Algebra of Programming using Dependent types

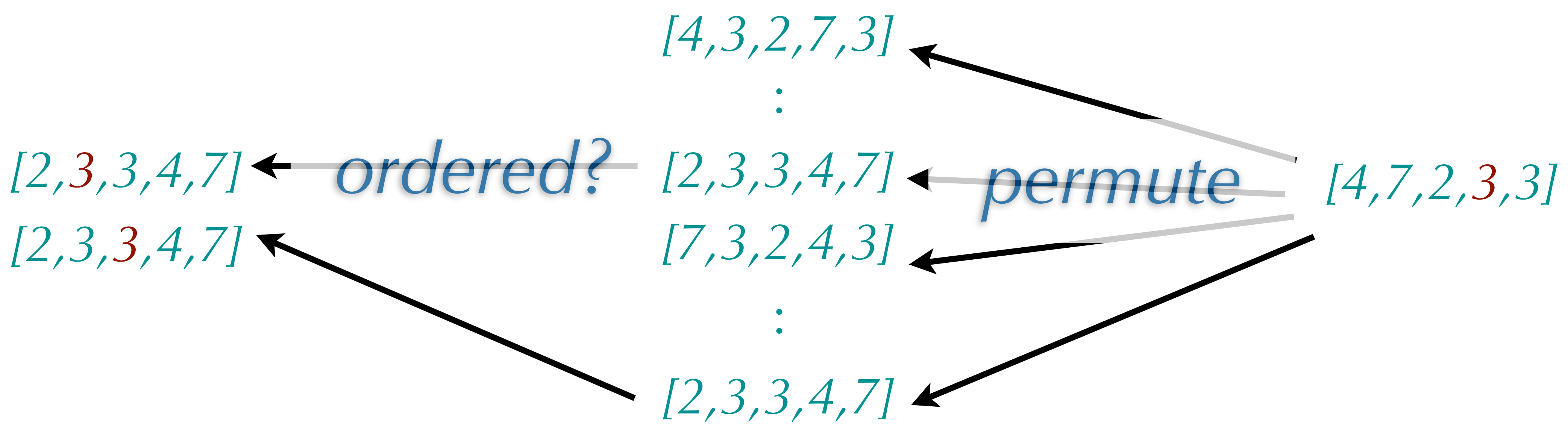
Shin-Cheng Mu, Hsiang-Shang Ko, and Patrik Jansson

Agda Intensive Meeting
Sendai, 27 Nov. 2008

In one sentence: encoding *relational program derivation* in Agda.

Program Derivation

- Refinement from specifications to programs, thereby ensure correctness.
    - Programming in the small.
- An (input-output) relational specification of sorting:
    - *sort* = *ordered?* ∘ *permute*



Program Derivation

- A typical derivation in the AoP style:

$$
\begin{aligned}
& \mathit{ordered?} \circ \mathit{permute} \\
= \quad & \{ \text{ since } \mathit{permute} \text{ is a fold } \} \\
& \mathit{ordered?} \circ \mathit{foldr\ combine\ []} \\
\supseteq \quad & \{ \text{ fold fusion, see proof 1 } \} \\
& \mathit{foldr\ (ordered \circ combine)\ []} \\
\supseteq \quad & \{ \text{ since } \mathit{ordered} \circ \mathit{combine} \supseteq \mathit{insert} \} \\
& \mathit{foldr\ insert\ []}
\end{aligned}
$$

- Typically, all done by hand.
- Several tools has been developed, none of them in active use now. ....AFAIK.

Relational Derivation in Agda

```
sort-der :  ∃ (\f → ordered? ○ permute ⊒ fun f )
sort-der = (_ ,
      (⊒-begin
          ordered? ○ permute
       ⊒⟨ ○-monotonic-r permute-is-fold ⟩
          ordered? ○ foldR combine nil
       ⊒⟨ foldR-fusion-⊒ ordered? { }0 { }1  ⟩
         { }2
      ))
```

