

Ocaml

Autores : Felipe Assad , Jorge Chagas, Thiago

Data : 17/05/2019

Universidade Federal Fluminense

- Ocaml é uma linguagem funcional da família ML desenvolvida pelo INRIA (Instituto Nacional de Pesquisa em Informática e Automação da França) em 1996
- Usado pela Jane Street
- Usado pelo COQ (Provedor de Teoremas)



Global Market Making

We trade an average of **\$13 billion** in global equities every day. Learn how we make the markets more efficient.

- Documentação da linguagem, Real World OCAM, Tutoriais Oficiais e Inria
- Dificuldades para definir a estrutura do projeto (de classes para registros)
- O desafio da recursão mutua

```
type arithmeticExpression =  
  | Num of int  
  | Sum of expression * expression  
  | Sub of expression * expression  
  | Mul of expression * expression  
  | Div of expression * expression  
  
and booleanExpression =  
  | Boo of bool  
  | Eq of expression * expression  
  | Lt of expression * expression  
  | Le of expression * expression  
  | Gt of expression * expression  
  | Ge of expression * expression  
  | And of expression * expression  
  | Or of expression * expression  
  | Not of expression  
  
and expression =  
  | AExp of arithmeticExpression  
  | BExp of booleanExpression  
  | Id of string
```

Estrutura do Projeto

 automaton.ml

 lexer.mll

 main.ml

 makefile

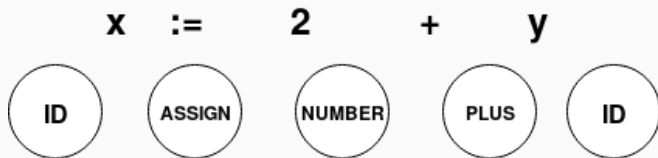
 parser.mly

 pi.ml

 util.ml

O Lexer - Ocamllex

- É um comando que produz um analisador léxico de um conjunto de expressões regulares com ações associadas, gerando tokens.



```
rule token = parse
```

```
  [' ' '\t' '\r' '\n' ] { token lexbuf }      (* skip blanks *)  
| ['0'-'9']+ as lxm    { NUMBER( int_of_string lxm) }  
| '+'                  { PLUS }  
| "[:="                { ASSIGN }  
| ([ 'a'-'z' 'A'-'Z' '_' ] [ 'a'-'z' 'A'-'Z' '0'-'9' '_' ]*) as lxm { ID(lxm) }
```

O Parser - Ocamlyacc

- Ocamlyacc é um Parser Generator de propósito geral que converte uma descrição de uma gramática para uma LALR(1) (Look Ahead Left to Right) em um programa Ocaml para realizar o parser da gramática.
- ASSIGN (ID (x), SUM (NUM (2), ID (y)))

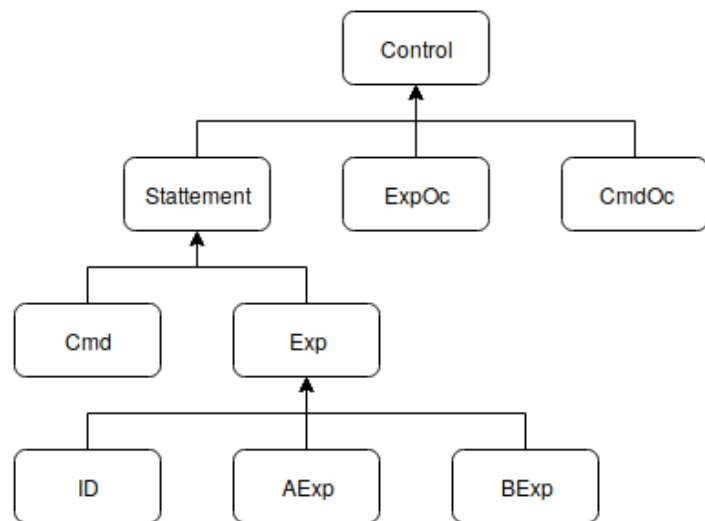


0 Parser - Ocamlyacc

```
statement:
    expression { Pi.Exp($1)}
    | command   {Pi.Cmd($1)}
;
```

```
expression:
    arithmeticExpression      { Pi.AExp( $1) }
    | booleanExpression       { Pi.BExp( $1) }
    | ID                      { Pi.Id( $1) }
    | LPAREN expression RPAREN { $2 }
```

[illegible]



$\delta(\text{Sum}(E_1, E_2) :: C, V, S) = \delta(E_1 :: E_2 :: \#SUM :: C, V, S)$

