Gramáticas de atributos

J.F. Cardona McCormick

Universidad EAFIT

20 de Febrero de 2009

Agenda

Introducción

- Mostraremos las características de la programación funcional.

Introducción

- Mostraremos las características de la programación funcional.
- Mostraremos un tipo que problema que se resuelven más fácilmente utilizando un lenguaje de programación funcional puro y "perezoso".

Introducción

- Mostraremos las características de la programación funcional.
- Mostraremos un tipo que problema que se resuelven más fácilmente utilizando un lenguaje de programación funcional puro y "perezoso".
- Utilizaremos las gramáticas de atributos basadas en lenguajes funcionales para facilitar la construcción de este tipo de programas.

Programación funcional

- Los lenguajes funcionales son una clase de lenguajes basados en el *lambda-cálculo*, un muy simple pero poderoso modelo de computación.

Programación funcional

- Los lenguajes funcionales son una clase de lenguajes basados en el *lambda-cálculo*, un muy simple pero poderoso modelo de computación.
- Ventajas:

Programación funcional

- Los lenguajes funcionales son una clase de lenguajes basados en el *lambda-cálculo*, un muy simple pero poderoso modelo de computación.
- Ventajas:
  - Producción de software más rápida.

Programación funcional

- Los lenguajes funcionales son una clase de lenguajes basados en el *lambda-cálculo*, un muy simple pero poderoso modelo de computación.
- Ventajas:
  - Producción de software más rápida.
  - Programas más cortos.

Programación funcional

- Los lenguajes funcionales son una clase de lenguajes basados en el *lambda-cálculo*, un muy simple pero poderoso modelo de computación.
- Ventajas:
  - Producción de software más rápida.
  - Programas más cortos.
  - Código más leíble y verificable que el código utilizado en los llamados lenguajes imperativos.

Programación funcional

- Los lenguajes funcionales son una clase de lenguajes basados en el *lambda-cálculo*, un muy simple pero poderoso modelo de computación.
- Ventajas:
  - Producción de software más rápida.
  - Programas más cortos.
  - Código más leíble y verificable que el código utilizado en los llamados lenguajes imperativos.
- Usos:

Programación funcional

- Los lenguajes funcionales son una clase de lenguajes basados en el *lambda-cálculo*, un muy simple pero poderoso modelo de computación.
- Ventajas:
  - Producción de software más rápida.
  - Programas más cortos.
  - Código más leíble y verificable que el código utilizado en los llamados lenguajes imperativos.
- Usos:
  - Inteligencia artificial.

Programación funcional

- Los lenguajes funcionales son una clase de lenguajes basados en el *lambda-cálculo*, un muy simple pero poderoso modelo de computación.
- Ventajas:
  - Producción de software más rápida.
  - Programas más cortos.
  - Código más leíble y verificable que el código utilizado en los llamados lenguajes imperativos.
- Usos:
  - Inteligencia artificial.
  - Cálculo simbólico.

Programación funcional

- Los lenguajes funcionales son una clase de lenguajes basados en el *lambda-cálculo*, un muy simple pero poderoso modelo de computación.
- Ventajas:
  - Producción de software más rápida.
  - Programas más cortos.
  - Código más leíble y verificable que el código utilizado en los llamados lenguajes imperativos.
- Usos:
  - Inteligencia artificial.
  - Cálculo simbólico.
  - Pruebas de teoremas.

Programación funcional

- Los lenguajes funcionales son una clase de lenguajes basados en el *lambda-cálculo*, un muy simple pero poderoso modelo de computación.
- Ventajas:
  - Producción de software más rápida.
  - Programas más cortos.
  - Código más leíble y verificable que el código utilizado en los llamados lenguajes imperativos.
- Usos:
  - Inteligencia artificial.
  - Cálculo simbólico.
  - Pruebas de teoremas.
  - Sistemas basados en pruebas.

Programación funcional

- Los lenguajes funcionales son una clase de lenguajes basados en el *lambda-cálculo*, un muy simple pero poderoso modelo de computación.
- Ventajas:
  - Producción de software más rápida.
  - Programas más cortos.
  - Código más leíble y verificable que el código utilizado en los llamados lenguajes imperativos.
- Usos:
  - Inteligencia artificial.
  - Cálculo simbólico.
  - Pruebas de teoremas.
  - Sistemas basados en pruebas.
  - Procesamiento de lenguaje natural.

Programación funcional

- Un programa escrito en un lenguaje funcional consiste de definición de funciones y aplicaciones de funciones.

Programación funcional

- Un programa escrito en un lenguaje funcional consiste de definición de funciones y aplicaciones de funciones.
- Como en las matemáticas, una función es una entidad que mapea cada entrada con un única salida.

