Coq における検証された プログラム運算の拡張

村田 康佑　　江本 健斗

九州工業大学

2018 年 8 月 31 日　JSSST 第 35 回大会 @大阪

まとめ

- **プログラム運算**: プログラムの意味を保存しながら代数的変形
  - 変形法則の**正当性の検証**をしたい
- **Coq 上でプログラム運算を行う試みを拡張を紹介**
  - 先行研究 [Tesson+, 2011]: 主にリストへの適用
    - Bird, *An Introduction to the Theory of Lists* の例を実装
  - **本研究: 一般の代数的データ型への適用**
    - Bird, De Moor: *Algebra of Programming* の例を (一部) 実装
- **適用例**: バナナ・スプリット則等の検証
- **課題**: 効率化検証，適用例の証明，ana や hylo・Allegory へ

プログラムなど

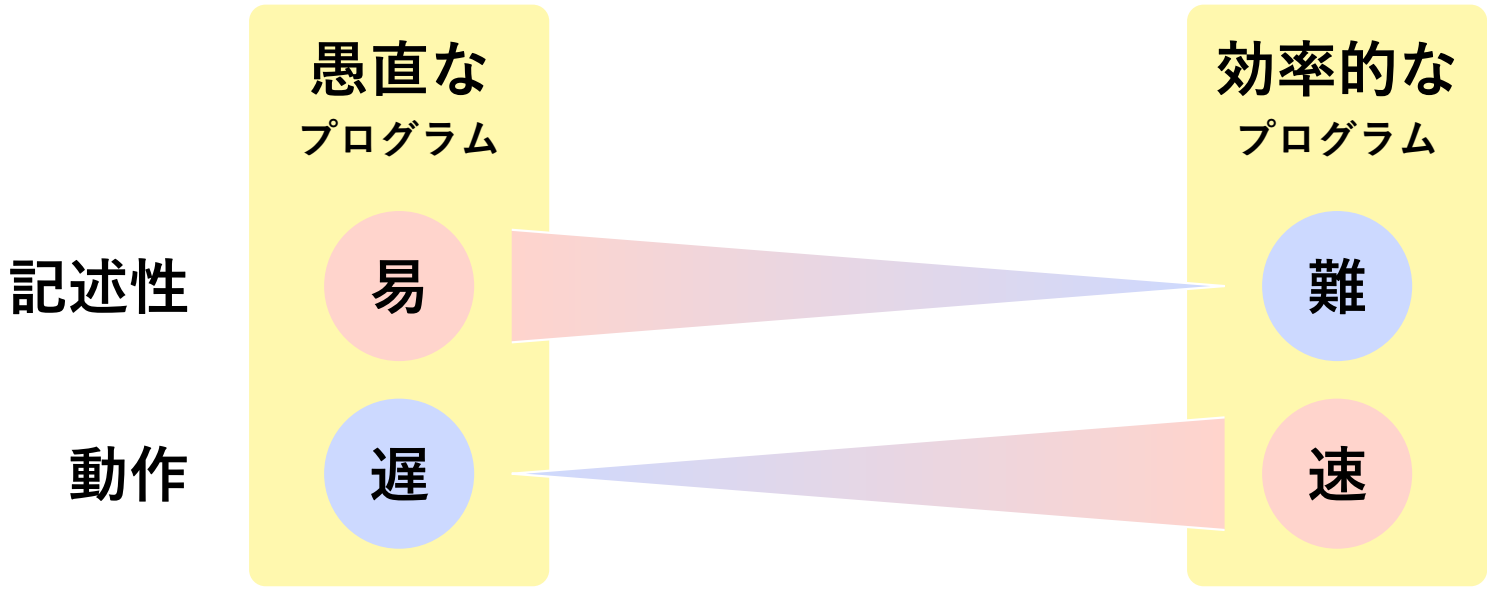
- https://github.com/MurataKosuke/AoP_in_Coq
- **プログラム・このスライド・予稿の正誤表**

