異なる定理証明支援系間の証明の再利用に向けた帰納型の変換

菅野 直孝　中野 圭介　浅田 和之　菊池 健太郎

定理証明支援系には様々なものが存在するが，それぞれ論理体系や証明の記述方法などが異なるため証明を互いに利用することはできない．そのため他の定理証明支援系で証明されていることであっても，最初から書き直す必要があり生産性が低下してしまう．本研究では同じ計算基盤を持つ Coq と Lean についてこの問題を解決することを目指す．Coq では，Lean で作成された証明が Coq 上の証明として受け入れられるようにする Gilbert によって開発された coq-lean-import がプラグインとして提供されている．しかし，このプラグインは Lean で書かれた型や関数などを Coq 上の全く新しい名前の型や関数として変換するため，変換したものを Coq 上ですでに定義されている型や関数と同じものとして利用することはできない．そこで本研究では Coq と Lean の型について対応するコンストラクタと引数を全て見つけ，その型を利用している関数や証明等を書き換えるアルゴリズムを提案し，定理証明支援系間の証明の再利用を目指す．

1 はじめに

定理証明支援系は，数学的な定理の証明やプログラムの正当性を検証するシステムであり，Agda [1]，Coq [2]，Isabelle [3]，Lean [4] などが知られている．中でも INRIA で開発された Coq と Microsoft によって開発された Lean は同じ計算体系を基盤としており，帰納型や証明の記述方法などにおいて共通点が多い．具体的には，Coq と Lean では，predicative Calculus of Inductive Constructions (pCIC) と呼ばれる型システムが採用されており，これは Calculus of Constructions (CoC) に帰納型を加える拡張を行ったものである．

本研究では，coq-lean-import [5] プラグインと併用することで，既存の Coq の型の Lean での証明を Coq で利用することを可能にする Coq プラグインの開発を目的としている．coq-lean-import は Lean で記述した証明や定義を Coq 上で受け入れられるようにした Coq プラグインである．これを利用することで，Lean は使ったことがないが Coq は分かる人が Lean の動作を確認することができたり，Lean で書かれた全く新しい型や関数についての定義，命題，証明などを Coq に移植することができる．しかし，これは Lean における定義や証明を Coq に移植できはするが，Coq に既にある型の Lean での証明を Coq のその型の証明として移植できるわけではない．なぜなら，coq-lean-import によって Lean から変換された型や関数などは変換時に新しく定義されるため， Coq 上にすでにあるライブラリの同じ（ユーザーが同じだと思っている）型や関数とは完全に異なり，両者の相互利用はできないためである．例えば，Lean で示した自然数型 `nat` に関する性質の証明を coq-lean-import で Coq に変換したとしても，これは新しく定義された `nat` 型に関する性質の証明であるため，Coq の既存ライブラリの `nat` 型の性質の証明とは解釈されない．このように，coq-lean-import を用いても，Lean で定義した型や関数，定理，証明などを Coq 上の既存ライブラリのものと組み合わせて利用することは難しい．このような問題に対処するために，本論文では Coq と Lean で

A transformation of inductive types for reuse of proofs between different proof assistants

Naotaka Kanno, 東北大学大学院情報科学研究科, Graduate School of Information Sciences, Tohoku University.

Keisuke Nakano, Kazuyuki Asada, Kentaro Kikuchi, 東北大学電気通信研究所, Research Institute of Electrical Communication, Tohoku University.

「同一視できる型」を特定し，その型を利用している定義や証明等を書き換えるアルゴリズムを提案することで，定理証明支援系間の証明の再利用を目指す．ここで,「同一視できる型」とはコンストラクタや引数の順番の入れ替えにより等しくなる型のこととする．

2 Coq における証明

本章では，最初に Coq と Lean の論理的基盤である pCIC と証明項について説明する．その後，Coq のプラグインである coq-lean-import の概要とその問題点について述べる．

2.1 pCIC

Coq はフランスの INRIA によって 1984 年開発された定理証明支援系である. 主な特徴として，カリー・ハワード同型対応に基づいた証明や依存型などがあげられる. 一方，Lean は Microsoft Research によって 2013 年に開発された定理証明支援系であり，比較的歴史は浅い．これらの定理証明支援系は，型や関数などを記述するための言語を備え，タクティクと呼ばれるコマンドを使って，対話的に証明を記述することができる．Coq と Lean の計算基盤には **predicative Calculus of Inductive Constructions** (pCIC) と呼ばれる型システムが採用されており，これは階層的な型宇宙を持つ可述的な Calculus of Constructions (CoC) に帰納的型を加えたものである．CIC の式は**項**で構成されており，全ての項は型を持つ．関数型，データ型だけではなく，命題の型，型の型が存在することが pCIC の特徴である．pCIC の項の定義を以下に示す．

定義 2.1 (pCIC の項)**.** ここでは x, y を変数，i を自然数，T, U, t, u を項とする．

$$\begin{aligned} T ::= {} & \mathbb{P} \mid \mathbb{T}_i \mid l \mid \Pi x : T.U \mid \lambda x : T.\, u \mid (t\ u) \\ & \mid \mathrm{Ind}(X : T)\{T_1, ..., T_n\} \mid \mathrm{Constr}(i, T) \\ & \mid \mathrm{Elim}(T, T)\{T_1, ..., T_n\} \end{aligned}$$