$\delta(\#SUM :: C, \text{Num}(N_1) :: \text{Num}(N_2) :: V, S) = \delta(C, N_1 + N_2 :: V, S)$

```
| Sum(AExp(x), AExp(y)) -> (  
  (Stack.push (Exp0c(OPSUM)) controlStack);  
  (Stack.push (Statement(Exp(AExp(y)))) controlStack);  
  (Stack.push (Statement(Exp(AExp(x)))) controlStack);  
| Sum(Id(x), AExp(y)) -> (  
  (Stack.push (Exp0c(OPSUM)) controlStack);  
  (Stack.push (Statement(Exp(AExp(y)))) controlStack);  
  (Stack.push (Statement(Exp(Id(x)))) controlStack);  
| Sum(AExp(x), Id(y)) -> (  
  (Stack.push (Exp0c(OPSUM)) controlStack);  
  (Stack.push (Statement(Exp(Id(y)))) controlStack);  
  (Stack.push (Statement(Exp(AExp(x)))) controlStack);  
| Sum(Id(x), Id(y)) -> (  
  (Stack.push (Exp0c(OPSUM)) controlStack);  
  (Stack.push (Statement(Exp(Id(y)))) controlStack);  
  (Stack.push (Statement(Exp(Id(x)))) controlStack);
```

```
| Exp0c(exp0c) -> (  
  match exp0c with  
  | OPSUM -> (  
    let x = (Stack.pop valueStack) in  
    match x with  
    | Int(i) -> (  
      let y = (Stack.pop valueStack) in  
      match y with  
      | Int(j) -> (  
        (Stack.push (Int(i + j)) valueStack);  
  
        );  
      | _ -> raise (AutomatonException "error on #SUM");  
    );  
    | _ -> raise (AutomatonException "erro on #SUM");  
  );  
);
```

$$\delta(\text{Not}(E) :: C, V, S) = \delta(E :: \#NOT :: C, V, S)$$
$$\delta(\#NOT :: C, \text{Boo}(\text{True}) :: V, S) = \delta(C, \text{False} :: V, S)$$
$$\delta(\#NOT :: C, \text{Boo}(\text{False}) :: V, S) = \delta(C, \text{True} :: V, S)$$

```
| Not(BExp(x)) -> (  
  (Stack.push (Exp0c(OPNOT)) controlStack);  
  (Stack.push (Statement(Exp(BExp(x)))) controlStack);  
  
);  
| Not(Id(x)) -> (  
  (Stack.push (Exp0c(OPNOT)) controlStack);  
  (Stack.push (Statement(Exp(Id(x)))) controlStack);
```

```
| OPNOT -> (  
  let x = (Stack.pop valueStack) in  
  match x with  
  | Bool(i) -> (  
    (Stack.push (Bool(not(i))) valueStack);  
    );  
  
  | _ -> raise (AutomatonException "erro on #NOT");  
  );
```

$$\delta(\text{Lt}(E_1, E_2) :: C, V, S) = \delta(E_1 :: E_2 :: \#LT :: C, V, S)$$

$$\delta(\#LT :: C, \text{Num}(N_1) :: \text{Num}(N_2) :: V, S) = \delta(C, N_1 < N_2 :: V, S)$$

```
| Lt(AExp(x), AExp(y)) -> (
  (Stack.push (ExpOc(OPLT)) controlStack);
  (Stack.push (Statement(Exp(AExp(y)))) controlStack);
  (Stack.push (Statement(Exp(AExp(x)))) controlStack);
| Lt(AExp(x), Id(y)) -> (
  (Stack.push (ExpOc(OPLT)) controlStack);
  (Stack.push (Statement(Exp(Id(y)))) controlStack);
  (Stack.push (Statement(Exp(AExp(x)))) controlStack);
| Lt(Id(x), AExp(y)) -> (
  (Stack.push (ExpOc(OPLT)) controlStack);
  (Stack.push (Statement(Exp(AExp(y)))) controlStack);
  (Stack.push (Statement(Exp(Id(x)))) controlStack);
| Lt(Id(x), Id(y)) -> (
  (Stack.push (ExpOc(OPLT)) controlStack);
  (Stack.push (Statement(Exp(Id(y)))) controlStack);
  (Stack.push (Statement(Exp(Id(x)))) controlStack);
```

```
| OPLT -> (  
  let x = (Stack.pop valueStack) in  
  match x with  
  | Bool(i) -> (  
    let y = (Stack.pop valueStack) in  
    match y with  
    | Bool(j) -> (  
      (Stack.push (Bool(j < i)) valueStack);  
  
    );  
    | _ -> raise (AutomatonException "erro on #OPLT");  
  );  
| Int(i) -> (  
  let y = (Stack.pop valueStack) in  
  match y with  
  | Int(j) -> (  
    (Stack.push ( Bool ( j < i)) valueStack);  
  
    );  
  | _ -> raise (AutomatonException "erro on #OPLT");  
);  
| -> raise (AutomatonException "erro on #OPLT");
```

