定性空間表現の Coq による形式化 およびその平面性の証明

後藤 瑞貴[1]　森口 草介[1]　高橋　和子[1]

概要：定性空間表現 PLCA を証明支援系 Coq を用いて帰納的に定義することでモデル化し，このモデルが平面性を満たす PLCA の集合と一致していることを証明する．PLCA は，空間データに対して，それを構成する点 (Point)，線 (Line)，閉路 (Circuit)，範囲 (Area) というオブジェクトを用い，それらの包含関係によって表現する手法である．この表現は座標データを用いずに領域同士の接続関係を定性的に表すもので，空間データ上に関して焦点をしぼった推論ができる．推論の妥当性を保証するためには，PLCA と空間データの対応関係を証明する必要があるが，与えられた PLCA 表現が 2 次元平面に埋め込めるための条件は示されているものの，平面性条件を満たす PLCA の集合を帰納的に構築する方法については議論されていない．本発表では，まず，3 つの構成子を用いて PLCA を帰納的に定義し，その等価性を定義する．そして，このようにして構築されたモデルが平面性条件を満たすことを Coq で証明する．証明は，帰納法に基づき，各構成子に対して場合分けして行う．一方，平面性条件を満たす PLCA が帰納的 PLCA として構築できることを，範囲の数に関する帰納法を使って証明する．証明では，範囲の数を増やした時の PLCA の型を平面性条件から得られるものと一致させる必要があり，帰納の仮定となる PLCA を適切に設定することで証明の道筋を得ることができた．

Formalization of a Qualitative Spatial Representation and the Proof for its Planarity Using Coq

Mizuki Goto[1]　Sosuke Moriguchi[1]　Kazuko Takahashi[1]

Abstract: We give a model to a qualitative spatial representation PLCA by formalizing it inductively using a proof assistant Coq, and prove that such a model coincides with the planar PLCA. PLCA provides a symbolic expression of spatial entities and allows reasoning on this expression. To justify the reasoning, we should prove the correspondence between PLCA expression and spatial data; for a given PLCA expression, the conditions for planarity have been shown, but the construction of such a PLCA expression has not been discussed.In this presentation, we give an inducive defnition to PLCA with three constructors and also its equivalence relation. We then prove that the obtained model satisfes the conditions for planarity using Coq. The proof is based on induction and using case split on each constructor. On the other hand, we prove that a planar PLCA expression can be constructed as an inductive PLCA, using an induction on the number of areas. In the proof, we should match the type of PLCA of the successive number of the areas with that obtained by the conditions for planarity. We have obtained the route of the proof by taking appropriate PLCA as an induction hypothesis.

1. はじめに

画像や特に 3 次元を取り扱うシステムには頂点の座標を記憶し，図形をポリゴンとして表現する方法が実装されることが多い．しかし，目的によっては厳密な数値データは必要ではなく，注目する側面だけについて定性的に表現されていれば，十分なことも多い．例えば，画像からオブジェクトの領域を取得したときに，領域同士が接している，重

[1] 関西学院大学
Gakuen 3–1, Sanda, Hyougo 669–1337, Japan

なっている，もしくは内包していると区別するだけで図形同士の位置関係を定性的に推論することも可能である．図形の特徴を定性的に記述し，ほしい情報を推論によって求める方法は定性空間推論とよばれ [1][2][3]，少ないデータ量や計算量で空間データを認識したり推論したりする人工知能システムへの応用が期待されている．例えば，カメラから取得した画像から図形を認識し，画像に写っていない領域を推論することで立体的な地形のマッピングができる．これまでに提案されている定性空間推論システムは多くが表現力に着目したもので，推論の妥当性に着目したものはほとんどない．これらのシステムでは，オブジェクト同士の関係の集合から新しい関係を導出することを推論としているものがほとんどであり，初期の研究で自動証明器を補助的に使ってこの推論の正しさを検証した事例がある [4]．しかし，提案された体系を実装したり推論の妥当性を機械的に証明した研究は著者らが知る限りではこの他には存在しない．定性空間推論システムを実装したときにエラーがないことが望ましいが，証明が机上である以上ヒューマンエラーが存在する可能性がある．ヒューマンエラーを排除する方法として，コンピュータ上で形式的証明をするツールを用いて数学的に厳密な証明を作ることが考えられる．

コンピュータ上で形式的証明をするツールとしてよく使われるのが定理証明器である．定理証明器は証明支援系とも呼ばれ，人間が行う証明を手助けするツールである．このツールはコンピュータ上で実行するため，机上ではできないような大量の場合分けを証明したり，人間の勝手な思いこみをなくすことができ，厳密で信頼性の高い証明を作ることができる．その代表的なツールとして Coq[5]，Isabelle/HOL[6]，PVS[7]，ACL2[8]，Agda[9] などがあり，様々なシステムの検証に利用されている．本研究で用いる Coq は対話的に証明を進めるツールであり，直観主義に基づいている．Coq は四色定理，奇数位数定理，シャノンの定理などの数学的問題に形式的な証明を与えるのに成功しているだけでなく，C コンパイラの検証など実用的な問題にも応用されている [10][11]．

本研究では定性空間表現の 1 つである PLCA 表現 [12] を Coq を使って形式化し，その性質を証明する．PLCA 表現は点 (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する．本研究では，PLCA を証明可能なモデルにするため，PLCA を帰納的に定義する．帰納的な PLCA はベースケースとなる単純な表現から始まり，2 つの操作のいずれかを施すことによって作られるものである．このように定義した帰納的な PLCA が平面性を満たすこと，すなわちある条件を満たす 2 次元平面上の図形が存在することと，逆に，ある条件を満たす平面図形は帰納的に定義した PLCA のクラスに属することを証明する．その際に，オブジェクト同士の同値類の定義，同値なものに対する包含関係の判定の定義，さらに，PLCA 表現の等価性についても定義する必要があることがわかった．これについては，机上の証明では特に記述することなく対処していた部分であり，今回の形式化で厳密に定義した．平面性の証明では，順方向については Coq による証明が完了し，逆方向については証明の戦略を与えた．同値性や等価性に関する性質も含めて証明はすべて帰納法に基づくものになっている．

本論文は以下のように構成される．第 2 章で PLCA の定義とその平面性について示し，第 3 章で，Coq で行ったモデル化および構成的に PLCA を作成する方法について説明する．第 4 章で，帰納的に定義した PLCA に平面性の証明を与え，第 5 章で証明を行ったときの問題点について議論する．最後に結論を述べる．

2. PLCA

2.1 定性空間表現

定性空間表現は，空間データに対して，数値を用いて定量的に表現せず，空間オブジェクト同士の相対的位置関係やサイズ，方向などの定性的な情報をとらえ，それらを記号表現する手法である [1][2][3]．そこから推論によって，ほしい情報を取り出す枠組みが定性空間推論と呼ばれる．定性空間表現では，1 つの記号表現に対応する図形は複数個存在する．例えば，「2 つの図形が接している」という表現に対して，サイズ，形，接続方法などを無視すれば様々な図形を描くことができる．

実際に図形を用いて説明すると，2 つの図形が交わっている，接している，離れている，という関係だけを区別したい場合，図 1 の 3 つの図形と図 2 の図形はすべて「接している」として同一視される．しかし，点で接しているか線で接しているかまで区別すると図 1 の 3 つの図形は同一でも，これらと図 2 の図形は異なるものと見なされる．

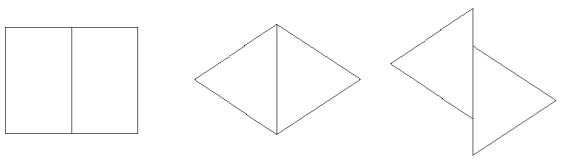

図 1 線で接している図形の例

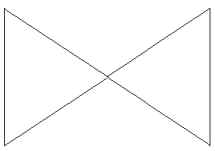

図 2 点で接している図形の例

2.2 PLCA の対象とする図形

PLCA 表現は領域同士の接続関係に着目した定性空間表現であり，Point，Line，Circuit，Area という 4 種類のオブジェクトを用いてオブジェクト同士の関係で図形を表現するものである [12] [*1]．

対象とする図形は，有限範囲に描かれたもので，領域は互いに交わりを持たず，孤立した点や閉曲線でない線は許さないという制約条件をもつものである．PLCA 表現は記号表現なので，一般的に空間図形を対象としているが，2 次元平面上の図形を対象とした場合は，平面分割 (図 3) と図形埋め込み (図 4) によって生成できる図形ととらえることができる．