Outline

1. 背景
2. 先行研究の紹介
3. 準備: 一般の代数的データ型への拡張
4. 運算規則の実装
5. 今後の課題とまとめ

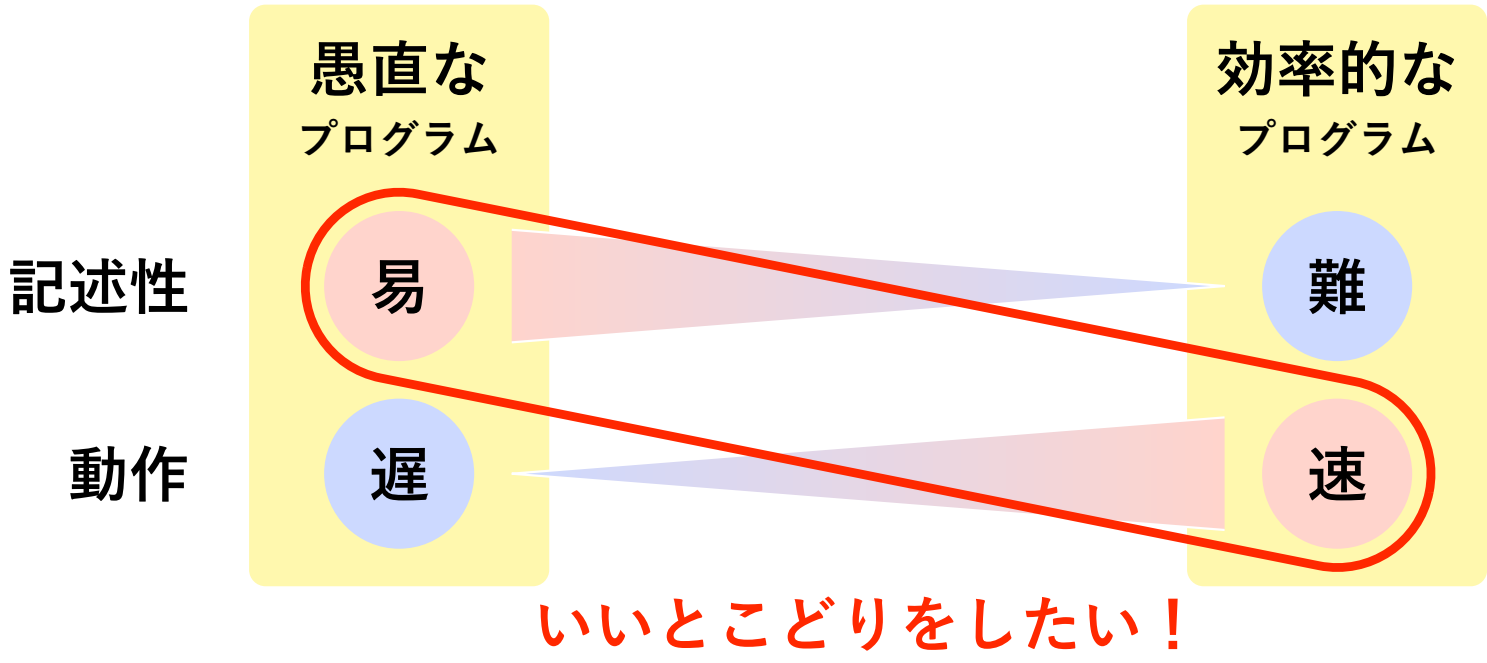
背景

- 効率的なプログラムを簡単に書けるようにしたい



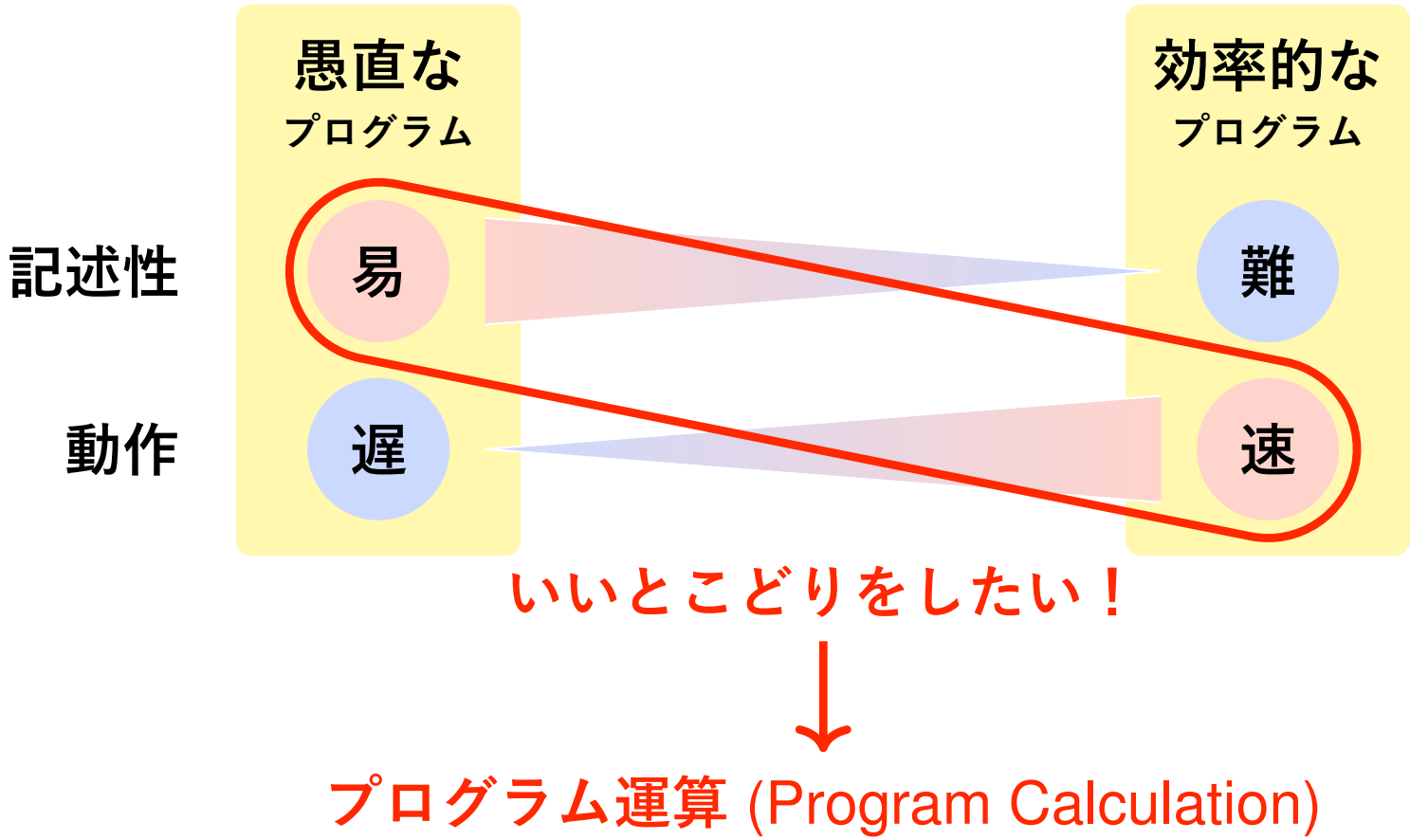
背景

- 効率的なプログラムを簡単に書けるようにしたい



背景

- 効率的なプログラムを簡単に書けるようにしたい



背景: プログラム運算とは (1)

- プログラムの意味を変えない**代数的変形**を繰り返す
  - 愚直な記述から効率的なプログラムが得られることが期待
- プログラムについての多数の代数的法則を利用
  - 例: map 分配則

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

遅　　速

背景: プログラム運算とは (2)

- 有名な運算の例: 最大部分和問題

$$
\begin{aligned}
& maximum \circ (map\ sum) \circ segs \\
= \quad & maximum \circ (map\ sum) \circ (concatMap\ inits) \circ tails \\
= \quad & maximum \circ (map\ sum) \circ concat \circ (map\ inits) \circ tails \\
& \vdots \\
= \quad & maximum \circ (scanr\ (\odot)\ 0) \\
& \quad \text{where } x \odot y = max\ 0\ (x + y)
\end{aligned}
$$

本研究の目的

- プログラムの代数的法則の正しさを保証したい
  - つまり，正当性の証明をつけたい
  - そしてその証明を機械的にチェックしたい
    - 人力証明は疲れる，バグが入る，…
  - なお，速さの保証はひとまずスコープ外 (今後の課題)
- 定理証明支援系 Coq を利用

背景: 定理証明支援系とは

- ユーザが形式的証明を記述するのを補助
