Departamento de Ciência da Computação – IME-USP

MAC 316 – Conceitos Fund. de Linguagens de Programação

Prof: Ana C. V. de Melo
Terceiro Trabalho – Implmentação de uma linguagem de expressões
Prazo de Entrega: até dia 19/06/2012
Número de integrantes da equipe: 5 (máximo)

SML - programação funcional

Considere uma linguagem funcional simplificada (LFSimp) definida informalmente como segue (sintaxe concreta):

```
SimbolosTerminais : fun, if, (, ), +, -, *, And, Or, ==, [, ]

Programa    ::= Aplicacao
Aplicacao   ::= (DecFuncao, ListaExp)
ListaExp    ::= [Expressao, ...]        {0 ou mais expressôes}
DecFuncao   ::= ("fun", ListaParam,  Expressao)
ListaParam  ::= [Id, ...]               {0 ou mais identificadores}
Id          ::= "a" | ... | "z"         {apenas 1  letra}
Expressao   ::= Valor | ExpBinaria | IfThenElse
Valor       ::= ValorInteiro | ValorBooleano
ExpBinaria  ::= ExpAritm | ExpBool
ExpAritm    ::= ValorInteiro
              | (Expressao, "+",   Expressao)
              | (Expressao, "-",   Expressao)
ExpBool     ::= ValorBooleano
              | (Expressao, "And", Expressao)
              | (Expressao, "Or",  Expressao)
              | (Expressao, "==",  Expressao)
IfThenElse  ::= ("if", ExpBool, Expressao, Expressao)

ValorInteiro ::= ... -1 | 0 | 1 ...  {valores inteiros de SML}
ValorBooleano::= true | false        {já definidos em SML}
```

Esta linguagem não possui declaração de tipos e cada um dos operadores, tanto aritmético quanto booleano, é aplicado a expressões. Neste trabalho não realizaremos a verificação de tipos sobre as funções criadas, nem sobre os operadores predefinidos. Assim, consideramos que os tipos associados a aplicação das funções estão sempre corretos.

A avaliação de um programa é a aplicação da definição da função aos termos (expressões) correspondentes: o primeiro termo é associado ao primeiro parâmetro e assim sucessivamente. A avaliação das expressões binárias é a usual: o operador só poderá ser aplicado depois de avaliado cada um dos operandos. Note que os valores considerados foram apenas informalmente definidos. Aqui, consideramos que os tipos inteiro e booleano da linguagem são os mesmos definidos em SML.

Uma outra simplificação usada aqui é a ausência de nomes para as funções (assim como temos no λ–cálculo). Um programa é então um par contendo a definição de uma função junto com uma lista de termos para a aplicação. Dessa forma, um termo de aplicação é uma expressão e pode, portanto, ser tanto um valor quanto uma expressão binária ou um IfThenElse. Para simplificar a avaliação das funções, cada elemento da expressão já tem um construtor associado (evita fazer o parser). Os construtores são definidos por:

```
datatype Expression =
      VInt of int
    | VBool of bool
    | Id of string
    | AExp of Expression * string * Expression
    | BExp of Expression * string * Expression
    | If of string * Expression * Expression * Expression;
```

Note que esta é apenas uma sugestão de definição de tipo, o que interessa é o uso dos construtores para cada tipo de expressão. Vocês podem definir os tipos necessários de forma diferente, mas conservem os construtores como definidos aqui. Usando os construtores acima, poderíamos ter por exemplo a execução dos seguintes programas:

- (("fun", [Id "x", Id "y"], AExp (Id "x", "+", Id "y")), [VInt 2, VInt 3]);
- (("fun", [Id "x", Id "y"], AExp (AExp (Id "x", "+", Id "y"),"+", VInt 10)), [VInt 2, VInt 3]);
- (("fun", [Id "x", Id "y"], BExp (Id "x", "And", Id "y")), [VBool false, VBool true]);
- (("fun", [Id "x", Id "y"], AExp (Id "x", "+", Id "y")), [AExp (VInt 2, "+", VInt 5), VInt 3]);

Você deve implementar, em SML, um sistema de redução (avaliação) para a linguagem sugerida acima (veja as reduções para λ-cálculus no Capítulo 8). Nesse sistema, devem ser implementadas ambas as estratégias de redução estudadas: *sob demanda* e *a priori*. Considere que todas as expressões são bem-formadas, e note que elas podem ser facilmente manipuladas em SML. Por isso, não há necessidade de desenvolver um *parser*, basta desenvolver as estratégias de avaliação sobre as expressões (tuplas). Crie quantas funções julgar necessário, mas o seu sistema deverá ter, no mínimo, as funções:

```
fun  sobdemanda ...      {avaliação sob demanda}
fun  apriori ...         {avaliação a priori}
```

O resultado da avaliação de um programa deve ser mostrado passo a passo: o primeiro passo da avaliação, segundo, ...

Bônus

1. Implementação de um Parser para a linguagem: 2 pontos extra
2. Implementação de expressões como uma aplicação: 2 pontos extra. Neste caso Vocês devem considerar a linguagem:

   ```
   Expressao   ::= Valor | ExpBinaria | IfThenElse | Aplicacao
   Valor       ::= ValorInteiro | ValorBooleano | ValorFuncao
   ValorFuncao ::= Aplicacao       {valor resultante da
                                    aplicação de uma função}
   ```

   Note que isso permite definição local de funções tanto nas expressões quanto nos termos de aplicação.

Material de Apoio A maioria das ferramentas para SML são de domínio público, a mais recomendada por ter bastante material de apoio é a SML/NewJersey, a qual foi desenvolvida por um consórcio de universidades e empresas: http://www.smlnj.org/

Existem alguns tutoriais sobre SMLa partir do site do smlnj, aqui estão alguns recomendados:

- Tutorial com alguns conceitos associados:
  `http://homepages.inf.ed.ac.uk/stg/NOTES/`
- Tutorial com dicas para a implementação de linguagens:
  `http://www.cs.cmu.edu/~rwh/smlbook/book.pdf`
- Tutorial + uso do smljn (bibliotecas e ferramentas):
  `http://www.cs.cornell.edu/riccardo/prog-smlnj/notes-011001.pdf`
- Um guia de sobrevivência smlnj:
  `http://www.cs.cmu.edu/afs/cs/local/sml/common/smlguide/smlnj.htm`

OBS: é proibido usar variáveis de referência neste EP.