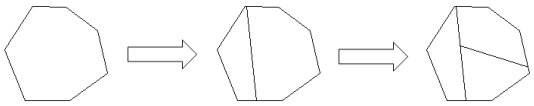

図 3 平面分割

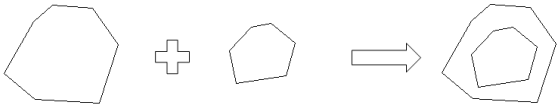

図 4 図形埋め込み

2.3 オブジェクト

PLCA の 4 つのオブジェクトは以下の通りである．

Point

Point とは点であり，プリミティブなオブジェクトである．Point は名前によって区別され，記号表現は名前しか持たない．

Line

Line とは線である． 2 つの Point を結ぶ線を表す．Line の記号表現は 2 つの Point のタプルである．なお，この 2 つの Point は同じ Point でもかまわない．

$Line\ l = (p_1, p_2)$

これは $Line\ l$ が p_1 と p_2 を結んでいることを表している．また，+，− を用いることで，有向線を表すことができる．

$l^+ = (p_1, p_2)$

$l^- = (p_2, p_1)$

Circuit

Circuit とは閉路である．環状に並んだ Line のリストで表された閉路を表す．Circuit の記号表現は長さ 1 以上の Line のリストで，以下の制約条件を満たすものである．

$$Circuit\ c = [(p_1, p_2), (p_2, p_3), (p_3, p_4), \ldots, (p_i, p_{i+1}), (p_{i+1}, p_{i+2}), \ldots, (p_n, p_1)]$$

リストの長さを n とすると，任意の $i\ (1 \le i \le n-1)$ に対してリストの第 i 要素の 2 番目と第 $i+1$ 要素の 1 番目が一致し，第 n 要素の 2 番目と第 1 要素の 1 番目が一致する．

Circuit は始点位置を変えても同じ Circuit を表す．以下の $Circuit\ c'$ を考える．

$$Circuit\ c' = [(p'_1, p'_2), (p'_2, p'_3), (p'_3, p'_4), \ldots, (p'_i, p'_{i+1}), (p'_{i+1}, p'_{i+2}), \ldots, (p'_n, p'_1)]$$

$Circuit\ c, c'$ における任意の $i\ (1 \le i \le n-1)$ に対して $P_i = p'_{i+j \bmod n} (1 \le j \le n-1)$ を満たすとき c と c' は同一のものと見なす．

また，1 つの閉曲線に対して右回りと左回りの Circuit を対応させることができ，その閉曲線の外側と内側をそれぞれの Circuit によって区別することができる．

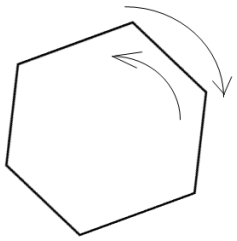

図 5 外側の Circuit と内側の Circuit

図 5 の場合，右回りの矢印が外側の Circuit に対応し，左回りの矢印が内側の Circuit に対応している．

Area

Area とは範囲である．Circuit によって区切られた空間を表す．Area の記号表現はその空間の中に入っている Circuit の集合である．ただし，その集合は少なくとも 1 つの Circuit を含み，同じ Area に属する任意の 2 つの Circuit は Point でも Line でも接していない．

$Area\ a_1 = \{c_2, c_3\}$

$Area\ a_2 = \{c_4\}$

図 6 の Area の記号表現

図 6 を記号で表現すると，a_1 が c_2 と c_3 を含む集合で構成され，a_2 が c_4 のみの集合で構成される．

Point 接続と Line 接続をそれぞれ以下のように定義する．

[*1] PLCA という名前はオブジェクトの頭文字を並べたものである．

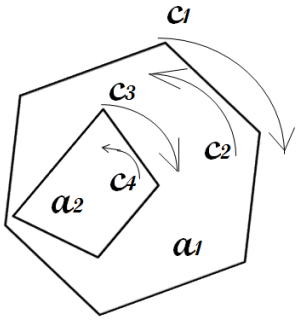

図 6　Area の例

(Point 接続)　$pc(c_1, c_2) = \exists p \in P\ \exists l_1, l_2 \in L(p \in l_1 \wedge p \in l_2 \wedge l_1 \in c_1 \wedge l_2 \in c_2)$

(Line 接続)　$lc(c_1, c_2) = \exists l \in L(l \in c_1 \wedge l \in c_2)$

なお，$p \in l_1$ とは p が l_1 の構成要素であることを示す．また同様に，$l_1 \in c_1$ は l_1 が c_1 のリストの中に含まれていることを表す．すると，Circuit が Point でも Line でも接していないことを以下のように表現できる．

$$\neg pc(c_1, c_2) \wedge \neg lc(c_1, c_2) \tag{1}$$

outermost

outermost とは図形の一番外側に相当する Circuit を表す．(図 7)

図 7　outermost の例

2.4　PLCA 表現

2.3 節で示されるような制約条件のある Point, Line, Circuit, Area の各集合 P，L，C，A と，$outermost$ の 5 つのタプル

$$E = \langle P, L, C, A, outermost \rangle \tag{2}$$

を **PLCA 表現**と呼ぶ．

図 8 に対応する PLCA 表現は以下のようになる．

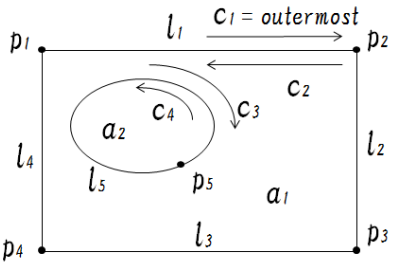

図 8　図形とオブジェクトの対応

$E = \langle P, L, C, A, outermost \rangle$

$P = \{p_1, p_2, p_3, p_4, p_5\}$

$L = \{l_1, l_2, l_3, l_4, l_5\}$

$C = \{c_1, c_2, c_3, c_4\}$

$A = \{a_1, a_2\}$

$l_1 = (p_1, p_2)$　$c_1 = [l_1^+, l_2^+, l_3^+, l_4^+]$　$a_1 = \{c_2, c_3\}$
$l_2 = (p_2, p_3)$　$c_2 = [l_1^-, l_4^-, l_3^-, l_2^-]$　$a_2 = \{c_4\}$
$l_3 = (p_3, p_4)$　$c_3 = [l_5^+]$
$l_4 = (p_4, P_1)$　$c_4 = [l_5^-]$
$l_5 = (p_5, p_5)$

2.5　Consistency

Consistency とはある PLCA 表現 E が 2.2 節で述べたような対象とする図形になっているかどうか判定するための条件である．(2) 式の P，L，C，A にはそれぞれの要素間に関係性がないため，例えば，L に含まれている Line l があったとき，それを構成する Point が集合 P に存在しないことがある．Consistency の条件を用いることで，このようなものを避ける．PLCA 表現 $E = \langle P, L, C, A, outermost \rangle$ が以下の (3) 式から (12) 式までを満たすとき，E を**無矛盾な PLCA** といい，$Consistency(E)$ と記述する．

P-L Consistency

これは Point と Line の間に矛盾がないことを示す条件であり，孤立した Point を許さないという制約条件に対応する．Line l が Point p を含むとは，l を構成するものの中に p が含まれていることである．

$$\forall p \in P \exists l \in L(p \in l) \tag{3}$$

任意の Point p に対して，集合 L に含まれる Line の中に p を含むようなものが存在する．

$$\forall l \in L \forall p \in l(p \in P) \tag{4}$$

任意の Line に対して，それを構成する Point 全てが集合 P に含まれる．

L-C Consistency

これは Line と Circuit の間に矛盾がないことを示す条

件であり，平面分割になっていない線や孤立した線などを許さないという制約条件に対応する．Circuit c が Line l を含むとは，c を構成する Line のリストが l が含むことである．

$$\forall l \in L \exists c \in C(l \in c) \tag{5}$$

任意の Line l に対して，集合 C に含まれる Circuit の中に l を含むようなものが存在する．

$$\forall c \in C \forall l \in c(l \in L) \tag{6}$$

任意の Circuit に対して，それを構成する Line 全てが集合 L に含まれる．

$$\forall l \in L \forall c1, c2 \in C(l \in c1 \wedge l \in c2 \Rightarrow c1 = c2) \tag{7}$$

ある Line を含む Circuit が 2 つあった場合，それらは同じ Circuit である．つまり，ある Line を含む Circuit は必ず 1 つである．