Programación funcional

- Un programa escrito en un lenguaje funcional consiste de definición de funciones y aplicaciones de funciones.
- Como en las matemáticas, una función es una entidad que mapea cada entrada con un única salida.
- Es un completo constrate con los lenguajes imperativos en el cual una función es simplemente una colección de instrucciones las cuales pueden modificar variables, permitiendo que la misma entrada sea mapeada a diferentes salidas sobre el curso de la computación.

Considere la función factorial, descrita formalmente por:

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n \times (n-1)! & \text{en otros casos.} \end{cases}$$

Considere la función factorial, descrita formalmente por:

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n \times (n-1)! & \text{en otros casos.} \end{cases}$$

En un lenguaje funcional, en el caso de Haskell, la definición ejecutable del factorial es normalmente escrita como:

Programación funcional

Considere la función factorial, descrita formalmente por:

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n \times (n-1)! & \text{en otros casos.} \end{cases}$$

En un lenguaje funcional, en el caso de Haskell, la definición ejecutable del factorial es normalmente escrita como:

```
fac :: Integer → Integer
fac 0 = 1
fac n = n × fac (n - 1)
```

Programación funcional

- Las funciones escritas en los lenguajes funcionales son funciones matemáticas en el verdadero sentido de la palabra.

Programación funcional

- Las funciones escritas en los lenguajes funcionales son funciones matemáticas en el verdadero sentido de la palabra.
- *No existe la noción de memoria y de modificación de la misma.*

Programación funcional

- Las funciones escritas en los lenguajes funcionales son funciones matemáticas en el verdadero sentido de la palabra.
- *No existe la noción de memoria y de modificación de la misma.*
- Por lo tanto no hay necesidad de un operador de asignación (esto es únicamente en los lenguajes considerados puros).

Programación funcional

- Las funciones escritas en los lenguajes funcionales son funciones matemáticas en el verdadero sentido de la palabra.
- *No existe la noción de memoria y de modificación de la misma.*
- Por lo tanto no hay necesidad de un operador de asignación (esto es únicamente en los lenguajes considerados puros).
- Al prevenir la modificación de las variables existentes, los lenguajes funcionales exhiben un comportamiento que es conocido como *transparencia referencial*.

Programación funcional

- Las funciones escritas en los lenguajes funcionales son funciones matemáticas en el verdadero sentido de la palabra.
- *No existe la noción de memoria y de modificación de la misma.*
- Por lo tanto no hay necesidad de un operador de asignación (esto es únicamente en los lenguajes considerados puros).
- Al prevenir la modificación de las variables existentes, los lenguajes funcionales exhiben un comportamiento que es conocido como *transparencia referencial*.
- La transparencia referencial permite que expresiones iguales puedan ser intercambiadas.

Programación funcional

- Las funciones escritas en los lenguajes funcionales son funciones matemáticas en el verdadero sentido de la palabra.
- *No existe la noción de memoria y de modificación de la misma.*
- Por lo tanto no hay necesidad de un operador de asignación (esto es únicamente en los lenguajes considerados puros).
- Al prevenir la modificación de las variables existentes, los lenguajes funcionales exhiben un comportamiento que es conocido como *transparencia referencial*.
- La transparencia referencial permite que expresiones iguales puedan ser intercambiadas.

```
let x = f a
in x + x
```

Programación funcional

- Una de las características más importantes de los lenguajes funcionales, es que tratan las funciones como datos.

Programación funcional

- Una de las características más importantes de los lenguajes funcionales, es que tratan las funciones como datos.
- Todos los lenguajes de programación funcional suministran un constructor para especificar un valor de función sin tener que declarar el nombre de la función, equivalente a las lambda abstracciones en el lambda calculo.

Programación funcional

- Una de las características más importantes de los lenguajes funcionales, es que tratan las funciones como datos.
- Todos los lenguajes de programación funcional suministran un constructor para especificar un valor de función sin tener que declarar el nombre de la función, equivalente a las lambda abstracciones en el lambda calculo.

λ `x` $\rightarrow$ `<expresión>`

Programación funcional

- Una de las características más importantes de los lenguajes funcionales, es que tratan las funciones como datos.
- Todos los lenguajes de programación funcional suministran un constructor para especificar un valor de función sin tener que declarar el nombre de la función, equivalente a las lambda abstracciones en el lambda calculo.

  λ `x` $\rightarrow$ `<expresión>`

- Un subconjunto de los lenguajes funcionales ofrece una estrategia de evaluación no estricta, también conocidad como evaluación perezosa.

Programación funcional

- Una de las características más importantes de los lenguajes funcionales, es que tratan las funciones como datos.
- Todos los lenguajes de programación funcional suministran un constructor para especificar un valor de función sin tener que declarar el nombre de la función, equivalente a las lambda abstracciones en el lambda calculo.

