Paradigmas Fundamentales de Programación

Del Lenguaje Núcleo a un Lenguaje Práctico

Juan Francisco Díaz Frias

Maestría en Ingeniería, Énfasis en Ingeniería de Sistemas y Computación
Escuela de Ingeniería de Sistemas y Computación,
home page: http://eisc.univalle.edu.co
Universidad del Valle - Cali, Colombia

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Conveniencias sintácticas (1)

- Escribir valores parciales anidados en una forma concisa.
- Definir y dar un valor inicial a las variables en un solo paso.
- Escribir expresiones en una forma concisa.
- Anidar las declaraciones `if` y `case` en una forma concisa.
- Operadores `andthen` y `orelse` para declaraciones `if` anidadas.
- Las declaraciones se pueden convertir en expresiones usando una marca de anidamiento.

Conveniencias sintácticas (1)

- Escribir valores parciales anidados en una forma concisa.
- Definir y dar un valor inicial a las variables en un solo paso.
- Escribir expresiones en una forma concisa.
- Anidar las declaraciones `if` y `case` en una forma concisa.
- Operadores `andthen` y `orelse` para declaraciones `if` anidadas.
- Las declaraciones se pueden convertir en expresiones usando una marca de anidamiento.

Conveniencias sintácticas (1)

- Escribir valores parciales anidados en una forma concisa.
- Definir y dar un valor inicial a las variables en un solo paso.
- Escribir expresiones en una forma concisa.
- Anidar las declaraciones `if` y `case` en una forma concisa.
- Operadores `andthen` y `orelse` para declaraciones `if` anidadas.
- Las declaraciones se pueden convertir en expresiones usando una marca de anidamiento.

Conveniencias sintácticas (1)

- Escribir valores parciales anidados en una forma concisa.
- Definir y dar un valor inicial a las variables en un solo paso.
- Escribir expresiones en una forma concisa.
- Anidar las declaraciones `if` y `case` en una forma concisa.
- Operadores `andthen` y `orelse` para declaraciones `if` anidadas.
- Las declaraciones se pueden convertir en expresiones usando una marca de anidamiento.

Conveniencias sintácticas (1)

- Escribir valores parciales anidados en una forma concisa.
- Definir y dar un valor inicial a las variables en un solo paso.
- Escribir expresiones en una forma concisa.
- Anidar las declaraciones `if` y `case` en una forma concisa.
- Operadores `andthen` y `orelse` para declaraciones `if` anidadas.
- Las declaraciones se pueden convertir en expresiones usando una marca de anidamiento.

Conveniencias sintácticas (1)

- Escribir valores parciales anidados en una forma concisa.
- Definir y dar un valor inicial a las variables en un solo paso.
- Escribir expresiones en una forma concisa.
- Anidar las declaraciones `if` y `case` en una forma concisa.
- Operadores `andthen` y `orelse` para declaraciones `if` anidadas.
- Las declaraciones se pueden convertir en expresiones usando una marca de anidamiento.

Conveniencias sintácticas (2)

Inicialización implícita de variables

```
local X=⟨expresión⟩
 in ⟨declaración⟩ end
```

Se declara X y se liga al resultado de ⟨expresión⟩.
El caso general es:

```
local ⟨patrón⟩=⟨expresión⟩
 in ⟨declaración⟩ end
```

donde ⟨patrón⟩ es cualquier valor parcial.

- Define todas las variables en ⟨patrón⟩.
- Liga ⟨patrón⟩ con el resultado de ⟨expresión⟩.
- Los identificadores de variables que aparecen en el lado derecho de la igualdad no se definen.

Conveniencias sintácticas (2)

Inicialización implícita de variables

```
local X=⟨expresión⟩
 in ⟨declaración⟩ end
```

Se declara X y se liga al resultado de ⟨expresión⟩.
El caso general es:

```
local ⟨patrón⟩=⟨expresión⟩
 in ⟨declaración⟩ end
```

donde ⟨patrón⟩ es cualquier valor parcial.

- Define todas las variables en ⟨patrón⟩.
- Liga ⟨patrón⟩ con el resultado de ⟨expresión⟩.
- Los identificadores de variables que aparecen en el lado derecho de la igualdad no se definen.