C-A Consistency

これは Circuit と Area の間に矛盾がないことを示す条件であり，図形埋め込みになっていない Circuit と Area の関係を許さないという制約条件に対応する．Area a が Circuit c を含むとは，a を構成する Circuit の集合が c を含むことである．

$$\forall c \in C \exists a \in A(c \neq outermost \Rightarrow c \in a) \tag{8}$$

outermost 以外のあらゆる Circuit c に対して，集合 A に含まれる Area 中に c を含むものが存在する．

$$\forall a \in A \forall c \in a(c \in C) \tag{9}$$

任意の Area に対して，それを構成する Circuit 全てが集合 C に含まれる．

$$\begin{aligned} &\forall c \in C \forall a1, a2 \in A \\ &((c \neq outermost \wedge c \in a1 \wedge c \in a2) \\ &\Rightarrow a1 = a2) \end{aligned} \tag{10}$$

outermost 以外のある Circuit を含む Area が 2 つあった場合，それらは同じ Area である．つまり，ある Circuit を含む Area は必ず 1 つである．

outermost Consistency

これは (2) 式の 5 つ目の要素である *outermost* と各集合の関係を表した条件である．

$$outermost \in C \tag{11}$$

集合 C は outermost を含む．

$$\forall a \in A(outermost \notin a) \tag{12}$$

集合 A が含むあらゆる Area は outermost を含まない．

2.6 PLCA の平面性

PLCA 表現が 2 次元平面上に描画できるとき，その PLCA 表現は**平面的である**という．ある PLCA 表現が平面的かどうかを判定するための条件が高橋らの研究 [13] によって明らかになっている．その条件は 3 つあり，1 つは 2.5 節で述べた無矛盾性である．以下では残る 2 つの条件について説明する．

2.6.1 PLCAconnected

ある PLCA 表現が PLCAconnected であるとは，その中に出現する任意の 2 つのオブジェクトが，別のオブジェクトを通じて繋がっていることを表す．すなわち，オブジェクト同士が切り離されたり，孤立していないことを表す．具体的には以下のような述語を用いて定義される．

(d-pcon) PLCA 表現 $E = \langle P, L, C, A, outermost \rangle$ のオブジェクトのペアを引数に取る述語 *d-con* を以下のように定義する．

1 d-pcon$(p, l) \Leftrightarrow p \in l$
2 d-pcon$(l, c) \Leftrightarrow l \in c$
3 d-pcon$(c, a) \Leftrightarrow c \in a$

(pcon) PLCA 表現のオブジェクト α, β, γ が与えられたとき，以下の 3 つの性質を満たすような関係を述語 *pcon* と定義する．

1 d-pcon$(\alpha, \beta) \Rightarrow pcon(\alpha, \beta)$
2 pcon$(\alpha, \beta) \Rightarrow pcon(\beta, \alpha)$
3 pcon$(\alpha, \beta) \wedge$ pcon$(\beta, \gamma) \Rightarrow$ pcon(α, γ)

PLCA 表現 $E = \langle P, L, C, A, outermost \rangle$ に対して

$$\forall o1, o2 \in P \cup L \cup C \cup A \ pcon(o1, o2)$$

を満たすときその E を **PLCAconnected** といい，$PLCAconnected(E)$ と記述する．

図 8 を例にとると，$o1 = p_1$ と $o2 = p_5$ の場合，以下の推論により，$pcon(p_1, p_5)$ が成り立つ．$p_1 \in l_4$ なので，*d-pcon* の 1 番目の定義により，$d\text{-}pcon(p_1, l_4)$ が成り立つ．すると，*pcon* の 1 番目の定義により $pcon(p_1, l_4)$ が成り立つ．また，$l_4 \in c_2$ なので，*d-pcon* の 2 番目の定義より $d\text{-}pcon(l_4, c_2)$ が成り立つ．すると，*pcon* の 1 番目の定義より $pcon(l_4, c_2)$ が成り立つ．したがって，*pcon* の 3 番目の定義より $pcon(p_1, c_2)$ が成り立つ．同様に，$pcon(c_2, a_1)$，$pcon(a_1, c_3)$，$pcon(c_3, l_5)$，$pcon(l_5, p_5)$ も成り立つことから，$pcon(p_1, p_5)$ が成り立つ．

2.6.2 PLCAeuler

PLCA 表現 $E = \langle P, L, C, A, outermost \rangle$ は

$$|P| - |L| - |C| + 2 * |A| = 0 \quad (13)$$

を満たすとき，E は **PLCAeuler** であるといい，$PLCAeuler(E)$ と記述する．

2.6.3 PLCA の平面性定理

PLCA において以下の平面性に関する定理が成り立つ [13].

(定理 1) ある PLCA 表現が Consistency を満たし, PLCA-connected であるとき，それが PLCAeuler である時かつその時に限り，その PLCA は平面的である．

本研究では新たな PLCA のクラスを提案する．それは基礎となる図形から始まり，2 つの操作によって得られる PLCA であり，帰納的 PLCA と呼ばれる．次章以降は，この帰納的 PLCA を Coq 上で定義することで，このクラスが平面である PLCA と一致することを示すのが目標である．

定理 1 から平面である PLCA は，Consistency，PLCA-connected，PLCAeuler を満たすことと同値なので，帰納的 PLCA がこの 3 つの条件を満たし，逆にこの 3 つの条件を満たすものは帰納的 PLCA となることを証明する．この 3 つの条件を **PLCA の平面性条件**と呼び，これらを満たす PLCA を**平面的 PLCA** と呼ぶ.

3. 形式化

PLCA 表現を Coq を使って形式化する．以降は Coq 上のコードを記載する．

3.1 概要

まず，Point，Line，Circuit，Area というオブジェクトを定義し，これらの同値類を定義する．集合はリストで実装するため，そのリストが重複する要素を含まないという条件が必要になる．また，Point 以外のオブジェクトの上で同値類を定義する必要がある．Line の向きを反転させたものや，Circuit の始点位置を変えただけのもの，Area が持つ Circuit の始点位置を変えただけのものなどはオブジェクトとしては同値と見なさなければならない．例えば，Circuit の集合に互いに同値なものが含まれていると，これらは要素として二重にカウントされてしまい，PLCAeuler で集合の要素数をカウントする際に問題になるからである．こういった問題を避けるために Point 以外のオブジェクトに対してその同値類を定義し，あるオブジェクトに対してそれと同値な要素が対象となる集合に含まれているか否かを判定する述語をオブジェクトごとに定義する．これらは，机上の証明だと証明者が認識して暗黙のうちに対処しているものであるが，計算機上でモデルを作成する場合には陽に記述する必要がある．同値関係にあるオブジェクトを同一と見なした上で，2 つの PLCA 表現の等価性を定義する．

次に，与えられた Point，Line，Circuit，Area の各集合および outermost に対して制約条件 Constraints，Consistency，PLCAconnected，PLCAeuler を定義し，平面的 PLCA を定義する．

最後に，構成子を用いて PLCA を帰納的に構築する．この定義では，Point，Line，Circuit，Area を順番に生成するのではなく，ある PLCA 表現に対して Area を 1 つ加える構成子を適用することで Area の数が 1 つ増えた PLCA 表現を生成する．Area を増やすことでそれに付随する他のオブジェクトの数も増える．

3.2 オブジェクト

2.3 節で示した 4 つのオブジェクトについて示す．

3.2.1 Point

```
Definition Point := nat.
```

Point は互いに区別できるものであればよいため, 自然数の構造を利用した．

3.2.2 Line

```
Definition Line := Point * Point
```

Line は 2 つの Point の組になるよう定義した．この定義では 1 つ目と 2 つ目の要素の位置を区別するため，この Line の定義は有向線を表している．したがって，逆向きの有向線は reverseを用いて表す.

```
Definition reverse(l : Line) : Line :=
(snd l, fst l).
```

fst，sndとはそれぞれ，組から 1 つ目の要素，2 つ目の要素をそれぞれ取り出す関数である．これにより，任意の有向線を表すことができる．

また，この Line を無向線として扱いたいときは以下の定義を用いる．

```
Inductive eq_ul : Line -> Line -> Prop :=
| ul_refl    : forall(l : Line), eq_ul l
   l
| reverse_ul :
forall(l1 l2 : Line),
reverse l1 = l2 -> eq_ul l1 l2.
```

eq_ulは Line の同値関係を表す述語である．引数に取った 2 つの Line が無向線として等しいことを表す．次に Line とそのリストの包含関係 LInを定義する．まずその前に，要素とリストの一般的な包含関係 EqInを定義する．

```
Fixpoint EqIn
(A : Type)(EQ : A -> A -> Prop)
(a : A)(l : list A) :=
match l with
| nil => False
| a' :: l' => EQ a a' \/ EqIn a l'
end.
```

