Graduação em Ciência da Computação
Disciplina: Compiladores
Professor: Guilherme Galante

Lista de Exercícios - Segunda Avaliação

1. Considere a gramática aumentada G:
0.S'→ E
1.E → BB
2.B → cB | d

a) Construa o conjunto de itens LR(0) para G.
b) Construa o autômato para o conjunto de itens.
c) Construa a tabela de análise sintática SLR.

2. Considere a gramática aumentada G:
0. S'→ S
1. S → aSbS
2. S → a
cujo conjunto de itens LR(0) é:

I_0
S'→ .S
S → .aSbS
S → .a

I_1
S' → S.

I_2
S → a.SbS
S → a.
S → .aSbS
S → .a

I_3
S → aS.bS

I_4
S → aSb.S
S → .aSbS
S → .a

I_5
S → aSbS.

e os conjuntos First e Follow são, respectivamente:
FIRST(S) = {a}
FOLLOW(S) = {b, $}

Construa a tabela de análise sintática SLR.

3. Assinale V ou F e justifique as alternativas falsas.
[ ] Um esquema S-atribuído utiliza apenas atributos herdados.
[ ] Um esquema L-atribuído pode combinar atributos herdados e sintetizados.
[ ] Um atributo herdado é aquele cujo valor é calculado em termos do pai ou irmão daquele nó na árvore gramatical.
[ ] Não é necessário alterar o esquema de tradução quando elimina-se a recursão à esquerda de uma gramática. O próprio processo de eliminação de recursão já faz as adequações necessárias.
[ ] Uma conversão de tipo realizada automaticamente pelo compilador é chamada de coerção e geralmente não há perda de informações após a conversão.
[ ] O uso de uma variável de ponto flutuante para indexação de um vetor causa um erro detectado geralmente na análise sintática bottom-up.
[ ] Na equivalência de tipos estrutural, duas variáveis são compatíveis se declaradas exatamente com o um tipo de mesmo nome.

4. Considere a gramática G abaixo e a sua tabela de análise sintática SLR:

0.S'→ E
1.E → E + T
2.E → T
3.T → T * F
4.T → F
5.F → (E)
6.F → a

	E	T	F	(	a	+	*	)	$
0	1	2	3	e4	e5	-	-	-	-
1	-	-	-	-	-	e6	-	-	*r0*
2	-	-	-	-	-	*r2*	e7	*r2*	*r2*
3	-	-	-	-	-	*r4*	*r4*	*r4*	*r4*
4	8	2	3	e4	e5	-	-	-	-
5	-	-	-	-	-	*r6*	*r6*	*r6*	*r6*
6	-	9	3	e4	e5	-	-	-	-
7	-	-	10	e4	e5	-	-	-	-
8	-	-	-	-	-	e6	-	e11	-
9	-	-	-	-	-	*r1*	e7	*r1*	*r1*
10	-	-	-	-	-	*r3*	*r3*	*r3*	*r3*
11	-	-	-	-	-	*r5*	*r5*	*r5*	*r5*

Realize a análise passo-a-passo da cadeia (a + a) * a

5. O esquema de tradução dirigida por sintaxe que segue traduz uma linguagem com terminais a, b, c, d em uma linguagem cujos terminais são 1,2,3,4,5,6. Usando um parser bottom-up que executa as ações entre chaves imediatamente após reduzir a regra correspondente, qual o resultado da tradução “aaadbc”?

S → AS{print “1”}
S → B {print “2”}
A → a {print “3”}
B → bC {print “4”}
B → dB {print “5”}
C → c {print “6”}

6. Considere o esquema de tradução abaixo:

```
E → E1 + T        {E.ptr:=geranodo("+", E1.ptr, T.ptr)}
E → E1 - T        {E.ptr:=geranodo("-", E1.ptr, T.ptr)}
E → T             {E.ptr:=T.ptr}
T → (E)           {T.ptr:=E.ptr}
T → id            {T.ptr:=gerafolha ("id", id.nome)}
T → num           {T.ptr:=gerafolha ("num", num.val)}
```

a) Mostre a construção da árvore usando o esquema acima para a expressão “num*(id-id)” usando análise Bottom-Up.
b) Por que o esquema não pode ser usado para análise Top-Down?
c) Faça as transformações necessárias para que possa ser analisado por um esquema Top-Down.

7. Conceitue e dê exemplos para:
a) Expressão de tipos;
b) Equivalência estrutural e de nome;
c) Coerção.

8. Considere o código abaixo em Pascal.

```
Soma (x,y); {Chamada da função em outro escopo}

...

function Soma(a, b : Integer) : Integer;
var
  i : Integer;
  w : Integer;
  z : Real;
begin
  i := a + b;
  w := i mod z;
  Soma := w;
end;
```

a) Aponte quais verificações o analisador semântico deverá realizar nesse código.
b) Há erros de tipagem nesse código?

9. Selecione uma linguagem de programação de sua escolha e analise seu sistema de tipos de acordo com os 4 elementos apresentados em aula.

10. A regra “definir antes de usar” especifica que cada variável usada em um procedimento deve ser declarada antes que apareça no texto. Esboce um esquema de gramática de atributo para verificar se um procedimento está de acordo com esta regra. O problema é mais fácil se a linguagem exigir que todas as declarações precedam qualquer comando executável?

11. Algumas linguagens, como APL ou PHP, não exigem declaração de variável nem forçam a consistência entre atribuições para a mesma variável (Um programa pode atribuir um inteiro e depois uma string a uma mesma variável). Este estilo de programação às vezes é chamado malabarismo de tipos. Suponha que você tenha uma implementação existente de uma linguagem que não possui declarações, mas exige usos consistentes em relação ao tipo. Como você a modificaria para permitir o malabarismo de tipos?

12. Que tipos de informação o compilador deve ter para garantir a segurança de tipos nas chamadas de procedimento? Esboce um esquema com base no uso de protótipos de função. Esboce um esquema que possa verificar a validade desses protótipos de função.

+ Exercícios Dragão 2ª edição:

4.5.1 4.5.3 4.6.4

5.1.1 5.2.2 5.3.1 (a) 5.4.3

[Desafio] Um programa em Pascal pode declarar duas variáveis inteiras a e b com a sintaxe:

var a, b: integer

Esta declaração poderia ser descrita com a seguinte gramática:

```
<VarDecl> → var <IDList> : <TypeID>
<IDList>  → <IDList>,ID| ID
<TypeID>  → integer | real
```

onde IDList deriva uma lista de nomes de variável separados por vírgulas, e TypeID deriva um tipo válido em Pascal (integer, real, p.ex).

É possível escrever (de alguma forma) um esquema de tradução atribua o tipo de dado correto a cada variável declarada? Considere a avaliação em um analisador Bottom-Up.