Relational Derivation in Agda

```
sort-der :  ∃ (\f → ordered? ○ permute ⊒ fun f )
sort-der = (_ ,
      (⊒-begin
          ordered? ○ permute
        ⊒⟨ ○-monotonic-r permute-is-fold ⟩
          ordered? ○ foldR combine nil
        ⊒⟨ foldR-fusion-⊒ ordered? ins-step ins-base ⟩
          foldR (fun (uncurry insert)) nil
        ⊒⟨ foldR-to-foldr insert [] ⟩
          fun (foldr insert [])
        ⊒∎))
isort : [ Val ] → [ Val ]
isort = proj₁ sort-der
```

Relations

- $R : B \leftarrow A$: subset of $B \times A$.
- $(c,a) \in R \circ S$ iff $\exists\, b.\ (c,b) \in R \wedge (b,a) \in S$.
- Functions: $(b,a) \in f \wedge (b', a) \in f$ implies $b = b'$.
- Converse: $(b,a) \in R^{\smile}$ iff. $(a,b) \in R$.

Fold and Hylomorphism

- Fold can be generalised to relations:
  - $subseq = foldr\ (cons \cup proj_2)\ nil$.
  - $(xs, [1,2,3]) \in subseq$ where xs may be $[]$, $[1]$, $[1,2]$, $[1,3]$,.. etc.

- Hylomorphism: a fold after converse of a fold: $foldr\ f\ e \circ (foldr\ g\ d)^{\smile}$. (1)
  - Bird & de Moor talked about inductive types only.
  - (1) is the unique solution of $X = f \circ (1 + id \times X) \circ g^{\smile}$ under certain conditions...

Concise Specifications

- $permute = bagify^{\smile} \circ bagify$.
- Building a binary tree from its traversals: $\langle inorder, preorder\rangle^{\smile}$,
  - $\langle f,g\rangle\,(x,y) = (f\ x,\ g\ y)$.
  - It expands to:
    - $(inorder^{\smile} \circ proj_1) \cap (preorder^{\smile} \circ proj_2)$.

$$\begin{tikzcd}
 & xs \arrow[dl, "inorder^{\smile}"'] & \\
t & & (xs,\ ys) \arrow[ul, "proj_1"'] \arrow[dl, "proj_2"] \\
 & ys \arrow[ul, "preorder^{\smile}"] &
\end{tikzcd}$$

