Ocampiler

Autores : Felipe Assad , Jorge Chagas, Thiago

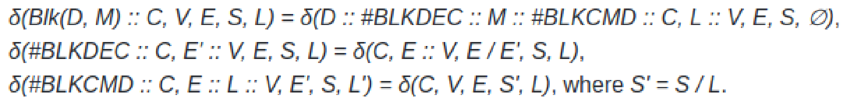
Data : 28/06/2019

Universidade Federal Fluminense

Objetivos desta apresentação

- Especificar a semântica de declarações utilizando o Pi Framework.
- Implementar um parser para a linguagem Imp-1 estendendo Imp-0 com declarações de variáveis e constantes.
- Implementar Pi IR-mark1: (i) Interpreting Automata com ambientes, (ii) declarações de variáveis e constantes.
- Implementar um compilador de Imp-1 para Pi IR-mark1.

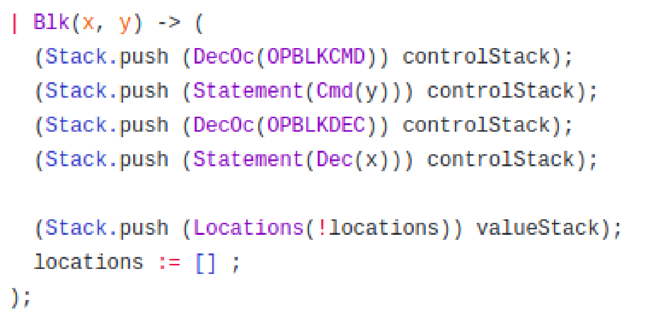
Implementação do Automaton



$\delta(Blk(D, M) :: C, V, E, S, L) = \delta(D :: \#BLKDEC :: M :: \#BLKCMD :: C, L :: V, E, S, \varnothing)$,

$\delta(\#BLKDEC :: C, E' :: V, E, S, L) = \delta(C, E :: V, E / E', S, L)$,

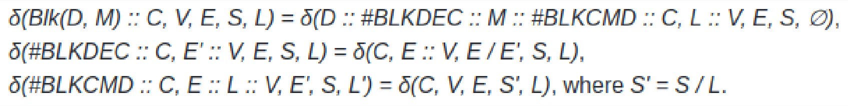
$\delta(\#BLKCMD :: C, E :: L :: V, E', S, L') = \delta(C, V, E, S', L)$, where $S' = S / L$.



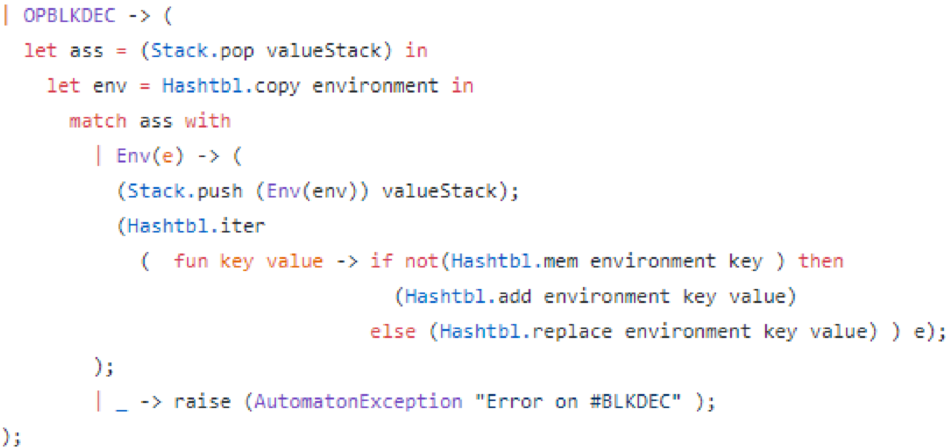
```
| Blk(x, y) -> (
  (Stack.push (DecOc(OPBLKCMD)) controlStack);
  (Stack.push (Statement(Cmd(y))) controlStack);
  (Stack.push (DecOc(OPBLKDEC)) controlStack);
  (Stack.push (Statement(Dec(x))) controlStack);

  (Stack.push (Locations(!locations)) valueStack);
  locations := [] ;
);
```

