Uma Introdução ao Refinamento de Dados

Métodos Formais de Programação II, 2003/0

J.N. Oliveira

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

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

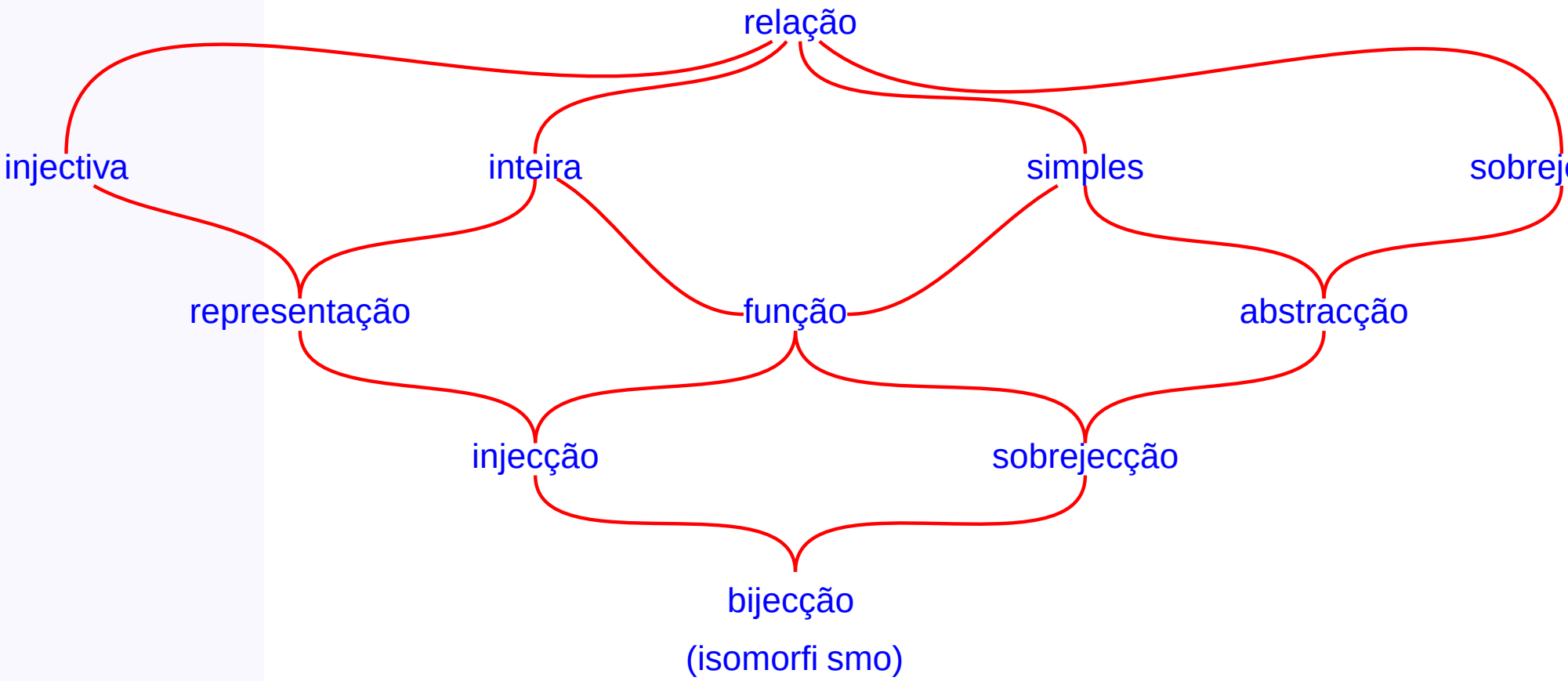
Imagem (o seu dual)

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

Taxonomia

	Reflexiva	Co-reflexiva
ker R	inteira R	injectiva R
img R	sobrejectiva R	simples R

Recordemos. . .


relação
injectiva
inteira
simples
sobrej
representação
função
abstracção
injecção
sobrejecção
bijecção
(isomorfi smo)

Refinamento de Dados

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

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

$$img\, F \;=\; id \tag{1}$$

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

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

Relações de Representação

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

$$F \cdot R \quad = \quad id \tag{2}$$

Prova-se isto por inclusão circular

$$F \cdot R \subseteq id \subseteq F \cdot R$$

Invertibilidade à Direita

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

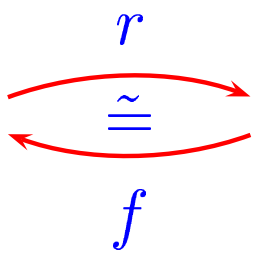
Inequações de Refinamento

$$A \underset{F}{\overset{R}{\leq}} B \quad \text{tais que} \quad F \cdot R = id_A$$

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)