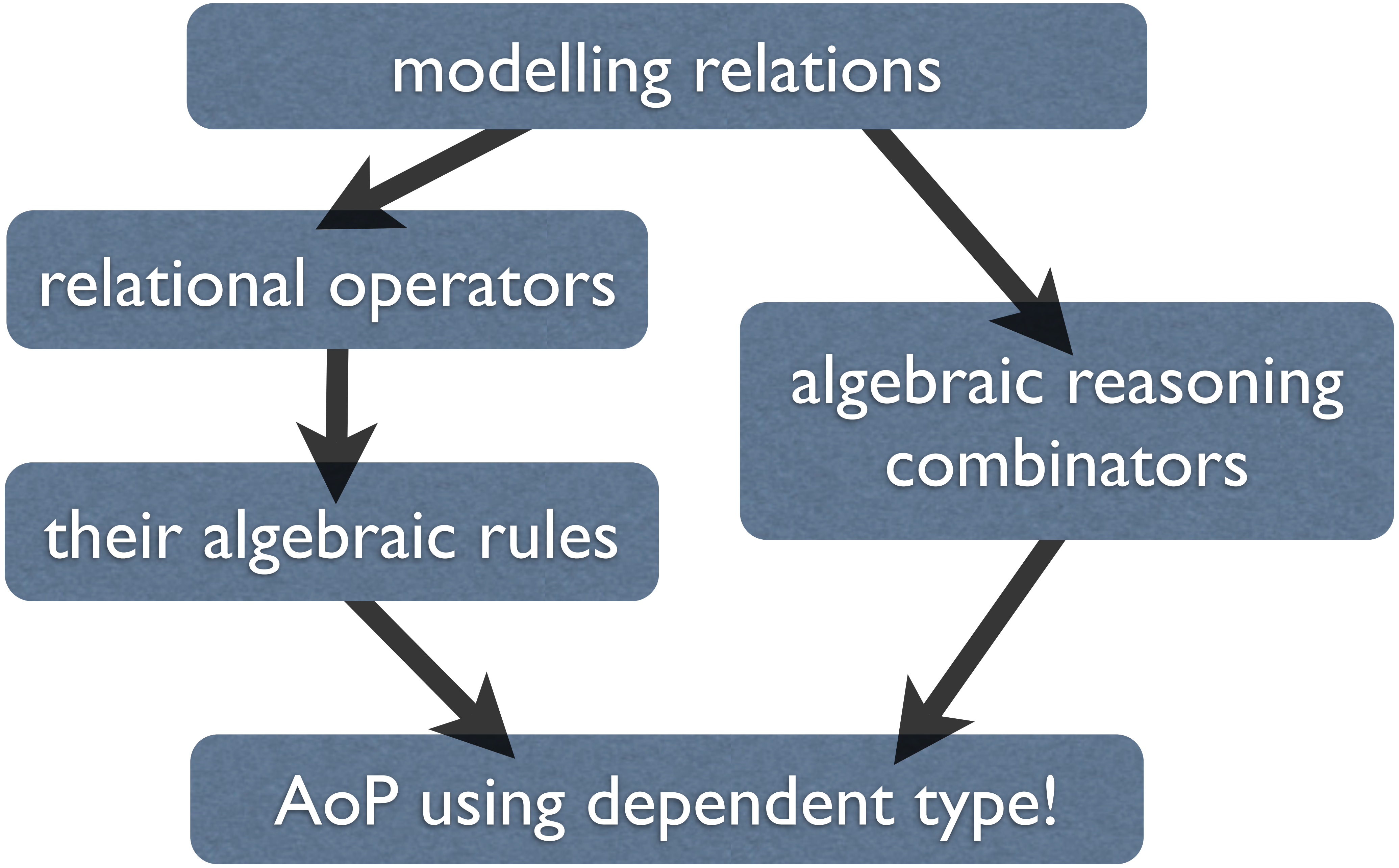
Concise Specifications

- $S \subseteq R/T$ iff $S \circ T \subseteq R$. A pointwise definition: $(c,b) \in R/T$ iff for all a, $(b,a) \in T \rightarrow (c,a) \in R$.

  $R : C \leftarrow A,\ S: C \leftarrow B,\ T: B \leftarrow A$
  $R/T : C \leftarrow B$

- $min\ R : A \leftarrow \mathbb{P}\, A$ is given by $\in \cap\ (R\ /\ \ni)$.
  - $\in\ : A \leftarrow \mathbb{P}\, A$, the membership relation.
  - $\ni$ is the converse of $\in$.
  - $(x,s) \in min\ R$ iff $x \in s$ and for all a, $s \ni a$ implies $(x,a) \in R$.

The (Optimistic) Plan...



Preorder Reasoning

$\sim$*-begin*
  e_1
$\sim\langle$ *reason*$_1$ $\rangle$
  $\vdots$
  e_{n-1}
$\sim\langle$ *reason*$_{n-1}$ $\rangle$
  e_n
$\sim\blacksquare$

- should be bracketed as
  $\sim$*-begin* $(e_1 \sim\langle$ *reason*$_1$ $\rangle$ …
      $(e_{n-1} \sim\langle$ *reason*$_{n-1}$ $\rangle$ $(e_n \sim\blacksquare))$…$)$

Preorder Reasoning

$$
\begin{array}{l}
\sim\text{-}begin \\
\quad e_1 \\
\sim\langle\ reason_1\ \rangle \\
\qquad \vdots \\
\quad e_{n-1} \\
\sim\langle\ reason_{n-1}\ \rangle \\
\boxed{\begin{array}{l} \quad e_n \\ \sim\blacksquare \end{array}} \quad e_n \sim e_n
\end{array}
$$

- should be bracketed as

  $\sim\text{-}begin\ (e_1 \sim\langle\ reason_1\ \rangle \ldots$

  $\qquad (e_{n-1} \sim\langle\ reason_{n-1}\ \rangle\ (e_n \sim\blacksquare))\ldots)$

Preorder Reasoning

$\sim$-begin

  e_1

$\sim\langle$ *reason*$_1$ $\rangle$

  $\vdots$

  e_{n-1}

$\sim\langle$ *reason*$_{n-1}$ $\rangle$ $e_2 \sim e_n$

  e_n

$\sim\blacksquare$ $e_n \sim e_n$

- should be bracketed as

  $\sim$-begin $(e_1 \sim\langle$ *reason*$_1$ $\rangle$ …

  $(e_{n-1} \sim\langle$ *reason*$_{n-1}$ $\rangle$ $(e_n \sim\blacksquare))\ldots)$

