: token

out: abstract syntax tree (a.s.t.)



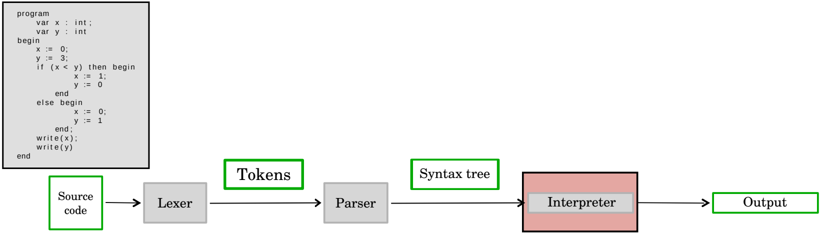
```
program
    var x : int;
    var y : int
begin
    x := 0;
    y := 3;
    if (x < y) then begin
            x := 1;
            y := 0
        end
    else begin
            x := 0;
            y := 1
        end;
    write(x);
    write(y)
end
```

Elements of interpreter

Interpreter

in: abstract syntax tree

out: output



```
program
    var x : int;
    var y : int
begin
    x := 0;
    y := 3;
    if (x < y) then begin
            x := 1;
            y := 0
        end
    else begin
            x := 0;
            y := 1
        end;
    write(x);
    write(y)
end
```

Base of the interpreter - 1

Download the source code from the website

Structure

- Definition of the lexer: `lexer.mll`
- Definition of the parser: `parser.mly`
- Definition for a.s.t: `syntaxtree.ml`
- Definition of the interpreter: `interpreter_base.ml`
- Main program: `main.ml`

Base of the interpreter - 1

Download the source code from the website

Structure

- Definition of the lexer: `lexer.mll`
- Definition of the parser: `parser.mly`
- Definition for a.s.t: `syntaxtree.ml`
- Definition of the interpreter: `interpreter_base.ml`
- Main program: `main.ml`

Base of the interpreter - 1

Download the source code from the website

Structure

- Definition of the lexer: `lexer.mll`
- Definition of the parser: `parser.mly`
- Definition for a.s.t: `syntaxtree.ml`
- Definition of the interpreter: `interpreter_base.ml`
- Main program: `main.ml`

Base of the interpreter - 1

Download the source code from the website

Structure

- Definition of the lexer: `lexer.mll`
- Definition of the parser: `parser.mly`
- Definition for a.s.t: `syntaxtree.ml`
- Definition of the interpreter: `interpreter_base.ml`
- Main program: `main.ml`

Base of the interpreter - 1

Download the source code from the website

Structure

- Definition of the lexer: `lexer.mll`
- Definition of the parser: `parser.mly`
- Definition for a.s.t: `syntaxtree.ml`
- Definition of the interpreter: `interpreter_base.ml`
- Main program: `main.ml`

Base of the interpreter - 2

Compilation

- `make all` compiles everything
- `make clean` cleans from the compiled files
- `./interpreter_base` starts the interpreter (input from console)
- `./interpreter_base < input/test_1.cre` interprets the input from test 1

How the interpreter is made

parser.mly	definition of tokens
lexer.mll	regular expressions and creation of tokens
syntaxtree.ml	declarations of types for the syntax tree
parser.mly	language grammar, creation of the syntax tree
main.ml	starts lexer, parser, executes syntax tree
interpreter_base.ml	functions for the execution of the syntax tree

Outline

1 Definition

2 Elements of interpreter

3 Semantic analysis

4 Example

5 Exercise

Definition of the memory and environment

Formal definition

- *Store*: $\mathrm{Loc} \longrightarrow \mathrm{Val}$

  type store = loc -> value

- *Env*: $\mathrm{Id} \longrightarrow (\mathrm{Loc} \cup \mathrm{Val})$

  type env = ide -> env_entry

Updating the memory

$$\mathrm{updatemem}(s, l, v)(x) = \begin{cases} v, & \text{if } x = l \\ s(x), & \text{if } x \neq l \end{cases}$$

```
let updatemem ((s:store), addr, (v:value)): store =
    function x -> if (x = addr) then v else s(x)
```

Definition of the memory and environment

Formal definition

- *Store*: Loc $\longrightarrow$ Val

  ```
  type store = loc -> value
  ```

- *Env*: Id $\longrightarrow$ (Loc $\cup$ Val)

  ```
  type env = ide -> env_entry
  ```

Updating the memory

$$\text{updatemem}(s, l, v)(x) = \begin{cases} v, & \text{if } x = l \\ s(x), & \text{if } x \neq l \end{cases}$$

```
let updatemem ((s:store), addr, (v:value)): store =
  function x -> if (x = addr) then v else s(x)
```

Arithmetic and boolean expressions: evaluation

Arithmetic expressions

$$E : \text{AExp} \times \text{Env} \times \text{Store} \longrightarrow \text{Val}$$

$$E\|\text{Sum}(n_1, n_2)\|_{r,s} = E\|n_1\|_{r,s} + E\|n_2\|_{r,s}$$

$$E\|i\|_{r,s} = \begin{cases} s(r(i)), & \text{if } r(i) \in \text{Loc} \\ r(i), & \text{if } r(i) \in \text{Val} \end{cases}$$

Boolean expressions

$$B : \text{BExp} \times \text{Env} \times \text{Store} \longrightarrow \{\text{True, False}\}$$

$$B\|\text{Or}(b_1, b_2)\|_{r,s} = \begin{cases} \text{true}, & \text{if } B\|b_1\|_{r,s}\text{is true} \\ B\|b_2\|_{r,s} & \text{otherwise} \end{cases}$$

