Paradigmas Fundamentales de Programación

Computaciones iterativas y recursivas

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

Modelos y Paradigmas de Programación

Plan

Plan

Computaciones iterativas (1)

Una computación iterativa ...

es un ciclo que durante su ejecución mantiene el tamaño de la pila acotado por una constante, independientemente del número de iteraciones del ciclo.

Una computación iterativa ...

comienza en un estado inicial S_0 y transforman su estado en etapas sucesivas hasta llegar a un estado final S_{final}:

$$S_0 \rightarrow S_1 \rightarrow \cdots \rightarrow S_{\text{final}}$$

Esquema general

```
fun {Iterar Si}
   if {EsFinal Si} then Si
   else Si+1 in
      Si+1={Transformar Si}
      {Iterar Si+1}
   end
end
```

`EsFinal` y `Transformar` son dependientes del problema.

Computaciones iterativas (1)

Una computación iterativa ...

es un ciclo que durante su ejecución mantiene el tamaño de la pila acotado por una constante, independientemente del número de iteraciones del ciclo.

Una computación iterativa ...

comienza en un estado inicial S_0 y transforman su estado en etapas sucesivas hasta llegar a un estado final S_{final}:

$$S_0 \rightarrow S_1 \rightarrow \cdots \rightarrow S_{\mathrm{final}}$$

Esquema general

```
fun {Iterar Si}
   if {EsFinal Si} then Si
   else Si+1 in
      Si+1={Transformar Si}
      {Iterar Si+1}
   end
end
```

`EsFinal` y `Transformar` son dependientes del problema.

Modelos y Paradigmas de Programación

Universidad del Valle

Computaciones iterativas (1)

Una computación iterativa ...

es un ciclo que durante su ejecución mantiene el tamaño de la pila acotado por una constante, independientemente del número de iteraciones del ciclo.

Una computación iterativa ...

comienza en un estado inicial S_0 y transforman su estado en etapas sucesivas hasta llegar a un estado final S_{final}:

$$S_0 \rightarrow S_1 \rightarrow \cdots \rightarrow S_{\text{final}}$$

Esquema general

```
fun {Iterar Si}
   if {EsFinal Si} then Si
   else Si+1 in
      Si+1={Transformar Si}
      {Iterar Si+1}
   end
end
```

EsFinal y *Transformar* son dependientes del problema.

Computaciones iterativas (2)

El esquema es iterativo

- Inicialmente [`R={Iterar` S_0`}`].
- Si `{EsFinal` S_0`}` devuelve **`false`**, después del **`if`**, la pila semántica es [S_1`={Transformar` S_0`}`, `R={Iterar` S_1`}`].
- Después de `{Transformar` S_0`}`, la pila semántica es [`R={Iterar` S_1`}`].
- Después de cada invocación recursiva: `R={Iterar` S_{i+1}`}`.

Método de Newton para $\sqrt{x}$

Modelos y Paradigmas de Programación

Universidad del Valle

Computaciones iterativas (2)

El esquema es iterativo

Método de Newton para $\sqrt{x}$

- Inicialmente [`R={Iterar` S_0`}`].
- Si `{EsFinal` S_0`}` devuelve **`false`**, después del **`if`**, la pila semántica es [S_1`={Transformar` S_0`}`, `R={Iterar` S_1`}`].
- Después de `{Transformar` S_0`}`, la pila semántica es [`R={Iterar` S_1`}`].
- Después de cada invocación recursiva: `R={Iterar` S_{i+1}`}`.

Computaciones iterativas (2)

El esquema es iterativo

- Inicialmente $\left[\texttt{R=\{Iterar } S_0\texttt{\}}\right]$.
- Si `{`*`EsFinal`* S_0`}` devuelve **`false`**, después del **`if`**, la pila semántica es $\left[S_1\texttt{=\{}\textit{Transformar } S_0\texttt{\},}\right.$ $\left.\texttt{R=\{Iterar } S_1\texttt{\}}\right]$.
- Después de `{`*`Transformar`* S_0`}`, la pila semántica es $\left[\texttt{R=\{Iterar } S_1\texttt{\}}\right]$.
- Después de cada invocación recursiva: $\texttt{R=\{Iterar } S_{i+1}\texttt{\}}$.

Método de Newton para $\sqrt{x}$

- Comenzar con una adivinanza a de la raíz cuadrada, y mejorar esta adivinanza iterativamente hasta que sea suficientemente buena.

Computaciones iterativas (2)

El esquema es iterativo

- Inicialmente [`R={Iterar` S_0`}`].
- Si `{EsFinal` S_0`}` devuelve **`false`**, después del **`if`**, la pila semántica es [S_1`={Transformar` S_0`},` `R={Iterar` S_1`}`].
- Después de `{Transformar` S_0`}`, la pila semántica es [`R={Iterar` S_1`}`].
- Después de cada invocación recursiva: `R={Iterar` S_{i+1}`}`.

Método de Newton para $\sqrt{x}$

- Comenzar con una adivinanza a de la raíz cuadrada, y mejorar esta adivinanza iterativamente hasta que sea suficientemente buena.
- La adivinanza mejorada es $a' = (a + x/a)/2$.
  $\varepsilon = a - \sqrt{x}$
  $\varepsilon' = a' - \sqrt{x} = (a + x/a)/2 - \sqrt{x} = \varepsilon^2/2a$
  $\varepsilon' < \varepsilon$ ssi $\varepsilon^2/2a < \varepsilon$ ssi $\varepsilon < 2a$.
  Substituyendo por la definición de ε, la condición resultante es $\sqrt{x} + a > 0$.
  Si $x > 0$ y la adivinanza inicial $a > 0$, entonces esta condición siempre es cierta.
  Entonces el algoritmo siempre converge.

Computaciones iterativas (2)

El esquema es iterativo

- Inicialmente [`R={Iterar` S_0`}`].
- Si `{EsFinal` S_0`}` devuelve **`false`**, después del **`if`**, la pila semántica es [S_1`={Transformar` S_0`}`, `R={Iterar` S_1`}`].
- Después de `{Transformar` S_0`}`, la pila semántica es [`R={Iterar` S_1`}`].
- Después de cada invocación recursiva: `R={Iterar` S_{i+1}`}`.