- 型検査器 (type checker) により証明を検証
  - Curry-Howard 同型対応を応用
- 応用先: プログラム検証・数学の証明の正しさ保証
  - C コンパイラの検証 (CompCert) [Leroy, X.+, 2009]
  - 四色定理の証明の検証 [Gontheir, G., 2008]
- Coq, Isabelle/HOL, Agda などが有名
- 最近 Coq/SSReflect/MathComp の日本語教科書が出版された
  - 萩原学，アフェルト・レナルド．Coq/SSReflect/MathComp による定理証明:フリーソフトではじめる数学の形式化．森北出版．2018.
  - 国内でも注目され始めている

Outline

1. 背景
2. 先行研究の紹介
3. 準備: 一般の代数的データ型への拡張
4. 運算規則の実装
5. 今後の課題とまとめ

Introduction to Theory of Lists

- Bird によるプログラム運算のレクチャーノート
- BMF (Bird Meertens Formalism) に基づくリスト上の運算
  - e.g. フィルタ・プロモーション則

$$
\begin{aligned}
& (p \lhd) \circ (+\!\!+ /) \\
= \quad & (+\!\!+ /) \circ (f *) \circ (+\!\!+ /) \\
= \quad & (+\!\!+ /) \circ (+\!\!+ /) \circ ((f *) *) \\
= \quad & (+\!\!+ /) \circ ((+\!\!+ /) *) \circ ((f *) *) \\
= \quad & (+\!\!+ /) \circ (((+\!\!+ /) \circ ((f *))) *) \\
= \quad & (+\!\!+ /) \circ ((p \lhd) *)
\end{aligned}
$$

フィルタ・プロモーション則の証明

基本的なリスト演算

- **結合** $(+\!\!+)$
- **フィルタ** $(\lhd)$
- **マップ** $(*)$
- **リデュース** $(/)$

先行研究 [Tesson+, 2011]