Arithmetic and boolean expressions: evaluation

Arithmetic expressions

$$E : \mathrm{AExp} \times \mathrm{Env} \times \mathrm{Store} \longrightarrow \mathrm{Val}$$

$$E\|\mathrm{Sum}(n_1, n_2)\|_{r,s} = E\|n_1\|_{r,s} + E\|n_2\|_{r,s}$$

$$E\|i\|_{r,s} = \begin{cases} s(r(i)), & \text{if } r(i) \in \mathrm{Loc} \\ r(i), & \text{if } r(i) \in \mathrm{Val} \end{cases}$$

Boolean expressions

$$B : \mathrm{BExp} \times \mathrm{Env} \times \mathrm{Store} \longrightarrow \{\mathrm{True},\ \mathrm{False}\}$$

$$B\|\mathrm{Or}(b_1, b_2)\|_{r,s} = \begin{cases} \text{true}, & \text{if } B\|b_1\|_{r,s}\text{is true} \\ B\|b_2\|_{r,s} & \text{otherwise} \end{cases}$$

Declarations: evaluation

$$D : \mathrm{Decl} \times \mathrm{Env} \times \mathrm{Store} \longrightarrow \mathrm{Env} \times \mathrm{Store}$$

Constant

$$D\|\text{const v := n}\|_{r,s} = (r', s)$$

where:

$$r'(y) = \begin{cases} r(y), & \text{if } y \neq v \\ n, & \text{if } y = v \end{cases}$$

Variable

$$D\|\text{var v := n}\|_{r,s} = (r', s')$$

where:

$$r'(y) = \begin{cases} r(y), & \text{if } y \neq v \\ l, & \text{if } y = v \end{cases}$$

$$s'(x) = \begin{cases} s(x), & \text{if } x \neq l \\ n, & \text{if } x = l \end{cases}$$

$l = \mathrm{newmem}(s)$ is a location in the memory s that is not used.

Declarations: evaluation

$$D : \mathrm{Decl} \times \mathrm{Env} \times \mathrm{Store} \longrightarrow \mathrm{Env} \times \mathrm{Store}$$

Constant

$$D\|\mathrm{const}\ \ \mathrm{v}\ := \mathrm{n}\|_{r,s} = (r', s)$$

where:

$$r'(y) = \begin{cases} r(y), & \text{if } y \neq v \\ n, & \text{if } y = v \end{cases}$$

Variable

$$D\|\mathrm{var}\ \ \mathrm{v}\ := \mathrm{n}\|_{r,s} = (r', s')$$

where:

$$r'(y) = \begin{cases} r(y), & \text{if } y \neq v \\ l, & \text{if } y = v \end{cases}$$

$$s'(x) = \begin{cases} s(x), & \text{if } x \neq l \\ n, & \text{if } x = l \end{cases}$$

$l = \mathrm{newmem}(s)$ is a location in the memory s that is not used.

Commands execution

$$C : \mathrm{Com} \times \mathrm{Env} \times \mathrm{Store} \longrightarrow \mathrm{Store}$$

Assignment

$$C\|X := e\|_{r,s} = s'$$

where:

$$l = \Lambda\|X\|_{r,s}$$

$$v = E\|e\|_{r,s}$$

$$s' = \mathrm{updatemem}(s, l, v)$$

Commands execution

$C : \text{Com} \times \text{Env} \times \text{Store} \longrightarrow \text{Store}$

if-then-else

$$C\|\text{if } b \text{ then } c_1 \text{ else } c_2\|_{r,s} = s'$$

where:

$$s'(x) = \begin{cases} C\|c_1\|_{r,s}, & \text{if } B\|b\|_{r,s} = \text{True} \\ C\|c_2\|_{r,s}, & \text{otherwise} \end{cases}$$

Commands execution

$C : \text{Com} \times \text{Env} \times \text{Store} \longrightarrow \text{Store}$

Loop

$$C\|\text{while } b \text{ do } c\|_{r,s} = \begin{cases} s, & \text{if } B\|b\|_{r,s} = \text{False} \\ C\|\text{while } b \text{ do } c\|_{r,s''}, & \text{otherwise} \end{cases}$$

where $s'' = C\|c\|_{r,s}$

Outline

1 Definition

2 Elements of interpreter

3 Semantic analysis

4 Example

5 Exercise

Example: `repeat-until`

Repeat

$$C\|\text{repeat } c \text{ until } b\|_{r,s} = s'$$

where:

$$s' = \begin{cases} s'', & \text{if } E\|b\|_{r,s''} = \text{True} \\ C\|\text{repeat } c \text{ until } b\|_{r,s''}, & \text{otherwise} \end{cases}$$

$$s'' = C\|c\|_{r,s}$$

Example: repeat-until

Repeat

- parser.mly: token REPEAT and UNTIL
- lexer.mll: strings repeat and until
- syntaxtree.ml: constructor Repeat of cmd * bexp for type cmd
- parser.mly: production REPEAT cmd UNTIL bexp ... for non-terminal symbol cmd
- main.ml: nothing :)
- interpreter_base.ml: execution of the command repeat - until

Outline

1 Definition

2 Elements of interpreter

3 Semantic analysis

4 Example

5 Exercise

Programming in crème CAraMeL

Fibonacci number

$$\mathrm{fib}(n) = \begin{cases} n, & \text{if } n < 2 \\ \mathrm{fib}(n-1) + \mathrm{fib}(n-2) & \text{otherwise} \end{cases}$$

Factorial

$$n! = \begin{cases} 1, & \text{if } n = 0 \\ n \times (n-1) & \text{otherwise} \end{cases}$$