Segunda Prova de MAB 471 2012.2 — Compiladores I

Fabio Mascarenhas

11 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	Total
Pontos:	4	6	10
Nota:			

1. A gramática abaixo dá o fragmento da sintaxe de uma linguagem orientada a objetos simples, estilo Java:

```
CLASSE  -> class TIPO extends TIPO { MEMBROS }
MEMBROS -> MEMBROS CAMPO
MEMBROS -> MEMBROS METODO
MEMBROS ->
CAMPO   -> TIPO id ;
METODO  -> TIPO id ( ) { }
TIPO    -> id
```

   (a) (2 pontos) Dê a sequência de ações *shift/reduce/accept* que um analisador bottom-up precisa fazer para analisar a sequência de tokens **class id extends id { id id ; id id ( ) { } }**.

   (b) (2 pontos) O autômato LR(0) dessa gramática possui 18 estados. O **núcleo** do estado 7 tem os seguintes items:

```
CLASSE  -> class TIPO extends TIPO { MEMBROS . }
MEMBROS -> MEMBROS . CAMPO
MEMBROS -> MEMBROS . METODO
```

   Desenhe o nó do autômato que corresponde a esse estado, acrescentando os items que formam o seu fecho. Faça as transições que saem desse estado, desenhando os nós para os quais elas vão. Escreva os items desses nós, incluindo items de fecho. Não é preciso desenhar as transições que saem desses outros nós.

2. As classes abaixo modelam o sistema de tipos de uma linguagem orientada a objetos simples, que não tem tipos primitivos, membros estáticos, interfaces ou herança múltipla:

```
class Classe {
  Classe superClasse;
  Map<String,Metodo> metodos;
  Map<String,Classe> campos;

  boolean subClasseDe(Classe sup) {
    ...
  }
}

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

interface Exp {
  Classe tipoExp(Env<Classe> tenv);
}
```

```
class Metodo {
  Classe tipoRet;
  List<String> nomeParams;
  List<Classe> tipoParams;
  List<Exp> corpo;

  void checaTipo(Classe tipoThis) {
   ...
  }
}

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

  Classe tipoExp(Env<Classe> tenv) {
    ...
  }
}
```

O método `subClasseDe` diz se uma classe é subclasse da outra, direta ou indiretamente. Uma classe sempre é subclasse dela mesma. Já as classes Java abaixo modelam o fragmento da AST da linguagem responsável pelas chamadas de métodos:

Para verificar se um método está corretamente tipado, o sistema de tipos deve criar ambiente para tipar o método, associar o nome `this` à classe do método, cada parâmetro a seu tipo, e tipar cada expressão do corpo nesse ambiente. A última expressão do corpo deve ter um tipo que é subclasse do tipo de retorno do método.

Para tipar uma chamada de método, o sistema de tipos primeiro tipa o objeto da chamada, para buscar os dados do método sendo chamado. Depois verifica se o número de argumentos bate com o número de parâmetros do método, e se cada argumento é subclasse do tipo de cada parâmetro. O tipo da chamada então é o tipo de retorno do método.

Um erro de tipagem pode ser sinalizado com uma `RuntimeException` simples, contendo a mensagem *"erro de tipagem"*.

(a) (2 pontos) Implemente o método `subClasseDe`.

(b) (2 pontos) Implemente o método `tipoExp` de `ChamadaMetodo`.

(c) (2 pontos) Implemente o método `checaTipo` de `Metodo`.

BOA SORTE!