Método de Newton para $\sqrt{x}$

- Comenzar con una adivinanza a de la raíz cuadrada, y mejorar esta adivinanza iterativamente hasta que sea suficientemente buena.
- La adivinanza mejorada es $a' = (a + x/a)/2$.
  $\varepsilon = a - \sqrt{x}$
  $\varepsilon' = a' - \sqrt{x} = (a + x/a)/2 - \sqrt{x} = \varepsilon^2/2a$
  $\varepsilon' < \varepsilon$ ssi $\varepsilon^2/2a < \varepsilon$ ssi $\varepsilon < 2a$.
  Substituyendo por la definición de ε, la condición resultante es $\sqrt{x} + a > 0$.
  Si $x > 0$ y la adivinanza inicial $a > 0$, entonces esta condición siempre es cierta.
  Entonces el algoritmo siempre converge.

Modelos y Paradigmas de Programación

Universidad del Valle

Computaciones iterativas (2)

El esquema es iterativo

- Inicialmente $\left[\texttt{R=\{Iterar } S_0\texttt{\}}\right]$.
- Si `{EsFinal` S_0`}` devuelve **`false`**, después del **`if`**, la pila semántica es $\left[S_1\texttt{=\{Transformar } S_0\texttt{\}},\ \texttt{R=\{Iterar } S_1\texttt{\}}\right]$.
- Después de `{Transformar` S_0`}`, la pila semántica es $\left[\texttt{R=\{Iterar } S_1\texttt{\}}\right]$.
- Después de cada invocación recursiva: `R={Iterar` S_{i+1}`}`.

Método de Newton para $\sqrt{x}$

- Comenzar con una adivinanza a de la raíz cuadrada, y mejorar esta adivinanza iterativamente hasta que sea suficientemente buena.
- La adivinanza mejorada es $a' = (a + x/a)/2$.
  $\varepsilon = a - \sqrt{x}$
  $\varepsilon' = a' - \sqrt{x} = (a + x/a)/2 - \sqrt{x} = \varepsilon^2/2a$
  $\varepsilon' < \varepsilon$ ssi $\varepsilon^2/2a < \varepsilon$ ssi $\varepsilon < 2a$.
  Substituyendo por la definición de ε, la condición resultante es $\sqrt{x} + a > 0$.
  Si $x > 0$ y la adivinanza inicial $a > 0$, entonces esta condición siempre es cierta.
  Entonces el algoritmo siempre converge.

Computaciones iterativas (3)

Primera versión:Principal

```
fun {Raíz X}
   Adiv=1.0
in
   {RaízIter Adiv X}
end
fun {RaízIter Adiv X}
   if {Buena Adiv X}
      then Adiv
      else
      {RaízIter {Mejorar
                    Adiv
                    X}
                 X}
   end
end
```

Primera versión: Auxiliares

```
fun {Mejorar Adiv X}
  (Adiv + X/Adiv) / 2.0
end

fun {Buena Adiv X}
  {Abs X-Adiv*Adiv}/X < 0.00001
end

fun {Abs X}
  if X<0.0 then ~X
           else X
  end
end
```

Computaciones iterativas (3)

Primera versión:Principal

```
fun {Raíz X}
   Adiv=1.0
in
   {RaízIter Adiv X}
end
fun {RaízIter Adiv X}
   if {Buena Adiv X}
      then Adiv
      else
      {RaízIter {Mejorar
                    Adiv
                    X}
                 X}
   end
end
```

Primera versión: Auxiliares

```
fun {Mejorar Adiv X}
  (Adiv + X/Adiv) / 2.0
end

fun {Buena Adiv X}
  {Abs X-Adiv*Adiv}/X < 0.00001
end

fun {Abs X}
  if X<0.0 then ~X
           else X
  end
end
```

Modelos y Paradigmas de Programación

Computaciones iterativas (4)

Segunda versión: esconder lo auxiliar

```
local
   fun {Mejorar Adiv X}
     (Adiv + X/Adiv) / 2.0
   end
   fun {Buena Adiv X}
     {Abs X-Adiv*Adiv}/X < 0.00001
   end
   fun {RaízIter Adiv X}
      if {Buena Adiv X} then Adiv
      else
         {RaízIter {Mejorar Adiv X}
                   X}
      end
   end
```

Segunda versión: sólo es visible lo principal

```
in
   fun {Raíz X}
      Adiv=1.0
   in
      {RaízIter
         Adiv
         X}
   end
end
```

Computaciones iterativas (4)

Segunda versión: esconder lo auxiliar

```
local
   fun {Mejorar Adiv X}
     (Adiv + X/Adiv) / 2.0
   end
   fun {Buena Adiv X}
     {Abs X-Adiv*Adiv}/X < 0.00001
   end
   fun {RaízIter Adiv X}
      if {Buena Adiv X} then Adiv
      else
         {RaízIter {Mejorar Adiv X}
                   X}
      end
   end
```

Segunda versión: sólo es visible lo principal

```
in
   fun {Raíz X}
      Adiv=1.0
   in
      {RaízIter
         Adiv
         X}
   end
end
```

Computaciones iterativas (5)

Compromiso eficiencia vs. ocultamiento

```
fun {Raíz X}
   fun {Mejorar Adiv}
      (Adiv + X/Adiv) / 2.0
   end
   fun {Buena Adiv}
      {Abs X-Adiv*Adiv}/X < 0.00001
   end
   fun {RaízIter Adiv}
      if {Buena Adiv} then Adiv
      else{RaízIter {Mejorar Adiv}}
      end
   end
   Adiv=1.0
in
   {RaízIter Adiv}
end
```

Del esquema general a una abstracción de control (1)

Esquema general

```
fun {Iterar S_i}
   if {EsFinal S_i} then S_i
   else S_{i+1} in
      S_{i+1}={Transformar S_i}
      {Iterar S_{i+1}}
   end
end
```

`EsFinal` y `Transformar` son dependientes del problema.

Programa

```
fun {Iterar S
            EsFinal
            Transformar}
   if {EsFinal S} then S
   else S1 in
      S1={Transformar S}
      {Iterar S1
              EsFinal
              Transformar}
   end
end
```

Los argumentos `EsFinal` y `Transformar` se presentan como funciones de un argumento.

Del esquema general a una abstracción de control (1)

Esquema general

```
fun {Iterar S_i}
   if {EsFinal S_i} then S_i
   else S_{i+1} in
      S_{i+1}={Transformar S_i}
      {Iterar S_{i+1}}
   end
end
```

EsFinal y *Transformar* son dependientes del problema.

Programa

```
fun {Iterar S
            EsFinal
            Transformar}
   if {EsFinal S} then S
   else S1 in
      S1={Transformar S}
      {Iterar S1
              EsFinal
              Transformar}
   end
end
```

Los argumentos `EsFinal` y `Transformar` se presentan como funciones de un argumento.

