Compiladores

Lista 1

José Lucas Rangel
29/04/00

1. (censurado)
2. 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?
3. Considere uma linguagem em que as declarações de variáveis tem o seguinte formato:
   $\texttt{iden}_0\ \texttt{iden}_1\texttt{, iden}_2\texttt{, ..., iden}_n\ \texttt{;}$

devendo $\texttt{iden}_0$ ser um identificador de um tipo e $\texttt{iden}_1$, $\texttt{iden}_2$, ..., etc identificadores de variáveis declaradas do tipo $\texttt{iden}_0$. Escreva uma gramática que descreva a sintaxe dessas declarações.

4. 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> ::= + | -
   ```

Explique como esta especificação pode ser usada para programar a parte correspondente de um analisador léxico de Pascal.

5. Classifique a gramática abaixo, dizendo se ela é LR(1), LL(1), ambígua, sLR(1). Justifique todas as suas respostas.
   ```
   L → ( S )
   S → S a
      | S L
      | ε
   ```
6. Mostre que a gramática abaixo não é ambígua.
   ```
   S → A a a
      | B a b
   A → c
   B → c
   ```
7. Mostre que a gramática abaixo não é LL(1), mas que existe uma gramática LL(1) equivalente a ela.
   ```
   L → L ; S
   ```

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

Justifique sua resposta.

8. 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> ::=
```

9. 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.
- explique a “rationale” do seu projeto.

10. A gramática a seguir não é LL(1). Apresente uma gramática LL(1) equivalente a ela. Justifique sua resposta.

```
S → if E then L X Y end | s
E → e
L → L ; S | S
X → X elsif L | ε
Y → else L | ε
```

11. Classifique a gramática abaixo, dizendo se ela é LL(1), LR(1), laLR(1), sLR(1), ambígua.

```
S → Z )
Z → Z , E | I ( E
E → e
I → a
```

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

```
L → L ; S
   | S
S → loop L end
```

```
    | exit
    | V := E
    | if B then S
```

Modifique a sintaxe de forma a garantir que

- todo comando `loop ... end` tem pelo menos um comando `exit`
- nenhum comando `exit` pode existir fora de um `loop ... end`

13. 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
```

14. Escreva um analisador recursivo para a gramática de atributos:

```
S → A B C
   S.ok := B.ok and C.ok;
   B.m := A.n;
   C.m := A.n;
A⁰ → a A¹
   A⁰.n := A¹.n +1;
A → ε
   A.n :=0;
B⁰ → b B¹
   B⁰.ok := B¹.ok;
   B¹.m := B⁰.m -1;
B → ε
   B.ok := (B.m=0);
C⁰ → c C¹
   C⁰.ok := C¹.ok;
   C¹.m := C⁰.m -1;
C → ε
   C.ok := (C.m=0);
```

15. Descreva a linguagem formada pelas cadeias para as quais `S.ok` é true.

16. Escreva uma gramática equivalente à gramática de (14), que só tenha atributos sintetizados. Equivalente significa que o valor do atributo `ok` na raiz é exatamente o mesmo para todas as cadeias.

Adicionalmente, sugiro ver no final dos capítulos do seu livro de compiladores favorito os exercícios correspondentes à matéria.