Ejemplo

```
local árbol(llave:A
            izq:B
            der:C
            valor:D)=T
 in ⟨declaración⟩ end
```

es estrictamente equivalente a:

```
local A B C D in
   T=árbol(llave:A
           izq:B
           der:C
           valor:D)
   ⟨declaración⟩
end
```

Conveniencias sintácticas (2)

Inicialización implícita de variables

```
local X=⟨expresión⟩
 in ⟨declaración⟩ end
```

Se declara X y se liga al resultado de ⟨expresión⟩.
El caso general es:

```
local ⟨patrón⟩=⟨expresión⟩
 in ⟨declaración⟩ end
```

donde ⟨patrón⟩ es cualquier valor parcial.

- Define todas las variables en ⟨patrón⟩.
- Liga ⟨patrón⟩ con el resultado de ⟨expresión⟩.
- Los identificadores de variables que aparecen en el lado derecho de la igualdad no se definen.

Ejemplo

```
local árbol(llave:A
            izq:B
            der:C
            valor:D)=T
 in ⟨declaración⟩ end
```

es estrictamente equivalente a:

```
local A B C D in
   T=árbol(llave:A
           izq:B
           der:C
           valor:D)
   ⟨declaración⟩
end
```

Conveniencias sintácticas (2)

Inicialización implícita de variables

```
local X=⟨expresión⟩
 in ⟨declaración⟩ end
```

Se declara X y se liga al resultado de ⟨expresión⟩.
El caso general es:

```
local ⟨patrón⟩=⟨expresión⟩
 in ⟨declaración⟩ end
```

donde ⟨patrón⟩ es cualquier valor parcial.

- Define todas las variables en ⟨patrón⟩.
- Liga ⟨patrón⟩ con el resultado de ⟨expresión⟩.
- Los identificadores de variables que aparecen en el lado derecho de la igualdad no se definen.

Ejemplo

```
local árbol(llave:A
            izq:B
            der:C
            valor:D)=T
   in ⟨declaración⟩ end
```

es estrictamente equivalente a:

```
local A B C D in
   T=árbol(llave:A
           izq:B
           der:C
           valor:D)
   ⟨declaración⟩
end
```

Conveniencias sintácticas (2)

Inicialización implícita de variables

```
local X=⟨expresión⟩
 in ⟨declaración⟩ end
```

Se declara `X` y se liga al resultado de ⟨expresión⟩.
El caso general es:

```
local ⟨patrón⟩=⟨expresión⟩
 in ⟨declaración⟩ end
```

donde ⟨patrón⟩ es cualquier valor parcial.

- Define todas las variables en ⟨patrón⟩.
- Liga ⟨patrón⟩ con el resultado de ⟨expresión⟩.
- Los identificadores de variables que aparecen en el lado derecho de la igualdad no se definen.

Ejemplo

```
local árbol(llave:A
            izq:B
            der:C
            valor:D)=T
   in ⟨declaración⟩ end
```

es estrictamente equivalente a:

```
local A B C D in
   T=árbol(llave:A
           izq:B
           der:C
           valor:D)
   ⟨declaración⟩
end
```

Conveniencias sintácticas (3)

Exps. para calcular con números

⟨exp⟩ ::= ⟨variable⟩ | ⟨ent⟩ | ⟨flot⟩
| ⟨opUnario⟩ ⟨exp⟩
| ⟨exp⟩ ⟨opBinEval⟩ ⟨exp⟩
| '(' ⟨exp⟩ ')'
| '{' ⟨exp⟩ { ⟨exp⟩ } '}'
| ...

⟨opUnario⟩ ::= '~' | ...

⟨opBinEval⟩ ::= '+' | '-' | '*'
| '/' | **div** | **mod**
| '==' | '\=' | '<'
| '=<' | '>' | '>=' | ...

La declaración `if`

⟨dec⟩ ::= `if` ⟨exp⟩ `then` ⟨decEn⟩
{ `elseif` ⟨exp⟩ `then` ⟨decEn⟩ }
[`else` ⟨decEn⟩] `end`
| ...

