

Uma Introdução ao Refinamento de Dados

J.N. Oliveira

5 de Maio de 2004

Processo de Desenvolvimento de software de MF

- **Especificação Formal** — “o que” o sistema de software pretendido deve fazer
- **Implementação** — código-máquina produzido instruindo o hardware sobre “como” fazê-lo

Em geral, há mais do que uma maneira de uma dada máquina concretizar “o que” o especificador tinha em mente:

- A relação entre especificações e implementações é de **uma para muitas**
- As especificações são mais **abstractas** do que as implementações.

Recordemos...

Núcleo

$$\ker R \stackrel{\text{def}}{=} R^\circ \cdot R$$

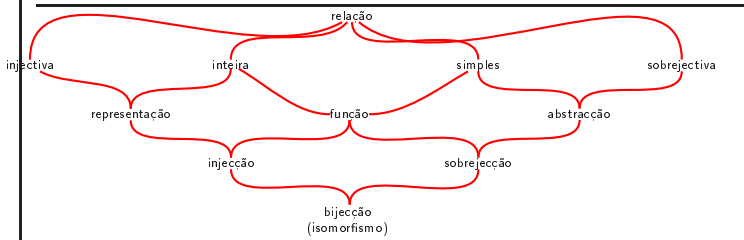
Imagem (o seu dual)

$$\text{img } R \stackrel{\text{def}}{=} \ker(R^\circ)$$

Taxonomia

	Reflexiva	Co-reflexiva
$\ker R$	inteira R	injectiva R
$\text{img } R$	sobrejectiva R	simples R

Recordemos...



Refinamento de Dados

Princípio da **abstracção de dados**: A é uma abstracção de B quando

- Existir uma **abstracção** sobrejectiva $A \xleftarrow{F} B$:

$$\text{img } F = \text{id} \quad (1)$$

F é, portanto, **simples** mas possivelmente parcial.

- Qualquer sub-relacção **inteira**, R , de F° diz-se uma **representacção** de F . Assim, $R \subseteq F^\circ$.

Relações de Representacção

- Logo, R é **injectiva**, uma vez que $\text{ker } R \subseteq \text{ker } F^\circ$ e $\text{ker } F^\circ = \text{img } F = \text{id}$.
- Deste modo, dois valores abstractos diferentes $a, a' \in A$ nunca são confundidos no processo de representacção.
- Resumindo, $\text{ker } R = \text{id}$ porque $\text{id} \subseteq \text{ker } R \subseteq \text{id}$ (R é inteira).
- Segue-se que R é uma **inversa à direita** de F , i.e.

$$F \cdot R = \text{id} \quad (2)$$

Prova-se isto por inclusão circular

$$F \cdot R \subseteq \text{id} \subseteq F \cdot R$$

no próximo diapositivo.

Invertibilidade à Direita

$$\begin{aligned} & \text{TRUE} \\ \equiv & \quad \{ \text{assume-se } R \subseteq F^\circ \} \\ & R \subseteq F^\circ \wedge R \subseteq F^\circ \\ \Rightarrow & \quad \{ (F \cdot) \text{ e } (R^\circ \cdot) \text{ são monótonos} \} \\ & F \cdot R \subseteq F \cdot F^\circ \wedge R^\circ \cdot R \subseteq R^\circ \cdot F^\circ \\ \equiv & \quad \{ \text{img } F = id, \text{ ker } R = id \text{ e conversos} \} \\ & F \cdot R \subseteq id \wedge id \subseteq F \cdot R \\ \equiv & \quad \{ \text{inclusão circular} \} \\ & F \cdot R = id \end{aligned}$$

Inequações de Refinamento

$$A \begin{array}{c} \xrightarrow{R} \\ \leq \\ \xleftarrow{F} \end{array} B \quad \text{tais que} \quad F \cdot R = id_A$$

Esta inequação admite várias interpretações informais:

- A é “menor” que B
- B pode “representar” A
- B é “abstraído” por A
- A é “implementado” por B
- B é um refinamento de A (“refina” A)