Del esquema general a una abstracción de control (2)

Newton usando la abstracción

```
fun {Raíz X}
   {Iterar
      1.0
      fun {$ Ad} {Abs X-Ad*Ad}/X<0.00001 end
      fun {$ Ad} (Ad+X/Ad)/2.0 end}
end
```

Plan

Modelos y Paradigmas de Programación

Universidad del Valle

Computaciones recursivas (1)

Características

- La recursión es más general que la iteración.
- Una función recursiva puede invocarse a sí misma en cualquier lugar en el cuerpo y puede hacerlo más de una vez.
- Dos formas de recursión: en funciones (una función que se invoca a sí misma) o en tipos de datos (v.gr. una lista).
- El tamaño de la pila en una computación recursiva puede crecer a medida que el tamaño de la entrada lo hace.
- Una parte importante de la programación declarativa es evitar que el tamaño de la pila crezca.

Definición ingenua del factorial

```
fun {Fact N}
   if N==0 then 1
   elseif N>0
      then N*{Fact N-1}
      else raise error
      end
   end
end
```

Computaciones recursivas (1)

Características

- La recursión es más general que la iteración.
- Una función recursiva puede invocarse a sí misma en cualquier lugar en el cuerpo y puede hacerlo más de una vez.
- Dos formas de recursión: en funciones (una función que se invoca a sí misma) o en tipos de datos (v.gr. una lista).
- El tamaño de la pila en una computación recursiva puede crecer a medida que el tamaño de la entrada lo hace.
- Una parte importante de la programación declarativa es evitar que el tamaño de la pila crezca.

Definición ingenua del factorial

```
fun {Fact N}
   if N==0 then 1
   elseif N>0
       then N*{Fact N-1}
       else raise error
       end
   end
end
```

Computaciones recursivas (1)

Características

- La recursión es más general que la iteración.
- Una función recursiva puede invocarse a sí misma en cualquier lugar en el cuerpo y puede hacerlo más de una vez.
- Dos formas de recursión: en funciones (una función que se invoca a sí misma) o en tipos de datos (v.gr. una lista).
- El tamaño de la pila en una computación recursiva puede crecer a medida que el tamaño de la entrada lo hace.
- Una parte importante de la programación declarativa es evitar que el tamaño de la pila crezca.

Definición ingenua del factorial

```
fun {Fact N}
   if N==0 then 1
   elseif N>0
       then N*{Fact N-1}
       else raise error
       end
   end
end
```

Computaciones recursivas (1)

Características

- La recursión es más general que la iteración.
- Una función recursiva puede invocarse a sí misma en cualquier lugar en el cuerpo y puede hacerlo más de una vez.
- Dos formas de recursión: en funciones (una función que se invoca a sí misma) o en tipos de datos (v.gr. una lista).
- El tamaño de la pila en una computación recursiva puede crecer a medida que el tamaño de la entrada lo hace.
- Una parte importante de la programación declarativa es evitar que el tamaño de la pila crezca.

Definición ingenua del factorial

```
fun {Fact N}
   if N==0 then 1
   elseif N>0
      then N*{Fact N-1}
      else raise error
      end
   end
end
```

- Función parcial.

Computaciones recursivas (1)

Características

- La recursión es más general que la iteración.
- Una función recursiva puede invocarse a sí misma en cualquier lugar en el cuerpo y puede hacerlo más de una vez.
- Dos formas de recursión: en funciones (una función que se invoca a sí misma) o en tipos de datos (v.gr. una lista).
- El tamaño de la pila en una computación recursiva puede crecer a medida que el tamaño de la entrada lo hace.
- Una parte importante de la programación declarativa es evitar que el tamaño de la pila crezca.

Definición ingenua del factorial

```
fun {Fact N}
   if N==0 then 1
   elseif N>0
      then N*{Fact N-1}
      else raise error
      end
   end
end
```

- Función parcial.
- Siempre termina.

Computaciones recursivas (1)

Características

- La recursión es más general que la iteración.
- Una función recursiva puede invocarse a sí misma en cualquier lugar en el cuerpo y puede hacerlo más de una vez.
- Dos formas de recursión: en funciones (una función que se invoca a sí misma) o en tipos de datos (v.gr. una lista).
- El tamaño de la pila en una computación recursiva puede crecer a medida que el tamaño de la entrada lo hace.
- Una parte importante de la programación declarativa es evitar que el tamaño de la pila crezca.

Definición ingenua del factorial

```
fun {Fact N}
   if N==0 then 1
   elseif N>0
      then N*{Fact N-1}
      else raise error
      end
   end
end
```

- Función parcial.
- Siempre termina.
- Es correcta: {Fact N} devuelve $n!$.

Computaciones recursivas (1)

Características

- La recursión es más general que la iteración.
- Una función recursiva puede invocarse a sí misma en cualquier lugar en el cuerpo y puede hacerlo más de una vez.
- Dos formas de recursión: en funciones (una función que se invoca a sí misma) o en tipos de datos (v.gr. una lista).
- El tamaño de la pila en una computación recursiva puede crecer a medida que el tamaño de la entrada lo hace.
- Una parte importante de la programación declarativa es evitar que el tamaño de la pila crezca.

Definición ingenua del factorial

```
fun {Fact N}
   if N==0 then 1
   elseif N>0
      then N*{Fact N-1}
      else raise error
      end
   end
end
```

- Función parcial.
- Siempre termina.
- Es correcta: `{Fact N}` devuelve $n!$.

Computaciones recursivas (1)

Características

- La recursión es más general que la iteración.
- Una función recursiva puede invocarse a sí misma en cualquier lugar en el cuerpo y puede hacerlo más de una vez.
- Dos formas de recursión: en funciones (una función que se invoca a sí misma) o en tipos de datos (v.gr. una lista).
- El tamaño de la pila en una computación recursiva puede crecer a medida que el tamaño de la entrada lo hace.
- Una parte importante de la programación declarativa es evitar que el tamaño de la pila crezca.

Definición ingenua del factorial

```
fun {Fact N}
   if N==0 then 1
   elseif N>0
      then N*{Fact N-1}
      else raise error
      end
   end
end
```

- Función parcial.
- Siempre termina.
- Es correcta: {Fact N} devuelve $n!$.

Computaciones recursivas (1)

Características

- La recursión es más general que la iteración.
- Una función recursiva puede invocarse a sí misma en cualquier lugar en el cuerpo y puede hacerlo más de una vez.
- Dos formas de recursión: en funciones (una función que se invoca a sí misma) o en tipos de datos (v.gr. una lista).
- El tamaño de la pila en una computación recursiva puede crecer a medida que el tamaño de la entrada lo hace.
- Una parte importante de la programación declarativa es evitar que el tamaño de la pila crezca.