EqIn とはある同値類 EQを引数に取ることで EQにおいて lが aを含むという述語である．これを使って Line の包含関係の述語 LInを定義する．

```
Definition LIn (l : Line)(L : list Line):
    =
EqIn eq_ul l L.
```

LInは EqInに Line の同値類 eq_ulを引数に代入することで Line とそのリストの包含関係を表している．

最後に，ある Line がある Point の構成要素であるという述語を定義する．

```
Definition InPL (p : Point)(l : Line) :=
fst l = p \/ snd l = p.
```

これはある Line がある Point を含むことを表す．

3.2.3 Line-constraints

Line を構成する 2 つの Point が同一だった場合，以下のような不都合が生じる．

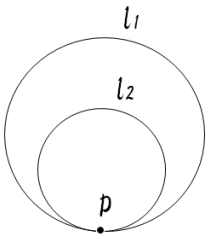

図 9 Line を区別できない図形

$Line\ l_1 = (p, p)$

$Line\ l_2 = (p, p)$

図 9 の Line の記号表現

図 9 のような例を考える．この図には l_1 と l_2 に囲まれる Area があるので，l_1 と l_2 を区別しなければならない．しかし，Line を構成する 2 つの Point が同一のものだとその 2 つの Line が同じ組の構造を表すことになり，これらを区別することができない．したがって，Line を構成する 2 つの Point は同一でないという制約条件を付け加えることでこの問題を回避した．制約条件を以下のように定義する．

```
Definition Line_constraint(l : Line) :=
(fst l) <> (snd l).
```

3.2.4 Circuit

Circuit の定義は以下の通りである．

```
Definition Circuit := list Line.
```

ただし，Circuit が環状構造であるため，制約条件が必要となる．制約条件は環状構造になっていることと，リストの長さが 3 以上であることである．

3.2.5 Circuit-constraints

環状構造を表現するため，Trailという述語を用意する．

```
Inductive trail :
Point -> Point -> list Point ->
list Line -> Prop :=
| nil_trail  :
forall(p : Point), trail p p (p::nil) nil
| step_trail :
forall(p1 p2 p3 : Point)(pl : list Point)
(ll : list Line),
   trail p2 p1 pl ll
-> p3 <> p2
-> ~LIn (p3, p2) ll
-> trail p3 p1 (p3::pl) ((p3, p2)::ll).
```

Trailは引数は左から順番に，始点，終点，いままで通った Point のリスト，いままで通った Line のリストになっている．これはグラフ理論におけるトレイルに相当するものを表し，Line は重複しないが，Point は重複してもよい．1 つの Point からなるトレイルから始まり，Line を 1 つずつ付け加えていくことで，任意の長さのトレイルを作ることができる．

次に，Circuit の長さについての制約条件を考える．Line-constraints から長さは 2 以上であるが，長さ 2 の Circuit の場合は以下のような図形で問題が生じる．

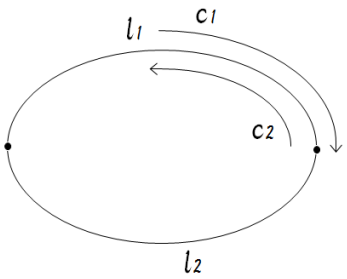

図 10 長さ 2 の Circuit で問題が生じる図形

$Circuit\ c_1 = [l_1^+, l_2^+]$

$Circuit\ c_2 = [l_1^-, l_2^-]$

図 10 の Circuit の記号表現

この図では，$Line\ l_1$ と $Line\ l_2$ は有向線としては異なっていても，無向線としては同じ線になってしまうため，$Circuit\ c_1, c_2$ が $Trail$ の性質を満たさず，さらに集合 L が重複する要素を持つことになる．したがって，任意の 2 つの Point に対して，それらを繋げる Line は高々 1 つとし，Circuit のリストの長さを 3 以上とした．これらによって，Circuit の制約条件を以下のように記述することができる．

```
Definition Circuit_constraints
(c : Circuit) :=
(exists x : Point, exists pl : list Point
    ,
trail x x pl c)
/\ 3 <= length c.
```

したがって，Coq による形式化では曲線は現れないと考えてよい．また，構成する点の数が異なる Circuit は異なるものとして扱われる．

lengthはリストを受け取ってその長さを返す関数である．

Circuit は回転したものも同じ Circuit として取り扱う．したがって，Circuit の同値類を定義する必要がある．Circuit における回転とはリストの先頭要素を最後尾に移動する操作を複数回繰り返して，各要素をずらすことである．

```
Inductive Rot : list A -> list A ->
Prop :=
| rotSame : forall (l : list A), Rot l l
| rotNext :
forall (l l' : list A) (a : A),
Rot l (a :: l') -> Rot l (l' ++ [a]).
```

これは 2 つの Circuit が回転させると一致する関係にあることを表す述語である．これを用いて，Circuit の包含関係も定義する．

```
Definition RIn
(c : Circuit)(C : list Circuit) :=
EqIn Rot c C.
```

3.2.6 Area

Area の定義は以下の通りである．

```
Definition Area := list Circuit.
```

これに Area に対する制約条件を追加する．

3.2.7 Area-constraints

まず，(1) 式の Point 接続と Line 接続を Coq で定義する．

```
Definition point_connect
(c1 c2 : Circuit) :=
 exists l1 : Line, exists l2 : Line,
 exists p : Point,
 In l1 c1 /\ In l2 c2 /\
 InPL p l1 /\ InPL p l2.
Definition line_connect
(c1 c2 : Circuit) :=
 exists l : Line,
 In l c1 /\ In (reverse l) c2.
```

これらを用いて，Area_constraintsを定義する．

```
Definition Area_constraints(a : Area) :=
 (forall(c1 c2 : Circuit),
 In c1 a -> In c2 a -> c1 <> c2
 -> ~point_connect c1 c2 /\
    ~line_connect c1 c2)
 /\
 (forall(c : Circuit),
 ~RIn c (set_remove eq_circuit_dec c a))
 /\
 1 <= length a.
```

これは 3 つの論理式が論理積によって結合している形をしている．そのうち左から 1 番目の論理式は Area 内の任意の 2 つの Circuit が Point または Line で接続していないことを表しており，2 番目の論理式は Area に含まれる Circuit に重複する要素がないことを表している．最後の論理式は Area が最低でも 1 つの Circuit を含むことを表している．

なお，Circuit に対して同値類を定義しているため，重複する要素を含まないことを，任意の要素をリストから削除するとその残ったリストは削除した要素の同値類を含まないと定義した．eq_circuit_decは，2 つの引数がそれらが定義されている集合の上で同値かどうかを判定する述語である．ただし，Coq は直観主義論理に基づくため，一般的にこのような述語が決定可能だという保証がない．したがって，まず，この述語の決定可能性を証明した．set_removeはリストの中から同値な要素のペアを見つけて，その中の 1 つだけを削除したリストを返す関数である．

次に，Area の上での同値類を定義する．

```
Inductive eq_area : Area -> Area -> Prop
    :=
| permute_area :
forall(a1 a2 : Area),
   Permutation a1 a2 -> eq_area a1 a2
| rotate_c_area :
forall(a : Area)(c1 c2 : Circuit),
   Rot c1 c2 -> eq_area (c1::a) (c2::a)
| trans_area :
forall(a1 a2 a3 : Area),
  eq_area a1 a2 -> eq_area a2 a3
  -> eq_area a1 a3.
```

permute_areaはリストの要素の入れ替えをした Area は同値であるというものである．rotate_c_areaは Area の中のある Circuit を回転すると一致する関係にある 2 つの Area が同値であるというものである．trans_areaは推移的であることを表している．

最後に，Area の要素とリストの包含関係を定義する．

```
Definition AIn(a : Area)(A : list Area) :
    =
EqIn eq_area a A.
```

3.3 Constraints

2.3 節で定義した PLCA 表現の制約条件の部分を Constraints として Coq で定義した．

```
Definition Lconstraint(L : list Line) :=
forall(l : Line), LIn l L ->
Line_constraint l.

Definition Cconstraint(C : list Circuit)
    :=
forall(c : Circuit), In c C ->
Circuit_constraints c.

Definition Aconstraint(A : list Area) :=
forall(a : Area), In a A ->
Area_constraints a.
```

これを用いて，PLCA 表現の Constraints を PLCAconstraintsと定義する．

```
Definition PLCAconstraints
(L : list Line)(C : list Circuit)
(A : list Area)(o : Circuit) :=
 Lconstraint L /\ Cconstraint C /\
 Aconstraint A /\ Circuit_constraints o.
```

