Terceira Prova de MAB 471 2012.2 — Compiladores I

Fabio Mascarenhas

13 de Março 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:	3	3	4	10
Nota:				

1. *JSON* (JavaScript Object Notation) é um padrão para comunicação de dados entre aplicações que é bastante usado em aplicações web. A sintaxe de JSON é derivada da sintaxe que a linguagem *JavaScript* usa para seus literais, vetores e tabelas associativas.

   Uma mensagem JSON descreve um *objeto*, que pode ser um número, uma string, `true`, `false`, `null`, um vetor ou uma tabela. Um vetor JSON é uma sequência de *objetos* separados por vírgula entre colchetes. Uma tabela JSON é uma sequência de pares de uma string e um *objeto* JSON, entre chaves. A string e o objeto de um par são separados por um caractere dois pontos, e cada par é separado do próximo por uma vírgula. Note que a definição é recursiva; os elementos de um vetor podem ser outros vetores, ou tabelas, e os elementos de uma tabela também podem ser vetores, ou outras tabelas.

   Um exemplo de mensagem JSON está abaixo:

```
{
  "nome" : "João",
  "idade" : 32,
  "empregado" : true,
  "filhos" : [
    { "nome" : "Joãozinho", "idade": 5 },
    { "nome" : "Mariazinha", "idade": 3 }
  ]
}
```

   (a) (2 pontos) Dê uma gramática **LL(1)** para mensagens JSON. Assuma que os terminais são `numero`, `string`, `true`, `false`, `null`, `,`, `:`, `{`, `}`, `[` e `]`.

   (b) (1 ponto) Dê uma derivação mais à esquerda para a mensagem acima, usando a sua gramática.

2. (3 pontos) Construa o autômato LR(0) para a gramática a seguir, e diga se ela é LR(0) e/ou SLR, justificando sua resposta:

```
S -> E
E -> ( L )
E -> a
L -> L E
L ->
```

3. As interfaces e classes a seguir modelam o fragmento da árvore sintática abstrata e o gerador de código de um compilador de uma linguagem orientada a objetos simples para Java:

```
interface Env<T> {
  T busca(String nome);
  void associa(String nome, T val);
  Env<T> extende();
}

class RepeatUntil implements Cmd {
  List<Cmd> corpo;
  Exp condParada;

  String codigo(Env<String> env) {
    ...
  }
}
```

```
interface Exp {
  String codigo(Env<String> env);
}

interface Cmd {
  String codigo(Env<String> env);
}

class ChamadaMetodo implements Exp {
  Exp objeto;
  String metodo;
  List<Exp> args;

  String codigo(Env<String> env) {
    ...
  }
}
```

O método `codigo` retorna o código Java gerado a partir de uma expressão ou comando na linguagem fonte. O ambiente `env` mapeia nomes de variáveis no código Java que acessa a variável, e o método `extende` cria um novo ambiente derivado do atual.

(a) (2 pontos) Assuma que o campo `objeto` de `ChamadaMetodo` é uma expressão cujo código Java, quando executado, dá um objeto Java que possuirá o método desejado, e que a quantidade e compatiblidade de tipos dos argumentos já foi verificada pela análise semântica. Escreva o método `codigo` de `ChamadaMetodo`, que retorna o código Java da chamada.

(b) (2 pontos) A classe `RepeatUntil` representa um laço **repeat/until**, executa o corpo do laço até que a condição de parada seja verdadeira (o corpo sempre executa pelo menos uma vez). A geração do código tanto do corpo quanto a condição de parada deve se dar em um novo ambiente derivado do ambiente de geração de código do laço. A tradução de um laço **repeat/until** para Java pode usar um laço `do { ... } while(...);`. Assuma que o código gerado pela condição de parada dará um valor booleano quando executado, e que o código gerado pelos comandos do corpo do laço inclui `;` quando necessário. Escreva o método `codigo` de `RepeatUntil`, que retorna o código Java do laço **repeat/until**.

BOA SORTE!