Clase Práctica de Iteradores

Matias Barbeito

Departamento de Computación,
Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires

October 2, 2015

Hoy en Algo 2

1. Iteradores básicos
   - Problemática
   - Iteradores Unidireccionales.
2. Otros iteradores
   - Iteradores modificables
   - Iteradores bidireccionales
3. Iterando otras estructuras.
4. Iteradores que describen conjuntos
5. Iteradores como punteros virtuales
6. Bonus track
   - Programación genérica con iteradores

Interfaz Secu

- ¿Cómo hacemos para calcular la sumatoria de una secu(nat)?

$\textsc{Secu}() \rightarrow \mathit{res}\colon \mathit{secu}(\mathit{nat})$
$P = \{\texttt{true}\}$ $\quad Q = \{\mathit{res} = <>\}$

$\textsc{Agregar}(\textbf{in/out } s\colon \mathit{secu}(\mathit{nat}), \textbf{ in } \mathit{elem}\colon \alpha)$
$P = \{s = s_0\}$ $\quad Q = \{s = \mathit{elem} \bullet s_0\}$

$\textsc{Primero}(\textbf{in } s\colon \mathit{secu}(\mathit{nat})) \rightarrow \mathit{res}\colon \alpha$
$P = \{\texttt{true}\}$ $\quad Q = \{\mathit{res} = \mathrm{Prim}(s)\}$

$\textsc{Fin}(\textbf{in/out } s\colon \mathit{secu}(\mathit{nat}))$
$P = \{s = s_0 \land \neg \mathrm{Vacia}(s)\}$ $\quad Q = \{s = \mathrm{Fin}(s_0)\}$

$\textsc{Vacia}(\textbf{in } s\colon \mathit{secu}(\mathit{nat})) \rightarrow \mathit{res}\colon \mathit{bool}$
$P = \{\texttt{true}\}$ $\quad Q = \{\mathit{res} = \mathrm{Vacia}(s)\}$

$\textsc{Copiar}(\textbf{in } s\colon \mathit{secu}(\mathit{nat})) \rightarrow \mathit{res}\colon \mathit{secu}(\mathit{nat})$
$P = \{\texttt{true}\}$ $\quad Q = \{\mathit{res} = s\}$

Interfaz Secu

- ¿Y ahora?

$\textsc{Secu}() \rightarrow \mathit{res}\colon \mathit{secu}(\mathit{nat})$
$P = \{\texttt{true}\}$ $\qquad Q = \{\mathit{res} = <>\}$

$\textsc{Agregar}(\textbf{in/out}\ s\colon \mathit{secu}(\mathit{nat}), \textbf{in}\ \mathit{elem}\colon \alpha)$
$P = \{s = s_0\}$ $\qquad Q = \{s = \mathit{elem} \bullet s_0\}$

$\textsc{Primero}(\textbf{in}\ s\colon \mathit{secu}(\mathit{nat})) \rightarrow \mathit{res}\colon \alpha$
$P = \{\texttt{true}\}$ $\qquad Q = \{\mathit{res} = \mathrm{Prim}(s)\}$

$\textsc{Fin}(\textbf{in/out}\ s\colon \mathit{secu}(\mathit{nat}))$
$P = \{s = s_0 \wedge \neg\mathrm{Vacia}(s)\}$ $\qquad Q = \{s = \mathrm{Fin}(s_0)\}$

$\textsc{Vacia}(\textbf{in}\ s\colon \mathit{secu}(\mathit{nat})) \rightarrow \mathit{res}\colon \mathit{bool}$
$P = \{\texttt{true}\}$ $\qquad Q = \{\mathit{res} = \mathrm{Vacia}(s)\}$

Usando iteradores

```
Sum(in s: secu(nat)) → res: nat
    res ← 0
    it:itersecu(nat) ← CrearIt(s)
    while hayMas(it):
        res ← res + Actual(it)
        Avanzar(it)

ContarPrimos(in s: secu(nat)) → res: nat
    res ← 0
    it:itersecu(nat) ← CrearIt(s)
    while hayMas(it):
        res ← res + EsPrimo(Actual(it))
        Avanzar(it)
```

Interfaz

$\text{CREARIT}(\textbf{in } s\colon \mathit{secu}(\alpha)) \to \mathit{res}\colon \mathit{itersecu}(\alpha)$
$P = \{???\}$ $Q = \{???\}$