最後の Circuit_constraints oは outermost が Circuit-constraints を満たすことを表している．

3.4 PLCAequivalence

PLCA 表現 $E = \langle P, L, C, A, outermost \rangle$，$E' = \langle P', L', C', A', outermost' \rangle$ が与えられたとき，これらが以下の 6 つの条件のいずれかを満たすとき，E, E' が **PLCA 等価である**といい，$PLCAequivalence(E, E')$ と記述する [*2]．PLCA 等価は以下の 6 つの構成子を使って帰納的に定義される．

なお，ここで用いられる permutation とは 2 つのリストに対して，片方の要素を並び替えるともう片方のリストに一致するような関係を意味している．

permutePLCA

各リストが permutation の関係にある PLCA は元の PLCA と PLCA 等価である．

reverseLine

Lの先頭要素の Line に reverseを適用した PLCA は元の PLCA と PLCA 等価である．

permuteArea

Aの先頭要素の Area を permutation の関係にある Area に置き換えたものを持つ PLCA は元の PLCA と PLCA 等価である．

[*2] E，E' は PLCA 表現なので，その定義から Constraints を満たすが，$(P, L, C, A, outermost), (P', L', C', A', outermost)$ はそれぞれ必ずしも Constraints を満たさないものについても等価性は定義できる．ここでは PLCA 表現に着目して話を進める．

rotateCircuit

Cと Aに対してある Circuit と同値関係にある Circuit 全てに対して回転をした PLCA は元の PLCA と PLCA 等価である．

rotateOutermost

outermost を回転させた PLCA は元の PLCA と PLCA 等価である．

PLCAtrans

A と B が PLCA 等価であり，B と C が PLCA 等価であるとき，A と C は PLCA 等価である．

このように定義することで，例えば，$E = \langle P, L, C, A, outermost \rangle, E' = \langle P', L', C', A', outermost \rangle$ に対して，P と P' が permutation の関係にあり，L と L' がある要素の Line を反転させると一致するような関係にあり，C，A がある Circuit に対して回転したものが C'，A' と一致するような関係であるとき，permutePLCA，reverseLine，rotateCircuit，PLCAtransを組み合わせることで，E，E' は PLCA 等価であると証明できる．

3.5 Consistency

3.5.1 P-L,L-C,C-A,outermost Consistency

2.5 節で示した条件を Coq で記述したコードの一部を示す．P，L，C，Aは引数として外部で定義されているため，実際は必要に応じてこれらの引数が最初に付く．

P-L Consistency

```
Definition PLconsistency : Prop :=
forall(p : Point), In p P ->
(exists l : Line,LIn l L /\ InPL p l).

Definition LPconsistency : Prop :=
forall(p : Point)(l : Line),
In l L -> InPL p l -> In p P.
```

これは (3)，(4) 式を Coq で表現したものであり，ほとんど定義そのままの形で書ける．

3.5.2 Distinct

集合 P，L，C，A をリストとして実装したため，これらがそれぞれ重複した要素を含まないという条件を追加する必要がある．

```
Definition DistinctP : Prop :=
forall(p : Point),
~In p (set_remove eq_point_dec p P).
Definition DistinctL : Prop :=
forall(l : Line),
~LIn l (set_remove eq_uline_dec l L).
Definition DistinctC : Prop :=
forall(c : Circuit),
~RIn c (set_remove eq_circuit_dec c C).
Definition DistinctA : Prop :=
forall(a : Area),
```

```
~AIn a (set_remove eq_area_dec a A).
```

eq_point_dec,eq_line_dec,eq_circuit_dec,eq_area_decはそれぞれ，各オブジェクトのペアが同値かどうかを判定する述語であり，その決定可能性は別途証明している．

3.5.3 PLCAconsistency

これらを全てまとめて，Consistency を以下のように表現した．

```
Definition PLCAconsistency : Prop :=
PLconsistency /\ LPconsistency /\
LCconsistency /\ CLconsistency /\
CAconsistency /\ ACconsistency /\
Outermostconsistency1 /\
Outermostconsistency2 /\
Circuitconsistency /\ Areaconsistency /\
DistinctP /\ DistinctL /\
DistinctC /\ DistinctA.
```

Consistency の定義は各オブジェクト間の Consistency と Distinct の全ての論理積である．煩雑な記述を避けるため，ここでは引数の *P*，*L*，*C*，*A*，*outermost* は省略して示す．

3.6 PLCAconnected

PLCAconnect は引数に 2 つのオブジェクトを取るが，そのオブジェクトは Point，Line，Circuit，Area の 4 種類あり，Line と Point，Line と Circuit，Area と Area など組み合わせは複数存在する．したがって，引数の型が一意に決まらない．全ての型を含む上位型として object 型を用意し，PLCAconnect の引数の型を object 型として一意的に定まるようにした．

```
Inductive object :=
| o_point   : Point   -> object
| o_line    : Line    -> object
| o_circuit : Circuit -> object
| o_area    : Area    -> object.
```

加えて，object 型に対する包含関係を表す述語も定義する．

```
Definition OIn(o : object) : Prop :=
 match o with
| o_point p   => In p P
| o_line l    => In l L
| o_circuit c => In c C
| o_area a    => In a A
 end.
```

これはある object oが *P* か *L* か *C* か *A* のいずれかに含まれることを意味している．すると，PLCAconnect は帰納的に定義することができる．

```
Inductive PLCAconnect :
object -> object -> Prop :=
| PLcon : forall(p : Point)(l : Line),
In p P  -> LIn l L -> InPL p l
-> PLCAconnect (o_point p) (o_line l)
| LCcon : forall(l : Line)(c : Circuit),
In l L  -> In c C  -> LIn l c  ->
   PLCAconnect (o_line l) (o_circuit c)
| CAcon : forall(c : Circuit)(a : Area),
In c C  -> In a A  -> In c a   ->
   PLCAconnect (o_circuit c) (o_area a)
| SYMME : forall(o1 o2 : object),
PLCAconnect o1 o2 -> PLCAconnect o2 o1
| TRANS : forall(o1 o2 o3 : object),
PLCAconnect o1 o2 -> PLCAconnect o2 o3
-> PLCAconnect o1 o3.
```

構成子やそれが意味するものは 2.6.1 節と同じである．

以上により，ある PLCA 表現に出現する全てのオブジェクトが互いに PLCAconnected であることを以下のように表現できる．

```
Definition PLCAconnected : Prop :=
forall(o1 o2 : object),
OIn o1 -> OIn o2 -> PLCAconnect o1 o2.
```

煩雑な記述を避けるため，ここでは引数の *P*，*L*，*C*，*A*，*outermost* は省略して示す．

3.7 PLCAeuler

```
Definition PLCAeuler : Prop :=
length P + 2 * length A
= length L + length C.
```

(13) 式をマイナスを使わないように式変形をしてこのように定義した．なぜならば，Coq は自然数を一般的な数として提供しており，その数式の四則演算も定義されているが，自然数であるためマイナスの演算結果が 0 を下回る場合，正しくない計算をしてしまう．Coq にはマイナスを正しく扱うためのライブラリも提供されているが，ここではマイナスを使わずに数式を記述することでそのような問題を回避した．

3.8 PLCAplanar

2.6.3 節から，以下のように PLCA の平面性を記述することができる．

```
PLCAconsistency /\ PLCAconnected
/\ PLCAeuler
```

煩雑な記述を避けるため，ここでは引数の *P*，*L*，*C*，*A*，*outermost* は省略して示す．

3.9 帰納的 PLCA

帰納的 PLCA を Coq 上で定義する．引数には P，L，C，A，$outermost$ を取る．構成子は 3 つあり，1 つは基礎となる PLCA を表し，1 つはある PLCA に対応する図形があった時にある Area を分割するような操作を行い，最後の 1 つはある PLCA に対応する図形があった時にある Area に孤立した閉曲線を追加する操作を行う．定義は全て記号表現上の処理だが，わかりやすい説明のため図を使って直観的に説明する．

以下がその 3 つの構成子である．

single_loop

これは基礎となる PLCA に対応する図形を表している．1 つの閉曲線からなる単純な図形である．

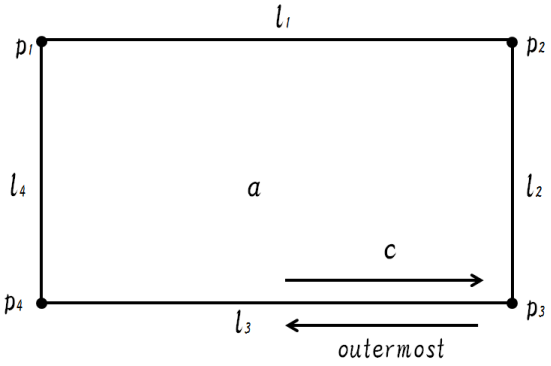

図 **11** single_loop

$$E = \langle P, L, C, A, outermost \rangle$$
$$P = \{p_1, p_2, p_3, p_4\}$$
$$L = \{l_1, l_2, l_3, l_4\}$$
$$C = \{c, outermost\}$$
$$A = \{a\}$$

