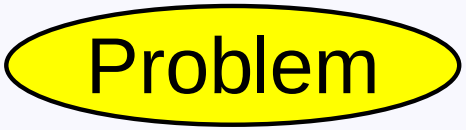
An Introduction to Relational Hylomorphisms

DI/UM, 2002

José N. Oliveira
Dept. Informatica
Universidade do Minho, 4700 Braga, Portugal
jno@di.uminho.pt

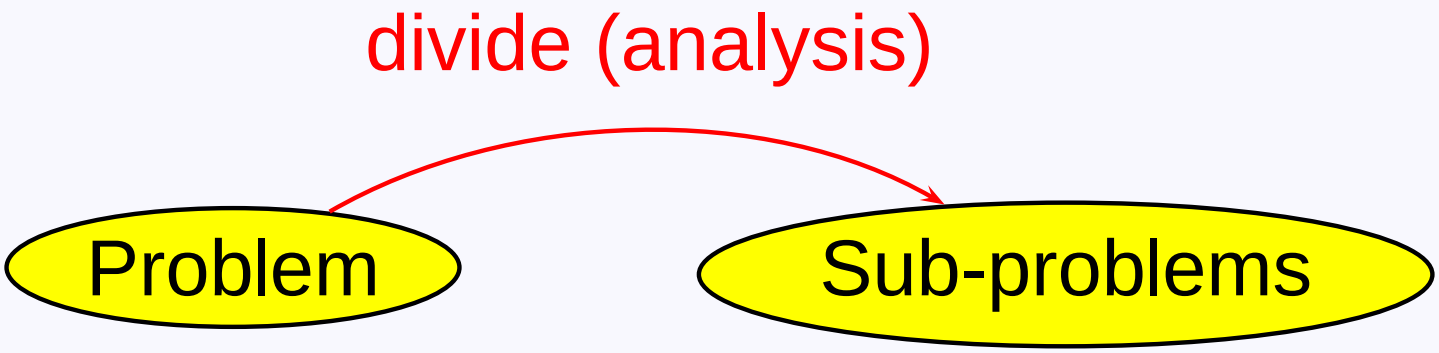
“How” does one specify?

General problem solving strategy?

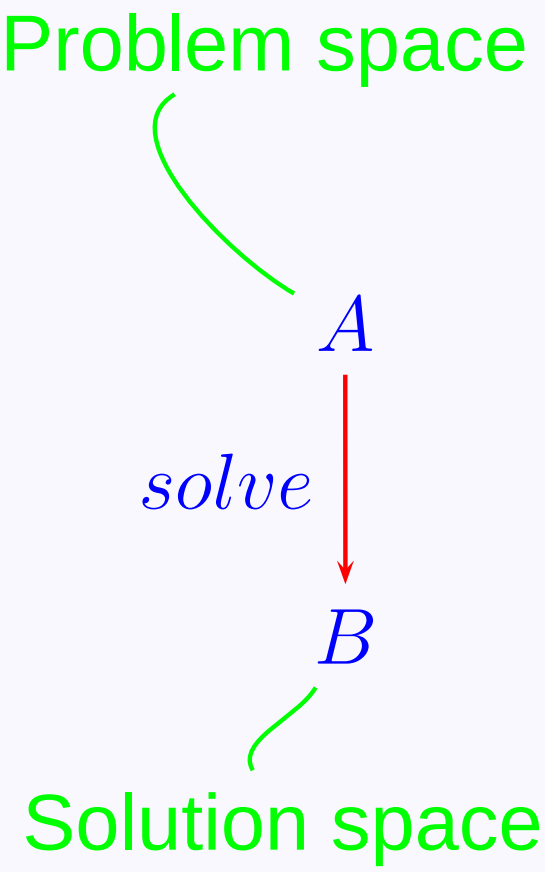


“How” does one specify?

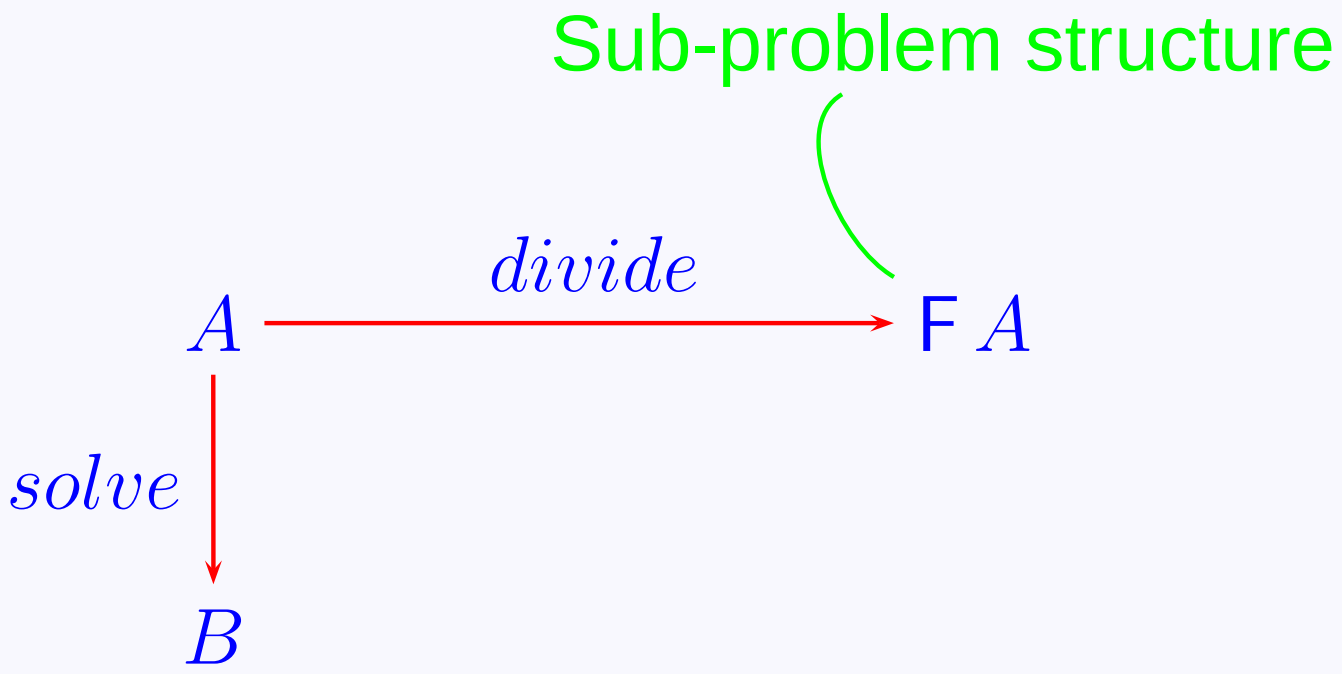
Divide-and-conquer:



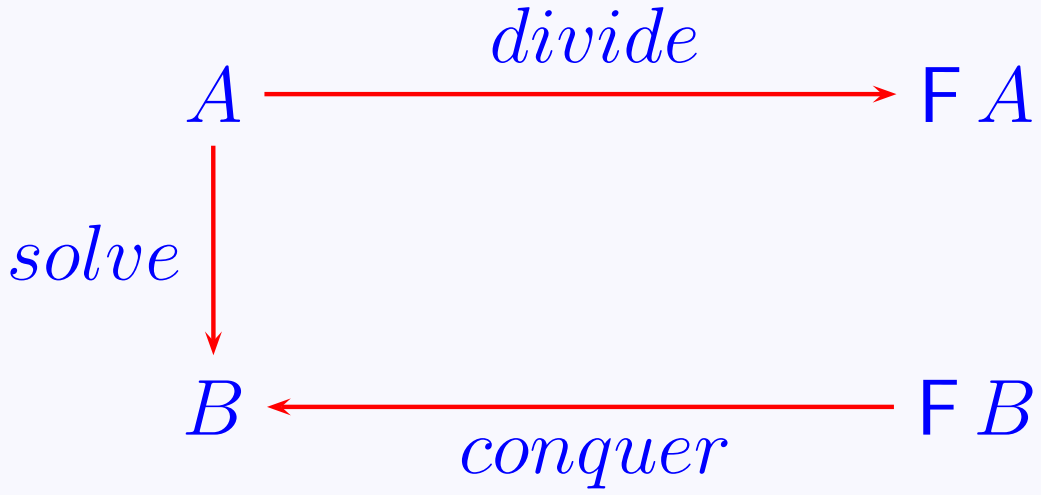
Divide-and-conquer (formally)



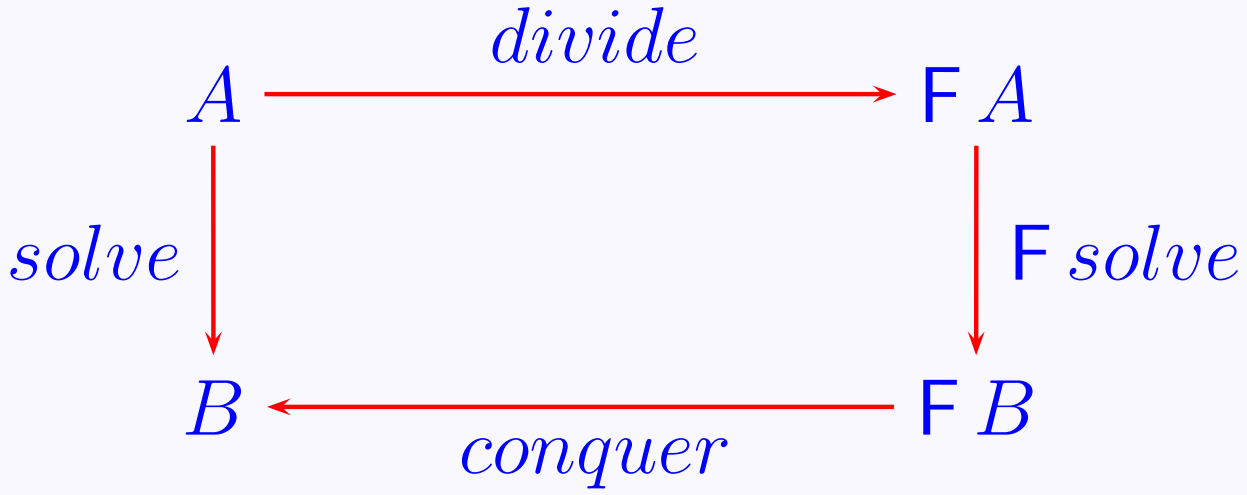
Divide-and-conquer (formally)



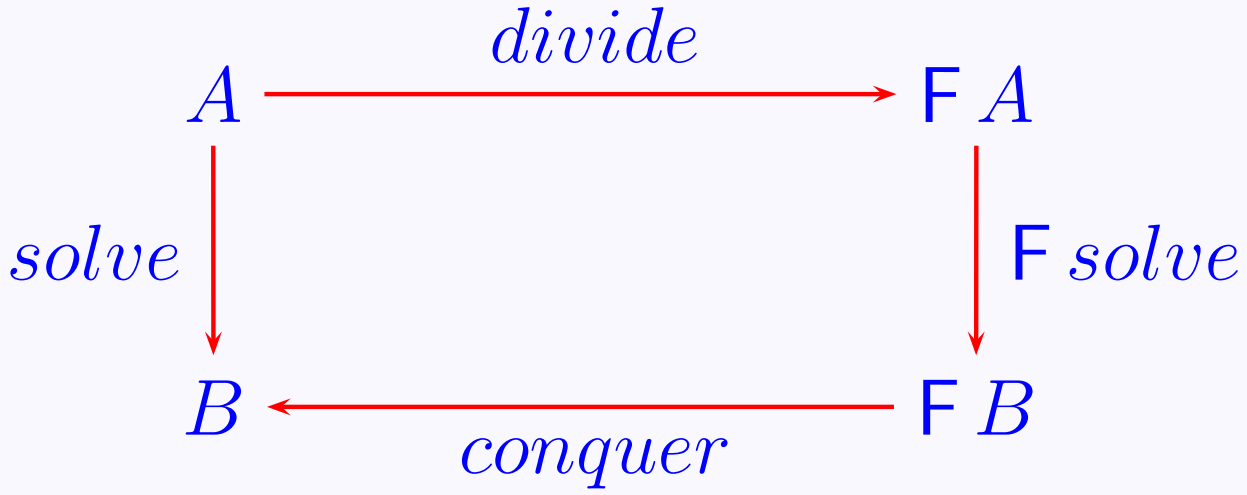
Divide-and-conquer (formally)



Divide-and-conquer (formally)



Divide-and-conquer (formally)



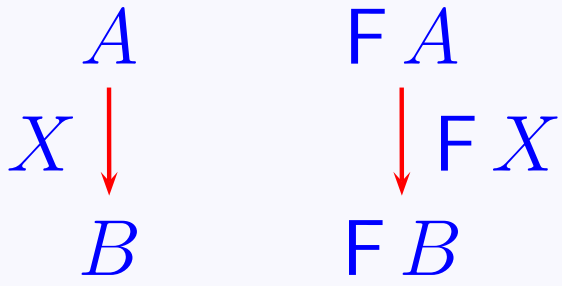
Questions:

- What are the mathematics of $divide$, $conquer$, $solve$?
- What do F A, F $solve$ mean?