Preorder Reasoning

$\sim$-*begin*

$\quad e_1$

$\sim\langle\ \textit{reason}_1\ \rangle \longleftarrow e_1{\sim}e_2$

$\quad \vdots$

$\quad e_{n-1}$

$\sim\langle\ \textit{reason}_{n-1}\ \rangle \qquad e_2{\sim}e_n$

$\quad e_n$

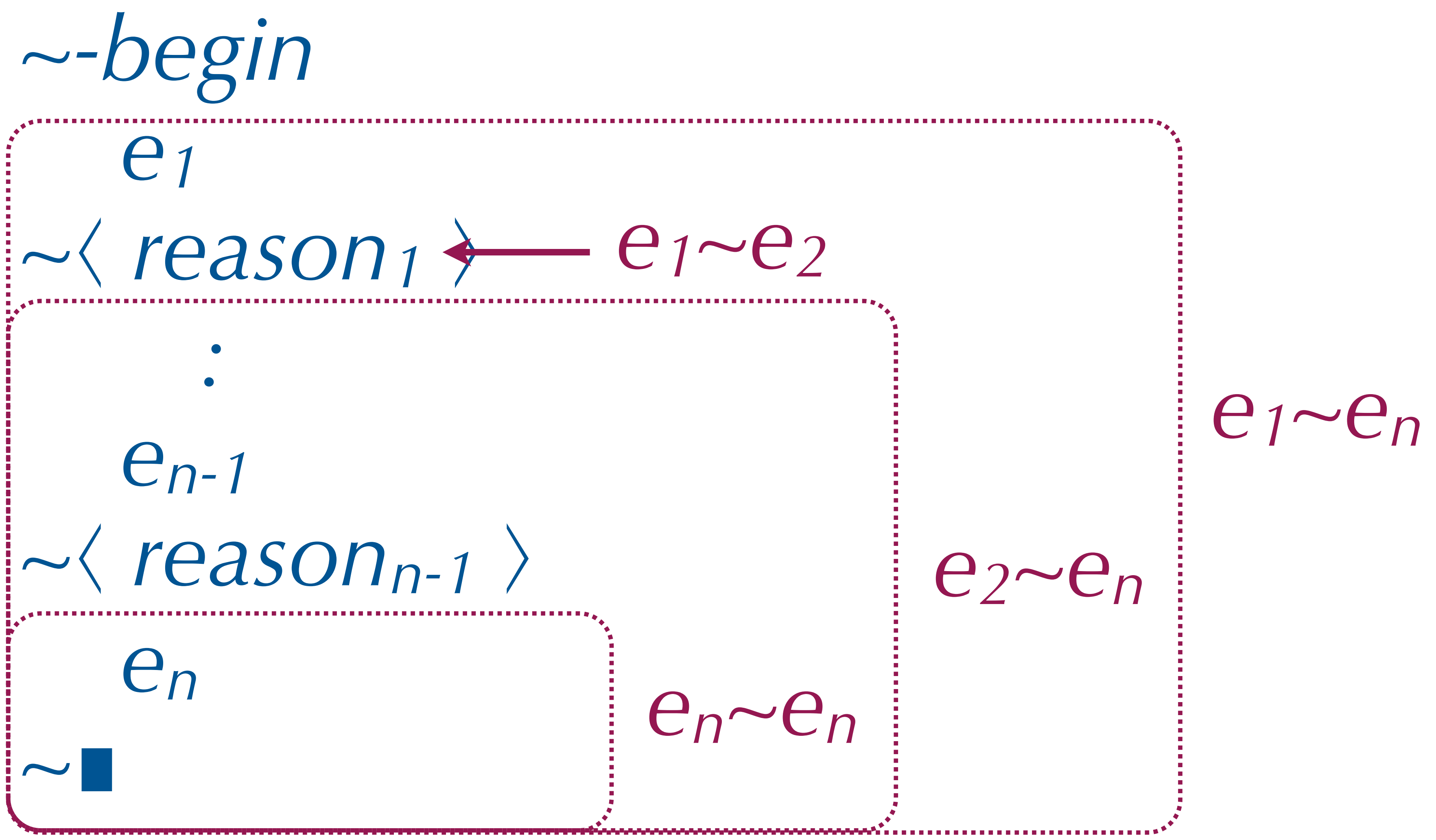
$\sim\blacksquare \qquad e_n{\sim}e_n$

- should be bracketed as

  $\sim$-*begin* $(e_1 \sim\langle\ \textit{reason}_1\ \rangle \ldots$

  $\qquad (e_{n-1} \sim\langle\ \textit{reason}_{n-1}\ \rangle\ (e_n \sim\blacksquare))\ldots)$

Preorder Reasoning



```
~-begin
    e_1
~⟨ reason_1 ⟩
        :
    e_{n-1}
~⟨ reason_{n-1} ⟩
    e_n
~■
```

- should be bracketed as

  ~-begin $(e_1 \sim\langle$ *reason*$_1 \rangle \ldots$

  $(e_{n-1} \sim\langle$ *reason*$_{n-1} \rangle\ (e_n \sim\blacksquare))\ldots)$

Preorder Reasoning

```
∼-begin : {A : Set}{x y : A} → x ∼ y → x ∼ y
∼-begin x∼y  =  x∼y

_∼⟨_⟩_ : {A : Set}(x : A){y z : A} →
                   x ∼ y → y ∼ z → x ∼ z,
x ∼⟨ x∼y ⟩ y∼z  =  ∼-trans x∼y y∼z

_∼∎ : {A : Set}{x : A} → x ∼ x
x ∼∎  =  ∼-refl
```

Modelling Sets & Relations

- $\mathbb{P} : Set \to Set1$

  $\mathbb{P}\ A = A \to Set.$

- $\_\!\leftarrow\!\_ \quad : Set \to Set \to Set1$

  $B \leftarrow A \ = \ A \to B \to Set.$

- $\_\circ\_ : \{A\ B\ C : Set\} \to$

  $\qquad (C \leftarrow B) \to (B \leftarrow A) \to (C \leftarrow A)$

  $(R \circ S)\ a\ c = \exists\ (\backslash b \to S\ a\ b \times R\ b\ c).$

