JSSST19@芝浦工業大学

高度な運算定理の Coq による証明とその自動化

2019/08/27

村田 康佑　江本 健斗

九州工業大学

本研究の貢献

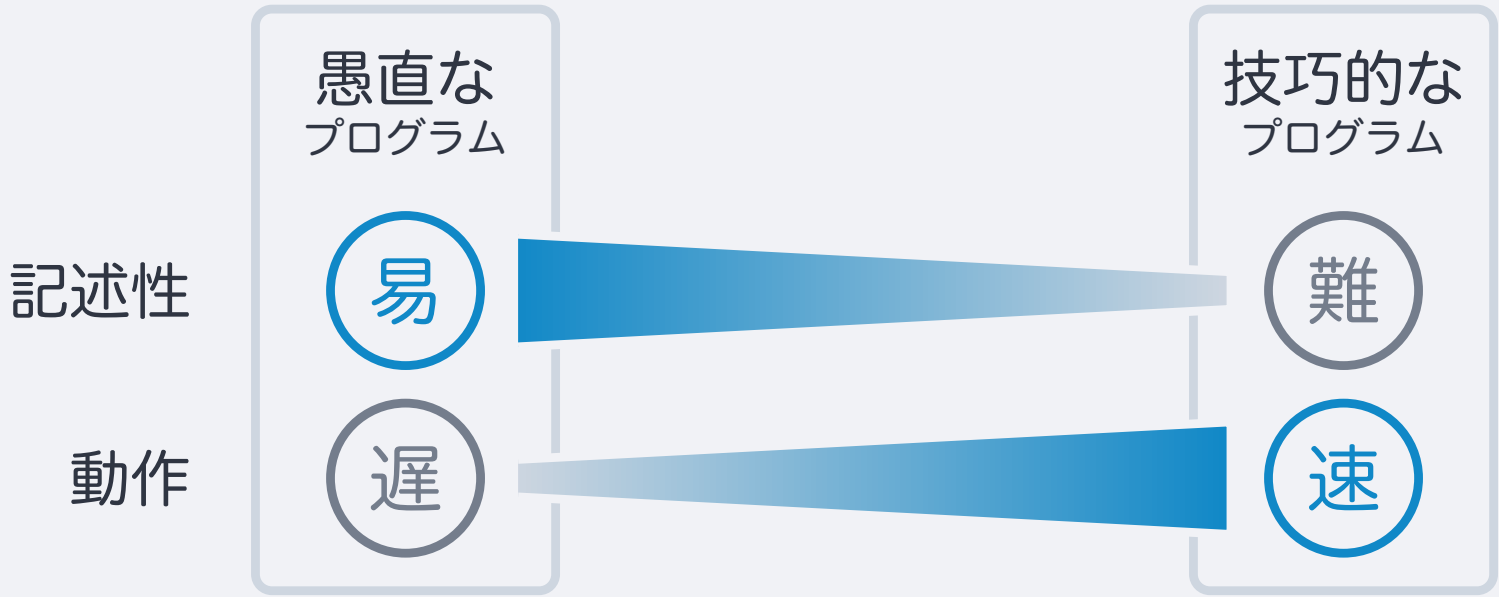
- 運算のための Coq ライブラリ [Tesson+, 2011] を拡張
- **再帰図式に基づく ADT-generic な規則を証明可能に**
- **再帰図式を使った Coq プログラミングを可能に**
  - プログラムには定理を適用可（運算ができる）
  - プログラムは実行・抽出が可能
  - ユーザは型クラスのインスタンス化を通して型ごとの実装を与える
    - 手間を省くため**インスタンス化の一部を自動化**

Outline

1. 概要（万人向けのお話）
2. F 始代数と catamorphism（ざっくりと）
3. 始代数のための型クラス
4. 今後とまとめ

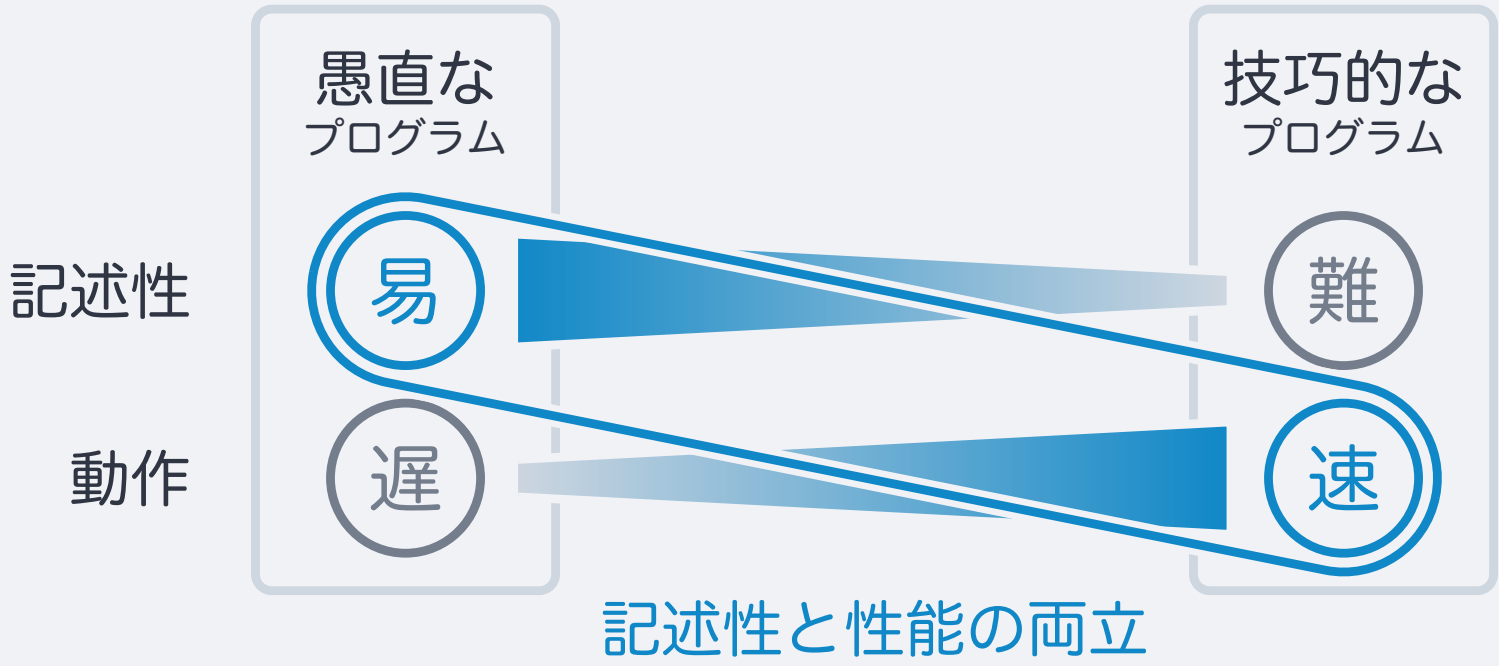
背景

- 効率的なプログラムを簡単に書きたい



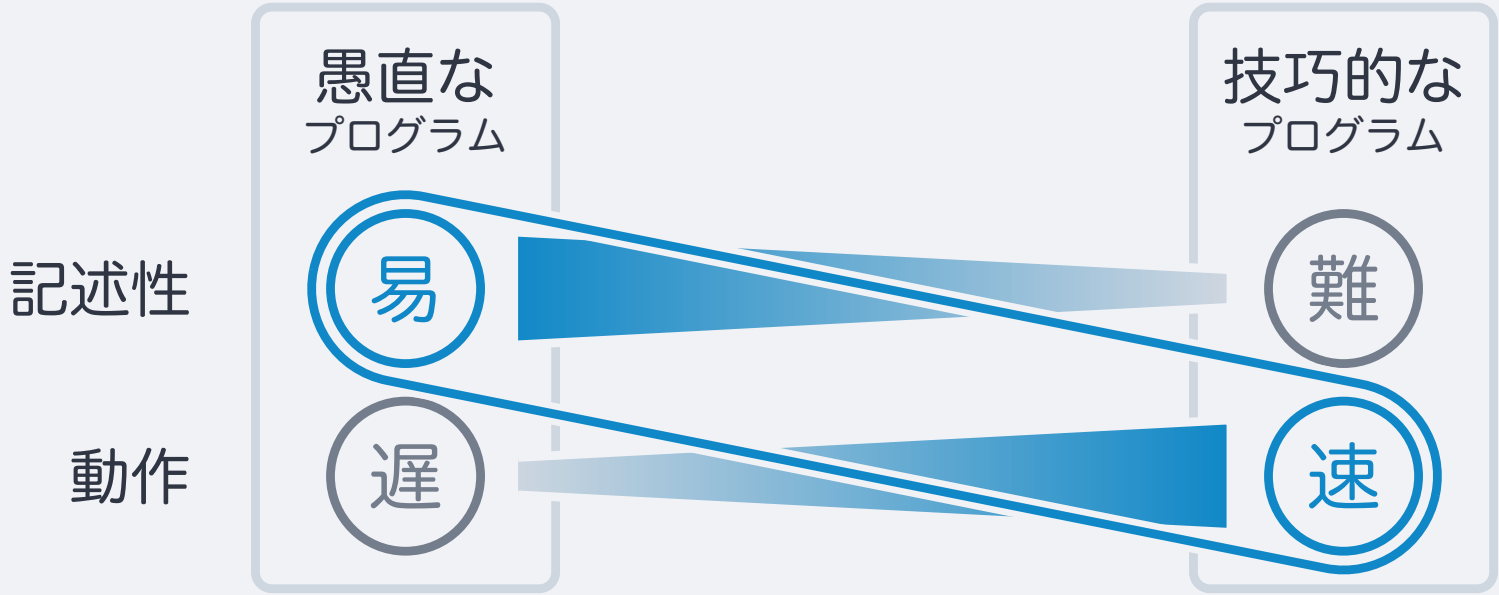
背景

- 効率的なプログラムを簡単に書きたい



背景

- 効率的なプログラムを簡単に書きたい



記述性と性能の両立

↓

プログラム運算 (Program Calculation)

