Coqにおける集合論的直観主義数学の実装

名古屋大学多元数理科学研究科

佐藤雅大

Coqとは・・・？

- 定理証明支援ツールの1つ
  - 他にもIsabelle,Agda,Mizar,...
- 定理を入力し、証明をコマンドで打ち込んでゆく
  - つまり、コンピューターを使って定理を証明する！！
  - 対話的に定理を証明できる
  - 証明が正しいことを確認することができる
- 型付きλ計算を利用
- 直観主義論理

目的

- 数学をCoqで構築する試みはたくさんある
- 集合論の上から数学を構築する
  - “自然数論だけ”というような特定の分野のみの実装を目的としない
  - 集合論がしっかりしていれば、そのあとの数学も忠実に実現できる？
  - “微分幾何学”などの、様々な分野と関わりのある分野を“集合論”という同じ土俵の上で実装したい
  - 集合論を実装する試みはあるけど、その集合論の上から数学を構築する試みはあまり見られない！！！
- できるだけ数学者の感覚に近いような記述をする
  - Coqの型理論を知らない人にも使いやすい形に
- 弱い論理体系を構築
  - 論理体系を強くすることはAxiom文を用いることにより容易に可能
  - 論理体系を弱くするためには実装を見直す他はない

直観主義論理の集合論

- ZFC = ZF + C
  - Zermelo–Fraenkel + Axiom of Choice
- IZF(直観主義的集合論)
  - 直観主義論理のためのZF
  - 選択公理から排中立が導かれるため、一般にこれを認めない
    - ただし弱い形の選択公理ならば認める場合もある
- CZF(構成的集合論)
  - IZFをさらに弱くしたもの
    - 冪集合の公理の排除(変わりにSubset Axiomを)
    - 分出公理$\{x \in A \mid P(x)\}$の制限
      - **P(x)の中に束縛されていない量化子を含めない(Δ0式のみを許す)**

集合論の実装

集合論の実装で代表的なものに以下がある

- B.Werner(CZF + ΣΠI-AC + REA + ...)[3,4,5]
  - P.AczelがCZFのモデルをType Theoryの上で構築
  - その結果を元にB.Wernerが実装
- B.Barras(IZF)[6]
  - Listに類似したデータ構造を定義
- C.Simpson(ZFC)[7]
  - 型と項を同一視する公理を加えることにより集合論的な性質を作り出す
- G.Alexandre(ZFC)[User's contributeより]
  - 各ZFCの公理と、その集合の存在公理をAxiom文で加える

方法

- 今回の実装はG.Alexandre流
- IZF(直感主義的集合論)で数学を実装
- CZF(構成的集合論)は未実装

  →いかにCoqにΔ0式を識別させるか

IZFの公理(ZFからの公理)

- Extensionality

  $\forall A \forall B[[\forall x, x \in A \Leftrightarrow x \in B] \Rightarrow A=B]$

- Empty ・・・空集合

  $\exists x \forall y, \sim y \in x$

- Paring ・・・二つの元を持つ集合

  $\forall a \forall b \exists c[\forall x, x \in c \Leftrightarrow x=a \vee x=b]$

- Union ・・・和集合

  $\forall a \exists b[\forall x, x \in b \Leftrightarrow \exists u, u \in a \wedge u \in x]$

- Powerset ・・・冪集合

  $\forall a \exists b[\forall x, x \in b \Leftrightarrow x \subset a]$

- Infinity ・・・自然数全体からなる集合

  $\exists N, \Phi \in N \wedge [\forall n, n \in N \Rightarrow n \cup \{n\} \in N]$

- Separation ・・・{x∈A | P(x)}という集合
  - $\forall A \exists S[\forall x, x \in S \Leftrightarrow P(x) \wedge x \in A]$

IZFの公理たち(Set Induction)

Foundationの代わりにSet Inductionを公理として採用

- Set Induction

  $[\forall a[\forall x, x\in a \Rightarrow P(x)] \Rightarrow P(a)] \Rightarrow \forall a, P(a)$

