Introdução à Compilação: Linguagem *Straight-line*

José Romildo Malaquias

Departamento de Computação
Universidade Federal de Ouro Preto

2019.2

1 A linguagem *straight-line*

Tópicos

Straight-line

- *Straight-line* é uma micro linguagem de programação usada na série de livros *Modern Compiler Implementation* escritos por Andrew Appel.
- É uma linguagem de comandos e expressões muito simples, sem laços e estruturas condicionais.
- A sintaxe é indicada por uma gramática livre de contexto.
- Um programa é um comando.
- Exemplo de programa:

```
a := 5+3;
b := (print(a, a-1), 10*a);
print(b)
```

- Quando executado, este programa produz a saída:

```
8 7
80
```

Sintaxe de *straight-line*

Stm → *Stm* `;` *Stm*	**CompoundStm**
Stm → `id` `:=` *Exp*	**AssignStm**
Stm → `print` `(` *ExpList* `)`	**PrintStm**
Exp → `id`	**IdExp**
Exp → `num`	**NumExp**
Exp → *Exp* *Binop* *Exp*	**OpExp**
Exp → `(` *Stm* `,` *Exp* `)`	**EseqExp**
ExpList → *Exp* `,` *ExpList*	**PairExpList**
ExpList → *Exp*	**LastExpList**
Binop → `+`	**Plus**
Binop → `-`	**Minus**
Binop → `*`	**Times**
Binop → `/`	**Div**

Semântica informal de *straight-line*

Comando composto s_1 `;` s_2

executa os comandos s_1 e s_2 em sequência

Comando de atribuição i `:=` e

avalia a expressão e, e então armazena o resultado na variável i

Comando print `print(`e_1`,` e_2`,` $\cdots$`,` e_n`)`

exibe os valores de todas as expressões avaliadas da esquerda para a direita, separados por por espaço, terminando com uma mudança de linha

Variável i

valor atual da variável i

Constante n

o próprio valor numérico n

Operação binária e_1 op e_2

avalia e_1, e então avalia e_2, e então aplica a operação indicada aos valores obtidos