プログラム運算:
うまく等式変形して高速に

$$
\begin{aligned}
& \underline{mss} \\
= \ & \{\ mss \text{ の定義}\ \} \\
& max \circ \underline{map\ sum \circ concat} \circ map\ tails \circ inits \qquad \#\ \mathrm{O}(n^3) \\
= \ & \{\ concat\text{-}map \text{ 融合 (逆)}\ \} \\
& max \circ \underline{concat \circ map\ (map\ sum)} \circ map\ tails \circ inits \\
= \ & \{\ concat\text{-}foldr \text{ 融合 (逆)}\ \} \\
& max \circ \underline{map\ (map\ max) \circ map\ (map\ sum) \circ map\ tails} \circ inits \\
= \ & \{\ map\text{-}map \text{ 融合 (2 回)}\ \} \\
& max \circ map\ (map\ max \circ map\ sum \circ tails) \circ inits \\
& \vdots \\
= \ & max \circ map\ (fst \circ foldr\ (\odot)\ e' \circ map\ f) \circ inits \qquad \#\ \mathrm{O}(n) \\
& \quad \textbf{where} \quad f\ a = (a, a),\ e' = (-\infty, 0), \\
& \qquad\qquad (m_1, s_1) \odot (m_2, s_2) = ((m_1 + s_2) \uparrow m_2,\ s_1 + s_2)
\end{aligned}
$$

プログラム運算:
うまく等式変形して高速に

愚直 (わかりやすいが非効率的) $\mathrm{O}(n^3)$ なプログラム

$$
\begin{aligned}
&= \{\ mss \text{ の定義}\ \} \\
&\quad max \circ \underline{map\ sum \circ concat} \circ map\ tails \circ inits \qquad \#\ \mathrm{O}(n^3) \\
&= \{\ concat\text{-}map \text{ 融合 (逆)}\ \} \\
&\quad max \circ \underline{concat \circ map\ (map\ sum)} \circ map\ tails \circ inits \\
&= \{\ concat\text{-}foldr \text{ 融合 (逆)}\ \} \\
&\quad max \circ \underline{map\ (map\ max) \circ map\ (map\ sum) \circ map\ tails} \circ inits \\
&= \{\ map\text{-}map \text{ 融合 (2 回)}\ \} \\
&\quad max \circ map\ (map\ max \circ map\ sum \circ tails) \circ inits \\
&\quad \vdots \\
&= \quad max \circ map\ (fst \circ foldr\ (\odot)\ e' \circ map\ f) \circ inits \qquad \#\ \mathrm{O}(n) \\
&\qquad \textbf{where} \quad f\ a = (a, a),\ e' = (-\infty, 0), \\
&\qquad\qquad (m_1, s_1) \odot (m_2, s_2) = ((m_1 + s_2) \uparrow m_2,\ s_1 + s_2)
\end{aligned}
$$

プログラム運算:
うまく等式変形して高速に

愚直 (わかりやすいが非効率的) $\mathrm{O}(n^3)$ なプログラム

代数法則を用いた変形を繰り返す
E.g. *map-map* 融合則: $(map\ g) \circ (map\ f) = map\ (g \circ f)$

$$
\begin{aligned}
&= \{\ mss \text{ の定義}\ \} \\
&\quad max \circ map\ sum \circ concat \circ map\ tails \circ inits \qquad \#\ \mathrm{O}(n^3) \\
&= \{\ \ldots\ \} \\
&\quad max \circ \underline{map\ (map\ max) \circ map\ (map\ sum) \circ map\ tails} \circ inits \\
&= \{\ map\text{-}map \text{ 融合 (2 回)}\ \} \\
&\quad max \circ map\ (map\ max \circ map\ sum \circ tails) \circ inits \\
&\quad \vdots \\
&= \quad max \circ map\ (fst \circ foldr\ (\odot)\ e' \circ map\ f) \circ inits \qquad \#\ \mathrm{O}(n) \\
&\qquad \textbf{where} \quad f\ a = (a, a),\ e' = (-\infty, 0), \\
&\qquad\qquad (m_1, s_1) \odot (m_2, s_2) = ((m_1 + s_2) \uparrow m_2,\ s_1 + s_2)
\end{aligned}
$$

プログラム運算:
うまく等式変形して高速に

愚直 (わかりやすいが非効率的) $\mathrm{O}(n^3)$ なプログラム

$$= \{\ mss \text{ の定義}\ \}$$
$$max \circ map\ sum \circ concat \circ map\ tails \circ inits \qquad \#\ \mathrm{O}(n^3)$$

代数法則を用いた変形を繰り返す
E.g. *map-map* 融合則: $(map\ g) \circ (map\ f) = map\ (g \circ f)$

$$max \circ map\ (map\ max) \circ map\ (map\ sum) \circ map\ tails \circ inits$$
$$= \{\ map\text{-}map \text{ 融合 (2 回)}\ \}$$
$$max \circ map\ (map\ max \circ map\ sum \circ tails) \circ inits$$
$$\vdots$$
$$= \quad max \circ map\ (fst \circ foldr\ (\odot)\ e' \circ map\ f) \circ inits \qquad \#\ \mathrm{O}(n)$$
$$\textbf{where} \quad f\ a = (a, a),\ e' = (-\infty, 0),$$
$$(m_1, s_1) \odot (m_2, s_2) = ((m_1 + s_2) \uparrow m_2,\ s_1 + s_2)$$

高速 $\mathrm{O}(n)$ なプログラムへ！

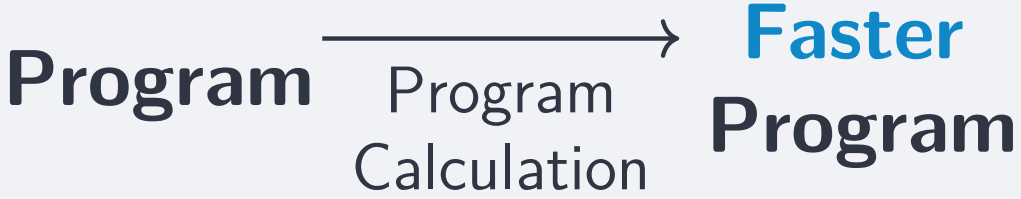
本研究の狙い

Program $\xrightarrow[\text{Calculation}]{\text{Program}}$ **Faster Program**

運算定理の適用

E.g. $\mathsf{map}\ g \circ \mathsf{map}\ f = \mathsf{map}\ (g \circ f)$

本研究の狙い



運算定理の適用

E.g. $\mathsf{map}\ g \circ \mathsf{map}\ f = \mathsf{map}\ (g \circ f)$

🤔 運算法則自体の正しさは？

🤔 運算法則は正しく適用できている？

本研究の狙い

運算定理の適用

E.g. $\mathrm{map}\ g \circ \mathrm{map}\ f = \mathrm{map}\ (g \circ f)$

- 🤔 運算法則自体の正しさは？
- 🤔 運算法則は正しく適用できている？

Coq で検証したい！

運算法則の例:
Map-map fusion law

$$(\mathsf{map}\ g) \circ (\mathsf{map}\ f) = \mathsf{map}\ (g \circ f)$$

For Lists

運算法則の例:
Map-map fusion law

$$(\mathsf{map}\ g) \circ (\mathsf{map}\ f) = \mathsf{map}\ (g \circ f)$$

For Lists

$$[a_1, \ldots, a_n]$$

運算法則の例:
Map-map fusion law

$$(\mathsf{map}\ g) \circ (\mathsf{map}\ f) = \mathsf{map}\ (g \circ f)$$

For Lists

$$\begin{array}{c} [a_1, \ldots, a_n] \\ \mathbf{map}\ f \downarrow \\ [f\ a_1, \ldots, f\ a_n] \end{array}$$

運算法則の例:

Map-map fusion law

$$(\mathsf{map}\ g) \circ (\mathsf{map}\ f) = \mathsf{map}\ (g \circ f)$$

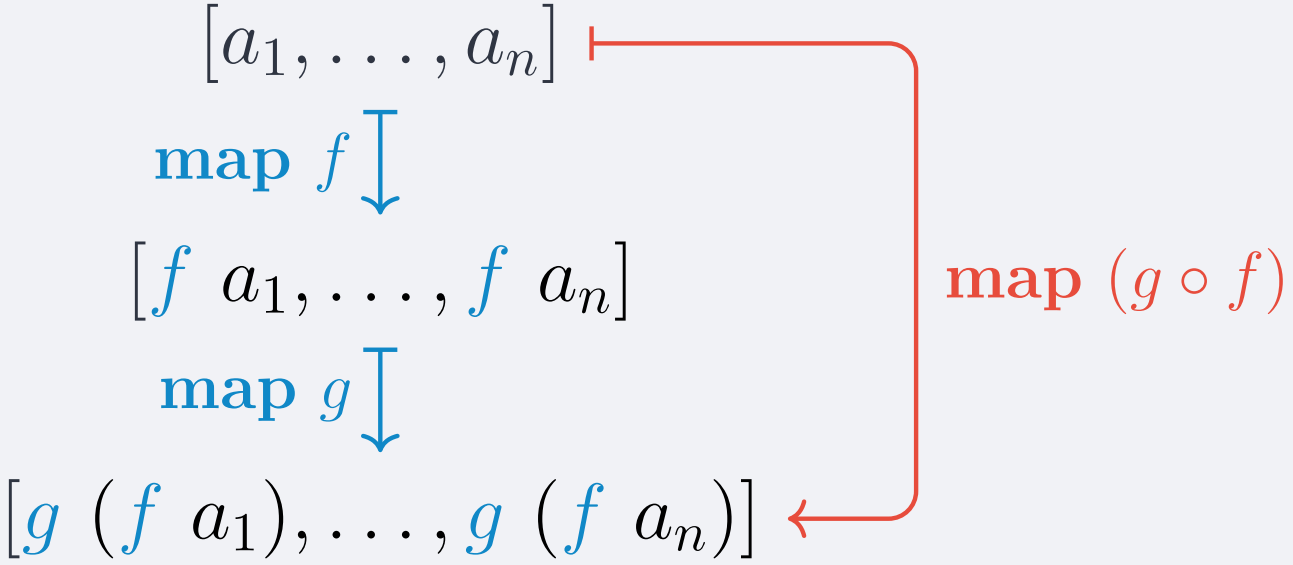
For Lists

$$
\begin{array}{c}
[a_1, \ldots, a_n] \\
\downarrow \mathbf{map}\ f \\
[f\ a_1, \ldots, f\ a_n] \\
\downarrow \mathbf{map}\ g \\
[g\ (f\ a_1), \ldots, g\ (f\ a_n)]
\end{array}
$$