Relators

Symbol F is overloaded:

- F A means a (parametric) **datatype**, *e.g.* $\mathcal{P}A$ — `set of A` in VDM-SL;
- F X means a relation



Eg. $\mathcal{P}X$ will relate every $s \subseteq A$ to

$$\{b \in B \mid \exists a \in s.bXa\}$$

Properties of relators

Every **relator** F is monotone,

$$R \subseteq S \;\Rightarrow\; (\mathsf{F}\, R) \subseteq (\mathsf{F}\, S)$$

and commutes with $(\cdot)$, $(\_)^\circ$ and id:

$$\begin{aligned} \mathsf{F}\,(R \cdot S) &= (\mathsf{F}\, R) \cdot (\mathsf{F}\, S) \\ \mathsf{F}\,(R^\circ) &= (\mathsf{F}\, R)^\circ \\ \mathsf{F}\, id &= id \end{aligned}$$

Terminology:

Properties of relators

Every **relator** F is monotone,

$$R \subseteq S \;\Rightarrow\; (\mathsf{F}\, R) \subseteq (\mathsf{F}\, S)$$

and commutes with $(\cdot)$, $(\_)^\circ$ and id:

$$\begin{aligned} \mathsf{F}\,(R \cdot S) &= (\mathsf{F}\, R) \cdot (\mathsf{F}\, S) \\ \mathsf{F}\,(R^\circ) &= (\mathsf{F}\, R)^\circ \\ \mathsf{F}\, id &= id \end{aligned}$$

Terminology:

$A \xleftarrow{R} \mathsf{F}\, A$ is called an F-**algebra**

Properties of relators

Every **relator** F is monotone,

$$R \subseteq S \;\Rightarrow\; (\mathsf{F}\, R) \subseteq (\mathsf{F}\, S)$$

and commutes with $(\cdot)$, $(\_)^\circ$ and id:

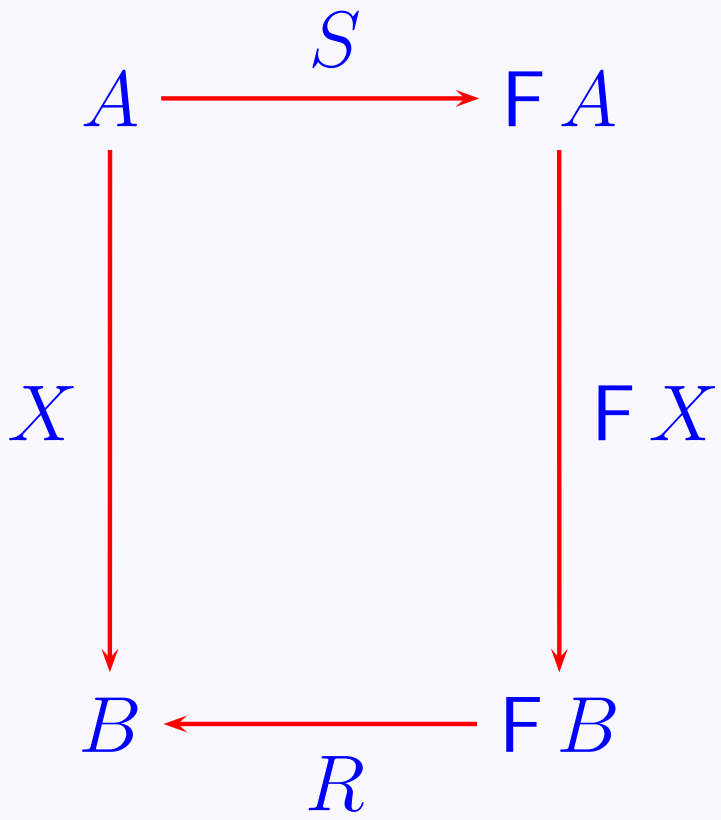
$$\begin{aligned} \mathsf{F}\,(R \cdot S) &= (\mathsf{F}\, R) \cdot (\mathsf{F}\, S) \\ \mathsf{F}\,(R^\circ) &= (\mathsf{F}\, R)^\circ \\ \mathsf{F}\, id &= id \end{aligned}$$

Terminology:

$A \xrightarrow{S} \mathsf{F}\, A$ is called an F-**coalgebra**

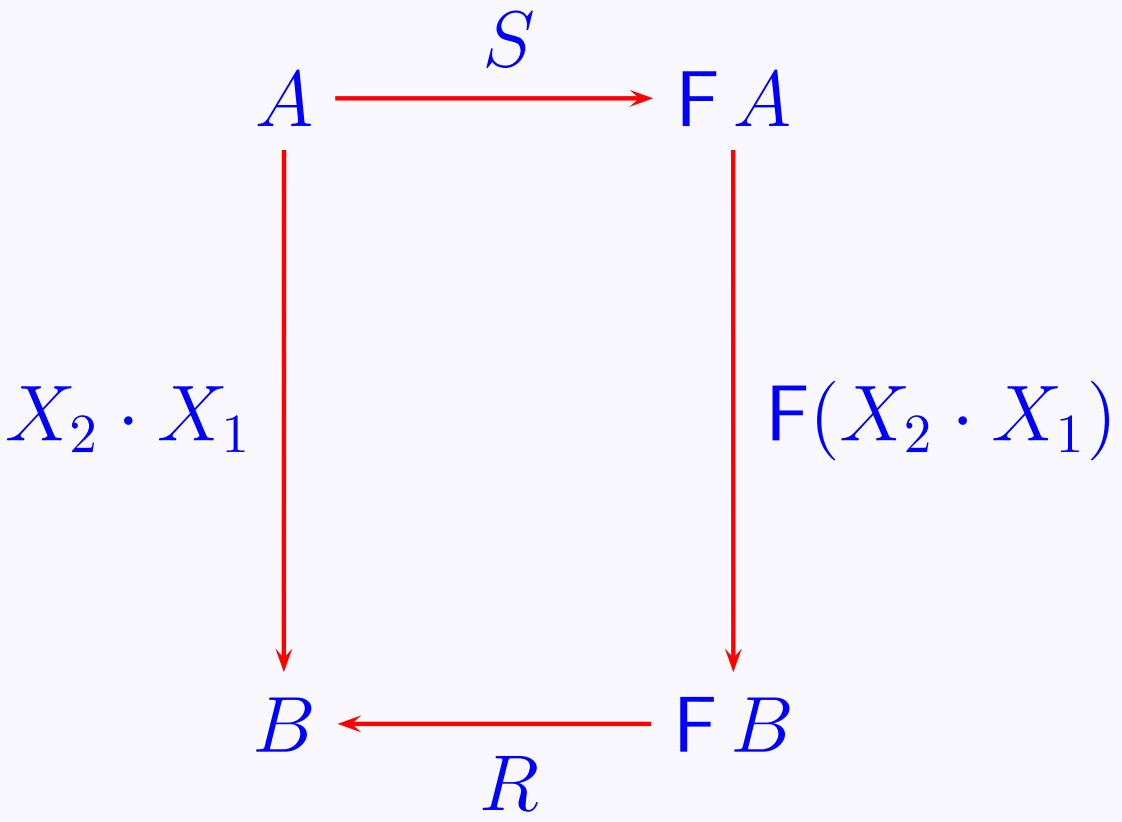
Back to divide-and-conquer

Hylo equation $X = R \cdot (\mathsf{F}\, X) \cdot S$:



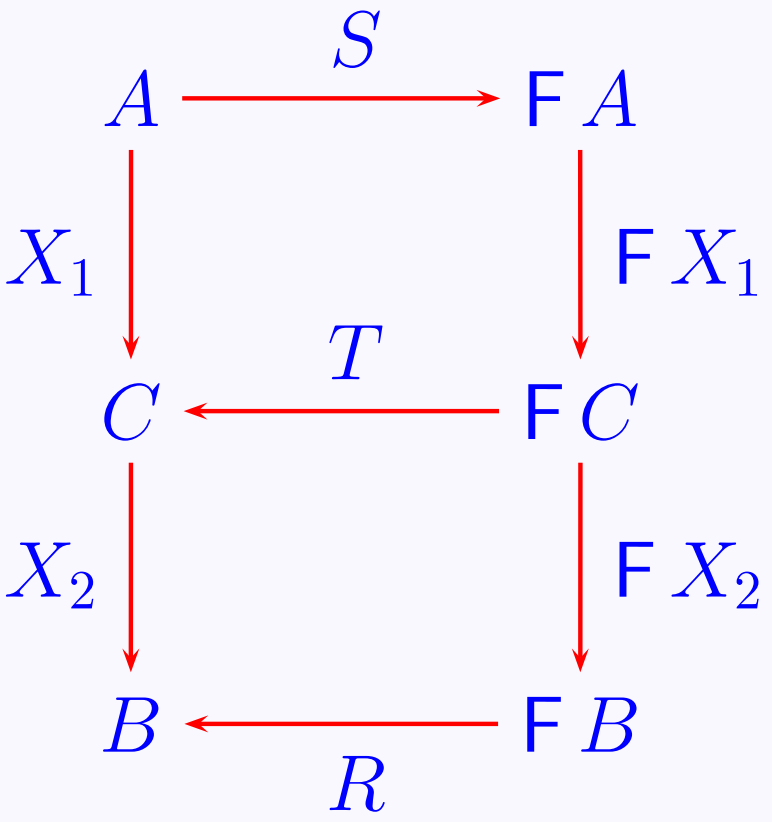
Back to divide-and-conquer

Assume X decomposes into $X_1 \cdot X_2$:

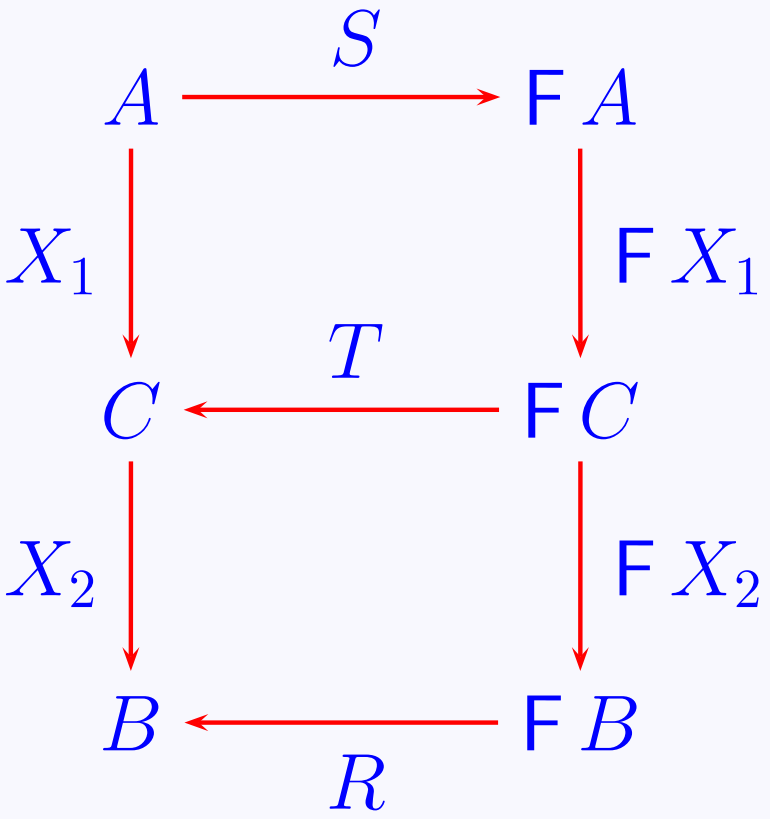


Back to divide-and-conquer

F commutes with composition:



Back to divide-and-conquer



What about C $\xleftarrow{T}$ F C?

Hylo factorization (1)

A standard result of the relational calculus establishes conditions on S, R and T for

$$\mu X.(R \cdot \mathsf{F}\, X \cdot S) \;=\; X_2 \cdot X_1$$

to be a unique solution:

Hylo factorization (1)

A standard result of the relational calculus establishes conditions on S, R and T for

$$\mu X.(R \cdot \mathsf{F}\, X \cdot S) \;=\; X_2 \cdot X_1$$

to be a unique solution:

- S is required to be "**well-founded**" — the "size" of a sub-problem generated by S is strictly smaller than its source, cf. **termination**.

Hylo factorization (1)

A standard result of the relational calculus establishes conditions on S, R and T for

$$\mu X.(R \cdot \mathsf{F}\, X \cdot S) \;=\; X_2 \cdot X_1$$

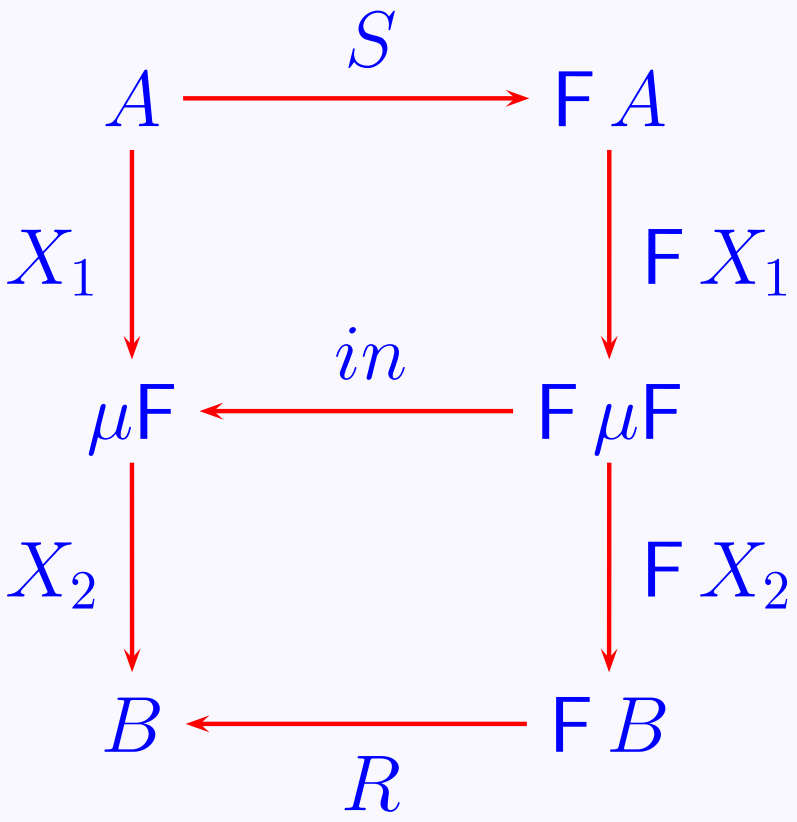
to be a unique solution:

- S is required to be "**well-founded**" — the "size" of a sub-problem generated by S is strictly smaller than its source, cf. **termination**.
- T is required to be **bijective** over the datatype which is inductively defined by F, that is, $C = \mu\mathsf{F}$.