$\text{HAYMAS}(\textbf{in } \mathit{it}\colon \mathit{itersecu}(\alpha)) \to \mathit{res}\colon \mathit{bool}$
$P = \{???\}$ $Q = \{???\}$

$\text{ACTUAL}(\textbf{in } \mathit{it}\colon \mathit{itersecu}(\alpha)) \to \mathit{res}\colon \alpha$
$P = \{???\}$ $Q = \{???\}$

$\text{AVANZAR}(\textbf{in/out } \mathit{it}\colon \mathit{itersecu}(\alpha))$
$P = \{???\}$ $Q = \{???\}$

TAD ITERADOR UNIDIRECCIONAL(α)

TAD ITERADOR UNIDIRECCIONAL(α)

géneros itUni(α)

observadores básicos

Siguientes : itUni(α) $\longrightarrow$ secu(α)

generadores

CrearItUni : secu(α) $\longrightarrow$ itUni(α)

otras operaciones

HayMas? : itUni(α) $\longrightarrow$ bool

Actual : itUni(α) *it* $\longrightarrow$ α {HayMas?(*it*)}

Avanzar : itUni(α) *it* $\longrightarrow$ itUni(α) {HayMas?(*it*)}

axiomas

Siguientes(CrearItUni(*i*)) $\equiv$ *i*

HayMas?(*it*) $\equiv$ $\neg$Vacia?(Siguientes(*it*))

Actual(*it*) $\equiv$ Prim(Siguientes(*it*))

Avanzar(*it*) $\equiv$ CrearItUni(Fin(Siguientes(*it*)))

Fin TAD

Interfaz Reloaded

$\textsc{CrearIt}(\textbf{in } s\colon \mathit{secu}(\alpha)) \to \mathit{res}\colon \mathit{itersecu}(\alpha)$
$P = \{\texttt{true}\}$ $\quad Q = \{\mathit{res} = \text{CrearItUni}(s)\}$

$\textsc{HayMas}(\textbf{in } \mathit{it}\colon \mathit{itersecu}(\alpha)) \to \mathit{res}\colon \mathit{bool}$
$P = \{\texttt{true}\}$ $\quad Q = \{\mathit{res} = \text{HayMas?}(\mathit{it})\}$

$\textsc{Actual}(\textbf{in } \mathit{it}\colon \mathit{itersecu}(\alpha)) \to \mathit{res}\colon \alpha$
$P = \{\text{HayMas?}(\mathit{it})\}$ $\quad Q = \{\mathit{res} = \text{Actual}(\mathit{it})\}$

$\textsc{Avanzar}(\textbf{in/out } \mathit{it}\colon \mathit{itersecu}(\alpha))$
$P = \{\mathit{it} = \mathit{it}_0 \wedge \text{HayMas}(\mathit{it})\}$ $\quad Q = \{\mathit{it} = \text{Avanzar}(\mathit{it}_0)\}$

Representación

- *secu*(α) **se representa con** *puntero*(*nodo*)
- **donde** *nodo* es *tupla*⟨dato: α, prox: puntero(nodo)⟩
- *itersecu*(α) **se representa con** *puntero*(*nodo*)

$$\text{Rep} \;:\; \text{puntero(nodo)} \longrightarrow \text{bool}$$
$$\text{Rep}(p) \equiv \texttt{true} \Leftrightarrow (\exists n : \text{nat})\ \text{Finaliza}(p,\ n)$$
$$\text{Finaliza}(p,\ n) \equiv n > 0 \wedge_L (p = nil \vee_L \text{Finaliza}(p \rightarrow \text{prox},\ n-1))$$

$$\text{Abs} \;:\; \text{puntero(nodo)}\ p \longrightarrow \text{secu}(\alpha) \qquad \{\text{Rep}(p)\}$$
$$\text{Abs}(p) \equiv \textbf{if}\ p = nil\ \textbf{then}\ <>\ \textbf{else}\ p \rightarrow \text{dato} \bullet \text{Abs}(p \rightarrow \text{prox})\ \textbf{fi}$$

