Segunda Prova de MAB 471 — Compiladores I

Fabio Mascarenhas

25 de Junho de 2012

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:	4	2	4	10
Nota:				

Para as questões seguintes considere os trechos das seguintes classes Java, que representam um fragmento da AST de programas TINY:

```
class Atrib implements Comando {
  String lval;
  Expressao rval;
  ...
}

class Num implements Expressao {
  int val;
  ...
}

class Mul implements Expressao {
  Expressao esq;
  Expressao dir;
  ...
}

class Var implements Expressao {
  String nome;
  ...
}

class Sub implements Expressao {
  Expressao esq;
  Expressao dir;
  ...
}

class ChamadaFuncao implements Expressao {
  String nomeFunc;
  Expressao[] args;
  ...
}
```

1. Considere a gramática a seguir, para um fragmento de TINY:

   ```
   LISTA -> LISTA ; CMD | CMD
   CMD -> id := EXP
   EXP -> id | id ( ) | num
   ```

(a) (2 pontos) Faça o autômato LR(0) dessa gramática. Pode abreviar os não-terminais LISTA, CMD e EXP como L, C e E. LISTA é o não-terminal inicial da gramática.

(b) (1 ponto) Essa gramática é LR(0)? E SLR(1)? Justifique suas respostas.

(c) (1 ponto) Dê a sequência de ações (shift, reduce, accept) do analisador LR para o termo `id := id ()`.

2. (2 pontos) Uma notação popular para o tipo de uma função que recebe n argumentos de tipos $\tau_1, \ldots, \tau_n$ e retorna um valor com tipo τ_r é $\tau_1 \times \ldots \times \tau_n \to \tau_r$. Assim, a tipagem de uma chamada de função pode ser dada pela seguinte regra, onde Γ é um mapa de nomes de variáveis para tipos, Φ é um mapa de nomes de funções para tipos, e $\Gamma, \Phi \vdash e : \tau$ significa que a expressão e tem o tipo τ dados os mapas Γ e Φ:

$$\frac{\Phi(f) = \tau_1 \times \ldots \times \tau_n \to \tau_r \quad \Gamma, \Phi \vdash e_1 : \tau_1 \quad \ldots \quad \Gamma, \Phi \vdash e_n : \tau_n}{\Gamma, \Phi \vdash f(e_1, \ldots, e_n) : \tau_r}$$

Dada a classe `TipoFuncao` abaixo, implemente o método de verificação de tipos da classe `ChamadaFuncao`, `Tipo tipo(Map<String,Tipo> vars, Map<String,TipoFuncao> funcs)`, seguindo a regra acima. Use `equals` para testar igualdade de tipos. Não se esqueça de verificar se o número de argumentos é compatível com o número de parâmetros (verificação de aridade).

```
class TipoFuncao {
  Tipo[] tipoArgs;
  Tipo tipoRet;
}
```

3. A JVM é uma máquina de pilha, ou seja, as instruções operam não sobre registradores mas sobre valores em uma pilha. A instrução **ldc** n empilha um número literal n, a instrução **iload** n empilha o valor da n-ésima variável local (que deve ser inteira), a instrução **istore** n desempilha o valor que está no topo da pilha e o guarda na n-ésima variável local, as instruções **iadd**, **isub**, **idiv** e **imul** desempilham dois valores e empilham o resultado da operação correspondente (soma, subtração, multiplicação e divisão, respectivamente), e a instrução **invokestatic** f desempilha quantos parâmetros a função f tiver e chama f com esses argumentos, empilhando o valor de retorno.

(a) (1 ponto) Escreva o código JVM para o seguinte comando TINY, assumindo que a, b, c e d são as variáveis locais de números 0, 1, 2 e 3, e sqr é uma função:

```
d := sqr(b) - 4 * a * c
```

(b) (3 pontos) Escreva o método `void geraCodigo(Saida s)` para as classes `Atrib`, `Var`, `Num`, `Sub`, `Mul` e `ChamadaFuncao`. Esse método deve gerar código JVM. Assuma que o objeto do tipo `Saida` tem métodos `void ldc(int val)`, `void iload(int var)`, `void istore(int var)`, `void isub()`, `void imul()` e `void invokestatic(String f)`, que emitem as instruções correspondentes, assim como um atributo `vars` do tipo `Map<String,Integer>` que representa a tabela de símbolos. Lembre que a análise semântica já verificou compatibilidade de tipos e de aridade de funções.