- Coq によるプログラム運算のためのタクティックライブラリを実装
  - Theory of Lists へ適用
  - 教科書に近い記法で記述することが可能

```
1  Theorem filter promotion :
2    p <| :o: @concat A = @concat A :o: p <| *.
3  Proof.
4      LHS
5  = { rewrite (filter mapreduce) }
6      (++ / :o: f * :o: @concat A).
7  = { rewrite map promotion }
8      (++ /:o: @concat (list A) :o: f* *).
9  = { rewrite comp assoc }
10     ((++ /:o: @concat (list A)) :o: f* *).
11 = { rewrite reduce promotion }
12     (++ / :o: (++ /)* :o: f* *).
13 = { rewrite concat reduce }
14     (@concat A :o: (++ /)* :o: f* *).
15 = { rewrite map distr comp }
16     (@concat A :o: (++ / :o: f *) *).
17 = { rewrite filter mapreduce }
18     (@concat A :o: (p <|) *).
19 [].
20 Qed.
```

Outline

1. 背景
2. 先行研究の紹介
3. 準備: 一般の代数的データ型への拡張
4. 運算規則の実装
5. 今後の課題とまとめ

一般の代数的データ型

- 先行研究: 主にリストへの適用
- 一般の代数的データ型への拡張
  - 代数的データ型: 自然数，リスト，木，…
  - Coq では `Inductive` コマンドを用いて定義
  - なお，ここでは余代数は含まない
- 教科書 Bird, De Moor: *Algebra of Programming*
  - 一般の代数的データ型へのプログラム運算の拡張
  - 圏論 (category theory) の言葉を使って議論
    - 代数的データ型 $=$ F 始代数

圏とは?

- 圏 (category): 「対象」と「射」からなる代数系
  - 1 つの圏のなかで，始対象，終対象，積，和 (余積) など，様々な概念が定義される
- 関手 (functor): 圏と圏の間の準同型
- ここでは，ざっくり言えば
  - 圏の対象: 型 (集合)
  - 圏の射: 型 A から型 B の関数
  - 関手: 型レベルの関数 (型演算; 型をとって型を返す関数)

  だと思えばよい
- 圏の例: 集合と関数の圏 **Set**…

始対象 (initial object)

- **始対象 0**
  - 集合の圏 Set では，$0 = \emptyset$

始対象 (initial object)

- **始対象 0**
  - 集合の圏 Set では，$0 = \emptyset$

始対象 (initial object)

- **始対象 0**
  - 集合の圏 Set では，$0 = \emptyset$


```
1  Inductive InitialObj : Type :=.
```

終対象 (terminal object)

- **終対象**
  - 集合の圏 **Set** では，$1 = \{☺\}$ (シングルトン)

1

終対象 (terminal object)

- **終対象**
  - 集合の圏 Set では，$1 = \{☺\}$ (シングルトン)

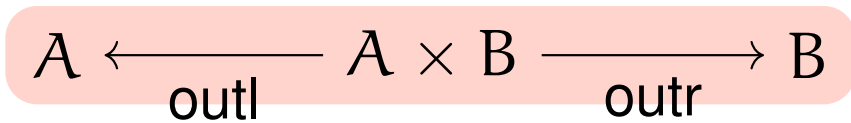
終対象 (terminal object)

- **終対象**
  - 集合の圏 Set では，$1 = \{☺\}$ (シングルトン)


```
1  Inductive TerminalObj : Type :=
2  |  singletonobj : TerminalObj.
```

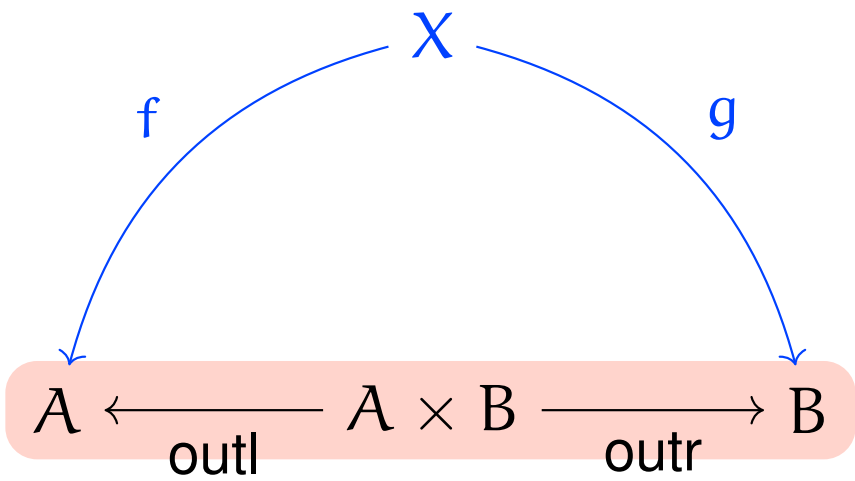
積 (product)

