定性空間表現PLCAのモデル化と妥当性の証明について

後藤 瑞貴　高橋 和子

定性空間推論の枠組みの１つである PLCA 表現の妥当性を検証する．PLCA 表現は点，線，閉路，範囲を，位置や大きさを指定しない定性的なオブジェクトとしてとらえ，それらの関係で図形を記号表現する方法である．PLCA 表現の性質やその上に記述された操作の妥当性や与えられた PLCA 表現に対して２次元平面図形が存在するための条件については証明されている．しかしながら，これらの証明については手作業で行われており，正しさが確実に保証されているとはいえない．本発表では，証明支援系 Coq を用いて PLCA 表現を体系としてモデル化し，その妥当性を証明する．表現の定義は帰納的にモデル化できるように変更した．また，２次元平面図形との関係を証明する際に出現した問題についても議論する．

1　はじめに

コンピュータ上で図形などを扱う場合に，ポリゴンを使って形を表現し，頂点の座標を使って正確にオブジェクトを記述することで必要な情報を数値計算で求める手法が一般的である．このような定量的な方法は画像処理やコンピュータグラフィクスで使われるがその中で定性的な情報も必要とすることがある．例えば，領域のオブジェクト同士が接している，重なっている，もしくは内包しているなどである．こういった情報については定性的に処理できるのが好ましい．従って，定性空間推論の手法を用いて推論を行うことで計算時間の削減が期待できる．しかし，一般的に提案されている定性空間推論システムの妥当性は机上での証明はされているが，すべての場合についての厳密な証明がなされているわけではない．従って，形式的な方法で妥当性を調べる必要がある．形式的に検証する方法として，ここではコンピュータ上で対象をモデル化して証明を行う定理証明器を使用する．定理証明器には Coq [1]，Isabelle/HOL [2]，PVS [3]，ACL2 [4]，Agda [5] などがある．定理証明器はそれぞれベースとなる論理をもっていて，その論理によって証明を進めることができる．一般に帰納的定義と帰納法によって証明を行うものが多く，型理論に基づいて検証するものもある．定理証明器で証明されたものは机上で証明されたものに比べて非常に信頼性が高いため，検証に用いられることがよくある．

本研究では定性空間推論の１つである PLCA 表現 [6] [7] の妥当性を定理証明器を使って証明し，PLCA 表現をより確固で信頼性の高いものにすることを目標にしている．PLCA 表現はその枠組みの１つであり，点 (Point)，線 (Line)，閉路 (Circuit)，範囲 (Area) の４つのオブジェクトで定性的に図形を表現し，それらの関係を推論するものである．定理証明器には Coq を使用し，帰納的アプローチによって証明を試みる．P，L，C，A はもともと集合として定義されているが，本研究では集合の要素数をカウントすることを念頭に入れているためリストで表現し，重複する要素が存在しないように定義した．PLCA 表現を帰納的に定義したのち，その定義が PLCA 表現として満たすべき制約条件を充足することを一部を除き証明した．

本発表は以下のように構成される．第２節で定理証

On modeling and proving validity of qualitative spatial representation PLCA.

Mizuki Goto, 関西学院大学大学院理工学研究科, Graduate School of Science and Technology, Kwansei Gakuin University.

Kazuko Takahashi, 関西学院大学理工学部, School of Science and Technology, Kwansei Gakuin University.