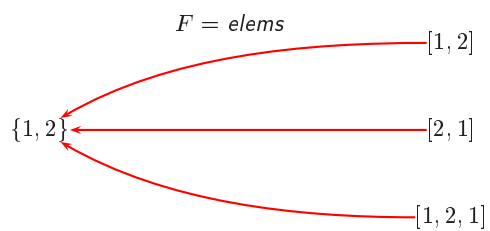
Equações de Refinamento

Isomorfismos : $A \begin{matrix} \xrightarrow{r} \\ \cong \\ \xleftarrow{f} \end{matrix} B$ tais que $r = f^\circ$

$$\begin{aligned} r &= f^\circ \\ \equiv & \{ \text{adicionando variáveis} \} \\ b r a &\equiv b f^\circ a \\ \equiv & \{ \text{funções e conversos} \} \\ b = r a &\equiv f b = a \end{aligned}$$

Exemplo

Representando **conjuntos** finitos através de **listas** finitas:



Das várias $R \subseteq F^\circ$, podemos escolher a seguinte:

Representação Relacional

```
Listifica : set of nat -> seq of nat
Listifica(s) ==
  if s = {} then []
  else let e in set s
        in [e] ^ Listifica(s \ {e});
```

Intuitivamente,

$$\text{rng Listifica} = \llbracket \text{semRepetidos} \rrbracket$$

onde

```
semRepetidos(s) == card elems s = len s
```

Representação Funcional

```
listifica : set of nat -> seq of nat
listifica(s) ==
  if s = {} then []
  else let e = minset(s)
        in [e] ^ listifica(s \ {e});
```

Intuitivamente,

$$\text{rng listifica} = \llbracket \text{EstaOrdenado} \rrbracket \cdot \llbracket \text{semRepetidos} \rrbracket$$

Invariantes Concretos

- Sempre que

$$A \begin{array}{c} \xrightarrow{R} \\ \leq \\ \xleftarrow{F} \end{array} B \quad \text{tais que } R \subseteq F^\circ \text{ e } \text{rng } R = \llbracket \phi \rrbracket$$

dizemos que ϕ é o **invariante concreto** induzido por R .

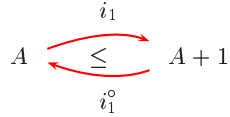
- Caso R seja uma função, e porque é sempre injectiva, temos

$$A \cong B_\phi$$

onde B_ϕ denota o subconjunto de B que satisfaz o invariante concreto ϕ .

Exemplo de abstracção parcial

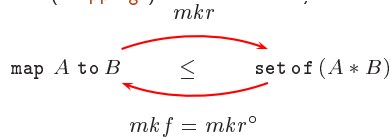
Todo o elemento do tipo de dados A pode ser representado por um "apontador":



- A **simplicidade** da abstracção é assegurada por um facto conhecido: o converso duma relação injectiva é simples.
- **Invariante** concreto: $\phi = [\text{TRUE}, \text{FALSE}]$

Outra abstracção parcial

Correspondências (**mappings**) finitas são relações finitas (simples):



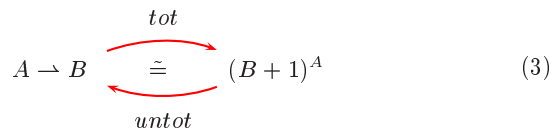
VDM-SL:

```
mkr : map A to B -> set of (A * B)
mkr(f) == { mk_(a,f(a)) | a in set dom f };

mkf : set of (A * B) -> map A to B
mkf(r) == { p.#1 |-> p.#2 | p in set r }
pre isSimple(r);
```

(Adivinhe o invariante concreto.)