Hylo factorization (2)

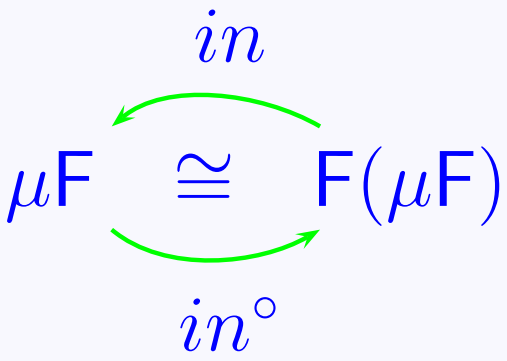
$C = \mu\mathsf{F}$, least fixpoints

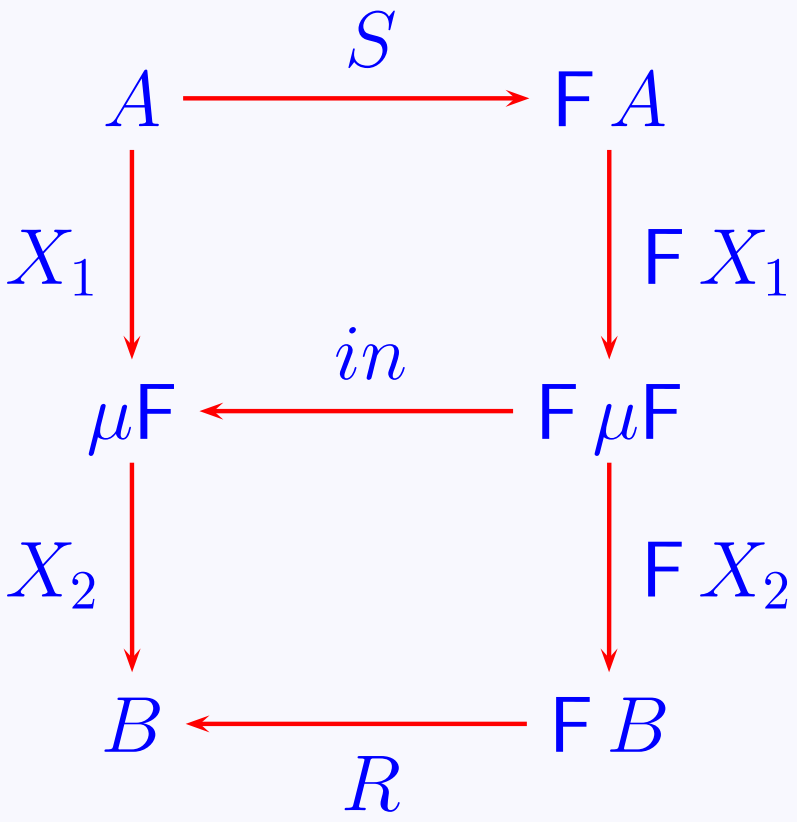
$$\mu\mathsf{F} \quad \cong \quad \mathsf{F}(\mu\mathsf{F})$$



Hylo factorization (2)

$C = \mu\mathsf{F}$, least fixpoints





Least fixpoints in VDM-SL

μF is the type one "defines" by writing domain equations, *e.g.*

```
LTree =  Leaf | Node ;
Leaf  :: value: int ;
Node  :: left: LTree right: LTree ;
```

where we define $\texttt{LTree} = \mu\mathsf{F}$ for

$$\mathsf{F}\, X \stackrel{\text{def}}{=} \texttt{int} \mid X * X$$

Is this a relator?

Polynomial relators

Instead of

$$\mathsf{F}\, X \stackrel{\text{def}}{=} \texttt{int} \mid X * X$$

write

$$\mathsf{F}\, X \stackrel{\text{def}}{=} int + X^2$$

where

Polynomial relators

Instead of

$$\mathsf{F}\, X \stackrel{\text{def}}{=} \texttt{int} \mid X * X$$

write

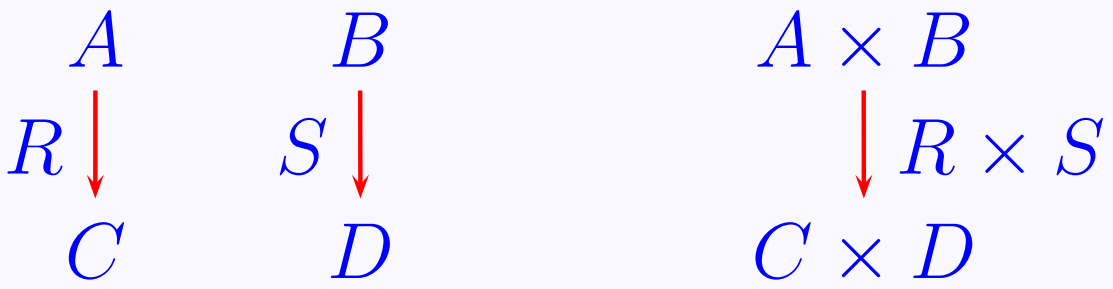
$$\mathsf{F}\, X \stackrel{\text{def}}{=} int + X^2$$

where

- X^2 means the same as $X \times X$
- **sum** and **product** are standard binary relators.