運算法則の例:
Map-map fusion law

$$(\text{map } g) \circ (\text{map } f) = \text{map } (g \circ f)$$

For Lists



先行研究 [Tesson+, 2011]

- プログラム運算のための Coq ライブラリ
  - 鎖状記法による equational reasoning
  - BMF（Bird-Meertens Formalism）に基づく
- Shallow embedding で Coq プログラムを運算
  - 適用例: リスト上のプログラムについての運算法則
  - Theory of Lists を形式化
    - Bird によるレクチャノート
    - リスト関数の運算規則集

```
Theorem filter promotion :
  p <| :o: @concat A = @concat A :o: p <| *.
Proof.
    LHS
= { rewrite (filter mapreduce) }
    (++ / :o: f * :o: @concat A).
= { rewrite map promotion }
    (++ /:o: @concat (list A) :o: f* *).
= { rewrite comp assoc }
    ((++ /:o: @concat (list A)) :o: f* *).
= { rewrite reduce promotion }
    (++ / :o: (++ /)* :o: f* *).
= { rewrite concat reduce }
    (@concat A :o: (++ /)* :o: f* *).
= { rewrite map distr comp }
    (@concat A :o: (++ / :o: f *) *).
= { rewrite filter mapreduce }
    (@concat A :o: (p <|) *).
[].
Qed.
```

本研究の問題意識:
一般の ADT へ拡張

- ADT = Algebraic Data Types
  - 関数プログラマは様々な ADT を用いる
    - 自然数，リスト，二分木，…
- 本研究: **一般の ADT をサポートしたい！**
  - 一般の ADT を使ったプログラムを運算
  - 一般の ADT に対して成り立つ運算定理を証明

本研究の問題意識:
一般の ADT へ拡張

- ADT = Algebraic Data Types
  - 関数プログラマは様々な ADT を用いる
    - 自然数，リスト，二分木，…
- 本研究: **一般の ADT をサポートしたい！**
  - 一般の ADT を使ったプログラムを運算
  - 一般の ADT に対して成り立つ運算定理を証明

再帰図式（Recursion Scheme）

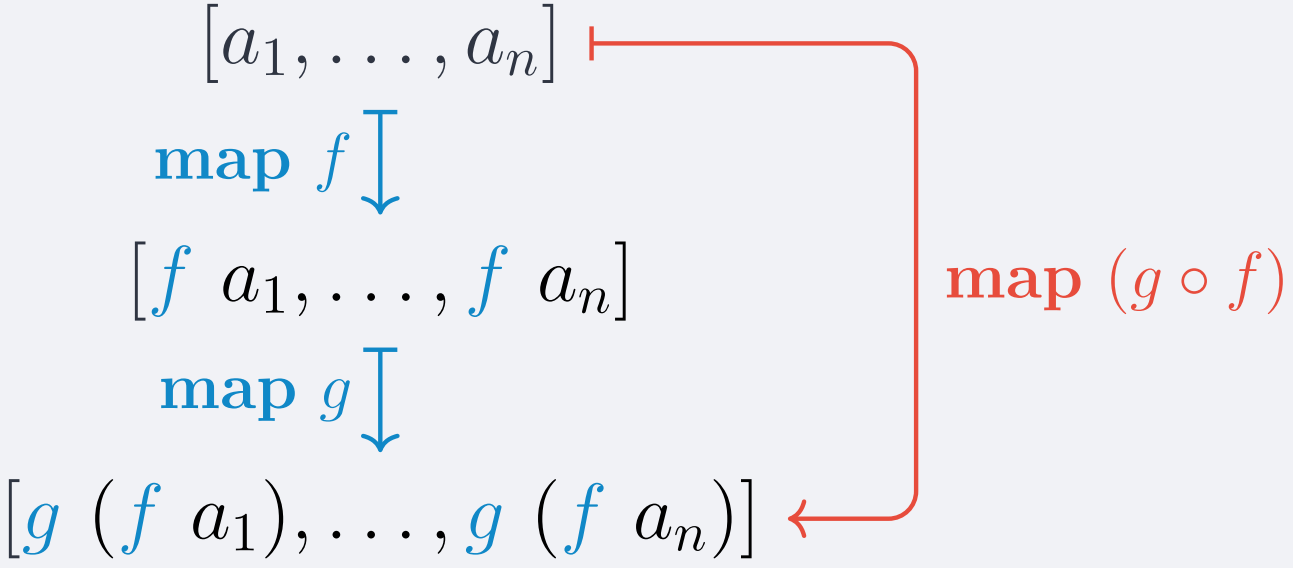
再帰図式 (Recursion Schemes)

- 典型的な再帰計算のパタンを捉える
- Generic Programming の文脈で精力的に研究
- ADT-generic な定義を与えることが可能
  - 様々な ADT での再帰計算を統一的に取り扱える
  - ADT-generic な運算規則を与えられる

ADT-generic な運算法則（雰囲気だけ）：
Map-map fusion law (generalized)

$$(\mathsf{map}\ g) \circ (\mathsf{map}\ f) = \mathsf{map}\ (g \circ f)$$

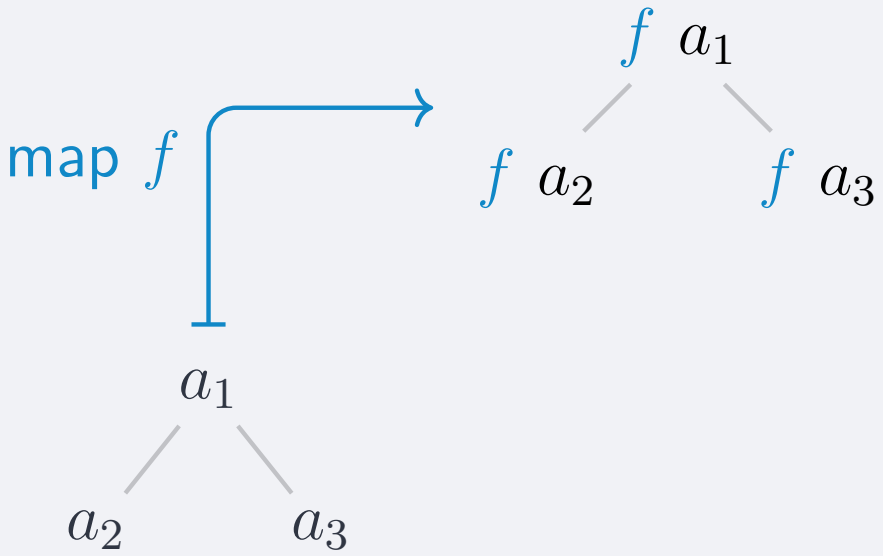
For Lists



ADT-generic な運算法則（雰囲気だけ）：
Map-map fusion law (generalized)

$$(\mathsf{map}\ g) \circ (\mathsf{map}\ f) = \mathsf{map}\ (g \circ f)$$

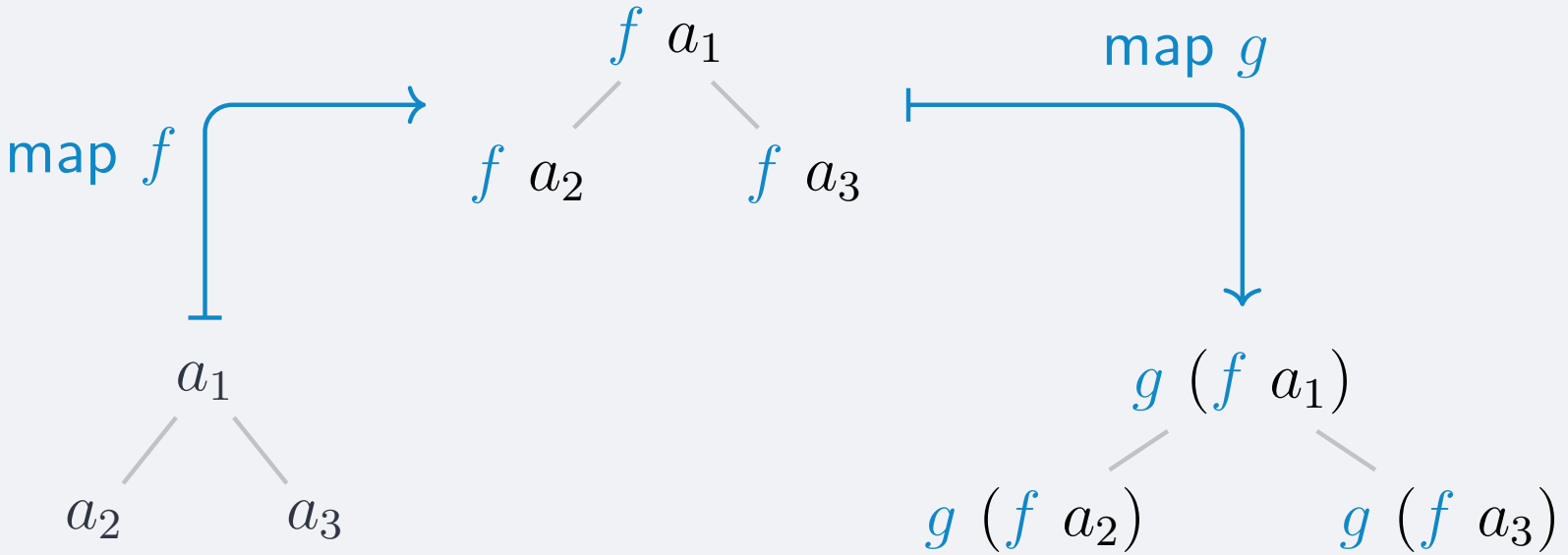
For Trees



ADT-generic な運算法則（雰囲気だけ）：
Map-map fusion law (generalized)

