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

村田 康佑 [1], 江本 健斗 [2]

[1] 九州工業大学大学院情報工学府
murata@pl.ai.kyutech.ac.jp

[2] 九州工業大学大学院情報工学研究院
emoto@ai.kyutech.ac.jp

概 要　Hylomorphism を用いたプログラム運算は，適用範囲が広く有用であり，その正しさを検証するためのシステムが望まれる．定理証明支援系 Coq を用いて運算を検証するシステムはいくつか知られているが，hylomorphism およびその運算規則を使いやすい形で形式化することは困難であり，著者の知る限り未だ実現されていない．その困難さは，Coq においては帰納的データ型と余帰納的データ型が区別される一方，hylomorphism はこれらが同一視されるような条件の下で定義されるというギャップにある．本研究では，このギャップを回避しつつ形式化するための技術として，遅延モナドおよび再帰的余代数の利用を検討し，例を用いてこれらの長所や短所を議論する．

1　はじめに

畳み込み (folding) や反畳み込み (unfolding) などの計算パタンは，関数プログラミングにおいて重要である．再帰図式 (recursion scheme) は，これらの計算パタンをデータ型汎用 (datatype-generic) な定義へと抽象化したものであり，プログラム運算や，汎用プログラミングの文脈で盛んに研究されている [2, 12, 16, 27, 30]．本研究の究極の目標は，定理証明支援系 Coq [25] でこれらの成果の形式化および検証を行い，誤りのないプログラム運算を支援することである．

Murata ら [19] は，Tesson らによるプログラム運算のための Coq ライブラリ [24] を拡張し，基本的な再帰図式の形式的定義および，それらの運算規則の証明を与えた．これにより，Coq で，データ型汎用な畳み込みや反畳み込みに関わるプログラム運算を，手軽に扱うことができるようになった．一方，Meijer らによる hylomorphism [16] や，Kabanov らによる dynamorphism [12] をはじめとする再畳み込み (refolding) の再帰図式については，その有用性が知られているにも関わらず，未だ「使いやすい」形式的定義は得られていない．

本論文のひとまずの目標は，**Coq による運算に適した実行可能な hylomorphism の定義を得る**ことである．ここでいう「実行可能」とは，Gallina[1] インタプリタを利用してプログラムを実行することができる状況を指している．実行可能な hylomorphism の定義の下では，ユーザはそれを用いて実行可能なプログラムを定義し，安全に運算を行い，得られたプログラムを実行したり，他の言語に抽出して利用したりすることが可能になると考えられる．実行可能な定義を得るために，Murata ら [19] と同様に，できるだけ shallow embedding に近い定義をしたい．Deep embedding を用いたアプローチも考えられるが，新たなホスト言語を設計したり，運算により得られたプログラムを利用するための評価器を実装したりすることが必要である．一方 shallow embedding であれば，ホスト言語として Gallina をそのまま利用することができる．もとより，既存の言語を利用できるとい

[1] Gallina は，Coq の中核を成す関数型言語である．

うことは，その言語についての既存の資産を利用できる可能性も視野に入ってくることになり，実用上望ましい．

では，hylomorphism が容易に Coq へ shallow embedding できるかというと，事態はそう単純ではない．というのも，一見すると矛盾しているように見える以下の二つの状況があるからである．

- **Coq で動くプログラムは全域関数である**．Coq の中核言語である Gallina は強正規化性が成り立っていて，停止するプログラムしか書くことができない．Gallina の関数 $f\colon X \to Y$ は，全域関数の圏 **Set** の射 $f\colon X \to Y$ に対応していると見做すのが自然である．
- **Hylomorphism は部分関数となり得る**．そもそも，hylomorphism は，代数コンパクト（algebrically compact）[10] な関手の下でしか定義することができない．ある関手が代数コンパクトとは，端的に言えば始代数と終余代数が一致することであるが，この条件は「めったなことでは満たされない」とも言われている [32]．例えば，Murata ら [19] が形式化のベースとしている圏 **Set** の自己関手は，通常代数コンパクトではない．この事実は，hylomorphism で定義された計算は，本質的に停止しない可能性を孕んでいることと深く関係している．

Meijer らの hylomorphism は，基点つき完備半順序集合（cppo）と正格連続関数の圏 $\mathbf{Cppo}_{\perp}$ の上で定義されており，$\mathbf{Cppo}_{\perp}$ の自己関手は代数コンパクトであることが知られている [10]．この事実から考えれば，Coq に cppo を実装すれば良いように思えるのだが，そもそも cppo は表示的意味論に用いられる道具であり，動作可能なプログラムを得るという目的とは相性が悪いと考えられる．

全域関数しか扱えない Coq で，本質的に部分関数になり得る hylomorphism をうまく扱う方法はないものだろうか？　本論文では，この問いに対する以下の二つのアプローチを検討する．

- 第一に，hylomorphism に用いる余代数を，Osius による再帰的余代数（recursive coalgebra）[20] とよばれる性質の良いものに制限するアプローチを検討する．このアプローチは，大雑把に言えば，そもそも部分関数になるような「行儀の悪い」hylomorphism は扱えないように制限するという素直なものであり，理論的にも **Set** で hylomorphism を扱うために広く用いられている [4, 11]．しかし，Coq でも容易に扱えるかどうかは定かではないため，これを確かめる．
- 第二に，Capretta による遅延モナド [3] を用いるアプローチを検討する．遅延モナドは，Coq のような強正規性が成り立つ型システム上で一般再帰を実現するための手法であり，Coq でもプログラミングに用いられる [6]．**Set** 上の遅延モナドが成す Kleisli 圏は，一種の部分関数の圏となるため，hylomorphism と相性が良さそうである．それどころか，遅延モナド（を弱双模倣で割ったもの）が成す Kleisli 圏は，$\mathbf{Cppo}_{\perp}$ 豊穣圏になる [26] ことが知られており，$\mathbf{Cppo}_{\perp}$ に近い状況の下で hylomorphism を定義することができそうだという希望がある．これと Fokkinga [8] や Pardo [21, 22] らによる Kleisli 圏上での再帰図式の研究と組み合わせることで，Coq 上でうまく hylomorphism を扱えるのではないか．本論文ではこのアイデアを検討する．

ところで，hylomorphism の定義のバリエーションの中には，関係の圏 **Rel** を用いたもの [1, 2] も知られており，これを形式化することも考えられる．関係の圏は定理証明支援系との相性が良いと考えられ，実際に関係の圏をベースとして運算定理を証明した研究も存在する [5, 18]．しかし，関係を Coq で素直に形式化すると $A \to A \to \mathsf{Prop}$ のような型となって，Gallina 処理系で動作させることのできるプログラムにはならないので，本論文の目的には適さないと考えられる．

本論文の貢献　本論文の貢献は，以下の 2 点である．

1. 上述の一つ目のアプローチ，すなわち再帰的余代数を用いたアプローチに検討を加えた．この方法で記述できるものは極めて再利用性の高いプログラムが得られる一方，記述しにくい例があることを指摘した．この詳細は，第 3 節で述べる．

2. Coq で hylomorphism を定義する第二の方法として，遅延モナドを用いたアプローチを検討した．Fokkinga [8] や Pardo [21, 22] による効果付き再帰図式と組み合わせて，遅延モナドを用いた遅延 hylomorphism を提案し，この定義の下で融合則などの運算法則が従うことを確かめた．また，遅延 hylomorphism の Coq での実装を行い，複数の例について動作するプログラムが得られることを確かめた．これらについての詳細は，第 4 節で述べる．

特に 2 が主要な貢献であり，本論文で提案する遅延 hylomorphism は，Coq 上で shallow embedding に近い形で hylomorphism が扱う方法として有力なものであると考えている．しかし，これらの性質についての Coq による証明は未完成であり，今後の課題となる．

記法　本論文では，プログラムや命題の提示に Coq 風の擬似コードを用いるが，可読性の向上のために，ところどころ，数学や，Haskell 等他のプログラミング言語において標準的な記法も混ぜて用いる．ただし，Coq で読み込み可能なコードに修正することが容易な範囲に留めることにする．

構成　本論文のこれ以降の構成は，次のようになる．まず，第 2 節では，準備として再帰図式の用語を導入する．また hylomorphism によるプログラミングの例をいくつか述べる．第 3 節では，Coq で hylomorphism を扱うための第一のアプローチである再帰的余代数を用いたアプローチについて検討する．第 4 節では，第二のアプローチである，遅延モナドと効果付き再帰図式を組み合わせたアプローチを提案し，検討する．第 5 節では，本研究のまとめおよび関連研究，今後の課題について述べる．

2　準備：再帰図式

本節では，再帰図式の用語を導入する．以下では，集合とその間の全域関数が成す圏を **Set** で表し，最小元を持つ ω-cpo とその間の連続写像がなす圏を **Cppo**，さらにそれを正格な写像に限定した圏を $\mathbf{Cppo}_{\perp}$ と書く．

2.1　F 代数，F 余代数

記法の確認のため，F 代数や F 余代数に関わる基本事項を確かめる．より詳細な解説は，Vene の博士論文 [28] や Bird による教科書 [2] などを参照せよ．