Implementação do Automaton

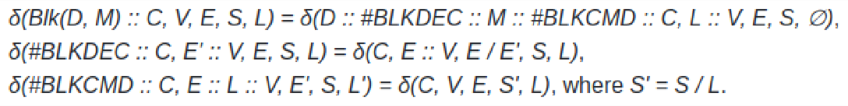


$\delta(Blk(D, M) :: C, V, E, S, L) = \delta(D :: \#BLKDEC :: M :: \#BLKCMD :: C, L :: V, E, S, \emptyset)$,
$\delta(\#BLKDEC :: C, E' :: V, E, S, L) = \delta(C, E :: V, E / E', S, L)$,
$\delta(\#BLKCMD :: C, E :: L :: V, E', S, L') = \delta(C, V, E, S', L)$, where $S' = S / L$.

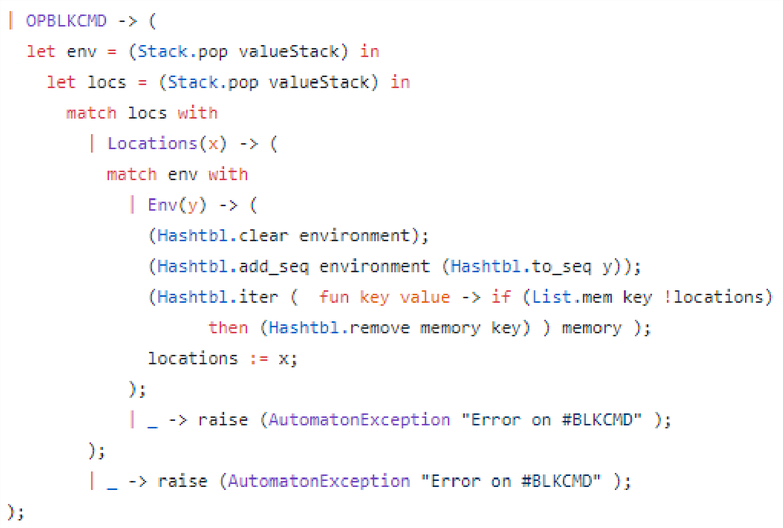


```
| OPBLKDEC -> (
  let ass = (Stack.pop valueStack) in
    let env = Hashtbl.copy environment in
      match ass with
        | Env(e) -> (
          (Stack.push (Env(env)) valueStack);
          (Hashtbl.iter
            (   fun key value -> if not(Hashtbl.mem environment key ) then
                                  (Hashtbl.add environment key value)
                                else (Hashtbl.replace environment key value) ) e);
        );
        | _ -> raise (AutomatonException "Error on #BLKDEC" );
);
```

Implementação do Automaton

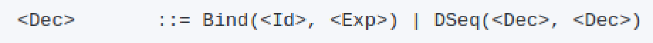


$\delta(Blk(D, M) :: C, V, E, S, L) = \delta(D :: \#BLKDEC :: M :: \#BLKCMD :: C, L :: V, E, S, \emptyset)$,
$\delta(\#BLKDEC :: C, E' :: V, E, S, L) = \delta(C, E :: V, E / E', S, L)$,
$\delta(\#BLKCMD :: C, E :: L :: V, E', S, L') = \delta(C, V, E, S', L)$, where $S' = S / L$.

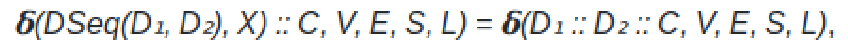


```
| OPBLKCMD -> (
  let env = (Stack.pop valueStack) in
    let locs = (Stack.pop valueStack) in
      match locs with
        | Locations(x) -> (
          match env with
            | Env(y) -> (
              (Hashtbl.clear environment);
              (Hashtbl.add_seq environment (Hashtbl.to_seq y));
              (Hashtbl.iter (  fun key value -> if (List.mem key !locations)
                          then (Hashtbl.remove memory key) ) memory );
              locations := x;
            );
            | _ -> raise (AutomatonException "Error on #BLKCMD" );
        );
        | _ -> raise (AutomatonException "Error on #BLKCMD" );
);
```

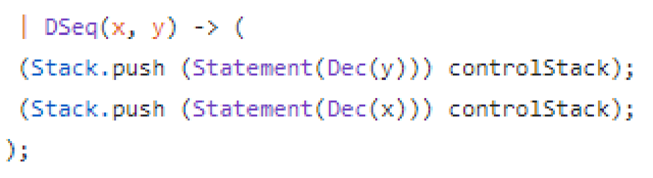
Implementação do Automaton



```
<Dec>       ::= Bind(<Id>, <Exp>) | DSeq(<Dec>, <Dec>)
```

