Resolução do Exame de Época Normal de Linguagens de Programação 2

18 de Janeiro de 2006

I. 1. Para o efeito da construção de uma hierarquia de classes Java para a representação abstracta de expressões da linguagem AKA considera-se a existência de um interface `IASTNode`. A seguir lista-se uma série de protótipos de constructores que resumem a lista de classes que implementam o interface `IASTNode` e que representam cada um das expressões da linguagem:

```
// Id
ASTId(String id)
// Num
ASTNum(int n)
// E {+} E
ASTAdd(IASTNode left, IASTNode right)
// E {-} E
ASTSub(IASTNode left, IASTNode right)
// E {*} E
ASTMul(IASTNode left, IASTNode right)
// E {/} E
ASTDiv(IASTNode left, IASTNode right)
// {fun} Id {->} E
ASTFun(String par, IASTNode body)
// E(E)
ASTCall(IASTNode f, IASTNode arg)
// {try} E {catch} E
ASTTryBlock(IASTNode body, IASTNode ex)
```

2. A implementação das funções de avaliação pressupõem a existência de um interface `IVal` e de classes que o implementam representando cada um dos tipos de valores: `IntVal`, `Closure`, `ExceptionThrown`.

   Como abreviação admite-se que existem, na classe `IntVal`, os métodos estáticos `add`, `sub`, `mul` e `div` que aceitam dois objectos `IntVal` e retornam um outro object `IntVal` cujo valor é o resultado óbvio das operações.

Sendo assim, a implementação da função `evaluate` em cada uma das classes introduzidas na alínea anterior é:

```
// ASTId(String id)
IVal evaluate(Environment a) {
  return a.Find(id);
}
// ASTNum(int n)
IVal evaluate(Environment a) {
  return new IntVal(n);
}
// ASTAdd(IASTNode left, IASTNode right)
IVal evaluate(Environment a) {
  IVal leftV = left.evaluate(a);
  IVal rightV = right.evaluate(a);

  if( leftV instanceof ExceptionThrown ||
      rightV instanceof ExceptionThrown)
    return new ExceptionThrown();
  else
    return IntVal.add(leftV,rightV);
}
// ASTSub(IASTNode left, IASTNode right)
IVal evaluate(Environment a) {
  IVal leftV = left.evaluate(a);
  IVal rightV = right.evaluate(a);

  if( leftV instanceof ExceptionThrown ||
      rightV instanceof ExceptionThrown)
    return new ExceptionThrown();
  else
    return IntVal.sub(leftV,rightV);
}
// ASTMul(IASTNode left, IASTNode right)
IVal evaluate(Environment a) {
  IVal leftV = left.evaluate(a);
  IVal rightV = right.evaluate(a);

  if( leftV instanceof ExceptionThrown ||
      rightV instanceof ExceptionThrown)
    return new ExceptionThrown();
  else
    return IntVal.mul(leftV,rightV);
}
// ASTDiv(IASTNode left, IASTNode right)
IVal evaluate(Environment a) {
```

```
  IVal leftV = left.evaluate(a);
  IVal rightV = right.evaluate(a);

  if( leftV instanceof ExceptionThrown  ||
      rightV instanceof ExceptionThrown ||
      ((IntVal) rightV).value() == 0)
    return new ExceptionThrown();
  else
    return IntVal.div(leftV,rightV);
}
// ASTFun(String par, IASTNode body)
IVal evaluate(Environment a) {
  return new Closure(par,body,a);
}
// ASTCall(IASTNode f, IASTNode arg)
IVal evaluate(Environment a) {
  IVal leftV = left.evaluate(a);
  IVal rightV = right.evaluate(a);

  if( leftV instanceof ExceptionThrown  ||
      rightV instanceof ExceptionThrown)
    return new ExceptionThrown();
  else {
    Closure c = (Closure)leftV;
    Environment e = c.getEnv().BeginScope();
    e = e.Assoc(c.getPar(),rightV);
    return c.getBody().evaluate(e);
  }
}
// ASTTryBlock(IASTNode body, IASTNode exc)
IVal evaluate(Environment a) {
  IVal leftV = left.evaluate(a);
  if( leftV instanceof ExceptionThrown)
    return right.evaluate(a);
  else
    return leftV;
}
```