以下，$\boldsymbol{C}$ を圏とし，$F\colon \boldsymbol{C} \to \boldsymbol{C}$ を自己関手とする．対象 X および射 $\varphi\colon F\,X \to X$ の組 (X, φ) を F 代数といい，対象 Y および射 $\psi\colon Y \to F\,Y$ の組 (Y, ψ) を F 余代数という．始 F 代数を $(\mu F, \mathsf{in}_F)$，終 F 余代数を $(\nu F, \mathsf{out}_F)$ と書くことにする．混乱のおそれがない場合には，F 代数 (X, φ) を単に φ，F 余代数 (Y, ψ) を単に ψ と見做すこともある．Lambeck の補題として，$\mathsf{in}_F\colon F\,\mu F \to \mu F$ および $\mathsf{out}_F\colon \nu F \to F\,\nu F$ には，それぞれ逆射 ${\mathsf{in}_F}^{-1}$ および ${\mathsf{out}_F}^{-1}$ が存在することが知られている．

μF には帰納的データ型，νF には余帰納的データ型が対応することが知られている．例えば，**Set** において，以下のような対応が知られている．以下，$\mathbf{1}$ は終対象であり，**Set** では単元集合 $\{()\}$ に対応する．

- 自然数関手 $N\,X = \mathbf{1} + X$ に対して，μN は自然数型 $\mathbb{N}$ に対応する．
- リスト関手 $L_A\,X = \mathbf{1} + A \times X$ に対して，μL_A は A 上の有限リスト型 $\mathbf{list}\,A$ に対応する．
- 木関手 $T_A\,X = \mathbf{1} + X \times A \times X$ に対して，μT_A は A 上の有限二分木型 $\mathbf{tree}\,A$ に対応する．

また，νL_A は（無限リストも許した）リスト型，νT_A は（無限木も許した）二分木型がそれぞれ対応する．

F 代数 $\varphi\colon F\,Y \to Y$ に対する catamorphism $(\!|\varphi|\!)_F\colon \mu F \to Y$ を

$$f = (\!|\varphi|\!)_F \iff \varphi \circ F\,f = f \circ \mathsf{in}_F \qquad \text{(Cata-Charn)}$$

で定義する．また，F 余代数 $\psi\colon X \to F\,X$ に対する anamorphism $[\![\psi]\!]_F\colon X \to \nu F$

$$f = [\![\varphi]\!]_F \iff F\,f \circ \psi = \mathsf{out}_F \circ f \qquad \text{(Ana-Charn)}$$

で定義する．本論文では，catamorphism および anamorphism が存在するような F のみを考える．例えば **Set** や $\mathbf{Cppo}_\perp$ 上の正則関手（regular functor）F に対しては，恒に $(\!|\varphi|\!)_F$ および $[\![\psi]\!]_F$ が存在することが知られている．

Coq による形式化　Murata ら [19] は，関手や始代数，終余代数といった概念を，Coq を用いて形式化した．型クラスを用いることで，素直な形式化を得ることができる．例えば，関手 F は，$F\colon \mathsf{Type} \to \mathsf{Type}$ と見做して，次のように定義できる．

Class Functor $(F : \mathsf{Type} \to \mathsf{Type}) :=$
{
　fmap : $\forall\ \{X\ Y : \mathsf{Type}\}\ (f : X \to Y),\ F\,X \to F\,Y$;
　fmap_id : $\forall\ \{X : \mathsf{Type}\},\ \mathsf{fmap}\ (@\mathsf{id}\,X) = @\mathsf{id}\,(F\,X)$;
　fmap_compose : $\forall\ \{X\ Y\ Z : \mathsf{Type}\}\ (f : X \to Y)\ (g : Y \to Z),\ \mathsf{fmap}\,(g \circ f) = \mathsf{fmap}\,g \circ \mathsf{fmap}\,f$
}.

同様に，始代数のための型クラスや，終余代数のための型クラスも容易に定義することができる．

Hylomorphism　自己関手 F および，F 余代数 $\psi\colon X \to F\,X$，F 代数 $\varphi\colon F\,Y \to Y$ に対して，$f\colon X \to Y$ の再帰方程式

$$f = \varphi \circ F\,f \circ \psi \qquad \text{(Hylo-Equation)}$$

を，**hylo 方程式**（hylo-equation）という．この Hylo-Equation は，分割統治に基づくプログラムの再帰的定義を与える上で有用である．というのも，ψ でデータを分割し，$F\,f$ による再帰で得られた結果を，φ で統治するという定義を与えられるためである．

Set では Hylo-Equation は解を持つとは限らないが，$\mathbf{Cppo}_\perp$ では恒に解を持つ．より一般に，F が代数コンパクト（algebraically compact）[10] である場合，すなわち $\mu F = \nu F$ かつ $\mathsf{in}_F = \mathsf{out}_F{}^{-1}$ とできる場合には，Hylo-Equation は恒に解を持つ：$f = (\!|\varphi|\!)_F \circ [\![\psi]\!]_F$ とすれば良い．

$\mathbf{Cppo}_\perp$ をはじめとする $\mathrm{O}_\perp$ 圏の正則関手は，代数コンパクトであることが知られている．$\mathrm{O}_\perp$ 圏のもとで，hylomorphism $[\![\varphi, \psi]\!]_F\colon X \to Y$ を，Hylo-Equation の最小解と定義する．すなわち，

$$([\![\varphi, \psi]\!]_F = \varphi \circ F\,[\![\varphi, \psi]\!]_F \circ \psi) \wedge (\forall\ f,\ f = \varphi \circ F\,f \circ \psi \implies [\![\varphi, \psi]\!]_F \sqsubseteq f). \qquad \text{(Hylo-Charn)}$$

この定義の下で，以下が成り立っている．

$$[\![\varphi, \psi]\!]_F = (\!|\varphi|\!)_F \circ [\![\psi]\!]_F \qquad \text{(Ana-Cata-Hylo)}$$

逆に，hylomorphism を用いて，catamorphism および anamorphism を次のように特徴付けることも可能である．

$$[\![\psi]\!]_F = [\![\mathsf{in}_F, \psi]\!] \qquad \text{(Hylo-Ana)}$$

$$(\!|\varphi|\!)_F = [\![\varphi, \mathsf{in}_F^{-1}]\!] \qquad \text{(Hylo-Cata)}$$

Hylomorphism には多くの運算法則が知られている．例えば以下の融合則が成り立っている [9].

$$h \circ \varphi = \varphi' \circ F\,h \implies h \circ [\![\varphi, \psi]\!]_F = [\![\varphi', \psi]\!]_F \qquad \text{(Hylo-FuxionC)}$$

$$\psi \circ h = F\,h \circ \psi' \implies [\![\varphi, \psi]\!]_F \circ h = [\![\varphi, \psi']\!]_F \qquad \text{(Hylo-FuxionA)}$$

2.2　Hylomorphism によるプログラミング

以下では，hylomorphism を用いたプログラムの具体例として，コラッツ列およびクイックソートの二つの話題を取り上げる．特に，後者は典型的な分割統治アルゴリズムである．

コラッツ列　まず, 実用性には欠けるが理論的には興味深い例として, コラッツ列 (Collatz sequence) を取り上げる．以下，$(\%) : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ を，第一引数を第二引数で割った場合の余りを返す関数 (Coq の標準ライブラリにおける `mod` に相当) とし，$(=_?) : \mathbb{N} \to \mathbb{N} \to \mathbf{bool}$ を，第一引数の自然数と第二引数の自然数が等しいか否かを返す関数 (Coq の標準ライブラリにおける`=?`に相当) とする．

さて，関数 *col_succ* を，

$$\mathbf{Definition}\ \mathit{col\_succ}\,(i : \mathbb{N}) : \mathbb{N} := \mathbf{if}\ (i \,\%\, 2 =_?\, 0)\ \mathbf{then}\ i/2\ \mathbf{else}\ 3 * i + 1$$

で定める．正の整数 n に対して，*col_succ* の適用を繰り返してできる列

$$n,\ \mathit{col\_succ}\, n,\ \mathit{col\_succ}^2\, n,\ \mathit{col\_succ}^3\, n, \ldots$$

を，n を始とするコラッツ列という．例えば，3 を始とするコラッツ列は,

$$3,\ 10,\ 5,\ 16,\ 8,\ 4,\ 2,\ 1,\ 4,\ 2,\ 1,\ 4,\ 2,\ 1, \ldots \qquad (*)$$

である．この例を見てもわかるように，コラッツ列が一度 1 に至ると，そのあとはずっと 4, 2, 1,... の繰り返しになることから，コラッツ列の定義として 1 が出たらそこで打ち切る定義を採用する場合もある．ところで,「任意の n に対して，n を始とするコラッツ列に 1 が含まれるか？」という問題は有名な未解決問題であり，その肯定予想はコラッツ予想として知られている．