Representación

- *secu*(α) **se representa con** *puntero*(*nodo*)
- **donde** *nodo* es *tupla*⟨dato: α, prox: puntero(nodo)⟩
- *itersecu*(α) **se representa con** *puntero*(*nodo*)

$$\text{Rep} : \text{puntero(nodo)} \longrightarrow \text{bool}$$
$$\text{Rep}(p) \equiv \texttt{true} \Leftrightarrow (\exists n : \text{nat})\ \text{Finaliza}(p, n)$$
$$\text{Finaliza}(p, n) \equiv n > 0 \wedge_L (p = nil \vee_L \text{Finaliza}(p\rightarrow\text{prox}, n-1))$$

$$\text{Abs} : \text{puntero(nodo)}\ p \longrightarrow \text{secu}(\alpha) \qquad \{\text{Rep}(p)\}$$
$$\text{Abs}(p) \equiv \textbf{if } p = nil \textbf{ then } <> \textbf{ else } p\rightarrow\text{dato} \bullet \text{Abs}(p\rightarrow\text{prox}) \textbf{ fi}$$

$$\text{Rep\_it} : \text{puntero(nodo)} \longrightarrow \text{bool}$$
$$\text{Rep\_it}(p) \equiv \text{Rep}(p)$$

$$\text{Abs\_it} : \text{puntero(nodo)}\ p \longrightarrow \text{itUni}(\alpha) \qquad \{\text{Rep}(p)\}$$
$$\text{Abs\_it}(p) \equiv \text{CrearItUni}(\text{Abs}(p))$$

Implementación

$\text{ICREARIT}(\textbf{in } p\text{: } \textit{puntero}(\textit{nodo})) \rightarrow \textit{res}\text{: } \textit{puntero}(\textit{nodo})$

1. $\textit{res} \leftarrow p$

$\text{IHAYMAS}(\textbf{in } p\text{: } \textit{puntero}(\textit{nodo})) \rightarrow \textit{res}\text{: } \textit{bool}$

1. $\textit{res} \leftarrow p \neq \textit{nil}$

$\text{IACTUAL}(\textbf{in } p\text{: } \textit{puntero}(\textit{nodo})) \rightarrow \textit{res}\text{: } \alpha$

1. $\textit{res} \leftarrow p \rightarrow \text{dato}$

$\text{IAVANZAR}(\textbf{in/out } p\text{: } \textit{puntero}(\textit{nodo}))$

1. $p \leftarrow p \rightarrow \text{prox}$

El metapredicado “alias”

- ¿Qué ocurre con el iterador si eliminamos un elemento de la secuencia?

El metapredicado "alias"

- ¿Qué ocurre con el iterador si eliminamos un elemento de la secuencia?

- `alias` permite describir la interacción entre variables que comparten la misma posición de memoria.
- Un único parámetro ϕ de tipo booleano o predicado en LPO.
- ϕ es una relación entre algunas variables del programa.

El metapredicado "alias"

- ¿Qué ocurre con el iterador si eliminamos un elemento de la secuencia?
- `alias` permite describir la interacción entre variables que comparten la misma posición de memoria.
- Un único parámetro ϕ de tipo booleano o predicado en LPO.
- ϕ es una relación entre algunas variables del programa.
- alias(ϕ) indica que ϕ es válido de ahora en más, hasta asignación de alguna de las variables.

El metapredicado "alias"

Ejemplos

- alias($a = b$) con a, b de tipo α.
- alias(esPermutación(s, c)) con s de tipo secu(α), c de tipo conj(α).
- alias($res =_{\mathrm{obs}}$ prim(s)) con s de tipo secu(α).

Interfaz Rereloaded

$\textsc{CrearIt}(\textbf{in}\ s\colon\ \textit{secu}(\alpha)) \rightarrow \textit{res}\colon\ \textit{itersecu}(\alpha)$
$P = \{\texttt{true}\}$ $\qquad Q = \{\textit{res} = \text{CrearItUni}(s)\}$

$\textsc{HayMas}(\textbf{in}\ it\colon\ \textit{itersecu}(\alpha)) \rightarrow \textit{res}\colon\ \textit{bool}$
$P = \{\texttt{true}\}$ $\qquad Q = \{\textit{res} = \text{HayMas?}(it)\}$