Definición ingenua del factorial

```
fun {Fact N}
   if N==0 then 1
   elseif N>0
      then N*{Fact N-1}
      else raise error
      end
   end
end
```

- Función parcial.
- Siempre termina.
- Es correcta: `{Fact N}` devuelve $n!$.

Computaciones recursivas (2)

En el lenguaje núcleo

```
proc {Fact N ?R}
   if N==0 then
      R=1
   elseif N>0
       then
        N1 R1 in
         N1=N-1
         {Fact N1
               R1}
         R=N*R1
       else
        raise error
       end
   end
end
```

Pila semántica al ejecutar `{Fact 5 R}`: tamaño creciente

- $[(\texttt{\{Fact N R\}}, \{N \rightarrow 5, R \rightarrow r_0\})]$.
- $[(\texttt{\{Fact N1 R1\}}, \{N1 \rightarrow 4, R1 \rightarrow r_1, \ldots\}), (\texttt{R=N*R1}, \{R \rightarrow r_0, R1 \rightarrow r_1, N \rightarrow 5, \ldots\})]$

Computaciones recursivas (2)

En el lenguaje núcleo

```
proc {Fact N ?R}
   if N==0 then
      R=1
   elseif N>0
       then
        N1 R1 in
         N1=N-1
         {Fact N1
               R1}
         R=N*R1
       else
        raise error
       end
   end
end
```

Pila semántica al ejecutar `{Fact 5 R}`: tamaño creciente

- $[(\texttt{\{Fact N R\}}, \{\texttt{N} \to 5, \texttt{R} \to r_0\})]$.
- $[(\texttt{\{Fact N1 R1\}}, \{\texttt{N1} \to 4, \texttt{R1} \to r_1, \ldots\}),$
  $(\texttt{R=N*R1}, \{\texttt{R} \to r_0, \texttt{R1} \to r_1, \texttt{N} \to 5, \ldots\})]$
- $[(\texttt{\{Fact N1 R1\}}, \{\texttt{N1} \to 3, \texttt{R1} \to r_2, \ldots\}),$
  $(\texttt{R=N*R1}, \{\texttt{R} \to r_1, \texttt{R1} \to r_2, \texttt{N} \to 4, \ldots\}),$
  $(\texttt{R=N*R1}, \{\texttt{R} \to r_0, \texttt{R1} \to r_1, \texttt{N} \to 5, \ldots\})]$
- $[(\texttt{\{Fact N1 R1\}}, \{\texttt{N1} \to 2, \texttt{R1} \to r_3, \ldots\}),$
  $(\texttt{R=N*R1}, \{\texttt{R} \to r_2, \texttt{R1} \to r_3, \texttt{N} \to 3, \ldots\}),$
  $(\texttt{R=N*R1}, \{\texttt{R} \to r_1, \texttt{R1} \to r_2, \texttt{N} \to 4, \ldots\}),$
  $(\texttt{R=N*R1}, \{\texttt{R} \to r_0, \texttt{R1} \to r_1, \texttt{N} \to 5, \ldots\})]$

- La pila aumenta su tamaño por cada invocación.
- La última invocación recursiva es la quinta: $r_5 = 1$.
- Las cinco multiplicaciones se realizan: $r_0 = 120$.

Computaciones recursivas (2)

En el lenguaje núcleo

```
proc {Fact N ?R}
   if N==0 then
      R=1
   elseif N>0
       then
        N1 R1 in
         N1=N-1
         {Fact N1
               R1}
         R=N*R1
       else
        raise error
       end
   end
end
```

Pila semántica al ejecutar {Fact 5 R}: tamaño creciente

- $[(\{\texttt{Fact N R}\}, \{\texttt{N} \rightarrow 5, \texttt{R} \rightarrow r_0\})]$.
- $[(\{\texttt{Fact N1 R1}\}, \{\texttt{N1} \rightarrow 4, \texttt{R1} \rightarrow r_1, \ldots\}),$
  $(\texttt{R=N*R1}, \{\texttt{R} \rightarrow r_0, \texttt{R1} \rightarrow r_1, \texttt{N} \rightarrow 5, \ldots\})]$
- $[(\{\texttt{Fact N1 R1}\}, \{\texttt{N1} \rightarrow 3, \texttt{R1} \rightarrow r_2, \ldots\}),$
  $(\texttt{R=N*R1}, \{\texttt{R} \rightarrow r_1, \texttt{R1} \rightarrow r_2, \texttt{N} \rightarrow 4, \ldots\}),$
  $(\texttt{R=N*R1}, \{\texttt{R} \rightarrow r_0, \texttt{R1} \rightarrow r_1, \texttt{N} \rightarrow 5, \ldots\})]$
- $[(\{\texttt{Fact N1 R1}\}, \{\texttt{N1} \rightarrow 2, \texttt{R1} \rightarrow r_3, \ldots\}),$
  $(\texttt{R=N*R1}, \{\texttt{R} \rightarrow r_2, \texttt{R1} \rightarrow r_3, \texttt{N} \rightarrow 3, \ldots\}),$
  $(\texttt{R=N*R1}, \{\texttt{R} \rightarrow r_1, \texttt{R1} \rightarrow r_2, \texttt{N} \rightarrow 4, \ldots\}),$
  $(\texttt{R=N*R1}, \{\texttt{R} \rightarrow r_0, \texttt{R1} \rightarrow r_1, \texttt{N} \rightarrow 5, \ldots\})]$

- La pila aumenta su tamaño por cada invocación.
- La última invocación recursiva es la quinta: $r_5 = 1$.
- Las cinco multiplicaciones se realizan: $r_0 = 120$.