さて, 本題に戻って,「i を始とするコラッツ列が最初に j になるのはいつか」という問題を解くための hylomorphism を定義したい．コラッツ列を生成するための余代数 $\psi_{\mathit{colgen}} : \mathbb{N} \to L_{\mathbb{N}}\,\mathbb{N}$ および，リストから要素 k を見つけてその番号 (index) を返すための代数 $\varphi_{\mathit{find}}\, k \colon L_N\,(\mathbf{maybe}\,\mathbb{N}) \to \mathbf{maybe}\,\mathbb{N}$ を，次のように定義する：

$$\begin{array}{l}
\mathbf{Definition}\ \psi_{\mathit{colgen}}\ (n : \mathbb{N}) : L_{\mathbb{N}}\,\mathbb{N} := \\
\quad \mathbf{if}\ n =_?\ \mathsf{O}\ \mathbf{then}\ \mathsf{inl}\,()\ \mathbf{else}\ \big(\mathbf{if}\ n =_?\ 1\ \mathbf{then}\ \mathsf{inr}\,(n,\ \mathsf{O})\ \mathbf{else}\ \mathsf{inr}\,(n,\ \mathit{col_succ}\, n)\big). \\
\mathbf{Definition}\ \varphi_{\mathit{find}}\ (k : \mathbb{N})\ (\mathit{lmn} : L_{\mathbb{N}}\,(\mathbf{maybe}\,\mathbb{N})) : \mathbf{maybe}\,\mathbb{N} := \\
\quad \mathbf{match}\ \mathit{lmn}\ \mathbf{with} \\
\quad \mid \mathsf{inl}\,() \qquad\quad \Rightarrow \mathsf{nothing} \\
\quad \mid \mathsf{inr}\,(n,\ \mathit{mn}) \Rightarrow \mathbf{if}\ n =_?\ k\ \mathbf{then}\ \mathsf{just}\,\mathsf{O}\ \mathbf{else}\ (\mathit{mn} \gg\!=^{\mathbf{maybe}} (\boldsymbol{\lambda}\,(n' : \mathbb{N}).\,(\mathsf{just}\,(\mathsf{S}\,n')))) \\
\quad \mathbf{end}.
\end{array}$$

ただし，**maybe** は Haskell 言語で一般的な maybe モナドを表し，nothing および just は maybe モナドのコンストラクタ，$\gg\!=^{\mathbf{maybe}}$ は maybe モナドの bind 演算子である．

このとき，$\mathit{col\_len} = [\![\varphi_{\mathit{find}}\, 1,\ \psi_{\mathit{colgen}}]\!]_F : \mathbb{N} \to \mathbf{maybe}\,\mathbb{N}$ は，入力 i を始とするコラッツ列が，1 を出力するまでの長さを調べる関数である．例えば，3 を始とするコラッツ列 $(*)$ は，先頭の 3 を 0 番目とすると，1 が初めて現れるのは 7 番目なので，$\mathit{col\_len}\, 3 = 7$ となる．なお，この関数 *col_len* が全域関数になるかどうかは，明らかにコラッツ予想と同値である．もし仮にコラッツ予想が肯定的に成り立っている，つまり *col_len* が全域関数だったとしても，*col_len* を Coq で記述することは，コラッツ予想の肯定を証明することと同程度の難しさの作業ということになる．

クイックソート　もう少し実用的な例として, クイックソートを考えてみよう．余代数 $\psi_{\mathit{bstree}} : \mathbf{list}\,\mathbb{N} \to T_{\mathbb{N}}\,(\mathbf{list}\,\mathbb{N})$ を

$$\begin{array}{l}
\mathbf{Definition}\ \psi_{\mathit{bstree}}\ (\mathit{ln} : \mathbf{list}\,\mathbb{N}) \colon T_{\mathbb{N}}\,(\mathbf{list}\,\mathbb{N}) := \\
\quad \mathbf{match}\ \mathit{ln}\ \mathbf{with} \\
\quad \mid [\,] \qquad\quad \Rightarrow \mathsf{inl}\,[\,] \\
\quad \mid n :: \mathit{ln}' \Rightarrow \mathsf{inr}\,([n' \leftarrow \mathit{ln}' \mid n' \leq n],\ n,\ [n' \leftarrow \mathit{ln}' \mid n' > n]) \\
\quad \mathbf{end}.
\end{array}$$

で定め，代数 $\varphi_{intrav} : T_{\mathbb{N}}\,(\mathbf{list}\,\mathbb{N}) \to \mathbf{list}\,\mathbb{N}$ を

$$
\begin{array}{l}
\mathbf{Definition}\ \varphi_{intrav}\,(tln : T_{\mathbb{N}}\,(\mathbf{list}\,\mathbb{N})) : \mathbf{list}\,\mathbb{N} := \\
\quad \mathbf{match}\ tln\ \mathbf{with} \\
\quad \mid \mathsf{inl}\,() \qquad\qquad\ \Rightarrow [\,] \\
\quad \mid \mathsf{inr}\,(ln'_0,\, n,\, ln'_1) \Rightarrow ln'_0 +\!\!+ [n] +\!\!+ ln'_1 \\
\quad \mathbf{end}.
\end{array}
$$

で定めると，$[\![\psi_{bstree},\, \varphi_{intrav}]\!]_{T_{\mathbb{N}}} : \mathbf{list}\,\mathbb{N} \to \mathbf{list}\,\mathbb{N}$ は，入力のリストをクイックソートする関数になる．

ところで，$[\![\psi_{bstree}]\!]_{T_{\mathbb{N}}} : \mathbf{list}\,\mathbb{N} \to \mathbf{tree}\,\mathbb{N}$ は入力のリストから二分探索木を構築する関数であり，$(\!|\varphi_{intrav}|\!)_{T_{\mathbb{N}}} : \mathbf{tree}\,\mathbb{N} \to \mathbf{list}\,\mathbb{N}$ は入力の木を走査して中間順で要素を出力する関数である．$qsort = (\!|\varphi_{intrav}|\!) \circ [\![\psi_{bstree}]\!]_{T_{\mathbb{N}}}$ であることを考えれば，$qsort$ とは，二分探索木を構築してから，中間順で要素を出力するアルゴリズムであるとも言える．

3 Coq における hylomorphism (1): 再帰的余代数

本節では，Coq で hylomorphism を定義するためのアプローチとして，再帰的余代数 [4, 11] を用いたものについて述べ，検討する．本節では，圏 **Set** 上で議論する．

余代数 $\psi : X \to F\,X$ が再帰的（recursive）であるとは，任意の代数 $\varphi : F\,Y \to Y$ に対して，

$$f = \varphi \circ F\,f \circ \psi \qquad (**)$$

を満たす f が唯一存在することである．$(**)$ は正に Hylo-Equation そのものであるから，余代数 ψ が再帰的であるとは，**Set** でも hylomorphism を考えることができるということに他ならない．この **Set** 上の hylomorhism についても，$\mathbf{Cppo}_{\perp}$ 上の hylomorphism と同様に，hylo 融合則などの多くの法則がそのまま成り立つ．

3.1 再帰的余代数の例と Coq による実装例

クイックソートの例で用いた $\psi_{bstree} : \mathbf{list}\,\mathbb{N} \to T_{\mathbb{N}}\,(\mathbf{list}\,\mathbb{N})$ は再帰的余代数である．これを示すには，$(**)$ を満たす f が φ に対して唯一存在することを言えば良い．Coq では，次のことを示すことになる：

$$
\begin{array}{l}
\mathbf{Proposition}\ \mathsf{bstree\_reccoalg} : \\
\quad \forall\ \{Y : \mathsf{Type}\}\ (\varphi : T_{\mathbb{N}}\,Y \to Y),\ \exists\ (h : \mathbf{list}\,\mathbb{N} \to Y),\ \forall\ (f : \mathbf{list}\,\mathbb{N} \to Y), \\
\qquad h = f \leftrightarrow f = \varphi \circ T_{\mathbb{N}}\,f \circ \psi_{bstree}.
\end{array}
$$

上の h の存在を示すため，あらかじめ次のように具体的に h を構成しておくと良い．ただし，$\leq_?$ は自然数の大小を bool 値で返す関数，$negb$ は bool 値を反転させる関数である．

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

この **Function** コマンドは，`Coq.funind.Recdef` ライブラリに定義されているコマンドで，整礎関係上の再帰を記述するための仕組みである．上の h の場合であれば，$\{\mathit{measure}\ \mathit{length}\ ln\}$ の部分で，再帰呼び出しごとに $\mathit{length}\ ln$ の値が減少していくことを主張しており，この後ユーザは Coq に $\mathit{length}\ ln$ が本当に減少していることの証明を求められることになる．こうして無事に h が定義できれば，あとは bstree_recoalg が成り立つことを示せばよい．この bstree_recoalg の証明は難しくはないが，通常のリストに関する帰納法が使うことができず，やや面倒な証明となる．

ひとたび具体的に h を与えてしまえば，$qsort$ は $h\,\varphi_{intrav} : \mathbf{list}\,\mathbb{N} \to \mathbf{list}\,\mathbb{N}$ として得ることができる．

3.2 考察：再帰的余代数を Coq で扱うことについて

再帰的余代数によるアプローチは何ら新しいものではないが，hylomorphism によるプログラミングおよびその運算を Coq 上で支援する手法として評価しなおすとき，以下の 2 点が指摘できる．