$\mathbb{P}$，$\mathbb{T}_i$ はそれぞれ命題，型を表し，まとめてソートと呼ばれる．l はラベル付けを行う．$\Pi x : T.U$ は U が x に依存する依存積型を表している．もし x が U に自由変数として存在しない場合，$T \to U$ と書くこともできる．$\lambda x : T.\, u$ は T の要素 x を項 u に移す関数を表す．$(t\ u)$ は t の u への関数適用を表す．また，項 t を項 u の自由変数 x に捕獲回避代入することを $u[t/x]$ と表記する．Ind，Constr，Elim の規則はそれぞれ帰納的データ型，そのコンストラクタの項および除去規則を形成する．

除去規則の型は以下のように定義される．

定義 2.2 (除去規則における型)**.** C をコンストラクタ X の型，Q と c を項とする．ここで除去規則 C の型 $\xi(I, Q, c, C) \equiv \xi_X(Q, c, C)$ を以下のように定義する．

$$\begin{aligned} \xi_X(Q, c, P \to N) = {} & \Pi p : P.(\Pi \vec{x} : \vec{M}.(Q\ \vec{m}\ (p\ \vec{x}))) \\ & \to \xi_X(Q, (c\ p), N) \\ & \text{for } P \equiv \Pi \vec{x} : \vec{M}.(X\ \vec{m}) \\ \xi_X(Q, c, \Pi x : M.N) = {} & \Pi x : M.\xi_X(Q, c\ x, N) \\ \xi_X(Q, c, X\ \vec{a}) = {} & Q\ \vec{a}\ c \end{aligned}$$

型付け規則とは，型システムが項に型を割り当てる方法を記述する推論規則である.「式 e が型 τ を持つ」ことを表す表現は型判断と呼ばれ，$e : \tau$ と略記する．ただ，変数の型は束縛されている型の情報によって異なるため，型判断はできない．そこで，変数に対して仮定している型の情報を**型環境**と呼ばれる変数から型への部分関数で表す必要がある．型環境は主にメタ変数 Γ で表され,「$\Gamma(e) = \tau$ のとき，$e : \tau$ である」ということが出来る．そして，この型環境を用いた型判断は $\Gamma \vdash e : \tau$ と表すことができ，これは Γ という仮定の下で型判断 $e : \tau$ が導出される，という意味である．そして，型判断 $\Gamma \vdash e : \tau$ が成り立つような Γ と τ が存在する場合，e は**型が付く**という．

型付け規則はこのような型判断を正しく導出する手段である．型付け規則の横線の上にある型判断を**前提**，横線の下にある型判断を**結論**と呼ぶ．型付け規則の意味は,「前提の型判断が全て導出出来るならば，結論の型判断が導出される」ことである．そして，型判断 $\Gamma \vdash e : \tau$ が正しく導出出来ているとは，根が $\Gamma \vdash e : \tau$ ですべての辺が型付け規則に沿っており，すべての葉が前提のない型付け規則が適用された形になっている木が存在することである．そのような木を導出木という．[8] を参考とした pCIC の型付け規則を図 1 に示す．

この型付け規則では，この後の変換アルゴリズムを考えるために $\mathcal{L}$ を導入している．$\mathcal{L}$ は以下のように

$$\frac{}{\bullet \vdash_{\mathrm{CIC}}} \text{(Empty)} \qquad \frac{\Gamma \vdash_{\mathrm{CIC}} T : s \qquad x \notin \Gamma}{\Gamma, x : T \vdash_{\mathrm{CIC}}} \text{(Decl)} \qquad \frac{\Gamma \vdash_{\mathrm{CIC}}}{\Gamma \vdash_{\mathrm{CIC}} \mathbb{T}_i : \mathbb{T}_{i+1}} \text{(Type)}$$

$$\frac{\Gamma \vdash_{\mathrm{CIC}}}{\Gamma \vdash_{\mathrm{CIC}} \mathbb{P} : \mathbb{T}_i} \text{(Prop)} \qquad \frac{\Gamma \vdash_{\mathrm{CIC}} \qquad (x : T) \in \Gamma}{\Gamma \vdash_{\mathrm{CIC}} x : T} \text{(Var)} \quad \frac{\Gamma \vdash_{\mathrm{CIC}} \qquad (\vdash_{\mathrm{CIC}} t : T, l) \in \mathcal{L}}{\Gamma \vdash_{\mathrm{CIC}} l : T} \text{(VarL)}$$

$$\frac{\Gamma \vdash_{\mathrm{CIC}} t : (\Pi x : A.B) \qquad \Gamma \vdash_{\mathrm{CIC}} t' : A}{\Gamma \vdash_{\mathrm{CIC}} (t\ t') : B[t'/x]} \text{(App)} \qquad \frac{\Gamma \vdash_{\mathrm{CIC}} A : s \qquad \Gamma, x : A \vdash_{\mathrm{CIC}} B : s' \qquad (s, s', s'') \in R_\Pi}{\Gamma \vdash_{\mathrm{CIC}} \Pi x : A.B : s''} \text{(Prod)}$$

$$\frac{\Gamma, x : A \vdash_{\mathrm{CIC}} t : B}{\Gamma \vdash_{\mathrm{CIC}} (\lambda x : A.\ t) : (\Pi x : A.B)} \text{(Lam)} \qquad \frac{\Gamma \vdash_{\mathrm{CIC}} t : A \qquad \Gamma \vdash_{\mathrm{CIC}} B : s \qquad A \preceq B}{\Gamma \vdash_{\mathrm{CIC}} t : B} \text{(Conv)}$$

