Universidade da Beira Interior
Departamento de Informática

Engenharia Informática
Processamento de Linguagens

Elaborado por:
Miguel Mendonça,nr 35388
João Domingos, nr 38023

Professor:
Simão Melo de Sousa

13 de Janeiro de 2019

Capítulo 1

Introdução

1.1 Estrutura do Relatório

O relatório encontra-se dividido em quatro capítulos e diferentes secções:

1. Introdução (1)
   - Estrutura do relatório (1.1)
   - Motivação (1.2)
   - Objetivos (1.3).
2. Estrutura do programa (2)
   - Sintaxe Abstrata (2.1)
   - Semântica Operacional (2.2)
   - Tipagem (2.3)
   - Implementação (2.4)
3. Manual de Utilizador (3)
4. Conclusão (4)

1.2 Motivação

O que motivou o grupo a desenvolver este trabalho foi o facto de poder interpretar e entender como funciona uma nova linguagem, bem como criar um compilador para ela.

1.3 Objetivos

O objetivo geral deste trabalho foi a construção de um compilador para a linguagem Arith, tendo por foco a arquitetura MIPS.

Capítulo 2

Estrutura do programa

2.1 Sintaxe Abstrata

typ ::=	**tipo**
| *i*	inteiro
| *var*	*string*
expr ::=	**expressão**
| *typ*	tipo
| *e* + *e*	adição
| *e* - *e*	subtração
| *e* * *e*	multiplicação
| *e* / *e*	divisão
stmt ::=	**instrução**
| *set x = expr*	atribuição
| *print expr*	impressão
stmts ::=	**instruções**
| *stmt*	instrução
| *stmts , stmt*	instruções

2.2 Semântica Operacional

Operações

Adição

$$\frac{E, e1 \Rightarrow v1 \quad e2 \Rightarrow v2}{e1 + e2 \rightarrow v1 + v2} \tag{2.1}$$

Subtração

$$\frac{E, e1 \Rightarrow v1 \quad e2 \Rightarrow v2}{e1 - e2 \rightarrow v1 - v2} \tag{2.2}$$

Multiplicação

$$\frac{E, e1 \Rightarrow v1 \quad e2 \Rightarrow v2}{e1 * e2 \rightarrow v1 * v2} \tag{2.3}$$

Divisão

$$\frac{E, e1 \Rightarrow v1 \quad e2 \Rightarrow v2}{e1/e2 \rightarrow v1/v2} \tag{2.4}$$

Atribuição

$$\frac{E, e \Rightarrow v}{E, x \leftarrow e \rightarrow E\{x \mapsto v\}} \tag{2.5}$$

2.3 Tipagem

Operações

$$\tau \vdash + : int * int \rightarrow int \tag{2.6}$$

$$\tau \vdash - : int * int \rightarrow int \tag{2.7}$$

$$\tau \vdash * : int * int \rightarrow int \tag{2.8}$$

$$\tau \vdash / : int * int \rightarrow int \tag{2.9}$$

Impressão

print();;

$$\tau \vdash int \quad ! \quad \textit{UNIT} \tag{2.10}$$

2.4 Implementação

No código seguinte (2.1) é apresentado o ficheiro *ast.mli* que representa a árvore de sintaxe abstrata implementada no sistema. Esta vai tratar dos tipos utilizados no programa bem como as operações que são realizáveis.

```
type typ =
    Int of int
  | Var of string

type binop =
  Add
  | Sub
  | Mul
  | Div

type expr =
  Typ of typ
 | Binop of binop * expr * expr

 type stmt =
  Set of string * expr
  | Print of expr

and pro = stmt list
```

Listing 2.1: ast.mli

Abaixo (2.2) está representado o ficheiro *parser.mly* e aqui encontram-se os *tokens* utilizados e as suas associações.

```
%{
  open Ast
%}

%token <int> INT
%token <string> ID

%token EOF
%token COLON
%token PLUS MINUS
%token TIMES DIV
%token EQ

%token SET
%token PRINT

%left PLUS MINUS
%left TIMES DIV
```