$\delta(\text{Id}(W) :: C, V, E, S) = \delta(C, B :: V, E, S)$, where $E[W] = I \wedge S[I] = B$,

```
| Id(id) -> (  
  let key = Hashtbl.find environment id in  
  match key with  
  | Value(x) -> ();  
  | Loc(x) -> (  
    let value = Hashtbl.find memory x in  
    match value with  
    | Integer(x) -> (Stack.push (Int(x)) valueStack);  
    | Boolean(x) -> (Stack.push (Bool(x)) valueStack);  
  )  
);
```

$$\delta(\text{Assign}(W, X) :: C, V, E, S) = \delta(X :: \#ASSIGN :: C, W :: V, E, S'),$$
$$\delta(\#ASSIGN :: C, T :: W :: V, E, S) = \delta(C, V, E, S'), \text{ where } E[W] = I \wedge S' = S/[I \mapsto T],$$

```
| Assign(Id(x), y) -> (  
  (Stack.push (Cmd0c(OPASSIGN)) controlStack );  
  (Stack.push (Statement(Exp(y))) controlStack );  
  (Stack.push (Str(x)) valueStack);
```



```
| OPASSIGN -> (  
  let value = (Stack.pop valueStack) in  
  let id = (Stack.pop valueStack) in  
  match id with  
  | Str(x) -> (  
    let env = (Hashtbl.find environment x ) in  
    match env with  
    | Loc(l) -> (  
      match value with  
      | Int(i) -> (  
        (Hashtbl.replace memory l (Integer(i)));  
      );  
      | Bool(b) -> (  
        (Hashtbl.replace memory l (Boolean(b)));  
      );  
      | _ -> raise (AutomatonException "erro on #assign")  
    );  
    | Value(v) -> (  
      );  
  );  
  | _ -> raise (AutomatonException "error on #assign")
```

$$\begin{aligned}\delta(\text{Cond}(X, M_1, M_2) :: C, V, E, S) &= \delta(X :: \#COND :: C, \text{Cond}(X, M_1, M_2) :: V, E, S), \\ \delta(\#COND :: C, \text{Boo}(\text{true}) :: \text{Cond}(X, M_1, M_2) :: V, E, S) &= \delta(M_1 :: C, V, E, S), \\ \delta(\#COND :: C, \text{Boo}(\text{false}) :: \text{Cond}(X, M_1, M_2) :: V, E, S) &= \delta(M_2 :: C, V, E, S),\end{aligned}$$

```
| Cond(BExp(x), y, z) -> (  
  (Stack.push (CmdOc(OPCOND)) controlStack);  
  (Stack.push (Statement(Exp(BExp(x)))) controlStack );  
  (Stack.push (Cmd_to_vstack(Statement(Cmd(Cond(BExp(x), y, z))))) valueStack );  
);  
| Cond(Id(x), y, z) -> (  
  (Stack.push (CmdOc(OPCOND)) controlStack);  
  (Stack.push (Statement(Exp(Id(x)))) controlStack );  
  (Stack.push (Cmd_to_vstack(Statement(Cmd(Cond(Id(x), y, z))))) valueStack );
```

```
| OPCOND -> (  
  let ifcond = (Stack.pop valueStack) in  
  let condV = (Stack.pop valueStack) in  
  match ifcond with  
  | Bool(true) -> (  
    match condV with  
    | Cmd_to_vstack(cond) ->(  
      match cond with  
      | (Statement(Cmd(Cond(x,m1,m2)))) -> (  
        (Stack.push (Statement(Cmd(m1))) controlStack);  
      )  
      | _ -> raise (AutomatonException "erro on #COND 1");  
    )  
    | _ -> raise (AutomatonException "erro on #COND 2");  
  );  
);
```

```
| Bool(false) -> (  
  match condV with  
  | Cmd_to_vstack(cond) ->(  
    match cond with  
    | (Statement(Cmd(Cond(x,m1,m2)))) -> (  
      (Stack.push (Statement(Cmd(m2))) controlStack);  
    );  
    | _ -> raise (AutomatonException "erro on #COND 3");  
  )  
  | _ -> raise (AutomatonException "erro on #COND 4");  
);  
| _ -> raise (AutomatonException "erro on #COND 5" );
```

Exemplos

- Vejamos agora alguns exemplos...
- Fatorial
- Fibonacci