$\textsc{Actual}(\textbf{in}\ it\colon\ \textit{itersecu}(\alpha)) \rightarrow \textit{res}\colon\ \alpha$
$P = \{\text{HayMas?}(it)\}$ $\qquad Q = \{\text{alias}(\textit{res} = \text{Actual}(it))\}$

$\textsc{Avanzar}(\textbf{in/out}\ it\colon\ \textit{itersecu}(\alpha))$
$P = \{it = it_0 \land \text{HayMas}(it)\}$ $\qquad Q = \{it = \text{Avanzar}(it_0)\}$

Iteradores modificables (motivación)

```
ELIMINARPARES(in/out s: secu(nat))
    it:secu(nat) ← CrearIt(s)
    while HayMas(it):
        if EsPar(Actual(it)) then Eliminar(it)
        else Avanzar(it)

DUPLICAR(in/out s: secu(nat))
    it:itersecu(nat) ← CrearIt(s)
    while HayMas(it):
        Agregar(it, Actual(it))
        Avanzar(it)
```

Iteradores modificables (interfaz)

$\textsc{Eliminar}(\textbf{in/out}\ it\colon\ itersecu(\alpha))$

$P = \{???\}$ $\qquad$ $Q = \{???\}$

$\textsc{Agregar}(\textbf{in/out}\ it\colon\ itersecu(\alpha),\ \textbf{in}\ val\colon\ \alpha) \to res\colon\ itersecu(\alpha)$

$P = \{???\}$ $\qquad$ $Q = \{???\}$

Las usuales del iterador unidireccional.

TAD ITERADOR MODIFICABLE(α)

TAD ITERADOR MODIFICABLE(α)

géneros itMod(α)

observadores básicos

$$\begin{array}{lclcl} \text{Anteriores} & : & \text{itMod}(\alpha) & \longrightarrow & \text{secu}(\alpha) \\ \text{Siguientes} & : & \text{itMod}(\alpha) & \longrightarrow & \text{secu}(\alpha) \end{array}$$

generadores

$$\text{CrearItMod} : \text{secu}(\alpha) \times \text{secu}(\alpha) \longrightarrow \text{itMod}(\alpha)$$

otras operaciones

$$\begin{array}{lclcll} \text{SecuSuby} & : & \text{itMod}(\alpha) & \longrightarrow & \text{secu}(\alpha) & \\ \text{Eliminar} & : & \text{itMod}(\alpha)\ it & \longrightarrow & \text{itMod}(\alpha) & \{\text{HayMas?}(it)\} \\ \text{Agregar} & : & \text{itMod}(\alpha) \times \alpha & \longrightarrow & \text{itMod}(\alpha) & \end{array}$$

...

axiomas

$$\begin{array}{lcl} \text{Anteriores}(\text{CrearItMod}(i,\ d)) & \equiv & i \\ \text{Siguientes}(\text{CrearItMod}(i,\ d)) & \equiv & d \\ \text{SecuSuby}(it) & \equiv & \text{Anteriores}(it)\ \&\ \text{Siguientes}(it) \\ \text{Eliminar}(it) & \equiv & \text{CrearItMod}(\text{Anteriores}(it),\ \text{Fin}(\text{Siguientes}(it))) \\ \text{Agregar}(it, a) & \equiv & \text{CrearItMod}(\text{Anteriores}(it) \circ a,\ \text{Siguientes}(it)) \end{array}$$

Fin TAD

Iteradores modificables (interfaz/aliasing)

$\textsc{Eliminar}(\textbf{in/out}\ it\colon\ itersecu(\alpha))$
$P = \{it_0 = it \land \text{HayMas?}(it)\}$ $Q = \{it = \text{Eliminar}(it_0)\}$

$\textsc{Agregar}(\textbf{in/out}\ it\colon\ itersecu(\alpha),\ \textbf{in}\ val\colon\ \alpha) \to res\colon\ itersecu(\alpha)$
$P = \{it_0 = it\}$ $Q = \{it = \text{Agregar}(it_0,\ val)\}$

$\textsc{CrearIt}(\textbf{in}\ s\colon\ secu(\alpha)) \to res\colon\ itersecu(\alpha)$
$P = \{\texttt{true}\}$
$Q = \{res =_{\text{obs}} \text{crearItMod}(<>,\ s) \land \text{alias}(\text{SecuSuby}(it) = s)\}$

Iterador Modificable (representación)

- *secu*(α) **se representa con** *estr*,
- **donde** *estr* es tupla⟨prim: *puntero*(*nodo*), ult: *puntero*(*nodo*)⟩, y
- *nodo* es *tupla*⟨prev: puntero(nodo), dato: α, prox: puntero(nodo)⟩

- *itersecu*(α) **se representa con** iterStruc
- **donde** *iterStruc* es *tupla*⟨laSecu: puntero(estr) $\times$ posicion: puntero(nodo)⟩

Nota: El rep y el abs de esta estructura es muy parecida a las que ya vimos.

