進捗ゼミ 2025/10/30

Coqでの型クラスを用いた抽象化・定理証明の実装

情報科学研究科 2年　佐藤龍之介

発表の流れ

1. Introduction
2. 型クラスについて
3. 実装の概要
4. 型クラスインスタンスと定理の応用
5. まとめ

Introduction

研究目的

・OTにおける合流性C_1($op_1 \cdot T(op_2, op_1) = op_2 \cdot T(op_1, op_2)$)

C_2の証明をRocqで行い、4ノード以上の合流性を証明する[参考文献1]

方針

・型クラスOTBase[2]を用いて**型list A**と**Op A**についての命題を用意し、

これらの型におけるOTのC_1を保証する型クラスのインスタンスを作成する

型クラスについて（ Eq型クラス）

型クラスとは、ユーザがインスタンスを宣言することで自動的にインスタンス内で指定された型から実装における型を推論して補う仕組み

・nat型の値に対して等価性判定を行うインスタンスを定義した
・同様にbool型の値に対して等価性判定を行うインスタンスを定義した

→ **Eq型クラスによって、等価性判定を行う値の型を抽象化し、柔軟なインスタンスの定義が可能になった**

```
Class Eq (A : Type) :=
 {eqb : A -> A -> bool}.

Instance eqNat : Eq nat :=
{eqb x y := Nat.eqb}.

Instance eqBool : Eq bool :=
 {eqb x y := Bool.eqb}.
```

実装の概要

- 関数insert, delete
- 型コンストラクタOp と型Op A
- 関数interp_op
- 関数inv_op
- 関数it_sc
- OTBase型クラス
- 合流性C1の補題ot_convergence_property_c1

関数insert, delete

・p番目に型Aの値を挿入する操作の**関数insert**

・p番目の型Aの値を削除する操作の**関数delete**

```
Fixpoint insert [A : Type] (p : nat) (x : A) (l : list A) :
option (list A) :=
 match p, l with
 | 0, _ => Some (x :: l)
 | S p', [] => None
 | S p', h :: t =>
    match insert p' x t with
    | None => None
    | Some l' => Some ( h :: l')
    end
end.
```

```
Fixpoint delete [A : Type] (p : nat) (x : A) (l : list A) `{Eq A} : option (list
A) :=
 match p, l with
 | 0, h :: t => if eqb h x then Some t else None
 | 0, [] => None
 | S p', h :: t =>
    match delete p' x t with
    | Some t'  => Some (h :: t')
    | None  => None
    end
 | S p', [] => None
end.
```

型Op Aと関数interp_op

・p番目に型Aの値を挿入する操作の **関数insert**

・p番目の型Aの値を削除する操作の **関数delete**

以上に後々対応づける為の **型OpInsとOpDel**を用意した

```
Inductive Op (A : Type) :=
| OpIns : nat -> A ->  Op A
| OpDel : nat -> A -> Op A.
```

・型Op Aである操作opに対するパターンマッチより操作の型OpInsとOpDelに対応する、
操作の関数をリスト lに適用した結果 を返す関数

```
Definition interp_op [A : Type] `{Eq A}
 (op : Op A) (l : list A) : option (list A) :=
 match op with
 | OpIns p x => insert p x l
 | OpDel p x => delete p x l
 end.
```

関数inv_op

・型Op Aである操作opに対するパターンマッチより操作の型OpInsとOpDelに対して、それぞれの逆の操作関数を返す
操作の関数をリスト lに適用した結果 を返す

```
Definition inv_op [A : Type] `{Eq A} (op : Op A) : Op A :=
 match op with
 | OpIns p x  => OpDel p x
 | OpDel p x => OpIns p x
 end.
```

例. リストlの1番目の要素に2を挿入する関数の逆の操作である、1番目の要素の2を削除する関数を返す

```
Compute inv_op (OpIns 1 2).
Messages（結果）
    = OpDel 1 2

    : Op nat
```

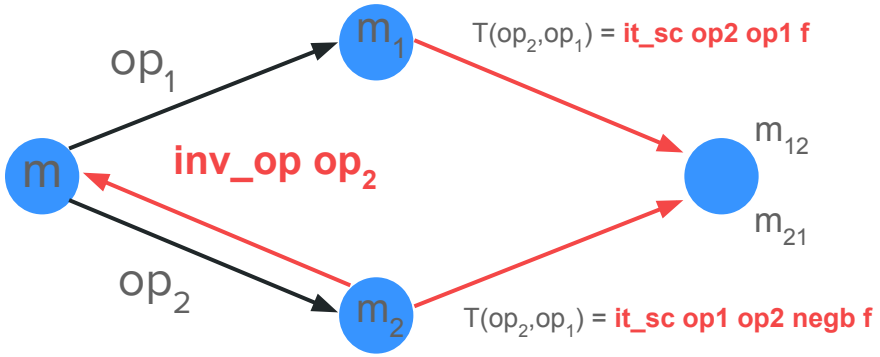
関数it_sc

fが必要な理由

op1 = ins “a” 0,op2 = ins “b” 0とすると、
op_1->op_2 …m_{12}= ba
op_2->op_1 … ,m_{21} =ab
最後の状態が一致しない為、どちらかの操作を優先する**フラグ f**を導入する

```
Definition it_sc {A : Type} `{Eq A} (op1 op2 : Op A)
(f : bool) : list(Op A) :=
 if f then []
else [inv_op op2 ; op1].
```

f=true の場合、クライアント優先
f=false の場合、サーバ優先であり優先度の低いクライアントの操作を逆関数inv_opを用いて戻し優先される操作を適用する