Modelos y Paradigmas de Programación

Universidad del Valle

Computaciones recursivas (2)

En el lenguaje núcleo

```
proc {Fact N ?R}
   if N==0 then
      R=1
   elseif N>0
       then
        N1 R1 in
         N1=N-1
         {Fact N1
               R1}
         R=N*R1
       else
        raise error
       end
   end
end
```

Pila semántica al ejecutar {Fact 5 R}: tamaño creciente

- $[(\{\mathtt{Fact\ N\ R}\}, \{\mathtt{N} \rightarrow 5, \mathtt{R} \rightarrow r_0\})]$.
- $[(\{\mathtt{Fact\ N1\ R1}\}, \{\mathtt{N1} \rightarrow 4, \mathtt{R1} \rightarrow r_1, \ldots\}),$ $(\mathtt{R{=}N{*}R1}, \{\mathtt{R} \rightarrow r_0, \mathtt{R1} \rightarrow r_1, \mathtt{N} \rightarrow 5, \ldots\})]$
- $[(\{\mathtt{Fact\ N1\ R1}\}, \{\mathtt{N1} \rightarrow 3, \mathtt{R1} \rightarrow r_2, \ldots\}),$ $(\mathtt{R{=}N{*}R1}, \{\mathtt{R} \rightarrow r_1, \mathtt{R1} \rightarrow r_2, \mathtt{N} \rightarrow 4, \ldots\}),$ $(\mathtt{R{=}N{*}R1}, \{\mathtt{R} \rightarrow r_0, \mathtt{R1} \rightarrow r_1, \mathtt{N} \rightarrow 5, \ldots\})]$
- $[(\{\mathtt{Fact\ N1\ R1}\}, \{\mathtt{N1} \rightarrow 2, \mathtt{R1} \rightarrow r_3, \ldots\}),$ $(\mathtt{R{=}N{*}R1}, \{\mathtt{R} \rightarrow r_2, \mathtt{R1} \rightarrow r_3, \mathtt{N} \rightarrow 3, \ldots\}),$ $(\mathtt{R{=}N{*}R1}, \{\mathtt{R} \rightarrow r_1, \mathtt{R1} \rightarrow r_2, \mathtt{N} \rightarrow 4, \ldots\}),$ $(\mathtt{R{=}N{*}R1}, \{\mathtt{R} \rightarrow r_0, \mathtt{R1} \rightarrow r_1, \mathtt{N} \rightarrow 5, \ldots\})]$

- La pila aumenta su tamaño por cada invocación.
- La última invocación recursiva es la quinta: $r_5 = 1$.
- Las cinco multiplicaciones se realizan: $r_0 = 120$.

Modelos y Paradigmas de Programación

Universidad del Valle

Computaciones recursivas (2)

En el lenguaje núcleo

```
proc {Fact N ?R}
   if N==0 then
      R=1
   elseif N>0
       then
        N1 R1 in
         N1=N-1
         {Fact N1
               R1}
         R=N*R1
       else
        raise error
       end
   end
end
```

Pila semántica al ejecutar {Fact 5 R}: tamaño creciente

- $[(\{\texttt{Fact N R}\}, \{N \to 5, R \to r_0\})]$.
- $[(\{\texttt{Fact N1 R1}\}, \{N1 \to 4, R1 \to r_1, \ldots\}),$ $(R{=}N{*}R1, \{R \to r_0, R1 \to r_1, N \to 5, \ldots\})]$
- $[(\{\texttt{Fact N1 R1}\}, \{N1 \to 3, R1 \to r_2, \ldots\}),$ $(R{=}N{*}R1, \{R \to r_1, R1 \to r_2, N \to 4, \ldots\}),$ $(R{=}N{*}R1, \{R \to r_0, R1 \to r_1, N \to 5, \ldots\})]$
- $[(\{\texttt{Fact N1 R1}\}, \{N1 \to 2, R1 \to r_3, \ldots\}),$ $(R{=}N{*}R1, \{R \to r_2, R1 \to r_3, N \to 3, \ldots\}),$ $(R{=}N{*}R1, \{R \to r_1, R1 \to r_2, N \to 4, \ldots\}),$ $(R{=}N{*}R1, \{R \to r_0, R1 \to r_1, N \to 5, \ldots\})]$

- La pila aumenta su tamaño por cada invocación.
- La última invocación recursiva es la quinta: $r_5 = 1$.
- Las cinco multiplicaciones se realizan: $r_0 = 120$.

Computaciones recursivas (2)

En el lenguaje núcleo

```
proc {Fact N ?R}
   if N==0 then
      R=1
   elseif N>0
       then
        N1 R1 in
         N1=N-1
         {Fact N1
               R1}
         R=N*R1
       else
        raise error
       end
   end
end
```

Pila semántica al ejecutar `{Fact 5 R}`: tamaño creciente

- $[(\{\texttt{Fact N R}\}, \{\texttt{N} \rightarrow 5, \texttt{R} \rightarrow r_0\})]$.
- $[(\{\texttt{Fact N1 R1}\}, \{\texttt{N1} \rightarrow 4, \texttt{R1} \rightarrow r_1, \ldots\}),$
  $(\texttt{R=N*R1}, \{\texttt{R} \rightarrow r_0, \texttt{R1} \rightarrow r_1, \texttt{N} \rightarrow 5, \ldots\})]$
- $[(\{\texttt{Fact N1 R1}\}, \{\texttt{N1} \rightarrow 3, \texttt{R1} \rightarrow r_2, \ldots\}),$
  $(\texttt{R=N*R1}, \{\texttt{R} \rightarrow r_1, \texttt{R1} \rightarrow r_2, \texttt{N} \rightarrow 4, \ldots\}),$
  $(\texttt{R=N*R1}, \{\texttt{R} \rightarrow r_0, \texttt{R1} \rightarrow r_1, \texttt{N} \rightarrow 5, \ldots\})]$
- $[(\{\texttt{Fact N1 R1}\}, \{\texttt{N1} \rightarrow 2, \texttt{R1} \rightarrow r_3, \ldots\}),$
  $(\texttt{R=N*R1}, \{\texttt{R} \rightarrow r_2, \texttt{R1} \rightarrow r_3, \texttt{N} \rightarrow 3, \ldots\}),$
  $(\texttt{R=N*R1}, \{\texttt{R} \rightarrow r_1, \texttt{R1} \rightarrow r_2, \texttt{N} \rightarrow 4, \ldots\}),$
  $(\texttt{R=N*R1}, \{\texttt{R} \rightarrow r_0, \texttt{R1} \rightarrow r_1, \texttt{N} \rightarrow 5, \ldots\})]$

- La pila aumenta su tamaño por cada invocación.
- La última invocación recursiva es la quinta: $r_5 = 1$.
- Las cinco multiplicaciones se realizan: $r_0 = 120$.