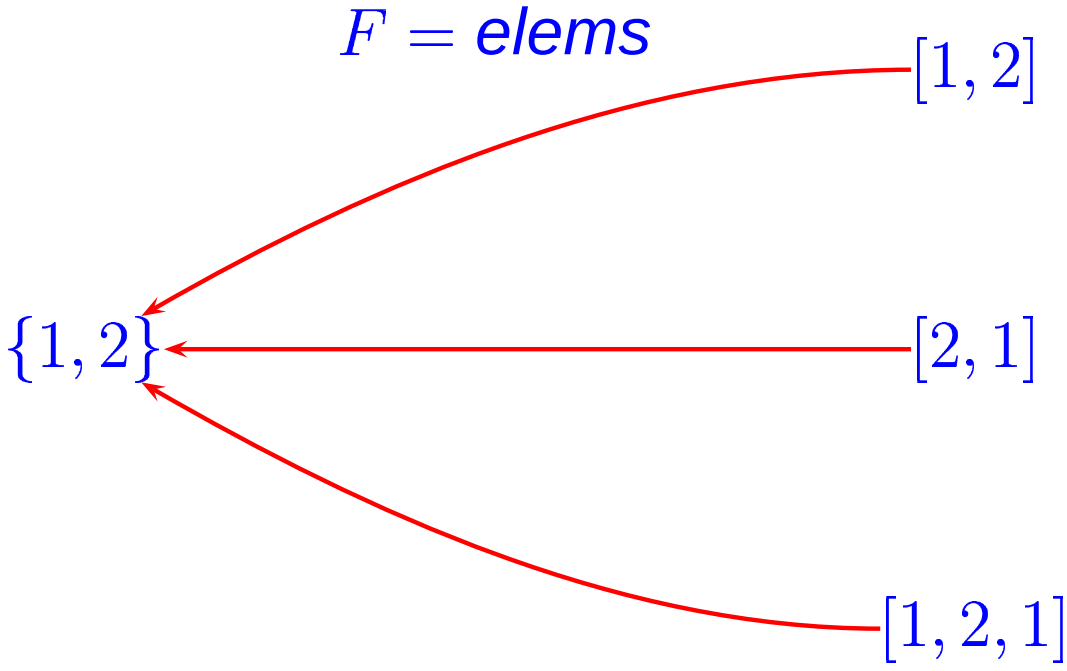
Equações de Refinamento

$Isomorfismos$:  tais que $r = f^\circ$

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

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,

$$\mathit{rng}\ Listifica = [\![semRepetidos]\!]$$

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,

$$\mathsf{rng}\, listifica = [\![EstaOrdenado]\!] \cdot [\![semRepetidos]\!]$$

Invariantes Concretos

- Sempre que

$$A \underset{F}{\overset{R}{\leq}} B \qquad \text{tais que } R \subseteq F^{\circ} \text{ e } \mathit{rng}\, R = [\![\phi]\!]$$

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

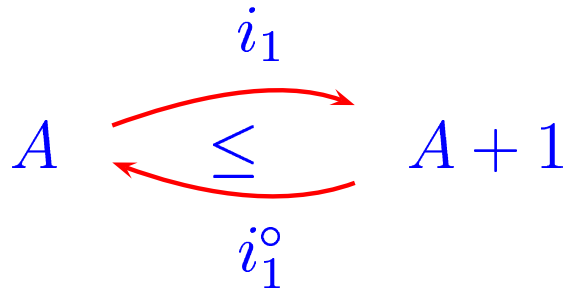
- Caso R seja uma função, e porque é sempre injectiva, temos

$$A \quad \tilde{=} \quad B_{\phi}$$

onde B_{ϕ} denota o subconjunto de B que satisfaz o invariante concreto ϕ.

Exemplo de abstracção parcial

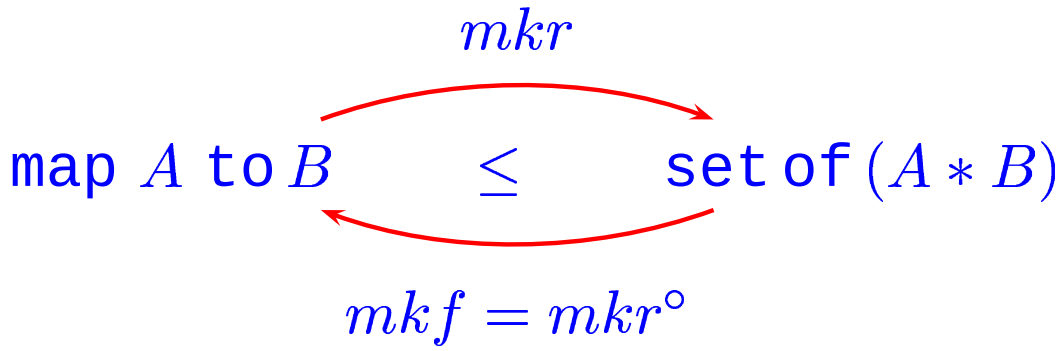
Todo o elemento do tipo de dados A pode ser representado por um "apontador":



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

Outra abstracção parcial

Correspondências (mappings) fi nitas são relações fi nitas (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

$$A \rightharpoonup B \quad \underset{untot}{\overset{tot}{\cong}} \quad (B+1)^A \tag{3}$$

em que, para os tipos A, B e `JustB::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:

$$untot\ f = i_1^\circ \cdot f$$

$\equiv$ { relações como conjuntos por compreensã

$$untot\ f = \{(b, a) \mid a \in A, b \in B : b(i_1^\circ \cdot f)a\}$$

$\equiv$ { usando a regra $b(f^\circ \cdot R \cdot g)a \equiv (f\ b)R(g\ a)$

$$untot\ f = \{(b, a) \mid a \in A, b \in B : i_1\ b = f\ a\}$$

$\equiv$ { notação VDM-SL}

$$untot\ f = \texttt{\{a|->b|a:A,b:B \& f(a)=mk\_JustB(b)\}}$$

Consequência de $tot/untot$:

$$A^1 \underset{}{\overset{\simeq}{\rightleftarrows}} A$$

estende a funções parciais:

$$1 \rightharpoonup A \underset{f}{\overset{f^\circ}{\rightleftarrows}} A + 1$$

(adivinhe f e f°).