- 停止性の証明が求められる．クイックソートの例では，h を構成するときに，h が整礎関係上の再帰になっていることの証明，すなわち h の停止性の証明を求められた．このように，具体的な余代数が再帰的余代数であることを示すには，hylomorphism の停止性（全域性）を示す必要がある．それゆえ，コラッツ列のように，停止性の非自明な hylomorphism を扱うのは困難である（もし扱うことができるとすれば，未解決問題を解決して，コラッツ列が 1 に至ることの証明を遂行した場合のみである）．
- 余代数ごとに証明が求められる．このアプローチでは，具体的な余代数を考えるごとに再帰的であることの証明を求められることになり，面倒である．さらに，その証明の中では，（Coq のような構成的論理の上では）具体的に hylo 方程式の解を構成することになるため，「汎用的な定義から手間をかけずに具体的なプログラムを導出する」ということができず，汎用プログラミングの技法としてはほとんど役に立たない．もちろん，hylomorphism について成り立つ運算法則を適用することで，高速なプログラムを導出することは可能であると考えられるため，安全な運算を支援するシステムを構築するためには有用な手法であると考えられる．

4 Coq における hylomorphism (2): 遅延 hylomorphism

本節では，Coq で hylomorphism を定義するための次なるアプローチである，遅延モナドを用いたアプローチについて述べる．そのために，まず効果付き再帰図式および遅延モナドについて紹介する．その後，本論文の主提案である遅延 hylomorphism を定義し，その性質について議論する．

4.1 準備: 効果付き再帰図式

モナド（monad）は，関数プログラムにおける計算効果の抽象化として重要である [17]．モナドの定義は同値なものが複数知られているが，以下では join と unit を用いた (M, μ^M, η^M) によるもの，return と bind を用いた $(M, \mathsf{return}^M, (\gg\!=^M))$ によるもの，Kleisli 星を用いた $(M, \mathsf{return}^M, -^*)$ によるものなどの，すべてを用いる．混乱のおそれがない場合には，単に関手 M をモナドと見做すこともある．また，$f\colon X \to M\,Y$ および $g\colon Y \to M\,Z$ の Kleisli 合成を $g \diamond^M f\colon X \to M\,Z$ と書く．なお，ここまで導入した記法のすべてについて，M が明らかな場合には，右上の添字の M を省略することがある．なお，$g \diamond f = \mu_X \circ M\,g \circ f = \boldsymbol{\lambda}\,x.\,(f\,x) \gg\!= g$ である．以下では，M の Kleisli 圏を $\mathbf{Kle}_M\,\boldsymbol{C}$ で書く．混乱を避けるために，$f \in \mathrm{Hom}_{\mathbf{Kle}_M\,\boldsymbol{C}}(X, Y)$ の射を書くときには $f\colon X \to Y$ とは書かず，必ずもとの圏 $\boldsymbol{C}$ の射として $f\colon X \to M\,Y$ と書くことにする．

計算効果を伴う再帰図式として，Fokkinga [8] による効果付き catamorphism や，Pardo [22] による効果付き anamorphism，効果付き hylomorphism が知られている[2]．これらの基本的なアイデアは，圏 $\boldsymbol{C}$ 上の自己関手 $F\colon \boldsymbol{C} \to \boldsymbol{C}$ を Kleisli 圏に持ち上げた関手 $\widehat{F}^M\colon \mathbf{Kle}_M\,\boldsymbol{C} \to \mathbf{Kle}_M\,\boldsymbol{C}$ を定義し，この $\widehat{F}^M$ を用いて再帰図式を考えることである．

さて，関手 $\widehat{F}^M$ は，分配則（distributive law）といわれる自然変換 $\mathsf{dist}^{M,F}\colon F \circ M \Rightarrow M \circ F$ を用いて，

$$\begin{aligned} \widehat{F}^M\,X &= M\,(F\,X), \\ \widehat{F}^M\,(f\colon X \to M\,Y) &= \mathsf{dist}^{M,F}_Y \circ F\,f\colon F\,X \to M\,(F\,Y) \end{aligned}$$

[2]「効果付き catamorphism」というような名称は本論文独自の呼称であり，元論文では monadic catamorphism などと呼んでいる．

と定められる．ただし，分配則は

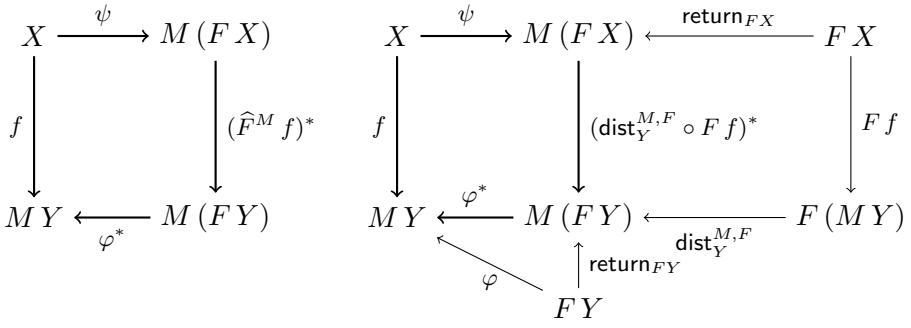
$$\widehat{F}^M\,\mathsf{return}^M_X = \mathsf{return}^M_{FX}, \qquad \widehat{F}^T\,(g \diamond f) = \widehat{F}^M\,g \diamond \widehat{F}^M\,f \tag{1}$$

の二つを満たすように定めるとする．このおかげで，$\widehat{F}^{\,M}$ は，$\mathbf{Kle}_M\,\boldsymbol{C}$ における関手則を満たすものとなる（$\mathsf{return}^M_X \in \mathrm{Hom}_{\boldsymbol{C}}(X,\,M\,X)$ は，$X \in \mathrm{Obj}_{\mathbf{Kle}_M\,\boldsymbol{C}}$ の単位射であることに注意）．M が強モナド [17] である場合には，(1) が満たされるような $\mathsf{dist}^{M,F}$ は F に関する帰納法で構成することができる．詳しくは [22] を参照せよ．

効果付き hylo 方程式　モナド M により関手 $F\colon \boldsymbol{C} \to \boldsymbol{C}$ 持ち上げた関手 $\widehat{F}^M$ を用いた，以下の効果付き hylo 方程式を考える：

$$f = \varphi \diamond \widehat{F}^M\,f \diamond \psi \tag{MHylo-Equation}$$

なお，このときの各射の型は，次の二つの図式ようになっている．なお，二つの図式はいずれも $\boldsymbol{C}$ の図式であり，右側の図式は，左側の図式の $\widehat{F}^M$ および Kleisli 星を展開したものになっている．



Pardo は，$\boldsymbol{C} = \mathbf{Cppo}$ の場合を考え，MHylo-Equation の最小解を効果付き hylomorphism $[\![\varphi,\,\psi]\!]^M_F$ と定めた．すなわち，

$$\begin{aligned}&([\![\varphi,\,\psi]\!]^M_F = \varphi \diamond \widehat{F}^M\,[\![\varphi,\,\psi]\!]^M_F \diamond \psi)\\ &\wedge (\forall\ f,\ f = \varphi \diamond \widehat{F}^M\,f \diamond \psi \implies [\![\varphi,\,\psi]\!]^M_F \sqsubseteq f).\end{aligned} \tag{MHylo-Charn}$$

この効果付き hylomorphism は，様々な運算法則を満たす．例えば，Hylo-FuxionC, Hylo-FuxionA と同様に，以下の融合則が成り立っている．

$$h \diamond \varphi = \varphi' \diamond \widehat{F}^M\,h \implies h \diamond [\![\varphi,\,\psi]\!]^M_F = [\![\varphi',\,\psi]\!]^M_F, \tag{MHylo-FusionC}$$

$$\psi \diamond h = \widehat{F}^M\,h \diamond \psi' \implies [\![\varphi,\,\psi]\!]^M_F \diamond h = [\![\varphi,\,\psi']\!]^M_F. \tag{MHylo-FusionA}$$

また，Hylo-Ana および Hylo-Cata と同様にして，効果付き anamorphism および効果付き catamorphism を定義することができる：

$$[\![\psi]\!]^M_F = [\![\mathsf{return}^M_{\mu F} \circ \mathsf{in}_F,\,\psi]\!]^M_F \qquad : X \to M\,\mu F, \tag{MHylo-MAna}$$

$$(\!|\varphi|\!)^M_F = [\![\varphi,\,\mathsf{return}^M_{F\,\mu F} \circ \mathsf{in}^{-1}_F]\!]^M_F \qquad : \mu F \to M\,Y. \tag{MHylo-MCata}$$

この定義の下で，Ana-Charn, Cata-Charn, Ana-Cata-Hylo に対応した以下が成り立っている：

$$[\![\psi]\!]^M_F \sqsubseteq f \iff f = (\mathsf{return}^M_{\mu F} \circ \mathsf{in}_F) \diamond \widehat{F}^M\,f \diamond \psi \tag{MAna-Charn}$$

$$(\!|\varphi|\!)^M_F \sqsubseteq f \iff f = \varphi \diamond \widehat{F}^M\,f \circ \mathsf{in}^{-1}_F \tag{MCata-Charn}$$

$$[\![\varphi,\,\psi]\!]^M_F = (\!|\varphi|\!)^M_F \diamond [\![\psi]\!]^M_F. \tag{MAna-MCata-MHylo}$$

