Lista I - Compiladores –as de 96.2 com 2000.1 - Rangel

Atenção! questões interessantes, que foram vistas em TC ou em Compiladores, mas que não vão cair na P2: 1-a, 1-b, 5-a, 5-b, 6, 8, 11(vale a pena explorar, caso tenha tempo), e 13.

1-a. No "Report" do Pascal, Niklaus Wirth define "números" (constantes dos tipos `integer` e `real`), através das seguintes regras:

```
<digit sequence> ::= <digit>{<digit>}
<unsigned integer> ::= <digit sequence>
<unsigned real> ::= <unsigned integer> . <digit sequence> |
   <unsigned integer> . <digit sequence> E <scale factor> |
   <unsigned integer> E <scale factor>
<unsigned number> ::= <unsigned integer> | <unsigned real>
<scale factor> ::= <unsigned integer> |
   <sign> <unsigned integer>
<sign> ::= + | -
```

Denote os números inteiro e real por expressões regulares.

1-b. Você foi encarregado de fazer parte da especificação léxica de uma linguagem de programação, mais precisamente a parte referente a números. As regras são as seguintes:

- números podem vir em base 2, 8, 10 e 16, usando os dígitos da forma habitual. Ou seja, os conjuntos de dígitos para os quatro casos são, respectivamente, {0, 1}, {0, ..., 7}, {0, ..., 9} e {0, ..., F}.
- formato empregado deve ser não ambíguo, isto é, não deve permitir confusão entre os diversos formatos de número, nem entre números e identificadores e palavras reservadas da linguagem. Identificadores e palavras reservadas são formados por letras e dígitos (0 ... 9), sempre começando com letra.

Complete o projeto, e apresente:

- a descrição dos formatos a serem usados para números
- a especificação (por exemplo através de um diagrama de estados) do analisador léxico.

Já caiu no ENADE questão próxima a esta.

2. Classifique a gramática abaixo, dizendo se ela é LR(1), LL(1), ambígua, sLR(1), laLR(1). Justifique todas as suas respostas.

```
L → ( S )
S → S a
   | S L
   | ε
```

3. Mostre que a gramática abaixo não é ambígua.

```
S → A a a
   | B a b
A → c
B → c
```

4. Mostre que a gramática abaixo não é LL(1), mas que existe uma gramática LL(1) equivalente a ela. Justifique sua resposta.

```
L → L ; S | S
S → if E th L el L fi | if E th L fi | s
```

5. a. Considere uma linguagem em que as declarações de variáveis tem o seguinte formato:

`iden0 iden1, iden2, ..., idenn ;`

devendo `iden0` ser um identificador de um tipo e `iden1`, `iden2`, `...`, etc identificadores de variáveis declaradas do tipo `iden0`. Escreva uma gramática que defina a sintaxe dessas declarações.

5.b. Considere o fragmento de gramática abaixo e explique porque o tratamento do símbolo "ponto-e-vírgula" é diferente para as declarações e para os comandos.

```
<bloco> ::= BEGIN <decls> <coms> END
<decls> ::= <decls> <decl> ; | <vazio>
<decl> ::= ....
....
<coms> ::= <coms> ; <com> | <com>
<com> ::= ....
....
<vazio> ::=
```

6. Suponha uma linguagem L, cuja parte léxica está especificada a seguir:

- identificadores de L podem ser compostos de letras, dígitos, sublinhado ("_"), e tralha ("#"), devendo o primeiro caracter ser sempre uma letra. Como C, L faz distinção entre letras minúsculas e maiúsculas.
- Números em L podem ser inteiros e reais.
- Inteiros são compostos de dígitos, e, como em Ada, o caracter sublinhado pode ser usado como separador em posições arbitrárias. Por exemplo, em vez de 1000000000, podemos escrever 1_000_000_000, aumentando a legibilidade.
- Reais podem ter um expoente composto por um "E" seguido de um inteiro, possivelmente com um sinal. Como usual, um "." separa a parte inteira da parte real. O ponto é obrigatório, mas uma das duas partes (inteira e fracionária) pode ser vazia.
- As palavras reservadas da linguagem são **begin**, **end**, **if**, **then**, **maybe**, e **obaoba**. As palavras reservadas podem ser escritas com letras minúsculas, letras maiúsculas, ou qualquer combinação das duas.
- Os operadores e delimitadores são "+", "*", "(", ")", "::", "++", e "/?".
- Comentários podem ser de duas maneiras: de "--" até o fim da linha, e entre as combinações "/*" e "*/", possívelmente incluindo trechos de várias linhas.