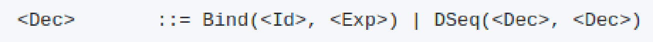


$$\delta(DSeq(D_1, D_2), X) :: C, V, E, S, L) = \delta(D_1 :: D_2 :: C, V, E, S, L),$$

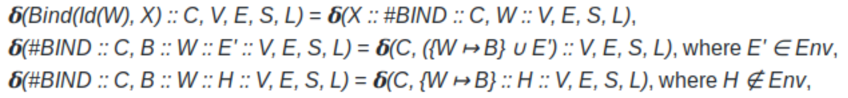


```
  | DSeq(x, y) -> (
  (Stack.push (Statement(Dec(y))) controlStack);
  (Stack.push (Statement(Dec(x))) controlStack);
);
```

Implementação do Automaton



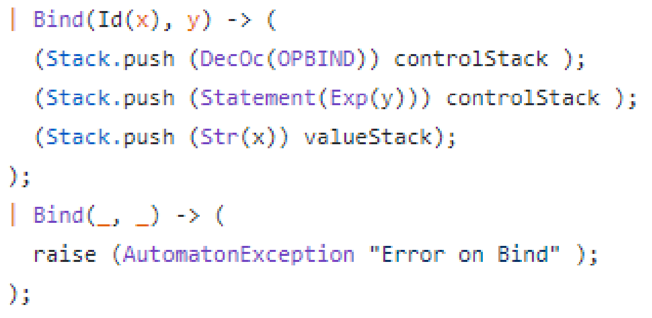
```
<Dec>       ::= Bind(<Id>, <Exp>) | DSeq(<Dec>, <Dec>)
```



$\boldsymbol{\delta}(Bind(Id(W), X) :: C, V, E, S, L) = \boldsymbol{\delta}(X :: \#BIND :: C, W :: V, E, S, L)$,

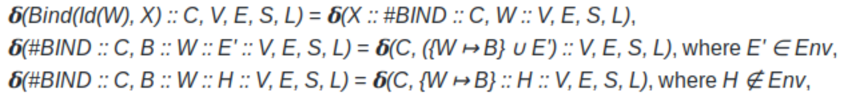
$\boldsymbol{\delta}(\#BIND :: C, B :: W :: E' :: V, E, S, L) = \boldsymbol{\delta}(C, (\{W \mapsto B\} \cup E') :: V, E, S, L)$, where $E' \in Env$,

$\boldsymbol{\delta}(\#BIND :: C, B :: W :: H :: V, E, S, L) = \boldsymbol{\delta}(C, \{W \mapsto B\} :: H :: V, E, S, L)$, where $H \notin Env$,



```
| Bind(Id(x), y) -> (
  (Stack.push (DecOc(OPBIND)) controlStack );
  (Stack.push (Statement(Exp(y))) controlStack );
  (Stack.push (Str(x)) valueStack);
);
| Bind(_, _) -> (
  raise (AutomatonException "Error on Bind" );
);
```

Implementação do Automaton



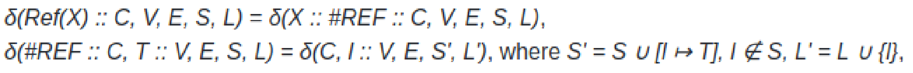
$\boldsymbol{\delta}(Bind(Id(W), X) :: C, V, E, S, L) = \boldsymbol{\delta}(X :: \#BIND :: C, W :: V, E, S, L),$

$\boldsymbol{\delta}(\#BIND :: C, B :: W :: E' :: V, E, S, L) = \boldsymbol{\delta}(C, (\{W \mapsto B\} \cup E') :: V, E, S, L),$ where $E' \in Env,$

$\boldsymbol{\delta}(\#BIND :: C, B :: W :: H :: V, E, S, L) = \boldsymbol{\delta}(C, \{W \mapsto B\} :: H :: V, E, S, L),$ where $H \notin Env,$