$$(\mathsf{map}\ g) \circ (\mathsf{map}\ f) = \mathsf{map}\ (g \circ f)$$

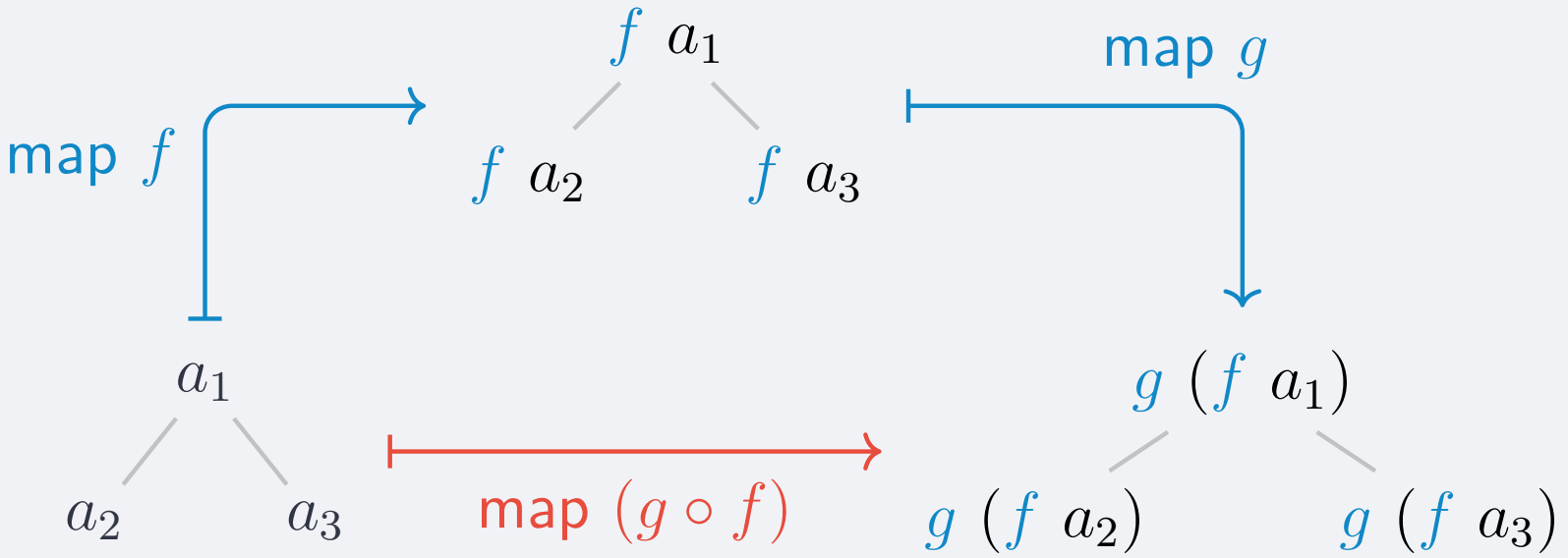
For Trees



ADT-generic な運算法則（雰囲気だけ）：
Map-map fusion law (generalized)

$$(\mathsf{map}\ g) \circ (\mathsf{map}\ f) = \mathsf{map}\ (g \circ f)$$

For Trees



ADT-generic な map（雰囲気だけ）

- 再帰図式で **ADT-generic な map を実装**できる:

$$\mathsf{map}_F\ f = (\!|\, \mathsf{in}_F \circ F(f, \mathrm{id}) \,|\!)$$

  - 双関手 F の選び方によって色々な ADT に対応
    - E.g. $F(A, X) = \mathbf{1} + A \times X$ ならリスト用の map

- **ADT-generic な map-map fusion 則を記述可能**

$$\mathsf{map}_F\ g \circ \mathsf{map}_F\ f = \mathsf{map}_F\ (g \circ f)$$

ADT-generic な map（雰囲気だけ）

- 再帰図式で **ADT-generic な map を実装**できる:

$$\mathsf{map}_F\ f = (\!|\, \mathsf{in}_F \circ F(f, \mathrm{id}) \,|\!)$$

  - 双関手 F の選び方によって色々な ADT に対応
    - E.g. $F(A, X) = \mathbf{1} + A \times X$ ならリスト用の map

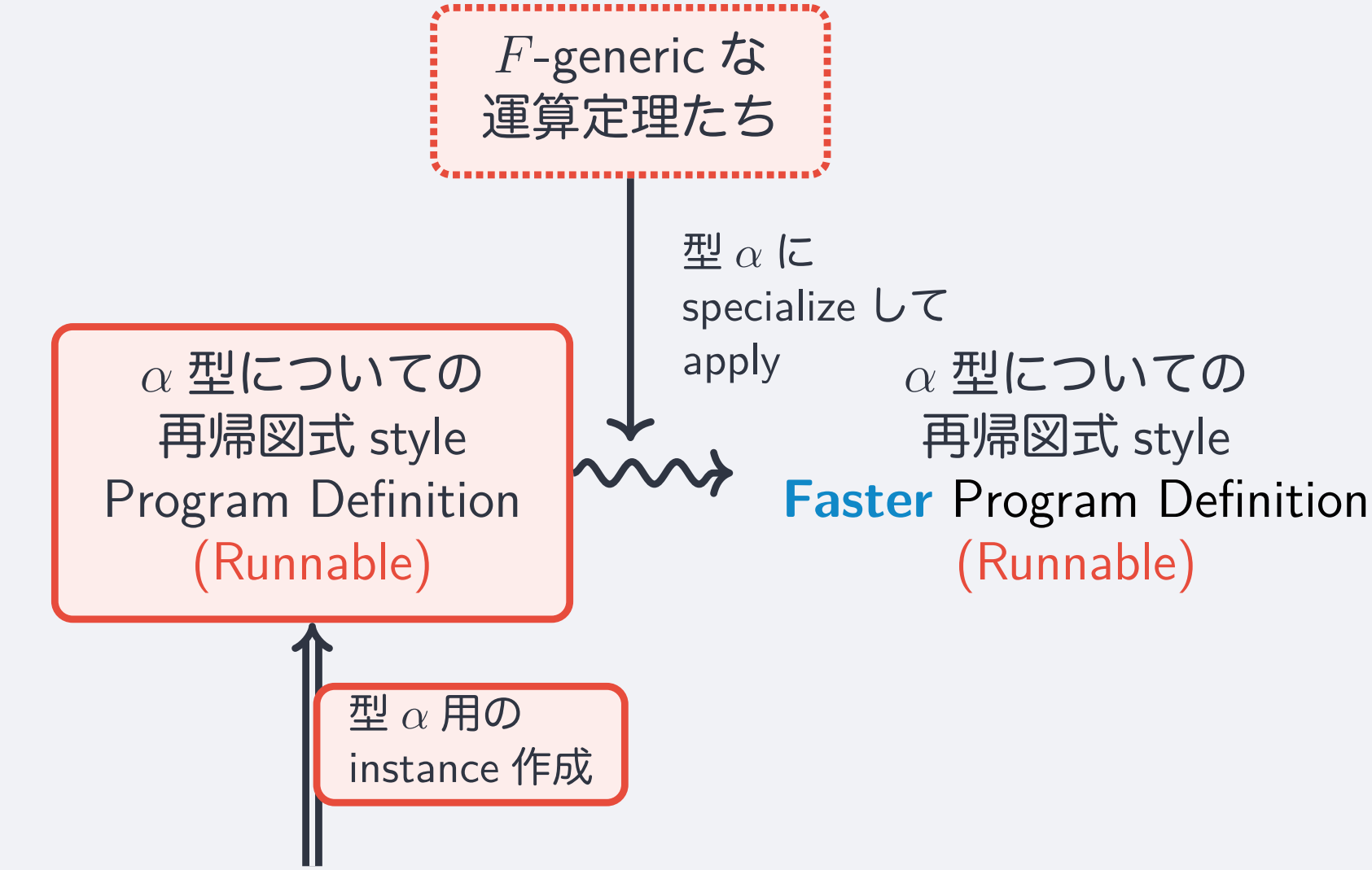
- **ADT-generic な map-map fusion 則を記述可能**

$$\mathsf{map}_F\ g \circ \mathsf{map}_F\ f = \mathsf{map}_F\ (g \circ f)$$

再帰図式は ADT-generic な運算に
適したフレームワーク 😊

本研究の貢献（再掲）

- 運算のための Coq ライブラリ [Tesson+, 2011] を拡張
- **再帰図式に基づく ADT-generic な規則を証明可能に**
- **再帰図式を使った Coq プログラミングを可能に**
  - プログラムには定理を適用可（運算ができる）
  - プログラムは実行・抽出が可能
  - ユーザは型クラスのインスタンス化を通して型ごとの実装を与える
    - 手間を省くため**インスタンス化の一部を自動化**


