DeceptiCompiler

Mais do que os olhos podem ver.

P2 de Compiladores (2018.1), por Raffael Paranhos, Wallace Baleroni e Júlia Falcão

13 de Junho de 2018

Universidade Federal Fluminense

Parser: Declarações

```
rule(:decl_seq) { decl >> seq_op >> decl_seq | decl }
```

var x = 1, y = 2; const z = 3

```
rule(:decl) { decl_op >> ini_seq }
```

var x = 1, y = 2

```
rule(:ini_seq) { ini >> com_op >> ini_seq | ini }
```

x = 1, y = 2

```
rule(:ini) { ident >> ini_op >> exp }
```

x = 1

E-SMC

Acrescentamos uma nova pilha à estrutura do SMC, a pilha E de ambientes.

```
@E = [{}]
@S = []
@M = {}
@C = []
```

A cada declaração, um novo ambiente é criado, que é uma cópia do anterior contendo também a nova variável ou constante declarada. Esse ambiente é empilhado em E, e passa a ser o ambiente vigente até o fim do escopo.

Ambiente

O ambiente é representado por uma hash que associa um identificador a um *bindable*.

Exemplo:

```
var x = 5, y = 1;
const w = 1
```

O ambiente será:

```
{"x" => (loc, 0), "y" => (loc, 1), "w" => (value, 1)}
```

As variáveis x e y estão nas localizações de memória 0 e 1, enquanto a constante w fica guardada no ambiente e tem valor 1.

Memória

```
{0 => "5", 1 => "1"}
```

A memória também é uma hash, e associa localizações aos valores que estão guardados nelas. No caso do exemplo anterior, temos 5 na localização 0 (da variável x), e 1 na localização 1 (da variável y).

Assign em constante

É proibido fazer a operação de assign e alterar o valor de uma constante. A tentativa causa um erro:

```
proc aumenta_nota(x) {
  const x = 9
  x := 10
}
```

```
(RuntimeError): Tentativa de assign em constante.
```

Palavras reservadas

O E-SMC também possui uma lista de palavras reservadas da linguagem. Uma tentativa de declarar uma variável ou constante com o nome de alguma palavra reservada causa um erro.

```
proc teste(x) {
   var add = 10
}
```

```
(RuntimeError): Palavra reservada usada como
identificador.
```

Parser: Bloco

```
rule(:block) { lcb >> decl_seq >> cmd >> rcb }
```

Um **bloco** é definido por uma sequência de declarações e em seguida uma sequência de comandos.

Exemplo:

```
proc teste(x) {
  const x = 5;
  var y = 2

  if (x >= y) {
    y := y * x
  };
  print(y)
}
```

Fim do bloco

Quando um bloco é empilhado na pilha de controle, antes dele empilhamos uma estrutura chamada `blockend`. O bloco será executado quando for retirado do topo da pilha, e quando sua execução terminar, o `blockend` no topo indicará o fim do escopo atual.

```
def blockend(val)
    $smc.popC()
    $smc.desempilhaAmbiente()
  end
```

Como o escopo terminou, precisamos desempilhar da pilha E os ambientes criados dentro desse bloco, a cada declaração que ocorreu.

Fim do bloco

Para saber quais ambientes desempilhar, guardamos no E-SMC uma pilha auxiliar, contendo listas de variáveis/constantes: a lista no topo indica quais foram criadas no bloco corrente.

```
proc teste(x) {
  var x = 3; var y = 2
  while (x > 0) do {
      const w = 1
      x := x - w                    <-
  }
}
```

A pilha auxiliar será `[["w"], ["y", "x"]]`, com a constante w criada dentro do while, e x e y criadas dentro do proc. Ao fim do while, um ambiente será desempilhado, e ao fim do proc, dois.