OcamlLex - Lexical Analyser

October 24, 2018

1 Based numbers

1.1 Intro

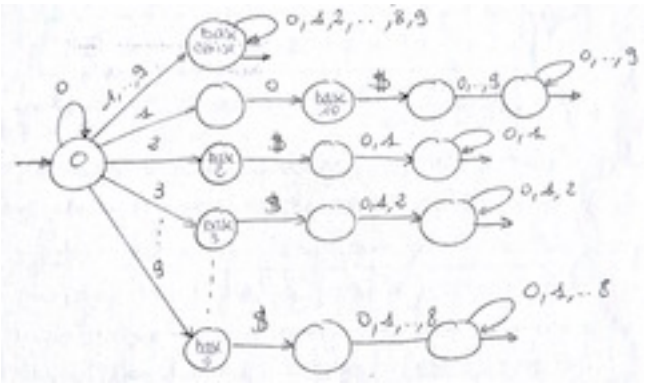
We want to analyse based numbers : recognized a given word and compute its value in the 10 base.
We choose the following representation: *base$value*. $C = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ et $X = C \cup \{\$\}$.

Examples : 2$1011 4$01232 006$1515 10$2025
Base varies from 2 to 10, in the case of base 10, it can be omitted, for instance: 10$27035 can be written 27035

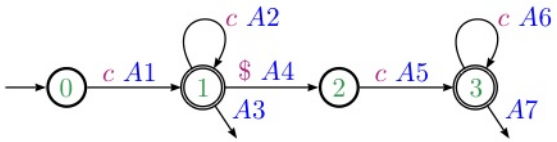
What we have done in class:

- The corresponding regular expression:

$$L = 0^*2\$(0+1)^+ + 0^*3\$(0+1+2)^+ + \ldots + 0^*10\$(0+1+\ldots+9)^+ + 0^*(0+1+\ldots+9)^+$$



- Let the analyser deal with the "..." and use $L = cc^* + cc^*\$cc^*, c \in C$ (without base + base$number).
- the corresponding DETERMINISTIC automaton:



- The actions (in a pseudo-language)

```
base : Integer;
valeur : Integer;
x : Integer;

A1 : base := val(c);

A2 : base := base * 10 + val(c);

A3 : return base;

A4 : if base<2 or base>10 then ERREUR_BASE;
```

```
A5 : x := val(c);
        if x>=base then ERREUR_CHIFFRE;
        else valeur := x;

A6 : x ;= val(c);
        if x>=base then ERREUR_CHIFFRE;
        else valeur := valeur * base + x;

A7 : return valeur;
```

Now we want to program it!
We will use Ocamllex.
In the archive, you will find:

- a Makefile
- `l`exer.mll the lexer divided in two parts:
  - Ocaml specific part with tool functions
  - The rules of the format:

```
l1  = parse
        | integer as i   { (*A2*) print_string ("A2 reads "^(Char.escaped i)^"\n");
          l1 lexbuf }
        | '\n'          { (*A3*) print_string ("A3 reads eol"^"\n")}

rule name (state) = parse
        | regular expression (transition word)     { ocaml program to applied - the
          ACTION ; the call the next state }
```

- prog.ml which calls the lexer.

```
(** lexer.mll *)
{
(*** OCAML PART for tool functions ***)
(* convert char in int *)
let digit_of_char c = (int_of_char c - int_of_char '0')
}

(*** ANALYSER PART ***)
(** Lexical Unit(s) **)
let integer = ['0' - '9']

(** Rules **)
rule l0 = parse
        | integer as i     { (*A1*) print_string ("A1 reads "^(Char.escaped i)^"\n");l1
           lexbuf}
        | _    {failwith "erreur_l0" } (** to exit properly, or add another rule *)
and
l1  = parse
        | integer as i   { (*A2*) print_string ("A2 reads "^(Char.escaped i)^"\n"); l1
           lexbuf }
        | '\n'           { (*A3*) print_string ("A3 reads eol"^"\n")}
        | eof            { (*A3*) print_string ("A3 reads eof"^"\n")}
        |'$'             { (*A4*) print_string ("A4 reads $"^"\n"); l2 lexbuf}
        | _    {failwith "erreur_l1" }
and
l2 = parse
        | integer as i   { (*A5*) print_string ("A5 reads "^(Char.escaped i)^"\n"); l3
           lexbuf}
        | _    {failwith "erreur_l2" }
and
l3 = parse
        | integer as i   { (*A6*) print_string ("A6 reads "^(Char.escaped i)^"\n"); l3
           lexbuf}
        | '\n'           { (*A7*) print_string ("A7 reads eol"^"\n")}
        | eof            { (*A7*) print_string ("A7 reads eof"^"\n")}
        | _    {failwith "erreur_l3" }
```

```
(** prog.ml **)
(* first starts to read the standard input, lexbuf contains the current symbol *)
let lexbuf = Lexing.from_channel stdin in
(* Lexer.l0 calls the l0 rule in the lexer on lexbuf *)
let result = Lexer.l0 lexbuf (* finally prints the result *)
in result;
```

1.2 Exercice

- "Play" with the analyser and call the program ./prog
- Complete the Action part for each rule and each transition. The call to the next rule/state can be done with an argument (the value(s) in construction for instance).

1.3 Resources

- the Makefile is a script helping to compile. It will call for you the commands (ocamllex/ocamlc/ocamlyacc ...)
- Ocaml : https://caml.inria.fr/pub/docs/manual-ocaml/index.html

2 Condensed representation

2.1 Intro

We will study the condensed representation of a suite of positive numbers. In principle condensing is expressing each element of the suite by its gap to a reference value, named "base". For instance, the following suite:

S: 149 147 147 151 151 151 150 150 162 162 143 143

can be consensed refering to 150 as a base:

150-1 150-3 150-3 150+1 150 +1 150+1 150 150 150+12 150+12 150-7 150-7

The condensed representation states first the base then the gaps. Base is separated from gaps by $. And when there are n consecutive occurrences of a same element in the suite, the representation is:

- for element different of the base : n times + (resp. -) followed by a white gap.

  Examples : base is 150 and "b" symbolises white gap, then "151 151 151" becomes "+++1b" ; "147 147" becomes "−−3b"

- for element equals to the base: n times 0 followed by a white gap. Example : base is 150 and "b" symbolises white gap, then "150 150" becomes "00b"

The condensed notation CS of S is thus: **CS: 150$-1b−−3b+++1b00b++12b−−7b**
Program an analyser accepting the notation of a condensed suite and rebuilding the initianl suite (from CS to S).

2.2 Steps

- Find the alphabet and the regular expression
- Find a deterministic automaton
- Find the semantic actions
- Program the semantic actions.