Listing 2.2: tokens - parser.mly

Neste código (2.3) é demonstrado a gramática e a forma como o compilador faz os *shifts* e os *reduces*.

```
%start pro

%type <Ast.pro> pro
%%
pro:
     s = stmts EOF { List.rev s }
     ;

stmts:
     s= stmt {[s]}
     | s1= stmts COLON s2=stmt {s2::s1}
     ;

typ:
     i = INT {Int i}
     | id = ID {Var id}
     ;

stmt:
     SET id = ID EQ e = expr { Set (id, e)}
    | PRINT e = expr { Print e}
    ;

expr:
     t= typ {Typ t}
     | e1=expr o=op e2=expr {Binop (o, e1, e2)}
     ;

%inline op:
     PLUS {Add}
     |MINUS {Sub}
     |TIMES {Mul}
     |DIV {Div}
     ;
```

Listing 2.3: gramática - parser.mly

Na listagem seguinte (2.4) encontra-se o ficheiro *lexer.mll* que converte para *tokens* os caracteres inseridos para compilação.

```
let integer = ['0'-'9']+
let digit = ['0' - '9']
let space = [' ' '\t']
let letter = ['a' - 'z' 'A'-'Z']
let ident = letter (letter | digit)*

rule token = parse
  | '\n' { newline lexbuf; token lexbuf }
  | ident as id { id_or_kwd id}
  | space+ { token lexbuf }
  | integer as i {INT (int_of_string i)}
  | "=" {EQ}
  | "+" {PLUS}
  | "-" {MINUS}
  | "*" {TIMES}
  | "/" {DIV}
  | "," {COLON}
  | eof {EOF}
  | _ as c {raise (let x = (Printf.sprintf "%c" c) in (ErrorLexing ("
    Unkown character " ^ x)))}
```

Listing 2.4: lexer.mll

Abaixo são apresentadas partes do ficheiro *compile.ml*, estas designam a interpretação para a arquitetura *MIPS* através do código inserido.
Neste trecho (2.5) encontra-se a forma como o programa guarda as variáveis a partir de uma *Hashtable*.

```
let (vars : (string, unit) Hashtbl.t) = Hashtbl.create 32
```

Listing 2.5: variáveis - compile.ml

Nesta parte do código (2.6) é usada uma função de forma recursiva para compilar expressões.

```
let rec compile_expr = function
  Typ t ->
  begin
      match t with
      Int i ->
        comment ("storing int")++
        li t0 i ++
        push t0
      | Var v ->
      begin
        if Hashtbl.mem vars v then
          comment ("storing var")++
          lw t0 alab v ++
          push t0
        else raise (ErrorCompiling ("Undefined variable '"^v^"'."))
      end
  end
  |Binop (Add, e1, e2) ->
    comment ("adding")++
    compile_expr e1 ++
    compile_expr e2 ++
    pop t0 ++
    pop t1 ++
    add t0 t0 oreg t1 ++
    push t0
  |Binop (Sub, e1, e2) ->
    comment ("subtracting")++
    compile_expr e1 ++
    compile_expr e2 ++
    pop t0 ++
    pop t1 ++
    sub t0 t1 oreg t0 ++
    push t0
  |Binop (Mul, e1, e2) ->
    comment ("multiplying")++
    compile_expr e1 ++
    compile_expr e2 ++
    pop t0 ++
    pop t1 ++
    mul t0 t0 oreg t1 ++
    push t0
  |Binop (Div, e1, e2) ->
```

```
    comment ("dividing")++
    compile_expr e1 ++
    compile_expr e2 ++
    pop t0 ++
    pop t1 ++
    div t0 t1 oreg t0 ++
    push t0
```

Listing 2.6: compilação de expressões - compile.ml

Neste trecho do programa (2.7) é usada uma função para compilar instruções.

```
let compile_stmt = function
  | Set (v, e) ->
    Hashtbl.replace vars v ();
    comment ("setting") ++
    compile_expr e ++
    pop t0 ++
    sw t0 alab v
  | Print e ->
    comment ("printing")++
    compile_expr e ++
    pop t0 ++
    move a0 t0 ++
    li v0 1 ++
    syscall
```

Listing 2.7: compilação de intruções - compile.ml

Capítulo 3

Manual de Utilizador

Neste capitulo, com base em alguns diagramas, são explicadas as funcionalidades deste compilador.

Na Figura 3.1 é apresentado um esquema geral das instruções que este compilador consegue desempenhar com sucesso.