⟨decEn⟩ ::= [{ ⟨parteDec⟩ }+ `in`] ⟨dec⟩

Conveniencias sintácticas (3)

Exps. para calcular con números

⟨exp⟩ ::= ⟨variable⟩ | ⟨ent⟩ | ⟨flot⟩
| ⟨opUnario⟩ ⟨exp⟩
| ⟨exp⟩ ⟨opBinEval⟩ ⟨exp⟩
| '(' ⟨exp⟩ ')'
| '{' ⟨exp⟩ { ⟨exp⟩ } '}'
| ...

⟨opUnario⟩ ::= '~' | ...

⟨opBinEval⟩ ::= '+' | '-' | '*'
| '/' | **div** | **mod**
| '==' | '\=' | '<'
| '=<' | '>' | '>=' | ...

La declaración `if`

⟨dec⟩ ::= **if** ⟨exp⟩ **then** ⟨decEn⟩
{ **elseif** ⟨exp⟩ **then** ⟨decEn⟩ }
[**else** ⟨decEn⟩] **end**
| ...

⟨decEn⟩ ::= [{ ⟨parteDec⟩ }+ **in**] ⟨dec⟩

Conveniencias sintácticas (4)

La declaración `case`

⟨dec⟩ ::= **`case`** ⟨exp⟩
`of` ⟨patrón⟩ [**`andthen`** ⟨exp⟩] **`then`** ⟨decEn⟩
{ `'[]'` ⟨patrón⟩ [**`andthen`** ⟨exp⟩] **`then`** ⟨decEn⟩ }
[**`else`** ⟨decEn⟩] **`end`**
| ...

⟨patrón⟩ ::= ⟨variable⟩ | ⟨átomo⟩ | ⟨ent⟩ | ⟨flot⟩
| ⟨string⟩ | **`unit`** | **`true`** | **`false`**
| ⟨etiq⟩ `'('` { [⟨campo⟩ `':'`] ⟨patrón⟩ } [`'...'`] `')'`
| ⟨patrón⟩ ⟨consOpBin⟩ ⟨patrón⟩
| `'['` { ⟨patrón⟩ }+ `']'`

⟨consOpBin⟩ ::= `'#'` | `'|'`

Conveniencias sintácticas (5)

El operador `andthen`

$\langle \text{exp} \rangle_1$ **`andthen`** $\langle \text{exp} \rangle_2$

se traduce en

```
if ⟨exp⟩1
   then ⟨exp⟩2
   else false end
```

La ventaja de usar **`andthen`** es que $\langle \text{expresión} \rangle_2$ no es evaluada si $\langle \text{expresión} \rangle_1$ es **`false`**.

El operador `orelse`

$\langle \text{exp} \rangle_1$ **`orelse`** $\langle \text{exp} \rangle_2$

se traduce en

```
if ⟨exp⟩1
   then true
   else ⟨exp⟩2 end
```

Es decir, $\langle \text{expresión} \rangle_2$ no es evaluada si $\langle \text{expresión} \rangle_1$ es **`true`**.

Conveniencias sintácticas (6)

La marca de anidamiento "`$`"

- Convierte cualquier declaración en una expresión.
- El valor de la expresión es lo que está en la posición indicada por la marca de anidamiento.
- La declaración `{P X1 X2 X3}` puede ser convertida en `{P X1 $ X3}`, la cual es una expresión cuyo valor es `X2`.
- Usarla sólo si incrementa grandemente la legibilidad. Por ejemplo, en lugar de escribir

```
local X in {Obj get(X)} {Browse X} end
```

escribir

```
{Browse {Obj get($)}}.
```

Conveniencias sintácticas (6)

La marca de anidamiento "$"

- Convierte cualquier declaración en una expresión.
- El valor de la expresión es lo que está en la posición indicada por la marca de anidamiento.
- La declaración `{P X1 X2 X3}` puede ser convertida en `{P X1 $ X3}`, la cual es una expresión cuyo valor es `X2`.
- Usarla sólo si incrementa grandemente la legibilidad. Por ejemplo, en lugar de escribir

```
local X in {Obj get(X)} {Browse X} end
```

escribir

```
{Browse {Obj get($)}}.
```

Conveniencias sintácticas (6)

La marca de anidamiento "`$`"