Uma abstracção iso fundamental



em que, para os tipos A , B e $\text{Just}B :: \text{value}:B$,

```
tot : map A to B -> A -> [JustB]
tot(sigma)(a) ==
  if a in set dom(sigma) then mk_JustB(sigma(a)) else nil;

untot : (A -> [JustB]) -> map A to B
untot(f) == { a |-> b | a: A, b: B & f(a) = mk_JustB(b) };
```

$untot = (i_1^\circ \cdot)$ "pointfree"

Verificado a seguir:

$$\begin{aligned}
 & untot\ f = i_1^\circ \cdot f \\
 \equiv & \quad \{ \text{relações como conjuntos por compreensão} \} \\
 & untot\ f = \{ (b, a) \mid a \in A, b \in B : b(i_1^\circ \cdot f)a \} \\
 \equiv & \quad \{ \text{usando a regra } b(f^\circ \cdot R \cdot g)a \equiv (f\ b)R(g\ a) \} \\
 & untot\ f = \{ (b, a) \mid a \in A, b \in B : i_1\ b = f\ a \} \\
 \equiv & \quad \{ \text{notação VDM-SL} \} \\
 & untot\ f = \{ a \mid \rightarrow b \mid a:A, b:B \ \& \ f(a)=mk_JustB(b) \}
 \end{aligned}$$

Consequência de $tot/untot$:

$$A^1 \begin{array}{c} \xrightarrow{\cong} \\ \xleftarrow{\cong} \end{array} A$$

estende a funções parciais:

$$1 \dashrightarrow A \begin{array}{c} \xrightarrow{f^\circ} \\ \xleftarrow{f} \end{array} A + 1 \quad (\text{adivinha } f \text{ e } f^\circ).$$

Ou seja, a correspondência "singular" é uma estrutura de "apontador" disfarçada.

Propriedades de \leq :

Reflexividade

$$A \begin{array}{c} \xrightarrow{id} \\ \xleftarrow{id} \end{array} A \quad \text{cf. } id \cdot id = id$$

Transitividade

$$A \begin{array}{c} \xrightarrow{R} \\ \xleftarrow{F} \end{array} B \wedge B \begin{array}{c} \xrightarrow{S} \\ \xleftarrow{G} \end{array} C \Rightarrow A \begin{array}{c} \xrightarrow{S \cdot R} \\ \xleftarrow{F \cdot G} \end{array} C$$

Prova da transitividade

- Composição preserva simplicidade e sobrejectividade:

$$\begin{aligned}
 & \text{img } (F \cdot G) = id \\
 \equiv & \quad \{ \text{expandindo e conversos} \} \\
 & F \cdot (\text{img } G) \cdot F^\circ = id \\
 \equiv & \quad \{ G \text{ é simples e sobrejectiva} \} \\
 & \text{img } F = id \\
 \equiv & \quad \{ F \text{ é simples e sobrejectiva} \} \\
 & id = id
 \end{aligned}$$

- Verifica-se $S \cdot R \subseteq (F \cdot G)^\circ$ por monotonia.

Refinamento Estruturado

$$\begin{array}{ccc}
 A & \begin{array}{c} \xrightarrow{R} \\ \leq \\ \xleftarrow{F} \end{array} & B \quad \Rightarrow \quad F A \quad \begin{array}{c} \xrightarrow{F R} \\ \leq \\ \xleftarrow{F F} \end{array} \quad F B
 \end{array}$$

onde F é um relacionador (functor) arbitrário:

$$\begin{aligned}
 & (F F) \cdot (F R) \\
 = & \quad \{ \text{os funtores comutam com a composição} \} \\
 & F (F \cdot R) \\
 = & \quad \{ R \text{ é inversa à direita de } F \} \\
 & F id \\
 = & \quad \{ \text{os funtores comutam com } id \} \\
 & id
 \end{aligned}$$

portanto $F R$ é inversa à direita de $F f$. Evidentemente, este resultado estende-se a bi-funtores.

Refinando conjuntos finitos (I)

$$\mathcal{P}A \overset{\cong}{\rightleftarrows} A \rightarrow 1$$

Cálculo:

$$\begin{aligned} & A \rightarrow 1 \\ \cong & \quad \{ \text{tot é representação} \} \\ & (1 + 1)^A \\ \cong & \quad \{ \text{básico} \} \\ & 2^A \\ \cong & \quad \{ 2^A \text{ é isomorfo a } \mathcal{P}A \} \\ & \mathcal{P}A \end{aligned}$$

Refinando conjuntos finitos (Ia)

$$\text{set of } A \overset{\text{set2fm}}{\overset{\cong}{\rightleftarrows}} \text{map } A \text{ to Nil} \underset{\text{dom}}{\rightleftarrows}$$