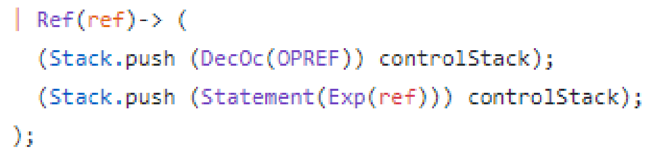
Ver o OPBIND no documento

Implementação do Automaton



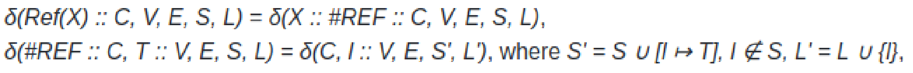
$\delta(Ref(X) :: C, V, E, S, L) = \delta(X :: \#REF :: C, V, E, S, L),$

$\delta(\#REF :: C, T :: V, E, S, L) = \delta(C, l :: V, E, S', L')$, where $S' = S \cup [l \mapsto T], l \notin S, L' = L \cup \{l\},$



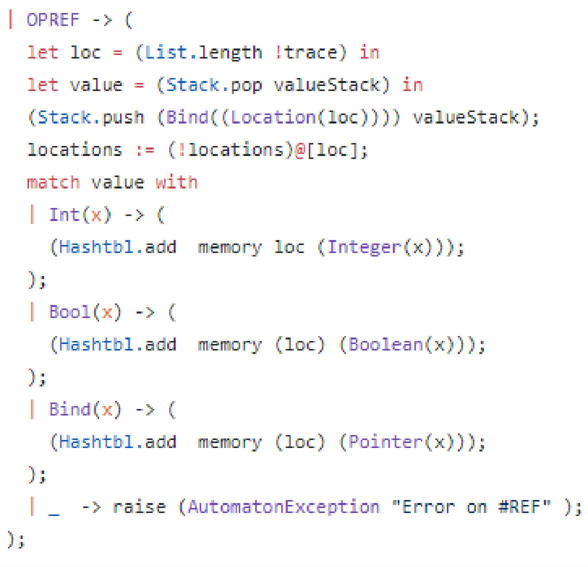
```
| Ref(ref)-> (
  (Stack.push (DecOc(OPREF)) controlStack);
  (Stack.push (Statement(Exp(ref))) controlStack);
);
```

Implementação do Automaton



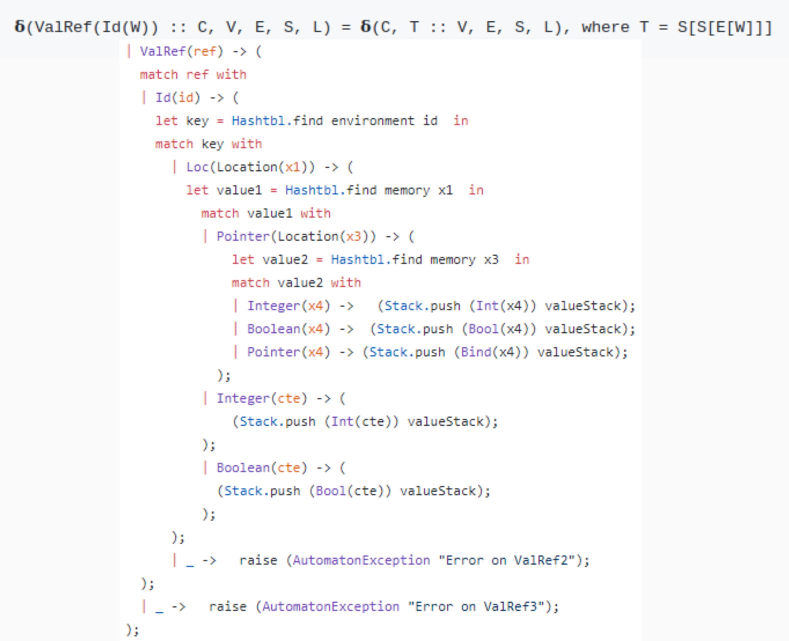
$\delta(Ref(X) :: C, V, E, S, L) = \delta(X :: \#REF :: C, V, E, S, L),$

$\delta(\#REF :: C, T :: V, E, S, L) = \delta(C, l :: V, E, S', L')$, where $S' = S \cup [l \mapsto T], l \notin S, L' = L \cup \{l\},$



```
| OPREF -> (
  let loc = (List.length !trace) in
  let value = (Stack.pop valueStack) in
  (Stack.push (Bind((Location(loc)))) valueStack);
  locations := (!locations)@[loc];
  match value with
  | Int(x) -> (
    (Hashtbl.add  memory loc (Integer(x)));
  );
  | Bool(x) -> (
    (Hashtbl.add  memory (loc) (Boolean(x)));
  );
  | Bind(x) -> (
    (Hashtbl.add  memory (loc) (Pointer(x)));
  );
  | _  -> raise (AutomatonException "Error on #REF" );
);
```