- Foundation

  $\forall a[\exists x[x\in a] \Rightarrow \exists x\in a \forall y\in a[\sim y\in x]]$

  →(“∈”に関する無限降下列は存在しない)

Foundationの方が公理としては強いが、
Foundationからは排中律が導かれてしまう！！

IZFの公理たち(Collection Scheme)

Replacementの代わりにCollectionを公理として採用

- Collection
  - $\forall x \in a, \exists y, P(x,y) \Rightarrow \exists b \forall x \in a \exists y \in b, P(x,y)$

- Replacement

  $\forall a,b,c[P(a,b) \wedge P(a,c) \Rightarrow b=c] \Rightarrow$
  - $\forall D \exists R \forall r[r \in R \Leftrightarrow \exists d \in D, P(d,r)]$

Collectionの方が公理として強い

実装

1.IZFの述語記号は'='と'∈'のみ

```
Axiom SET : Type.
Axiom In : SET → SET → Prop.
```

2.IZFの9つの公理を記述

```
Axiom EmptyAxiom : exist empty, forall x, ~In y empty.
Axiom UnorderdPairAxiom : forall a b, exists X, (forall c , In c X ↔ (c=a \/ c=b)).
                                    ⋮
```

3.定義関数'UniqueOut'の記述

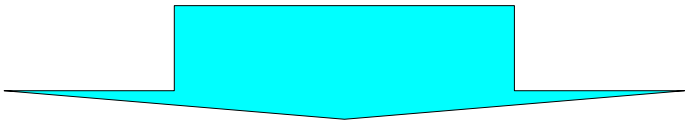
```
Definition Unique (P : SET → Prop) := (exists x, P x) /\ (forall x y, P x /\ P y → x = y).
Axiom UniqueOut (P : SET → Prop) : Unique P → SET.
Axiom HUniqueOut : forall (P : SET → Prop) (Unq : Unique P), P (UniqueOut Unq P).
```

定義関数を使うことの利点

- ∀x,∃!y,P(x,y)ならば、あるf(x)でP(x,f(x))となるものが存在する。このfを“Pの定義関数”と呼ぶ
  - 例:任意の集合a,bに対して、a,bのみを含む集合が存在する。これを{a,b}と書く

G.Alexandreの手法では・・・

```
Variable paire : E → E → E.
Axiom axs_paire : forall v0 v1 v3 : E, In v3 (pair v0 v1) ↔ v3 = v0 \/ v3 = v1.
```



独自の手法(定義関数を用いた手法)では・・・

```
Definition IsPair a b X := forall c , In c X ↔ (c=a \/ c=b).
Axiom PairAxiom : forall a b, exists X, IsPair a b X.
Theorem UniquePair : forall a b, Unique (fun X => IsPair a b X).
Definition Pair a b := UniqueOut (fun X => IsPair a b X) (UniquePair a b)
```

- **一見、複雑になったように見えるがVariableやAxiomを一つ減らすことができた！**
- **定義関数は、後で述べる写像の定義にも使うことができる**

集合から数学を展開するにあたって

- 既存の集合論の実装の上に、さらに工夫が必要

  →なぜなら、数学は“構造”を扱う学問

  - 例：写像・関係・群構造・位相構造・多様体...etc
- 例えば、写像N → Nを考えたいとき、部分写像が必要
  - SET→SETは作れるが、それを{0,1,2,3,・・・}(=N)というSETの部分型に制限したい
    - また、写像自体も集合、つまりSETを型とする項

Coercion(暗黙の型変換)

```
Definition bool_in_nat (b : bool) : nat :=
      if b then 1 else 0.
Coercion bool_in_nat : bool >-> nat.
```

これによりbool型の項をnat型の項に暗黙に型変換できる

例：

```
Check true.
      true : bool.
Check true + 1.
      true + 1 : nat.
Eval compute in true + 1.
      2 : nat.
```

bool型であるtrueがbool_in_nat関数を通してnat型に変換された

SubTyping1

- CoqにはSubTypingがない
  - でもSubTyping“みたいな”ものがある！！

    {x : A | P x} は、型Aの項aで述語P(a)を満たすものからなる型

    注:{x : A | P x}はCoqのNotationで定義されたもの(C言語におけるマクロ)

```
Inductive sig {A : Type} (P : A -> Prop) : Type :=
  exist : forall x : A, P x -> sig P.
Notation “{x : A | P x}” := sig A P.
```