$$\frac{A' \equiv \Pi \vec{x} : \vec{A}.s \qquad \Gamma \vdash_{\mathrm{CIC}} A' : s' \qquad (\Gamma, X : A' \vdash_{\mathrm{CIC}} C_i : s \quad C_i \in Co(X) \quad \forall 1 \le i \le n)}{\Gamma \vdash_{\mathrm{CIC}} \mathrm{Ind}(X : A')\{C_1, ..., C_n\} : A'} \text{(Ind)}$$

$$\frac{\vdash_{\mathrm{CIC}} \mathcal{L} \qquad (\vdash_{\mathrm{CIC}} \mathrm{Ind}(X : A)\{C_1, ..., C_n\} : A, l) \in \mathcal{L} \qquad \Gamma \vdash_{\mathrm{CIC}} I : A \qquad \forall 1 \le i \le n}{\Gamma \vdash_{\mathrm{CIC}} \mathrm{Constr}(i, l) : C_i[I/X]} \text{(Constr)}$$

$$\frac{\begin{array}{c}\vdash_{\mathrm{CIC}} \mathcal{L} \qquad (\vdash_{\mathrm{CIC}} \mathrm{Ind}(X : A)\{C_1, ..., C_n\} : A, l) \in \mathcal{L} \qquad \Gamma \vdash_{\mathrm{CIC}} \vec{d} : \vec{A} \qquad (s, s') \in R_\xi \\ \Gamma \vdash_{\mathrm{CIC}} c : I\ \vec{d} \qquad \Gamma \vdash_{\mathrm{CIC}} Q : (\Pi \vec{x} : \vec{A}.(I\ \vec{x}) \to s') \qquad (\Gamma \vdash_{\mathrm{CIC}} f_i : \xi(l, Q, \mathrm{Constr}(i, l), C_i) \quad \forall 1 \le i \le n)\end{array}}{\Gamma \vdash_{\mathrm{CIC}} \mathrm{Elim}(c, Q)\{f_1, ..., f_n\} : (Q\ \vec{d})c} \text{(Elim)}$$

図 1: pCIC の型付け規則

$$\frac{}{\vdash_{\mathrm{CIC}} []}$$

$$\frac{\vdash_{\mathrm{CIC}} \mathcal{L} \qquad \Gamma_{\mathcal{L}} \vdash_{\mathrm{CIC}} t : A}{\vdash_{\mathrm{CIC}} \mathcal{L} :: (\vdash_{\mathrm{CIC}} t : A, l)}$$

定義する．

$$\begin{aligned}\mathcal{L} &= [] \mid \mathcal{L} :: (\vdash_{\mathrm{CIC}} e : A, l) \\ \Gamma_{[]} &= \emptyset \\ \Gamma_{\mathcal{L}::(\vdash_{\mathrm{CIC}} t:A,l)} &= \Gamma_{\mathcal{L}}, l : A\end{aligned}$$

また，型付け規則における記法を以下で説明する．

1. $x, y, z, ..., X, Y, Z, ...$ は変数，$m, n, ...M, N, ...$ は項，$i, j, ...$ は自然数を表す．
2. $s, s', s'', ...$ は $\mathbb{P}$，$\mathbb{T}_i$ などのソートを表す．
3. $R_\Pi = \{(\_, \mathbb{P}, \mathbb{P}), (\mathbb{T}_i, \mathbb{T}_j, \mathbb{T}_{max(i,j)})\}$
4. $R_\xi = \{(\mathbb{P}, \mathbb{P}), (\mathbb{T}_i, \mathbb{T}_i), (\mathbb{T}_i, \mathbb{P})\}$
5. $\mathrm{Pos}(X)$，コンストラクタの型 $Co(X)$ はそれぞれ以下のように定義される．
   - $\mathrm{Pos}(X) := \Pi \vec{x} : \vec{M}.X\ \vec{m}$
   - $\mathrm{Co}(X) := X\ \vec{m} \mid \mathrm{Pos}(X) \to \mathrm{Co}(X)$ $\mid \Pi x : M.\mathrm{Co}(X)$.

なお，$\preceq$ は変換と累積を表現するものであり，この規則は図 2 のように定義される．

帰納的データ型の例として自然数 nat は pCIC で以下のように表される．

$$\mathrm{nat} \equiv \mathrm{Ind}(X : \mathbb{T}_0)\{X\ , X \to X\} : \mathbb{T}_0$$

また，そのコンストラクタである Zero と Succ は以下のように表される．

$$\mathrm{Zero} \equiv \mathrm{Constr}(1, \mathrm{nat}) : \mathrm{nat}$$
$$\mathrm{Succ} \equiv \mathrm{Constr}(2, \mathrm{nat}) : \mathrm{nat} \to \mathrm{nat}$$