Iterador Modificable (representación)

- *secu*(α) **se representa con** *estr*,
- **donde** *estr* es tupla⟨prim: *puntero*(*nodo*), ult: *puntero*(*nodo*)⟩, y
- *nodo* es *tupla*⟨prev: puntero(nodo), dato: α, prox: puntero(nodo)⟩

- *itersecu*(α) **se representa con** iterStruc
- **donde** *iterStruc* es *tupla*⟨laSecu: puntero(estr) $\times$ posicion: puntero(nodo)⟩

Nota: El rep y el abs de esta estructura es muy parecida a las que ya vimos.

Discutir la implementación

TAD Iterador Bidireccional(α)

TAD Iterador Bidireccional(α)

géneros itBi(α)

observadores básicos

$$
\begin{array}{lcll}
\text{Anteriores} & : & \text{itBi}(\alpha) & \longrightarrow \text{secu}(\alpha) \\
\text{Siguientes} & : & \text{itBi}(\alpha) & \longrightarrow \text{secu}(\alpha)
\end{array}
$$

generadores

$$
\text{CrearItBi} : \text{secu}(\alpha) \times \text{secu}(\alpha) \longrightarrow \text{itBi}(\alpha)
$$

otras operaciones

$$
\begin{array}{lclll}
\text{HayAnterior?} & : & \text{itBi}(\alpha) & \longrightarrow \text{bool} & \\
\text{Anterior} & : & \text{itBi}(\alpha)\ \mathit{it} & \longrightarrow \alpha & \{\text{HayAnterior?}(\mathit{it})\} \\
\text{Retroceder} & : & \text{itBi}(\alpha)\ \mathit{it} & \longrightarrow \text{itBi}(\alpha) & \{\text{HayAnterior?}(\mathit{it})\} \\
\text{AgregarComoAnterior} & : & \text{itBi}(\alpha) \times \alpha & \longrightarrow \text{itBi}(\alpha) &
\end{array}
$$

...

axiomas

$$
\begin{array}{lcl}
\text{HayAnterior?}(\mathit{it}) & \equiv & \neg\text{Vacia?}(\text{Anteriores}(\mathit{it})) \\
\text{Anterior}(\mathit{it}) & \equiv & \text{Ult}(\text{Anteriores}(\mathit{it})) \\
\text{Retroceder}(\mathit{it}) & \equiv & \text{CrearItBi}(\text{Com}(\text{Anteriores}(\mathit{it})), \text{Anterior}(\mathit{it}) \bullet \text{Siguientes}(\mathit{it})) \\
\text{AgregarComoAnterior}(\mathit{it}, a) & \equiv & \text{CrearItBi}(\text{Anteriores}(\mathit{it}) \circ a, \text{Siguientes}(\mathit{it}))
\end{array}
$$

Fin TAD

Iteradores bidireccionales (interfaz/aliasing)

$\textsc{CrearIt}(\textbf{in } s\colon \textit{secu}(\alpha)) \rightarrow \textit{res}\colon \textit{itersecu}(\alpha)$
$P = \{\texttt{true}\}$
$Q = \{\textit{res} =_{\text{obs}} \text{crearItBi}(<>, l) \land \text{alias}(\text{SecuSuby}(\textit{it}) = l)\}$

$\textsc{HayAnterior}(\textbf{in } \textit{it}\colon \textit{itersecu}(\alpha)) \rightarrow \textit{res}\colon \textit{bool}$
$P = \{\texttt{true}\}$ $Q = \{\textit{res} = \text{HayAnterior?}(\textit{it})\}$