※以下では{x : SET | x∈A}を “#A”と表す

```
Definition partial_SET (A : SET) : nat := {x : SET | x∈A}.
Notation ”#A” := partial_SET A.
```

直観的には・・・

a : #A ⇔ a ∈ A

SubTyping2

- (#A)型とSET型を同等に扱いたい
  - #A→SETへのCoercionを使う！！

```
Definition partial_SET_to_SET {A : SET} : (partial_SET A) → SET := ...
Coercion partial_SET_to_SET : partial_SET >-> SET.
```

これにより、ある項nに対して
n : #N , n : SET
を(仮想的に)同時に満たすことが可能！！

- 等号“==”の定義

```
Inductive IZFEq (a b : SET) : Prop :=  (a=b).
Notation “a == b” := IZFEq a b.
```

等号比較をする際,
CoercionによりSET型に型変換してから
等号比較してくれる！！

アップキャスト・ダウンキャスト

```
Definition DownCast {A} x (prf : x∈A) : #A := …
Notation ”{x ! prf}” := DownCast x prf.

Definition UpCast {A B} (sub : A ⊂ B) (a : #A): #B := …
Notation ”a{<sub}” := UpCast sub a.
```

- A⊂B で a:#A b:#Bのとき
  - a{<sub}は型#Bをもつ

    →ここsubは命題A⊂Bの証明
  - {b ! prf}は型#Aをもつ

    →ここでprfは命題b∈Aの証明

関係演算子

- 集合AとBに対してA×Bの冪集合の元をA,B上の“関係演算子”と呼ぶ

```
Definition Relation (A B : SET) : SET := PowerSet (Cartesian A B).

Definition If {A B : SET} (rel : #Relation A B) (a : #A) (b : #B) : Prop :=
  In (Pair a b) rel.

Notation ”&&rel” := If rel.
```

例：
x y : #N**として(**Nは**自然数全体の集合)**
gt:#(Relation N N)**を関係演算子「<」**
とすると、x<yは

- &&gt x y

と記述する

Relation A Bは
A,B**上の関係**
全体からなる集合

gtは#(Relation N N)**型であるが**&&**を付けることにより**#N→#N→Prop**に変換**

写像

- A,B上の関係演算子relに対して
  - ∀a : #A, ∃!b : #B, &&rel a b
- が成り立つときrelをAからBへの写像という

```
Definition Map (A B : SET) : SET := ....

Definition App {A B : SET} (F : #(Map D R)) (a : #A) : #B :=
  UniqueOut (fun r : #R => &&(rel{<Map_Rel}) a b).

Notation "% F" := App F.
```

例：
x : #Nとして(Nは自然数全体の集合)
f : #(Map N N)とするとf(x)は
　%f x
と記述する

Map A Bは
AからBへの写像
全体からなる集合

fは#(Map N N)型であるが%を付けることにより#N→#Nに変換

他にも・・・

- 群のとき
  - 「A上の群」全体からなる集合を考える
    - たとえば、それを“Group A”とおく。

      →“Group A”は、集合A上の群の公理を満たす演算子全体の集合となる
  - 「群ならば半群である」という命題を表現したいときは
    - (Group A) ⊂ (Semigroup A)

      →ここで“Semigroup A”とは,A上の半群の公理を満たす演算子全体からなる集合
  - **群Gの単位元がeである**
    - IsIdentという関係演算子を定義！！
      - &&IsIdent G e → 群Gの単位元がeである

“構造”を扱いたい時は
それを満たす集合全体を考える！！

他にも・・・

**数学において“構造”は、
複数の“構造”の組になっていることが多い**

- 環のとき
  - 「o1を加法,o2を乗法としたときo1,o2は環となる」
    - &&Ring o1 o2 のように、Ringという関係演算子を定義
- 線形空間
  - 「Kを体,Vをアーベル群とすると、K,Vは線形空間となる」
- 多様体
  - 「Mを位相空間,(U,V)を座標近傍系とすると・・・」