明器について簡単に触れ，第３節で PLCA 表現について説明する．そして第４節で帰納的に PLCA 表現を定義する方法について述べる．第５節では証明を行い，第６章で結論を述べる．

2 Coq

本研究に使用した Coq は対話型定理証明器であり，直観主義論理に基づいている定理証明器である．逐一手動で式変形を行い，証明の状況を使用者に全て提示されるようになっているため，手順を考えて証明を組み立てることができる．しかし，デメリットとしてライブラリが少なく自分で定義する量が多くなることがある．一方で Isabelle/HOL などは強力な自動推論機構を持つ．PLCA 表現の検証は必ず証明できる目算がなかったため，自動的に証明を進める定理証明器は好ましくない．よって手動で証明を行いかつ，理解しやすい Coq を本研究の定理証明器に採用した．

2.1 Coq の表記法

Coq では帰納的定義は以下のように表記される．

```
Inductive nat : Set := 0 : nat | S : nat → nat.
```

nat(natural number) は帰納的に定義されておりターミネータとなる 0 と後者関数である S(Successor) の２つコンストラクタからなる．Set とは型の上位に存在する型で帰納的定義は必ず Set か Type か Prop を型にとらなければならない．今回の **nat** は命題 (Proposition) ではなく Type ほど上位の型ではないので Set となっている．

次にリスト構造を表す List の定義は以下の通りである．

```
Inductive List(A : Type) : Type :=
| nil : List A
| cons : A → List A → List A.
```

見やすくするために改行しているが，改行することで Coq 上で意味合いが変わることはない．リストは Type 型を引数にもち，その Type 型の A のリストを作る．cons は一般的に::と表記しリスト同士の結合は++と表記する．なお，::，++は中置演算子である．

同値関係は Definition を使って行う．

```
Definition three := S (S (S 0)).
```

これは three という型が **nat** の３と等しいことを表している．

3 PLCA 表現

PLCA 表現は領域同士の接続関係を扱うための最低限の情報をもっている．

3.1 オブジェクト

ここでは PLCA 表現に現れる４つのオブジェクトについて詳しく説明する．

3.1.1 Point(点)

Point とはいわゆる一般的な点を指す．

3.1.2 Line(線)

Line とはある Point から Point への直線または曲線を表す．つまり，持っている情報は始点と終点のみである．また，Line は有向線であり，始点と終点が反転したものは別の線として区別する．今後，Line は２つの点のペアとして () を使って表現する．

3.1.3 Circuit(閉路)

Circuit とは Line のリストであり，その線が閉路のように環状に並んだものを指す．Circuit であるためにはいくつか制約がある．まず，Line リストは必ず長さが１以上でなくてはならない．これは線が存在しない閉路は考えられないためである．そして，リストの要素にあたる Line は終点と次の要素 (Line) の始点が一致していなければならない．すなわち

$[(p_0, p_1), (p_1, p_2), (p_2, p_3), (p_3, p_4), ..., (p_{n-1}, p_n)]$

ただし，$p_0 = p_n$ である．Circuit には Line 同様，向きが存在する．つまり，右回りと左回りの Circuit は内向きと外向きの２つとして区別する．つまり，右回りか左回りかによって内向きか外向きか決めることができる．

3.1.4 Area(範囲)

Area とはある範囲の中の存在する閉路を集めたものであり，Circuit のリストで表現される．これも Circuit と同じく，リストの長さが１以上でなければならない．そして，同じ Area に含まれるどの Circuit

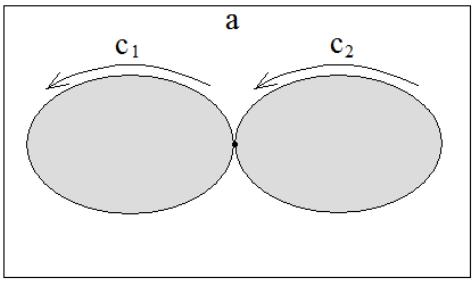

図 1　ありえない PLCA 表現

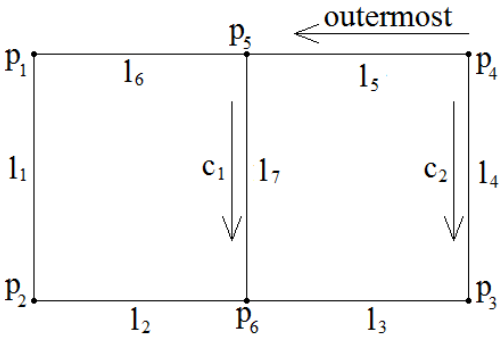

図 2　PLCA 表現の例

も他の Circuit と点または線で接してはならない．この条件を例を使って説明する．もし領域 a 内に異なる閉路 c_1, c_2 があり，これらが 1 点で接しているとする (図 1)．しかし，PLCA は外側をまとめて 1 つの閉路で表現すべきであるため，この例は条件を満たさない．