4.2 準備 2: 遅延モナド

Capretta による遅延モナド [3] について詳しく説明する．本論文で提案する遅延 hylomorphism の核は，MHylo-Equation において，M が以下で述べる遅延モナドである場合を考えることである．

データ型 X に対して，余帰納的データ型 $\mathsf{D}\,X$ を，二つのコンストラクタ $\ulcorner - \urcorner$ および $\triangleright$ を用いて，以下のように定義する．

$$\textbf{CoInductive}\ \mathsf{D}\ \{X : \mathsf{Type}\} : \mathsf{Type} := \ulcorner - \urcorner : X \to \mathsf{D}\,X \mid \triangleright : \mathsf{D}\,X \to \mathsf{D}\,X.$$

あとで示すように，D はモナドになっているので，D を遅延モナド（delay monad）という．

直観的には，$\ulcorner x \urcorner$ は今すぐ項 x が得られることを意味し，$\triangleright$ は項が得られるのが 1 ステップ遅延することを表す．例えば，$\triangleright\triangleright\triangleright\ulcorner \mathsf{true} \urcorner$: $\mathsf{D}\,\mathbf{bool}$ であれば，3 ステップ後に値 true が返されることを意味する．D は余帰納的に定義されているので，$\triangleright$ が無限に続く元 $\triangleright\ \triangleright\ \triangleright \cdots = \triangleright^\infty$ が存在して，$\triangleright^\infty : \mathsf{D}\,X$ となる．Coq では，以下のように定義することができる：

$$\textbf{CoFixpoint}\ \triangleright^\infty\ \{X : \mathsf{Type}\} : \mathsf{D}\,X = \triangleright\triangleright^\infty.$$

直観的には，$\triangleright^\infty$ は，項が永久に返らないことを意味する値，すなわち無限ループに陥ることを意味する値である．

一度遅延モナドに包まれた値は，余帰納的データ型となるため，Coq では外から match 式で分解されない限り評価が進まない．したがって，遅延データ型から値を取り出すための関数として，

$$\begin{aligned}
&\mathsf{Fixpoint}\ \mathsf{evalstep}\ \{X : \mathsf{Type}\}\ (n : \mathbb{N})\ (dx : \mathsf{D}\,X) : \mathbf{Maybe}\,X := \\
&\quad \textbf{match}\ (dx,\, n)\ \textbf{with} \\
&\quad \mid (\ulcorner x \urcorner,\, _) \quad\ \Rightarrow \mathsf{just}\,x \\
&\quad \mid (\triangleright _,\, \mathsf{O}) \quad\ \Rightarrow \mathsf{nothing} \\
&\quad \mid (\triangleright\, dx',\, \mathsf{S}\,n') \Rightarrow \mathsf{evalstep}\ n'\ dx' \\
&\quad \textbf{end}.
\end{aligned}$$

を定義する．直観的には，$\mathsf{evalstep}\,n\,dx$ は，dx の評価を n ステップだけ進め，その間に値 $\ulcorner x \urcorner$ が得られたらその値を $\mathsf{just}\,x$ の形で返し，値が得られずに力尽きてしまったら nothing を返すものである．例えば，$\mathsf{evalstep}\,2\,(\triangleright\triangleright\ulcorner a \urcorner) = \mathsf{just}\,a$ であるが，$\mathsf{evalstep}\,1\,(\triangleright\triangleright\ulcorner a \urcorner) = \mathsf{nothing}$ である．

次に，遅延モナドに包まれた値どうしの同値性を議論するために，次の強双模倣関係 $\sim$ を考える．

$$\begin{aligned}
&\textbf{CoInductive}\ (\sim)\ \{X : \mathsf{Type}\} : \mathsf{D}\,X \to \mathsf{D}\,X \to \mathsf{Prop} := \\
&\mid \mathsf{sim_value} : \forall\ (x : X),\ (\ulcorner x \urcorner \sim \ulcorner x \urcorner) \\
&\mid \mathsf{sim_later}\ \ : \forall\ (dx_0\ dx_1 : \mathsf{D}\,X),\ (dx_0 \sim dx_1 \to \triangleright\, dx_0 \sim \triangleright\, dx_1).
\end{aligned}$$

強双模倣 $\sim$ は $\mathsf{D}\,X$ の型がつく要素同士の等しさとしてかなり「自然」なものであるから，一見すると $dx_0 \sim dx_1 \to dx_0 = dx_1$ が成り立つように見えるが，Coq ではこれは示すことができないことが知られている [6]．したがって，以下の仮定を追加しておく．

$$\textbf{Axiom}\ \mathsf{bisimulation\_as\_equality} : \forall\ \{X : \mathsf{Type}\}\ (dx_0\ dx_1 : \mathsf{D}\,X),\ (dx_0 \sim dx_1 \to dx_0 = dx_1).$$

$\mathsf{D} : \mathsf{Type} \to \mathsf{Type}$ は関手である．実際，写像 $f : X \to Y$ に対して，$\mathsf{fmap}^{\mathsf{D}}\,f = \mathsf{D}\,f : \mathsf{D}\,X \to \mathsf{D}\,Y$ を，以下のように定義できる．

$$\begin{aligned}
&\textbf{CoFixpoint}\ \mathsf{fmap}^{\mathsf{D}}\ \{X\ Y : \mathsf{Type}\}\ (f : X \to Y)\ (dx : \mathsf{D}\,X) : \mathsf{D}\,Y := \\
&\ \textbf{match}\ dx\ \textbf{with} \\
&\mid \ulcorner x \urcorner \ \Rightarrow \ulcorner f\,x \urcorner \\
&\mid \triangleright dx' \Rightarrow \triangleright(\mathsf{fmap}^{\mathsf{D}}\ f\ dx') \\
&\ \textbf{end}.
\end{aligned}$$

この $\mathsf{fmap}^{\mathsf{D}}$ のもとで，D は関手則を満たす（証明は簡単な余帰納法による）．さらに，D はモナドである．$\mathsf{return}^{\mathsf{D}}$ および $(\gg\!=^{\mathsf{D}})$ は，以下のように定義することができる．

$$
\begin{aligned}
&\textbf{Definition } \mathsf{return}^{\mathsf{D}}\ \{X : \mathsf{Type}\}\ (x : X) : \mathsf{D}\,X := \ulcorner x \urcorner.\\
&\textbf{CoFixpoint } (\gg\!=^{\mathsf{D}})\ \{X\ Y : \mathsf{Type}\}\ (dx : \mathsf{D}\,X)\ (f : X \to \mathsf{D}\,Y) : \mathsf{D}\,Y\\
&\quad := \textbf{ match } dx \textbf{ with}\\
&\qquad \mid \ulcorner x \urcorner \ \Rightarrow f\,x\\
&\qquad \mid \triangleright dx' \Rightarrow \triangleright(f \gg\!=^{\mathsf{D}} dx')\\
&\qquad \textbf{end}.
\end{aligned}
$$

この $\mathsf{return}^{\mathsf{D}}$ および $(\gg\!=^{\mathsf{D}})$ のもとで，D はモナド則を満たす（証明は簡単な余帰納法による）．

次に，$dx : \mathsf{D}\,X$ が値 $x : X$ に収束する（converge），すなわち，有限時間内に値 x を返すことを表す $dx \downarrow x$ および，dx が発散する（diverge），すなわち有限時間内には値を返さないことを表す $dx^{\uparrow}$ を，以下のように定義する．

$$
\begin{aligned}
&\textbf{Inductive } (\downarrow)\ \{A : \mathsf{Type}\} : \mathsf{D}\,A \to A \to \mathsf{Prop} :=\\
&\ \mid \mathsf{converge\_now}\ : \forall\ (a : A),\ (\ulcorner a \urcorner \downarrow a)\\
&\ \mid \mathsf{converge\_later} : \forall\ (da : \mathsf{D}\,A)\ (a : A),\ (da \downarrow a \to \triangleright da \downarrow a).\\
&\textbf{CoInductive } (\uparrow)\ \{A : \mathsf{Type}\} : \mathsf{D}\,A \to \mathsf{Prop} :=\\
&\ \mid \mathsf{diverge} : \forall\ (da : \mathsf{D}\,A),\ da^{\uparrow} \to (\triangleright da)^{\uparrow}.
\end{aligned}
$$

例えば，$\triangleright\triangleright\ulcorner \mathsf{O} \urcorner \downarrow \mathsf{O}$ であり（これは明らかである），$(\triangleright^{\infty})^{\uparrow}$ である（これは余帰納法により容易に示せる）．

次に，弱双模倣関係 $\approx$ を，$\downarrow$ を使って次のように定義する．

$$
\begin{aligned}
&\textbf{CoInductive } (\approx)\ \{A : \mathsf{Type}\} : \mathsf{D}\,A \to \mathsf{D}\,A \to \mathsf{Prop} :=\\
&\ \mid \mathsf{wsim\_value} : \forall\ (x\ y : \mathsf{D}\,A)\ (a : A),\ (x \downarrow a \to y \downarrow a \to x \approx y)\\
&\ \mid \mathsf{wsim\_later}\ : \forall\ (x\ y : \mathsf{D}\,A),\ (x \approx y \to \triangleright x \approx \triangleright y).
\end{aligned}
$$

