後のための覚書：本スライドは，「第 22 回プログラミングおよびプログラミング言語ワークショップ（PPL 2020）」のカテゴリ 1 へ投稿されたワークショップ論文「Coq における hylomorphism を用いたプログラム運算の検証に向けて」の発表スライドですが，PPL 2020 は新型コロナウイルス感染症の流行のため現地開催が中止となったため，（2020/03/04 時点では）このスライドを用いた口頭発表は行われていません．

- 発表者は，カテゴリ 3 において，本発表と同タイトル・同発表者のポスタ発表を予定していましたが，本スライドと内容も同一のポスタ発表を予定していたため，カテゴリ 3 の発表を取り下げて，本スライドのみの提出といたしました．

Coq における hylomorphism を用いたプログラム運算の検証に向けて

2020/03/03

村田 康佑　　江本 健斗

九州工業大学

本研究の概要:
Coq で hylomorphism を定義したい

- 可能な限り **shallow embedding** をしたい
  - つまり Coq の中核言語（Gallina）で hylomorphism を実装したい
    （deep embedding のように，ホスト言語を一から Coq 上に設計するのは避けたい）
  - 期待される利点：Coq で容易に実行・運算が可能な hylomorphism の定義が得られる
- **しかし**：Coq では帰納データ型（μF）と余帰納データ型（νF）は別物．Hylomorphism の定義に必要な関数合成ができない
- **本研究**：hylomorphism の定義を修正して，Coq による実装と相性の良い “新たな hylomorphism” の定義を得る

Outline

背景：再帰図式（recursion scheme）

- データ型汎用な畳み込み（fold）・反畳み込み（unfold）等の計算パタン
  - **プログラム運算**，generic programming などの文脈で盛んに研究
  - **関手**をパラメタ化することによってデータ型汎用性を実現
    - 念の為：本研究でいう recursion scheme は HORS の recursion scheme とは別物（と考えて良い）
- **再帰図式の理論を Coq で検証し，ライブラリとして整備したい**
  - （データ型汎用な）certified programming への応用を期待

発表者による先行研究 [Murata+, APLAS 2019]

- Catamorphism, anamorphism などの基本的な再帰図式を形式化
  - 型クラスを用いて catamorphism や anamorphism の「公理」を表現し，その下で様々な運算規則等を証明
- 論文 [Uustalu+, 1999] に現れる運算規則をすべて形式化
  - Histomorphism/futumorphism の提案論文
  - 6 種の再帰図式に関する運算規則を与えている
- しかし **hylomorphism の形式化は行なっていない**
  - 本研究ではこれを行う

Recursion Schemes in Coq

畳み込み
(帰納的データ型を消費)

Catamorphism[Malcom, 1990]

$$f = (\!|\varphi|\!)_F \Leftrightarrow f = \varphi \circ F(f) \circ \mathbf{in}_F^{-1}$$

Paramorphism[Meertens, 1992]

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

Histomorphism[Uustalu+, 1999]

$$\{\!|\varphi|\!\}_F \Leftrightarrow f = \varphi \circ F([\![\langle f, \mathbf{in}_F^{-1}\rangle]\!]) \circ \mathbf{in}_F^{-1}$$

反畳み込み
(余帰納的データ型を構築)

Anamorphism[Fokkinga+, 1991]

$$f = [\![\psi]\!]_F \Leftrightarrow f = \mathbf{out}_F^{-1} \circ F(f) \circ \psi$$

Apomorphism[Vos, 1995]

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

Futumorphism[Uustalu+, 1999]

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

再畳み込み
中間データ構造を構築し，それを消費

Hylomorphism[Meijer+, 1991]

$$f = [\![\varphi, \psi]\!]_F \Leftrightarrow f = \varphi \circ F\, f \circ \psi$$

Recursion Schemes in Coq

畳み込み
(帰納的データ型を消費)

Catamorphism[Malcom, 1990]
$f = (\!|\varphi|\!)_F \Leftrightarrow f = \varphi \circ F(f) \circ \mathbf{in}_F^{-1}$

Paramorphism[Meertens, 1992]
$(\!|\varphi|\!)_F \Leftrightarrow f = \varphi \circ F(\langle f, \mathrm{id}_{\mu F}\rangle) \circ \mathbf{in}_F^{-1}$