Binary relators $\times$ and $+$

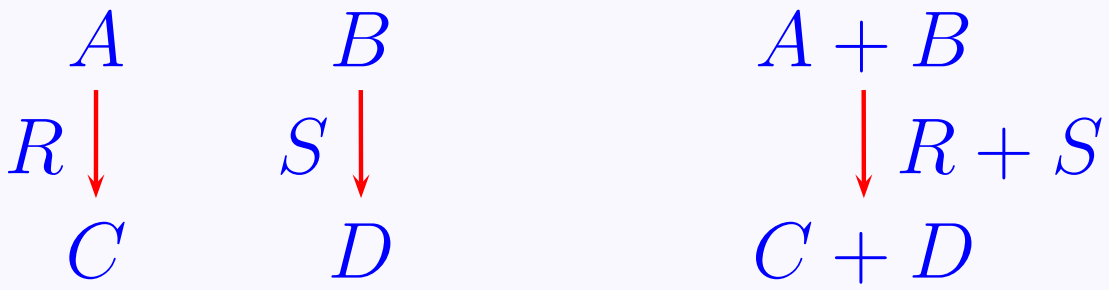
Product:



$$\langle b, d\rangle (R \times S) \langle a, c\rangle \equiv (bRa) \wedge (dSc)$$

Binary relators $\times$ and $+$

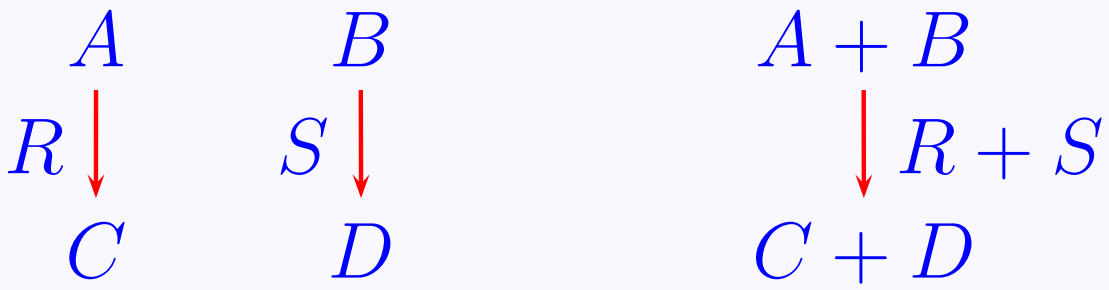
Sum:



$$R + S = [i_1 \cdot R, i_2 \cdot S]$$

Binary relators $\times$ and $+$

Sum:



$$R + S = [i_1 \cdot R, i_2 \cdot S]$$

where

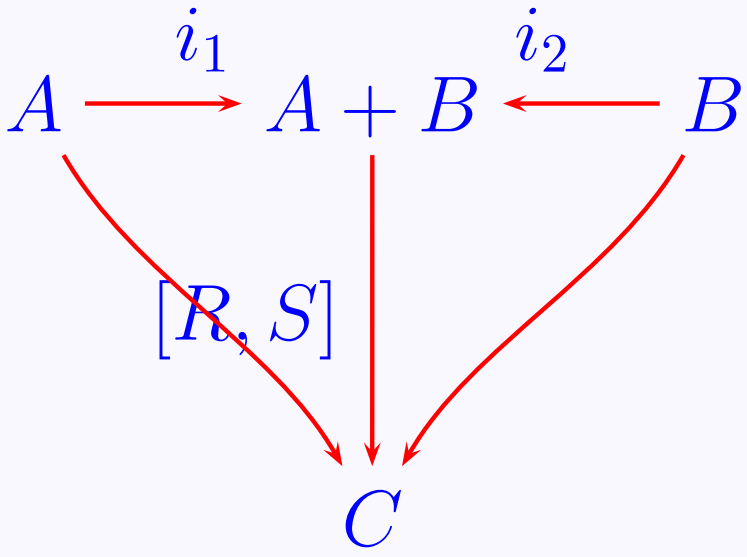
$$A + B = \{i_1\, a \mid a \in A\} \cup \{i_2\, b \mid b \in B\}$$

i_1,i_2 — arbitrary (but consistent) disjoint injections.

Case analysis combinator

$$c[R, S](i_1\, a) \equiv cRa$$
$$c[R, S](i_2\, b) \equiv cSb$$

In a diagram:



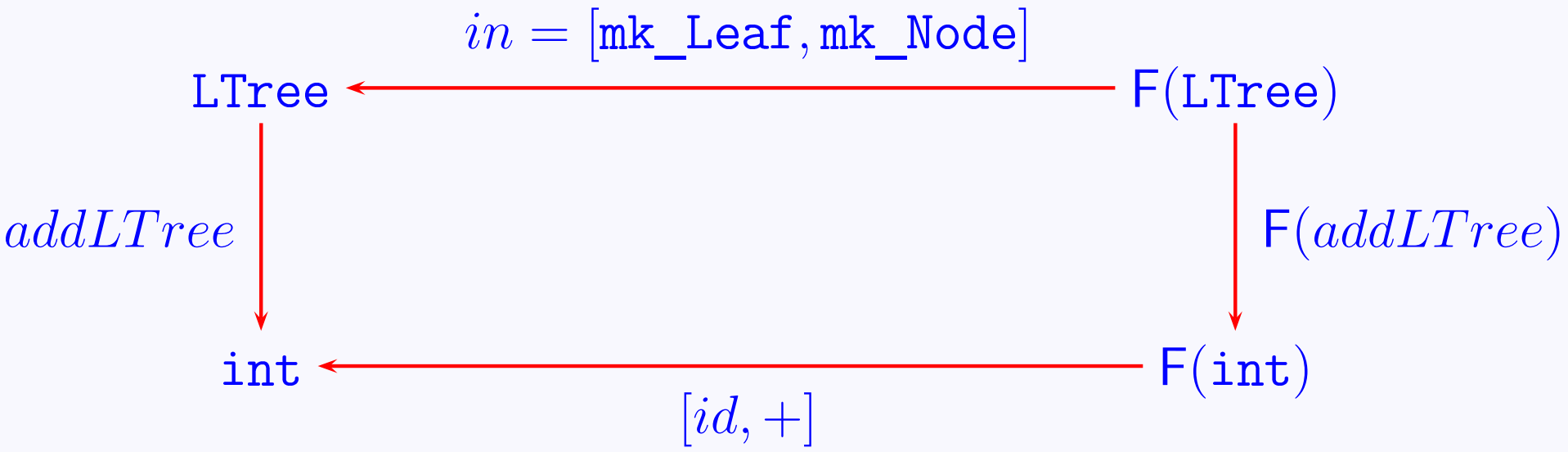
LTree example

whereby

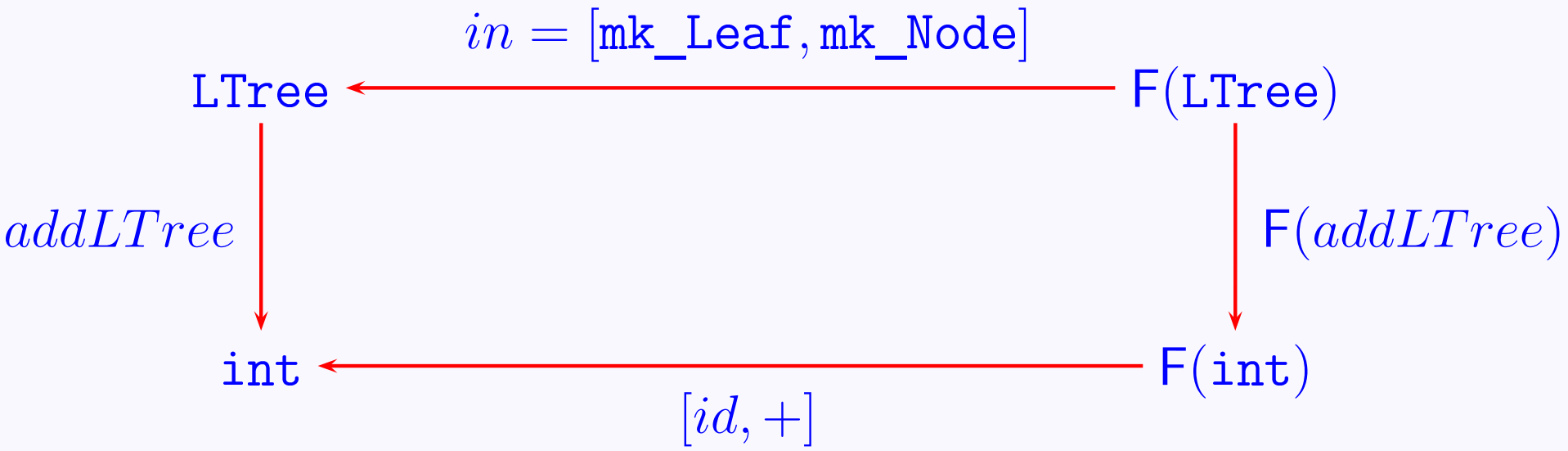
```
addLTree: LTree -> int
addLTree(t) ==
   cases t :
      mk_Leaf(i) -> i ,
      mk_Node(t1,t2) -> addLTree(t1) + addLTree(t2)
   end;
```

can be depicted in a diagram:

Remarkable facts about $\mu\mathsf{F}$



Remarkable facts about μF



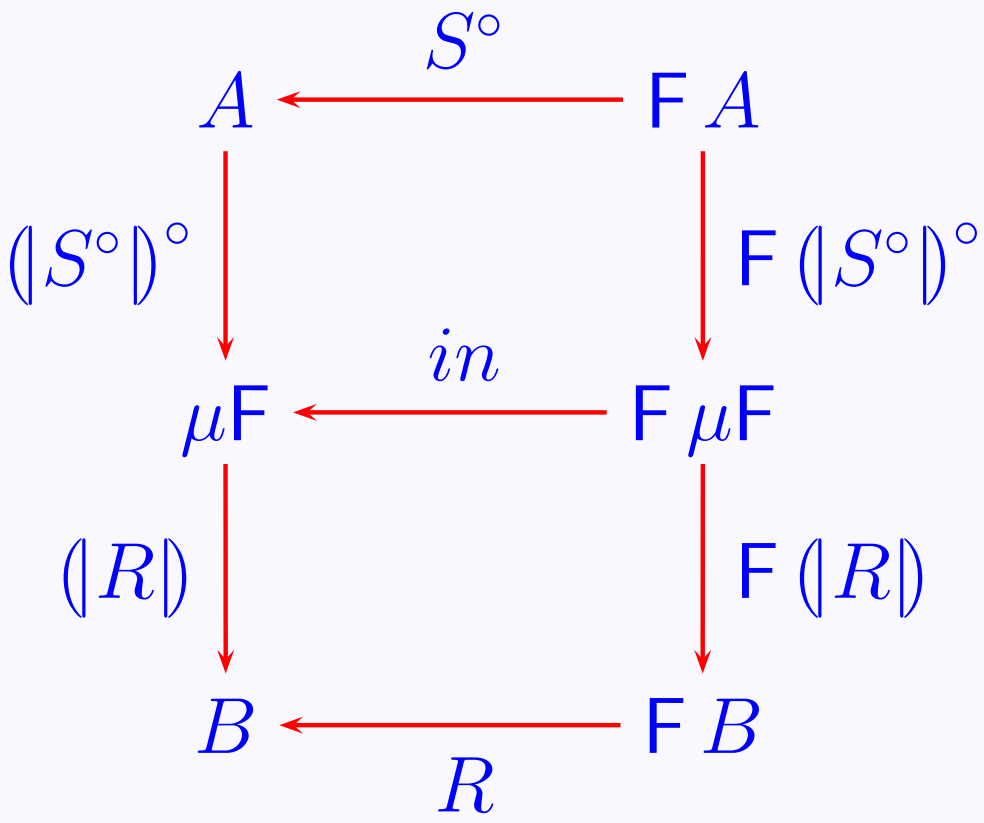
Facts:

- $addLTree$ is **uniquely** determined by $[id, +]$ and one writes $addLTree = (\!|[id, +]|\!)$
- Given $A \xleftarrow{R}$ F A, $(\!|R|\!)$ is called the "inductive extension of R, "fold R" or "**catamorphism** of R".

Hylo-factorization Theorem

Using $(\!|\_|\!)$ notation:

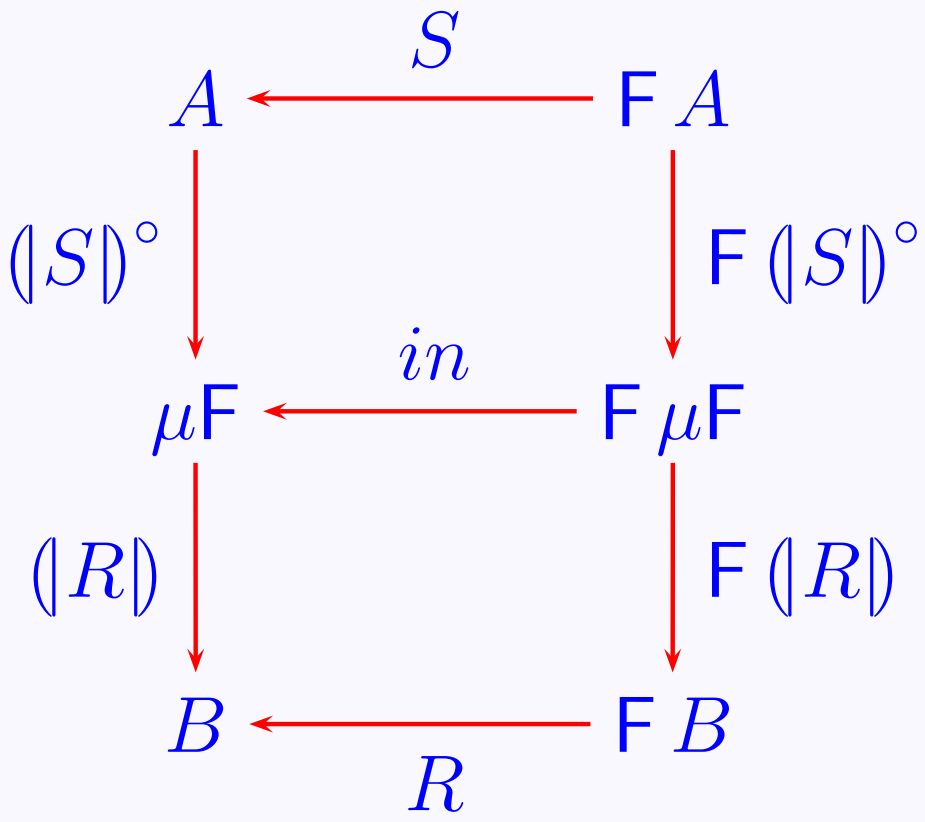
$$\mu X.(R \cdot \mathsf{F}\, X \cdot S) = (\!|R|\!) \cdot (\!|S^\circ|\!)^\circ$$