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

Propriedades de $\leq$:

Reflexividade

$$A \underset{id}{\overset{id}{\leq}} A \quad \textit{cf.} \quad id \cdot id = id$$

Transitividade

$$A \underset{F}{\overset{R}{\leq}} B \;\wedge\; B \underset{G}{\overset{S}{\leq}} C \;\Rightarrow\; A \underset{F \cdot G}{\overset{S \cdot R}{\leq}} C$$

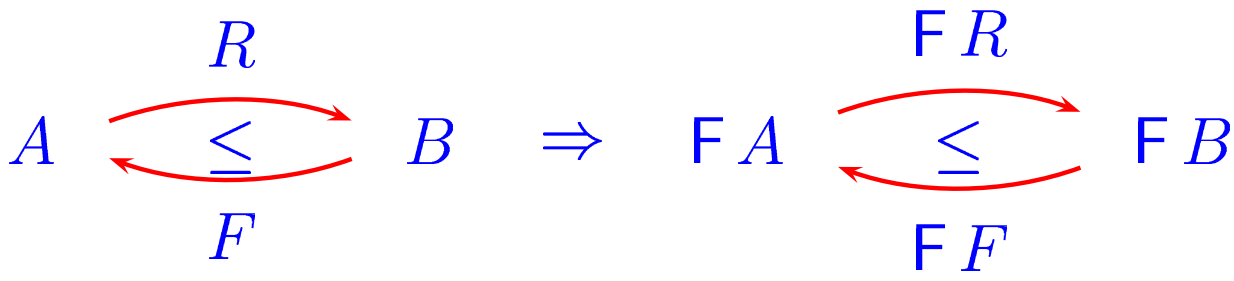
Prova da transitividade

- Composição preserva simplicidade e sobrejectividade:

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

- Verifi ca-se $S \cdot R \subseteq (F \cdot G)^{\circ}$ por monotonia.

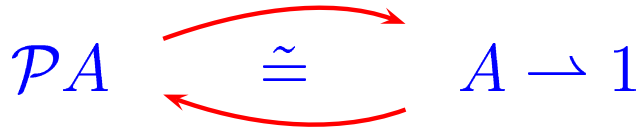
Refinamento Estruturado



onde F é um relacionador (functor) arbitrário:

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

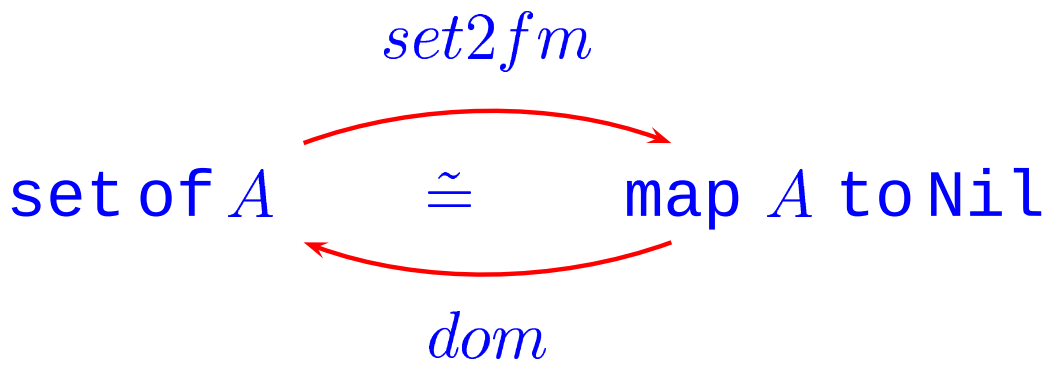
Refinando conjuntos finitos (I)



Cálculo:

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

Refinando conjuntos finitos (Ia)



VDM-SL

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

Pointfree

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

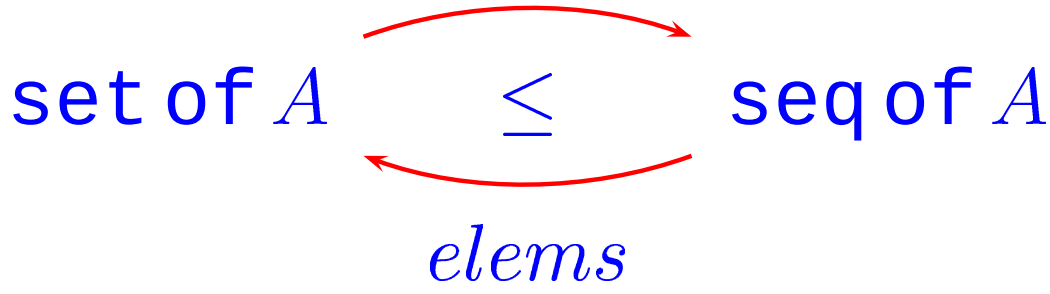
Invertibilidade à Direita

Cálculo:

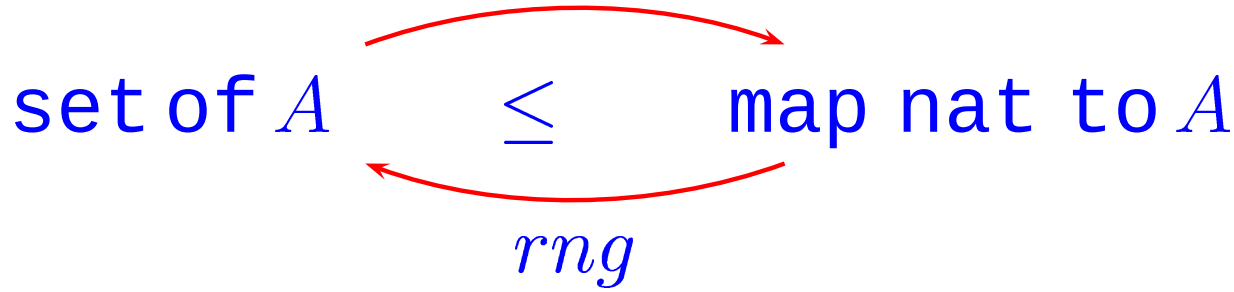
$$
\begin{array}{ll}
 & \mathit{dom} \cdot set2fm = id \\
\equiv & \quad \{\ \} \\
 & \mathit{dom}\ (set2fm\ s) = s \\
\equiv & \quad \{\ \} \\
 & \mathit{dom}\ (! \cdot s) = s \\
\equiv & \quad \{\ !\ \text{é uma função, } \mathit{dom}\ (f \cdot R) = \mathit{dom}\ R\} \\
 & \mathit{dom}\ s = s \\
\equiv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