3.1.5　PLCA 表現

点集合，線集合，閉路集合，範囲集合そして図形上で一番外側の閉路を表す outermost の組を PLCA 表現という．点集合を A，線集合を L，閉路集合を C，範囲集合を A とすると，PLCA 表現 E は

$$E =< P, L, C, A, outermost >$$

となる．

ただし，L は有向線の集合ではなく無向線としての線集合である．つまり,「L にある線が含まれる」とは集合 L にはそのままの形で含まれているか，逆向きで含まれているかのどちらかであることを示す．

図 2 は PLCA 表現の例を表している．P，L，C，A のそれぞれの集合は以下のようになる．

$$\begin{aligned} P &= \{p_1, p_2, p_3, p_4, p_5, p_6\} \\ L &= \{l_1, l_2, l_3, l_4, l_5, l_6, l_7\} \\ C &= \{c_1, c_2, outermost\} \\ A &= \{a_1, a_2\} \end{aligned}$$

4　妥当性

与えられた PLCA 表現

$$E =< P, L, C, A, outermost >$$

が妥当な PLCA 表現であるためには 3 つ制約条件を満たす必要がある．以下はその条件であり，証明を行うにあたって [6] の表現方法を意味を変えることなくいくつか変更した．

4.1　Point-Line 間の制約

全ての点はある線に含まれる．点が線に含まれるとはその点が始点か終点のどちらかであることを表す．

$$\forall p \in P \ \exists l \in L \ (p \in l) \qquad (1)$$

全ての線の始点，終点は点集合に含まれる．

$$\forall l \in L \ (p \in l \Rightarrow p \in P) \qquad (2)$$

4.2　Line-Circuit 間の制約

全ての線はある閉路に含まれる．線が閉路に含まれるとは線が閉路を構成する線の一部であることを表す．

$$\forall l \in L \ \exists c \in C \ (l \in c) \qquad (3)$$

全ての閉路に含まれるいずれの線も L に含まれる．

$$\forall c \in C \ (l \in c \Rightarrow l \in L) \qquad (4)$$

ある線を含む 2 つの閉路があった場合，それらは同じ閉路である．つまり，ある線を含む閉路は 1 つしかない．

$$\forall l \in L \ \forall c1 \ c2 \in C$$
$$(l \in c1 \wedge l \in c2 \Rightarrow c1 = c2) \qquad (5)$$

4.3　Circuit-Area 間の制約

outermost 以外の全ての閉路はある範囲に含まれる．閉路が範囲に含まれるとは閉路が範囲を構成する閉路の一部であることを表す．

$$\forall c \in C, \exists a \in A, (c \neq outermost \Rightarrow c \in a) \qquad (6)$$

全ての範囲が持ついずれの閉路も C に含まれる．

$$\forall a \in A, (c \in a \Rightarrow c \in C) \tag{7}$$

outermost でないある閉路を含む２つの範囲があった場合，それらは同じ閉路である．つまり，ある閉路を含む閉路は１つしかない．

$$\forall c \in C\ \forall a1\ a2 \in A$$
$$(c \neq outermost \Rightarrow c \in a1 \Rightarrow c \in a2$$
$$\Rightarrow a1 = a2) \tag{8}$$

