# An Introduction to Data Refinement

J.N. Oliveira
DIUM March 19, 2007

Formal Methods II, 2002-06,

March 19, 2007

## FM software design process

- Formal specification — "what" the intended software system should do
- Implementation — machine code produced instructing the hardware about "how" to do it

In general, there is more than one way in which a particular machine can accomplish "what" the specifier bore in mind:

- Relationship between specifications and implementations is **one-to-many**
- Specifications are more abstract than implementations.

## Overall idea

- Calculate implementations from specifications

$$\begin{array}{rcl} Spec & = & X \\ & \leq & X' \\ & \leq & X'' \\ & \leq & \cdots \\ & \leq & Imp \end{array}$$

  by adding details in a controlled manner.

- Define a suitable ordering $\leq$ on datatypes and develop corresponding **data refinement** theory

## Example of data refinement

Finite **sets** represented by finite **lists**:

$$
\begin{tikzcd}
 & {[1,2]} \arrow[dl, bend right, "F = \mathsf{elems}"'] \\
\{1,2\} & {[2,1]} \arrow[l] \\
 & {[1,2,1]} \arrow[ul, bend left]
\end{tikzcd}
$$

## Refinement inequation

$$
\begin{tikzcd}
\mathcal{P}_f\ A \arrow[r, bend left] \arrow[r, phantom, "\leq"] & A^{\star} \arrow[l, bend left, "elems"]
\end{tikzcd}
$$

meaning that

- sets are "implemented" by lists
- $A^{\star}$ is able to "represent" $\mathcal{P}_f\ A$
- $A^{\star}$ is "abstracted" by $\mathcal{P}_f\ A$
- $A^{\star}$ is a refinement ("refines") $\mathcal{P}_f\ A$

## Refinement inequations

$A$ *is implemented by* $B$, as witnessed by pair $f, r$, iff

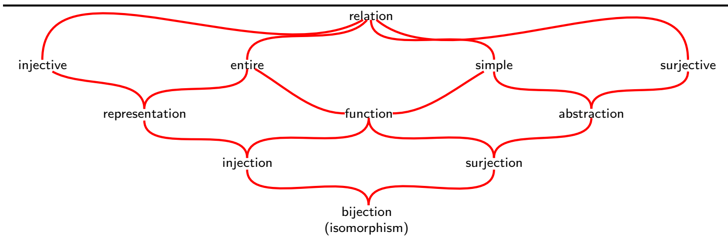
$$\begin{tikzcd} A \arrow[r, bend left, "r"] \arrow[r, phantom, "\leq"] & B \arrow[l, bend left, "f"] \end{tikzcd}$$

such that

- **representation** $r$ is injective
- **abstraction** $f$ is surjective
- that is,

$$f \cdot r \quad = \quad id$$

## Recall...




## Recall...

Taxonomy

| | Reflexive | Coreflexive |
|---|---|---|
| ker $R$ | entire $R$ | injective $R$ |
| img $R$ | surjective $R$ | simple $R$ |

Kernel

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

Image (its dual)

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

## Not general enough (I)

In the following inequation

$$\begin{tikzcd} A \arrow[r, bend left, "i_1"] \arrow[r, phantom, "\leq"] & A+1 \arrow[l, bend left, "i_1^\circ"] \end{tikzcd}$$

expressing the fact that every element of datatype $A$ can be represented by a "pointer",

- $r = i_1$ is injective, but
- its converse $i_1^\circ$ is partial (=not entirely defined)

## Not general enough (II)

Representations $r$ need not be functions. Back to

$$\begin{tikzcd} \mathcal{P}_f\, A \arrow[r, bend left, "R"] \arrow[r, phantom, "\leq"] & A^\star \arrow[l, bend left, "elems"] \end{tikzcd}$$

relation $R = elems^\circ$ will be perfectly acceptable as a representation since

$$elems \cdot elems^\circ = id$$

because $elems$ is a surjection.

## Data refinement

Principle of **data abstraction**: $A$ abstracts $B$ wherever

- A surjective abstraction $A \xleftarrow{F} B$ can be found:

$$\mathsf{img}\, F \quad = \quad id \tag{1}$$

  $F$ is thus **simple** but possibly partial.

- Any **entire** subrelation $R$ of $F^\circ$ is said to be a representation for $F$. So $R \subseteq F^\circ$.

## Representation relations

- It follows that $R$ is **injective**, since $\ker R \subseteq \ker F^\circ$ and $\ker F^\circ = \mathrm{img}\, F = id$.
- So, no two different abstract values $a, a' \in A$ get mixed up along the representation process.
- Altogether, $\ker R = id$ because $id \subseteq \ker R \subseteq id$ ($R$ is entire).
- It follows that $R$ is a **right-inverse** of $F$, that is

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

This is proved by circular inclusion

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

in the next slide.