Hylo-factorization Theorem

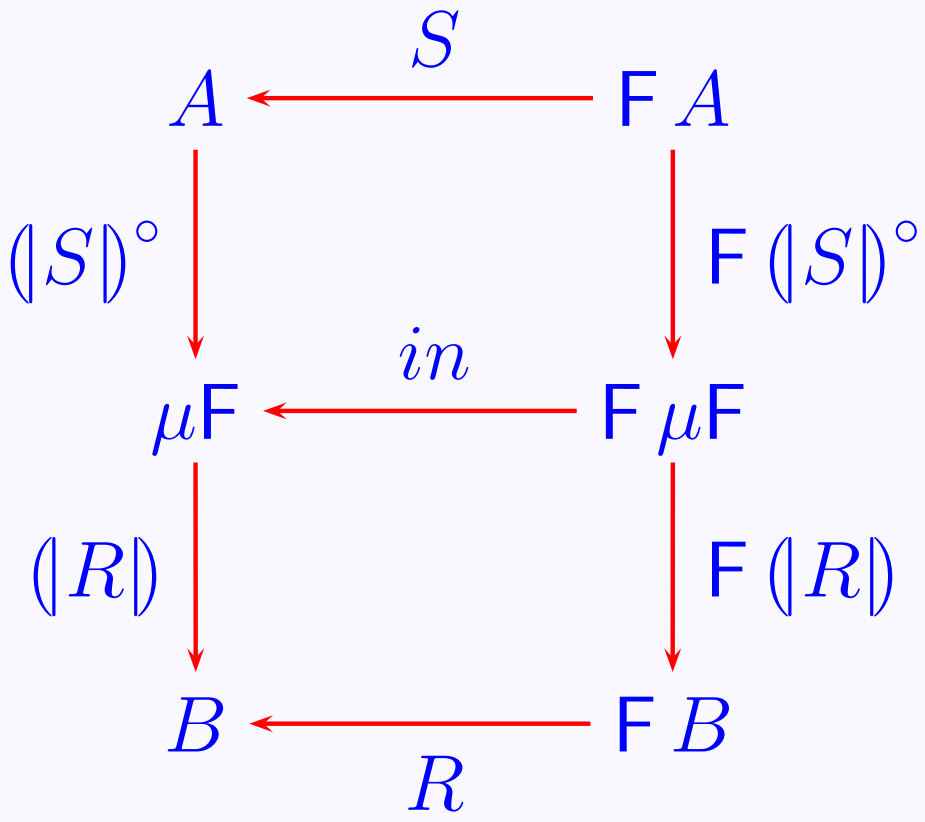
Taking converses:

$$\mu X.(R \cdot \mathsf{F}\, X \cdot S^\circ) = (\!|R|\!) \cdot (\!|S|\!)^\circ$$



Hylo-factorization Theorem

Entire/simple factorization if both R and S° are entire/simple (=S surjective/injective)



Example: `mergeSort`

```
mergeSort : seq of int -> seq of int
mergeSort (l) ==
  cases l :
    [] -> l ,
    [e] -> l ,
    others -> let l1 ^ l2 in set {l}
                  be st abs(len l1 - len l2) < 2
                in let l_l = mergeSort (l1),
                       l_r = mergeSort (l2)
                   in lmerge (l_l, l_r)
  end;
```

Case analysis following Hu *et al*'s algorithm [HIT96b]:

mergeSort

From

$$mergeSort = [singl, lmerge \cdot (mergeSort \times mergeSort)] \cdot S$$

$$\equiv \quad \{ \text{ law } [R \cdot T, S \cdot U] = [R, S] \cdot (T + U) \}$$

$$mergeSort = [singl, lmerge] \cdot (id + mergeSort \times mergeSort) \cdot S$$

infer

- $\mathsf{F}\, f = id + f \times f$
- $\mathsf{F}\, X = \mathtt{int} + X \times X$
- $\mu \mathsf{F} = \mathtt{LTree}$

mergeSort

mergeSort's ***formula***:

$$mergeSort = (\!|[singl, lmerge]|\!) \cdot (\!|[singl, pconc]|\!)^{\circ}$$

where $singl = \lambda e.[e]$ and

```
pconc :  seq of int * seq of int -> seq of int
pconc(l1,l2) ==  l1 ^ l2
pre abs (len l1 - len l2) < 2 ;
```

Factorization helps in understanding the *"Equal-size, Easy Split, Hard Join"* classification of the algorithm.

quickSort

```
quickSort : seq of int -> seq of int
quickSort (l) ==
  cases l:
    [] -> [],
    -^[x]^- -> quickSort ([y | y in set elems l & y < x]) ^ [x] ^
               quickSort ([y | y in set elems l & y > x])
  end
```

Here F $X = 1 + \texttt{int} \times X^2$, so μF is BTree (binary search tree):

$$quickSort = (\!|[\underline{[\,]}, inord]|\!) \cdot (\!|[\underline{[\,]}, pinord]|\!)^{\circ}$$

where

quickSort

```
inord: int * (seq of int * seq of int) -> seq of int
inord(x,mk_(l,r)) == l ^ [x] ^ r;

pinord: int * (seq of int * seq of int) -> seq of int
pinord(x,mk_(l,r)) == inord(x,mk_(l,r))
pre forall y in set elems l & y < x and
    forall y in set elems r & y > x;
```

- *quickSort* = `BTree` *in-order (conquer)* following the converse of a *partial* in-order traversal *(divide).*
- *Divide* does the hard job of ensuring that the intermediate tree is bi-ordered.
- Classification: “Equal-size, Hard Split, Easy Join”.

Virtual data-structuring

- Particular choice of F for sub-problem organization induces **intermediate** type μF.

  This is made explicit by hylo-factorization.

Virtual data-structuring

- Particular choice of F for sub-problem organization induces **intermediate** type μF.

  This is made explicit by hylo-factorization.

- **Intermediate** data-structure saves the outcome of a "one go" ***divide*** step $(\!|S|\!)^\circ$ and passes it on to the ***conquer*** step $(\!|R|\!)$ for processing.

Virtual data-structuring

- Particular choice of F for sub-problem organization induces **intermediate** type μF.

  This is made explicit by hylo-factorization.
- **Intermediate** data-structure saves the outcome of a "one go" ***divide*** step $(\!|S|\!)^\circ$ and passes it on to the ***conquer*** step $(\!|R|\!)$ for processing.
- In general, people **"fuse"** things very early in design, thus ***virtualizing*** this structure.
- Factorization helps in spec **understanding** and **classification**.