VDM-SL

```
set2fm : set of A -> map A to Nil
set2fm(s) == { a |> nil | a in set s };
```

Pointfree

$$\text{set2fm} \stackrel{\text{def}}{=} (!.)$$

Invertibilidade à Direita

Cálculo:

$$\begin{aligned}
 & dom \cdot set2fm = id \\
 \equiv & \quad \{ \} \\
 & dom (set2fm s) = s \\
 \equiv & \quad \{ \} \\
 & dom (! \cdot s) = s \\
 \equiv & \quad \{ ! \text{ é uma função, } dom (f \cdot R) = dom R \} \\
 & dom s = s \\
 \equiv & \quad \{ s \text{ é co-reflexiva} \} \\
 & s = s
 \end{aligned}$$

Refinando conjuntos finitos (II)

Lista (cf. exemplo atrás):

$$\begin{array}{ccc}
 \text{set of } A & \xrightarrow{\leq} & \text{seq of } A \\
 & \xleftarrow{\text{elems}} &
 \end{array}$$

Índice A :

$$\begin{array}{ccc}
 \text{set of } A & \xrightarrow{\leq} & \text{map nat to } A \\
 & \xleftarrow{\text{rng}} &
 \end{array}$$

Refinando conjuntos finitos (III)

Classificar A por B ($B \supset \{ \}$):

$$\begin{array}{ccc}
 \text{set of } A & \xrightarrow{\leq} & \text{map } A \text{ to } B \\
 & \xleftarrow{\text{dom}} &
 \end{array}$$

Quantificar A ("multisets"):

$$\begin{array}{ccc}
 \text{set of } A & \xrightarrow{\leq} & \text{map } A \text{ to nat} \\
 & \xleftarrow{\text{dom}} &
 \end{array}$$

Refinando correspondências (I)

```
JustB::value:B;
JustC::value:C;
BorC = JustB | JustC ;
```

$$\text{map } (BorC) \text{ to } A \quad \cong \quad (\text{map } B \text{ to } A) \times (\text{map } C \text{ to } A)$$

peither

```
peither: (map B to A) * (map C to A) -> map BorC to A
peither(m,n) == { mk_JustB(b) |-> m(b) | b in set dom m} union
                { mk_JustC(c) |-> n(c) | c in set dom n};
```

NB: uma regra de representação "1FN"

Refinando correspondências (Ia)

$$(B + C) \rightarrow A \quad \cong \quad (B \rightarrow A) \times (C \rightarrow A)$$

peither

unpeither

onde

$$\text{peither}(\sigma, \tau) = [\sigma, \tau]$$

onde $[R, S] = (R \cdot i_1^\circ) \cup (S \cdot i_2^\circ)$, i.e.

$$\text{peither} = \cup \cdot ((\cdot i_1^\circ) \times (\cdot i_2^\circ))$$

Bi-functor correspondência

- Notem-se os $(\cdot i_1^\circ)$ s, $(i_1 \cdot)$ s, etc.
- Em geral, para f injectiva e qualquer g , define-se o bi-functor

$$f \mapsto g \stackrel{\text{def}}{=} (g \cdot) \cdot (\cdot f^\circ)$$

ou seja,

$$(f \mapsto g)\sigma = g \cdot \sigma \cdot f^\circ$$

- Assim, podíamos ter escrito p.e.

$$peither = \cup \cdot ((i_1 \mapsto id) \times (i_2 \mapsto id))$$

Refinando correspondências (II)

$$A \mapsto (B + C) \begin{array}{c} \xrightarrow{\text{uncojoin}} \\ \leq \\ \xleftarrow{\text{cojoin}} \end{array} (A \mapsto B) \times (A \mapsto C)$$

onde

$$\text{cojoin} = \cup \cdot ((i_1 \cdot) \times (i_2 \cdot))$$

NB: *cojoin* é parcial uma vez que a união de duas funções parciais nem sempre é uma função parcial.