**このようなときは、
その関係演算子を上手く用いる！！**

結果

- CoercionやNotationなどを上手く利用して、数学者の見やすい形に記述することができた
- 様々な数学の分野を独自の手法を用いずに“集合論の上で”そのまま実装できた
  - 基本的な群・環の定義ができた
    - Coqの上での代数的な計算
  - 自然数から整数の実装ができた
  - 有理数体の定義ができた(有理数の定義はまだ未完成)

展望

- 実装を進めていく
  - 微分幾何学の実装？
  - CZFでの構築
- どこまで抽象化してから実装するか
  - 例: 極限 → filter, Lie群 → 位相群, 多様体 → orbifold...
- 新しく加えた公理が、Coqの型理論の中で(直観主義的に)矛盾を引き起こさないか

  →それらの公理が無矛盾であることを証明する必要がある
- しかしCZFという限られた論理体系でどこまで数学が展開できるか

  →構成的数学

ソースファイル

- URL
  - http://proofcafe.org/k27c8/IMath.tar.gz
- ファイル構成
  - logic.v ・・・ 基本的な処理
  - IZF.v ・・・ 集合論の公理
  - Relation.v ・・・ 関係演算子や写像などの定義
  - BaseMap.v ・・・ 基本的な写像のライブラリ
  - Maps.v ・・・ BaseMap.vをいくつか組み合わせて作った写像ライブラリ
  - TransitiveRecursiveFunction.v ・・・ 推移的帰納関数の存在の証明
  - BOperation1.v ・・・ 半群・群などの定義
  - BOperation2.v ・・・ 環などの定義
  - AddMultPower.v ・・・ 自然数の和・積
  - Integer.v ・・・ 整数

BaseMap内で定義した関数の例

- RightPair, LeftPair・・・２変数の射影関数
- CombineRelation・・・２つの関係演算子を合成したものを返す
- RistrictRelation・・・関係演算子の制限
- Currying・・・A×B→CをA→(B→C)に変換
- EqCannProj・・・同値類への自然な写像
- RelationImage・・・関係演算子から冪写像を
  - $b \in (\mathrm{RelationImage}(R))\ (S) \Leftrightarrow \exists a \in S,\ aRb$

Maps内で定義した関数の例

- IdMap・・・恒等関数(RightPairをカリー化)
- ConstMap・・・定数関数(LeftPairをカリー化)
- CombineMap・・・写像の合成
- CartesianDMap・・・(A→B)×(A→C)→(A→(B×C))
- MapImage・・・冪写像(RelationImage)
  - (MapImage(f)) (X) := f[X] (X⊂Dom(f))
- InversePair・・・ペアの入れ替え
  - InversePair(a,b)=(b,a)
- InverseRel・・・逆関係(InversePairとMapImageを)
- InverseMap・・・全単射の集合全体を定義域とした。写像を逆写像へ写す

参考文献

- [1]P.Aczel, M. Rathjen, “*Note on Constructive Set theory(DRAFT)*”, 2008
- [2]Benjamin C.Pierce, “*Types and Programming Languages*”, 2002, “The MIT Press”
- [3] P.Aczel, “*The type theoretic interpretation of constructive set theory*”, 1977, “Logic Colloquium '77”
- [4] P.Aczel, “*The type theoretic interpretation of constructive set theory:Choice principles*”, 1982, “The L.E.J. Brower Centenary Symposium”
- [5] P.Aczel, “*The type theoretic interpretation of constructive set theory:Inductive definitions*”, 1986, “Logic, Methodology and Philosophy of Science VII”
- [6]B.Barras, ”Sets in Coq, Coq in Sets”, 2010, “Journal of Formalized Reasoning”
- [7]C.Simpson, ”Set-theoretical mathematics in Coq”, 2004

ご静聴ありがとうございました。