5 帰納的な PLCA 表現

本節では式の定義を Coq のコードで記述する．PLCA 表現はグラフ表現と対応づけて考えられる．従って，PLCA 表現の定義は帰納的にオブジェクトである Point または Line の数が増えるように行うのが自然と考えられる．しかし，１つずつ Point または Line を追加していくとグラフ上に突如 Circuit や Area が現れる性質を持つので，単純な帰納的なグラフの定義ができない．従って，ある操作によって同時に４つのオブジェクトを追加していくような帰納的定義が必要となる．

5.1 オブジェクトの定義

定理証明器で定義した各オブジェクトは以下の通りである．

```
Definition Point := nat
Definition Line := Point * Point
Definition Circuit := list Line
Definition Area := list Circuit
```

5.2 PATH

Circuit は list Line として定義されているが Circuit としての性質を満たしている必要がある．そのため，PATH という述語を用意することで，Circuit の性質を持っている list Line に特定できるようにした．PATH とはある２つの Point 間を結ぶループをもたない経路が存在することを表している．経路とは Line のリストであり，経路の長さは０の場合も許される．

PATH は帰納的に定義され，引数として P，L，始点，終点，通った点のリスト，経路を持つ．
構成子は以下の通りにした．

nil_path

ある Point が P に含まれるならその Point を始点，終点とする長さ０の PATH がある．

$$\forall p \in P \Rightarrow PATH\ P\ L\ p\ p\ (p :: nil)\ nil$$

step_path

ある PATH が存在し，さらにその PATH に含まれない別の Point $x \in P$ があったとき，x と PATH の始点との Line $l \in L$ を経路に加えたものも PATH になる．

$$\forall P, L, \forall p1, p2, p3 \in P, \forall l \in L, \forall pl, ll,$$
$$PATH\ P\ L\ p2\ p1\ pl\ ll \Rightarrow p3 \notin pl \Rightarrow$$
$$PATH\ P\ L\ p3\ p2\ (p3 :: pl)\ ((p3, p2) :: ll)$$

閉路とは一般的に始点と終点が同じ PATH であるといえるが，この PATH だけでは閉路を構成できないため

$$\forall P\ L\ pl\ ll\ \forall x\ y \in P$$
$$PATH\ P\ L\ x\ y\ pl\ ll\ \wedge (x,\ y) \in L \wedge length\ ll \geq 2$$

を満たす場合，$((y, x) :: ll)$ を閉路とした．

5.3 PLCA 表現の定義

構成子は３つとし，引数は P，L，C，A とした．

e_outermost

e_outermost とはベースケースに相当する構成子であり，outermost となる閉路およびその閉路と線を共有する閉路の２つだけ存在する．P と L は閉路を構成する点と線のみとなり，A は outermost の内側にある範囲のみとなる．