- 積 $A \times B$
  - 集合の圏 Set では，$A \times B = \{(a, b) \mid a \in A, b \in B\}$



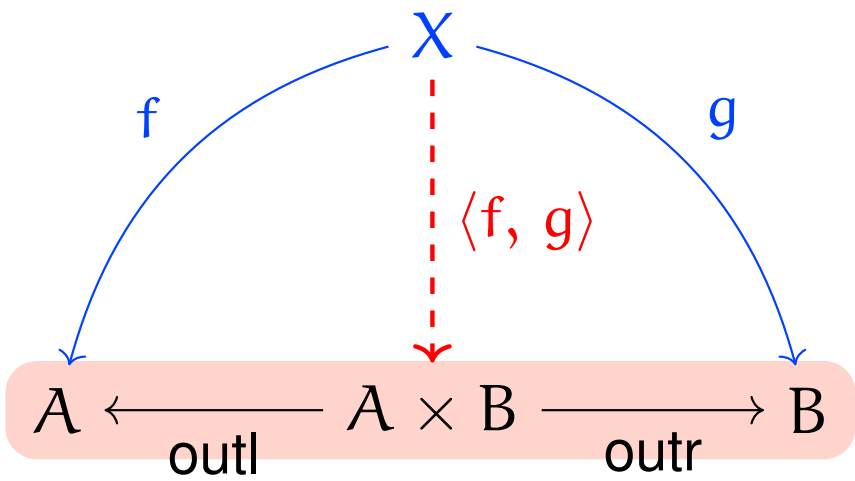
積 (product)

- **積 $A \times B$**
  - **集合の圏 Set では，** $A \times B = \{(a, b) \mid a \in A, b \in B\}$



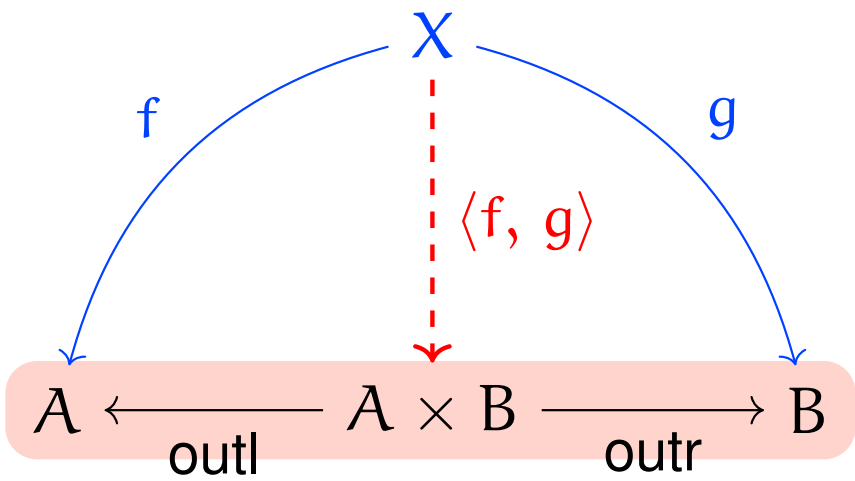
積 (product)

- **積 $A \times B$**
  - **集合の圏 Set では，** $A \times B = \{(a, b) \mid a \in A, b \in B\}$



積 (product)

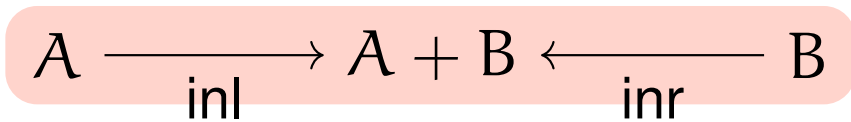
- **積 $A \times B$**
  - **集合の圏 Set では，** $A \times B = \{(a, b) \mid a \in A, b \in B\}$




```
1  Inductive Pair {A B : Type} : Type :=
2  | pair (x : A) (y : B) : @Pair A B.
```

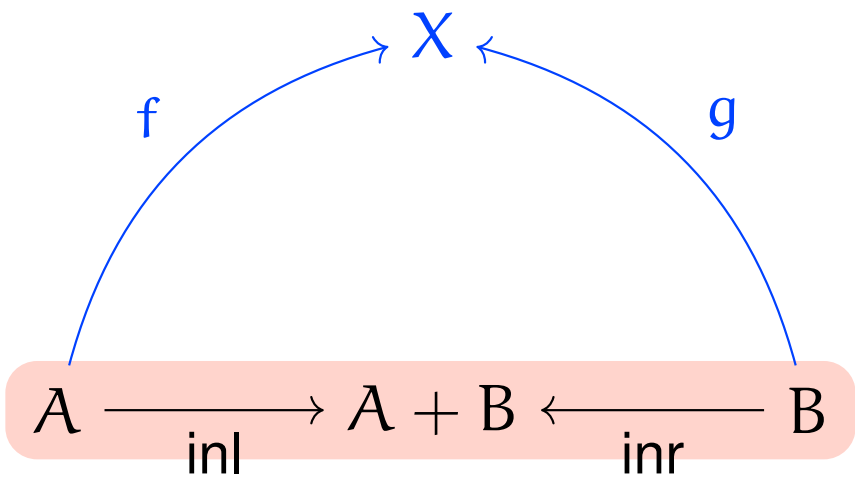
余積 (coproduct)・和 (sum)