## Right invertibility

$$
\begin{array}{ll}
& F \cdot R \subseteq id \ \wedge \ id \subseteq F \cdot R \\
\equiv & \qquad \{ \ \mathrm{img}\, F = id \text{ and } \ker R = id \} \\
& F \cdot R \subseteq F \cdot F^\circ \ \wedge \ R^\circ \cdot R \subseteq F \cdot R \\
\equiv & \qquad \{ \ \text{converses} \ \} \\
& F \cdot R \subseteq F \cdot F^\circ \ \wedge \ R^\circ \cdot R \subseteq R^\circ \cdot F^\circ \\
\Leftarrow & \qquad \{ \ (F \cdot) \text{ and } (R^\circ \cdot) \text{ are monotone } \} \\
& R \subseteq F^\circ \ \wedge \ R \subseteq F^\circ \\
\equiv & \qquad \{ \ R \subseteq F^\circ \text{ is assumed } \} \\
& \mathrm{TRUE}
\end{array}
$$

## Refinement inequations

$$A \underset{F}{\overset{R}{\underset{\longleftarrow}{\overset{\longrightarrow}{\leq}}}} B \quad \text{such that} \quad F \cdot R = id_A$$

This inequation has several informal interpretations:

- $A$ is “smaller” than $B$
- $B$ is able to “represent” $A$
- $B$ is “abstracted” by $A$
- $A$ is “implemented” by $B$
- $B$ is a refinement (“refines”) $A$

## Refinement equations

Isomorphisms: $$A \underset{f}{\overset{r}{\underset{\longleftarrow}{\overset{\longrightarrow}{\cong}}}} B \quad \text{such that} \quad r = f^\circ$$

$$\begin{array}{cl} & r = f^\circ \\ \equiv & \quad \{ \text{ add variables } \} \\ & b\ r\ a \quad \equiv \quad b f^\circ a \\ \equiv & \quad \{ \text{ functions and converses } \} \\ & b = r\ a \quad \equiv \quad f\ b = a \end{array}$$

## Example

Back to representing finite **sets** by finite **lists**:

$$
\begin{tikzcd}
 & {[1,2]} \arrow[dl, bend right, "F = \mathsf{elems}"'] \\
\{1,2\} & {[2,1]} \arrow[l] \\
 & {[1,2,1]} \arrow[ul, bend left]
\end{tikzcd}
$$

Among the many $R \subseteq F^{\circ}$, we may choose the following:

## Relational representation

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

Intuitively,

$$\rho\, Listify = [\![ noRepeats ]\!]$$

where

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

## Functional representation

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

Intuitively,

$$\rho\, listify = [\![ IsOrdered ]\!] \cdot [\![ noRepeats ]\!]$$

## Concrete invariants

- Wherever

$$\begin{tikzcd} A \arrow[r, bend left, "R"] \arrow[r, phantom, "\leq"] & B \arrow[l, bend left, "F"] \end{tikzcd} \qquad \text{such that } R \subseteq F^{\circ} \text{ and } \rho\, R = [\![\phi]\!]$$

  we say that $\phi$ is the **concrete invariant** induced by $R$.

- In case $R$ is a function, and because it always is injective, one has

$$A \quad \cong \quad B_{\phi}$$

  where $B_{\phi}$ denotes the subset of $B$ which satisfies concrete-invariant $\phi$.

## Example of a partial abstraction

Every element of datatype $A$ can be represented by a "pointer":

$$\begin{tikzcd} A \arrow[r, bend left, "i_1"] \arrow[r, phantom, "\leq"] & A+1 \arrow[l, bend left, "i_1^{\circ}"] \end{tikzcd}$$

- Simplicity of the abstraction is ensured by a known fact: the converse of an injective relation is simple.
- Concrete invariant: $\phi = [\underline{\mathrm{TRUE}}\ , \underline{\mathrm{FALSE}}]$

## Another partial abstraction

Finite mappings "are" (simple) finite relations:

$$\begin{tikzcd} \texttt{map}\ A\ \texttt{to}\ B \arrow[r, bend left, "mkr"] \arrow[r, phantom, "\leq"] & \texttt{set of}\ (A * B) \arrow[l, bend left] \end{tikzcd}$$

VDM-SL: $mkf = mkr^{\circ}$

```
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);
```

(Guess the concrete invariant.)

## Properties of $\leq$:

### Reflexivity

$$
\begin{tikzcd}
A \arrow[r, bend left, "id"] \arrow[r, phantom, "\leq"] & A \arrow[l, bend left, "id"]
\end{tikzcd}
\qquad \textit{cf.} \quad id \cdot id = id
$$

### Transitivity

$$
\begin{tikzcd}
A \arrow[r, bend left, "R"] \arrow[r, phantom, "\leq"] & B \arrow[l, bend left, "F"]
\end{tikzcd}
\ \wedge\
\begin{tikzcd}
B \arrow[r, bend left, "S"] \arrow[r, phantom, "\leq"] & C \arrow[l, bend left, "G"]
\end{tikzcd}
\ \Rightarrow\
\begin{tikzcd}
A \arrow[r, bend left, "S \cdot R"] \arrow[r, phantom, "\leq"] & C \arrow[l, bend left, "F \cdot G"]
\end{tikzcd}
$$

## Proof of transitivity

- First show that composition preserves simplicity and surjectiveness:

$$
\begin{array}{cl}
 & \mathsf{img}\,(F \cdot G) = id \\
\equiv & \qquad \{ \text{ expanding and converses} \} \\
 & F \cdot (\mathsf{img}\,G) \cdot F^{\circ} = id \\
\equiv & \qquad \{ \ G \text{ is simple and surjective} \} \\
 & \mathsf{img}\,F = id \\
\equiv & \qquad \{ \ F \text{ is simple and surjective} \} \\
 & id = id
\end{array}
$$

- Then note that $S \cdot R \subseteq (F \cdot G)^{\circ}$ by monotonicity.

