Primeira Prova de MAB 471 2013.1 — Compiladores I

Fabio Mascarenhas

27 de Maio de 2013

A prova é individual e sem consulta. Responda as questões na folha de respostas, a lápis ou a caneta. Se tiver qualquer dúvida consulte o professor.

Nome: _____

DRE: _____

Questão:	1	2	3	Total
Pontos:	2	6	2	10
Nota:				

Vimos em sala de aula que a notação *EBNF* para gramáticas estende a notação tradicional com construções de repetição, opcional, agrupamento e alternativa. EBNF pode ser vista como uma linguagem para descrever gramáticas, e portanto possui uma estrutura léxica e uma sintaxe, que por sua vez pode ser descrita usando uma gramática EBNF.

1. Podemos classificar a estrutura léxica de EBNF usando dez classes de tokens: terminais, não-terminais, `{`, `}`, `[`, `]`, `(`, `)`, `->` e `|`. Não-terminais são sequências de letras maiúsculas, terminais são sequências de letras minúsculas, strings delimitadas por aspas simples ou strings delimitadas por aspas duplas.
   (a) (1 ponto) Escreva a(s) regra(s) léxica(s) para terminais e não-terminais, no estilo do JFlex.
   (b) (1 ponto) Escreva um autômato finito determinístico para reconhecer terminais.

2. A gramática a seguir descreve um fragmento da gramática de EBNF, em EBNF:

```
EBNF  -> EBNF EBNF | REGRA
REGRA -> naoterm "->" EXP
EXP   -> EXP EXP
       | EXP "|" EXP
       | term
       | naoterm
```

   (a) (1 ponto) Mostre que essa gramática é ambígua.
   (b) (1 ponto) Modifique `REGRA` para permitir regras com produção vazia.
   (c) (2 pontos) Modifique `EXP` para permitir as construções que estão faltando: agrupamento, repetição, opcional. Pode deixar a gramática ambígua.
   (d) (2 pontos) Reescreva a gramática (sem as mudanças feitas nas partes b e c) para não ser mais ambígua, corrigindo a precedência da barra em relação à justaposição. Faça a operação de justaposição ser associativa à **direita**. A gramática final não precisa ser LL(1)!

3. A gramática EBNF abaixo representa um fragmento de uma linguagem estilo TINY:

```
CMD -> read id
     | write EXP
     | id ":=" EXP
     | id "(" ")"
EXP -> num | id | "(" EXP "+" EXP ")"
```

   (a) (1 ponto) Por que essa gramática não é LL(1)? Reescreva a gramática mudando apenas o necessário para que ela se torne LL(1).

   (b) (1 ponto) Escreva o pseudocódigo para analisar o não-terminal `EXP` de forma recursiva sem retrocesso. Lembre-se de construir a árvore corretamente.

BOA SORTE!