- **余積・和** $A + B$
  - **圏 Set では，** $A + B = \{\mathtt{inl}\ a \mid a \in A\} \cup \{\mathtt{inr}\ b \mid b \in B\}$



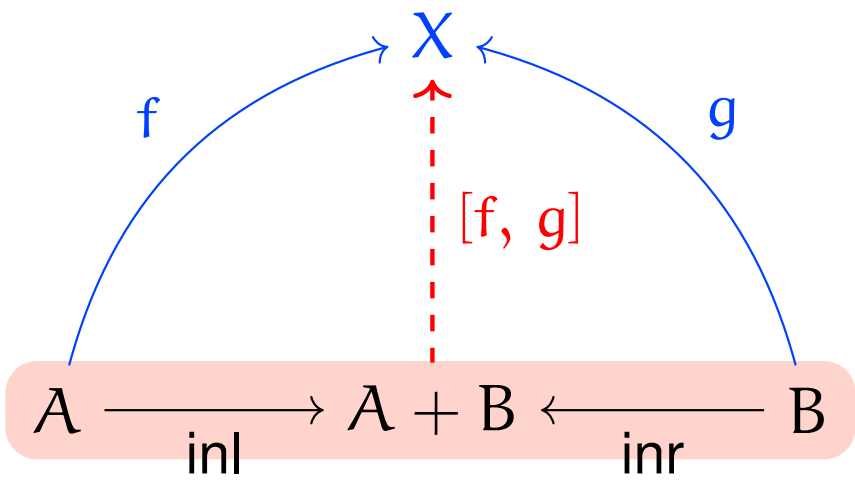
余積 (coproduct)・和 (sum)

- **余積・和** $A + B$
  - **圏 Set では，** $A + B = \{\mathtt{inl}\ a \mid a \in A\} \cup \{\mathtt{inr}\ b \mid b \in B\}$



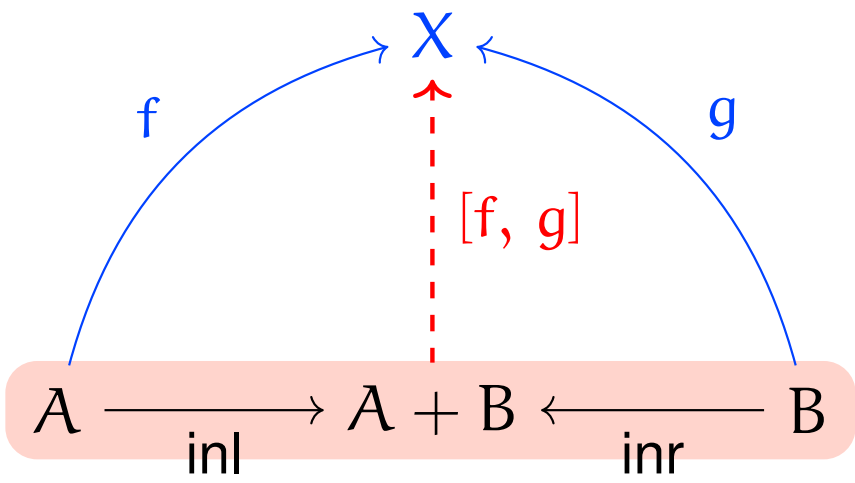
余積 (coproduct)・和 (sum)

- **余積・和** $A + B$
  - **合の圏 Set では，** $A + B = \{\mathtt{inl}\ a \mid a \in A\} \cup \{\mathtt{inr}\ b \mid b \in B\}$



余積 (coproduct)・和 (sum)

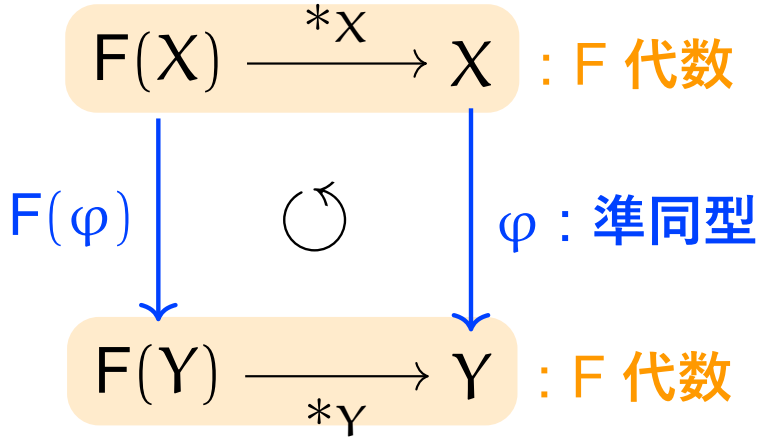
- **余積・和** $A + B$
  - **圏 Set では，** $A + B = \{\mathtt{inl}\ a \mid a \in A\} \cup \{\mathtt{inr}\ b \mid b \in B\}$




```
1  Inductive Either {A B : Type} : Type :=
2  | inl (a : A) : @Either A B
3  | inr (b : B) : @Either A B.
```