## Structural data refinement

$$\begin{tikzcd} A \arrow[r, bend left, "R"] \arrow[r, phantom, "\leq"] & B \arrow[l, bend left, "F"] \end{tikzcd} \quad \Rightarrow \quad \begin{tikzcd} \mathsf{F}\,A \arrow[r, bend left, "\mathsf{F}\,R"] \arrow[r, phantom, "\leq"] & \mathsf{F}\,B \arrow[l, bend left, "\mathsf{F}\,F"] \end{tikzcd}$$

where F is an arbitrary relator (functor):

$$\begin{array}{ll} & (\mathsf{F}\,F)\cdot(\mathsf{F}\,R) \\ = & \quad \{\text{ relators commute with composition} \} \\ & \mathsf{F}\,(F\cdot R) \\ = & \quad \{\ R \text{ is right-inverse of } F\ \} \\ & \mathsf{F}\,id \\ = & \quad \{\text{ relators commute with } id \} \\ & id \end{array}$$

therefore F $R$ is right-inverse of F $f$. Of course, this result extends to bifunctors.

## Relators

A **relator** is a functor on relations

$$\begin{tikzcd} A \arrow[d, "X"'] \\ B \end{tikzcd} \qquad \begin{tikzcd} \mathsf{F}\,A \arrow[d, "\mathsf{F}\,X"] \\ \mathsf{F}\,B \end{tikzcd}$$

which is monotonic and commutes with converse:

$$\begin{array}{rcl} R \subseteq S & \Rightarrow & (\mathsf{F}\,R) \subseteq (\mathsf{F}\,S) \\ \mathsf{F}\,(R^\circ) & = & (\mathsf{F}\,R)^\circ \end{array}$$

## Relators

Recall that $F$ will commute with composition and identity too:

$$\mathsf{F}\,(R\cdot S) = (\mathsf{F}\,R)\cdot(\mathsf{F}\,S) \tag{3}$$

$$\mathsf{F}\,id = id \tag{4}$$

Example: $X^\star$ will be such that

$$l(X^\star)l' \quad \equiv \quad \mathsf{len}\,l = \mathsf{len}\,l' \ \wedge\ \forall i \in \mathsf{inds}\,l . (l\ i) X (l'\ i)$$

## Polynomial relators

$$\begin{array}{lrcl}
\textbf{Identity:} & \mathsf{Id}\ R & = & R \\
\textbf{Constant:} & \mathsf{K}\ R & = & id_K \\
\textbf{Product:} & R \times S & = & \langle R \cdot \pi_1, S \cdot \pi_2 \rangle \\
\textbf{Sum:} & R + S & = & [i_1 \cdot R\ , i_2 \cdot S]
\end{array}$$

where

$$\begin{array}{rcl}
\langle R, S \rangle & = & \pi_1^\circ \cdot R \cap \pi_2^\circ \cdot S \\
[R\ , S] & = & (R \cdot i_1^\circ) \cup (S \cdot i_2^\circ)
\end{array}$$

For instance,

$$Maybe\ A \quad \cong \quad (\mathsf{Id} + 1)A = A + 1$$

## "Maybe" transpose

Useful isomorphism for conversion of simple relations into a $Maybe$-valued functions

$$\begin{tikzcd}
(B+1)^A \arrow[r, phantom, "\cong"] \arrow[r, bend left, "untot = (i_1^\circ \cdot\ )"] & A \rightharpoonup B \arrow[l, bend left, "tot"]
\end{tikzcd}$$

where $A \rightharpoonup B$ denotes the set of all simple relations from $A$ to $B$:

$$f = tot\ R \quad \equiv \quad (b\ R\ a \quad \equiv \quad (f\ a = i_1\ b))$$

## "Maybe" transpose — VDM-SL

$$\begin{tikzcd}
A \rightharpoonup B \arrow[r, phantom, "\cong"] \arrow[r, bend left, "tot"] & (B+1)^A \arrow[l, bend left, "untot"]
\end{tikzcd} \tag{5}$$