connect_circuit

outermost 以外のある閉路に含まれる２つの点を結ぶ新たな経路を作った時，その経路を追加することによって変化したオブジェクトを更新する．

isolate_circuit

ある範囲に含まれる全ての閉路と線または点を

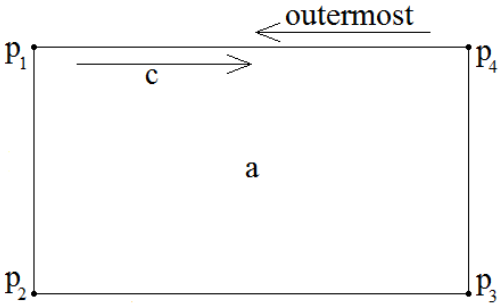

図 3　e_outermost

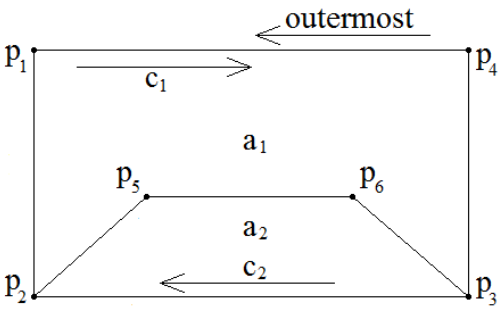

図 4　e_outermost ⇒ connect_circuit

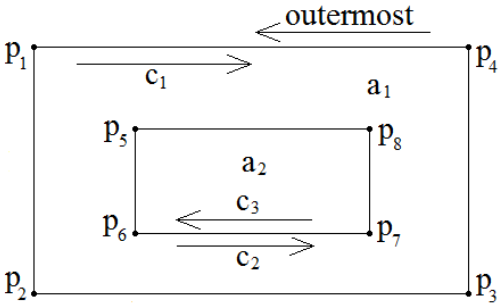

図 5　e_outermost ⇒ isolate_circuit

共有しない新たな閉路を追加する．

構成子が e_outermost のみの場合は図 3 のようになる．そして，図 3 に connect_circuit を適用すると図 4 のようになり，図 3 に isolate_circuit を適用すると図 5 のようになる．

6　証明

第 5 節の帰納的な PLCA 表現が正しいことを証明する．P，L，C，A は集合であるが，定理証明器ではリストとして定義したので「P，L，C，A が重複する要素が存在しないこと」と「$a \in A$ となる a が重複する要素を含まないこと」，さらに「outermost がいずれの範囲に属さないこと」も加えて証明した．また，PLCA のもつ制約条件 (1) ～ (8) の内，(5) を除いて全て証明が完了した．

以下の補題は第 4 節で説明した制約条件を Coq で記述したものである．

6.1　P から L への制約条件 (1)

```
Lemma PL_limitation :
∀ (P : set Point)(L : set Line) (C : set Circuit)
  (A : set Area)(e : E P L C A) (p : Point),
  set_In p P →
  (∃ l : Line, InL l L ∧ Point_In_Line p l)
```

set という型は **list** と同値定義されているため，P，L，C，A はリストである．set_In もまた，リストの包含関係を示す In の関数と同じであるが，集合ということを強調するためにこちらを使用した．In a A で要素 a がリスト A に含まれることを指す．Point_In_Line とはある点が線の始点または終点であることを表す．Line はリストではないため In の関数が使えない．従って，このような関数を導入した．

6.2　L から P への制約条件 (2)

```
Lemma LP_limitation :
∀ (P : set Point)(L : set Line) (C : set Circuit)
(A : set Area)(e : E P L C A) (p : Point)(l : Line),
  set_In l L → Point_In_Line p l → set_In p P
```

6.3　L から C への制約条件 (3)

```
Lemma LC_limitation :
∀ (P : set Point)(L : set Line) (C : set Circuit)
  (A : set Area)(e : E P L C A) (l : Line),
  InL l L → (∃ c : Circuit, In l c ∧ set_In c C)
```

InL とは線が無向線として L に含まれることを意味

する．

6.4 C から L への制約条件 (4)

Lemma CL_limitation :
∀ (P : set Point)(L : set Line) (C : set Circuit)
(A : set Area)(e : **E** P L C A) (l : Line)(c : Circuit),
set_In c C → In l c → InL l L

6.5 ある線を含む閉路はただ１つである．(5)

Lemma Circuit_only_one :
∀ (P : set Point)(L : set Line) (C : set Circuit)
(A : set Area)(e : **E** P L C A) (l : Line)
($c1$ $c2$: Circuit),
set_In $c1$ C → set_In $c2$ C →
In l $c1$ → In l $c2$ → $c1 = c2$

6.6 C から A への制約条件 (6)

Lemma CA_limitation :
∀ (P : set Point)(L : set Line) (C : set Circuit)
(A : set Area)(e : **E** P L C A) (c : Circuit),
set_In c C → $c \neq$ ($outermost$ e) →
(∃ a : Area, In c a ∧ set_In a A)

6.7 A から C への制約条件 (7)

Lemma AC_limitation :
∀ (P : set Point)(L : set Line) (C : set Circuit)
(A : set Area)(e : **E** P L C A) (a : Area)(c : Circuit),
set_In a A → In c a → set_In c C