2.2 証明項

Coq と Lean ではカリー・ハワード同型対応に基づき，プログラムとして証明を表現する．カリー・ハワード同型対応とは，形式論理と型理論の対応関係を示すものである．「命題」と「型」，「証明」と「式」が対応していることを表している．Coq である「命題」の「証明」を行うためには，それに対応した「型」を持つ「式」を書けば良いと言える．また，Coq ではこのような証明を表す項のことを**証明項**と呼ぶ．Coq に

$$\frac{}{\mathbb{P} \preceq \mathbb{T}_i}\ \text{(C-Prop)} \qquad \frac{i \leq j}{\mathbb{T}_i \preceq \mathbb{T}_j}\ \text{(C-Type)} \qquad \frac{A \simeq A' \qquad B \simeq B'}{\Pi x : A.B \preceq \Pi x : A'.B'}\ \text{(C-Prod)}$$

$$\frac{A \simeq B}{A \preceq B}\ \text{(C-Conv)} \qquad \frac{A \simeq A' \qquad A' \preceq B' \qquad B \simeq B'}{A \preceq B}\ \text{(C-Congr)}$$

$$I \equiv \mathrm{Ind}(X : \Pi\vec{x} : \vec{N}.s)\{\Pi\vec{x_1} : \vec{M_1}.X/\vec{m_1}, ..., \Pi\vec{x} : \vec{N}.s\Pi\vec{x_n} : \vec{M_n}.X/\vec{m_n}\}$$
$$I' \equiv \mathrm{Ind}(X : \Pi\vec{x} : \vec{N'}.s')\{\Pi\vec{x_1} : \vec{M'_1}.X/\vec{m'_1}, ..., \Pi\vec{x} : \vec{N'}.s'\Pi\vec{x_n} : \vec{M'_n}.X/\vec{m'_n}\}$$
$$s \preceq s' \qquad \forall i.\ N_i \simeq N'_i \qquad \forall i, j.(M_i)_j \preceq (M'_i)_j$$
$$\frac{length(\vec{m}) = length(\vec{x}) \qquad \forall i.\ X\ \vec{m_i} \simeq X\ \vec{m'_i}}{N\ \vec{m} \preceq N'\ \vec{m}}\ \text{(C-Ind)}$$

図 2: 変換と累積の変換規則

ソースコード 1: Lean 上での自然数 (nat) の定義

```
inductive nat : Type
| zero : nat
| succ (n : nat) : nat
```

ソースコード 2: Coq に変換した自然数 (nat) の定義

```
Inductive nat :=
| nat_zero
| nat_succ ( n : nat ).
```

おいて「プログラムを証明する」ということは「プログラムの証明に対応した証明項を構築する」と言い変えることが出来る．

2.3 coq-lean-import の概要

coq-lean-import は Lean 上の定義や定理を Coq の記述に変換する Coq のプラグインである．具体例を交えながら coq-lean-import を使った Lean から Coq への変換について述べる．Lean では自然数 nat はソースコード 1 のように定義できる．coq-lean-import はこれをソースコード 2 のように Coq 上で受けいれられる形に変換する．

Lean のソースコードでは namespace と呼ばれるスコープに分離される．つまり，一度定義した nat を外部で参照する際には，nat.zero や nat.succ のように記述する必要がある．Coq にはこのような機能は存在しないため，ドットの代わりにアンダースコアに置き換えることで対処している．

ソースコード 3: Coq 上で定義した二分木 (btree)

```
Inductive btree (A B : Type) : Type :=
| Leaf : A -> btree A B
| Node : btree A B -> B
          -> btree A B -> btree A B.
```

ソースコード 4: Lean から変換した二分木 (btree0)

```
Inductive btree0 (A B : Type) : Type :=
| btree_Node : btree0 A B -> B
          -> btree0 A B -> btree0 A B.
| btree_Leaf : A -> btree0 A B
```

この nat は Coq 上で記述したそれとは独立しているため，Lean で記述した自然数の性質を Coq 上で再利用することはできない．

例 2.1. 二分木について，Coq ではソースコード 3 のように定義される．Lean 上で定義して coq-lean-import によって Coq 上に移植したものはソースコード 4 に記述される．ここでは葉が型 A を持ち，ノードが型 B を持つ．

これらの型は pCIC では以下のように記述する．

$$\begin{aligned}
&\text{btree} \equiv \text{Ind}(A\ B : \mathbb{T}_0) \\
&\qquad \{A \to X\ , X \to B \to X \to X\} : \mathbb{T}_0 \to \mathbb{T}_0 \\
&\text{btree0} \equiv \text{Ind}(A\ B : \mathbb{T}_0) \\
&\qquad \{X \to B \to X \to X, A \to X\} : \mathbb{T}_0 \to \mathbb{T}_0
\end{aligned}$$

コンストラクタは以下のような形になる．

$$\begin{aligned}
&\text{Leaf} \equiv \text{Constr}(1, \text{btree}) : A \to \text{btree}\ A\ B \\
&\text{Node} \equiv \text{Constr}(2, \text{btree}) : \text{btree}\ A\ B \to\ B \\
&\qquad\qquad \to \text{btree}\ A\ B \to \text{btree}\ A\ B \\
&\text{btree}_{\text{N}}\text{ode} \equiv \text{Constr}(1, \text{btree0}) : \text{btree0}\ A\ B \to B \\
&\qquad\qquad \to \text{btree0}\ A\ B \to \text{btree0}\ A\ B \\
&\text{btree}_{\text{L}}\text{eaf} \equiv \text{Constr}(2, \text{btree0}) : A \to \text{btree0}\ A\ B
\end{aligned}$$

3 アルゴリズム

本章では，帰納型の変換を行うアルゴリズムについて説明する．これは，coq-lean-import によって Coq から変換してきた定義や証明を Coq の対応する型で置き換えることにより，最初から証明を記述し直すことなく，異なる定理証明支援系間での証明の再利用を実現することを目的としている．

