Primeira Prova de MAB 471 2017.2 — Compiladores I

27 de Setembro de 2017

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	2	6	10
Nota:				

1. (2 pontos) Comentários de múltiplas linhas na linguagem Lua começam com `--[[`, opcionalmente contendo uma sequência de caracteres `=` entre o primeiro e segundo colchetes, e vão até `]]`, também possivelmente com uma sequência de `=` *com o mesmo tamanho da sequência usada na abertura do comentário*: um comentário começando com `--[[` vai até `]]`, um começando com `--[=[` vai até `]=]`, um começando com `--[==[` até `]==]`, e assim por diante. Não há aninhamento. Essa sintaxe de comentários pode ser expressa por uma expressão regular? Justifique sua resposta.

2. (2 pontos) Os numerais inteiros na linguagem Ruby podem ter caracteres `_` no seu interior (por exemplo, `123456789`, `1_234_567_89` e `123_456_789` são todos numerais válidos), contanto que o numeral não comece ou termine com `_` e não existam dois `_` em sequência. Escreva uma expressão regular para os numerais inteiros de Ruby, e um autômato finito determinístico para reconhecê-los.

3. As duas gramáticas abaixo mostram um pequeno fragmento das gramáticas das linguagens Java e Kotlin (respectivamente), em EBNF, com não-terminais em maiúsculas e terminais em minúsculas e entre aspas simples:

```
STMT -> TYPE id ['=' EXP] ';'
      | VAR '=' EXP ';'
      | '{' { STMT } '}'
TYPE -> id ['<' TARGS '>']
VAR  -> id {'.' id}
```

```
STMT -> var id [':' TYPE] ['=' EXP] [';']
      | VAR '=' EXP [';']
      | '{' { STMT } '}'
TYPE -> id ['<' TARGS '>']
VAR  -> id {'.' id}
```

   (a) (2 pontos) Diga se cada um dos dois fragmentos dados é ou não LL(1), justificando sua resposta.

   (b) (4 pontos) Dê o pseudocódigo para um analisador recursivo para cada um dos dois fragmentos. Um dos analisadores deve ser preditivo e o outro com retrocesso, mas você é livre para escolher qual será o quê (apenas documente a escolha que foi feita). Não é necessário construir a árvore sintática. Para o analisador com retrocesso, assuma que a posição atual está no campo `pos`, e que o método `match(`*tipo*`)` avança para a próxima posição se o token na posição atual tiver o tipo dado, ou falha (uma exceção `Falha`) caso contrário. Para o analisador preditivo, assuma que o token de lookahead está no campo `la`, e que o método `match(`*tipo*`)` avança para a próxima posição se o token de lookahead tiver o tipo dado, ou aborta com um erro caso contrário.

BOA SORTE!