Modelos y Paradigmas de Programación

Universidad del Valle

Computaciones recursivas (2)

En el lenguaje núcleo

```
proc {Fact N ?R}
   if N==0 then
      R=1
   elseif N>0
       then
        N1 R1 in
         N1=N-1
         {Fact N1
               R1}
         R=N*R1
       else
        raise error
       end
   end
end
```

Pila semántica al ejecutar `{Fact 5 R}`: tamaño creciente

- $[(\texttt{\{Fact N R\}}, \{\texttt{N} \rightarrow 5, \texttt{R} \rightarrow r_0\})]$.
- $[(\texttt{\{Fact N1 R1\}}, \{\texttt{N1} \rightarrow 4, \texttt{R1} \rightarrow r_1, \ldots\})$, $(\texttt{R=N*R1}, \{\texttt{R} \rightarrow r_0, \texttt{R1} \rightarrow r_1, \texttt{N} \rightarrow 5, \ldots\})]$
- $[(\texttt{\{Fact N1 R1\}}, \{\texttt{N1} \rightarrow 3, \texttt{R1} \rightarrow r_2, \ldots\})$, $(\texttt{R=N*R1}, \{\texttt{R} \rightarrow r_1, \texttt{R1} \rightarrow r_2, \texttt{N} \rightarrow 4, \ldots\})$, $(\texttt{R=N*R1}, \{\texttt{R} \rightarrow r_0, \texttt{R1} \rightarrow r_1, \texttt{N} \rightarrow 5, \ldots\})]$
- $[(\texttt{\{Fact N1 R1\}}, \{\texttt{N1} \rightarrow 2, \texttt{R1} \rightarrow r_3, \ldots\})$, $(\texttt{R=N*R1}, \{\texttt{R} \rightarrow r_2, \texttt{R1} \rightarrow r_3, \texttt{N} \rightarrow 3, \ldots\})$, $(\texttt{R=N*R1}, \{\texttt{R} \rightarrow r_1, \texttt{R1} \rightarrow r_2, \texttt{N} \rightarrow 4, \ldots\})$, $(\texttt{R=N*R1}, \{\texttt{R} \rightarrow r_0, \texttt{R1} \rightarrow r_1, \texttt{N} \rightarrow 5, \ldots\})]$

- La pila aumenta su tamaño por cada invocación.
- La última invocación recursiva es la quinta: $r_5 = 1$.
- Las cinco multiplicaciones se realizan: $r_0 = 120$.

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Modelos y Paradigmas de Programación

Construyendo iteraciones (1)

¿Cómo lograr computaciones iterativas?

- Una computación iterativa se caracteriza por comenzar en un estado inicial S_0, y transformar ese estado en un conjunto de estados intermedios hasta llegar a un estado final S_f:

$$S_0 \longrightarrow S_1 \longrightarrow S_2 \ldots \longrightarrow S_f$$

- Todo estado debe caracterizarse por una condición que siempre se cumple y que se denomina Invariante.
- Tanto el estado inicial, como los intermedios, como el final siempre cumplen esa condición.

Cálculo iterativo del factorial

- **Entrada:**
- **Salida:**

Modelos y Paradigmas de Programación

Universidad del Valle

Construyendo iteraciones (1)

¿Cómo lograr computaciones iterativas?

- Una computación iterativa se caracteriza por comenzar en un estado inicial S_0, y transformar ese estado en un conjunto de estados intermedios hasta llegar a un estado final S_f:

$$S_0 \longrightarrow S_1 \longrightarrow S_2 \ldots \longrightarrow S_f$$

- Todo estado debe caracterizarse por una condición que siempre se cumple y que se denomina Invariante.
- Tanto el estado inicial, como los intermedios, como el final siempre cumplen esa condición.

Cálculo iterativo del factorial

- Entrada: $N \geq 0$
- Salida: $R = N!$
- Idea: Iteración ascendente:

$$(0, 1) \rightarrow (1, 1) \rightarrow (2, 2) \rightarrow (3, 6) \rightarrow \ldots \rightarrow (N, N!)$$

- Estado:

(Invariante)

- Estado Inicial
- Estado Final
- Transformación de estados:

Construyendo iteraciones (1)

¿Cómo lograr computaciones iterativas?

- Una computación iterativa se caracteriza por comenzar en un estado inicial S_0, y transformar ese estado en un conjunto de estados intermedios hasta llegar a un estado final S_f:

$$S_0 \longrightarrow S_1 \longrightarrow S_2 \ldots \longrightarrow S_f$$

- Todo estado debe caracterizarse por una condición que siempre se cumple y que se denomina Invariante.
- Tanto el estado inicial, como los intermedios, como el final siempre cumplen esa condición.

Cálculo iterativo del factorial

- Entrada: $N \geq 0$
- Salida: $R = N!$
- Idea: Iteración ascendente:

$$(0,1) \rightarrow (1,1) \rightarrow (2,2) \rightarrow (3,6) \rightarrow \ldots \rightarrow (N, N!)$$

- Estado: Tupla de la forma (I, R) tal que $I \leq N \wedge R = I!$ (Invariante)
- Estado inicial:
- Estado Final:
- Transformación de estados:

Modelos y Paradigmas de Programación

Universidad del Valle

Construyendo iteraciones (1)

¿Cómo lograr computaciones iterativas?

- Una computación iterativa se caracteriza por comenzar en un estado inicial S_0, y transformar ese estado en un conjunto de estados intermedios hasta llegar a un estado final S_f:

$$S_0 \longrightarrow S_1 \longrightarrow S_2 \ldots \longrightarrow S_f$$

- Todo estado debe caracterizarse por una condición que siempre se cumple y que se denomina Invariante.
- Tanto el estado inicial, como los intermedios, como el final siempre cumplen esa condición.

Cálculo iterativo del factorial

- Entrada: $N \geq 0$
- Salida: $R = N!$
- Idea: Iteración ascendente:

$$(0,1) \longrightarrow (1,1) \longrightarrow (2,2) \longrightarrow (3,6) \longrightarrow \ldots \longrightarrow (N, N!)$$

- Estado: Tupla de la forma (I, R) tal que $I \leq N \wedge R = I!$ (Invariante)
- Estado Inicial: $I = 0, \ R = 1$
- Estado Final $I = N$
- Transformación de estados:

$$(I, R) \longrightarrow (I+1, (I+1) * R)$$

Construyendo iteraciones (1)

¿Cómo lograr computaciones iterativas?

- Una computación iterativa se caracteriza por comenzar en un estado inicial S_0, y transformar ese estado en un conjunto de estados intermedios hasta llegar a un estado final S_f:

$$S_0 \rightarrow S_1 \rightarrow S_2 \ldots \rightarrow S_f$$

- Todo estado debe caracterizarse por una condición que siempre se cumple y que se denomina Invariante.
- Tanto el estado inicial, como los intermedios, como el final siempre cumplen esa condición.