以下では，メタ変数を新たに生成する (「′」や添え字と同様の) 記法として既存のメタ変数の右上に • を付ける．例えば，$a^\bullet$，$M^\bullet$，$\Gamma^\bullet$，$a'^\bullet$，$a_i^\bullet$ などであり，これらは • がついていないメタ変数とは独立したメタ変数である．これらのメタ変数は主にアルゴリズムによる変換後の構文の中で用いられる．

3.1 アルゴリズムの入力

本節で導入する帰納型変換アルゴリズムの入力は次のようになる．

$$\begin{aligned}
&\mathcal{L}^L = [(\vdash_{\text{CIC}} t_1 : A_1, l_1), ..., (\vdash_{\text{CIC}} t_n : A_n, l_n)] \\
&\mathcal{L}^C = [(a_1, l'_1), ..., (a_n, l'_n)] \\
&\mathcal{J} \subseteq \{1, ..., n\}
\end{aligned}$$

これらの入力は以下の条件を満たす．

すべての $j \in \mathcal{J}$ について，

$$t_j = \text{Ind}(X : A)\{C_1, ..., C_m\}$$

$j \in \mathcal{J}$ のときは，

$$a_j = \vdash_{\text{CIC}} t'_j : A'_j \wedge t'_j = \text{Ind}(X : A')\{C'_1, ..., C'_m\}$$

$j \notin \mathcal{J}$ のときは，

$$a_j = \_$$

$\mathcal{L}^L$ は coq-lean-import によって Lean から変換してきた型や関数，証明の列であり，$\mathcal{L}^C$ は Coq 上で定義した型や関数, 証明の列である．$\mathcal{J} \subseteq \{1, ..., n\}$ によって，対応する l_j と l'_j を示している．$j \notin \mathcal{J}$ とは Lean にはあるが Coq にはないような関数や証明等が対象となる．

これらの入力をもとに，対応する型について，$\Delta(l_j) = (\sigma_p, \sigma_c, (\sigma_1, ..., \sigma_m))$ を構築する．σ_p，σ_c，$(\sigma_1, ..., \sigma_m)$ は l_j と l'_j 間で対応する型コンストラクタの引数とコンストラクタ，コンストラクタの引数のそれぞれの順番の情報を置換として保持している．

例 3.1. 例 2.1 では，アルゴリズムの入力は以下の通りになる．

$$\begin{aligned}
\mathcal{L}^L =&[(\vdash_{CIC} \text{Ind}(A\ B : \mathbb{T}_0) \\
&\{X \to B \to X \to X,\ A \to X\} : \mathbb{T}_0 \to \mathbb{T}_0, \text{btree0})] \\
\mathcal{L}^C =&[(\vdash_{CIC} \text{Ind}(A\ B : \mathbb{T}_0) \\
&\{A \to X\ , X \to B \to X \to X\} : \mathbb{T}_0 \to \mathbb{T}_0, \text{btree})] \\
\mathcal{J} =&\{1\}
\end{aligned}$$

この場合，Δ の σ_p は型コンストラクタの引数である A, B について，σ_c はコンストラクタ $X, X \to X \to X$ について，そして σ_1, σ_2 は Leaf と Node の引数についてのそれぞれの順番を保持している置換である．次節で説明する変換アルゴリズムを実行すると，σ_p は

$$\begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}$$

σ_c は

$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$$

σ_1 は Coq の 1 つ目のコンストラクタである Leaf で，これは

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

σ_1 は Coq の 2 つ目のコンストラクタである Node で，

これは

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$$

あるいは，

$$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}$$

に決定する．

3.2 帰納型変換アルゴリズムの定義

本論文で提案する帰納型変換アルゴリズムを図 3 に示す．これらのアルゴリズムは前節で説明した条件を満たす証明項を入力として想定している．Δ について導出木を作ることができるものがあるか調べる．前提を全て導出できたときに変換可能として出力される．真偽値 `bool` では，2 つのコンストラクタをそれぞれ入れ替えたとしてもこの変換アルゴリズムでは変換可能と判断されるため，この場合は 2 通りの出力が存在する．

規則 t-Constr1 の $IOcc(l)$ は帰納型 l を再帰的に含むことを示す．また，$Dom(\Delta)$ はアルゴリズムの入力である $\{t_j \mid (\vdash_{\mathrm{CIC}} t_j : A, l_j) \in \mathcal{L}^L, j \in \mathcal{J}\}$ を指す．

この変換アルゴリズムによる命題を以下に示す．

命題 3.1. $\Gamma \vdash t : A \Downarrow_\Delta \Gamma^\bullet \vdash t^\bullet : A^\bullet$ が導出できるとき，$\Gamma \vdash_{\mathrm{CIC}} t : A$ ならば $\Gamma^\bullet \vdash_{\mathrm{CIC}} t^\bullet : A^\bullet$ が成り立つ．

Proof. $\Gamma \vdash t : A \Downarrow_\Delta \Gamma^\bullet \vdash t^\bullet : A^\bullet$ の導出木に関する帰納法を用いて証明する． □