Especifique as expressões regulares para os tokens acima.

7. Classifique as gramáticas seguintes:

7-a) $E \rightarrow E + T \mid E - T \mid T$
$T \rightarrow T * F \mid T / F \mid F$
$F \rightarrow (E) \mid \text{if } E \text{ then } E \text{ else } E \mid a$

7-b) $E \rightarrow E \text{ v } T \mid T$
$T \rightarrow T \text{ ^ } T \mid T$
$F \rightarrow \sim P \mid P$
$P \rightarrow P * \mid (E) \mid a$

7-c) $S \rightarrow Z)$
$Z \rightarrow Z , E \mid I (E$
$E \rightarrow e$
$I \rightarrow a$

Para cada gramática, à esquerda, diga se ela é ambígua, LL(1), sLR(1), laLR(1) ou LR(1). Sugestão: Como as classes de linguagens mencionadas são inter-relacionadas, vale a pena examinar com cuidado por onde começar.

8. Use o seu gerador de analisadores sintáticos favorito para gerar um parser para a gramática: Exercício para laboratório.

```
L → L ; C
    | C
C → V := E                      atribuição
    | if E then L X             if, com partes else e elsif opcionais
    | while E do L end
    | V                         chamada de procedimento
    | begin L end
    | ε                         comando vazio
    | goto I
    | I : C                     comando rotulado
X → else L end
    | elsif L X
    | end
E → B
    | A
B → B or B
    | B and B
    | not B
    | A = A
    | I                         variável booleana
E → E + E
    | E * E
    | id
V → V ( Z )
    | I
Z → Z , E                       lista de expressões (índices ou parâmetros)
    E
```

Se necessário adapte a gramática, para que seja aceita pelo analisador.

9. Considere a sintaxe abaixo para os comandos loop-end e exit.

S → loop L end | exit | V := E | if B then S

L → L ; S | S

Modifique a sintaxe, de forma a garantir que

(1) todo comando loop tenha pelo menos um comando exit

(2) nenhum comando exit possa existir fora de um loop.

10. A gramática abaixo é ambígua. Construa um analisador "sLR(1)-like" para ela resolvendo manualmente os conflitos encontrados na construção do analisador sLR(1).

```
E → E ∨ T | T
T → T ∧ F | F
F → ( E ) | if E then F else F | if E then F | a
```

11. Classifique a gramática a seguir, indicando se é LL(1), sLR(1), laLR(1), LR(1).

```
S → { Ls }
   | V := E
   | if E then S else S
   | K
Ls → Ls ; S
   | S
E → E + T
   | T
T → T * F
   | F
F → ( E )
   | a
K → I ( Le )
   | I
V → I [ Le ]
   | I
Le → Le , E
   | E
I → id
```

12. A gramática a seguir é sLR(1)? Justifique sua resposta.

```
S → V := E | I ( E )
E → E + V | V
V → I ( E ) | I
I → a
```

13. Em algumas linguagens, brancos podem ser usados em qualquer ponto de um programa, sem nenhum significado, exceto dentro de uma cadeia de símbolos. Suponha que essa regra é acrescentada à linguagem Pascal, de forma que, por exemplo, as duas linhas a seguir são equivalentes:

```
XR3:=XR+3;
X   R   3   :   =   X   R   +   3   ;
```

Como, e por qual parte do compilador, pode ser tratada essa complicação adicional? A complicação vale a pena?

19/9/96