F 始代数

- 自己関手 $F : \mathbf{Set} \to \mathbf{Set}$
- F 代数: 射 $* \in \mathrm{Hom}_{\mathbb{C}}(F(X), X)$
- F 代数の間の準同型
  - 右の図式を可換にする φ
- F 代数とその間の準同型がなす圏: $\mathbf{Alg}_F$
  - F が多項式関手のとき $\mathbf{Alg}_F$ は必ず始対象を持つ: F 始代数
  - F 始代数が代数的データ型に対応



F 始代数と catamorphism

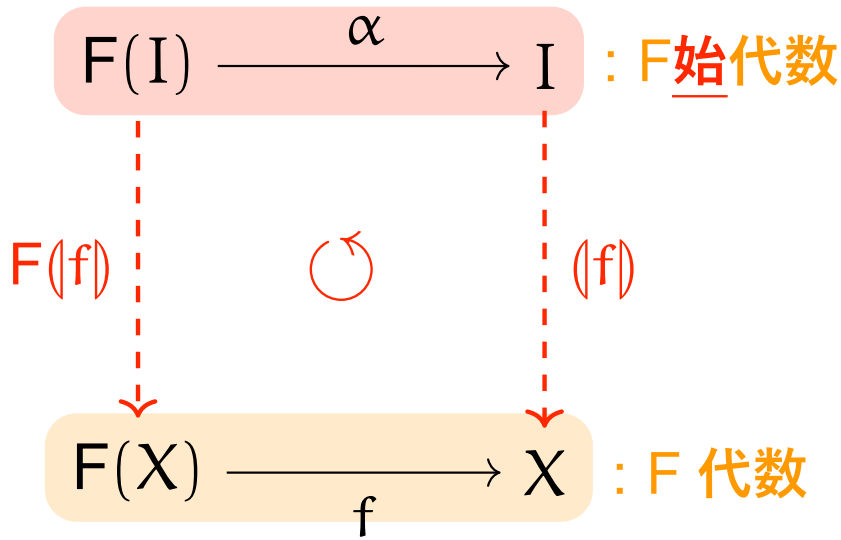
- f の **catamoprhism**: F 始代数 $\alpha : F(I) \to I$ から F 代数 $f : F(X) \to X$ への準同型
  - F に応じた再帰パタンを表現
  - Haskell でいう `foldr` の一般化

$$F(I) \xrightarrow{\alpha} I$$: F始代数

$$F(X) \xrightarrow[f]{} X$$: F 代数

F 始代数と catamorphism

- f の **catamoprhism**: F 始代数 $\alpha : F(I) \to I$ から F 代数 $f : F(X) \to X$ への準同型
  - F に応じた再帰パタンを表現
  - Haskell でいう `foldr` の一般化



F 始代数の例: nat(1)

- 自己関手 F を次のように定義

$$\begin{aligned} F(X) &= 1 + X && (X \in \mathrm{Obj}_{\mathbf{Set}}) \\ F(f) &= [\mathrm{inl}, \mathrm{inr} \circ f] && (f \in \mathrm{Hom}_{\mathbf{Set}}) \end{aligned}$$

- nat と $\alpha : F(\mathsf{nat}) \to \mathsf{nat}$ を次のように定義
  - $\alpha : F(\mathsf{nat}) \to \mathsf{nat}$ は F 始代数になる

```
Inductive nat : Type :=
| O : nat
| S : nat → nat
```

```
α x = match x with
  | inl t ⇒ O
  | inr n ⇒ S n
end.
```

F 始代数の例: nat(2)

- $(\!|\mathsf{f}|\!)$ の実装

$$
\begin{aligned}
(\!|[\mathsf{c},\ \mathsf{g}]|\!)\ \mathsf{x} \quad = \quad & \mathbf{match}\ \mathsf{x}\ \mathbf{with} \\
& \mid \mathsf{0} \quad \Rightarrow \quad \mathsf{c}\ \text{☺} \\
& \mid \mathsf{S}\ \mathsf{n} \quad \Rightarrow \quad \mathsf{g}\ ((\!|[\mathsf{c},\ \mathsf{g}]|\!)\ \mathsf{n}) \\
& \mathbf{end}
\end{aligned}
$$