また，$\approx$ は外延性によって，関数の間の関係 $\approx^{ext}$ へと次のように拡張される：

$$
\textbf{Definition } (\approx^{ext})\ \{X\ Y : \mathsf{Type}\}\ (f_0\ f_1 : X \to \mathsf{D}\,Y) := \forall\ (x : X),\ f_0\,x \approx f_1\,x.
$$

最後に，次の順序関係 $\sqsubseteq$ および $\sqsubseteq^{ext}$ を定める．

$$
\begin{aligned}
&\textbf{CoInductive } (\sqsubseteq)\ \{X : \mathsf{Type}\} : \mathsf{D}\,X \to \mathsf{D}\,X \to \mathsf{Prop} :=\\
&\ \mid \mathsf{leq\_value}\ : \forall\ (dx_0\ dx_1 : \mathsf{D}\,X)\ (x : X),\ dx_0 \downarrow x \to dx_1 \downarrow x \to dx_0 \sqsubseteq dx_1\\
&\ \mid \mathsf{leq\_blater} : \forall\ (dx_0\ dx_1 : \mathsf{D}\,X),\ dx_0 \sqsubseteq dx_1 \to \triangleright dx_0 \sqsubseteq \triangleright dx_1\\
&\ \mid \mathsf{leq\_llater}\ : \forall\ (dx_0\ dx_1 : \mathsf{D}\,X),\ dx_0 \sqsubseteq dx_1 \to \triangleright dx_0 \sqsubseteq dx_1.\\
&\textbf{Definition } (\sqsubseteq^{ext})\ \{X\ Y : \mathsf{Type}\}\ (f_0\ f_1 : X \to \mathsf{D}\,Y) := \forall\ (x : X),\ f_0\,x \sqsubseteq f_1\,x.
\end{aligned}
$$

直観的には，$dx_0 \sqsubseteq dx_1$ は，dx_0 は dx_1 に（無限個も含めて）$\triangleright$ をいくつか追加して得られることを表す．特に，任意の dx に対して，$\triangleright^{\infty} \sqsubseteq dx$ が成り立っている． これらの定義の下で，以下のことが成り立つ．

$$
\begin{aligned}
&\textbf{Lemma } \mathsf{weakbisim\_converge} :\\
&\quad \forall\ \{X : \mathsf{Type}\}\ (dx_0\ dx_1 : \mathsf{D}\,X),\ dx_0 \approx dx_1 \leftrightarrow (\forall\ (x : X),\ dx_0 \downarrow x \leftrightarrow dx_1 \downarrow x).\\
&\textbf{Lemma } \mathsf{sqsubseteq\_converge} :\\
&\quad \forall\ \{X : \mathsf{Type}\}\ (dx_0\ dx_1 : \mathsf{D}\,X),\ dx_0 \sqsubseteq dx_1 \leftrightarrow (\forall\ (x : X),\ dx_0 \downarrow x \to dx_1 \downarrow x).
\end{aligned}
$$

不動点コンビネータ　Φ: $(X \to \mathsf{D}\,Y) \to X \to \mathsf{D}\,Y$ が finitary であるとは，任意の f: $X \to \mathsf{D}\,Y$ および x: X に対して，$\Phi\,f\,x \downarrow y$ ならば，或る $n : \mathbb{N}$ および $x_1, \ldots x_n : X$, $y_1, \ldots, y_n : Y$ が存在して，以下の 2 条件が成り立つことである．

- $\bigwedge_{i=1}^{n} f\,x_i \downarrow y_i$

- $\forall\ (g : X \to \mathsf{D}\,Y),\ (\bigwedge_{i=1}^{n} g\,x_i \downarrow y_i) \to \Phi\,g\,x \downarrow y$

直観的には，$\Phi\,f\,x$ は，有限個の入力 $x_1, \ldots, x_n$ に対する f の値にのみ出力が依存するということである．

Capretta は，遅延モナドを用いた不動点コンビネータ

$$\mathsf{dfix} : \forall\ \{X\ Y : \mathsf{Type}\},\ ((X \to \mathsf{D}\,Y) \to X \to \mathsf{D}\,Y) \to X \to \mathsf{D}\,Y$$

の定義を与えた．この不動点コンビネータは，以下を満たすことが示されている．

Theorem dfix_least_fixed_point :
$\forall\ \{X\ Y : \mathsf{Type}\}\ (\Phi : (X \to \mathsf{D}\,Y) \to X \to \mathsf{D}\,Y)$,
$$\Phi \text{ が finitary} \to \left(\begin{array}{l} \Phi\,(\mathsf{dfix}\,\Phi) \approx^{ext} \mathsf{dfix}\,\Phi \\ \wedge\ \forall\ (\mathit{prefix} : X \to \mathsf{D}\,Y),\ \Phi\,\mathit{prefix} \sqsubseteq^{ext} \mathit{prefix} \to \mathsf{dfix}\,\Phi \sqsubseteq^{ext} \mathit{prefix} \end{array} \right).$$

したがって，$\mathsf{dfix}\,\Phi$ は（順序 $\sqsubseteq^{ext}$ の下で）Φ の最小不動点である．Capretta は，この命題（と明らかに同値な命題）の Coq による証明を与えている[3]．以下では，この dfix のことを，遅延不動点コンビネータということにする．

遅延モナドの分配則　正則関手 F に対して，分配則 $\mathsf{dist}^{\mathsf{D},F} : F \circ \mathsf{D} \Rightarrow \mathsf{D} \circ F$ を，F の構造に関する帰納的定義によって定義することができる [22]．各 $\mathsf{dist}^{\mathsf{D},F}$ は，直観的には $\triangleright$ を全て外側に追い出すような定義になる．以下の 2 つの例を具体的に与えておく．

- リスト関手 L_A および D の分配則 $\mathsf{dist}^{\mathsf{D},L_A} : L_A \circ \mathsf{D} \Rightarrow \mathsf{D} \circ L_A$

```
Definition dist^{D,L_A} {A X : Type} (ldx : L_A (D X)) : D (L_A X) :=
  match ldx with
  | inl ()          ⇒ ⌜inl ()⌝
  | inr (a, dax) ⇒ dax ≫= (λ ax. ⌜inr (a, ax)⌝)
  end.
```

- 木関手 T_A および D の分配則 $\mathsf{dist}^{\mathsf{D},T_A} : T_A \circ \mathsf{D} \Rightarrow \mathsf{D} \circ T_A$

```
Definition dist^{D,T_A} {A X : Type} (ldx : T_A (D X)) : D (T_A X) :=
  match ldx with
  | inl ()                    ⇒ ⌜inl ()⌝
  | inr (dxax0, a, dxax1) ⇒ dxax0 ≫= (λ xax0. dxax1 ≫= (λ xax1. ⌜inr (xax0, a, xax1)⌝))
  end.
```

4.3　遅延 hylomorphism

さて，遅延 hylomorphism のアイデアは，効果付き hylomorphism において，モナド M が D であった場合を考えることである．すなわち，$\psi : X \to \mathsf{D}\,(F\,X)$ および $\varphi : F\,Y \to \mathsf{D}\,Y$ に対して，$[\![\varphi, \psi]\!]_F^{\mathsf{D}}$ を考えれば良い．しかし，我々の目的は通常の hylomorphism を Coq で扱うことなので，代数および余代数は，最初から D で汚れているものではなくて，純粋なものを与えられるようにする．すなわち，$\psi : X \to F\,X$ および $\varphi : F\,Y \to Y$ に対して，

$$[\![\varphi, \psi]\!]_F^{\triangleright} = [\![\mathsf{return}_Y^{\mathsf{D}} \circ \varphi, \mathsf{return}_X^{\mathsf{D}} \circ \psi]\!]$$

となる $[\![-, -]\!]_F^{\triangleright}$ を，遅延 hylomorphism の定義としたい．

以下の補題は，遅延 hylomorphism の **Set** での解を与えるためのものである．

[3] 証明の全体は，`http://www.duplavis.com/venanzio/publications/rec_coind.v` に公開されている．

補題 1（遅延 **hylomorphism** の基本補題） F を正則関手とする．関数 $\Phi_F : (X \to \mathsf{D}\,Y) \to X \to \mathsf{D}\,Y$ を，

$$\Phi_F\, f = (\mathsf{return}^{\mathsf{D}}_X \circ \varphi) \diamond \widehat{F}^{\mathsf{D}}\, f \diamond (\mathsf{return}^{\mathsf{D}}_{FX} \circ \psi)$$

で定める．MHylo-Equation において，M を遅延モナド $\mathsf{D} : \mathbf{Set} \to \mathbf{Set}$ に特化させた方程式

$$f \approx^{ext} (\mathsf{return}^{\mathsf{D}}_X \circ \varphi) \diamond \widehat{F}^{\mathsf{D}}\, f \diamond (\mathsf{return}^{\mathsf{D}}_{FX} \circ \psi).$$

