Lista de Linguagens de Programação – 24

Nome: _____ Matrícula: _____

Os próximos dois exercícios dizem respeito à linguagem `Three`, cuja gramática está transcrita abaixo. Existe um interpretador para esta linguagem, em Prolog, na página do curso. Este intepretador foi escrito pelo autor do livro adotado no curso.

$$
\begin{array}{rcl}
< exp > & = & \text{fn } < var > \rightarrow < exp > \\
 & | & < addexp > \\
< addexp > & = & < addexp > \ + \ < mulexpr > \\
 & | & < mulexpr > \\
< mulexp > & = & < mulexpr > \ * \ < funexp > \\
 & | & < funexp > \\
< funexp > & = & < funexp > < rootexp > \\
 & | & < rootexp > \\
< rootexp > & = & \text{let val } < var > = < exp > \text{ in } < exp > \text{ end} \\
 & | & (< exp >) \\
 & | & < var > \\
 & | & < const >
\end{array}
$$

1. Considere as duas expressões: `let val x = e1 in e2` e `(fn x => e2) e1`. Estas expressões são equivalentes.

   (a) A partir do interpretador de `Three` que implemente associação dinâmica (*dynamic binding*), prove que as expressões são de fato equivalentes. Para isto, mostre que, dada a semântica natural de `Three`, as condições para avaliar a expressão $\langle \text{apply}(\text{fn}(x, e_2), e_1), C\rangle \rightarrow v$ são equivalentes às condições necessárias para avaliar $\langle \text{let}(x, e1, e2), C\rangle \rightarrow v$.

   (b) Mostre a mesma coisa, mas desta vez usando o interpretador que implementa associação estática.

2. O objetivo deste exercício é definir a linguagem `Four`, que é uma extensão da linguagem `Three`. Um exemplo de programa na linguagem `Four` é dado abaixo:

```
let
  val fact = fn x => if x < 2 then x else x * x fact(x-1)
in
  fact 5
end
```

Como você pode ver, a linguagem `Four` estende a linguagem `Three` com três novos constructos: o operador $<$ para comparações, o operador $-$ para subtrações, e a expressão condicional `if-then-else`. O programa de exemplo acima define a função fatorial recursiva, e a utiliza para calcular o fatorial de 5.

(a) Defina a sintaxe da linguagem `Four` a partir da linguagem `Three`. Isto é, estenda `Three` com sintaxe para incorporar os três novos constructos. Cuide para que a sua gramática não seja ambígua.

(b) Defina três novos tipos de nodos AST para incorporar a nova sintaxe. Estenda a implementação de nosso interpretador Prolog para lidar com estes novos nodos. Você vai ter de usar a implementação que usa escopo dinâmico, pois a implementação que usa escopo estático não consegue lidar com definições recursivas. Isto quer dizer que aquele programa de exemplo não é ML padrão. Verifique que a sua implementação avalie programas corretamente. Por exemplo, o fatorial de 5 é 120.

(c) Dê uma semântica natural para a linguagem `Four`.

3. A semântica com escopo estático da linguagem `Three` não suporta recursão porque o escopo de uma definição de variável não inclui o corpo da função. ML funciona do mesmo jeito. O escopo de uma definição de `f`, que é produzido por `fun f...` inclui o corpo da função que está sendo definida, mas o mesmo não acontece com o escopo da definição produzido por `val f = fn...` Assim, em ML apenas funções declaradas com `fun` podem ser recursivas.

   (a) Estenda a sintaxe da linguagem `Four` vista no exercício anterior para que esta permita definições simples de funções, como por exemplo:

   ```
   let fun f x = x + 1 in f 1 end
   ```

   (b) Defina os novos tipos de nodos AST que você irá precisar para incorporar a sua sintaxe estendida. Estenda o interpretador Prolog da linguagem `Four` para lidar com eles. Use uma implementação de escopo estático para ambos os tipos de funções: `fun f...` e `val f = fn...` Funções definidas com `fun` precisam poder ser recursivas. Dica: dê uma olhada no termo `fval`, no interpretador exemplo (disponível na página do curso). Em particular, dê uma olhada em `val3`. Use um termo diferente para representar funções criadas com `fun` – um termo que armazene o nome da função – e uma cláusula diferente para `apply`. Teste sua

implementação em uma versão recursiva da função fatorial, que você deve definir usando `fun`.

(c) Dê uma semântica natural para a sua linguagem estendida.

4. Mudando um pouco de assunto, escreva um predicado `gcd(N1, N2, Q)` em Prolog que seja verdadeiro se `Q` for o maior divisor comum de `N1` e `N2`. Por exemplo:

```
?- gcd(36, 63, G).
G = 9
```