$\textsc{Anterior}(\textbf{in } \textit{it}\colon \textit{itersecu}(\alpha)) \rightarrow \textit{res}\colon \alpha$
$P = \{\text{HayAnterior?}(\textit{it})\}$ $Q = \{\text{alias}(\textit{res} = \text{Anterior}(\textit{it}))\}$

$\textsc{Retroceder}(\textbf{in/out } \textit{it}\colon \textit{itersecu}(\alpha))$
$P = \{\textit{it} = \textit{it}_0 \land \text{HayAnterior?}(\textit{it})\}$
$Q = \{\textit{it} = \text{Retroceder}(\textit{it}_0)\}$

$\textsc{AgregarComoAnterior}(\textbf{in/out } \textit{it}\colon \textit{itersecu}(\alpha), \textbf{ in } a\colon \alpha)$
$P = \{\textit{it} = \textit{it}_0\}$ $Q = \{\textit{it} =_{\text{obs}} \text{AgregarComoAnterior}(\textit{it}_0, a)\}$

Iterando un conjunto

se explica con Iterador Bidireccional(α)
genero: iterconj(α)

$\textsc{CrearIt}(\textbf{in } c\colon \mathit{conj}(\alpha)) \to \mathit{res}\colon \mathit{iterconj}(\alpha)$
$P = \{\texttt{true}\}$
$Q = \{\text{alias}(\text{esPermutacion?}(\text{SecuSuby}(\mathit{res}), c)) \wedge \text{vacia?}(\text{Anteriores}(\mathit{res}))\}$

$\textsc{HaySiguiente}(\textbf{in } \mathit{it}\colon \mathit{itConj}(\alpha)) \to \mathit{res}\colon \mathit{bool}$
$P = \{\texttt{true}\}$ $Q = \{\mathit{res} =_{\text{obs}} \text{haySiguiente?}(\mathit{it})\}$

$\textsc{Siguiente}(\textbf{in } \mathit{it}\colon \mathit{itConj}(\alpha)) \to \mathit{res}\colon \alpha$
$P = \{\text{HaySiguiente?}(\mathit{it})\}$ $Q = \{\text{alias}(\mathit{res} = \text{Siguiente}(\mathit{it}))\}$

$\textsc{Avanzar}(\textbf{in/out } \mathit{it}\colon \mathit{itConj}(\alpha))$
$P = \{\mathit{it} = \mathit{it}_0 \wedge \text{HaySiguiente?}(\mathit{it})\}$
$Q = \{\mathit{it} =_{\text{obs}} \text{Avanzar}(\mathit{it}_0)\}$

Representación

- *conj*(α) **se representa con** *secu*(α)
- *iterconj*(α) **se representa con** itersecu(α)

$$\begin{array}{lcl} \text{Rep} & : & \text{secu}(\alpha) \longrightarrow \text{bool} \\ \text{Rep}(s) & \equiv & \text{Long}(s) = \#\text{Conj}(s) \end{array}$$

$$\begin{array}{lcll} \text{Abs} & : & \text{secu}(\alpha)\ s \longrightarrow \text{conj}(\alpha) & \{\text{Rep}(s)\} \\ \text{Abs}(s) & \equiv & \textbf{if } \text{vacia?}(s) \textbf{ then } \emptyset \textbf{ else } \text{Ag}(\text{prim}(s), \text{Abs}(\text{fin}(s))) \textbf{ fi} & \end{array}$$

Representación

- *conj*(α) **se representa con** *secu*(α)
- *iterconj*(α) **se representa con** itersecu(α)

$$\begin{array}{lcl} \text{Rep} & : & \text{secu}(\alpha) \longrightarrow \text{bool} \\ \text{Rep}(s) & \equiv & \text{Long}(s) = \#\text{Conj}(s) \end{array}$$

$$\begin{array}{lcll} \text{Abs} & : & \text{secu}(\alpha)\ s \longrightarrow \text{conj}(\alpha) & \{\text{Rep}(s)\} \\ \text{Abs}(s) & \equiv & \textbf{if } \text{vacia?}(s) \textbf{ then } \emptyset \textbf{ else } \text{Ag}(\text{prim}(s), \text{Abs}(\text{fin}(s))) \textbf{ fi} & \end{array}$$

$$\begin{array}{lcl} \text{Rep\_it} & : & \text{itBi}(\alpha) \longrightarrow \text{bool} \\ \text{Rep\_it}(i) & \equiv & \text{Rep}(\text{SecuSuby}(it)) \end{array}$$

$$\begin{array}{lcll} \text{Abs\_it} & : & \text{itBi}(\alpha)\ i \longrightarrow \text{itBi}(\alpha) & \{\text{Rep}(s)\} \\ \text{Abs\_it}(i) & \equiv & i & \end{array}$$