Existe uma outra alternativa onde a excepção não é propagada através do valor de retorno mas sim utilizando o mecanismo nativo do Java. Considere-se a classe `DivisionException` como extendendo a classe `java.lang.Exception` no seguinte código:

```
// ASTId(String id)
IVal evaluate(Environment a) throws DivisionException {
  return a.Find(id);
}
```

```
// ASTNum(int n)
IVal evaluate(Environment a) throws DivisionException {
  return new IntVal(n);
}
// ASTAdd(IASTNode left, IASTNode right)
IVal evaluate(Environment a) throws DivisionException {
  IVal leftV = left.evaluate(a);
  IVal rightV = right.evaluate(a);
  return IntVal.add(leftV,rightV);
}
// ASTSub(IASTNode left, IASTNode right)
IVal evaluate(Environment a) throws DivisionException {
  IVal leftV = left.evaluate(a);
  IVal rightV = right.evaluate(a);
  return IntVal.sub(leftV,rightV);
}
// ASTMul(IASTNode left, IASTNode right)
IVal evaluate(Environment a) throws DivisionException {
  IVal leftV = left.evaluate(a);
  IVal rightV = right.evaluate(a);
  return IntVal.mul(leftV,rightV);
}
// ASTDiv(IASTNode left, IASTNode right)
IVal evaluate(Environment a) throws DivisionException {
  IVal leftV = left.evaluate(a);
  IVal rightV = right.evaluate(a);

  if (((IntVal) rightV).value() == 0)
    throw new DivisionException();
  else
    return IntVal.div(leftV,rightV);
}
// ASTFun(String par, IASTNode body)
IVal evaluate(Environment a) throws DivisionException {
  return new Closure(par,body,a);
}
// ASTCall(IASTNode f, IASTNode arg)
IVal evaluate(Environment a) throws DivisionException {
  IVal leftV = left.evaluate(a);
  IVal rightV = right.evaluate(a);
  Closure c = (Closure)leftV;
  Environment e = c.getEnv().BeginScope();
  e = e.Assoc(c.getPar(),rightV);
  return c.getBody().evaluate(e);
}
// ASTTryBlock(IASTNode body, IASTNode exc)
```

```
IVal evaluate(Environment a) throws DivisionException {
  try {
    return left.evaluate(a);
  } catch(DivisionException e) {
    return right.evaluate(a);
  }
}
```

II Para definir um sistema de tipos para a linguagem AKAT considera-se, para além dos tipos introduzidos pelo enunciado, o tipo funcional $T \to S$. As regras de inferência que definem o sistema de tipos são:

Para as constantes:

$$E \vdash 0 : ZERO \qquad E \vdash n : POS \;\; n > 0 \qquad E \vdash n : NEG \;\; n < 0$$

Para as variáveis, funções e chamadas de função:

$$E, x : T, E' \vdash x : T \qquad \frac{E, x : T \vdash e_1 : S}{E \vdash \texttt{fun}\;\; x : T \to e_1 : T \to S} \qquad \frac{E \vdash e_1 : T \to S \quad E \vdash e_2 : T}{E \vdash e_1(e_2) : S}$$

Adição:

$$\frac{E \vdash e_1 : ZERO \quad E \vdash e_2 : ZERO}{E \vdash e_1 + e_2 : ZERO} \qquad \frac{E \vdash e_1 : POS \quad E \vdash e_2 : POS}{E \vdash e_1 + e_2 : POS}$$

$$\frac{E \vdash e_1 : POS \quad E \vdash e_2 : ZERO}{E \vdash e_1 + e_2 : POS} \qquad \frac{E \vdash e_1 : ZERO \quad E \vdash e_2 : POS}{E \vdash e_1 + e_2 : POS}$$

$$\frac{E \vdash e_1 : NEG \quad E \vdash e_2 : NEG}{E \vdash e_1 + e_2 : NEG} \qquad \frac{E \vdash e_1 : ZERO \quad E \vdash e_2 : NEG}{E \vdash e_1 + e_2 : NEG} \qquad \frac{E \vdash e_1 : NEG \quad E \vdash e_2 : ZERO}{E \vdash e_1 + e_2 : NEG}$$

$$\frac{E \vdash e_1 : ANY \quad E \vdash e_2 : T}{E \vdash e_1 + e_2 : ANY} \qquad \frac{E \vdash e_1 : T \quad E \vdash e_2 : ANY}{E \vdash e_1 + e_2 : ANY}$$

$$\frac{E \vdash e_1 : POS \quad E \vdash e_2 : NEG}{E \vdash e_1 + e_2 : ANY} \qquad \frac{E \vdash e_1 : NEG \quad E \vdash e_2 : POS}{E \vdash e_1 + e_2 : ANY}$$

Subtracção:

$$\frac{E \vdash e_1 : POS \quad E \vdash e_2 : NEG}{E \vdash e_1 - e_2 : POS} \qquad \frac{E \vdash e_1 : POS \quad E \vdash e_2 : ZERO}{E \vdash e_1 - e_2 : POS}$$

$$\frac{E \vdash e_1 : NEG \quad E \vdash e_2 : POS}{E \vdash e_1 - e_2 : NEG} \qquad \frac{E \vdash e_1 : NEG \quad E \vdash e_2 : ZERO}{E \vdash e_1 - e_2 : NEG}$$

$$\frac{E \vdash e_1 : POS \quad E \vdash e_2 : POS}{E \vdash e_1 - e_2 : ANY} \qquad \frac{E \vdash e_1 : NEG \quad E \vdash e_2 : NEG}{E \vdash e_1 - e_2 : ANY}$$

