Departamento de Informática

2º Ciclo Engenharia Informática
Linguagens de Programção e Compiladores
2023/2024

Nome: Lab#05 | 09/11/2023

Número de Aluno: Duração: 120 min. + 5 min. tolerância

Tópicos

- *Gramáticas Atributivas.*

Exercício 1

Pretende-se controlar um semáforo de 3 estados: Encarnado, Amarelo e Verde, representados pelos valores numéricos 2, 1 e 0, respectivamente.

O semáforo é controlado por um temporizador que emite, regularmente, o token **NEXT** que faz o semáforo evoluir para o estado seguinte, na sequência (Encarnado, Verde, Amarelo e novamente Encarnado). O semáforo tem um botão de pânico que gera o token **PANIC** e coloca o semáforo no estado Encarnado, independentemente do estado anterior. O estado inicial do sistema é Encarnado.

Construa a gramática atributiva que permite controlar o estado do semáforo (considere apenas os tokens indicados). Indique que tipo de gramática atributiva que obteve. Realize a árvore semântica anotada para a sequência de tokens **NEXT, NEXT, PANIC e NEXT**.

Exercício 2

A gramática seguinte admite atribuições dentro de expressões.
S -> E
E -> E + E | E := E | (E) | id

Quando uma expressão está a ser analisada é necessário diferenciar os casos onde a expressão está à direita, daqueles onde a expressão está à esquerda, do sinal de atribuição. Note que não é possível atribuir um valor a uma expressão. Por exemplo a expressão **id + id := id** é ilegal. No entanto, as expressões **id := (id + id)**, quer **id := (id := id)**, ou ainda **(id) := (id := (id + id))** são autorizadas.

Considere que o operador **:=** é menos prioritário que o operador **+**, além de associativo à direita, enquanto o operador **'+'** é associativo à esquerda.

Escreva um esquema de tradução que usa um atributo herdado (e nenhum sintetizado) chamado **lado** associado ao símbolo **E**, e que determina se a expressão corrente está à direita ou à esquerda da atribuição. O código associado à gramática emitirá uma mensagem de erro caso a expressão seja ilegal.

Exercício 3

Pretende-se criar uma gramática atributiva que some uma sequência de inteiros separados por **+**. Os inteiros podem estar codificados sob o formato decimal (sequência de dígitos entre 0 e 9, representados pelo elemento lexical **DIG**), ou em qualquer base entre 2 e 36 (sequências de dígitos de 0 a 9 e, acima de 10,

também contendo as letras de A a Z que forem necessárias para representar todos os valores da base, também representados pelo elemento lexical **DIG**). Quando a base não é decimal, a representação do inteiro é precedida pelo valor decimal da base e pelo símbolo #.

O exemplo seguinte tem como resultado 241+33=274.
2 4 1 + 1 7 # 1 G

Crie a gramática para realizar a função descrita e construa a árvore semântica anotada para a entrada acima.

Exercício 4

Considere uma linguagem constituída por uma sequência, não vazia, de linhas. Cada linha é constituída por uma sequência, possivelmente vazia, de caracteres. Pretende-se determinar o número de controlo de erro (**CRC**) de cada linha de texto. Para tal, o analisador lexical produz dois tipos de lexemas: **CHR** (número inteiro que representa o código do carácter lido); **EOL** (fim de linha). Escreva um programa que imprima, após o fim de cada linha (**EOL**), o correspondente **CRC** (iniciado a 0 em cada linha, é calculado através de um "ou exclusivo" com o valor inteiro do carácter lido).

A. Identifique uma gramática que reconheça a linguagem descrita.

B. Determine os atributos de cada símbolo e indique, justificadamente, o tipo de gramática obtida.

C. Represente a árvore decorada para a frase seguinte, indicando a sequência de valores impressos no seu processamento.
CHR.i=12 CHR.i=7 CHR.i=9 EOL EOL CHR.i=92 EOL

D. Escreva uma especificação YACC que realize a solução do problema acima. A gramática não deve ter conflitos.

Exercício 5

Pretende-se criar uma gramática atributiva que calcule no símbolo inicial o valor das expressões fornecidas. As expressões são codificadas (i) como números inteiros (i.e., sequências de algarismos em base 10, cada um representado pelo token **DIG**, ao qual está associado o atributo **val** com o valor do algarismo); (ii) como somas ou subtracções unárias de uma unidade a uma expressão (respectivamente, **>** e **<**); (iii) somas ou subtracções binárias (respectivamente, # e **!**) de expressões; ou (iii) como percentagens de expressões (indicada como uma expressão, após o operador @).

Os operadores unários têm precedência sobre todos os operadores binários. O operador @ tem precedência superior aos operadores # e **!**. Todos os operadores binários são associativos à esquerda. Podem ser utilizados parênteses para alteração das precedências.

Exemplos:

- 37 é representado pela sequência **DIG DIG** (o primeiro token tem o atributo val com valor 3 e o segundo com o valor 7).
- **4>** tem o valor 5
- **<3** tem o valor 2
- **36@20** tem o valor 7.2
- **(24>#25)@20#7!<15#6** tem o valor 9

A. Identifique a gramática atributiva correspondente ao problema. Descreva o significado e o tipo de cada atributo. Que tipo de gramática obteve?

B. Identifique a árvore de sintaxe decorada e o grafo de dependências para a frase **(15@30#<3)>!4** (valor 3.5).

C. Escreva uma especificação YACC que implemente a gramática descrita em 1. Codifique toda a especificação (incluindo as zonas de declarações e de regras) e todas as funções auxiliares. Não utilizar variáveis globais.