Refinando correspondências (IIa)

Note-se a função de representação:

```
uncojoin : map A to BorC -> (map A to B) * (map A to C)
uncojoin(f) ==
  mk_( { a |-> f(a).value
        | a in set dom f & is_JustB(f(a)) },
        { a |-> f(a).value
        | a in set dom f & is_JustC(f(a)) }
  );
```

Refinando correspondências (III)

$$\text{map } A \text{ to } B * C \quad \leq \quad (\text{map } A \text{ to } B) \times (\text{map } A \text{ to } C)$$

\bowtie

onde (escrevendo join em vez de \bowtie)

```
join : (map A to B) * (map A to C) -> map A to (B * C)
join(m,n) == { a |-> mk_(m(a),n(a))
              | a in set dom m inter dom n };
```

Correspondências (IIIa)

$$A \rightarrow B \times C \quad \leq \quad (A \rightarrow B) \times (A \rightarrow C)$$

\bowtie

onde

$$\sigma \bowtie \tau \stackrel{\text{def}}{=} \langle \sigma, \tau \rangle$$

onde $\langle R, S \rangle \stackrel{\text{def}}{=} (\pi_1^\circ \cdot R) \cap (\pi_2^\circ \cdot S)$. Uma inversa à direita de *join* é

$$\text{unjoin} \stackrel{\text{def}}{=} \langle \text{id} \rightarrow \pi_1, \text{id} \rightarrow \pi_2 \rangle$$

Correspondências (IV)

Como estender

$$B^{C \times A} \begin{array}{c} \xrightarrow{\text{curry}} \\ \cong \\ \xleftarrow{\text{uncurry}} \end{array} (B^A)^C$$

a funções parciais? Caso $B := B + 1$

$$\begin{aligned} (B + 1)^{C \times A} &\cong ((B + 1)^A)^C \\ &\equiv \quad \{ \text{i.e.} \} \\ (C \times A) \rightarrow B &\cong (A \rightarrow B)^C \end{aligned}$$

Correspondências (IVa)

Em geral:

$$(C \times A) \rightarrow B \begin{array}{c} \xrightarrow{pcurry} \\ \leq \\ \xleftarrow{uncurry} \end{array} C \rightarrow (A \rightarrow B)$$

```
uncurry : map C to (map A to B) -> map (C * A) to B
uncurry(f) ==
  merge { let g=f(a)
          in { mk_(a,b) |-> g(b) | b in set dom g }
          | a in set dom f };
```

Correspondências (IVb)

```
pcurry : map (C * A) to B -> map C to (map A to B)
pcurry(f) ==
  let y = { x.#1 | x in set dom f }
  in { a |-> { p.#2 |-> f(p)
             | p in set dom f & p.#1=a }
      | a in set y };
```

Transpondo relações

Seja $B := 2$ no isomorfismo *curry/uncurry* obtendo-se

$$\mathcal{P}(A \times C) \begin{array}{c} \xrightarrow{\Lambda} \\ \cong \\ \xleftarrow{\Lambda^\circ} \end{array} (\mathcal{P}A)^C$$

onde

$$f = \Lambda R \equiv R = \in \cdot f \quad (4)$$

sendo $A \xleftarrow{\in} \mathcal{P}A$ a relação de pertença.

Transpondo relações finitas

$$\text{set of } (C * A) \leq \text{map } C \text{ to set of } A$$

$\xrightarrow{\text{collect}}$
 $\xleftarrow{\text{discollect}}$

```

collect : set of (C * A) -> map C to set of A
collect(r) == { c |-> { q.#2 | q in set r & c=q.#1 }
                | c in set { p.#1 | p in set r } };

discollect : map C to set of A -> set of (C * A)
discollect(f) == dunion { { mk_(c,a) | a in set f(c) }
                        | c in set dom f };
    
```

Correspondências (V)

Por fim, embora importante unjoin

$$A \rightarrow D \times (B \rightarrow C) \leq_{\mathbb{N}_n} (A \rightarrow D) \times ((A \times B) \rightarrow C) \quad (5)$$

onde

$$\mathbb{N}_n \stackrel{\text{def}}{=} \mathbb{N} \cdot \langle \pi_1, \dagger \cdot ((id \rightarrow \emptyset) \times pcurry) \rangle \quad (6)$$

e

$$\text{unjoin} \stackrel{\text{def}}{=} (id \times \text{unpcurry}) \cdot \text{unjoin}$$