は最小解 $\mathsf{dfix}\,\Phi_F$ をもつ．

証明　Φ_F が finitary であることと，fix_least_fixed_point から直ちに従う．　□

したがって，上の $\mathsf{dfix}\,\Phi_F$ を，遅延 **hylomorphism** といい，$[\![\varphi, \psi]\!]^{\triangleright}_F$ と書くことにする．この定義の下で，

$$\begin{aligned}&\big([\![\varphi, \psi]\!]^{\triangleright}_F \approx^{ext} (\mathsf{return}^{\mathsf{D}}_Y \circ \varphi) \diamond \widehat{F}^{\mathsf{D}}\, [\![\varphi, \psi]\!]^{\triangleright}_F \diamond (\mathsf{return}^{\mathsf{D}}_{FX} \circ \psi)\big)\\ &\wedge \big(\forall\ f,\ f \approx^{ext} (\mathsf{return}^{\mathsf{D}}_Y \circ \varphi) \diamond \widehat{F}^{\mathsf{D}}\, f \diamond (\mathsf{return}^{\mathsf{D}}_{FX} \circ \psi) \implies [\![\varphi, \psi]\!]^{\triangleright}_F \sqsubseteq^{ext} f\big).\end{aligned} \tag{DHylo-Charn}$$

が成り立つ．

運算法則　MHylo-FusionC および MHylo-FusionA を，D に特化させて整理することにより，以下の DHylo-FusionC および DHylo-FusionA を得る．ただし，遅延コンビネータの性質から，MHylo-FusionC および MHylo-FusionA の $=$ は $\approx^{ext}$ に変わることに注意が必要である．

$$h \circ \varphi \approx^{ext} (\mathsf{return}^{\mathsf{D}} \circ \varphi') \diamond \widehat{F}^{\mathsf{D}}\, h \implies h \diamond [\![\varphi, \psi]\!]^{\triangleright}_F \approx^{ext} [\![\varphi', \psi]\!]^{\triangleright}_F \tag{DHylo-FusionC}$$

$$(\mathsf{return}^{\mathsf{D}} \circ \psi) \diamond h \approx^{ext} \widehat{F}^{\mathsf{D}}\, h \circ \psi' \implies [\![\varphi, \psi]\!]^{\triangleright}_F \diamond h \approx^{ext} [\![\varphi, \psi']\!]^{\triangleright}_F \tag{DHylo-FusionA}$$

Coq による実装　遅延不動点コンビネータのおかげで，DHylo-Charn は，ほぼそのまま Coq 風の定義に書き直すことができる．

Definition $\Phi_F\ \{X\ Y : \mathsf{Type}\}\ (\varphi : F\,Y \to Y)\ (\psi : X \to F\,X)\ (f : X \to \mathsf{D}\,Y) : X \to \mathsf{D}\,Y$
$:= \mathsf{fmap}^{\mathsf{D}}\, \varphi \circ \mu^{\mathsf{D}}_{FY} \circ \mathsf{D}\,(\mathsf{dist}^{\mathsf{D},F}_Y \circ F\, f) \circ \mathsf{return}^{\mathsf{D}}_{FX} \circ \psi.$
Definition $[\![-, -]\!]^{\triangleright}_F\ \{X\ Y : \mathsf{Type}\}\ (\varphi : F\,Y \to Y)\ (\psi : X \to F\,X) := \mathsf{dfix}\,(\Phi_F\, \varphi\, \psi)$

これで，動く hylomorphism の定義を得ることができた．例えば，クイックソートの場合，

Eval compute in $\mathsf{evalstep}\,5\,([\![\varphi_{bstree}, \psi_{intrav}]\!]^{\triangleright}_{T_{\mathbb{N}}}\,[4;2;3;1])$

などとすることで，実行結果として $\mathsf{jsut}\,[1;2;3;4]$ を得ることができる．コラッツ列の例についても，次のように動かすことができる．

Eval compute in $\mathsf{evalstep}\,10\,([\![\varphi_{find}\,1, \psi_{colgen}]\!]^{\triangleright}_{L_{\mathbb{N}}}\,3)$ (* $\implies \mathsf{just}\,(\mathsf{just}\,7)$ *).

もちろん，evalstep に渡す「燃料」が足りなければ，すなわち，評価のステップ数が足りなければ，途中で力尽きて nothing を返してしまう：

Eval compute in $\mathsf{evalstep}\,4\,([\![\varphi_{bstree}, \psi_{intrav}]\!]^{\triangleright}_{T_{\mathbb{N}}}\,[4;2;3;1])$ (* $\implies$ nothing *).

したがって，評価して具体的な値を得たいと思うたびに，入力に対して十分大きな「燃料」を与える必要がある．

4.4　遅延 catamorphism および遅延 anamorphism

遅延 hylomorphism が得られたので，遅延 catamorphism $(\!|-|\!)^{\triangleright}_F$ と遅延 anamorphism $[\!(-)\!]^{\triangleright}_F$ を，それぞれ次のように定義することができる．

Definition $(\!|-|\!)^{\triangleright}_F\ \{Y : \mathsf{Type}\}\ (\varphi : F\,Y \to Y) : \mu F \to \mathsf{D}\,Y := [\![\varphi, \mathsf{in}^{-1}_F]\!]^{\triangleright}_F.$
Definition $[\!(-)\!]^{\triangleright}_F\ \{X : \mathsf{Type}\}\ (\psi : X \to F\,X) : X \to \mathsf{D}\,\mu F := [\![\mathsf{in}_F, \psi]\!]^{\triangleright}_F.$

注目すべきは，遅延 anamorphism の型が，$\psi : X \to F\,X$ に対して $[\![\psi]\!]^{\triangleright}_F : X \to \mathsf{D}\,\mu F$ となっており，D に包まれてはいるものの μF を返している点である．**Set** の anamorphism は νF を返す射であったのに対し，遅延 anamorphism は μF を返す射とすることができる．これは，直観的には，無限木を成長させる anamorphism $[\![\psi]\!]_F\,x$ は，遅延 anamorphism においては $[\![\psi]\!]^{\triangleright}_F\,x = \triangleright^{\infty}$ とすることができるためである．

MAna-MCata-MHylo を D に特化することにより，以下が得られる．

$$[\![\varphi,\,\psi]\!]^{\triangleright}_F \approx^{ext} (\!|\,\varphi\,|\!)^{\triangleright}_F \diamond [\![\psi]\!]^{\triangleright}_F. \qquad \text{(DAna-DCata-DHylo)}$$

Coq による実装　Coq による実装は上の通りであるが，動作例も二つほど示しておく．

Eval compute in evalstep 10 ($[\![\psi_{colgen}]\!]^{\triangleright}_{L_{\mathbb{N}}}$ 3)　　(* $\Longrightarrow$ just [3; 10; 5; 16; 8; 4; 2; 1] *).
Eval compute in evalstep 10 ($(\!|\,\varphi_{find}\,3\,|\!)^{\triangleright}_{L_{\mathbb{N}}}$ [0; 1; 2; 3; 4])　(* $\Longrightarrow$ just 3 *).

4.5　考察：遅延 hylomorphism を Coq で扱うことについて

本論文では，Coq への shallow embedding ができる hylomorphism として，遅延 hylomorphism を提案した．3.2 節と同様に，遅延 hylomorphism を Coq による hylomorphism を用いたプログラミングおよびその運算を支援する手法として評価するとすれば，以下の 2 点が指摘できる．

- 汎用プログラミングの技法として簡便で強力である．遅延 hylomorphism を用いることで，Coq の中に，F, φ, ψ について汎用な高階関数 $[\![\varphi,\,\psi]\!]^{\triangleright}_F$ を定義することができた．それゆえ，ユーザは，F, φ, ψ の定義を与えるだけで，動作するプログラムを得ることができる．特に，コラッツ列のような停止性が非自明な関数となる hylomorphism を容易に扱うことができた．したがって，プログラミングの手法としては，再帰的余代数を用いたアプローチよりかなり簡便で強力な手法であると言えそうである．
- 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 D で包まれているため，Coq の評価器で具体的な値を取り出すには eval_step のような関数が必要である．これは再帰的余代数による hylomorphism にはなかった面倒であると言える．また，D に包まれているぶん，計算にはオーバヘッドが生じている．特に，Coq で遅延 hylomorphism を定義するために用いた dfix コンビネータは，実用を目指して定義されたものというよりは，遅延モナドを用いた場合の計算可能性について議論するために理論的に導入されたものであり，その動作は実用に耐えるほど高速とは言えないと考えられる．

5　まとめ，関連研究，今後の課題

本研究は，Tesson らによるプログラム運算のための Coq タクティックライブラリ [24] や，Murata らによる再帰図式のための Coq ライブラリ [19] を先行研究と位置付けて，Hylomorphism の形式化を試みたものである．第一の案として，再帰的余代数 [20] を用いたアプローチを検討した．確かに hylo 方程式を満たす関数を定義することはできるものの，関数を定義するだけで面倒な証明が伴うことを指摘した．第二の案として，遅延モナド [3] および効果付き再帰図式 [8, 22] を用いた遅延 hylomorphism を提案および検討した．このアプローチの下では，実行可能な hylomorphism が簡単に得られることを確かめた．