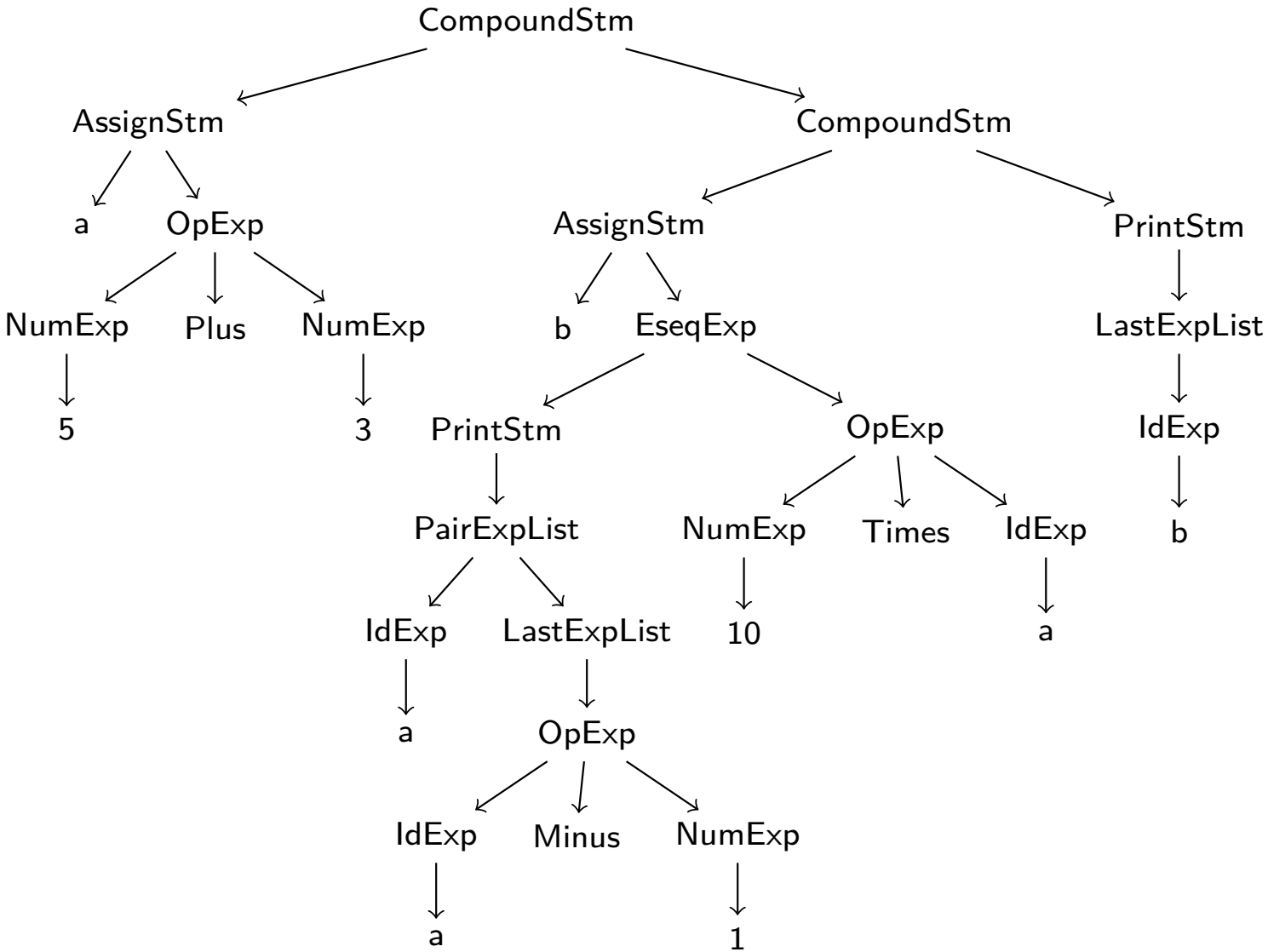
Expressão sequência `(`s`,` e`)`

executa o comando s (pelos seus efeitos colaterais), e então avalia a expressão e.

Representação de programas internamente no compilador

- Código fonte (sequência linear de caracteres): inconveniente
- Estrutura de dados **árvore**, com um nó para cada frase no programa
  - comandos: *Stm*
  - expressões: *Exp*
- Os símbolos do lado direito da regra de produção correspondem a subárvores.
- A gramática pode ser traduzida diretamente para definições de estruturas de dados:
  - A cada símbolo gramatical corresponde um tipo de dados.
  - Para cada regra de produção há um construtor de dados que pertence ao tipo correspondente ao seu lado esquerdo.

Árvore sintática



Tipos na linguagem OCaml

```
type id = string

type binop = Plus | Minus | Times | Div

type stm = CompoundStm of stm * stm
         | AssignStm of id * exp
         | PrintStm of exp list

and exp = IdExp of id
        | NumExp of int
        | OpExp of exp * binop * exp
        | EseqExp of stm * exp
```

Atividades

1. Escrever o programa ilustrado anteriormente como um valor em OCaml.
2. Definir uma função `maxargs` que recebe um comando e resulta no número máximo de argumentos fornecidos em algum `print` neste comando.

Exemplos de representação de programas

```
(*
  peso := 73
 *)

let prog1 = AssignStm ("peso", NumExp 73)
```

Exemplos de representação de programas (cont.)

```
(*
  print(43, 7, 0)
 *)

let prog2 = PrintStm [NumExp 43; NumExp 7; NumExp 0]
```

Exemplos de representação de programas (cont.)

```
(*
  x := 2 + 3;
  print(x)
 *)

let prog3 =
  CompoundStm (AssignStm ("x", OpExp (NumExp 2, Plus, NumExp 3)),
               PrintStm [IdExp "x"])
```

Exemplos de representação de programas (cont.)

```
(*
  x := 2 + 3 * 4;
  print(x)

 *)

let prog4 =
  CompoundStm (AssignStm ("x",
                          OpExp (NumExp 2,
                                 Plus,
                                 OpExp (NumExp 3,
                                        Times,
                                        NumExp 4))),
               PrintStm [IdExp "x"])
```

Interpretador de *straight-line*

1. Para representar a memória use uma tabela hash. Consulte a documentação do módulo Hashtbl no manual da linguagem OCaml.
   - defina o tipo `memory` usando o construtor de tipo `Hashtbl.t`
   - a função `Hashtbl.replace` será usada para atribuir um valor a uma variável na memória
   - a função `Hashtbl.find_opt` será usada para pesquisar uma variável na memória
2. Definir uma função `run` que recebe uma memória e um comando como argumentos, e executa o comando, resultando na tupla vazia.
3. Será necessária uma função auxiliar `eval` que recebe uma memória e uma expressão como argumentos, e avalia a expressão, resultando no valor da mesma.