Implementación

$\text{ICREARIT}(\textbf{in } s\text{: } \textit{secu}(\alpha)) \rightarrow \textit{res}\text{: } \textit{iterconj}(\alpha)$

1. $\textit{res} \leftarrow \text{CrearIt}(s)$

$\text{IHAYMAS}(\textbf{in } i\text{: } \textit{itersecu}(\alpha)) \rightarrow \textit{res}\text{: } \textit{bool}$

1. $\textit{res} \leftarrow \text{HayMas}(s)$

$\text{IACTUAL}(\textbf{in } i\text{: } \textit{itersecu}(\alpha)) \rightarrow \textit{res}\text{: } \alpha$

1. $\textit{res} \leftarrow \text{Actual}(i)$

$\text{IAVANZAR}(\textbf{in/out } i\text{: } \textit{itersecu}(\alpha))$

1. $\text{Avanzar}(i)$

- ¿Qué ocurre si cambiamos el valor de Siguiente(*it*) desde afuera?

$\textsc{Siguiente}(\textbf{in } it\colon itConj(\alpha)) \to res\colon \alpha$	
$P = \{\text{HaySiguiente?}(it)\}$	$Q = \{\text{alias}(res = \text{Siguiente}(it))\}$

- ¿Qué ocurre si cambiamos el valor de Siguiente(it) desde afuera?

$\text{SIGUIENTE}(\textbf{in } it\text{: } itConj(\alpha)) \rightarrow res\text{: } \alpha$
$P = \{\text{HaySiguiente?}(it)\}$ $\quad Q = \{\text{alias}(res = \text{Siguiente}(it))\}$

- Hay que aclarar que algunos valores no son modificables.
- De no hacerlo, se puede romper el invariante de representación.

Iterando un diccionario

se explica con: Iterador Bidireccional(tupla(κ, ρ))
género: itDicc(κ, σ).

$\textsc{CrearIt}(\textbf{in } d\colon \mathit{dicc}(\kappa,\sigma)) \rightarrow \mathit{res}\colon \mathit{itDicc}(\kappa,\sigma)$
$P = \{\texttt{true}\}$
$Q = \{\text{alias}(\text{esPermutación}(\text{SecuSuby}(\mathit{res}),\ d)) \wedge \text{vacia?}(\text{Anteriores}(\mathit{res}))\}$

$\textsc{Siguiente}(\textbf{in } \mathit{it}\colon \mathit{itDicc}(\kappa,\sigma)) \rightarrow \mathit{res}\colon \mathit{tupla}(\kappa,\sigma)$
$P = \{\text{HaySiguiente?}(\mathit{it})\}$ $Q = \{\text{alias}(\mathit{res}, = \text{Siguiente}(\mathit{it}))\}$

$\textsc{SiguienteClave}(\textbf{in } \mathit{it}\colon \mathit{itDicc}(\kappa,\sigma)) \rightarrow \mathit{res}\colon \kappa$
$P = \{\text{HaySiguiente?}(\mathit{it})\}$ $Q = \{\text{alias}(\mathit{res} = \text{Siguiente}(\mathit{it}).\text{clave})\}$

$\textsc{SiguienteSignificado}(\textbf{in } \mathit{it}\colon \mathit{itDicc}(\kappa,\sigma)) \rightarrow \mathit{res}\colon \sigma$
$P = \{\text{HaySiguiente?}(\mathit{it})\}$ $Q = \{\text{alias}(\mathit{res} = \text{Siguiente}(\mathit{it}).\text{significado})\}$

Resumen

- Recorrido vs. Estructura.
- Iteradores unidireccionales.
- Iteradores bidireccionales.
- Iteradores modificables.
- Aliasing con alias y esAlias.
- Iteración de otras estructuras.
- Resultados no modificables.

Iteradores que describen conjuntos

- Supongamos que queremos representar un diccionario como un conjunto(tupla(κ, σ)).