$l_1 = (p_1, p_2)$　$outermost = [l_1^+, l_2^+, l_3^+, l_4^+]$　$a = \{c\}$
$l_2 = (p_2, p_3)$　$c = [l_1^-, l_4^-, l_3^-, l_2^-]$
$l_3 = (p_3, p_4)$
$l_4 = (p_4, P_1)$

図 11 の記号表現

図 11 では四角形で表現しているが，四角形である必要はなく 3 つ以上の Point からなる環状構造になっていればよい．出現するオブジェクトは Area が内側の空間の a，Circuit がその閉曲線の内と外の c と $outermost$ である．Line と Point はこの閉曲線に含まれるものだけである．

add_path

add_path は Area を分割することで新しい PLCA を得るためのオペレータである．ある Circuit を指定し，その中の 2 つの Point 間を結ぶパスを作る．パスの長さは任意であり，間にいくつ Point を取っても構わない．まず，パスを表す simplepathを示す．

```
Inductive simplepath :
Point -> Point -> list Point ->
list Line -> Prop :=
| nil_path  : forall(p : Point),
simplepath p p (p::nil) nil
| step_path :
forall(p1 p2 p3 : Point)
(pl : list Point)(ll : list Line),
   simplepath p2 p1 pl ll
-> ~In p3 pl
-> simplepath p3 p1
(p3::pl) ((p3, p2)::ll).
```

simplepathは Trailと似ているが，Trailが Point の重複を許すのに対して，simplepathは Point の重複を許す点が異なる．

次に，add_path によって作られる PLCA の例を示す．

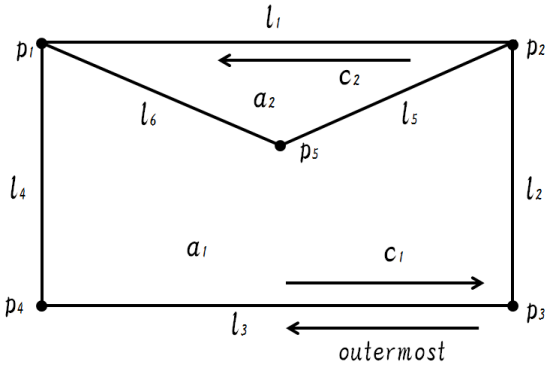

図 **12** single_loop → add_path

$$E = \langle P, L, C, A, outermost \rangle$$
$$P = \{p_1, p_2, p_3, p_4, p_5\}$$
$$L = \{l_1, l_2, l_3, l_4, l_5, l_6\}$$
$$C = \{c_1, c_2, outermost\}$$
$$A = \{a_1, a_2\}$$

$l_1 = (p_1, p_2)$　$outermost = [l_1^+, l_2^+, l_3^+, l_4^+]$　$a_1 = \{c_1\}$
$l_2 = (p_2, p_3)$　$c_1 = [l_5^+, l_6^-, l_4^-, l_3^-, l_2^-]$　$a_2 = \{c_2\}$
$l_3 = (p_3, p_4)$　$c_2 = [l_1^-, l_6^+, l_5^-]$
$l_4 = (p_4, P_1)$
$l_5 = (p_2, p_5)$
$l_6 = (p_1, p_5)$

図 12 の記号表現

図 12 は single_loop に add_path を適用した後を表している．つまり，図 11 に add_path を適用した図が図 12 である．オブジェクトの変化を記号表現で見ると，

Circuit c が c_1 へ変化し，c_2 が新しく追加されている．さらに *Area a* が a_1 に変化し，a_2 が追加されている．Line と Point は追加されたパスを構成するものだけ追加される．

add_loop

add_loop は add_path と同じく，新しい PLCA を得るためのオペレータである．PLCA に孤立した閉曲線を追加することで新しい PLCA を得る．

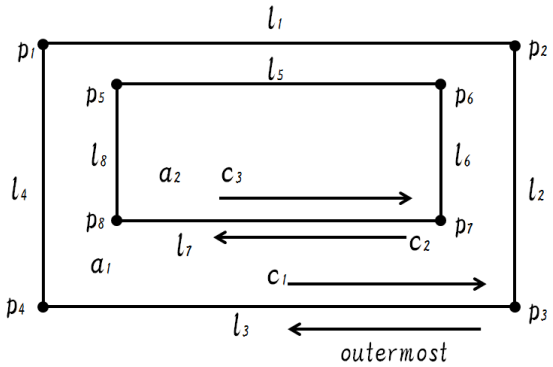

図 **13** single_loop → add_loop

$$E = \langle P, L, C, A, outermost \rangle$$
$$P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$$
$$L = \{l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8\}$$
$$C = \{c_1, c_2, c_3, outermost\}$$
$$A = \{a_1, a_2\}$$

$l_1 = (p_1, p_2)$　$outermost = [l_1^+, l_2^+, l_3^+, l_4^+]$
$l_2 = (p_2, p_3)$　$c_1 = [l_1^-, l_4^-, l_3^-, l_2^-]$
$l_3 = (p_3, p_4)$　$c_2 = [l_5^+, l_6^+, l_7^+, l_8^+]$
$l_4 = (p_4, P_1)$　$c_3 = [l_5^-, l_8^-, l_7^-, l_5^-]$
$l_5 = (p_5, p_6)$　$a_1 = \{c_1, c_2\}$
$l_6 = (p_6, p_7)$　$a_2 = \{c_3\}$
$l_7 = (p_7, p_8)$
$l_8 = (p_8, p_5)$

図 13 の記号表現

図 13 は single_loop に add_loop を適用した後を表している．つまり，図 11 に add_loop を適用した図が図 13 である．オブジェクトの変化を記号表現で見ると，*Circuit* c_3 と c_4 が新しく追加され，*Area a* が a_1 に変化し，新しく *Area* a_2 が生成されている．Line と Point は追加された閉曲線を構成するものだけ追加される．

帰納的 PLCA(IPLCA)　single_loop，add_path，add_loop を用いて帰納的に作られる PLCA E を**帰納的 PLCA(IPLCA)** といい，*Inductive*(E) と記述する．

帰納的 PLCA の構成方法は 1 つの閉曲線からはじめてその内側に線を引くことによって対象となる図形を描くことに相当する．

例えば，図 14 を考える．この図の描き方は図 15，図 16 の 2 通りあり，それぞれ対応する PLCA 表現上では構成子の適用順序が異なる．

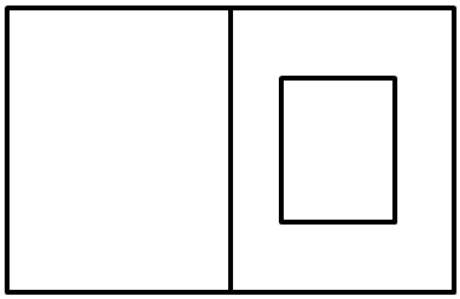

図 **14**　複数の構成方法がある図形

図 **15**　構成方法その 1

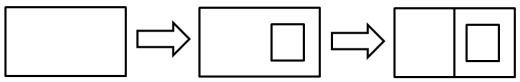

図 **16**　構成方法その 2

図 15 は，single_loop → add_path → add_loop という手順であり，図 16 は，single_loop → add_loop → add_path という手順である．結果として得られる PLCA 表現は P，L，C，A のリストの並びが変わるだけで，PLCA 等価である．

次に図 17 を考える．

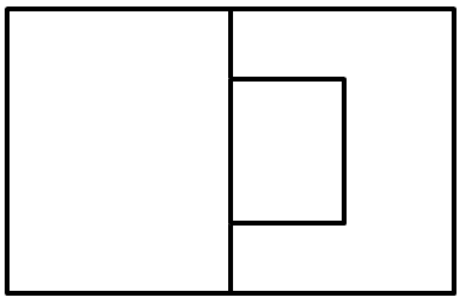

図 **17**　構成方法が 1 つしかない図形

この図の描き方は図 18，図 19 の 2 通りあるように思われる．

図 18 ただ 1 つの構成方法

図 18 は single_loop → add_path → add_path という手順である．

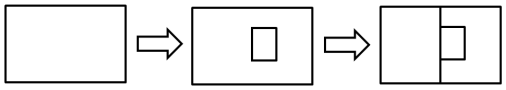

図 19 間違った構成方法

しかし，図 19 に示す手順では構築できない，single_loop に add_loop を適用することで中に別の領域を追加し，残りの部分を add_path を適用することで接続できるように見えるが，add_path は同じ Circuit 内の 2 つの Point を指定しないといけないため，ここでは適用することができない．