F-generic な
運算定理たち
型 α に
specialize して
apply
α 型についての
再帰図式 style
Program Definition
(Runnable)
α 型についての
再帰図式 style
Faster Program Definition
(Runnable)
型 α 用の
instance 作成
initial algebra および
terminal coalgebra のための
型クラス

```
Theorem data_functor :
  forall {A B C : Type} (f : A -> B) (g : B -> C),
    (fmap g) ∘ (fmap f) = fmap (g ∘ f).
Proof.
  intros; unfold fmap.
  assert ( (fmap g) ∘ in_ ∘ F⟨f⟩[id] = in_ ∘ F⟨g ∘ f⟩[id] ∘ F⟨id⟩[fmap g] ).
  {
    unfold fmap.
      Left
    = ( (⦇in_ ∘ F⟨g⟩[id]⦈) ∘ in_ ∘ F⟨f⟩[id] ).
    = ( in_ ∘ (F⟨g⟩[id] ∘ F⟨id⟩[(⦇in_ ∘ F⟨g⟩[id]⦈)]) ∘ F⟨f⟩[id] )
        { by cata_cancel }.
    = ( in_ ∘ (F⟨g⟩[(⦇in_ ∘ F⟨g⟩[id]⦈)] ∘ F⟨f⟩[id]) )
        { by fmap_functor_dist }.
    = ( in_ ∘ (F⟨g ∘ f⟩[(⦇in_ ∘ F⟨g⟩[id]⦈)]) )
        { by fmap_functor_dist }.
    = ( in_ ∘ (F⟨g ∘ f⟩[id] ∘ F⟨id⟩[(⦇in_ ∘ F⟨g⟩[id]⦈)]) )
        { by fmap_functor_dist }.
    = Right.
  }
  unfold fmap in H0.
  Left
  = ( (⦇ in_ ∘ F⟨g⟩[id] ⦈) ∘ (⦇ in_ ∘ F⟨f⟩[id] ⦈) ).
  = ( (⦇in_ ∘ F ⟨g ∘ f⟩[id] ⦈) )  { apply cata_fusion }.
  = Right.
Qed.
```

Folding

(Tearing downs the Inductive Dataypes)

Catamorphism[Malcom, 1990]

$$f = \varphi \circ F(f) \circ \mathsf{in}_F^{-1}$$

Paramorphism[Meertens, 1992]

$$f = \varphi \circ F(\langle f, \mathrm{id}_{\mu F} \rangle) \circ \mathsf{in}_F^{-1}$$

Histomorphism[Uustalu+, 1999]

$$f = \varphi \circ F(\llbracket \langle f, \mathsf{in}_F^{-1} \rangle \rrbracket) \circ \mathsf{in}_F^{-1}$$

Unfolding

(Building up the Coinductive Dataypes)

Anamorphism[Fokkinga+, 1991]

$$f = \mathsf{out}_F^{-1} \circ F(f) \circ \psi$$

Apomorphism[Vos, 1995]

$$f = \mathsf{out}_F^{-1} \circ F([f, \mathrm{id}_{\nu F}]) \circ \psi$$

Futumorphism[Uustalu+, 1999]

$$f = \mathsf{out}_F^{-1} \circ F((\!| [f, \mathsf{out}_F^{-1}] |\!)) \circ \psi$$

Refolding

(Building up an intermediate data structure then tearing down it)

Hylomorphism[Meijer+, 1991]

$$cata \circ ana$$

Dynamorphism[Kabanov+, 2006]

$$histo \circ ana$$

Folding

(Tearing downs the Inductive Dataypes)

Catamorphism[Malcom, 1990]

$$f = \varphi \circ F(f) \circ \mathsf{in}_F^{-1}$$

Paramorphism[Meertens, 1992]

$$f = \varphi \circ F(\langle f, \mathrm{id}_{\mu F} \rangle) \circ \mathsf{in}_F^{-1}$$

Histomorphism[Uustalu+, 1999]

$$f = \varphi \circ F(\llbracket \langle f, \mathsf{in}_F^{-1} \rangle \rrbracket) \circ \mathsf{in}_F^{-1}$$

Unfolding

(Building up the Coinductive Dataypes)

Anamorphism[Fokkinga+, 1991]

$$f = \mathsf{out}_F^{-1} \circ F(f) \circ \psi$$

Apomorphism[Vos, 1995]

$$f = \mathsf{out}_F^{-1} \circ F([f, \mathrm{id}_{\nu F}]) \circ \psi$$

Futumorphism[Uustalu+, 1999]

$$f = \mathsf{out}_F^{-1} \circ F((\!| [f, \mathsf{out}_F^{-1}] |\!)) \circ \psi$$

Refolding

(Building up an intermediate data structure then tearing down it)

Hylomorphism[Meijer+, 1991]

$$cata \circ ana$$

Dynamorphism[Kabanov+, 2006]

$$histo \circ ana$$

運算規則ライブラリ

- [Uustalu+ 1999] に現れる運算規則をすべて形式化
  - Histo/Futu-morphism の提案論文
  - Histo/Futu に関する運算定理までサポート

cata_charn	:	$f \circ \mathtt{in\_} = \varphi \circ F[f] \leftrightarrow f = (\!\lvert \varphi \rvert\!)$
cata_cancel	:	$(\!\lvert \varphi \rvert\!) \circ \mathtt{in\_} = \varphi \circ F[(\!\lvert \varphi \rvert\!)]$
cata_refl	:	$\mathrm{id} = (\!\lvert \mathtt{in\_} \rvert\!)$
cata_fusion	:	$f \circ \varphi = \psi \circ F[f] \to f \circ (\!\lvert \varphi \rvert\!) = (\!\lvert \psi \rvert\!)$
lemma1	:	$\mathtt{in\_} \circ (\!\lvert F[\mathtt{in\_}] \rvert\!) = \mathrm{id} \wedge (\!\lvert F[\mathtt{in\_}] \circ \mathtt{in\_} \rvert\!) = \mathrm{id}$
Def. of $\mathtt{in}^{-1}$		$\mathtt{in\_inv} := (\!\lvert F[\mathtt{in\_}] \rvert\!)$
in_inv_charn	:	$\mathtt{in} \circ \mathtt{in\_inv} = \mathrm{id} \wedge \mathtt{in\_inv} \circ \mathtt{in\_} = \mathrm{id}$
ana_charn	:	$\mathtt{out\_} \circ f = F[f] \circ \varphi \leftrightarrow f = [\!\lvert \varphi \rvert\!]$
ana_cancel	:	$\mathtt{out\_} \circ [\!\lvert \varphi \rvert\!] = F[[\!\lvert \varphi \rvert\!]] \circ \varphi$
ana_refl	:	$[\!\lvert \mathtt{out\_} \rvert\!] = \mathrm{id}$
ana_fusion	:	$\psi \circ f = F[f] \circ \varphi \to [\!\lvert \psi \rvert\!] \circ f = [\!\lvert \varphi \rvert\!]$
Def. of $\mathtt{out}^{-1}$		$\mathtt{out\_inv} := [\!\lvert F[\mathtt{out\_}] \rvert\!]$
out_inv_charn	:	$\mathtt{out\_inv} \circ \mathtt{out\_} = \mathrm{id} \wedge \mathtt{out\_} \circ \mathtt{out\_inv} = \mathrm{id}$
lemma2	:	$f \circ \mathtt{in\_} = \varphi \circ F[\langle f, \mathrm{id} \rangle] \leftrightarrow f = \mathsf{fst} \circ (\!\lvert \langle \varphi, \mathtt{in\_} \circ F[\mathsf{snd}] \rangle \rvert\!)$
Def. of para.		$\langle\!\lvert \varphi \rvert\!\rangle := \mathsf{fst} \circ (\!\lvert \langle \varphi, \mathtt{in\_} \circ F[\mathsf{snd}] \rangle \rvert\!)$
para_charn	:	$f \circ \mathtt{in\_} = \varphi \circ F[\langle f, \mathrm{id} \rangle] \leftrightarrow f = \langle\!\lvert \varphi \rvert\!\rangle$
para_cancel	:	$\langle\!\lvert \varphi \rvert\!\rangle \circ \mathtt{in\_} = \varphi \circ F[\langle \langle\!\lvert \varphi \rvert\!\rangle, \mathrm{id} \rangle]$
para_refl	:	$\mathrm{id} = \langle\!\lvert \mathtt{in\_} \circ F[\mathsf{fst}] \rvert\!\rangle$
para_fusion	:	$f \circ \varphi = \psi \circ F[f \otimes \mathrm{id}] \to f \circ \langle\!\lvert \varphi \rvert\!\rangle = \langle\!\lvert \psi \rvert\!\rangle$
para_cata	:	$(\!\lvert \varphi \rvert\!) = \langle\!\lvert \varphi \circ F[\mathsf{fst}] \rvert\!\rangle$
para_any	:	$f = \langle\!\lvert f \circ \mathtt{in\_} \circ F[\mathsf{snd}] \rvert\!\rangle$
Def. of apo.		$[\!\lvert \varphi \rvert\!] := [\!\lvert [\varphi, F[\mathsf{inr}] \circ \mathtt{out\_}] \rvert\!] \circ \mathsf{inl}$
apo_charn	:	$\mathtt{out\_} \circ f = F[f, \mathrm{id}] \circ \varphi \leftrightarrow f = [\!\lvert \varphi \rvert\!]$
apo_cancel	:	$\mathtt{out\_} \circ [\!\lvert \varphi \rvert\!] = F[[\![[\!\lvert \varphi \rvert\!], \mathrm{id}]\!]] \circ \varphi$
apo_refl	:	$\mathrm{id} = [\!\lvert F[\mathsf{inl}] \circ \mathtt{out\_} \rvert\!]$
apo_fusion	:	$\psi \circ f = F[f \oplus \mathrm{id}] \circ \varphi \to [\!\lvert \psi \rvert\!] \circ f = [\!\lvert \varphi \rvert\!]$
apo_ana	:	$[\!\lvert \varphi \rvert\!] = [\!\lvert F[\mathsf{inl}] \circ \varphi \rvert\!]$
apo_any	:	$f = [\!\lvert F[\mathsf{inr}] \circ \mathtt{out\_} \circ f \rvert\!]$
lemma3	:	$f \circ \mathtt{in\_} = \varphi \circ F[(\!\lvert \langle f, \mathtt{in\_inv} \rangle \rvert\!)]$ $\leftrightarrow f = \mathsf{fst} \circ \mathtt{out\_} \circ (\!\lvert \mathtt{out\_inv} \circ \langle \varphi, \mathrm{id} \rangle \rvert\!)$
Def. of histo.		$\{\!\lvert \varphi \rvert\!\} := \mathsf{fst} \circ \mathtt{out\_} \circ (\!\lvert \mathtt{out\_inv} \circ \langle \varphi, \mathrm{id} \rangle \rvert\!)$
histo_charn	:	$f \circ \mathtt{in\_} = \varphi \circ F[[\![\langle f, \mathtt{in\_inv} \rangle]\!]] \leftrightarrow f = \{\!\lvert \varphi \rvert\!\}$
histo_cancel	:	$\{\!\lvert \varphi \rvert\!\} \circ \mathtt{in\_} = \varphi \circ F[[\![\langle \{\!\lvert \varphi \rvert\!\}, \mathtt{in\_inv} \rangle]\!]]$
histo_refl	:	$\mathrm{id} = \{\!\lvert \mathtt{in\_} \circ F[\mathsf{fst} \circ \mathtt{out\_}] \rvert\!\}$
histo_fusion	:	$f \circ \varphi = \psi \circ F[[\![(f \otimes \mathrm{id}) \circ \mathtt{out\_}]\!]] \to f \circ \{\!\lvert \varphi \rvert\!\} = \{\!\lvert \psi \rvert\!\}$
histo_cata	:	$(\!\lvert \varphi \rvert\!) = \{\!\lvert \varphi \circ F[(\mathsf{fst} \circ \mathtt{out\_})] \rvert\!\}$
Def. of futu.		$[\![\varphi]\!] := [\![[\varphi, \mathrm{id}] \circ \mathtt{in\_inv}]\!] \circ \mathtt{in\_} \circ \mathsf{inl}$
futu_charn	:	$\mathtt{out\_} \circ f = F[(\!\lvert f, \mathtt{out\_inv} \rvert\!)] \circ \varphi \leftrightarrow f = [\![\varphi]\!]$
futu_cancel	:	$\mathtt{out\_} \circ [\![\varphi]\!] = F[(\!\lvert [\![\varphi]\!], \mathtt{out\_inv} \rvert\!)] \circ \varphi$
futu_refl	:	$\mathrm{id} = [\![F[\mathtt{in\_} \circ \mathsf{inl}] \circ \mathtt{out\_}]\!]$
futu_fusion	:	$\psi \circ f = F[(\!\lvert \mathtt{in\_} \circ (f \oplus \mathrm{id}) \rvert\!)] \circ \varphi \to [\![\psi]\!] \circ f = [\![\varphi]\!]$
futu_ana	:	$[\![\varphi]\!] = [\![F[(\mathtt{in\_} \circ \mathsf{inl}) \circ \varphi]]\!]$