  λ `x` $\rightarrow$ `<expresión>`

- Un subconjunto de los lenguajes funcionales ofrece una estrategia de evaluación no estricta, también conocidad como evaluación perezosa.
- En la evaluación perezosa, el evaluador hace *solamente lo preciso*

El problema Repmin

Suponga usted que tiene que tomar un árbol que contiene valores númericos y buscar en el árbol el menor valor y reemplazar todos los nodos con dicho valor. Implemente una solución eficiente. [Bird, 1984].

Solución en Haskell del problema Repmin

```
data Tree = Tree_Leaf Int
          | Tree_Bin Tree Tree
          deriving Show


repmin :: Tree -> Tree
repmin t
  = t'
  where (t',tmin)                 = r t tmin
        r (Tree_Leaf i    ) m = (Tree_Leaf m      , i                 )
        r (Tree_Bin  lt rt) m = (Tree_Bin lt' rt', lmin `min` rmin)
                                where (lt',lmin) = r lt m
                                      (rt',rmin) = r rt m
```

Gramáticas de attributos

- Fueron diseñadas por Donald Knuth [Knuth, 1968], para mostrar como describir la semántica de un programa.
- Están basadas en durante la compilación se produce un árbol sintactico abstracto (AST).
- El árbol sintático abstracto, puede ser módificado utilizando dos tipos de atributos que aparecen en cada nodo, atributos heredables y atributos sintetizables.

Gramáticas de attributos

- Los atributos heredables, son valores que se generan en los nodos superiores del árbol y son “heredados” a los nodos hijos, los nodos hijos puede agregar o modificar los valores de los atributos heredables.
- Los atributos sintetizables, son valores que se generan en los nodos hijos y pueden ser utilizados por los padres de los hijos para generar nuevos árboles sintacticos abstractos.
- Aunque es una técnica superior para generar compiladores, los lenguajes imperativos, no han sido muy dúctiles a la hora de implementar gramáticas de atributos, por que básicamente tiene problemas a analizar el orden del cálculo de los atributos (heredables y sintetizables).
- A menos que se utilice evaluación perezosa.

Programación funcional y gramáticas de atributos

- Las gramáticas de atributos pueden ser mapeadas dentro de programas funcionales ([Swierstra and Kuiper, 1986], [Johnson and Walz, 1988], [Bird, 1984]).
- Los lenguajes funcionales puede ser mapeados y descritos por gramáticas de atributos.

El problema Repmin a la AG

```
data Tree
   | Leaf int : Int
   | Bin  lt  : Tree
          rt  : Tree

attr Tree [||min:Int]

sem Tree
  | Leaf lhs.min = @int
  | Bin  lhs.min = @lt.min `min` @rt.min

attr Tree [rmin:Int||]

sem Tree
  | Bin  lt.rmin = @lhs.rmin
         rt.rmin = @lhs.rmin
```

Repmin a la AG

```
data Root
  | Root tree:Tree

sem Root
  | Root tree.rmin = @tree.min

attr Root Tree [||tree:Tree]

sem Tree
  | Leaf lhs.tree = Tree_Leaf @lhs.rmin
  | Bin  lhs.tree = Tree_Bin  @lt.tree @rt.tree

sem Root
  | Root lhs.tree = @tree.tree

deriving Tree:Show
```

Conclusiones

- La programación funcional permite manejar los mismos tipos de problemas que cualquier lenguaje de programación imperativo puede hacer, pero en una forma diferente.

Conclusiones

- La programación funcional permite manejar los mismos tipos de problemas que cualquier lenguaje de programación imperativo puede hacer, pero en una forma diferente.
- La evaluación perezosa ofrece el soporte para implementar estrategias de soluciones.

Conclusiones

- La programación funcional permite manejar los mismos tipos de problemas que cualquier lenguaje de programación imperativo puede hacer, pero en una forma diferente.
- La evaluación perezosa ofrece el soporte para implementar estrategias de soluciones.
- Las gramáticas de atributos basada en lenguajes funcionales, permite tomar la potencia de la programación funcional, con la facilidad para describir la semántica de los atributos, para manejar fácilmente ese tipo de problemas.

Bibliografía

Bird, R. (1984).

Using Circular Programs to Eliminate Multiple Traversals of Data.

Acta Informatica, 21(3).

Johnson, G. F. and Walz, J. A. (1988).

Incremental Evaluation for a General Class of Circular Attribute Grammars.

In *PLDI '88: Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation*, SIGPLAN.

Knuth, D. E. (1968).

Semantics of Context-free Languages.

Mathematical System Theory, 2(2).

Swierstra, S. D. and Kuiper, M. F. (1986).

Using Attribute Grammars to Derive Efficient Functional Programs.

Technical Report RUU–CS–86–16, Utrecht University.