PLCA 図形の中に，構築手順が複数あるものと 1 つしかないものがあるのが帰納的 PLCA の特徴である．これは帰納的 PLCA 上で証明を行ったり，構成子以外のオペレータを定義する場合に留意しなければならない点である．

4. 証明

$$
\begin{aligned}
&S = \{(P, L, C, A, outermost) \mid \\
&P\ is\ a\ set\ of\ Point, L\ is\ a\ set\ of\ Line, \\
&C\ is\ a\ set\ of\ Circuit, A\ is\ a\ set\ of\ Area\}
\end{aligned} \tag{14}
$$

として以下を説明する．

$E \in S$ に対して 3.3 節で定義された Constraints を満たすとき，$Constraints(E)$ と記述する．

4.1 PLCAequivalence の満たす性質

PLCA 等価 (PLCAequivalence) は PLCA の主たる性質である Constraints，Consistency，PLCAconnected，PLCAeuler を保持する．これらの性質はいずれも PLCAequivalence に対する帰納法によって，証明することができる．例えば，Constraints を保持することについては以下の式で表される．

$$
\begin{aligned}
&\forall E, E' \in S, \\
&Constraints(E) \wedge PLCAequivalence(E, E') \\
&\Rightarrow Constraints(E')
\end{aligned} \tag{15}
$$

この性質は約 300 行で証明が完了した．

他の性質についても同様に表される．Consistency の場合の証明が約 500 行，PLCAeuler の場合の証明が約 50 行で完了した．

4.2 平面的 PLCA と帰納的 PLCA の一致

3.9 節で示した帰納的 PLCA のクラスが平面的 PLCA のクラスと一致することを証明する．具体的には，任意の平面的 PLCA に対してそれと PLCA 等価な帰納的 PLCA が存在することを証明する．2.6.3 節で示した PLCA の平面性条件から証明すべき性質は以下のように表現できる．

ここで，$Consistency(E)$ は PLCA 表現に対して定義されている述語だが，S の要素である E に対して定義を拡張したものとする．$PLCAeuler(E)$，$PLCAconnected(E)$ についても同様とする．

$$
\begin{aligned}
&\forall E \in S, \\
&(\ Constraints(E) \wedge Consistency(E) \\
&\wedge PLCAeuler(E) \wedge PLCAconnected(E) \\
&\Leftrightarrow \\
&(\exists E' \in S, Inductive(E') \wedge \\
&PLCAequivalence(E, E')\ )
\end{aligned} \tag{16}
$$

4.2.1 帰納的 PLCA の平面性

まず，(16) 式の左方向について証明する．帰納的 PLCA は平面的 PLCA である，すなわち帰納的 PLCA が PLCA の平面性条件および Constraints を満たすことをそれぞれ証明する．

PLCAconstraints，PLCAconsistency

```
Theorem IPLCA_satisfy_consistency_
and_constraints :
forall(P : list Point)(L : list Line)
(C : list Circuit)(A : list Area)
(o : Circuit),
   I P L C A o
-> PLCAconsistency P L C A o /\
    PLCAconstraints L C A o.
```

この 2 つの条件は切り離して証明しようとしたが，帰納法の帰納段階においてお互いの仮定を必要としたため，この 2 つを同時に証明をした．証明の行数は約 3000 行になった．

PLCAconnected

```
Theorem IPLCA_satisfy_PLCAconnected :
forall(P : list Point)(L : list Line)
(C : list Circuit)(A : list Area)
(o : Circuit),
I P L C A o -> PLCAconnected P L C A.
```

この条件を証明するためには，帰納的 PLCA の 3 つの構成子のうち single_loop が PLCAconnected を満たすことを証明し，かつ add_path，add_loop を適用したときに PLCAconnected を保存しているかを調べる必要がある．後者の部分は補題に分割し PLCAconnect に対する帰納法によって証明することでできた．証明の行数は全体で約 1500 行になった．

PLCAeuler

```
Theorem IPLCA_satisfy_PLCAeuler :
forall(P : list Point)(L : list Line)
(C : list Circuit)(A : list Area)
(o : Circuit),
I P L C A o -> PLCAeuler P L C A.
```

PLCAeuler はリストの要素数だけを見るため，帰納的 PLCA のオブジェクト同士の関係性を考える必要がなく，数式の変形だけで証明を完成させることができる．また，Coq には omegaという数式を自動証明することに特化したコマンドがあるため，これを用いることで補題を含めても約 100 行に満たないほどの小さな証明になった．

いずれも帰納的 PLCA に対する帰納法によって証明することができた．

4.2.2 平面的 PLCA が帰納的 PLCA であること

次に (16) 式の右方向について証明する．証明すべき式は，以下の式である．

$$
\begin{aligned}
&\forall E \in S, \\
&(\mathit{Constraints}(E) \wedge \mathit{Consistency}(E) \qquad (17)\\
&\wedge \mathit{PLCAeuler}(E) \wedge \mathit{PLCAconnected}(E)) \\
&\Rightarrow \\
&(\exists E' \in S, \mathit{Inductive}(E') \wedge \mathit{PLCAequivalence}(E, E'))
\end{aligned}
$$

この式は平面性条件を満たす PLCA と PLCA 等価なものが存在して，それが帰納的 PLCA で構築できるということを表している．

帰納的 PLCA を構築する際に 1 つの構成子を適用するたびにオブジェクトの数が増えるので，オブジェクトに対して帰納法を適用して証明したい．構成子を適用するたびに，1 つずつ数が増えるオブジェクトが Area のみであるため，証明は Area の数 $|A|$ に対する帰納法を用いて行う．

証明の方針は以下の通りである．まず，PLCA の平面性条件に，適切な方法で Area の数を 1 つ減らすことで Area の数が 1 つ少ない平面性の条件を得る．そこに帰納の仮定を適用することで Area の数が 1 つ少ない帰納的 PLCA を得る．そして最後に，add_path，add_loop で Area の数を 1 つ増やすことで証明を行う．Area の減らし方には 2 通りのものが存在し，それぞれの場合において証明を完成させる必要がある．4.1 節より PLCAequivalenceが平面性に必要な条件を保存することが証明されているので，この性質を利用する．以下に証明のアウトラインを示す．

まず，$|A|$ を $|A| = 0$ か $1 \leq |A|$ で場合分けを行う．

$|A| = 0$ の場合は，Consistency から以下のように矛盾を導くことができる．

(1) (11) 式によって，C は少なくとも *outermost* を含む．
(2) (6) 式によって，outermost を構成する *Line* l が L に含まれているはずである．
(3) (5) 式によって，outermost とは逆方向の *Line* $l^{\vdash}$ を含む *Circuit* c が存在するはずである．
(4) (8) 式によって，その *Circuit* c は outermost ではないため，c を含むような *Area* a が存在するはずである．これは，$|A| = 0$ の仮定と矛盾する．

$1 \leq |A|$ の場合は，さらに $|A|$ に対する帰納法によって証明を行う．

$|A| = 1$ の場合

この場合は，1 つの閉曲線だけからなる単純な図形しか考えられない．したがって，帰納的 PLCA は構成子 single_loop によって構築できる．

$|A| = n$ $(2 \leq n)$ を仮定して，$|A| = n + 1$ を証明する場合

帰納の仮定は，$|A| = n$ のときに平面性条件と Constraints を満たすならば，その PLCA が帰納的に構築されるということである．これを利用するためには，$|A| = n$ のときの平面性条件と Constraints を満たす必要がある．そのためには，証明に出現する $|A|$ を 1 つだけ減らせばよい．ただし，平面性条件を崩すような減らし方はできない．例えば，図 19 を考える．この 3 番目の図では Area の数は 3 個，2 番目の図では 2 個である．しかし，このような減らし方をすると，得られた PLCA 表現は平面性条件を満たさない．正しい手順によって，A から特定の要素を削除し，それに合わせて影響を受けたオブジェクトを P，L，C から削除または追加する．以上の手続きによって，ある PLCA 表現から Area の数を 1 つ減らした PLCA 表現を生成するような関数 MergeArea_Function を定義する．内部にオブジェクトがないような *Area* a_1 を 1 つとり，a_1 に対して関数 MergeArea_Function を適用することで，a_1 と a_1 に隣接する Area の合成を行う．内部にオブジェクトのない Area が存在することは以下の補題で証明している [14]．