6.8 ある閉路を含む範囲はただ１つである．(8)

Lemma Area_only_one :
∀ (P : set Point)(L : set Line) (C : set Circuit)
(A : set Area)(e : **E** P L C A) (c : Circuit)
($a1$ $a2$: Area),
set_In $a1$ A → set_In $a2$ A →
In c $a1$ → In c $a2$ → $a1 = a2$

(2)，(4)，(5) そして，(7) の他に「集合 A が重複する要素を含まないこと」(9) と，「集合 A の要素 a が重複する要素を含まないこと」(10) を加えた６つの補題は単独で証明することができなかった．それぞれ補題は PLCA 表現に対する帰納法で証明するのだが，ベースケースが証明できても帰納段階において他の補題を必要とした．表 1 はそれらの関係を表にしたものである．ベースケースとなる e_outermost は何も必要としないため問題ないが，それ以外の場合は帰納段階における仮定がたりないため証明できない．例えば，(2) の isolate_circuit は帰納における仮定があるため証明できる．しかし，connect_circuit では (4)，(7) が足りない．同様のことが以下５つの補題で発生した．全てまとめてから帰納法を適用することでベースケースはもちろん，connect_circuit や isolate_circuit も仮定の過不足なく証明できた．

7 おわりに

PLCA 表現を帰納的に定義する方法を示すことで証明可能なモデルへ変換した．そして，それが３つの制約条件のうち一部を除き全てを満たすことを証明した．また，PLCA 表現の上で曖昧なところを一部発見し，修正を試みた．今回，帰納的に定義した PLCA 表現が元の定義と意味的に等しいことを確かめることが今後の課題である．また，PLCA 表現のグラフを連結グラフに変換する関数を定義した上で，その関数の性質を証明し，P，L，C，A の集合の要素の関係式[2] について厳密な証明を与えることも考えている．

8 謝辞

本研究は科研費 25330274 の補助を受けて行われた．

参 考 文 献

[1] Bertot, Y. and Casteran, P. : *Interative Theorem Proving and Program Development*, Springer, 2004.
[2] Nipkow, T. Paulson, C, L. and Wenzel, M.: *Isabelle/HOL A Proof Assistant for Higher-Order Logic*, Springer, 2008.
[3] Crow, J., Owre, S., Rushby, J., Shankar, N. and Srivas, M.: *A Tutorial Introduction to PVS*, 1995.
[4] Kaufmann, M. and Moore, S. J.: *ACL2-Tutorial*, 1997.
[5] Norell, U.: *Dependently Typed Programming in Agda*, Advanced Functional Programming 2008, pp.230-266, 2008.
[6] 住友孝郎 ：定性空間推論における新しい枠組みの提案, 関西学院大学理工学研究科修士論文, 2006.
[7] Takahashi, K., Sumitomo, T. and Takeuti,I.: *On Embedding a Qualitative Representation in a Two-Dimensional Plane*, Spatial Cognition and Computation, Vol.8, No.1-2, pp.4-26, April, 2008.

表 1　それぞれの補題に PLCA 表現に対する帰納法を適用した場合に必要となる補題

証明する補題	e_outermost	connect_circuit	isolate_circuit
(2)	None	(2), (4), (7)	(2)
(4)	None	(4), (7)	(4)
(7)	None	(7), (8), (9), (10)	(7)
(8)	None	(2), (4), (7), (10)	(2), (4), (7), (8), (9)
(9)	None	(2), (4), (7), (9)	(2), (4), (7), (9)
(10)	None	(2), (4), (7), (8), (9)	(2), (4), (7), (10)
(2) + (4) + (7) (8) + (9) + (10)	None	(2) + (4) + (7) (8) + (9) + (10)	(2) + (4) + (7) (8) + (9) + (10)