Boolean Gröbner基底を用いた並行分散プログラムの整合性検査

Consistency Checking of Distributed Programs via Boolean Gröbner Bases

京都大学理学研究科 数学教室　　西村 進 [*1]
Susumu Nishimura
Department of Mathematics, Kyoto University

Abstract

This paper studies an application of Boolean Gröbner bases to the verification of distributed programs. In the topological theory of distributed computing, distributed programs are modeled as a composition of topological maps over simplicial complexes. This leads to a declarative programming style, wherein the topological maps are defined as functions over simplicial complexes. However, the resulting function must be checked for its topological consistency. This consistency checking process corresponds to solving certain set constraints, due to the combinatorial nature of the simplicial complex model of distributed computing. In this paper, we present a consistency checking procedure that detects possible inconsistencies by employing Inoue's singleton set constraint solving algorithm for boolean polynomial rings with singleton parameters. We have also implemented a prototype consistency checker and discuss certain optimization methods to achieve better efficiency.

1　はじめに

並行分散計算の組合せトポロジー論に基づいてプログラムを構成したときのプログラムの幾何的整合性検査を Boolean Gröbner 基底 [SIS+11] を用いて行う方法を提案する．並行分散計算の組合せトポロジー論 [HKR13] とは，組合せトポロジーにおける単体的複体の概念を用いて分散計算の性質を明らかにしようとする分野である．この分野の重要な成果として，並行分散計算は単体的複体上の写像の合成として必ず表現できることが知られている．[HS99] この事実から，並行分散プログラミングを単体的複体上の写像を記述することで実現する方法が考えられるが，このような形でプログラミングを行なった場合，その定義が幾何的な整合性を満たしていることの保証が必要不可欠である．

本稿では，上記のような並行分散計算の幾何的整合性が集合制約に対応し，その集合制約を Boolean Gröbner 基底アルゴリズムを用いて解くことによって整合性検査器を構成できることを示す．さらに，この整合性検査器のプロトタイプ実装を通してこの手法を実用的な効率で動作させるためのいくつかの最適化手法について議論する．

本稿の構成は以下のとおりである．第 2 節では，並行分散計算の組合せトポロジー論に基づくプログラム記述とそれがみたすべき幾何的整合条件を示し，第 3 節で Boolean Gröbner 基底アルゴリズムによる既存の単元集合制約解消手法についてその概略を述べる．第 4 節において，集合制約の生成とその解消による並行分散プログラムの整合性検査アルゴリズムを示す．このアルゴリズムのプロトタイプ実装を通して得られた最適化手法に関する知見について第 5 節で議論する．最後に，第 6 節で結論と将来の展望を述べる．

[*1] 〒 606–8502 京都市左京区北白川追分町　E-mail: susumu@math.kyoto-u.ac.jp

2　並行分散システムの組合せトポロジー論と関数型プログラミング

2.1　並行分散システムの組合せトポロジー論

並行分散システムの組合せトポロジー論について，本稿に関連する部分を抜粋する．より詳細については [HKR13] を参照されたい．

$n+1$ 個の並行プロセスからなる並行分散システムを考える．各並行プロセスはそれぞれ一意なプロセス識別子で区別され，これら識別子の集合は $\{0,\dots,n\}$ であるとする．以下では，これら $n+1$ 個の並行プロセスが Read-Write 共有分散メモリを介してプロセス間通信を行うことによって実現できる wait-free な並行分散アルゴリズム [GK18] のみを考える．

組合せ幾何における (抽象) 単体的複体は，V を頂点の集合としたとき以下のように通常定義される．

単体 $d+1$ 個の頂点の集合 $\{v_0,\dots,v_d\}\subseteq V$ を (d 次元) 単体という．単体 σ の次元を $\dim(\sigma)$ で表す．

複体 単体の有限集合 $\mathcal{C}\subseteq 2^V$ であって集合の包含関係について下に閉じているもの，すなわち $\sigma\in\mathcal{C}$ かつ $\sigma'\subseteq\sigma$ ならば $\sigma'\in\mathcal{C}$ であるものを複体という．[1] 複体 $\mathcal{C}$ の次元を $\dim(\mathcal{C})=\max_{\sigma\in\mathcal{C}}\dim(\sigma)$ で定める．$d+1$ 個の頂点の集合 $\{v_0,\dots,v_d\}\subseteq V$ を (d 次元) 単体という．$V(\mathcal{C})=\bigcup_{\sigma\in\mathcal{C}}\sigma$ で $\mathcal{C}$ が含む頂点の集合を表す．

単体 $\sigma\in\mathcal{C}$ が $\mathcal{C}$ の極大な単体 (すなわち，$\tau\in\mathcal{C}$ かつ $\sigma\subseteq\tau$ ならば $\tau=\sigma$) であるとき，σ を $\mathcal{C}$ の**ファセット**という．