- catamorphism の例
  - 和 $plus : \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat}$

$$plus = (\!|[\lambda\ \mathsf{t} \Rightarrow \lambda\ \mathsf{y} \Rightarrow \mathsf{y},\ \lambda\ \mathsf{f} \Rightarrow \lambda\ \mathsf{y} \Rightarrow \mathsf{S}\ (\mathsf{f}\ \mathsf{y})]|\!)$$

  - 積 $mult : \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat}$

$$mult = (\!|[\lambda\ \mathsf{t} \Rightarrow \lambda\ \mathsf{y} \Rightarrow \mathsf{0},\ \lambda\ \mathsf{f} \Rightarrow \lambda\ \mathsf{y} \Rightarrow plus\ \mathsf{y}\ (\mathsf{f}\ \mathsf{y})]|\!)$$

始代数と catamorphism の性質

- F 始代数 $\alpha : F(I) \to I$
- **補題** $(\!|\alpha|\!) = \mathrm{id}_I$
- **定理** $h \circ f = g \circ F(h)$ **ならば** $h \circ (\!|f|\!) = (\!|g|\!)$
- **定理** $\alpha \circ (\!|F\alpha|\!) = \mathrm{id}_I$, $(\!|F\alpha|\!) \circ \alpha = \mathrm{id}_{F(I)}$
  - すなわち，α は同型射

バナナ・スプリット則

定理 バナナ・スプリット則

$$\langle (\!|h|\!), (\!|k|\!) \rangle = (\!|\langle h \circ F(\mathsf{outl}), k \circ F(\mathsf{outr}) \rangle|\!)$$

- catamorphism と pair の入れ替え
- 応用例: $average = div \circ \langle sum, length \rangle$
  - 実は $(\!|[zeros, pluss]|\!)$ という 1 回の catamoprhism で計算できる

Outline

1. 背景
2. 先行研究の紹介
3. 準備: 一般の代数的データ型への拡張
4. 運算規則の実装
5. 今後の課題とまとめ

Coq による実装

- **運算の正当性を証明したい**
  - 定理証明支援系 Coq の利用
- **基本方針**
  - 教科書の記法になるべく近い形で
  - 動く Coq のプログラムに運算を適用したい

等式変形のためのタクティックライブラリ

- **定理証明支援系 Coq における不等式変形記法 [村田+, 2018] で作成したタクティックライブラリを用いる**
  - Tesson らのライブラリ [Tesson+, 2011] でも同じことはおそらく可能

```
(* induction step *)
- @ goal : (2 ^ (S m + 1) <= (S m)!).
  @ IHle : (2 ^ (m + 1) <= m !).
  Left
  = (2 ^ (S m + 1)).
  = (2 * 2 ^ (m + 1)).
  <= (2 * m !)      { by IHle }.
  <= ((S m) * m !)
              { because (2 <= (S m)) by omega }.
  = ((S m)!).
  = Right.
ed.
```

基本方針

- **Coq の型を圏の対象と見做す**
  - 型 A から型 B への Coq の関数は圏の射
  - 関手 F は $\mathrm{Type} \to \mathrm{Type}$ なる関数で関手の条件を満たすもの
- **順次 F 代数，F 始代数などの概念が定義できる**
  - **型クラス**を用いて，「$\alpha : \mathrm{F}(\mathrm{I}) \to \mathrm{I}$ が F 始代数」などの述語を定義
- **適宜`Notation`コマンドを用いて教科書に近い形で記述**

実装

- **具体的な実装を見ながら…**
  - `https://github.com/MurataKosuke/AoP_in_Coq`

Outline

1. 背景
2. 先行研究の紹介
3. 準備: 一般の代数的データ型への拡張
4. 運算規則の実装
5. 今後の課題とまとめ

今後の課題: 自動化タクティック

- このシステムでは
  1. F の定義
  2. F が関手であることの証明
  3. 代数的データ型 I の定義
  4. α の定義
  5. $\alpha : \mathrm{F}(\mathrm{I}) \to \mathrm{I}$ が F 始代数であることの証明

  を要求する
- しかし原理上 3. 以外は自動化できるはず？
  - Coq 上で代数的データ型 I から F を抽出するのは難しそう
  - しかし F さえあれば，2.・4.・5. を自動化するタクティックは作れるのではないか？

そのほかの今後の課題

- anamorphism, hylomorphism についての運算への拡張
- 関係上の運算への拡張

まとめ (再掲)

- **プログラム運算**: プログラムの意味を保存しながら代数的変形
  - 変形法則の**正当性の検証**をしたい
- **Coq 上でプログラム運算を行う試みを拡張を紹介**
  - 先行研究: 主にリストへの適用
    - Bird, *An Introduction to the Theory of Lists* の例を実装
  - **本研究: 一般の代数的データ型への適用**
    - Bird, De Moor: *Algebra of Programming* の例を (一部) 実装
- **適用例: バナナ・スプリット則等の検証**
- **課題: 自動化，適用例の証明，** ana や hylo・Allegory へ