(補題 1) 平面的 PLCA において，内部にオブジェクトがないような Area，つまりそれがただ 1 つの Circuit のみを含むような Area が必ず 1 つ以上は存在する．

MergeArea_Function は a_1 の外側の Circuit の状態がどのようになっているかによって場合分けして定義する．

まず，外側にある Circuit がただ 1 つしかない場合を考える．

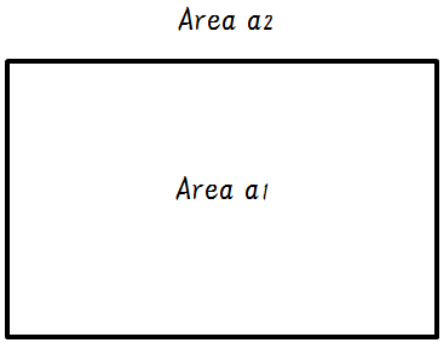

図 20 外側の Circuit がただ 1 つしかない場合

この場合，図 20 に示すような孤立した閉曲線であると特定できる．したがって，閉曲線を削除することで $|A|$ を 1 つ減らすことができる．
次に外側にある Circuit が 2 個以上である場合を考える．

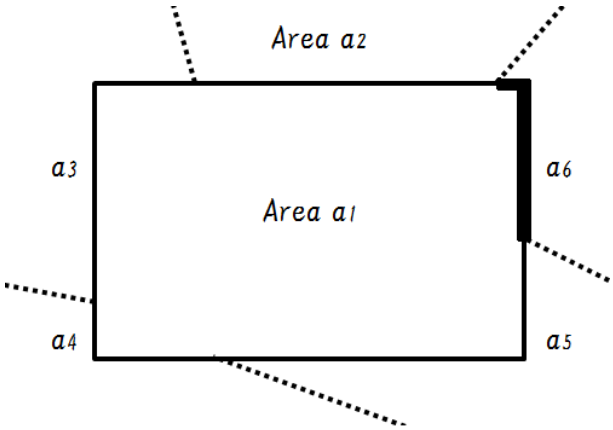

図 21 外側の Circuit がただ 1 つではない場合の例

この場合, 図 21 に示すように，*Area* a_1 の内側の 1 つの Circuit と複数の外側の Circuit がそれぞれの複数の Line を共有している状態であり，ある 1 つの Area との共有部分を削除することで，$|A|$ を 1 つ減らすことができる．例えば，図 21 の太線がそのような部分であり，太線を除去すると a_1 と a_6 が併合されて 1 つの Area になる．
MergeArea_Function を以上のように定義して，適用すると $|A| = n$ である PLCA 表現 E' を得る．E' が帰納の仮定である $|A| = n$ のときの平面性の条件と PLCA-constraints を満たすと証明できるので，帰納の仮定を適用できる．その結果，$|A| = n$ の場合に以下の式が成り立つ．

$$\exists E' \in S,$$
$$Inductive(E') \wedge PLCAequivalence(E, E') \qquad (18)$$

最後に，add_path，add_loop のどちらかを適用することで $|A|$ を 1 つ増やす．どちらを適用するかは，MergeArea_Function を適用した際にどのように Area を削除したかに依存する．閉曲線を削除した場合は，add_loop によって元の PLCA を復元する．また，共有している複数の Line を削除した場合は，add_path によって $|A| = n + 1$ である元の PLCA を復元する．帰納の仮定を利用することで，証明すべき式の結論が元の PLCA に対して成り立つことがいえる．

5. 議論

証明を行う際に，問題になった部分について詳しく述べる．

5.1 PLCAequivalence が満たす性質の証明

PLCAequivalence に関して PLCAconnected が保存されるという性質について証明を試みると予想以上に証明が複雑になり，証明を完成させることはできなかった．証明は PLCAequivalence に対する帰納法を使い，各構成子で場合分けして進める．どの構成子を適用してもその結果得られるものが PLCAconnected を満たすことを，すべての場合に対して証明する必要がある．このとき構成子の適用の前後で型が変わることがあるため，証明すべき PLCAconnect の型が変化してしまう点に注意しなければならない．例えば，Line 型の変数 `l`に対して，関数 `reverse`を適用することがあり，その場合は `reverse l`となり型が変化してしまう．したがって，予想以上の数の補題が必要となり，証明が困難になった．

5.2 平面的 PLCA が帰納的 PLCA である証明

証明の方針は 5 節で示した通りである．まず，補題 1 については，手証明はされている [14] ものの，形式化自体を検討する必要があり，Coq による形式化はまだできていない．さらに，平面的 PLCA が帰納的 PLCA であることの証明の中で Consistency を保持するように PLCA 等価な 2 つの PLCA 表現から共に複数のオブジェクトを削除，追加する必要があったが，このような操作をしても PLCA 等価性が保持されることの証明も未完成である．

6. 結論

定性空間表現 PLCA に対して帰納的に構築する方法について示し，それが平面性を満たすことを証明した．形式化は Coq を用いて行い，順方向の証明では Coq による証明が完了した．これによって，基礎となる図形から 2 つの操作によって作られる図形は平面に必ず描画できるという結論

を得ることができた．また，平面に描画できる PLCA が帰納的 PLCA で構築できる証明の戦略を示すことができた．また，PLCA の等価性についても形式的な定義を与え，その性質として PLCAconnect を除いて証明が完了している．証明は最後まで完了していない部分もあるが，形式化を行うことで，PLCA を実装をする場合に定義に不十分なものがあることもわかった．

本研究は，これまで妥当性についてほとんど言及されていなかった定性空間表現に対して計算可能なモデルを与えた初めての試みといってよい．また，定理証明の研究分野においては，定性空間推論という，これまで応用のなかった分野に対して新たな分野を開拓したといえる．

今後の課題は，残ったいくつか証明を完成させて証明を完全なものにすることや，先行研究で行われていた机上の証明を定理証明器で厳密な証明を与えることなどが挙げられる．

参考文献

[1] G. Ligozat. : *Qualitative Spatial and Temporal Reasoning*, wiley, (2011).

[2] J. Renz. : *Qualitative Spatial Reasoning with Topological Information*, LNAI-2293, Springer-Verlag, (2002).

[3] G.A. Cohn and J. Renz. : *Qualitative spatial reasoning,* Handbook of Knowledge Representation. F. Harmelen, V. Lifschitz and B. Porter(eds.), Chapt 13, pp.551-596, Elsevier, (2007).

[4] A.D. Randell, G.A. Cohn and Z. Cui. : Computing Transitivity Tables: A Challenge for Automated Theorem Provers, *Proceedings of Automated Deduction (CADE-11).*, LNCS-607, pp.786-790, Springer, (1992).

[5] Y. Bertot and P. Casteran. : *Interative Theorem Proving and Program Development*, Springer， (2004).

[6] T. Nipkow， L.C. Paulson，and M. Wenzel. : *Isabelle/HOL A Proof Assistant for Higher-Order Logic*, Springer，(2008).

[7] J. Crow, S. Owre, J. Rushby, N. Shankar and M. Srivas. : *A Tutorial Introduction to PVS*, (1995)

[8] M. Kaufmann and J.S. Moore. : *ACL2-Tutorial*, (1997).

[9] U. Norell. : *Dependently Typed Programming in Agda.*, Advanced Functional Programming, (2008).

[10] G. Georges. : Formal proof—the four-color theorem, *Notices Amer. Math. Soc. 55, no.11, pp.1382-1393*， (2008).

[11] X. Leroy. : Formal verification of a realistic complier, *Communication of the ACM. 52, no.7, pp.107-115*, (2009)

[12] K. Takahashi. : PLCA: A Framework for Qualitative Spatial Reasoning Based on Connection Patterns of Regions, *Qualitative Spatio-Temporal Representation and Reasoning: Trends and Future Directions, S.Hazarika(ed.), Chapt 2, pp.63-96, IGI Publishers*, (2012).

[13] K. Takahashi, T. Sumitomo and I. Takeuti. : On Embedding a Qualitative Representation in a Two-Dimensional Plane, *Spatial Cognition & Computation: An Interdisciplinary Journal Volume 8, Issue 1-2*, (2008).

[14] K. Takahashi, M. Goto and H. Miwa. : A Qualitative Representation of a Figure and Construction of Its Planar Class, *7th International Conference on Agents and Artificial Intelligence*, (2015).