Histomorphism

- Cata + 過去の計算履歴へのアクセス可能
  - 余自由余モナドという中間データ構造を使う
  - DP っぽいことができる
- 例: Unbounded Knapsack Problem

```
Definition knapsack (wvs : list (nat * nat))
:= {|
    [fun _ => 0,
     fun t => maximize
              (fun p => match p with
                        | (w,v) => match t !! (w - 1) with
                                   | Nothing => 0
                                   | Just a  => v + a
                                   end
                        end) wvs]
    |}.
```

Outline

1. 概要（万人向けのお話）
2. F 始代数と catamorphism（ざっくりと）
3. 始代数のための型クラス
4. 今後とまとめ

F 始代数と catamorphism （ざっくりと）

- 😭 チョットだけ圏論の言葉を使います
  - ただし圏 $\mathbf{Set}$ だけ考える
    （つまり全域関数だけ考え，partiality は考えない）
- **F 始代数**: ADT に対応
  - 多項式関手 F のによって様々な ADT が作れる
  - E.g. $L_A(X) = \mathbf{1} + (A \times X)$:
    L_A 始代数は A 上のリスト $\mathbf{list}\ A$
- **Catamorphism**: ADT を「たたみ込んで」一つの値を返す計算に対応
  - いわゆる fold

F 始代数と catamorphism （圏論的に）

- 多項式関手 $F\colon \mathbf{Set} \to \mathbf{Set}$
  - $(\mu F, \mathsf{in}_F)$ が F 始代数（F-initial algebra）

$\Longleftrightarrow$

$$F(\mu F) \overset{\mathsf{in}_F}{\to} \mu F$$

F 始代数と catamorphism （圏論的に）

- 多項式関手 $F\colon \mathbf{Set} \to \mathbf{Set}$
  - $(\mu F, \mathsf{in}_F)$ が F 始代数（F-initial algebra）
    $\iff$ 任意の $\varphi\colon F(X) \to X$ に対して，

$$F(\mu F) \overset{\mathsf{in}_F}{\to} \mu F$$

$$F(X) \underset{\varphi}{\longrightarrow} X$$

F 始代数と catamorphism （圏論的に）

- 多項式関手 $F\colon \mathbf{Set} \to \mathbf{Set}$
  - $(\mu F, \mathsf{in}_F)$ が F 始代数（F-initial algebra）
    $\iff$ 任意の $\varphi\colon F(X) \to X$ に対して，或る
    $f\colon \mu F \to X$ が唯一存在し，$f \circ \mathsf{in}_F = \varphi \circ F(f)$

$$\begin{array}{ccc} F(\mu F) & \overset{\mathsf{in}_F}{\to} & \mu F \\ {\scriptstyle F(f)}\downarrow & & \downarrow{\scriptstyle f} \\ F(X) & \underset{\varphi}{\longrightarrow} & X \end{array}$$

F 始代数と catamorphism （圏論的に）

- 多項式関手 $F\colon \mathbf{Set} \to \mathbf{Set}$
  - $(\mu F, \mathsf{in}_F)$ が F 始代数（F-initial algebra）
    $\iff$ 任意の $\varphi\colon F(X) \to X$ に対して，或る
    $f\colon \mu F \to X$ が唯一存在し，$f \circ \mathsf{in}_F = \varphi \circ F(f)$

$$\begin{array}{ccc} F(\mu F) & \overset{\mathsf{in}_F}{\to} & \mu F \\ {\scriptstyle F(f)}\downarrow & & \downarrow{\scriptstyle f} \\ F(X) & \underset{\varphi}{\longrightarrow} & X \end{array}$$

  - 唯一の射 f を φ の catamorphism といい，$(\!|\varphi|\!)_F$ と書く：$f \circ \mathsf{in}_F = \varphi \circ F(f) \iff f = (\!|\varphi|\!)_F$

始代数について成り立つ性質

- $(\mu F, \mathsf{in}_F)$: F 始代数
- $\mathsf{in_F}$: $F(\mu F) \to \mu F$ は**同型射** (Lambeck)

$$\mathsf{in}_F^{-1} \circ \mathsf{in}_F = \mathrm{id}_{F(\mu F)}, \quad \mathsf{in}_F \circ \mathsf{in}_F^{-1} = \mathrm{id}_{\mu F}$$

  - つまり $F(\mu F) \cong \mu F$

$$F(\mu F) \underset{\mathsf{in}_F^{-1}}{\overset{\mathsf{in}_F}{\rightleftarrows}} \mu F$$

  - なお，in_F^{-1} は具体的に構成可: $\mathsf{in}_F^{-1} = (\!| F(\mathsf{in}_F) |\!)_F$

例題 リスト関手 $L_A \colon \mathbf{Set} \to \mathbf{Set}$

$$\begin{aligned} L_A(X) &= \mathbf{1} + (A \times X) \\ L_A(f) &= \mathrm{id}_{\mathbf{1}} + (\mathrm{id}_A \times f) \end{aligned}$$

の L_A 始代数および catamorphism $(\!|\varphi|\!)_{L_A}$ は？

例題 リスト関手 $L_A : \mathbf{Set} \to \mathbf{Set}$

$$\begin{aligned} L_A(X) &= \mathbf{1} + (A \times X) \\ L_A(f) &= \mathrm{id}_{\mathbf{1}} + (\mathrm{id}_A \times f) \end{aligned}$$

の L_A 始代数および catamorphism $(\!|\varphi|\!)_{L_A}$ は？

L_A 始代数: $(\mathbf{list}\ A,\ \mathsf{in}_{L_A})$

$$\mathsf{in}_{L_A} = \left[\begin{array}{l} \boldsymbol{\lambda}\ \_ \Rightarrow [\,] \\ \boldsymbol{\lambda}\ (x,\ xs) \Rightarrow x :: xs \end{array} \right] : L_A(\mathbf{list}\ A) \to \mathbf{list}\ A$$

例題 リスト関手 $L_A : \mathbf{Set} \to \mathbf{Set}$

$$\begin{aligned} L_A(X) &= \mathbf{1} + (A \times X) \\ L_A(f) &= \mathrm{id}_{\mathbf{1}} + (\mathrm{id}_A \times f) \end{aligned}$$

の L_A 始代数および catamorphism $(\!|\varphi|\!)_{L_A}$ は？

L_A 始代数: $(\mathbf{list}\ A,\ \mathsf{in}_{L_A})$

$$\mathsf{in}_{L_A} = \left[\begin{array}{l} \boldsymbol{\lambda}\ \_ \Rightarrow [\,] \\ \boldsymbol{\lambda}\ (x,\ xs) \Rightarrow x :: xs \end{array}\right] : L_A(\mathbf{list}\ A) \to \mathbf{list}\ A$$

$$\mathbf{1} + (A \times \mathbf{list}\ A)$$

例題 リスト関手 $L_A : \mathbf{Set} \to \mathbf{Set}$

$$\begin{aligned} L_A(X) &= \mathbf{1} + (A \times X) \\ L_A(f) &= \mathrm{id}_{\mathbf{1}} + (\mathrm{id}_A \times f) \end{aligned}$$

の L_A 始代数および catamorphism $(\!|\varphi|\!)_{L_A}$ は？

L_A 始代数: $(\mathbf{list}\ A,\ \mathsf{in}_{L_A})$

$$\mathsf{in}_{L_A} = \left[\begin{array}{l} \boldsymbol{\lambda}\ \_ \Rightarrow [\,] \\ \boldsymbol{\lambda}\ (x,\ xs) \Rightarrow x :: xs \end{array}\right] : L_A(\mathbf{list}\ A) \to \mathbf{list}\ A$$

$$\mathbf{1} + (A \times \mathbf{list}\ A)$$

pointwise に:

$$\begin{aligned} \mathsf{in}_{L_A}\ (\mathsf{inl}\ ()) &= [\,] \\ \mathsf{in}_{L_A}\ (\mathsf{inr}\ (x,\ xs)) &= \mathsf{cons}\ x\ xs \end{aligned}$$