例 3.2. (変換アルゴリズムによる btree の大きさを返す size 関数の変換) アルゴリズムの例として，例 2.1 の btree の大きさを返す size 関数の変換を考える．アルゴリズムの入力は以下の通りになる．

$$\begin{aligned}
\mathcal{L}^L =&[(\vdash_{CIC} \mathrm{Ind}(A\ B : \mathbb{T}_0) \\
&\{X \to B \to X \to X,\ A \to X\} : \mathbb{T}_0 \to \mathbb{T}_0, \mathrm{btree0}), \\
&(\vdash_{CIC} \mathrm{Ind}(X : \mathbb{T}_0)\{X, X \to X\} : \mathbb{T}_0, \mathrm{nat0}), \\
&(\vdash_{CIC} t_{\mathrm{plus0}} : A_{\mathrm{plus0}}, \mathrm{plus0}), \\
&(\vdash_{CIC} t_{\mathrm{size0}} : A_{\mathrm{size0}}, \mathrm{size0})] \\
\mathcal{L}^C =&[(\vdash_{CIC} \mathrm{Ind}(A\ B : \mathbb{T}_0) \\
&\{A \to X\ , X \to B \to X \to X\} : \mathbb{T}_0 \to \mathbb{T}_0, \mathrm{btree}), \\
&(\vdash_{CIC} \mathrm{Ind}(X : \mathbb{T}_0)\{X, X \to X\} : \mathbb{T}_0, \mathrm{nat}), \\
&(\vdash_{CIC} _, \mathrm{plus}), \\
&(\vdash_{CIC} _, \mathrm{size})] \\
\mathcal{J} =&\{1, 2\}
\end{aligned}$$

最初に coq-lean-import によって Lean から自然数の加法を構成する関数を移植した plus0 関数を pCIC で以下のように記述する．

$$\begin{aligned}
t_{\mathrm{plus0}} =&\lambda m : \mathrm{nat0}.\lambda n : \mathrm{nat0}.\mathrm{Elim}(m, \lambda\langle\rangle : 1.\mathrm{nat0}) \\
&\{n, \mathrm{Constr}(2, \mathrm{nat0})\} : \mathrm{nat0} \to \mathrm{nat0} \to \mathrm{nat0}
\end{aligned}$$

図 3 の帰納型変換アルゴリズムによって，btree0 と btree については例 3.1 で示したような置換が得られる．nat0 と nat に対しては，すべて恒等置換である．さらに，図 3 の帰納型変換アルゴリズムによって，t_{plus0} は以下のように plus に変換される．

$$\begin{aligned}
\mathrm{plus} =&\lambda m : \mathrm{nat}.\lambda n : \mathrm{nat}.\mathrm{Elim}(m, \lambda\langle\rangle : 1.\mathrm{nat}) \\
&\{n, \mathrm{Constr}(2, \mathrm{nat})\} : \mathrm{nat} \to \mathrm{nat} \to \mathrm{nat}
\end{aligned}$$

coq-lean-import によって Lean から Coq 上に移植された size0 関数は pCIC では以下のように記述する．

$$\begin{aligned}
t_{\mathrm{size0}} =&\lambda A : \mathbb{T}_0.\lambda B : \mathbb{T}_0.\lambda t : \mathrm{btree0}\ A\ B. \\
&\mathrm{Elim}(t, \lambda\langle\rangle : 1.\mathrm{nat0}) \\
&\{\lambda x_1 : \mathrm{nat0}.\lambda b : B.\lambda x_2 : \mathrm{nat0} \\
&\qquad \mathrm{Constr}(2, \mathrm{nat0})(\mathrm{plus0}\ x_1\ x_2), \\
&\lambda a : A.\mathrm{Constr}(2, \mathrm{nat0})(\mathrm{Constr}(1, \mathrm{nat0}))\} \\
&\qquad : \mathrm{btree0}\ A\ B \to \mathrm{nat0}
\end{aligned}$$

図 3 の帰納型変換アルゴリズムによって，t_{size0} は以下のような size に変換される．

$$\frac{\Gamma \vdash T : s \Downarrow_\Delta \Gamma^\bullet \vdash T^\bullet : s^\bullet \qquad x \notin \Gamma}{\Gamma x : T \vdash \Downarrow_\Delta \Gamma^\bullet x : T^\bullet \vdash} \text{(t-Decl)} \qquad \frac{\mathbb{T}_i \equiv \mathbb{T}_i^\bullet \qquad \mathbb{T}_{i+1} \equiv \mathbb{T}_{i+1}^\bullet \qquad \Gamma \vdash \Downarrow_\Delta \Gamma^\bullet \vdash}{\Gamma \vdash \mathbb{T}_i : \mathbb{T}_{i+1} \Downarrow_\Delta \Gamma^\bullet \vdash \mathbb{T}_i^\bullet : \mathbb{T}_{i+1}^\bullet} \text{(t-Type)}$$

$$\frac{\mathbb{P} \equiv \mathbb{P}^\bullet \qquad \mathbb{T}_i \equiv \mathbb{T}_i^\bullet}{\Gamma \vdash \mathbb{P} : \mathbb{T}_i \Downarrow_\Delta \Gamma^\bullet \vdash \mathbb{P}^\bullet : \mathbb{T}_i^\bullet} \text{(t-Prop)} \qquad \frac{(x : T) \in \Gamma \qquad (x : T^\bullet) \in \Gamma^\bullet \qquad \Gamma \vdash \Downarrow_\Delta \Gamma^\bullet \vdash}{\Gamma \vdash x : T \Downarrow_\Delta \Gamma^\bullet \vdash x : T^\bullet} \text{(t-Var)}$$

$$\frac{(\vdash_{\text{CIC}} t : T, l) \in \mathcal{L} \qquad (\vdash_{\text{CIC}} t^\bullet : T^\bullet, l^\bullet) \in \mathcal{L}^\bullet \qquad \Sigma \vdash t : T \Downarrow_\Delta \Sigma^\bullet \vdash t^\bullet : T^\bullet \qquad \Gamma \vdash \Downarrow_\Delta \Gamma^\bullet \vdash}{\Gamma \vdash l : T \Downarrow_\Delta \Gamma^\bullet \vdash l^\bullet : T^\bullet} \text{(t-VarL)}$$