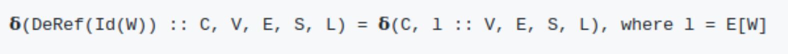
Implementação do Automaton



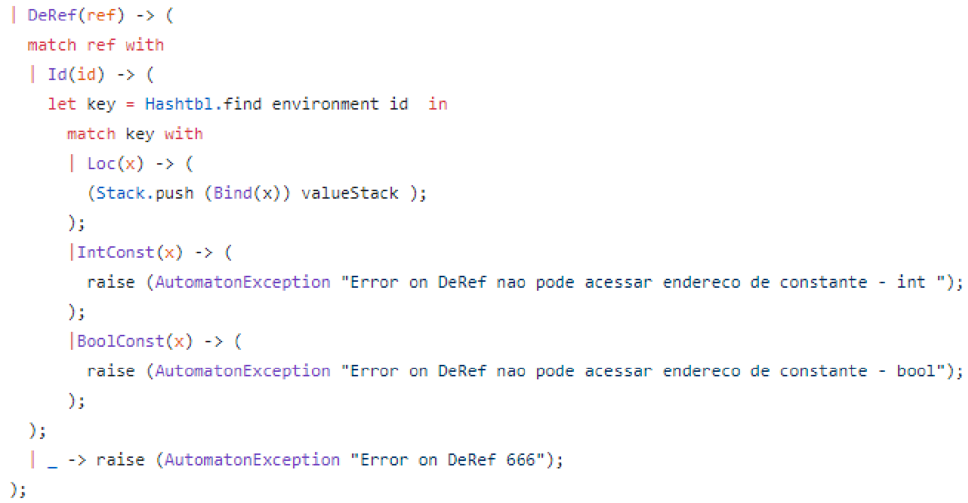
```
δ(ValRef(Id(W)) :: C, V, E, S, L) = δ(C, T :: V, E, S, L), where T = S[S[E[W]]]
```

```
| ValRef(ref) -> (
  match ref with
  | Id(id) -> (
    let key = Hashtbl.find environment id  in
    match key with
      | Loc(Location(x1)) -> (
        let value1 = Hashtbl.find memory x1  in
          match value1 with
          | Pointer(Location(x3)) -> (
              let value2 = Hashtbl.find memory x3  in
              match value2 with
              | Integer(x4) ->   (Stack.push (Int(x4)) valueStack);
              | Boolean(x4) ->  (Stack.push (Bool(x4)) valueStack);
              | Pointer(x4) -> (Stack.push (Bind(x4)) valueStack);
            );
          | Integer(cte) -> (
              (Stack.push (Int(cte)) valueStack);
          );
          | Boolean(cte) -> (
            (Stack.push (Bool(cte)) valueStack);
          );
      );
      | _ ->   raise (AutomatonException "Error on ValRef2");
  );
  | _ ->   raise (AutomatonException "Error on ValRef3");
);
```

Implementação do Automaton



```
δ(DeRef(Id(W)) :: C, V, E, S, L) = δ(C, l :: V, E, S, L), where l = E[W]
```



```
| DeRef(ref) -> (
  match ref with
  | Id(id) -> (
    let key = Hashtbl.find environment id  in
      match key with
      | Loc(x) -> (
        (Stack.push (Bind(x)) valueStack );
      );
      |IntConst(x) -> (
        raise (AutomatonException "Error on DeRef nao pode acessar endereco de constante - int ");
      );
      |BoolConst(x) -> (
        raise (AutomatonException "Error on DeRef nao pode acessar endereco de constante - bool");
      );
  );
  | _ -> raise (AutomatonException "Error on DeRef 666");
);
```

Avaliação da evolução do trabalho

- Implementar um parser para a linguagem Imp-1 estendendo Imp-0 com declarações de variáveis e constantes. (OK)
- Implementar IR-mark1: (i) Interpreting Automata com ambientes , (ii) declarações de variáveis e constantes.(OK)
- Implementar um compilador de Imp-1 para IR-mark1. (OK)