Histomorphism[Uustalu+, 1999]
$\{\!|\varphi|\!\}_F \Leftrightarrow f = \varphi \circ F([\![\langle f, \mathbf{in}_F^{-1}\rangle]\!]) \circ \mathbf{in}_F^{-1}$

反畳み込み
(余帰納的データ型を構築)

Anamorphism[Fokkinga+, 1991]
$f = [\![\psi]\!]_F \Leftrightarrow f = \mathbf{out}_F^{-1} \circ F(f) \circ \psi$

Apomorphism[Vos, 1995]
$f = [\![\varphi]\!]_F \Leftrightarrow f = \mathbf{out}_F^{-1} \circ F([f, \mathrm{id}_{\nu F}]) \circ \psi$

Futumorphism[Uustalu+, 1999]
$f = [\![\psi]\!]_F \Leftrightarrow f = \mathbf{out}_F^{-1} \circ F((\!|[f, \mathbf{out}_F^{-1}]|\!)) \circ \psi$

再畳み込み
中間データ構造を構築し，それを消費

Hylomorphism[Meijer+, 1991]
$f = [\![\varphi, \psi]\!]_F \Leftrightarrow f = \varphi \circ F\, f \circ \psi$

先行研究
[Murata+, 2019]

Recursion Schemes in Coq

畳み込み
(帰納的データ型を消費)

Catamorphism[Malcom, 1990]
$f = (\!|\varphi|\!)_F \Leftrightarrow f = \varphi \circ F(f) \circ \mathbf{in}_F^{-1}$

Paramorphism[Meertens, 1992]
$(\!|\varphi|\!)_F \Leftrightarrow f = \varphi \circ F(\langle f, \mathrm{id}_{\mu F}\rangle) \circ \mathbf{in}_F^{-1}$

Histomorphism[Uustalu+, 1999]
$\{\!|\varphi|\!\}_F \Leftrightarrow f = \varphi \circ F([\![\langle f, \mathbf{in}_F^{-1}\rangle]\!]) \circ \mathbf{in}_F^{-1}$

反畳み込み
(余帰納的データ型を構築)

Anamorphism[Fokkinga+, 1991]
$f = [\![\psi]\!]_F \Leftrightarrow f = \mathbf{out}_F^{-1} \circ F(f) \circ \psi$

Apomorphism[Vos, 1995]
$f = [\![\varphi]\!]_F \Leftrightarrow f = \mathbf{out}_F^{-1} \circ F([f, \mathrm{id}_{\nu F}]) \circ \psi$

Futumorphism[Uustalu+, 1999]
$f = [\![\psi]\!]_F \Leftrightarrow f = \mathbf{out}_F^{-1} \circ F((\!|[f, \mathbf{out}_F^{-1}]|\!)) \circ \psi$

再畳み込み
中間データ構造を構築し，それを消費

Hylomorphism[Meijer+, 1991]
$f = [\![\varphi, \psi]\!]_F \Leftrightarrow f = \varphi \circ F\, f \circ \psi$

本研究

Outline

1. 背景
2. 復習：再帰図式
3. 再帰的余代数
4. 遅延モナド＋効果付き再帰図式
5. まとめ

Hylo-equation

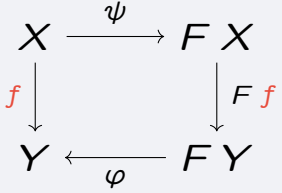
- 以下，C を圏，$F : C \to C$ を自己関手

Hylo-equation[Meijer+, 1991]

関数 $f : X \to Y$ の方程式

$$f = \varphi \circ F\, f \circ \psi$$

(ψ は F 余代数，φ は F 代数)