各頂点 v に対して対応するプロセスの識別子を $\chi(v)$ で表すこととしたとき，$\chi(v)$ を頂点 v の色といい，$\chi: V\to\{0,\dots,n\}$ を彩色関数という．プロセスの識別子が一意であることから，任意の単体 σ およびその頂点 $u,v\in\sigma$ について，$\chi(u)=\chi(v)$ ならば $u=v$ である．この性質を満たす単体のことを色付き単体という．色付き単体のみからなる複体を色付き複体といい，その色集合を $\chi(\mathcal{C})=\bigcup_{\sigma\in\mathcal{C}}\chi(\sigma)$ で定める．また，$\mathcal{C}$ のすべてのファセットの次元が一致するとき，$\mathcal{C}$ は pure な複体であるという．

並行分散システムのモデルは，pure な n 次元色付き複体で与えられる．ここで，頂点は各プロセスの状態に，n 次元単体は $n+1$ プロセス並行分散システムのあるひとつの実行状態に，複体は並行分散システムの非決定的実行状態 (システムのとり得る状態すべての集合) に対応する．また，複体が pure かつ n 次元色付きであることは，並行分散システムの非決定的な状態のひとつひとつがプロセス識別子で区別された $n+1$ 個の並行プロセスの状態で定まることと対応する．以降，複体と言えば pure な n 次元色付き複体のことを指すとする．

以下の定理は，並行分散システムの組合せトポロジー論でもっとも中核的な結果として知られている事実である．

定理 1 (Asynchronous Computability Theorem (ACT)[HS99])
ある並行分散計算が，並行分散システムにおいて *wait-free* 実現可能なことと，その並行分散計算が標準色付き細分 $\mathrm{Chr}^m:\mathcal{I}\to\mathrm{Chr}^m\mathcal{I}$ および 色付き単体写像 $\delta: V(\mathrm{Chr}^m\mathcal{I})\to V(\mathcal{O})$ のふたつの複体上の写像の合成 $\delta\circ\mathrm{Chr}^m$ で表現できることとは同値である．

Chr^m は入力複体 $\mathcal{I}$ を特定の方法で細分する操作を m 回繰り返して得られる複体を返す写像である．単体写像 δ は各頂点 $v\in V(\mathrm{Chr}^m\mathcal{I})$ を $\delta(v)\in V(\mathcal{O})$ に写す細分 $\mathrm{Chr}^m\mathcal{I}$ から出力複体 $\mathcal{O}$ への写像で，単体および色を保存するもの，すなわち 任意の $\sigma\in\mathrm{Chr}^m\mathcal{I}$ および $v\in V(\mathrm{Chr}^m\mathcal{I})$ について $\delta(\sigma)\in\mathcal{O}$ かつ $\chi(\delta(v))=\chi(v)$ をみたすものである．

標準色付き細分は，以下の組合せ表現との対応関係で特徴づけられる．

[1] この性質は，並行分散システムのどのプロセスも故障してシステムから離脱する可能性があることに対応するが，本稿ではプロセス故障については考えなくてよい．

定理 2 ([Koz12])
複体 $\mathcal{C}$ の標準色付き細分 $\mathrm{Chr}^m \mathcal{C}$ の各ファセットと，$\{0,\ldots,n\}$ の *ordered set partition* $[S_1;\ldots;S_r]$ は *1* 対 *1* の対応関係を持つ．ただしここで，*ordered set partition* $[S_1;\ldots;S_r]$ とは，$\bigcup_{i=1}^{r} S_i = \{0,\ldots,n\}$ をみたすような $\{0,\ldots,n\}$ の非空な部分集合 $S_1,\ldots,S_r$ の互いに素な列 (並び順を区別する) のことである．