- ¿Cuánto cuesta programar la siguiente funcion? ¿Tiene sentido hacerlo?

$\textsc{claves}(\textbf{in } d\text{: } dicc(\kappa, \rho)) \rightarrow res\text{: } conj(\kappa)$
$P = \{\texttt{true}\}$ $\quad\quad Q = \{res = \text{claves}(d)\}$

Iteradores que describen conjuntos

- Supongamos que queremos representar un diccionario como un conjunto(tupla(κ, σ)).

- ¿Cuánto cuesta programar la siguiente funcion? ¿Tiene sentido hacerlo?

$\textsc{Claves}(\textbf{in } d\colon \textit{dicc}(\kappa, \rho)) \to \textit{res}\colon \textit{conj}(\kappa)$
$P = \{\texttt{true}\}$ $\qquad Q = \{\textit{res} = \text{claves}(d)\}$

- A veces conviene devolver un iterador en lugar de un conjunto, o incluso dar las dos opciones:

$\textsc{Claves}(\textbf{in } d\colon \textit{dicc}(\kappa, \rho)) \to \textit{res}\colon \textit{itClaves}$
$P = \{\texttt{true}\}$
$Q = \{\text{alias}(\text{esPermutación}(\text{SecuSuby}(\textit{res}), \text{claves}(d))) \wedge \text{vacia?}(\text{Anteriores}(\textit{res}))\}$

Iteradores que describen conjuntos

- Supongamos que queremos representar un diccionario como un conjunto(tupla(κ, σ)).

- ¿Cuánto cuesta programar la siguiente funcion? ¿Tiene sentido hacerlo?

$\textsc{Claves}(\textbf{in } d\colon \mathit{dicc}(\kappa, \rho)) \to \mathit{res}\colon \mathit{conj}(\kappa)$	
$P = \{\texttt{true}\}$	$Q = \{\mathit{res} = \text{claves}(d)\}$

- A veces conviene devolver un iterador en lugar de un conjunto, o incluso dar las dos opciones:

$\textsc{Claves}(\textbf{in } d\colon \mathit{dicc}(\kappa, \rho)) \to \mathit{res}\colon \mathit{itClaves}$
$P = \{\texttt{true}\}$
$Q = \{\text{alias}(\text{esPermutación}(\text{SecuSuby}(\mathit{res}), \text{claves}(d))) \wedge \text{vacia?}(\text{Anteriores}(\mathit{res}))\}$

- Ejercicio: completar la definicion del modulo itClaves.

Iteradores como punteros virtuales

Queremos diseñar una secuencia con los siguientes requerimientos de complejidad temporal:

- Prim, Ultimo, Comienzo y Fin en $O(1)$.
- AgregarAdelante y AgregarAtras en $O(n)$ donde n es la longitud de la secuencia.
- Maximo, Minimo, QuitarMaximo y QuitarMinimo en $O(1)$.

Actualmente poseemos un modulo Lista que implementa una secu con Prim, Ultimo, Comienzo, Fin, AgregarAdelante y AgregarAtras en $O(1)$, y que tiene iteradores bidireccionales modificables (ver Apunte).

Programación genérica con iteradores

- ¿Cómo seria el algoritmo para sumar todos los números de una secuencia con iteradores?
- ¿Cómo seria el algoritmo para sumar todos los números de un conjunto con iteradores?
- ¿Cómo seria el algoritmo para sumar todos los números de un árbol con iteradores?
- ¿Cómo seria el algoritmo para sumar todos los números de un ... con iteradores?

Solución (*à la C++*)

Parámetros formales (ojo!, de implementación)

generos α, *it*(α)

operaciones

$\textsc{HayMas}$(**in** *it*: *it*(α)) $\rightarrow$ *res*: *bool*

$\textsc{Actual}$(**in** *it*: *it*(α)) $\rightarrow$ *res*: α

$\textsc{Avanzar}$(**in/out** *it*: *it*(α))

$\bullet + \bullet$(**in** *x*: α, **in** *y*: α) $\rightarrow$ *res*: α

$\textsc{Sum}$(**in** *it*: *it*(α)) $\rightarrow$ *res*: α

1. **while** HayMas(*it*):
2. *res* $\leftarrow$ *res* + Actual(*it*)
3. Avanzar(*it*)