$$\frac{\Gamma \vdash t : \Pi x : A.B \Downarrow_\Delta \Gamma^\bullet \vdash t^\bullet : \Pi x : A^\bullet.B^\bullet \qquad \Gamma \vdash t' : A \Downarrow_\Delta \Gamma^\bullet \vdash t'^\bullet : A^\bullet}{\Gamma \vdash (t\ t') : B \Downarrow_\Delta \Gamma^\bullet \vdash (t^\bullet\ t'^\bullet) : B^\bullet} \text{(t-App)}$$

$$\frac{\Gamma \vdash A : s \Downarrow_\Delta \Gamma^\bullet \vdash A^\bullet : s^\bullet \qquad \Gamma, x : A \vdash B : s' \Downarrow_\Delta \Gamma^\bullet, x : A^\bullet \vdash B^\bullet : s'^\bullet \qquad (s, s', s'') \in R_\Pi}{\Gamma \vdash \Pi x : A.B : s'' \Downarrow_\Delta \Gamma^\bullet \vdash \Pi x : A^\bullet. : B^\bullet s''^\bullet} \text{(t-Prod)}$$

$$\frac{\Gamma, x : A \vdash t : B \Downarrow_\Delta \Gamma^\bullet, x : A^\bullet \vdash t^\bullet : B^\bullet}{\Gamma \vdash (\lambda x : A.\ t) : (\Pi x : A.B) \Downarrow_\Delta \Gamma^\bullet, \Gamma_0' \vdash (\lambda x : A^\bullet.\ t^\bullet) : (\Pi x : A^\bullet.B^\bullet)} \text{(t-Lam)}$$

$$\frac{\Gamma \vdash t : A \Downarrow_\Delta \Gamma^\bullet \vdash t^\bullet : A^\bullet \qquad \Gamma \vdash B : s \Downarrow_\Delta \Gamma^\bullet \vdash B^\bullet : s^\bullet \qquad A \preceq B}{\Gamma \vdash t : B \Downarrow_\Delta \Gamma^\bullet \vdash t^\bullet : B^\bullet} \text{(t-Conv)}$$

$$\frac{\begin{array}{c} A' \equiv \Pi \vec{x} : \vec{A}.s \quad \Gamma \vdash A' : s' \Downarrow_\Delta \Gamma^\bullet \vdash A'^\bullet : s'^\bullet \\ \Gamma, X : A' \vdash C_i : s \Downarrow_\Delta \Gamma^\bullet, X : A'^\bullet \vdash C_i^\bullet : s^\bullet \quad C_i \in Co(X) \quad \forall 1 \le i \le n \end{array}}{\Gamma \vdash \text{Ind}(X : A')\{C_1, ..., C_n\} : A' \Downarrow_\Delta \Gamma^\bullet \vdash \text{Ind}(X : A'^\bullet)\{C_1^\bullet, ..., C_n^\bullet\} : A'^\bullet} \text{(t-Ind)}$$

$$\frac{\begin{array}{c} IOcc(l) \cap Dom(\Delta) = \emptyset \\ \vdash_{\text{CIC}} \mathcal{L} \quad (\vdash_{\text{CIC}} \text{Ind}(X : A)\{C_1, ..., C_n\} : A, l) \in \mathcal{L} \quad \Gamma \vdash_{\text{CIC}} I : A \quad \forall 1 \le i \le n \end{array}}{\Gamma \vdash \text{Constr}(i, l) : C_i[I/X] \Downarrow \Gamma \vdash \text{Constr}(i, l) : C_i[I/X]} \text{(t-Constr1)}$$

$$\frac{\begin{array}{c} \Delta(l) = (\sigma_p, \sigma_c, (\sigma_1, ..., \sigma_n)) \\ \vdash_{\text{CIC}} \mathcal{L}^L \quad (\vdash_{\text{CIC}} \text{Ind}(X : A)\{C_1, ..., C_n\} : A, l) \in \mathcal{L}^L \\ \vdash_{\text{CIC}} \mathcal{L}^C \quad (\vdash_{\text{CIC}} \text{Ind}(X : A^\bullet)\{C_1^\bullet, ..., C_n^\bullet\} : A^\bullet, l^\bullet) \in \mathcal{L}^C \end{array}}{\Gamma \vdash \text{Constr}(i, l) : C_i[I/X] \Downarrow_\Delta \Gamma^\bullet \vdash \lambda x_1 ... x_m.\text{Constr}(C_{\sigma_c(i)}, l^\bullet) x_{\sigma_i(1)} ... x_{\sigma_i(m)} : C_i^\bullet[I/X]} \text{(t-Constr2)}$$