Multiplicação:

$$\frac{E \vdash e_1 : POS \quad E \vdash e_2 : POS}{E \vdash e_1 * e_2 : POS} \qquad \frac{E \vdash e_1 : NEG \quad E \vdash e_2 : NEG}{E \vdash e_1 * e_2 : POS}$$

$$\frac{E \vdash e_1 : NEG \quad E \vdash e_2 : POS}{E \vdash e_1 * e_2 : NEG} \qquad \frac{E \vdash e_1 : POS \quad E \vdash e_2 : NEG}{E \vdash e_1 * e_2 : NEG}$$

$$\frac{E \vdash e_1 : T \quad E \vdash e_2 : ANY}{E \vdash e_1 * e_2 : ANY} \; T \neq ZERO \qquad \frac{E \vdash e_1 : ANY \quad E \vdash e_2 : T}{E \vdash e_1 * e_2 : ANY} \; T \neq ZERO$$

$$\frac{E \vdash e_1 : ZERO}{E \vdash e_1 * e_2 : ZERO} \qquad \frac{E \vdash e_2 : ZERO}{E \vdash e_1 * e_2 : ZERO}$$

Divisão:

$$\frac{E \vdash e_1 : POS \quad E \vdash e_2 : POS}{E \vdash e_1 / e_2 : POS} \qquad \frac{E \vdash e_1 : NEG \quad E \vdash e_2 : NEG}{E \vdash e_1 / e_2 : POS}$$

$$\frac{E \vdash e_1 : NEG \quad E \vdash e_2 : POS}{E \vdash e_1 / e_2 : NEG} \qquad \frac{E \vdash e_1 : POS \quad E \vdash e_2 : NEG}{E \vdash e_1 / e_2 : NEG}$$

$$\frac{E \vdash e_1 : ANY \quad E \vdash e_2 : NEG}{E \vdash e_1 / e_2 : ANY} \qquad \frac{E \vdash e_1 : ANY \quad E \vdash e_2 : POS}{E \vdash e_1 / e_2 : ANY}$$

$$\frac{E \vdash e_1 : ZERO \quad E \vdash e_2 : POS}{E \vdash e_1 / e_2 : ZERO} \qquad \frac{E \vdash e_1 : ZERO \quad E \vdash e_2 : NEG}{E \vdash e_1 / e_2 : ZERO}$$

Try-Catch:

$$\frac{E \vdash e_1 : T}{E \vdash \texttt{try } e_1 \texttt{ catch } e_2 : T}$$

(note-se que a divisão por zero não é bem tipificada pelo que só interessa o tipo de e_1.)

2. a) $NEG \rightarrow POS$
   b) $NEG \rightarrow NEG$

3. Sim, impede erros de execução relativos a todas as outras construções. O sistema de tipos garante que as chamadas de funções ($e_1(e_2)$) só são feitas se a expressão e_1 de facto avaliar para uma função e o tipo do argumento corresponder ao tipo do parâmetro, e também que uma operação aritmética bem tipificada tem sempre dois operandos números (portanto não são funções). Também se garante que todos os identificadores referidos nas expressões bem tipificadas são conhecidos.

4. $2/(2-1)$

III. 1. Considera-se os preâmbulo e epílogo habituais.

```
.locals init (int32, int32)
// x = 1
ldc.i4 1
stloc 0
// z = 1
ldc.i4 1
stloc 1
// z + 3 (A)
ldloc 1
ldc.i4 3
add
// x = 2
ldc.i4 2
stloc 1
// x*3 (B)
ldloc 1
ldc.i4 3
mul
// y = A + B
add
stloc 1
// x+y
ldloc 0
ldloc 1
add
// print(...)
call void class [mscorlib]System.Console::WriteLine(int32)
```

2. A compilação de linguagens com declarações embricadas, funções como cidadãos de primeira classe (closures) e escopo estático como é o caso da linguagem CORE requer a utilização da chamada pilha-esparguete, composta por objectos alocados em memória dinâmica. A utilização de uma pilha "normal" não permite manter o contexto da avaliação de expressões "função" fora do seu âmbito de declaração. Quando se desempilha um

registo de activação de uma função todas as variáveis locais, que podem ser capturadas num qualquer valor de tipo funcional declarado no âmbito dessa função, desaparecem, podendo dar origem a erros de execução por falta de identificador. Com uma pilha esparguete, os contextos das declarações de função são mantidos em memória (associados aos valores-função) e são depois utilizados para executar o corpo das funções.