Parsing Programming Languages
Errata Corrige

Lorenzo Ceragioli

November 19, 2024

IMT Lucca

Lexing Integers (with ocamllex)

```
1  (* code to be copied in the scanner module *)
2  {
3  open Myparser  (* <-- where we define the tokens *)
4  exception LexingError of string
5  }
6
7  (* some named RExp *)
8  let integer = '-'?['0'-'9']['0'-'9']*
9  let white = [' ' '\t']+ | '\r' | '\n' | "\r\n"
10
11 (* lexing rules *)
12 rule read = parse
13 | white {read lexbuf}
14 | integer {INT(int_of_string (Lexing.lexeme lexbuf))}
15 | "+" {PLUS}
16 | "-" {MINUS}
17 | "*" {TIMES}
18 | eof {EOF}
19 | _ { raise (LexingError (Lexing.lexeme lexbuf)) }
```

The Problem

From the string $6 - 5$

We want: $(\texttt{INT}, 6), \texttt{MINUS}, (\texttt{INT}, 5)$

But we get: $(\texttt{INT}, 6), (\texttt{INT}, -5)$

- After that, the parsing **cannot** be successful!
- We must solve this problem of the lexing phase
- Lexing is unique, while minus has two meanings
- **Solution:**
  - lexing just recognizes naturals and minus
  - parsing solves ambiguities!

Idea of the Solution

From the string $6 - 5$

We get: $(\texttt{INT}, 6), \texttt{MINUS}, (\texttt{INT}, 5)$

From the string $6 + -5$

We get: $(\texttt{INT}, 6), \texttt{PLUS}, \texttt{MINUS}, (\texttt{INT}, 5)$

- In the first case, minus is a binary operator
- In the second case, minus is the sign of the second integer

New Lexer (with ocamllex)

```
1  (* code to be copied in the scanner module *)
2  {
3  open Myparser  (* <-- where we define the tokens *)
4  exception LexingError of string
5  }
6
7  (* some named RExp *)
8  let integer = ['0'-'9']['0'-'9']*
9  let white = [' ' '\t']+ | '\r' | '\n' | "\r\n"
10
11 (* lexing rules *)
12 rule read = parse
13 | white {read lexbuf}
14 | integer {INT(int_of_string (Lexing.lexeme lexbuf))}
15 | "+" {PLUS}
16 | "-" {MINUS}
17 | "*" {TIMES}
18 | eof {EOF}
19 | _ { raise (LexingError (Lexing.lexeme lexbuf)) }
```

New Grammar

$$Exp ::= Int \mid Exp + Exp \mid Exp - Exp \mid Exp \times Exp$$

$$Int ::= n \mid -Int$$

- It is ambiguous because of associativity of operators
- But there is no problem with the new minus
  - If the minus comes after a number then it is the binary operator
  - Otherwise it is the sign of an integer

New Parser (with menhir)

```
1  %{
2    open Aexp
3  %}
4  %token <int> INT
5  %token PLUS MINUS TIMES EOF
6  %start <aexp> prg
7  %left PLUS MINUS  /* lowest precedence */
8  %left TIMES       /* highest precedence */
9
10 %%
11
12 prg:
13     | t = trm; EOF                  {t}
14 trm:
15     | i = int                       {Intliteral i}
16     | t1 = trm; PLUS; t2 = trm      {Plus (t1, t2)}
17     | t1 = trm; MINUS; t2 = trm     {Minus (t1, t2)}
18     | t1 = trm; TIMES; t2 = trm     {Times (t1, t2)}
19 int:
20     | i = INT                       {i}
21     | MINUS; i = int                {-i}
```

Project Fragment

Same as before, but

- for miniImp and MiniFun this lexing problem is not your fault
- either use this updated version or the previous faulty one
- in any case, **this one (and only this one)** will not count as an error for your project!