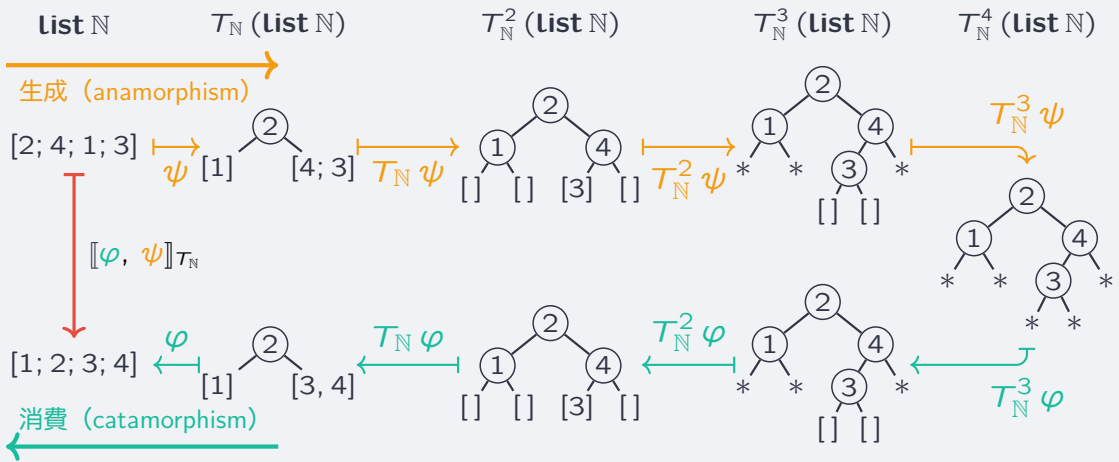
- 様々なプログラムが hylo-equation の解として記述できる

Hylomorphism の定義：大きく分けて二流儀 しかしいずれも Coq による実装には不向き

- 定義 A：最小不動点を利用 $[\![\varphi, \psi]\!]_F = \boldsymbol{\mu} f.\ \psi \circ F\ f \circ \varphi$
  - Coq の **fix** は制約が厳しく，上の定義は（そのままは）使えない
    - 入力が帰納データ型で，再帰呼び出し時に引数がある意味で「減少」していることを要求
    - 例えば上の定義では f の入力が帰納データ型であるかすら不明
- 定義 B：具体的に解を構成 $[\![\varphi, \psi]\!]_F = (\!|\varphi|\!)_F \circ [\!(\psi)\!]_F$
  - Coq では帰納データ型と余帰納データ型を区別するため，ana の出力は余帰納データ型，cata の入力は帰納データ型となり，上のような関数合成ができない

クイックソート *qsort* by hylomorphism

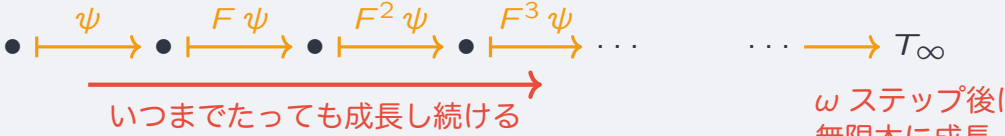
- 関手 $T_{\mathbb{N}}\,X = \mathbf{1} + X \times \mathbb{N} \times X$
  - （なお, 左の $\mathbf{1}$ は空リスト用）
- $qsort = [\![\psi, \varphi]\!]_{T_{\mathbb{N}}}$
  $= \boldsymbol{\mu}\, f.\ \varphi \circ T_{\mathbb{N}}\, f \circ \psi$
  - ψ で先頭要素をピボットにデータを大小で分割
  - $T_{\mathbb{N}}\, f$ で再帰
  - φ で再帰結果を統合



※ **inl** [ ] を $*$，**inr** $(t_0,\ n,\ t_1)$ を  で表している

Hylo は部分関数になり得る

- 生成（ana）が停止しない（＝無限木を生成する）場合
  - Coq でも余帰納データ型でこうした anamorphism は表現可
  - しかしそれでも **hylo は定義することができない**
    - 続く cata に余帰納的データ型を渡せないため（$\nu F \neq \mu F$）
- 部分関数になるかどうかすら難しい例：コラッツ列が停止するまでの長さ（擬似コードは論文参照）



ではどうするか？

- **本研究**: 以下の二つの方針を吟味
- 方針 1. 停止しない anamorphism の使用を制限する
  - 再帰的余代数 [Osius, 1974][Capretta+, 2006]
- 方針 2. 余帰納データ型を使って停止しない計算を表現
  - 遅延モナド [Capretta, 2005] + 効果付き再帰図式 [Fokkinga, 1994]

Outline

1. 背景
2. 復習：再帰図式
3. 再帰的余代数
4. 遅延モナド＋効果付き再帰図式
5. まとめ

再帰的余代数 [Osius, 1974][Capretta+, 2006]

- 余代数 $\psi : X \to F\,X$ が**再帰的**（recursive）
  $:\Longleftrightarrow \forall\ \varphi,\ \exists!\ f,\ f = \varphi \circ F\,f \circ \psi$
  - 要するに，任意の代数 φ に対して，hylomorphism $[\![\varphi, \psi]\!]$ が（唯一）作れることを保証
- ψ が再帰的であるとき **Set** でも hylo $[\![\varphi, \psi]\!]_F$ が定義可能
  - つまり Coq でも定義可能
  - 上の f を具体的に構成することになる
- この hylo は種々の運算規則を満たす