$$\frac{\begin{array}{c} \vdash_{\text{CIC}} \mathcal{L}^L \quad (\vdash_{\text{CIC}} \text{Ind}(X : A)\{C_1, ..., C_n\} : A, l) \in \mathcal{L}^L \\ \vdash_{\text{CIC}} \mathcal{L}^C \quad (\vdash_{\text{CIC}} \text{Ind}(X : A^\bullet)\{C_1^\bullet, ..., C_n^\bullet\} : A^\bullet, l^\bullet) \in \mathcal{L}^C \\ \Gamma \vdash_{\text{CIC}} \vec{a} : \vec{A} \quad (s, s') \in R_\xi \quad \Gamma \vdash_{\text{CIC}} c : I\ \vec{a} \\ \Gamma \vdash Q : (\Pi \vec{x} : \vec{A}.(I\ \vec{x}) \to s') \Downarrow_\Delta \Gamma^\bullet \vdash Q^\bullet : (\Pi \vec{x^\bullet} : \vec{A^\bullet}.(I^\bullet\ \vec{x^\bullet}) \to s'^\bullet) \\ \Gamma \vdash f_i : \xi(l, Q, \text{Constr}(i, l), C_i) \Downarrow_\Delta \Gamma^\bullet \vdash f_i^\bullet : \xi(l^\bullet, Q^\bullet, \text{Constr}(i, l^\bullet), C_i^\bullet) \quad \forall 1 \le i \le n \end{array}}{\Gamma \vdash \text{Elim}(c, Q)\{f_1, ..., f_n\} : (Q\ \vec{a})c \Downarrow_\Delta \Gamma^\bullet \vdash \text{Elim}(c^\bullet, Q^\bullet)\{f_1^\bullet, ..., f_n^\bullet\} : (Q^\bullet\ \vec{a})c^\bullet} \text{(t-Elim)}$$

図 3: 帰納型変換アルゴリズム

```
size =λA : 𝕋0.λB : 𝕋0.λt : btree A B.
      Elim(t, λ⟨⟩ : 1.nat)
      {λa : A.Constr(2, nat)(Constr(1, nat)),
       λx1 : nat.λb : B.λx2 : nat
            Constr(2, nat)(plus x1 x2)}
                  : btree A B → nat
```

4 関連研究

本章では，Coq で既に提供されている別の変換プラグインについて紹介する．Ringer によって開発された DEVOID [7] や Proof Repair [6] はオーナメントと

ソースコード 5: 自然数 (nat) の定義

```
Inductive nat : Type :=
| O : nat
| S : nat -> nat.
```

ソースコード 6: リスト (list) の定義

```
Inductive list (A : Type): Type :=
| nil : list A
| cons : A -> list A -> list A.
```

図 4: オーナメントによって関係付けられる型

呼ばれるものを利用した証明項ツールである．オーナメントは McBride によって開発された，ある帰納的データ型とそれと類似している帰納的な構造を持つ，より情報量が多い帰納的データ型との関係性を表現するものである．オーナメントによって関係性が表現可能な帰納的データ型の例として自然数を表す nat 型とリストを表す list 型を Coq 上で表現すると図 4 のようになる．

2 つの型は別の意味を持つが，類似している点が見られる．nat と list を比較すると，O と nil，nat と list A，S と cons のように nat の各要素は必ず list の要素に対応していることが分かる．また，帰納的な構造も保たれている．オーナメントはこのような対応関係を明示している．また，オーナメントは A ->の部分のような対応していない部分に「飾り」を付けることにより，型の拡張部分を明示する役割もある．これらの役割により，オーナメントはある帰納的データ型と，より情報量の多い元の型を拡張したような帰納的データ型との関係性を表現している．

DEVOID や Proof Repair は同型である型の変換に対応した証明項の自動的な変換を実現している．1 対 1 対応を持つ，オーナメントによって関係付けられた型のペアを入力すると，それらの型を関係付けるオーナメントを自動的に推論し，そのオーナメントに基づき証明変換を行う．

本論文においては，類似している型ではなく，Coq と Lean の間で対応している型を対象としており，それらのコンストラクタや引数の対応付けに基づいて定理や証明の変換を行うアルゴリズムを提案した．

5 まとめと今後の課題

本論文では，定理証明支援系間での証明の再利用を目的として，Coq と Lean で対応する型について，コンストラクタや引数を入れ替えることで変換を行うアルゴリズムを提案した．そして，これに基づいて関数や証明の変換についても考えることができた．

今後の課題としては，本論文で提案した変換アルゴリズムを全て網羅した Coq のプラグインの実装をすることが求められる．また，より複雑な証明に関しても変換が可能であるか検証する必要がある．

謝辞 本研究の一部は JSPS 科研費 JP21K11744，JP19K11891，22H00520 の助成を受けて行われた．

参考文献

[1] The Agda Wiki. https://wiki.portal.chalmers.se/agda/pmwiki.php/.
[2] The Coq Proof Assistant. https://coq.inria.fr/.
[3] Isabelle. https://isabelle.in.tum.de/.
[4] Lean. https://leanprover.github.io/.
[5] Gilbert, G.: Alpha Announcement: Coq is a Lean Typechecker. https://coq.discourse.group/t/alpha-announcement-coq-is-a-lean-typechecker/581.
[6] Ringer, T., Porter, R., Yazdani, N., Leo, J., and Grossman, D.: Proof repair across type equivalences, *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2021, pp. 112–127.
[7] Ringer, T., Yazdani, N., Leo, J., and Grossman, D.: Ornaments for Proof Reuse in Coq, *10th International Conference on Interactive Theorem Proving (ITP 2019)*, Vol. 141, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019, pp. 26:1–26:19.
[8] Timany, A. and Jacobs, B.: First Steps Towards Cumulative Inductive Types in CIC, *Theoretical Aspects of Computing - ICTAC 2015*, Leucker, M., Rueda, C., and Valencia, F. D.(eds.), Cham, Springer International Publishing, 2015, pp. 608–617.