OTBase型クラス

・操作の適用（interp）

・操作変換（it）

・抽象的な型Xとcmdについて
合流性C_1が成立すること（it_c1）

を記述する土台

```
Class OTBase (X cmd : Type) := {
interp : cmd -> X -> option X;
it     : cmd -> cmd -> bool -> list cmd;
it_c1 : forall (op1 op2 : cmd) (f : bool) m (m1 m2 : X),
 interp op1 m = Some m1 -> interp op2 m = Some m2 ->
 let m21 := (exec_all interp) (Some m2) (it op1 op2 f) in
  let m12 := (exec_all interp) (Some m1) (it op2 op1 (~~f)) in
   m21 = m12 /\ exists node, m21 = Some node}.
```

合流性C1についての補題

OTBase型クラスを用いて、2つの型list A と Op Aについての命題を記述した

```
Lemma ot_convergence_property_c1 (A : Type) `{Eq A} :
 forall (op1 op2 : Op A) (f : bool) m (m1 m2 : list A),
   interp_op op1 m = Some m1 ->
   interp_op op2 m = Some m2 ->
   let m21 := (exec_all interp_op) (Some m2) (it_sc op1 op2 f) in
   let m12 := (exec_all interp_op) (Some m1) (it_sc op2 op1 (negb f)) in
     m21 = m12 /\ exists node, m21 = Some node.
Proof.
Admitted.
```

型クラスインスタンスと定理の応用

型クラスインスタンス内で関数（interp_op、it_sc）を宣言することで型list AとOp Aが束縛される。加えて、示した補題をインスタンス内に含めることで操作などの関数だけではなく定理や性質C_1が成立することを特定の型について示すことができる[3]

インスタンスOTBaseListSC の宣言

・型制約 `{Eq A}
　-> 関数delete内で削除対象の値とp番目の値の等価性判定に必要
・暗黙引数の明示 @ [4]
　-> 型推論のエラーの解消
・it_c1を補題で示したc1を用いて局所関数で記述
　-> 補題をそのまま与えると型推論のエラーが起きる為、明示的に型を割り当てる必要がある[4]

```
Instance OTBaseListSC {A : Type} `{Eq A} :
OTBase (list A) (Op A) :=
{
 interp := @interp_op A _;
 it := @it_sc A _;
 it_c1 := fun op1 op2 f m m1 m2 =>
        ot_convergence_property_c1 A op1 op2 f m
m1 m2;
}.
```

疑問（関数it_sc）

```
Lemma ot_convergence_property_c1 (A : Type) `{Eq A} :
 forall (op1 op2 : Op A) (f : bool) m (m1 m2 : list A),
   interp_op op1 m = Some m1 ->
   interp_op op2 m = Some m2 ->
   let m21 := (exec_all interp_op) (Some m2) (it_sc op1 op2 f) in
   let m12 := (exec_all interp_op) (Some m1) (it_sc op2 op1 (negb f)) in
     m21 = m12 /\ exists node, m21 = Some node.
Proof.
 intros.
 split.
 unfold m21, m12.
 unfold it_sc.
 unfold inv_op.
 destruct f.
   simpl.
Admitted.
```

```
現在のゴール（1/3）
Some m2 =
          Basics.flip (fun c : Op A => Commons.bind (interp_op c))
          (Basics.flip (fun c : Op A => Commons.bind (interp_op c))
          (Some m1) match op1 with
                      | OpIns p x => OpDel p x
                      | OpDel p x => OpIns p x
                       end) op2
```

```
現在のゴール（2/3）
exec_all interp_op (Some m2) [match op2 with
| OpIns p x => OpDel p x
| OpDel p x => OpIns p x
end; op1] =
exec_all interp_op (Some m1) (if negb false then [] else
[match op1 with
| OpIns p x => OpDel p x
| OpDel p x => OpIns p x
end; op2])
```

```
現在のゴール（3/3）
exists node : list A, m21 = Some node.
```

まとめ

・型クラスを使うことで短くて抽象度が高い命題の記述をすることができる為、特定の型についての命題をインスタンスとして表すことができ、インスタンス内で操作や性質をまとめて保証できること

・加えて、型クラスを土台とすることで特定の型についてのインスタンスを柔軟に用意できる為、実装が簡単になる

参考文献

1. https://www.arxiv.org/pdf/2409.09934 (p8)
2. https://github.com/JetBrains/ot-coq/blob/master/OtDef.v (l36-43)
3. 型クラス — Coq 8.8.2 ドキュメント
4. Implicit arguments — Coq 8.18.0 documentation
   （Deactivation of implicit arguments for parsing）

・Verified operational transformation for trees