例題 リスト関手 $L_A : \mathbf{Set} \to \mathbf{Set}$

$$\begin{aligned} L_A(X) &= \mathbf{1} + (A \times X) \\ L_A(f) &= \mathrm{id}_{\mathbf{1}} + (\mathrm{id}_A \times f) \end{aligned}$$

の L_A 始代数および catamorphism $(\!|\varphi|\!)_{L_A}$ は？

L_A 始代数: $(\mathbf{list}\ A,\ \mathsf{in}_{L_A})$

$$\mathsf{in}_{L_A} = \left[\begin{array}{l} \boldsymbol{\lambda}\ \_ \Rightarrow [\,] \\ \boldsymbol{\lambda}\ (x,\ xs) \Rightarrow x :: xs \end{array}\right] : L_A(\mathbf{list}\ A) \to \mathbf{list}\ A$$

$$\mathbf{1} + (A \times \mathbf{list}\ A)$$

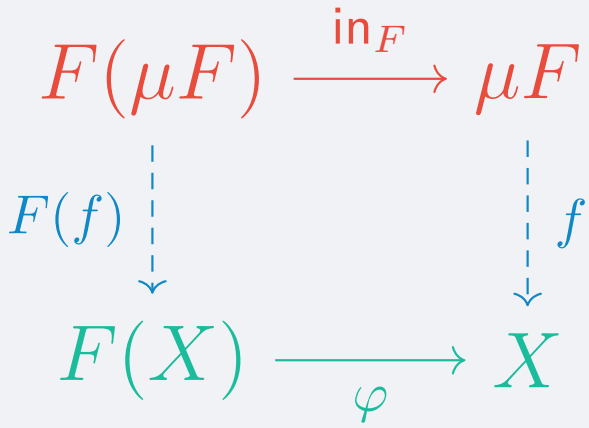
pointwise に:

$$\begin{aligned} \mathsf{in}_{L_A}\ (\mathsf{inl}\ ()) &= [\,] \\ \mathsf{in}_{L_A}\ (\mathsf{inr}\ (x,\ xs)) &= \mathsf{cons}\ x\ xs \end{aligned}$$

始代数: 引数を受け取って適当なデータ構造を作る
（つまり場合分けの入ったコンストラクタ）

$$\varphi \colon L_A(X) \to X$$

$$\mathbf{1} + (A \times X)$$

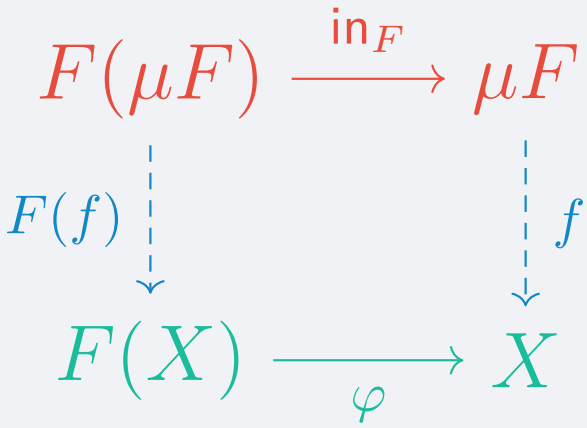


$$\varphi \colon L_A(X) \to X$$

$$\mathbf{1} + (A \times X)$$

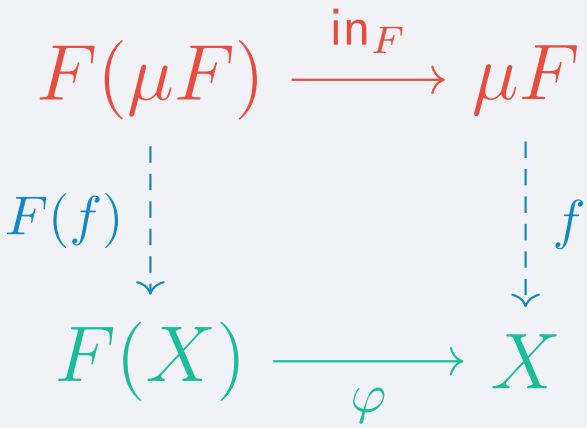
型から言って φ は次の形:

$$\varphi = \left[\begin{array}{lcl} \boldsymbol{\lambda}\, \_ & \Rightarrow & c \\ \boldsymbol{\lambda}\, (x, xs) & \Rightarrow & x \circledast xs \end{array} \right]$$



$$\varphi \colon L_A(X) \to X$$

$$\mathbf{1} + (A \times X)$$



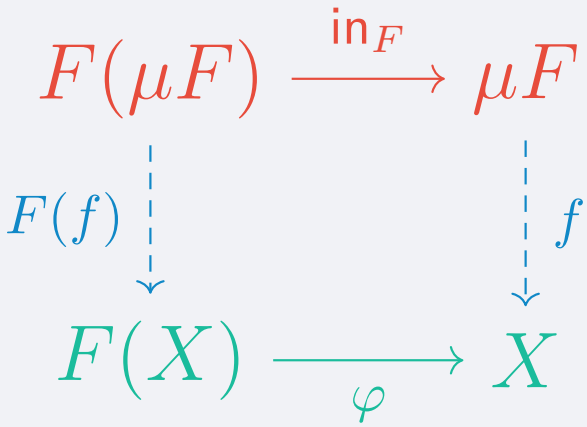
型から言って φ は次の形:

$$\varphi = \begin{bmatrix} \boldsymbol{\lambda}\, \_ & \Rightarrow & c \\ \boldsymbol{\lambda}\, (x, xs) & \Rightarrow & x \circledast xs \end{bmatrix}$$

$$\begin{aligned} & f \circ \mathsf{in}_F = \varphi \circ L_A(f) \\ \iff \quad & \{\ \text{定義を展開}\ \} \\ & f \circ \begin{bmatrix} \boldsymbol{\lambda}\, \_ \Rightarrow [\,] \\ \boldsymbol{\lambda}\, (x,\ xs) \Rightarrow x :: xs \end{bmatrix} \\ & = \begin{bmatrix} \boldsymbol{\lambda}\, \_ \Rightarrow c \\ \boldsymbol{\lambda}\, (x, xs) \Rightarrow x \circledast xs \end{bmatrix} \circ \begin{bmatrix} \mathsf{inl} \circ \mathrm{id}_{\mathbf{1}} \\ \mathsf{inr} \circ \langle \mathrm{id}_A,\ f \rangle \end{bmatrix} \end{aligned}$$

$\varphi \colon L_A(X) \to X$

$\mathbf{1} + (A \times X)$



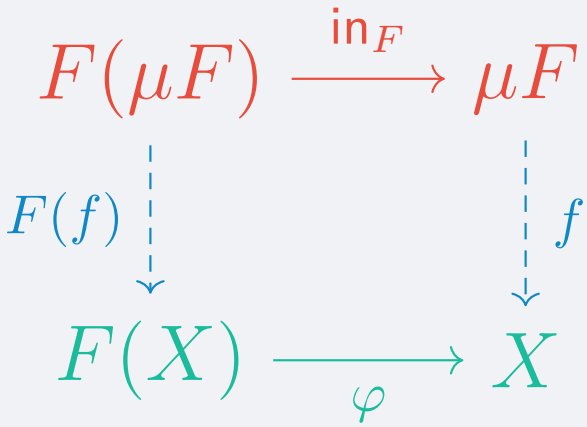
型から言って φ は次の形:

$$\varphi = \begin{bmatrix} \boldsymbol{\lambda}\, \_ & \Rightarrow & c \\ \boldsymbol{\lambda}\, (x, xs) & \Rightarrow & x \circledast xs \end{bmatrix}$$

$$\begin{aligned}
& f \circ \mathsf{in}_F = \varphi \circ L_A(f) \\
\iff \quad & \{\ \text{定義を展開}\ \} \\
& f \circ \begin{bmatrix} \boldsymbol{\lambda}\, _ \Rightarrow [\,] \\ \boldsymbol{\lambda}\, (x,\ xs) \Rightarrow x :: xs \end{bmatrix} \\
& = \begin{bmatrix} \boldsymbol{\lambda}\, _ \Rightarrow c \\ \boldsymbol{\lambda}\, (x, xs) \Rightarrow x \circledast xs \end{bmatrix} \circ \begin{bmatrix} \mathsf{inl} \circ \mathrm{id}_{\mathbf{1}} \\ \mathsf{inr} \circ \langle \mathrm{id}_A,\ f \rangle \end{bmatrix} \\
\iff \quad & \{\ \text{pointwise に書けば}\ \} \\
& \begin{cases} f\ [\,] & = & c \\ f\ (x :: xs) & = & x \circledast (f\ xs) \end{cases}
\end{aligned}$$

$$\varphi \colon L_A(X) \to X$$

$$\mathbf{1} + (A \times X)$$



型から言って φ は次の形:

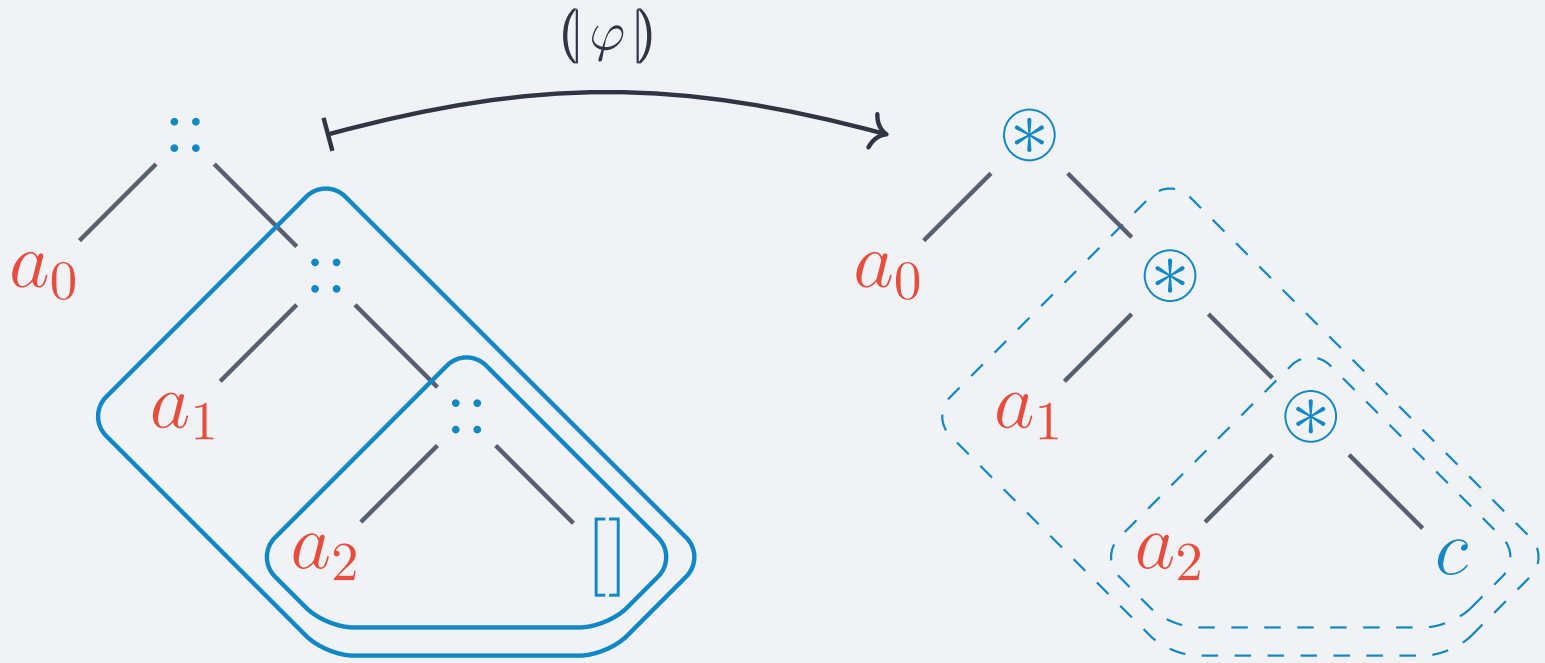
$$\varphi = \begin{bmatrix} \boldsymbol{\lambda}\ \_ & \Rightarrow & c \\ \boldsymbol{\lambda}\ (x, xs) & \Rightarrow & x \circledast xs \end{bmatrix}$$

$$\begin{aligned}
& f \circ \mathsf{in}_F = \varphi \circ L_A(f) \\
\iff \quad & \{\ \text{定義を展開}\ \} \\
& f \circ \begin{bmatrix} \boldsymbol{\lambda}\ _ \Rightarrow [\,] \\ \boldsymbol{\lambda}\ (x,\ xs) \Rightarrow x :: xs \end{bmatrix} \\
& = \begin{bmatrix} \boldsymbol{\lambda}\ _ \Rightarrow c \\ \boldsymbol{\lambda}\ (x, xs) \Rightarrow x \circledast xs \end{bmatrix} \circ \begin{bmatrix} \mathsf{inl} \circ \mathrm{id}_{\mathbf{1}} \\ \mathsf{inr} \circ \langle \mathrm{id}_A,\ f \rangle \end{bmatrix} \\
\iff \quad & \{\ \text{pointwise に書けば}\ \} \\
& \begin{cases} f\ [\,] & = & c \\ f\ (x :: xs) & = & x \circledast (f\ xs) \end{cases}
\end{aligned}$$

要するに *foldr*

Catamorphism（直観的には）

- ADT のコンストラクタを関数に置き換え



さて，色々言ったが大事なことは…

さて，色々言ったが大事なことは…

Catamorphism の特徴づけは

$$f \circ \mathsf{in}_F = \varphi \circ F(f) \iff f = (\!|\varphi|\!)$$

であること（Cata-Charn）

Outline

1. 概要（万人向けのお話）
2. F 始代数と catamorphism（ざっくりと）
3. 始代数のための型クラス
4. 今後とまとめ

多項式関手にまつわる定義 (1/3)

- 多項式関手を表す型 `PolyF : Type`

```
Inductive PolyF : Type :=
| zer  : PolyF
| one  : PolyF
| arg1 : PolyF
| arg2 : PolyF
| Sum  : PolyF -> PolyF -> PolyF
| Prod : PolyF -> PolyF -> PolyF.
```

  - AST の形で形式的に持っている
    - 細かい話: L_A のような indexing された関手を表現するため，実は bifunctor になっている

多項式関手にまつわる定義 (2/3)

- F : PolyF から具体的な関手
  $[\![F]\!]\ A$: Type $\to$ Type を返す関数

```
Fixpoint inst (F : PolyF) (A X : Type) : Type :=
  match F with
    zer => Empty_set | one => unit
  | arg1 => A | arg2 => X
  | Sum F G  => (inst F A X) + (inst G A X)
  | Prod F G => (inst F A X) * (inst G A X)
  end.
Notation "[[ F ]]" := (inst F) (at level 20).
```

多項式関手にまつわる定義 (3/3)

- いわゆる fmap

```
Fixpoint fmap (F : PolyF) {A0 A1 X0 X1 : Type}
  (f : A0 -> A1) (g : X0 -> X1) : ⟦F⟧ A0 X0 -> ⟦F⟧ A1 X1 :=
  match F with
    zer => id | one => id | arg1 => f | arg2 => g
  | Sum F G  => fun x
                => match x with
                   | inl x => inl (fmap F f g x)
                   | inr x => inr (fmap G f g x)
                   end
  | Prod F G => fun x
                => (fmap F f g (fst x), fmap G f g (snd x))
  end.
```

- $[\![F]\!]\ A : \mathrm{Type} \to \mathrm{Type}$ が関手則を満たす証明
- ほか，色々な Notation

始代数のための型クラス

```
Class F_initial_algebra (F : PolyF) (A mF : Type)
:={
    cata : forall (X : Type), (⟦F⟧ A X -> X) -> (mF -> X);
    in_  : ⟦F⟧ A mF -> mF;
    cata_charn : forall (X : Type) (f : mF -> X)
                        (φ : ⟦F⟧ A X -> X),
        f ∘ in_ = φ ∘ F[f] <-> f = cata X φ
  }.
```

- 始代数・Catamorphism の定義を素直に形式化
  - インスタンスは Coq プログラムとしての in_F と $(\!|\varphi|\!)_F$ の定義（動く）
  - Cata-Charn の証明も要求

```
Instance list_ia (A : Type)
  : F_initial_algebra (Sum one (Prod arg1 arg2)) A (list A)
    := {
          cata := list_cata A;
          in_  := list_in A
       }.
Proof. (* 残りの cata_charn の証明 *)
  intros. split.
  - intros H; extensionality l; induction l.
    + specialize (equal_f H (inl tt)) as H0.
      cbv in H0; cbv; easy.
    + specialize (equal_f H (inr (a, l))) as H0.
      cbv in H0; cbv. rewrite H0. rewrite IHl. easy.
  - intros H; extensionality l. induction l.
    + rewrite H; induction a;
      specialize (equal_f H []) as H0. easy.
    + rewrite H. induction b;
      specialize (equal_f H (cons a b)) as H0. easy.
Defined.
```

動く再帰図式プログラム

- ここまでの準備のもと，再帰図式でプログラムを書いて実行できるように

```
Definition map {F : PolyF} {mF A B : Type}
           {ia1 : F_initial_algebra F A mF}
           {ia2 : F_initial_algebra F B mF}
           (f : A -> B)
:= (| in_ ∘ F⟨ f ⟩[id] |).

Eval cbv in (map (plus 10) [1 ; 2 ; 3 ; 4 ; 5]).
```

ちょっとだけ自動化

```
cata_charn : forall (X : Type) (f : mF -> X)
                    (φ : ⟦F⟧ A X -> X),
    f ∘ in_ = φ ∘ F[f] <-> f = cata X φ
```

- マトモな in_ と cata の定義が与えられていれば, Cata-charn の証明は自動化できそう
- <- の自動化は易しい
  - $\forall x : F(X), (\llparenthesis \varphi \rrparenthesis \circ \mathsf{in}_F)\ x = (\varphi \circ F[\llparenthesis \varphi \rrparenthesis])\ x$ を示せばよい
  - 丁寧に場合分けするだけ→実装

```
Instance list_ia (A : Type)
  : F_initial_algebra (Sum one (Prod arg1 arg2)) A (list A)
    := {
         cata := list_cata A;
         in_  := list_in A
       }.
Proof.
  intros. split.
  - auto_instance_onlyif. (* AUTOMIZED *)
  - intros H; extensionality l. induction l.
    + rewrite H; induction a;
      specialize (equal_f H []) as H0. easy.
    + rewrite H. induction b;
      specialize (equal_f H (cons a b)) as H0. easy.
Defined.
```

- 5 例で成功を確認

Outline

1. 概要（万人向けのお話）
2. F 始代数と catamorphism（ざっくりと）
3. 始代数のための型クラス
4. 今後とまとめ

本研究の貢献

- 運算のための Coq ライブラリ [Tesson+, 2011] を拡張
- **再帰図式に基づく ADT-generic な規則を証明可能に**
- **再帰図式を使った Coq プログラミングを可能に**
  - プログラムには定理を適用可（運算ができる）
  - プログラムは実行・抽出が可能
  - ユーザは型クラスのインスタンス化を通して型ごとの実装を与える
    - 手間を省くため**インスタンス化の一部を自動化**

今後

- 逆方向の自動化
  - やればできるはずだがまだ綺麗にはできていない
  - 帰納法の仮定を使うタイミングなどが難しい
- Refolding 系の再帰図式 (hylomorphism など)
  - 重要な運算定理が多く知られている
- 再帰図式の統一的な扱い
  - 余モナドを使ったもの [Uustalu+, 2001]
  - Adjoint fold を使ったもの [Hinze+, 2013]