Refinando conjuntos finitos (II)

Lista (cf. exemplo atrás):

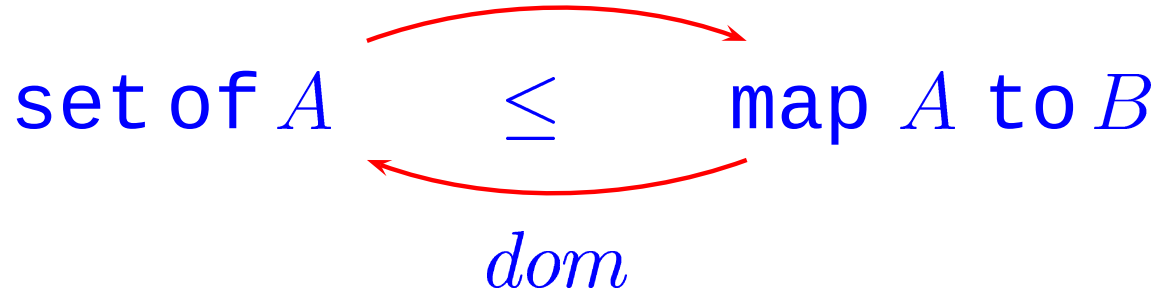


Índice A:

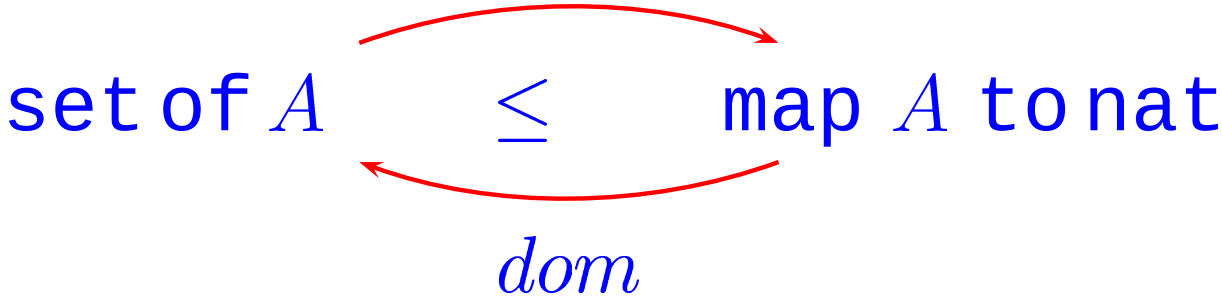


Refinando conjuntos finitos (III)

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

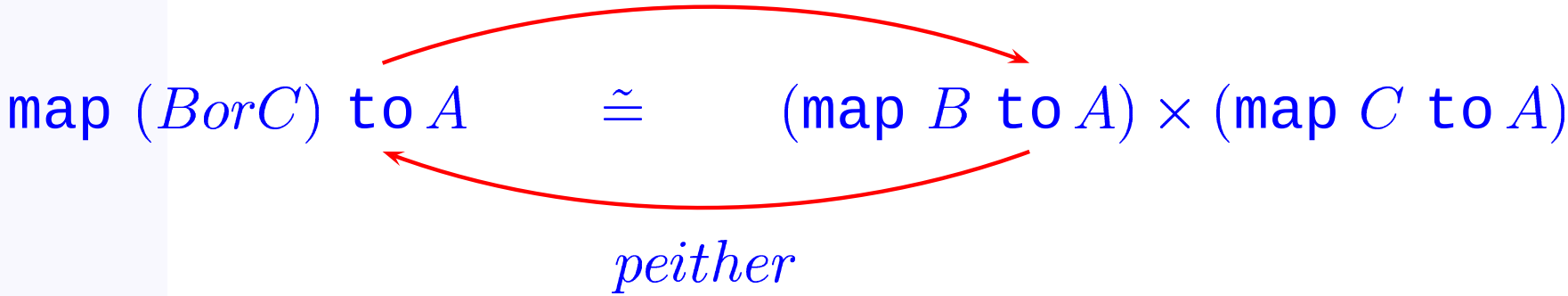


Quantificar A (“multisets”):



Refinando correspondências (I)

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




```
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} munion
                { mk_JustC(c) |-> n(c) | c in set dom n};
```

Refinando correspondências (Ia)

$$(B + C) \rightharpoonup A \quad \underset{peither}{\overset{unpeither}{\cong}} \quad (B \rightharpoonup A) \times (C \rightharpoonup A)$$

onde

$$peither(\sigma, \tau) \;=\; [\sigma, \tau]$$

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

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

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

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

ou seja,

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

- Assim, podíamos ter escrito p.e.