- $\_\sqsubseteq\_ : \{A\ B : Set\} \to (B \leftarrow A) \to (B \leftarrow A) \to Set$

  $R \sqsubseteq S = \textbf{forall}\ a\ b \to R\ a\ b \to S\ a\ b.$

Polymorphic Universe?

- $\in\ \ : \{A : Set\} \to (A \leftarrow\ \ \mathbb{P}\, A)$

  $\in s\ \ =\ \ s.$

- $\_\leftarrow\ \ \_ : Set \to Set\ \ \ \to Set1$

  $B \leftarrow\ \ A = A \to B \to Set.$

- $\_\circ_1\_ : \{A : Set1\}\ \{B\ C : Set\} \to$

  $(C \leftarrow B) \to (B \leftarrow_1 A) \to (C \leftarrow_1 A).$

Polymorphic Universe?

- $\in \ : \{A : Set\} \to (A \leftarrow \ \mathbb{P}\, A)$
  $\in s \ = \ s.$ $\qquad = (A \to Set) \to A \to Set$
- $\_\leftarrow \ \_ : Set \to Set \quad \to Set1$
  $B \leftarrow \ A = A \to B \to Set.$
- $\_\circ_1\_ : \{A : Set1\} \ \{B \ C : Set\} \to$
  $(C \leftarrow B) \to (B \leftarrow_1 A) \to (C \leftarrow_1 A).$

Polymorphic Universe?

- $\in\ : \{A : Set\} \to (A \leftarrow_1 \mathbb{P}\, A)$
  $\in s\ =\ s.$ $\quad = (A \to Set) \to A \to Set$
- $\_\leftarrow_1\_ : Set \to Set1 \to Set1$
  $B \leftarrow_1 A = A \to B \to Set.$
- $\_\circ_1\_ : \{A : Set1\}\ \{B\ C : Set\} \to$
  $(C \leftarrow B) \to (B \leftarrow_1 A) \to (C \leftarrow_1 A).$

Polymorphic Universe?

-
  ```
  min : {A : Set} → (A ← A) → (A ← ℙ A)
  min R = ∈ ⊓ (R / ∋).
  ```

-
  ```
  _/_ : {A B : Set} {C : Set } →
       (B ← A) → (C ← A) → (B ← C)
   (R / S) c b = forall a → S a c → R a b.
  ```

Polymorphic Universe?