Cálculo iterativo del factorial

- Entrada: $N \geq 0$
- Salida: $R = N!$
- Idea: Iteración ascendente:

$$(0,1) \rightarrow (1,1) \rightarrow (2,2) \rightarrow (3,6) \rightarrow \ldots \rightarrow (N, N!)$$

- Estado: Tupla de la forma (I, R) tal que $I \leq N \wedge R = I!$ (Invariante)
- Estado Inicial: $I = 0, \; R = 1$
- Estado Final $I = N$
- Transformación de estados:

$$(I, R) \rightarrow (I+1, (I+1) * R)$$

Modelos y Paradigmas de Programación

Universidad del Valle

Construyendo iteraciones (1)

¿Cómo lograr computaciones iterativas?

- Una computación iterativa se caracteriza por comenzar en un estado inicial S_0, y transformar ese estado en un conjunto de estados intermedios hasta llegar a un estado final S_f:

$$S_0 \rightarrow S_1 \rightarrow S_2 \ldots \rightarrow S_f$$

- Todo estado debe caracterizarse por una condición que siempre se cumple y que se denomina Invariante.
- Tanto el estado inicial, como los intermedios, como el final siempre cumplen esa condición.

Cálculo iterativo del factorial

- Entrada: $N \geq 0$
- Salida: $R = N!$
- Idea: Iteración ascendente:

$$(0,1) \rightarrow (1,1) \rightarrow (2,2) \rightarrow (3,6) \rightarrow \ldots \rightarrow (N, N!)$$

- Estado: Tupla de la forma (I, R) tal que $I \leq N \wedge R = I!$ (Invariante)
- Estado Inicial: $I = 0, \; R = 1$
- Estado Final $I = N$
- Transformación de estados:

$$(I, R) \rightarrow (I+1, (I+1) * R)$$

Construyendo iteraciones (1)

¿Cómo lograr computaciones iterativas?

- Una computación iterativa se caracteriza por comenzar en un estado inicial S_0, y transformar ese estado en un conjunto de estados intermedios hasta llegar a un estado final S_f:

$$S_0 \rightarrow S_1 \rightarrow S_2 \ldots \rightarrow S_f$$

- Todo estado debe caracterizarse por una condición que siempre se cumple y que se denomina Invariante.
- Tanto el estado inicial, como los intermedios, como el final siempre cumplen esa condición.

Cálculo iterativo del factorial

- Entrada: $N \geq 0$
- Salida: $R = N!$
- Idea: Iteración ascendente:

$$(0,1) \rightarrow (1,1) \rightarrow (2,2) \rightarrow (3,6) \rightarrow \ldots \rightarrow (N, N!)$$

- Estado: Tupla de la forma (I, R) tal que $I \leq N \wedge R = I!$ (Invariante)
- Estado Inicial: $I = 0, \ R = 1$
- Estado Final $I = N$
- Transformación de estados:

$$(I, R) \rightarrow (I+1, (I+1) * R)$$

Modelos y Paradigmas de Programación

Universidad del Valle

Construyendo iteraciones (1)

¿Cómo lograr computaciones iterativas?

- Una computación iterativa se caracteriza por comenzar en un estado inicial S_0, y transformar ese estado en un conjunto de estados intermedios hasta llegar a un estado final S_f:

$$S_0 \rightarrow S_1 \rightarrow S_2 \ldots \rightarrow S_f$$

- Todo estado debe caracterizarse por una condición que siempre se cumple y que se denomina Invariante.
- Tanto el estado inicial, como los intermedios, como el final siempre cumplen esa condición.

Cálculo iterativo del factorial

- Entrada: $N \geq 0$
- Salida: $R = N!$
- Idea: Iteración ascendente:

$$(0,1) \rightarrow (1,1) \rightarrow (2,2) \rightarrow (3,6) \rightarrow \ldots \rightarrow (N, N!)$$

- Estado: Tupla de la forma (I, R) tal que $I \leq N \wedge R = I!$ (Invariante)
- Estado Inicial: $I = 0,\ R = 1$
- Estado Final $I = N$
- Transformación de estados:

$$(I, R) \rightarrow (I+1, (I+1) * R)$$

Modelos y Paradigmas de Programación

Universidad del Valle

Construyendo iteraciones (1)

¿Cómo lograr computaciones iterativas?

- Una computación iterativa se caracteriza por comenzar en un estado inicial S_0, y transformar ese estado en un conjunto de estados intermedios hasta llegar a un estado final S_f:

$$S_0 \rightarrow S_1 \rightarrow S_2 \ldots \rightarrow S_f$$

- Todo estado debe caracterizarse por una condición que siempre se cumple y que se denomina Invariante.
- Tanto el estado inicial, como los intermedios, como el final siempre cumplen esa condición.

Cálculo iterativo del factorial

- Entrada: $N \geq 0$
- Salida: $R = N!$
- Idea: Iteración ascendente:

$$(0,1) \rightarrow (1,1) \rightarrow (2,2) \rightarrow (3,6) \rightarrow \ldots \rightarrow (N, N!)$$

- Estado: Tupla de la forma (I, R) tal que $I \leq N \wedge R = I!$ (Invariante)
- Estado Inicial: $I = 0, \ R = 1$
- Estado Final $I = N$
- Transformación de estados:

$$(I, R) \rightarrow (I+1, (I+1) * R)$$

Modelos y Paradigmas de Programación

Universidad del Valle

Construyendo iteraciones (2)

Factorial usando la abstracción

```
fun {Factorial N}
   {Iterar
      t(0 1)
      fun {$ t(I R)}
          I==N
      end
      fun {$ t(I R)}
          t(I+1 R*(I+1))
      end}
end
```

O más eficientemente:

```
fun {Factorial N}
   {Iterar
      t(0 1)
      fun {$ t(I R)}
          I==N
      end
      fun {$ t(I R)}
          J=I+1 in
            t(J R*J)
      end}
end
```

Construyendo iteraciones (2)

Factorial usando la abstracción

```
fun {Factorial N}
   {Iterar
      t(0 1)
      fun {$ t(I R)}
          I==N
      end
      fun {$ t(I R)}
          t(I+1 R*(I+1))
      end}
end
```

O más eficientemente:

```
fun {Factorial N}
   {Iterar
      t(0 1)
      fun {$ t(I R)}
          I==N
      end
      fun {$ t(I R)}
          J=I+1 in
            t(J R*J)
      end}
end
```

Construyendo iteraciones (3)