where, for types `A`, `B` and `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) };
```

## Pointwise $untot = (i_1^\circ \cdot)$

As checked next:

$$
\begin{array}{ll}
 & untot\ f = i_1^\circ \cdot f \\
\equiv & \qquad \{ \text{ relations as set comprehensions} \} \\
 & untot\ f = \{(b,a) \mid a \in A, b \in B : b(i_1^\circ \cdot f)a\} \\
\equiv & \qquad \{ \text{ using rule } b(f^\circ \cdot R \cdot g)a \equiv (f\ b)R(g\ a) \ \} \\
 & untot\ f = \{(b,a) \mid a \in A, b \in B : i_1\ b = f\ a\} \\
\equiv & \qquad \{ \text{ VDM-SL notation} \} \\
 & untot\ f = \texttt{\{a|->b|a:A,b:B \& f(a)=mk\_JustB(b)\}}
\end{array}
$$

## Corol. of "Maybe" transpose (I)

Isomorphism

$$
\begin{tikzcd}
A^1 \arrow[r, bend left] \arrow[r, phantom, "\cong"] & A \arrow[l, bend left]
\end{tikzcd}
$$

extends to partial functions as follows:

$$
\begin{tikzcd}
1 \rightharpoonup A \arrow[r, bend left, "f^\circ"] \arrow[r, phantom, "\cong"] & A+1 \arrow[l, bend left, "f"]
\end{tikzcd}
\qquad \text{(guess } f \text{ and } f^\circ\text{).}
$$

That is, the "singleton" finite map is a disguise of a "pointer".

## Corol. of "Maybe" transpose (II)

Sets are degenerated maps:

$$
\begin{tikzcd}
\mathcal{P} A \arrow[r, bend left] \arrow[r, phantom, "\cong"] & A \rightharpoonup 1 \arrow[l, bend left]
\end{tikzcd}
$$

Calculation:

$$
\begin{array}{ll}
 & A \rightharpoonup 1 \\
\cong & \qquad \{ \ tot \text{ representation } \} \\
 & (1+1)^A \\
\cong & \qquad \{ \text{ basic} \} \\
 & 2^A \\
\cong & \qquad \{ \ 2^A \text{ is isomorphic to } \mathcal{P} A \ \} \\
 & \mathcal{P} A
\end{array}
$$

## Corol. of "Maybe" transpose (IIa)

$$
\begin{tikzcd}
\mathcal{P}A \arrow[r, bend left, "set2fm"] \arrow[r, phantom, "\cong"] & A \rightharpoonup 1 \arrow[l, bend left, "dom"]
\end{tikzcd}
$$

VDM-SL

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

Pointfree

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

## Right-invertibility

Calculation:

$$
\begin{array}{ll}
 & \delta \cdot set2fm = id \\
\equiv & \quad \{\ \} \\
 & \delta\,(set2fm\ s) = s \\
\equiv & \quad \{\ \} \\
 & \delta\,(! \cdot s) = s \\
\equiv & \quad \{\ !\text{ is a function, } \delta\,(f \cdot R) = \delta\,R \} \\
 & \delta\,s = s \\
\equiv & \quad \{\ s \text{ is coreflexive} \} \\
 & s = s
\end{array}
$$

## Corol. of "Maybe" transpose (III)

Isomorphism

$$\begin{tikzcd} B \times C \rightharpoonup A \arrow[r, phantom, "\cong"] \arrow[r, bend left, "scurry"] & (C \rightharpoonup A)^B \arrow[l, bend left] \end{tikzcd}$$

extends currying

$$\begin{tikzcd} B^{C \times A} \arrow[r, phantom, "\cong"] \arrow[r, bend left, "curry"] & (B^A)^C \arrow[l, bend left] \end{tikzcd}$$

to simple relations, as calculated in the next slide.

## Corol. of "Maybe" transpose (III)

$$\begin{array}{ll} & B \times C \rightharpoonup A \\ \cong & \quad \{ \ tot/untot \ \} \\ & (A+1)^{B \times C} \\ \cong & \quad \{ \ curry/uncurry \ \} \\ & ((A+1)^C)^B \\ \cong & \quad \{ \ (i_1^\circ \cdot)^B \ \} \\ & (C \rightharpoonup A)^B \end{array}$$

This is referred to as the **multiple-key** decomposition / synthesis isomorphism.

## Corol. of "Maybe" transpose (III)

The $scurry$ isomorphism is as follows, where we abbreviate $scurry\ R$ to $\overline{R}$:

$$f = \overline{R} \quad \equiv \quad \langle \forall\ a, b, c\ ::\ c\ (f\ a)\ b \equiv c\ R\ (a, b) \rangle$$

Its VDM-SL equivalent for finite mappings is

```
scurry : map A*B to C -> (A -> map B to C)
scurry(M)(a) == { b |-> M(mk_(a',b))
                  | mk_(a',b) in set dom M
                    & a'=a };
```

## Corol. of "Maybe" transpose (IV)

Refinement of **nested** simplicity by decomposition:

$$
\begin{tikzcd}
A \rightharpoonup (D \times (B \rightharpoonup C)) \arrow[r, phantom, "\leq"] \arrow[r, bend left, "unnjoin"] & (A \rightharpoonup D) \times ((A \times B) \rightharpoonup C) \arrow[l, bend left, "\bowtie_n"]
\end{tikzcd}
$$

where $\quad R \bowtie_n S \quad = \quad \langle R, \overline{S} \rangle$
and $\quad unnjoin\ R \quad = \quad (\pi_1 \cdot R, unpcurry(\pi_2 \cdot R))$ (see definition of $unpcurry$ in the sequel.)

## Calculation

$$
\begin{array}{ll}
& A \rightharpoonup (D \times (B \rightharpoonup C)) \\
\cong & \qquad \{ \ \mathit{Maybe}\ \text{transpose} \ \} \\
& ((D \times (B \rightharpoonup C)) + 1)^A \\
\leq & \qquad \{ \ \mathit{Maybe}\text{-(right)strength is involved in the abstraction} \ \} \\
& ((D + 1) \times (B \rightharpoonup C))^A \\
\cong & \qquad \{ \ \text{splitting} \ \} \\
& (D + 1)^A \times (B \rightharpoonup C)^A \\
\cong & \qquad \{ \ \mathit{Maybe}\ \text{transpose and multiple-key synthesis} \ \} \\
& (A \rightharpoonup D) \times (A \times B \rightharpoonup C)
\end{array}
$$

## Details on the $\bowtie_n$ abstraction

Pointwise:

$$
\begin{aligned}
(d, M)(R \bowtie_n S)a \quad &\equiv \quad d\ R\ a\ \wedge\ M = (\overline{S})a \\
&\equiv \qquad \{\ scurry\ \} \\
&\quad\ \ d\ R\ a\ \wedge\ (c\ M\ b \equiv c\ S\ (a, b))
\end{aligned}
$$

VDM-SL equivalent for finite mappings:

```
njoin : (map A to D)*(map A*B to C)
          -> map A to (D* (map B to C))
njoin(M,N) ==
    { a |-> mk_(M(a), { b |-> N(mk_(a,b))
                           | mk_(a,b) in set dom N })
        | a in set dom M };
```

## Its representation is

```
unnjoin : map A to (D* map B to C) ->
          (map A to D)*(map A*B to C)
unnjoin(M) ==
    mk_({ a |-> M(a).#1 | a in set dom M },
        merge {{ mk_(a,b) |-> M(a).#2(b)
                            | b in set dom M(a).#2 }
               | a in set dom M }
);
```

Concrete invariant induced by $unnjoin$:

$$\phi_{unnjoin}(M, N) \quad = \quad N \preceq M \cdot \pi_1$$

where $R \preceq S \quad \equiv \quad \delta\, R \subseteq \delta\, S$

## Corol. of "Maybe" transpose (V)

$$
\begin{tikzcd}
(B + C) \rightharpoonup A \arrow[rr, bend left, "unpeither"] & \cong & (B \rightharpoonup A) \times (C \rightharpoonup A) \arrow[ll, bend left, "peither"]
\end{tikzcd}
$$

where

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

for $[R \ , S] = (R \cdot i_1^\circ) \cup (S \cdot i_2^\circ)$, that is

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

## Corol.of "Maybe" transpose (Va)

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

$$
\begin{tikzcd}
\mathtt{map}\ (BorC)\ \mathtt{to}\ A \arrow[rr, bend left] & \cong & (\mathtt{map}\ B\ \mathtt{to}\ A) \times (\mathtt{map}\ C\ \mathtt{to}\ A) \arrow[ll, bend left, "peither"]
\end{tikzcd}
$$

```
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};
```

NB: a "1st NF" representation rule

## Relational projection

Given a binary relation $R$ and suitably typed functions $f$ and $g$,

- the $g, f$-projection of $R$ is defined as binary relation

$$\pi_{g,f} R \quad \stackrel{\text{def}}{=} \quad g \cdot R \cdot f^{\circ} \tag{6}$$

- wherever $R$ is simple and $g \cdot R \cdot f^{\circ}$ is also simple, we write $f \rightharpoonup g$ instead of $\pi_{g,f} R$. So,

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

- $(f \rightharpoonup g)R$ is always simple when $f$ is injective.
- So, we could have written *e.g.*

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

## Refining finite sets (II)

List (cf. example before):

$$\begin{tikzcd} \texttt{set of } A \arrow[r, bend left] \arrow[r, phantom, "\leq"] & \texttt{seq of } A \arrow[l, bend left, "elems"] \end{tikzcd}$$

Index $A$:

$$\begin{tikzcd} \texttt{set of } A \arrow[r, bend left] \arrow[r, phantom, "\leq"] & \texttt{map nat to } A \arrow[l, bend left, "rng"] \end{tikzcd}$$

## Refining finite sets (III)

Classify $A$ by $B$ ($B \supset \{\}$):

$$\begin{tikzcd} \texttt{set of } A \arrow[r, bend left] \arrow[r, phantom, "\leq"] & \texttt{map } A \texttt{ to } B \arrow[l, bend left, "dom"] \end{tikzcd}$$

Quantify $A$ ("multisets"):

$$\begin{tikzcd} \texttt{set of } A \arrow[r, bend left] \arrow[r, phantom, "\leq"] & \texttt{map } A \texttt{ to nat} \arrow[l, bend left, "dom"] \end{tikzcd}$$

## Refining finite maps (II)

$$\begin{tikzcd}
A \rightharpoonup (B + C) \arrow[r, phantom, "\leq"] \arrow[r, bend left, "uncojoin"] & (A \rightharpoonup B) \times (A \rightharpoonup C) \arrow[l, bend left, "cojoin"]
\end{tikzcd}$$

where

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

NB: $cojoin$ is partial since the union of two partial functions not always is a partial function.

## Refining finite maps (IIa)

Note the representation function:

```
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)) }
       );
```

## Refining finite maps (III)

$$\begin{tikzcd}
A \rightharpoonup B \times C \arrow[r, phantom, "\leq"] \arrow[r, bend left, "unjoin"] & (A \rightharpoonup B) \times (A \rightharpoonup C) \arrow[l, bend left, "\bowtie"]
\end{tikzcd}$$

where

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

where $\langle R, S \rangle \stackrel{\mathrm{def}}{=} (\pi_1^\circ \cdot R) \cap (\pi_2^\circ \cdot S)$. A right-inverse of $join$ is

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

## Refining finite maps (IIIa)

$$\begin{tikzcd}[column sep=huge] \mathtt{map}\ A\ \mathtt{to}\ B * C \arrow[r, phantom, "\leq"] \arrow[r, bend left] & (\mathtt{map}\ A\ \mathtt{to}\ B) \times (\mathtt{map}\ A\ \mathtt{to}\ C) \arrow[l, bend left, "\bowtie"] \end{tikzcd}$$

where (writing join for $\bowtie$)