クイックソートの場合

- クイックソートの例の ψ は再帰的：以下を示すことになる

Proposition bstree_reccoalg :
$\forall \{Y\}\ (\varphi : T_{\mathbb{N}}\, Y \to Y).\ \exists (h : \mathbf{list}\,\mathbb{N} \to Y).\ \forall (f : \mathbf{list}\,\mathbb{N} \to Y).$
$h = f \leftrightarrow f = \varphi \circ T_{\mathbb{N}}\, f \circ \psi_{bstree}$.

  - h の具体的な構成

Function $h\ \{Y\}\ (\varphi : T_{\mathbb{N}}\, Y \to Y)\ (ln : \mathbf{list}\,\mathbb{N})\ \{measure\ length\ ln\} : Y :=$
match ln **with**
$\mid [\,] \Rightarrow \varphi\,(\mathbf{inl}\,())$
$\mid n :: ln' \Rightarrow \varphi \left(\mathbf{inr} \left(\begin{array}{l} h\,(filter\,(\boldsymbol{\lambda}\, x.\, x \leq_? n)\, ln'), \\ n, \\ h\,(filter\,(\boldsymbol{\lambda}\, x.\, negb\,(x \leq_? n))\, ln') \end{array} \right) \right)$
end.

議論：再帰的余代数 in Coq

- 余代数ごとに hylo を構成する必要があり，**面倒**
- 余代数が再帰的であることを示すのは結構**面倒**
  - 停止性が非自明な問題もある（コラッツ列の生成など）
- ただし，運算規則の証明は易しい？（と思われる）

Outline

1. 背景
2. 復習：再帰図式
3. 再帰的余代数
4. 遅延モナド＋効果付き再帰図式
5. まとめ

遅延モナド [Capretta, 2005]

- 次で**余帰納的**に定義される $\mathbf{D} : \mathbf{Set} \to \mathbf{Set}$

$$\frac{x : X}{\ulcorner x \urcorner : \mathbf{D}\, X}\ (\textsc{Now}) \qquad \frac{dx : \mathbf{D}\, X}{\triangleright dx : \mathbf{D}\, X}\ (\textsc{Later})$$

  - 例: $\triangleright\triangleright\triangleright\ulcorner 0 \urcorner$「3 ステップ後に値 0 が得られる」
  - 余帰納的定義なので $\triangleright$ が無限に続く元 $\triangleright^{\infty}$ も存在
    - 直観的には「何ステップたっても値が得られない」= undefined

Coq での遅延モナド：一般再帰の実現

- CoFixpoint を利用して，ungarded な一般再帰を実現
  - **返り値を遅延モナドに包むことで，CoFixpoint を使う**
  - Fixpoint としては ungarded だが CoFixpoint としては guarded
    - なお，$\mathit{illrec}\ n \approx \triangleright^{\infty}$　　（$\approx$ は弱双模倣）

```
(* NG *)
```
Fail Fixpoint $\mathit{illrec}\ (n : \mathbb{N}) : \mathbb{N} := \mathit{illrec}\ (\mathbf{S}\ n)$

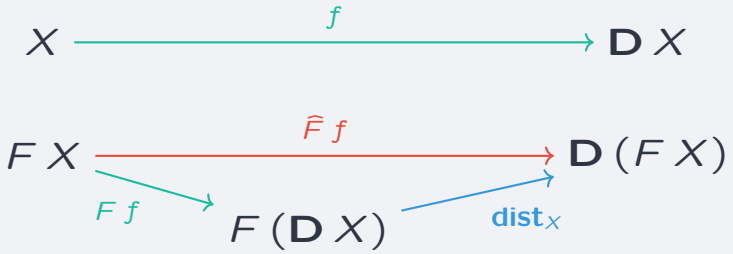
```
(* OK *)
```
CoFixpoint $\mathit{illrec}\ (n : \mathbb{N}) : \mathbf{D}\,\mathbb{N} := \triangleright(\mathit{illrec}\ (\mathbf{S}\ n))$

$\mathbf{Kle}_{\mathbf{D}}\ \mathbf{Set}$ の性質 [Capretta, 2005]