Recursividade

Como refinar modelos **recursivos** do VDM-SL tais como p.e.

```
SF :: D: map Id to Nodo; -- SF significa sistema de ficheiros
Nodo = Ficheiro | SF; -- um Nodo pode ser um ficheiro
-- ou uma pasta
Id = seq of char; -- identificador dum Nodo
Ficheiro :: F: seq of token -- ficheiros sequenciais
```

ou seja, $FS = \mu F$ para $F X = Id \rightarrow (File + X)$:

$$\mu F \begin{array}{c} \xrightarrow{\text{out}} \\ \cong \\ \xleftarrow{\text{in}} \end{array} Id \rightarrow (File + \mu F)$$

Recursividade

ou...

```
ArvDec :: questao: OQue
         resposta: map Resposta to ArvDec
OQue = seq of char;
Resposta = seq of char;
```

ou seja, $ArvDec = \mu F$ em

$$ArvDec \cong OQue \times (Resposta \rightarrow ArvDec)$$

onde $F X = OQue \times (Resposta \rightarrow X)$

“Remoção” de Recursão

Dado

$$\mu F \begin{array}{c} \xrightarrow{\text{out}} \\ \cong \\ \xleftarrow{\text{in}} \end{array} F \mu F$$

temos

$$\mu F \begin{array}{c} \xrightarrow{\text{out}} \\ \leq \\ \xleftarrow{\text{in}} \end{array} (K \rightarrow F K) \times K \quad (7)$$

para K domínio de “apontadores” tais que $K \cong \mathbb{N}$.

Função de Abstracção

- O papel principal na representação é desempenhado por uma co-álgebra-F (parcial) $F K \xleftarrow{\sigma} K$, assumida como um segmento (finito) de “memória linear”, uma “heap” ou uma “base de dados”.
- \bar{F} (a transposta da abstracção F) é do tipo $(K \rightarrow F K) \rightarrow K \rightarrow \mu F$ e pode-se construir o hilomorfismo

$$\begin{array}{ccc}
 \mu F & \xleftarrow{\bar{F}\sigma} & K \\
 \text{in} \uparrow & & \downarrow \sigma \\
 F(\mu F) & \xleftarrow{F(\bar{F}\sigma)} & F K
 \end{array}
 \quad \bar{F}\sigma = \text{in} \cdot F(\bar{F}\sigma) \cdot \sigma$$

Parcialidade da implementação

$F(\sigma, k) = (\bar{F}\sigma)k$ será indefinida quando

- $k \notin \text{dom } \sigma$
- σ não é “fechado” sobre si (ver em baixo)
- σ não é bem-fundado (ver em baixo)

Temos assim, o invariante concreto

$$\phi(\sigma, k) \stackrel{\text{def}}{=} k \in \text{dom } \sigma \wedge (\text{closed } \sigma) \wedge (\text{wellf } \sigma)$$

De modo a definir $\text{closed } \sigma$ e $\text{wellf } \sigma$, precisamos da relação de **acessibilidade** de σ , \prec_σ (próximo diapositivo).

Acessibilidade e pertinência

Relação de **acessibilidade** para σ :

$$K \xleftarrow{\sigma} K$$

$$\xleftarrow{\sigma} \stackrel{\text{def}}{=} \in_F \cdot \sigma$$

onde $K \xleftarrow{\in_F} FK$ estende $K \xleftarrow{\in} PK$ indutivamente sobre funtores **polinomiais**, como se segue:

$$\begin{aligned} \in_{\mathcal{P}} &\stackrel{\text{def}}{=} \in \\ \in_G &\stackrel{\text{def}}{=} \perp \\ \in_{\lambda X.X} &\stackrel{\text{def}}{=} id \\ \in_{F \times G} &\stackrel{\text{def}}{=} (\in_F \cdot \pi_1) \cup (\in_G \cdot \pi_2) \\ \in_{F+G} &\stackrel{\text{def}}{=} [\in_F, \in_G] \end{aligned}$$