関連研究　以下では，ここまでに触れられなかった，定理証明支援系を用いた運算定理の形式検証という点で本研究と類似する研究をいくつか挙げておく．

Mu ら [18] や Chiang ら [5] は，定理証明支援系 Agda を用いて，プログラム運算の教科書である Algebra of Programming [2] の一部の形式化した．これらの研究は，関係の圏 **Rel** をベースにした運算定理の形式化を与えており，実際に動作するプログラムの抽出は考慮に入れていないようである．

Emoto らによる生成集約検査プログラミングの Coq による検証 [7] や，Frederic らによる検証された並列プログラムを生成する Coq ライブラリ [15, 14] は，本研究と同様にプログラム抽出を視野に入れた研究であるが，本研究とは扱っている運算定理が異なる．

今後の課題　今後の課題として，以下の 4 点を挙げておく．

1. 遅延 **hylomorphism** について成り立つ法則の検証．Pardo らの効果付き hylomorphism について成り立つ性質を遅延モナド D へ特化させることによって，遅延 hylomorphism について成り立つ法則が得られるが，これらの法則の Coq による証明は未完成である．これらの証明は通常等式や不等式を用いた推論によって行われるが，少なくとも等式推論の部分については，Tesson ら [24] や Murata ら [19] と同様の手法を使うことができると考えている．Hylomorphism は順序集合の上で定義されるため，不等式推論も必要であるが，これについては村田らによる不等式変形のためのタクティックライブラリ [31] も参考になると考えている．
2. 遅延 dfix コンビネータを用いることによる計算速度への影響の調査．4.5 節でも指摘したように，Coq における遅延 hylomorphism は，遅延モナドを用いた dfix コンビネータを用いて定義されているが，この dfix コンビネータは実用に耐えるほど高速とは言えないと考えられる．具体的にどれぐらいのオーバヘッドが生ずるのか調査したり，より高速で簡便な代替実装を検討することは，今後の課題である．
3. 酸性雨定理の証明．Takano らによる酸性雨定理 [23] は，hylomorphism に関わる強力な運算法則の一つであり，是非とも Coq での形式化が望まれる．酸性雨定理の証明には free theorem [29] を用いるため，Coq で free theorem をうまく扱う方法を模索する必要がある．Keller による paramcoq ライブラリ [13] が利用できると考えられるが，詳細な検討は今後の課題である．
4. 遅延 **Dynamorphism**．Kabanov らによる dynamorphism [12] は，動的計画法を扱う再帰図式として有用なものである．本研究で遅延 hylomorphism を提案したのと同様に，遅延 dynamorphism のようなものを定義すれば，Coq で動的計画法を用いたプログラミングやその運算を行う上が可能になると考えられる．このアイデアの詳細な検討は，今後の課題である．

謝辞

原稿を注意深くお読みいただき，的確で貴重なコメントをくださった匿名の 3 名の査読者に深く感謝いたします．本研究は JSPS 科研費 JP19K11903 の助成を受けたものです．

参考文献

[1] Aarts, C., Backhouse, R., Hoogendijk, P., Voermans, E., and van der woude, J.: A relational theory of datatypes, 1992.

[2] Bird, R. and de Moor, O.: *Algebra of Programming*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

[3] Capretta, V.: General recursion via coinductive types, *Logical Methods in Computer Science*, Vol. 1, No. 2(2005), pp. 1–28.

[4] Capretta, V., Uustalu, T., and Vene, V.: Recursive coalgebras from comonads, *Information and Computation*, Vol. 204(2006), pp. 437–468.

[5] Chiang, Y.-H. and Mu, S.-C.: Formal derivation of Greedy algorithms from relational specifications: A tutorial, *Journal of Logical and Algebraic Methods in Programming*, Vol. 85, No. 5, Part 2(2016), pp. 879 – 905. Articles dedicated to Prof. J. N. Oliveira on the occasion of his 60th birthday.

[6] Chlipala, A.: *Certified Programming with Dependent Types: A Pragmatic Introduction to the Coq Proof Assistant*, The MIT Press, 2013.

[7] Emoto, K., Loulergue, F., and Tesson, J.: A Verified Generate-Test-Aggregate Coq Library for Parallel Programs Extraction, *Interactive Theorem Proving*, Cham, Springer International Publishing, 2014, pp. 258–274.

[8] Fokkinga, M.: Monadic Maps and Folds for Arbitrary Datatypes, Technical report, University of Twente, 1994.

[9] Fokkinga, M. M. and Meijer, E.: Program Calculation Properties of Continuous Algebras, Technical report, CS-R9104, CWI, Amsterdam, 1991.

[10] Freyd, P.: Algebraically complete categories, *Category Theory*, Carboni, A., Pedicchio, M. C., and Rosolini, G.(eds.), Springer Berlin Heidelberg, 1991, pp. 95–104.

[11] Hinze, R., Wu, N., and Gibbons, J.: Conjugate hylomorphisms – Or: The mother of all structured recursion schemes, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2015, pp. 527–538.

[12] Kabanov, J. and Vene, V.: Recursion Schemes for Dynamic Programming, *Mathematics of Program Construction*, Berlin, Heidelberg, Springer Berlin Heidelberg, 2006, pp. 235–252.

[13] Keller, C. and Lasson, M.: Parametricity in an Impredicative Sort, *CSL-26th International Workshop/21st Annual Conference of the EACSL*, Vol. 16(2012), pp. 381–395.

[14] Loulergue, F., Bousdira, W., and Tesson, J.: Calculating Parallel Programs in Coq Using List Homomorphisms, *International Journal of Parallel Programming*, Vol. 45, No. 2(2017), pp. 300–319.

[15] Loulergue, F. and Tesson, J.: Certified Parallel Program Calculation in Coq: A Tutorial, *International Conference on High Performance Computing and Simulation (HPCS 2014)*, IEEE, 2014.

[16] Meijer, E., Fokkinga, M., and Paterson, R.: Functional Programming with Bananas, Lenses, Envelopes and Barbed Wire, *Proceedings of the 5th ACM Conference on Functional Programming Languages and Computer Architecture*, Springer-Verlag, 1991, pp. 124–144.

[17] Moggi, E.: Notions of computation and monads, *Information and Computation*, Vol. 93, No. 1(1991), pp. 55 – 92. Selections from 1989 IEEE Symposium on Logic in Computer Science.

[18] Mu, S.-C., Ko, H.-S., and Jansson, P.: Algebra of programming in Agda: Dependent types for relational program derivation, *Journal of Functional Programming*, Vol. 19, No. 5(2009), pp. 545–579.

[19] Murata, K. and Emoto, K.: Recursion Schemes in Coq, *Programming Languages and Systems - 17th Asian Symposium, APLAS 2019, Nusa Dua, Bali, Indonesia, December 1-4, 2019, Proceedings*, 2019, pp. 202–221.

[20] Osius, G.: Categorical set theory: A characterization of the category of sets, *Journal of Pure and Applied Algebra*, Vol. 4, No. 1(1974), pp. 79 – 119.

[21] Pardo, A.: Fusion of recursive programs with computational effects, *Theorical Computer Science*, Vol. 260(2001), pp. 165–207.

[22] Pardo, A.: Combining Datatypes and Effects, *Advanced Functional Programming*, Vene, V. and Uustalu, T.(eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, 2005, pp. 171–209.

[23] Takano, A. and Meijer, E.: Shortcut Deforestation in Calculational Form, *Proceedings of the Seventh International Conference on Functional Programming Languages and Computer Architecture (FPCA '95)*, New York, NY, USA, ACM, 1995, pp. 306–313.

[24] Tesson, J., Hashimoto, H., Hu, Z., Loulergue, F., and Takeichi, M.: Program Calculation in Coq, *Proceedings of the 13th International Conference on Algebraic Methodology and Software Technology (AMAST'10)*, Springer-Verlag, 2011, pp. 163–179.

[25] The Coq development team: *The Coq proof assistant reference manual*, 2019. Version 8.10.2.

[26] Uustalu, T. and Veltri, N.: The Delay Monad and Restriction Categories, *Theoretical Aspects of Computing ICTAC 2017*, Springer, Cham, 2017, pp. 32–50.

[27] Uustalu, T. and Vene, V.: Primitive (Co)Recursion and Course-of-Value (Co)Iteration, Categorically, *Informatica*, Vol. 10(1999), pp. 5–26.

[28] Vene, V.: *Categorical Programming with inductive and coinductive types*, PhD Thesis, University of Tartuensis, 2000.

[29] Wadler, P.: Theorems for Free!, *Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture (FPCA '89)*, New York, NY, USA, ACM, 1989, pp. 347–359.

[30] 岩崎英哉: 構成的アルゴリズム論, コンピュータ ソフトウェア, Vol. 15, No. 6(1998), pp. 547–560.

[31] 村田康佑, 江本健斗: 定理証明支援系 Coq における不等式変形記法, 情報処理学会論文誌プログラミング（*PRO*）, Vol. 11, No. 4(2018), pp. 1–12.

[32] 長谷川立: パラメトリック・ポリモルフィズム, コンピュータ ソフトウェア, Vol. 20, No. 2(2003), pp. 175–194.