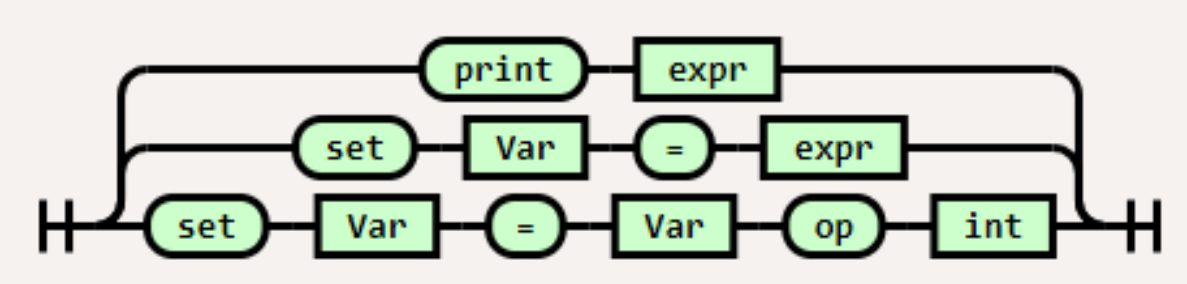

Figura 3.1: Diagrama geral do compilador.

Na Figura 3.2 são mostradas as duas formas de definir uma variavel. Atraves do diagrama, é possivel perceber que para se definir uma variavel é necessario, obrigatoriamente escrever a palavra *"set"* antes da variavel. Sendo que, de seguida é igualada à expressão pretendida. É também possivel redifinir uma variavel.

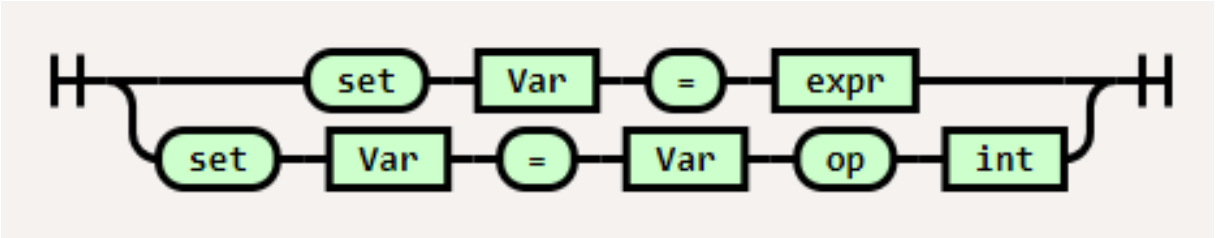

Figura 3.2: Instrução *"set"*.

Na figura 3.3 é mostrada a forma como é possivel imprimir uma expressão : sendo que será necessário digitar a palavra *"print"* antes da expressão a imprimir.

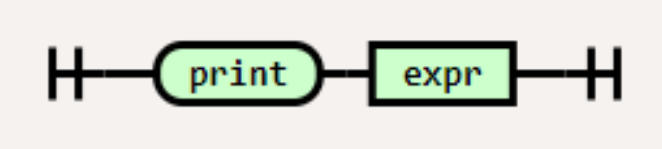

Figura 3.3: Instrução *print*.

Na Figura 3.4 são apresentadas as operações que podem ser utilizadas, bem como a sua sintaxe.

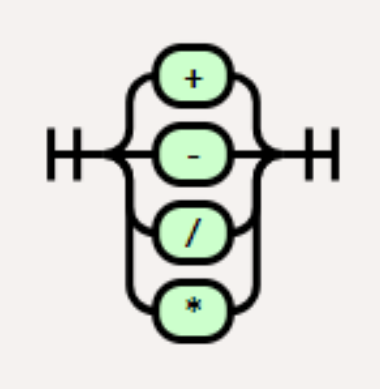

Figura 3.4: Operações possiveis.

Capítulo 4

Conclusão

Neste relatório foi estudada e documentada a sintaxe abstrata, a semântica operacional e o sistema de tipos de um compilador para arquitectura *MIPS*. Estão também presentes neste relatório alguns excertos de código acompanhados da respectiva explicação.

Em suma, pensamos que os objetivos principais do projeto foram alcançados, no entanto não serão descartadas possíveis melhorias a realizar no futuro.