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

Refinando correspondências (II)

$$A \rightharpoonup (B + C) \quad \underset{cojoin}{\overset{uncojoin}{\leq}} \quad (A \rightharpoonup B) \times (A \rightharpoonup C)$$

onde

$$cojoin = \cup \cdot ((i_1 \cdot) \times (i_2 \cdot))$$

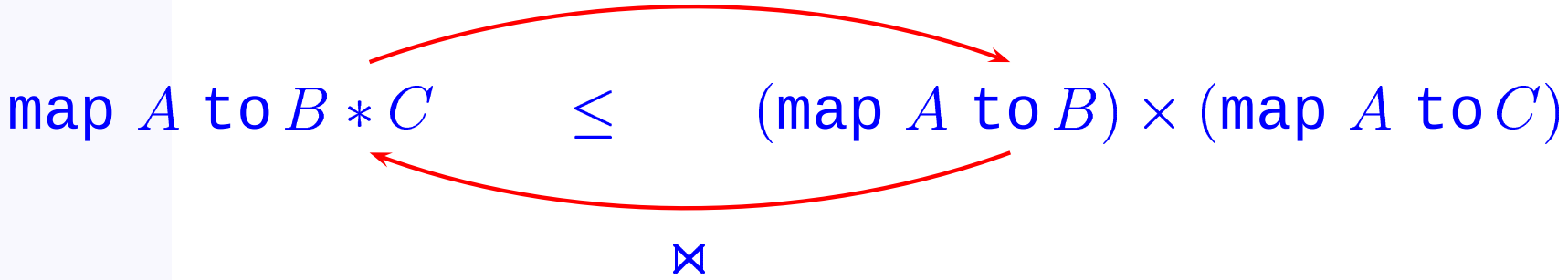
NB: $cojoin$ é parcial uma vez que a união de duas funções parciais nem sempre é uma função parcial.

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)



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)

$$A \rightharpoonup B \times C \quad \underset{\bowtie}{\overset{unjoin}{\leq}} \quad (A \rightharpoonup B) \times (A \rightharpoonup C)$$

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

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

Correspondências (IV)

Como estender

$$B^{C \times A} \underset{uncurry}{\overset{curry}{\cong}} (B^A)^C$$

a funções parciais? Caso $B := B + 1$

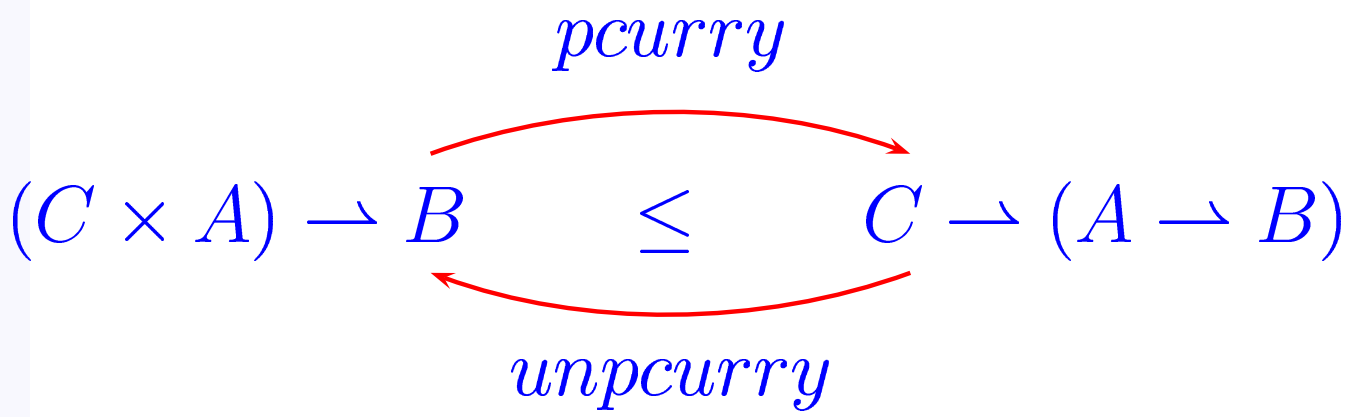
$$(B+1)^{C \times A} \cong ((B+1)^A)^C$$

$$\equiv \quad \{ \text{ i.e. } \}$$

$$(C \times A) \rightharpoonup B \cong (A \rightharpoonup B)^C$$

Correspondências (IVa)

Em geral:




```
unpcurry : map C to (map A to B) -> map (C * A) to B
unpcurry(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 isomorfi smo $curry/uncurry$ obtendo-se

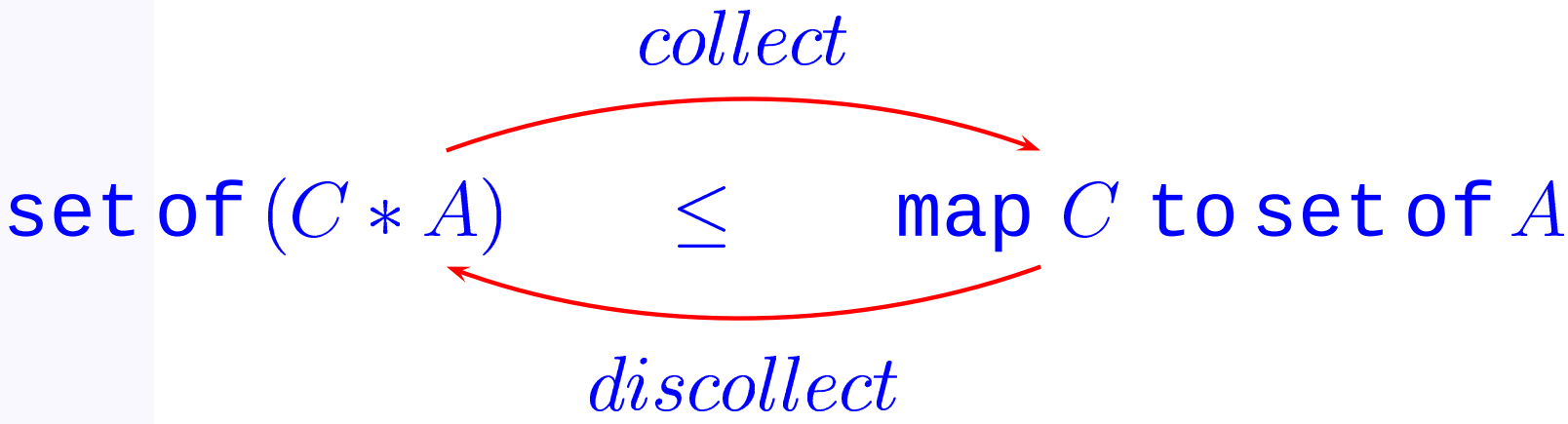
$$\mathcal{P}(A \times C) \underset{\Lambda^\circ}{\overset{\Lambda}{\cong}} (\mathcal{P}A)^C$$

onde

$$f = \Lambda R \quad \equiv \quad R = \in \cdot f \tag{4}$$

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

Transpondo relações finitas




