Universidade de Évora

Compiladores

1ª Frequência - 17 de Março de 2015 - **correcção**

Primeira Parte - Aquecimento

(0.5) 1. O flex produz:

A. Analisadores semânticos, em C

B. Analisadores sintácticos, em Java

C. Analisadores lexicais, em C

D. Compiladores para código MIPS

(0.5) 2. Em C ou Java, a sintaxe `a = b = 2 + 1` é válida. Isso acontece porque:

A. `2 + 1` é uma expressão

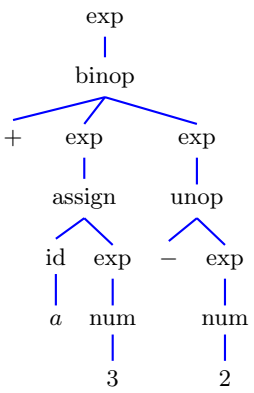
B. `a = b` é o nome de uma variável à qual se afecta `2 + 1`

C. uma afectação é uma expressão e produz um valor

D. uma afectação é uma *statement* e tem uma sintaxe abstracta

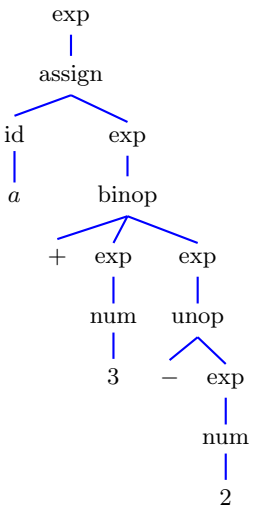
3. As APTs das seguintes expressões estão correctas?
Justifique, no caso de não estarem correctas.

(1) (a) `a = 3 + -2`

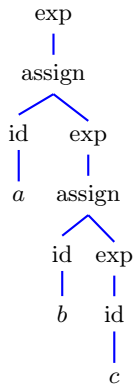


Solução: Esta APT está errada, pois resulta da avaliação da expressão como se estivesse na forma `(a = 3) + (-2)` (provavelmente devido a um erro na definição das precedências, na gramática → '=' com mais precedência que '+').

A APT correcta seria do tipo:

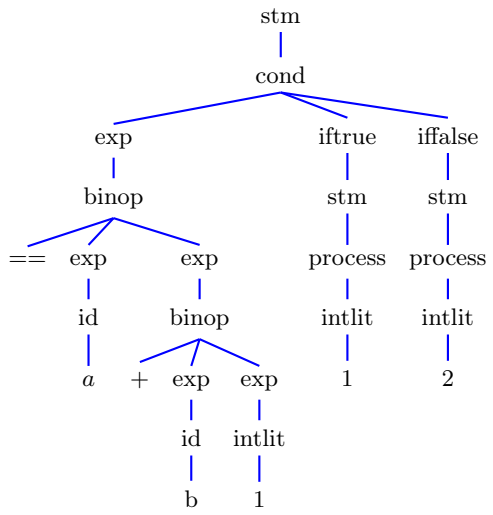


(1) (b) `a = b = c`



Solução: Esta APT está correcta, assumindo que afectações são expressões, produzindo um valor.

(1.5) (c) `if a == b+1 then process(1) else process(2)`



Solução: A parte esquerda está correcta (o operador '==' é um operador binário, podendo ser representado como *binop* → `a == b + 1` é uma expressão com valor booleano).

A parte direita, porém, omite várias coisas:

- a função/procedimento `process()` é um *builtin*? Se sim, só pode receber `intlit`s, o que não parece fazer muito sentido...
- se a função/procedimento `process()` for definida por nós, então a sintaxe abstracta deveria reflectir isso: chamada a função, argumentos, que no final deveriam ser expressões.

Segunda Parte - Aceleração

(2.5) 4. Considere os literais numéricos de vírgula flutuante, presentes na maioria das linguagens de programação:

- têm sempre um ponto decimal (**.**);
- o zero antes do ponto é opcional (**1.2**, **.234**);
- a parte decimal é obrigatória (~~**23.**~~, ~~**0.**~~);
- são permitidos números em notação científica (**2.443E50**, **2.443e-20**, **.34e+12**):
  - o *E* pode ser maiúsculo ou minúsculo;
  - o número que segue o *E* é sempre inteiro, e pode ter um sinal + ou –.

Indique uma expressão regular para reconhecer qualquer literal de vírgula flutuante, nesta forma.

Solução:

```
[0-9]*\.[0-9]+([eE][\-\+]?[0-9]+)?
```

5. Considere um *subset* de uma nova linguagem de programação, *yalang*, com a seguinte sintaxe:

- Tipos atómicos: `int, float, string`
- Literais (com a forma "habitual")
- Variáveis (idem)
- Declarações: `variável : tipo`
  ou
  `variável : tipo = valor` (o valor pode ser qualquer expressão válida)
- Expressões: afectações (símbolo =) e operações matemáticas
- Funções pré-definidas (constituem palavras reservadas): `output(expressão)` e `input(variável)` (as chamadas a estas funções são statements, sem qualquer valor)
- O separador de instruções é o `;` (ponto-e-vírgula)

Exemplo:

```
a : int;
b : string = "Hello, world";
input(a);
c : float = a + b;
output(c * 4 + 1);
```

(3) (a) Proponha uma gramática para esta linguagem.