- $min : \{A : Set\} \to (A \leftarrow A) \to (A \leftarrow_1 \mathbb{P}\, A)$
  $min\ R = \in \sqcap_1\ (R\ /_1 \ni)$.

- $\_/_1\_ : \{A\ B : Set\}\ \{C : Set1\} \to$
  $(B \leftarrow A) \to (C\ {}_1{\leftarrow}\ A) \to\ (B \leftarrow_1 C)$
  $(R\ /_1 S)\ c\ b = \textbf{forall}\ a \to S\ a\ c \to R\ a\ b$.

Poly. Universe Not Helping!

- Let $R : C \leftarrow B$, $S : B \leftarrow A$, $T : C \leftarrow A$.
- Univ. property: $R \circ S \sqsubseteq T \longleftrightarrow R \sqsubseteq T \,/\, S$.
    - We cannot even talk about $R \circ S \sqsubseteq T$.
    - $T : C \leftarrow A = A \rightarrow C \rightarrow \mathit{Set}$.
    - $(R \circ S)\ a\ c = \exists\ (\backslash b \rightarrow S\ a\ b \times R\ b\ c)$, but b is in $\mathit{Set1}$, so $R \circ S$ cannot have type $A \rightarrow C \rightarrow \mathit{Set}$.
- Fortunately, the only ${}_1\!\leftarrow$ arrow we need so far is $\ni$. We may take $(\circ \ni)$ as one operator.

Poly. Universe Not Helping!

- Let $R : C \leftarrow B, \quad S : B \;{}_1\!\!\leftarrow A, \quad T : C \leftarrow A$.
- Univ. property: $R \circ S \sqsubseteq T \quad \longleftrightarrow \quad R \sqsubseteq T /_1 S$.
  - We cannot even talk about $R \circ S \sqsubseteq T$.
  - $T : C \leftarrow A = A \rightarrow C \rightarrow Set$.
  - $(R \circ S)\; a\; c = \exists\; (\backslash b \rightarrow S\; a\; b \times R\; b\; c)$, but b is in $Set1$, so $R \circ S$ cannot have type $A \rightarrow C \rightarrow Set$.
- Fortunately, the only ${}_1\!\!\leftarrow$ arrow we need so far is $\ni$. We may take $(\circ \ni)$ as one operator.

Relational Fold & Fusion

- $foldR : \{A\ B : Set\} \rightarrow$
  $(B \leftarrow (A \times B)) \rightarrow \mathbb{P}\ B \rightarrow (B \leftarrow [\ A\ ])$
  $foldR\ R\ s =$
  $\in\ \circ\ foldr\ (\Lambda\ (R \circ (idR \times \in)))\ s.$
- $foldR\text{-}fusion\text{-}\sqsupseteq : \{A\ B\ C : Set\}\ (R : C \leftarrow B) \rightarrow$
  $\{S : B \leftarrow (A \times B)\}\ \{T : C \leftarrow (A \times C)\} \rightarrow$
  $\{u : \mathbb{P}\ B\}\ \{v : \mathbb{P}\ C\} \rightarrow$
  $(R \circ S) \sqsupseteq (T \circ (idR \times R)) \rightarrow$
  $\mathcal{E}\ R\ u \supseteq v \rightarrow$
  $(R \circ foldR\ S\ u) \sqsupseteq foldR\ T\ v.$

Relational Fold & Fusion

- $foldR : \{A\ B : Set\} \rightarrow$

  $(B \leftarrow (A \times B)) \rightarrow \mathbb{P}\ B \rightarrow (B \leftarrow [\ A\ ])$

  $foldR\ R\ s =$

  $\in\ {}_1\!\circ\ foldr_1\ (\Lambda_1\ (R \circ_1 (idR \times_1 \in)))\ s.$

- $foldR\text{-}fusion\text{-}\sqsupseteq\ : \{A\ B\ C : Set\}\ (R : C \leftarrow B) \rightarrow$