```
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 fi m, embora importante

$$A \rightharpoonup D \times (B \rightharpoonup C) \quad \underset{\bowtie_n}{\overset{unnjoin}{\leq}} \quad (A \rightharpoonup D) \times ((A \times B) \rightharpoonup C) \tag{5}$$

onde

$$\bowtie_n \stackrel{\text{def}}{=} \bowtie \cdot \langle \pi_1, \dagger \cdot ((id \rightharpoonup \underline{\emptyset}) \times pcurry) \rangle \tag{6}$$

e

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

Recursividade

Como refi nar modelos recursivosdo VDM-SL tais como p.e.

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

ou seja, $FS = \mu\mathsf{F}$ para $\mathsf{F}\, X = Id \xrightarrow{m} (File + X)$:

$$\mu\mathsf{F} \underset{in}{\overset{out}{\cong}} Id \xrightarrow{m} (File + \mu\mathsf{F})$$

Recursividade

ou...

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

ou seja, $ArvDec = \mu$F em

$$ArvDec \;\tilde{=}\; OQue \times (Resposta \rightharpoonup ArvDec)$$

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

“Remoção” de Recursão

Dado

$$\mu\mathsf{F} \underset{in}{\overset{out}{\tilde{=}}} \mathsf{F}\,\mu\mathsf{F}$$

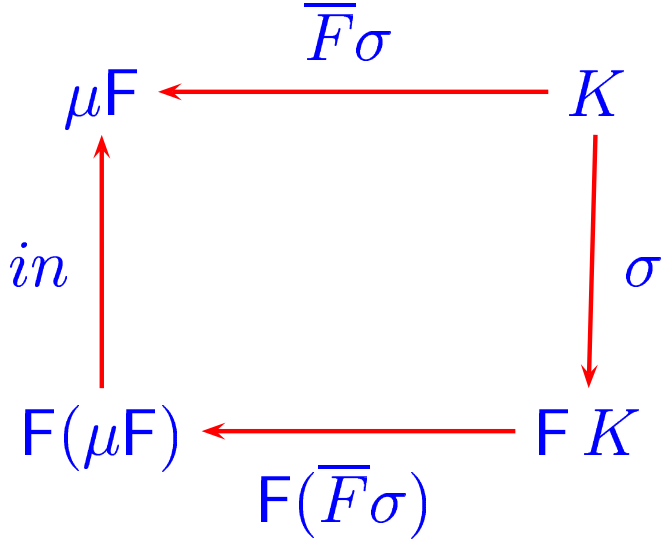
temos

$$\mu\mathsf{F} \underset{F}{\overset{\leq}{\rightleftarrows}} (K \rightharpoonup \mathsf{F}\,K) \times K \qquad (7)$$

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

Função de Abstracção

- O papel principal na representação é desempenhado por uma co-álgebra-F (parcial) $\mathsf{F}\,K \xleftarrow{\sigma} K$, assumida como um segmento (fi nito) de "memória linear", uma "heap" ou uma "base de dados".
- $\overline{F}$ (a transposta da abstracção F) é do tipo $(K \rightharpoonup \mathsf{F}\,K) \longrightarrow K \longrightarrow \mu\mathsf{F}$ e pode-se construir o hilomorfi smo



$$\overline{F}\sigma = in \cdot \mathsf{F}(\overline{F}\sigma) \cdot \sigma$$

Parcialidade da implementação

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

- $k \notin \mathit{dom}\,\sigma$
- σ não é "fechado" sobre si (ver em baixo)
- σ não é bem-fundado (ver em baixo)

Temos assim, o invariante concreto

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

De modo a definir $closed\ \sigma$ e $well f\ \sigma$, precisamos da relação de acessibilidade de σ, $\prec_\sigma$ (próximo diapositivo).

Acessibilidade e pertença

Relação de acessibilidade para σ:

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

$$\prec_\sigma \stackrel{\text{def}}{=} \in_{\mathsf{F}} \cdot \sigma$$

onde $K \xleftarrow{\in_{\mathsf{F}}} \mathsf{F}\,K$ estende $K \xleftarrow{\in} \mathcal{P}K$ indutivamente sobre functores polinomiais, como se segue:

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

Exemplo

Seja F $X = 1 + A \times X$. Então,

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

Exemplo (“pointfree”)

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

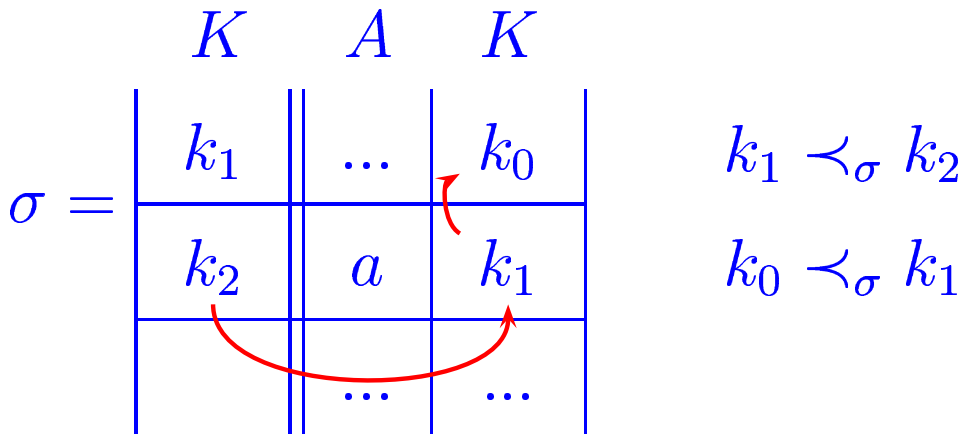
(Exemplo de) Acessibilidade

Acessibilidade dum apontador no caso duma heap "linear"