Exemplo

Seja $F X = 1 + A \times X$. Então,

$$\begin{aligned} &\in_{1+A \times X} \\ = &\quad \{ \in \text{ para o bi-functor do co-produto } \} \\ &[\in_1, \in_{A \times X}] \\ = &\quad \{ \in \text{ para bi-funtores constantes e de produto } \} \\ &[\perp, (\in_A \cdot \pi_1) \cup (\in_{\lambda X.X} \cdot \pi_2)] \\ = &\quad \{ \in \text{ para o functor identidade e constante} \} \\ &[\perp, (\perp \cdot \pi_1) \cup (id \cdot \pi_2)] \\ = &\quad \{ \perp \text{ e } [R, S] = (R \cdot i_1^\circ) \cup (S \cdot i_2^\circ) \} \\ &\pi_2 \cdot i_2^\circ \end{aligned}$$

Exemplo (“pointfree”)

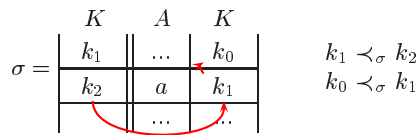
$$\begin{aligned}
 & k \in_{1+A \times X} x \\
 \equiv & \quad \{ \text{cálculo acima} \} \\
 & k(\pi_2 \cdot i_2^o)x \\
 \equiv & \quad \{ \text{composição relacional} \} \\
 & k(\pi_2)(a, k') \wedge x = i_2(a, k') \\
 \equiv & \quad \{ \text{trivial} \} \\
 & x = i_2(a, k') \wedge k = k' \\
 \equiv & \quad \{ \text{trivial} \} \\
 & x = i_2(a, k)
 \end{aligned}$$

(Exemplo de) Acessibilidade

Acessibilidade dum apontador no caso duma **heap** “linear”
 $(1 + A \times K) \xrightarrow{\sigma} K$:

$$k_1 \prec_{\sigma} k_2 \equiv k_2 \in \text{dom } \sigma \wedge (\sigma k_2) = i_2(a, k_1)$$

Desenhando:



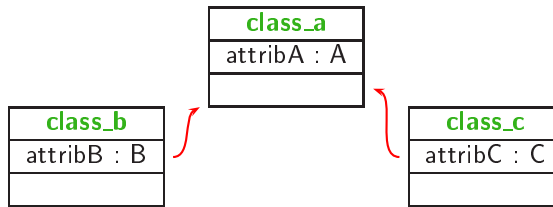
Fecho e Boa-formação

Seja \prec_{σ}^+ denotando o fecho transitivo de \prec_{σ} . Então,

- σ *fechado* = $\text{rng } \prec_{\sigma}^+ \subseteq \text{dom } \sigma$ i.e., todos os k acessíveis estão definidos.
- σ *bem – fundado* = $(\prec_{\sigma}^+) \cap \text{id} = \perp$, i.e., \prec_{σ}^+ é irreflexiva (sem ciclos)

O.O. Implementação de Dados

UML:

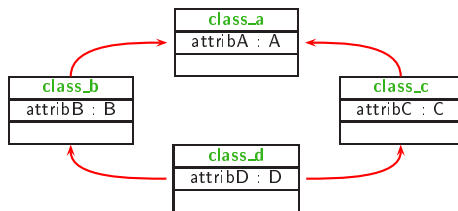


Modelo formal: $K \rightarrow Estrutura$ onde

$$\begin{aligned} Estrutura &= A + A \times B + A \times C \\ &\cong A \times (1 + B + C) \end{aligned}$$

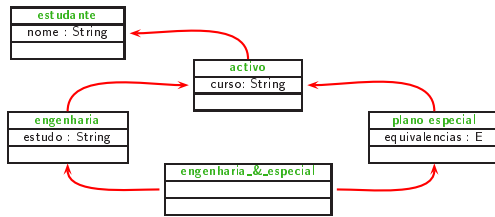
$$K \rightarrow (A + A \times B)$$

Herança múltipla



$$K \rightarrow A \times (1 + B + C + B \times C \times D)$$

Exemplo



$$K \rightarrow A \times (1 + B + C + B \times C)$$