- $\mathbf{Kle}_{\mathbf{D}}\ \mathbf{Set}$：遅延モナドの Kleisli 圏
  - 一般再帰計算を表現する圏
  - 参考：$\mathbf{D}/\approx$ の Kleisli 圏は $\mathbf{Cppo}$ 豊穣圏 [Uustalu, 2017]
- 重要事実：$\mathbf{Kle}_{\mathbf{D}}\ \mathbf{Set}$ では**不動点コンビネータ** $\mathbf{dfix}$ が存在
  - $\forall\,(\Phi : (X \to \mathbf{D}\,X) \to X \to \mathbf{D}\,X),\ \mathbf{dfix}\,\Phi \approx \Phi\,(\mathbf{dfix}\,\Phi)$
  - これを使って hylomorphism を作れば良さそうだが…
    - そのためには関手 $F : \mathbf{Set} \to \mathbf{Set}$ の Kleisli 圏への持ち上げ $\widehat{F} : \mathbf{Kle}_{\mathbf{D}}\ \mathbf{Set} \to \mathbf{Kle}_{\mathbf{D}}\ \mathbf{Set}$ を作る必要

関手の Kleisli 持ち上げ

- $F : \mathbf{Set} \to \mathbf{Set}$ に対して $\widehat{F} : \mathbf{Kle}_{\mathbf{D}}\,\mathbf{Set} \to \mathbf{Kle}_{\mathbf{D}}\,\mathbf{Set}$
- 分配則 **dist** : $F \circ \mathbf{D} \Rightarrow \mathbf{D} \circ F$ を利用



提案：遅延 hylomorphism

$\psi\colon X \to F\,X$ と $\varphi\colon F\,Y \to Y$ に対して，

$$[\![\varphi, \psi]\!]_F^{\triangleright} = \mathbf{dfix}\,((\mathbf{return} \circ \psi) \diamond \widehat{F}\,f \diamond (\mathbf{return} \circ \psi))$$

- 効果付き再帰図式 [Pardo,2004] 等 を利用し，遅延 hylomorphism を定義
  - 効果を与えるモナドが遅延モナドの場合を考えれば良い
- 効果付き fusion 則などを満たす

遅延 hylomorphism の fusion 則

$$
\begin{aligned}
& h \circ \varphi \approx^{ext} (\mathbf{return}^{\mathbf{D}} \circ \varphi') \diamond \widehat{F}^{\mathbf{D}}\, h \\
\Longrightarrow\ & h \diamond [\![\varphi,\, \psi]\!]^{\triangleright}_{F} \approx^{ext} [\![\varphi',\, \psi]\!]^{\triangleright}_{F} \\
& (\mathbf{return}^{\mathbf{D}} \circ \psi) \diamond h \approx^{ext} \widehat{F}^{\mathbf{D}}\, h \circ \psi' \\
\Longrightarrow\ & [\![\varphi,\, \psi]\!]^{\triangleright}_{F} \diamond h \approx^{ext} [\![\varphi,\, \psi']\!]^{\triangleright}_{F}
\end{aligned}
$$

- ただし $\approx^{ext}$ は関数外延性による $\approx$ の関数への拡張

議論：遅延 hylo in Coq

- **汎用プログラミングの技法として簡便で強力**
  - ユーザは，F, φ, ψ の定義を与えるだけで遅延 hylomorphism $[\![\varphi, \psi]\!]_F^{\triangleright}$ を得ることができる
- **出力が遅延モナド型で包まれていることによる面倒やオーバーヘッドがあり得る**
  - 遅延 hylomorphism の出力は，余帰納的なデータ型 $\mathbf{D}$ で包まれている
  - Coq では余帰納的データ型そのままでは評価されない
    - 外から強制的にデストラクトするような仕組みが必要

Outline

1. 背景
2. 復習：再帰図式
3. 再帰的余代数
4. 遅延モナド＋効果付き再帰図式
5. まとめ

Hylomorphism in Coq

Coq で hylomorphism を扱うための 2 手法を検討・比較

	再帰的余代数	遅延 hylomorphism
長所	（運算規則の理論的取り扱いやすさ？）	ユーザのプログラミングのしやすさ
短所	再帰的であることの証明が面倒・プログラミングのしにくさ	出力が余帰納的データ型で汚れて扱いにくい

今後の課題：各アプローチでの各種運算規則の形式化