- Convierte cualquier declaración en una expresión.
- El valor de la expresión es lo que está en la posición indicada por la marca de anidamiento.
- La declaración `{P X1 X2 X3}` puede ser convertida en `{P X1 $ X3}`, la cual es una expresión cuyo valor es `X2`.
- Usarla sólo si incrementa grandemente la legibilidad. Por ejemplo, en lugar de escribir

```
local X in {Obj get(X)} {Browse X} end
```

escribir

```
{Browse {Obj get($)}}.
```

Conveniencias sintácticas (6)

La marca de anidamiento "`$`"

- Convierte cualquier declaración en una expresión.
- El valor de la expresión es lo que está en la posición indicada por la marca de anidamiento.
- La declaración `{P X1 X2 X3}` puede ser convertida en `{P X1 $ X3}`, la cual es una expresión cuyo valor es `X2`.
- Usarla sólo si incrementa grandemente la legibilidad. Por ejemplo, en lugar de escribir

```
local X in {Obj get(X)} {Browse X} end
```

escribir

```
{Browse {Obj get($)}}.
```

Universidad del Valle

Definición de funciones

```
fun {F X1 ... XN}
    ⟨decl⟩  ⟨exp⟩
end
```

Se traduce en:

```
proc {F X1 ... XN ?R}
     ⟨decl⟩ R=⟨exp⟩
end
```

Por ejemplo:

```
fun {Max X Y}
   if X>=Y then X
           else Y
   end
end
```

Se traduce a:

```
proc {Max X Y ?R}
   R = if X>=Y then X
               else Y
       end
end
```

Y esto se traduce a:

```
proc {Max X Y ?R}
   if X>=Y then R=X
           else R=Y
   end
end
```

Definición de funciones

```
fun {F X1 ... XN}
    ⟨decl⟩  ⟨exp⟩
end
```

Se traduce en:

```
proc {F X1 ... XN ?R}
     ⟨decl⟩ R=⟨exp⟩
end
```

Por ejemplo:

```
fun {Max X Y}
   if X>=Y then X
           else Y
   end
end
```

Se traduce a:

```
proc {Max X Y ?R}
   R = if X>=Y then X
               else Y
       end
end
```

Y esto se traduce a:

```
proc {Max X Y ?R}
   if X>=Y then R=X
           else R=Y
   end
end
```

Definición de funciones

```
fun {F X1 ... XN}
    ⟨decl⟩  ⟨exp⟩
end
```

Se traduce en:

```
proc {F X1 ... XN ?R}
     ⟨decl⟩ R=⟨exp⟩
end
```

Por ejemplo:

```
fun {Max X Y}
   if X>=Y then X
           else Y
   end
end
```

Se traduce a:

```
proc {Max X Y ?R}
   R = if X>=Y then X
               else Y
       end
end
```

Y esto se traduce a:

```
proc {Max X Y ?R}
   if X>=Y then R=X
           else R=Y
   end
end
```

Definición de funciones

```
fun {F X1 ... XN}
    ⟨decl⟩  ⟨exp⟩
end
```

Se traduce en:

```
proc {F X1 ... XN ?R}
     ⟨decl⟩ R=⟨exp⟩
end
```

Por ejemplo:

```
fun {Max X Y}
   if X>=Y then X
           else Y
   end
end
```

Se traduce a:

```
proc {Max X Y ?R}
   R = if X>=Y then X
               else Y
       end
end
```

Y esto se traduce a:

```
proc {Max X Y ?R}
   if X>=Y then R=X
           else R=Y
   end
end
```

Invocación de funciones

Invocación anidada

```
{Q {F X1 ... XN} ...}
```

se traduce en:

```
local R in
   {F X1 ... XN R}
   {Q R ...}
end
```

Invocación de funciones en estructuras de datos

```
Ys={F X}|{Map Xr F}
```

La traducción coloca las invocaciones anidadas después de la operación de ligadura:

```
local Y Yr in
   Ys=Y|Yr
   {F X Y}
   {Map Xr F Yr}
end
```

Invocación de funciones

Invocación anidada

```
{Q {F X1 ... XN} ...}
```

se traduce en:

```
local R in
   {F X1 ... XN R}
   {Q R ...}
end
```

Invocación de funciones en estructuras de datos

```
Ys={F X}|{Map Xr F}
```

La traducción coloca las invocaciones anidadas después de la operación de ligadura:

```
local Y Yr in
   Ys=Y|Yr
   {F X Y}
   {Map Xr F Yr}
end
```