Otra idea para Factorial

- Entrada: $N \geq 1$
- Salida: $R = N!$
- Idea: Iteración descendente:

$$(N, N) \rightarrow (N-1, N*(N-1)) \rightarrow$$

$$(N-2, N*(N-1)*(N-2)) \ldots \rightarrow (1, N!)$$

- Estado: Tupla de la forma (I, R) tal que $I \geq 1 \wedge R = N*(N-1)*(N-2)*\ldots*(I)$ (Invariante)

- Estado Inicial: $I = N, \; R = N$
- Estado Final $I = 1$
- Transformación de estados:

$$(I, R) \rightarrow (I-1, (I-1)*R)$$

Programa resultante

```
fun {Factorial N}
   {Iterar
      t(N N)
      fun {$ t(I R)}
         I==1
      end
      fun {$ t(I R)}
         J=I-1 in
           t(J R*J)
      end}
end
```

Modelos y Paradigmas de Programación

Universidad del Valle

Construyendo iteraciones (3)

Otra idea para Factorial

- Entrada: $N \geq 1$
- Salida: $R = N!$
- Idea: Iteración descendente:

$$(N, N) \rightarrow (N-1, N*(N-1)) \rightarrow$$

$$(N-2, N*(N-1)*(N-2)) \ldots \rightarrow (1, N!)$$

- Estado: Tupla de la forma (I, R) tal que $I \geq 1 \wedge R = N*(N-1)*(N-2)*\ldots*(I)$ (Invariante)

- Estado Inicial: $I = N, \ R = N$
- Estado Final $I = 1$
- Transformación de estados:

$$(I, R) \rightarrow (I-1, (I-1)*R)$$

Programa resultante

```
fun {Factorial N}
   {Iterar
      t(N N)
      fun {$ t(I R)}
         I==1
      end
      fun {$ t(I R)}
         J=I-1 in
            t(J R*J)
      end}
end
```

Construyendo iteraciones (3)

Otra idea para Factorial

- Entrada: $N \geq 1$
- Salida: $R = N!$
- Idea: Iteración descendente:

$$(N, N) \rightarrow (N-1, N*(N-1)) \rightarrow$$

$$(N-2, N*(N-1)*(N-2)) \ldots \rightarrow (1, N!)$$

- Estado: Tupla de la forma (I, R) tal que $I \geq 1 \wedge R = N*(N-1)*(N-2)*\ldots*(I)$ (Invariante)

- Estado Inicial: $I = N,\ R = N$
- Estado Final $I = 1$
- Transformación de estados:

$$(I, R) \rightarrow (I-1, (I-1)*R)$$

Programa resultante

```
fun {Factorial N}
   {Iterar
      t(N N)
      fun {$ t(I R)}
         I==1
      end
      fun {$ t(I R)}
         J=I-1 in
            t(J R*J)
      end}
end
```

Construyendo iteraciones (3)

Otra idea para Factorial

- Entrada:$N \geq 1$
- Salida: $R = N!$
- Idea: Iteración descendente:

$$(N, N) \rightarrow (N-1, N*(N-1)) \rightarrow$$

$$(N-2, N*(N-1)*(N-2)) \ldots \rightarrow (1, N!)$$

- Estado: Tupla de la forma (I, R) tal que $I \geq 1 \wedge R = N*(N-1)*(N-2)*\ldots*(I)$ (Invariante)

- Estado Inicial: $I = N, \; R = N$
- Estado Final $I = 1$
- Transformación de estados:

$$(I, R) \rightarrow (I-1, (I-1)*R)$$

Programa resultante

```
fun {Factorial N}
   {Iterar
      t(N N)
      fun {$ t(I R)}
         I==1
      end
      fun {$ t(I R)}
         J=I-1 in
            t(J R*J)
      end}
end
```

Modelos y Paradigmas de Programación

Construyendo iteraciones (3)

Otra idea para Factorial

- Entrada:$N \geq 1$
- Salida: $R = N!$
- Idea: Iteración descendente:

$$(N, N) \rightarrow (N-1, N*(N-1)) \rightarrow$$

$$(N-2, N*(N-1)*(N-2)) \ldots \rightarrow (1, N!)$$

- Estado: Tupla de la forma (I, R) tal que $I \geq 1 \wedge R = N*(N-1)*(N-2)*\ldots*(I)$ (Invariante)

- Estado Inicial: $I = N, \; R = N$
- Estado Final $I = 1$
- Transformación de estados:

$$(I, R) \rightarrow (I-1, (I-1)*R)$$

Programa resultante

```
fun {Factorial N}
   {Iterar
      t(N N)
      fun {$ t(I R)}
         I==1
      end
      fun {$ t(I R)}
         J=I-1 in
           t(J R*J)
      end}
end
```

Construyendo iteraciones (3)

Otra idea para Factorial

- Entrada: $N \geq 1$
- Salida: $R = N!$
- Idea: Iteración descendente:

$$(N, N) \rightarrow (N-1, N*(N-1)) \rightarrow$$

$$(N-2, N*(N-1)*(N-2)) \ldots \rightarrow (1, N!)$$

- Estado: Tupla de la forma (I, R) tal que $I \geq 1 \wedge R = N*(N-1)*(N-2)*\ldots*(I)$ (Invariante)

- Estado Inicial: $I = N, \; R = N$
- Estado Final $I = 1$
- Transformación de estados:

$$(I, R) \rightarrow (I-1, (I-1)*R)$$

Programa resultante

```
fun {Factorial N}
   {Iterar
      t(N N)
      fun {$ t(I R)}
         I==1
      end
      fun {$ t(I R)}
         J=I-1 in
           t(J R*J)
      end}
end
```

Modelos y Paradigmas de Programación

Universidad del Valle

Construyendo iteraciones (3)

Otra idea para Factorial

- Entrada: $N \geq 1$
- Salida: $R = N!$
- Idea: Iteración descendente:

$$(N, N) \rightarrow (N-1, N*(N-1)) \rightarrow$$

$$(N-2, N*(N-1)*(N-2)) \ldots \rightarrow (1, N!)$$

- Estado: Tupla de la forma (I, R) tal que $I \geq 1 \wedge R = N*(N-1)*(N-2)*\ldots*(I)$ (Invariante)

- Estado Inicial: $I = N, \ R = N$
- Estado Final $I = 1$
- Transformación de estados:

$$(I, R) \rightarrow (I-1, (I-1)*R)$$

Programa resultante

```
fun {Factorial N}
   {Iterar
      t(N N)
      fun {$ t(I R)}
         I==1
      end
      fun {$ t(I R)}
         J=I-1 in
           t(J R*J)
      end}
end
```