$(1 + A \times K) \overset{\sigma}{\longleftarrow} K$:

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

Desenhando:



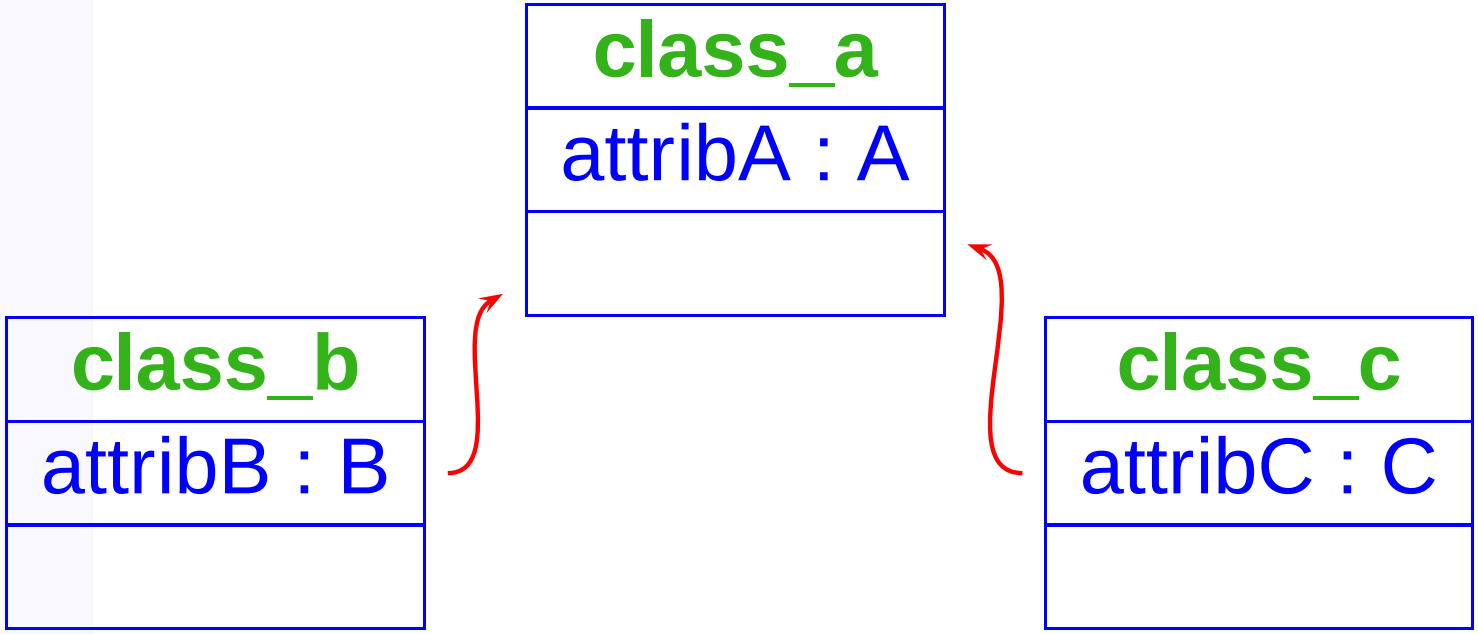
Fecho e Boa-formação

Seja $\prec_\sigma^+$ denotando o fecho transitivo de $\prec_\sigma$. Então,

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

O.O. Implementação de Dados

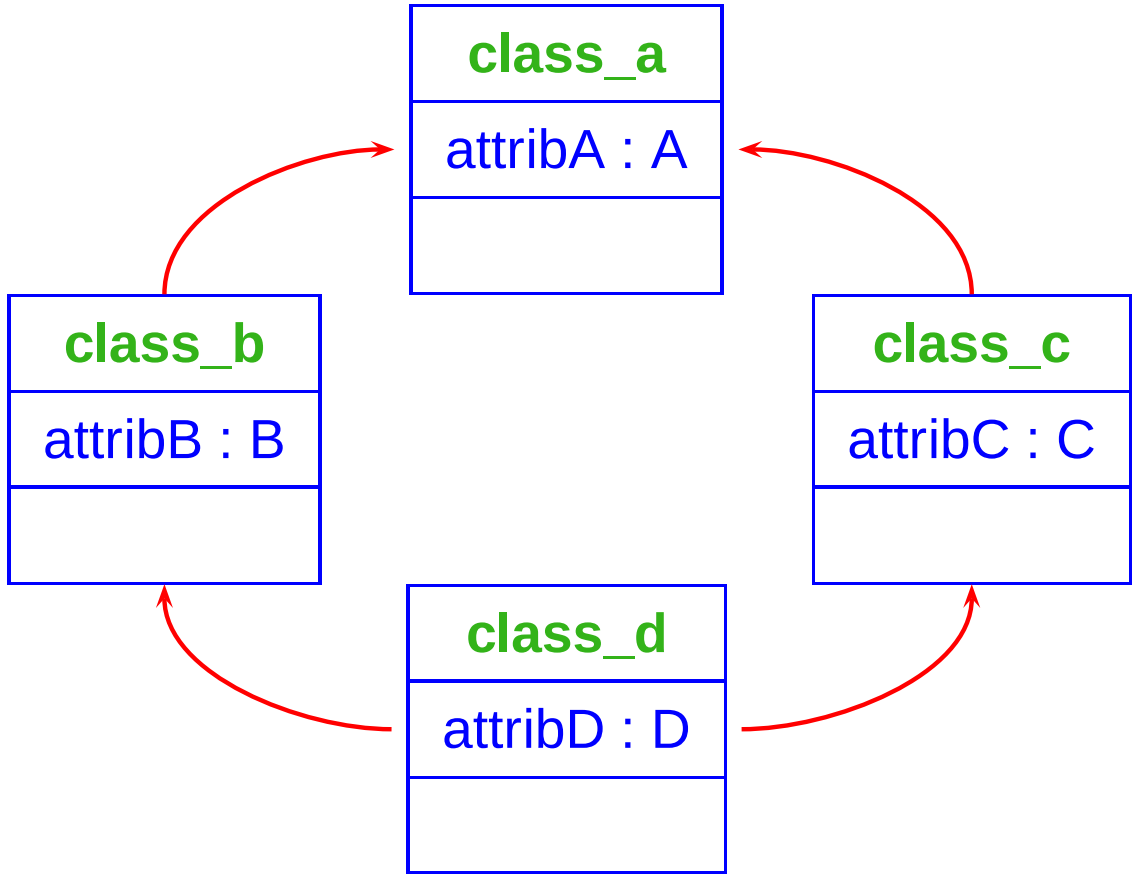
UML:



