Prova com consulta. Duração: 1h30m. Primeiro Mini-Teste

```
S →  "url" "(" STRING ")"            // basic services
   | S "?" S                         // sequential execution
   | S "|" S                         // concurrent execution
   | "timeout" "(" REAL "," S ")"    // timeout combinator
   | "repeat" "(" S ")"              // repetition
   | "stall"                         // nontermination
   | "fail"                          // failure
```

Trecho da gramática apresentada em [Cardelli and Davies, 1999]

Grupo 1. Análise Lexical e Sintáctica (11 valores)

Considere a gramática apresentada à esquerda na qual as produções estão comentadas (texto iniciado com os símbolos "//").

1.a) [1v] Apresente as expressões regulares para definir os tokens STRING e REAL para a gramática apresentada. Considere que STRING deve representar possíveis URLs HTTP (considere que podem conter "/", ".", ":", letras do alfabeto, e algarismos) entre aspas duplas, e REAL deve representar números inteiros ou reais, ambos sem sinal. Use o "." para separar a parte inteira da fraccionária dos números reais.

1.b) [2v] Desenhe um DFA para a análise lexical.

1.c) [1v] A gramática apresentada é ambígua? Caso seja, desenhe duas árvores sintácticas diferentes para uma dada entrada.

1.d) [1v] A gramática tem recursividade à esquerda. Elimine a recursividade e indique a gramática modificada para a linguagem.

1.e) [0,5v] Apresente a árvore sintáctica concreta para a entrada:

repeat(url("http://www.fe.up.pt") ? timeout(10, stall)).

1.f) [1v] Apresente uma possível árvore sintáctica abstracta (AST[1]) para o exemplo da alínea anterior.

1.g) [2v] Apresente as funções necessárias e o respectivo pseudo-código para implementar a linguagem definida pela gramática apresentada com um analisar sintáctico descendente preditivo LL(1). Assuma a existência do analisador lexical que devolve a sequência de tokens, a existência da variável global *token*, a função *next()* que devolve o token a seguir na sequência de tokens na entrada (no início a variável *token* identifica o primeiro token na sequência), e do atributo *id* de *token* que identifica o tipo de token. Indique os IDs que utilizou considerando os tipos de tokens utilizados.

1.h) [1v] Indique como poderemos adicionar as produções

"gateway" "get" "(" STRING ")"

e

"gateway" "post" "(" STRING ")"

à variável S mantendo um *lookahead* de 1.

1.i) [0,5v] Defina as produções para a variável *Args* para que esta substitua STRING nas produções da alínea anterior, e de forma a que *Args* possa derivar um ou mais terminais do tipo STRING separados pelo símbolo ",".

1.j) [0,5v] Apresente as modificações à gramática original de forma a que a derivação "timeout" "(" REAL "," S ")" possa aceitar opcionalmente a seguir a REAL a identificação das unidade de tempo a usar: "ms", "us", e "s".

1.k) [0,5v] Indique se a adição das produções

"limit" "(" REAL "," REAL "," S ")"

e

"index" "(" STRING "," STRING ")"

à variável S implica alterações no valor do *lookahead* do parser em 1.g). Justifique a resposta dada.

---

[1] Do Inglês: *Abstract Syntax Tree*

Grupo 2. Análise Semântica I (5 valores)

Considere o trecho de código Java do exemplo seguinte.

Exemplo

```
1.  void insertionSort(int[] num, int size) {
2.    int j;
3.    for (int i = 1; i < size; i++) {
4.       int key = num[i];
5.       j = i - 1;
6.       while((j>=0) && (key < num[j])) {
7.          num[j+1] = num[j];
8.          j--;
9.       }
10.      num[j+1] = key;
11.   }
12. }
```

2.a) [2v] Assumindo que as variáveis usadas no código só podem ser parâmetros ou variáveis locais, indique uma possível tabela de símbolos para este exemplo indicando a informação a incluir nos descriptores.

2.b) [2v] Indique a representação intermédia de alto nível para o exemplo tendo como referência a representação intermédia baseada em árvores de espressões apresentada nos slides das aulas teóricas.

2.c) [1v] Descreva possíveis verificações semânticas a ter em conta no código das linhas 6 a 9 e indique como poderiam ser realizadas essas verificações.

Grupo 3. Análise Semântica II (4 valores)

3.a) [2v] Explique se no contexto de linguagens de programação, o uso de gramáticas livres de contexto (CFGs[2]) permite incluir verificações semânticas.

3.b) [2v] Indique exemplos de verificações semânticas que eventualmente só podem ser feitas em tempo de execução. Justifique sucintamente os exemplos indicados.

(Fim.)

[2] Do Inglês: Context-Free Grammars.