```
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 };
```

## Refining finite maps (IVa)

In general:

$$\begin{tikzcd}[column sep=huge] (C \times A) \rightharpoonup B \arrow[r, phantom, "\leq"] \arrow[r, bend left, "pcurry"] & C \rightharpoonup (A \rightharpoonup B) \arrow[l, bend left, "unpcurry"] \end{tikzcd}$$

```
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 };
```

## Refining finite maps (IVb)

Pointwise

```
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 };
```

Pointfree

$$pcurry\ M \quad = \quad \overline{M} - \underline{\perp}$$

(recall $scurry$)

## Transposing relations

Let $B := 2$ in the $curry/uncurry$ isomorphism and obtain

$$\begin{tikzcd} \mathcal{P}(A \times C) \arrow[rr, bend left, "\Lambda"] \arrow[rr, phantom, "\cong"] & & (\mathcal{P}A)^C \arrow[ll, bend left, "\Lambda^\circ"] \end{tikzcd}$$

where

$$f = \Lambda R \quad \equiv \quad R = \in \cdot f \tag{7}$$

and $A \xleftarrow{\in} \mathcal{P}A$ is the membership relation.

## Transposing finite relations

$$\begin{tikzcd} \texttt{set of } (C * A) \arrow[rr, bend left, "collect"] \arrow[rr, phantom, "\leq"] & & \texttt{map } C \texttt{ to set of } A \arrow[ll, bend left, "discollect"] \end{tikzcd}$$

```
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 };
```

## What about recursive data?

How does one refine recursive VDM-SL models such as *e.g.*

```
FS :: D: map Id to Node; -- FS means file system
Node = File | FS;        -- a Node is either a file
                         --   or a directory
Id = seq of char;        -- node identifiers
File :: F: seq of token  -- sequential files
```

that is, $FS = \mu\mathsf{F}$ for $\mathsf{F}\,X = Id \rightharpoonup (File + X)$:

$$\begin{tikzcd} \mu\mathsf{F} \arrow[rr, bend left, "out"] \arrow[rr, phantom, "\cong"] & & Id \rightharpoonup (File + \mu\mathsf{F}) \arrow[ll, bend left, "in"] \end{tikzcd}$$

## The `DecTree` example

or...

```
DecTree :: question: What
           answers: map Answer to DecTree
What = seq of char;
Answer = seq of char;
```

that is, $DecTree = \mu\mathsf{F}$ in

$$DecTree \quad \cong \quad What \times (Answer \rightharpoonup DecTree)$$

for $\mathsf{F}\, X = What \times (Answer \rightharpoonup X)$

## The `Exp` example

or even...

```
     Exp    = Var | Term ;
     Var    :: variable: Symbol ;
     Term   :: operator: Symbol
               arguments: seq of Exp
               inv t == len t.arguments <= 20 ;
     Symbol = seq of char
              inv s == len s <= 10 ;
```

that is, $Exp = \mu\mathsf{F}$ in

$$Exp \quad \cong \quad Symbol + Symbol \times Exp^{\star}$$

for $\mathsf{F}\, X = Symbol + Symbol \times X^{\star}$

## Getting away with recursion

Given

$$\begin{tikzcd} \mu\mathsf{F} \arrow[r, bend left, "out"] \arrow[r, phantom, "\cong"] & \mathsf{F}\,\mu\mathsf{F} \arrow[l, bend left, "in"] \end{tikzcd}$$

one has

$$\begin{tikzcd} \mu\mathsf{F} \arrow[r, bend left] \arrow[r, phantom, "\leq"] & \underbrace{(K \rightharpoonup \mathsf{F}\,K)}_{\text{"}heap\text{"}} \times K \arrow[l, bend left, "F"] \end{tikzcd}$$

for $K$ a data type of "heap addresses", or "pointers", such that $K \cong I\!N$.

## An example to start with

Since

$$Exp = \mu X.(Symbol + Symbol \times X^{\star})$$

we have:

$$\begin{array}{ll} & Exp \\ \leq & \quad \{ \text{ remove recursion } \} \\ & (K \rightharpoonup (Symbol + Symbol \times K^{\star})) \times K \\ \leq & \quad \{ \text{ remove finite lists } \} \\ & (K \rightharpoonup (Symbol + Symbol \times (I\!N \rightharpoonup K))) \times K \end{array}$$

## Example continued

$$\begin{array}{ll} \leq & \quad \{ \text{ recall } A \rightharpoonup (B + C) \leq (A \rightharpoonup B) \times (A \rightharpoonup C) \ \} \\ & (K \rightharpoonup Symbol) \times (K \rightharpoonup (Symbol \times (I\!N \rightharpoonup K))) \times K \\ \leq & \quad \{ \text{ remove nested } \rightharpoonup \ \} \\ & (K \rightharpoonup Symbol) \times (K \rightharpoonup Symbol) \times ((I\!N \times K) \rightharpoonup K) \times K \\ \cong & \quad \{ \ A \times A \cong A^2 \ \} \\ & (K \rightharpoonup Symbol)^2 \times ((I\!N \times K) \rightharpoonup K) \times K \\ \cong & \quad \{ \text{ recall } (C \rightharpoonup A)^B \cong B \times C \rightharpoonup A \} \\ & \underbrace{((2 \times K) \rightharpoonup Symbol)}_{SYMBOLS} \times \underbrace{((I\!N \times K) \rightharpoonup K)}_{EXPRESSIONS} \times K \end{array}$$

## SQL encoding

Symbols table:

```
CREATE TABLE SYMBOLS (
       Symbol   CHAR     (20) NOT NULL,
       NodeId   NUMERIC  (10) NOT NULL,
       IfVar    BOOLEAN       NOT NULL
       CONSTRAINT SYMBOLS_pk
                  PRIMARY KEY(NodeId,IfVar)
       );
```

## SQL encoding

Expressions table:

```
CREATE TABLE EXPRESSIONS (
       FatherId  NUMERIC (10) NOT NULL,
       ArgNr     NUMERIC (10) NOT NULL,
       ChildId   NUMERIC (10) NOT NULL
       CONSTRAINT EXPRESSIONS_pk
                  PRIMARY KEY (FatherId,ArgNr)
       );
```

Can you rely on this implementation? Need for an abstraction invariant!

## Abstraction function

- Main rôle in representation is played by simple $\mathsf{F}$-coalgebra $K \rightharpoonup \mathsf{F}\,K$, understood as a (finite) piece of "linear storage", a "heap" or a "database" file.
- $\overline{F}$ (recall $\overline{F}$ notation from above), of type $(K \rightharpoonup \mu\mathsf{F})^{(K \rightharpoonup \mathsf{F}\,K)}$, is nothing but the $\mathsf{F}$-anamorphism combinator:

$$\begin{tikzcd} \mu\mathsf{F} & \mathsf{F}(\mu\mathsf{F}) \arrow[l, "in"'] \\ K \arrow[u, "\overline{F} H"] \arrow[r, "H"'] & \mathsf{F}\,K \arrow[u, "\mathsf{F}(\overline{F} H)"'] \end{tikzcd}$$

$$\begin{aligned} \overline{F} &= [\!(\_)\!]_{\mathsf{F}} \\ \overline{F}\ H &= \mu X.\ in \cdot (\mathsf{F}\,X) \cdot H \end{aligned}$$

## Partiality of implementation

$F(\sigma, k) = (\overline{F}\sigma)k$ will be undefined wherever

- $k \notin \delta\,\sigma$
- $\sigma$ is not "closed" over itself (see below)
- $\sigma$ is non-well-founded (see below)

Thus concrete invariant

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

In order to define $closed\ \sigma$ and $wellf\ \sigma$ we need $\sigma$'s accessibility relation $\prec_\sigma$ (next slide).

## Accessibility and membership

Accessibility relation for $\sigma$:

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

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

where $K \xleftarrow{\in_{\mathsf{F}}} \mathsf{F}\, K$ extends $K \xleftarrow{\in} \mathcal{P}K$ inductively over polynomial functors, as follows:

- Constant and identity functors:

$$\begin{aligned} \in_C &\stackrel{\mathrm{def}}{=} \bot \\ \in_{\lambda X.X} &\stackrel{\mathrm{def}}{=} id \end{aligned}$$

## Membership (continued)

- Product and coproduct

$$\begin{aligned} \in_{\mathsf{F}\times\mathsf{G}} &\stackrel{\mathrm{def}}{=} (\in_{\mathsf{F}} \cdot \pi_1) \cup (\in_{\mathsf{G}} \cdot \pi_2) \\ \in_{\mathsf{F}+\mathsf{G}} &\stackrel{\mathrm{def}}{=} [\in_{\mathsf{F}} , \in_{\mathsf{G}}] \end{aligned}$$

- Functor composition

$$\in_{\mathsf{F}\cdot\mathsf{G}} \stackrel{\mathrm{def}}{=} \in_{\mathsf{G}} \cdot \in_{\mathsf{F}}$$

- Type functors: just an example,

$$\in_{X^\star} \stackrel{\mathrm{def}}{=} \in \cdot\, elems$$

## Example

Recall $\mathsf{F}\, X = Symbol + Symbol \times X^{\star}$

$$
\begin{array}{cl}
& \in_{Symbol+Symbol\times X^{\star}} \\
= & \qquad \{\ \in \text{ for coproduct bifunctor } \} \\
& [\in_{Symbol}\ , \in_{Symbol\times X^{\star}}] \\
= & \qquad \{\ \in \text{ for constant and product (bi)functors } \} \\
& [\bot\ , (\in_{Symbol} \cdot \pi_1) \cup (\in_{X^{\star}} \cdot \pi_2)] \\