  $\{S : B \leftarrow (A \times B)\}\ \{T : C \leftarrow (A \times C)\} \rightarrow$

  $\{u : \mathbb{P}\ B\}\ \{v : \mathbb{P}\ C\} \rightarrow$

  $(R \circ S) \sqsupseteq (T \circ (idR \times R)) \rightarrow$

  $\mathcal{E}\ R\ u \supseteq v \rightarrow$

  $(R \circ foldR\ S\ u) \sqsupseteq foldR\ T\ v.$

Permutation, Order, and Sort

- $permute : [\,Val\,] \leftarrow [\,Val\,]$
  $permute = bagify^{\breve{}} \circ bagify.$
- $ordered? : [\,Val\,] \leftarrow [\,Val\,]$
  $ordered? = foldR\ (cons \circ lbound?)\ nil.$
- $sort\text{-}der :$
  $\exists\ (\backslash f \rightarrow ordered? \circ permute \sqsupseteq fun\ f).$
- Deriving insertion sort: about 700 lines of derivation + 700 lines of "library code."

Optimisation Problems

- $min : \{A : Set\} \to (A \leftarrow A) \to (A \leftarrow_1 \mathbb{P}\, A)$
  $min\ R = \in \sqcap_1 (R /_1 \ni)$.

- $greedy\text{-}thm : \{A\ B : Set\}$
  $\{S : B \leftarrow (A \times B)\}\ \{s : \mathbb{P}\, B\}\ \{R : B \leftarrow B\} \to$
  $R \circ R \sqsubseteq R \;\to\; S \circ (idR \times R\,\breve{}\,) \sqsubseteq R\,\breve{} \circ S \;\to$
  $foldR\ (min\ R\ {}_1{\circ}\ \Lambda\ S)\ (min\ R\ s) \sqsubseteq$
  $min\ R\ {}_1{\circ}\ \Lambda\ (foldR\ S\ s)$.

- “Activity selection problem:” about 500 lines of proofs/derivation (plus library code).

Deriving Quicksort

```
qsort-der : ∃ (\f -> ordered? ○ permute ⊒ fun f )
qsort-der = (_ ,  (⊒-begin
      ordered? ○ permute
         ...
      fun flatten ○
                (foldT ((fun partition)˘○ ...) ...)˘
   ⊒⟨ ○-monotonic-r (foldT-to-unfoldt partition partition-wf) ⟩
      fun flatten ○ fun (unfoldt partition partition-wf)
   ⊒⟨ fun○-⊒ ⟩
      fun (flatten ∘ unfoldt partition partition-wf)
   ⊒∎ ))
```

- Around 500 lines of code and proof.

Inductively Defined Unfoldr

- $unfoldr : \{A\ B : Set\} \to (A \to (\top \uplus (A \times B)))$
  $\to \quad B \quad \to A$
  $unfoldr\ f\ b$ $\quad$ **with** $f\ b$
  $\quad | \ inj_1\ \_ = []$
  $\quad | \ inj_2\ (a\ ,\ b') = a :: unfoldr\ f\ b'$.

Inductively Defined Unfoldr

- *unfoldr : {A B : Set} → (A → (⊤ ⊎ (A × B))*
  *→ (b : B) → Acc (***ε***-listF ∘ fun f) b → A*
  unfoldr f b (acc .b h) ***with*** *f b*
  | inj$_1$ *_ = []*
  | inj$_2$ *(a , b′) = a :: unfoldr f b′*.
  (h b' (inj$_2$ *(a , b') , ≡-refl , ≡-refl))*

Deriving Quicksort

```
well-found : {A : Set} → (A → A → Set) → Set
well-found R = forall x → Acc R x

partition-wf : well-found (ε-TreeF ○ fun partition)
partition-wf xs = acc-fRfº xs
          (acc-⊑ partition⊑> (length xs) (ℕ>-wf (length xs)))
```

Conclusions

- We can encode relational derivations in dependent types.
  - Correctness guaranteed by type checker.
  - Program extracted as witness.
- To model hylomorphism, we need accessibility/reductivity.