Modelo formal: $K \rightharpoonup Estrutura$ onde

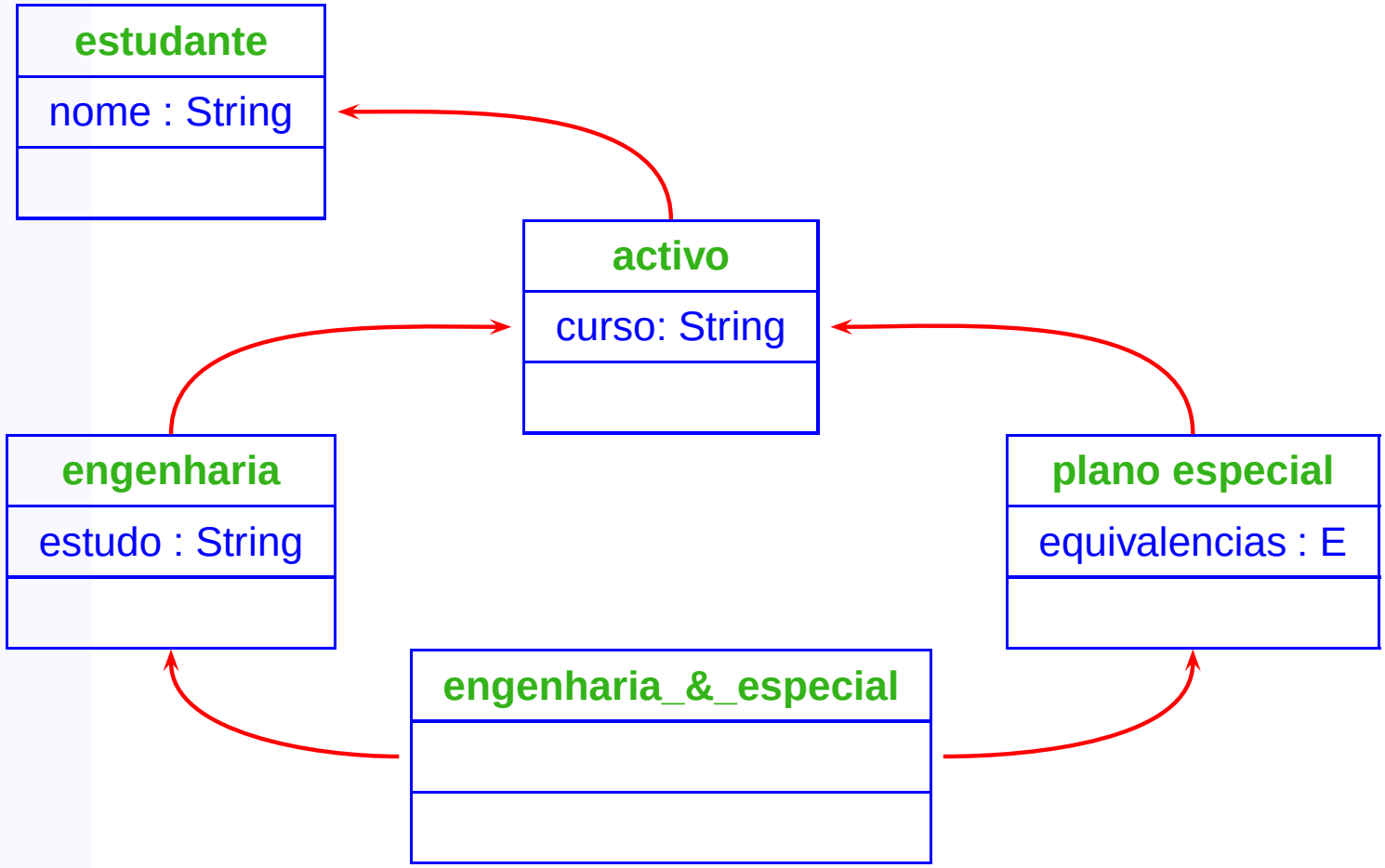
$$\begin{aligned} Estrutura &= A + A \times B + A \times C \\ &\cong A \times (1 + B + C) \end{aligned}$$

Herança múltipla



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

Exemplo



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