= & \qquad \{\ \in \text{ for constant and identity functor } \} \\
& [\bot\ , (\bot \cdot \pi_1) \cup (\in \cdot\, elems \cdot \pi_2)] \\
= & \qquad \{\ \bot \text{ and } [R\ , S] = (R \cdot i_1^{\circ}) \cup (S \cdot i_2^{\circ})\ \} \\
& \in \cdot\, elems \cdot \pi_2 \cdot i_2^{\circ}
\end{array}
$$

## Example (pointwise)

$$
\begin{array}{cl}
& k \in_{Symbol+Symbol\times X^{\star}} x \\
\equiv & \qquad \{\ \text{calculation above } \} \\
& k(\in \cdot\, elems \cdot \pi_2 \cdot i_2^{\circ})x \\
\equiv & \qquad \{\ \text{relational composition } \} \\
& k(\in \cdot\, elems \cdot \pi_2)(a,l)\ \wedge\ x = i_2(a,l) \\
\equiv & \qquad \{\ \text{trivia } \} \\
& k \in (elems\ l)\ \wedge\ x = i_2(a,l)
\end{array}
$$

## Another example

Let F $X = 1 + A \times X$. Then,

$$
\begin{array}{ll}
 & \in_{1+A\times X} \\
= & \quad \{\ \in \text{ for coproduct bifunctor } \} \\
 & [\in_1 \ , \in_{A\times X}] \\
= & \quad \{\ \in \text{ for constant and product (bi)functors } \} \\
 & [\bot \ , (\in_A \cdot \pi_1) \cup (\in_{\lambda X.X} \cdot \pi_2)] \\
= & \quad \{\ \in \text{ for constant and identity functor } \} \\
 & [\bot \ , (\bot \cdot \pi_1) \cup (id \cdot \pi_2)] \\
= & \quad \{\ \bot \text{ and } [R \ , S] = (R \cdot i_1^\circ) \cup (S \cdot i_2^\circ) \ \} \\
 & \pi_2 \cdot i_2^\circ
\end{array}
$$

## Example (pointwise)

$$
\begin{array}{ll}
 & k \in_{1+A\times X} x \\
\equiv & \quad \{\ \text{calculation above } \} \\
 & k(\pi_2 \cdot i_2^\circ)x \\
\equiv & \quad \{\ \text{relational composition } \} \\
 & k(\pi_2)(a, k') \ \wedge \ x = i_2(a, k') \\
\equiv & \quad \{\ \text{trivia } \} \\
 & x = i_2(a, k') \ \wedge \ k = k' \\
\equiv & \quad \{\ \text{trivia } \} \\
 & x = i_2(a, k)
\end{array}
$$

## Accessibility (linear example)

Pointer reachability in case of a "linear" heap $(1 + A \times K) \xleftarrow{\sigma} K$:

$$k_1 \prec_\sigma k_2 \quad \equiv \quad k_2 \in \delta\,\sigma \;\wedge\; (\sigma\ k_2) = i_2(a, k_1)$$

In a drawing:

$$\sigma = \begin{array}{|c||c|c|} \multicolumn{1}{c}{K} & \multicolumn{1}{c}{A} & \multicolumn{1}{c}{K} \\ k_1 & \ldots & k_0 \\ \hline k_2 & a & k_1 \\ \hline & \ldots & \ldots \end{array} \qquad \begin{array}{l} k_1 \prec_\sigma k_2 \\ k_0 \prec_\sigma k_1 \end{array}$$

## Closure and wellfoundedness

Let $\prec_\sigma^+$ denote the transitive closure of $\prec_\sigma$. Then we define

$$closed\ \sigma \quad = \quad \rho \prec_\sigma^+ \subseteq \delta\,\sigma$$

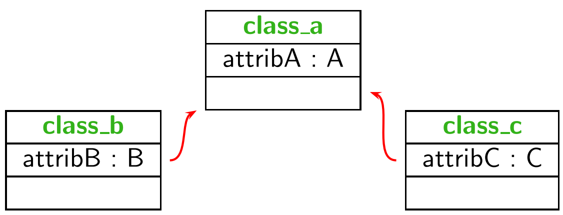
that is, all reacheable $k$ are defined, and

$$wellf\ \sigma \quad = \quad (\prec_\sigma^+) \cap id = \bot$$

that is, $\prec_\sigma^+$ is irreflexive (no cycles, no looping)

## O.O. Data Implementation

UML:

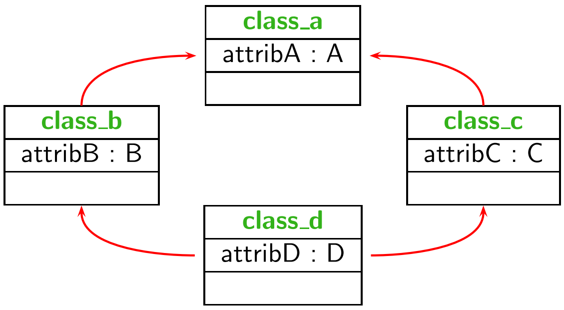



Formal model: $K \rightharpoonup Structure$ where

$$\begin{array}{rcl} Structure & = & A + A \times B + A \times C \\ & \cong & A \times (1 + B + C) \end{array}$$

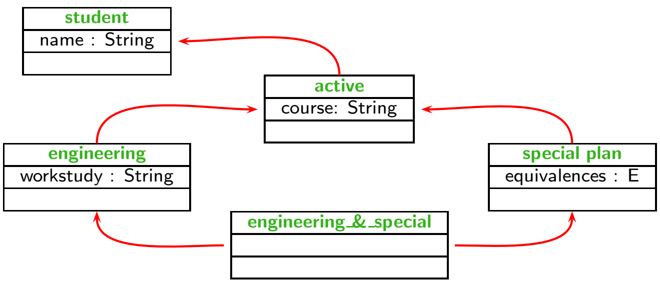
$$K \rightharpoonup (A + A \times B)$$

## Multiple inheritance




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

## Example




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