Solução:

```
program:          stmlist
;
stmlist:          stm ';'      /* podia ser vazio */
          |       stm ';' stmlist
;
stm:              decl
          |       assign
          |       fcall
;
decl:             ID ':' TYPE
          |       ID ':' TYPE '=' exp
;
assign:           ID '=' exp    /* esta é opcional, não foi pedida */
;
fcall:            OUTPUT '(' exp ')'
          |       INPUT '(' ID ')'
;
exp:              ID
          |       LITERAL
          |       assign
          |       exp '+' exp
          |       exp '-' exp
          |       exp '/' exp
          |       exp '*' exp
          |       '(' exp ')'
;
```

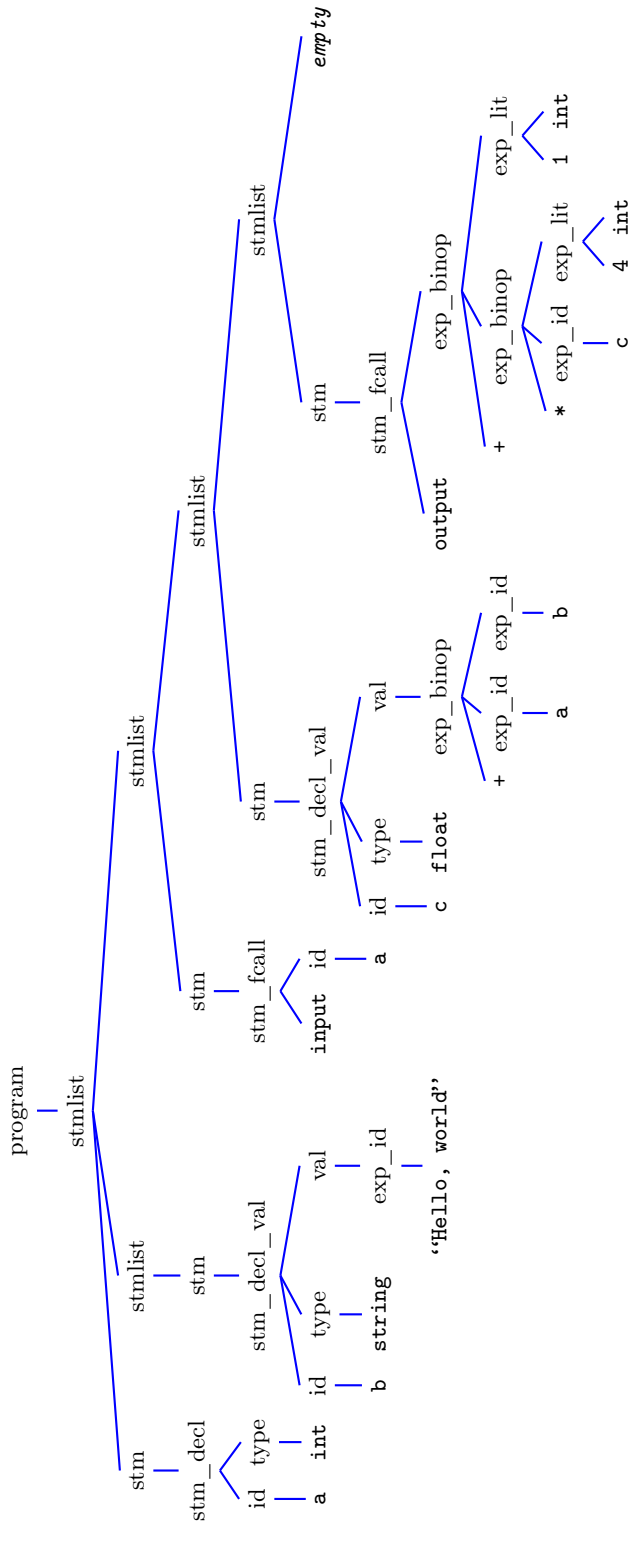
(3) (b) Proponha uma sintaxe abstracta para a linguagem.

Solução: A sintaxe abstracta vem quase directamente da gramática, mas podemos eliminar algumas redundâncias...

```
program(stmlist)

stmlist(stm, stmlist)
stmlist(stm, empty)

stm_decl(id, type)
stm_decl_val(id, type, val)

stm_assign(id, exp)
stm_fcall(output, exp)
stm_fcall(input, id)

exp_id(id)
exp_lit(literal, type)
exp_assign(id, exp)
exp_binop(op, exp, exp)
exp_par(exp)    /* opcional, podemos colocar exp no nível superior */
```

(3) (c) Mostre a APT para o exemplo fornecido, de acordo com a sintaxe abstracta que propôs na alínea anterior.

Solução:



Terceira Parte - Turbo

(2) 6. Considerando ainda o exemplo da linguagem *yalang* e, assumindo que a análise semântica continua apesar dos erros, diga um resultado possível para a análise semântica do exemplo dado.

Solução: Exemplo de output:

```
Erro na linha 4: operador '+' não suportado entre 'int' e 'string'
```

7. Relativamente à linguagem *yalang* e à gramática que propôs anteriormente, dê um exemplo de uma declaração:

(1) (a) com um erro sintáctico.

Solução: Qualquer *coisa* que a gramática não contemple, por exemplo:

- `int a = 1;`
- `123 :  int;`
- `o rato comeu o gato.`

(1) (b) com um erro sintáctico que não seja detectado pela gramática.

Solução: Não é possível haver erros sintácticos que uma gramática não detecta. Pelo contrário, se a gramática não detecta, é porque não existe erro (de acordo com a gramática...).