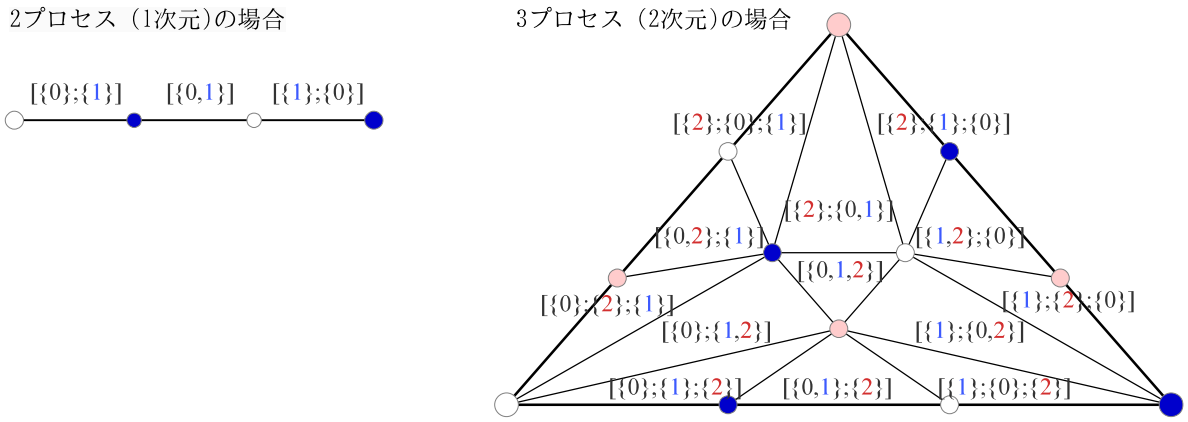


図 1: 標準色付き細分の例

標準色付き細分を 1 次元および 2 次元の単体に対して一回だけ適用すると，図 1 に示す様にそれぞれ 3 個および 13 個のファセットに細分される．

2.2 組合せトポロジー論に基づく関数型並行分散プログラミング

定理 1 により，すべての wait-free 実現可能な並行分散計算は標準色付き細分 Chr^m および色付き単体写像 δ の合成で表せる．このうち，標準色付き細分は immediate snapshot と呼ばれる特定の並行分散プロトコルによる計算手続き [GK18] で実現できる．(ただし本稿では，$m = 1$ の場合のみ取り扱う．) したがって，標準色付き細分は所与のものとして，残りの色付き単体写像 δ をプログラミングすることによって目的とする並行分散計算を定義することになる．

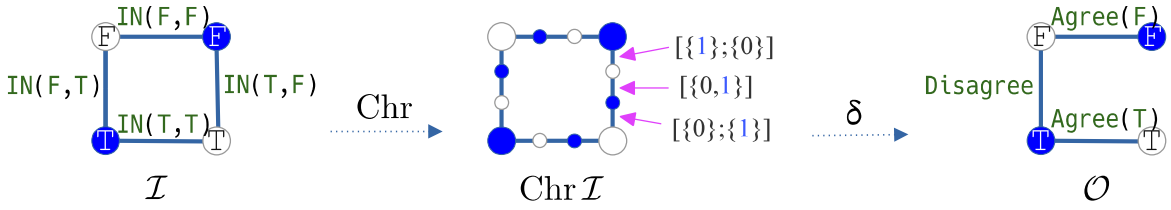
単体写像はその名の通り関数であるから，関数型プログラミングと相性が良いと考えられる．また，単体写像は通常，頂点から頂点への写像として規定されるが，[GLR21] で示されている様に，これをファセットからファセットへの写像として規定しても等価な定義となることが知られている．この事実と，ファセットと ordered set partition が 1 対 1 対応であること (定理 2) より，関数型プログラミングで多用されるパターンマッチ構文を利用して関数を定義することとする．関数は以下の様な形式のパターンマッチ構文

| 入力ファセットのパターン，ordered set partition のパターン -> 出力ファセット

を 1 個以上並べて表す．

例として 2 プロセスの 2 値 quasi-consensus 問題のプログラミングを考えよう．quasi-consensus 問題とは，ふたつのプロセスがそれぞれ T,F いずれかの値を入力とするとき，計算結果として両プロセスとも同一の値 (T もしくは F) を出力とするか，あらかじめ定められた一方のプロセスが T，もう一方が F を出力 (逆は認めない) を出力とするような分散プログラムを与えよ，というものである．

この問題を単体的複体の写像として表現すると以下の様になる．



入力複体 $\mathcal{I}$ は 2 つのプロセスが独立に 2 種類の値を入力とする可能性があるので四辺形である．これの標準色付き細分 Chr $\mathcal{I}$ は四辺形の各辺を 3 つに細分したものになる．出力複体 $\mathcal{O}$ は四辺形から縦の辺を一本除いた 3 辺からなる逆コの字である．

quasi-consensus を実現する単体写像 δ はパターンマッチ構文を用いて以下のようにプログラミングできる．

```
1: let δ = function
2:   | IN(F,F), [_] -> Agree(F)
3:   | IN(F,T), [_] -> Disagree
4:   | IN(T,T), [_] -> Agree(T)
5:   | IN(T,F), [{1};{0}] -> Agree(F)
6:   | IN(T,F), [{0,1}] -> Disagree
7:   | IN(T,F), [{0};{1}] -> Agree(T)
```

例えば，プログラム 2 行目のパターン構文は `IN(F,F)` が指し示す入力複体 $\mathcal{I}$ のファセット (四辺形の上辺) の細分で，パターン `[_]` に合致する ordered set partition に対応するファセットを `IN(F,F)` が指し示す出力複体 $\mathcal{O}$ のファセット (逆コの字の上辺) に写すことを定義している．この場合，パターン `[_]` は任意の ordered set partition に合致するので，細分による 3 つのファセットはすべて同じ出力ファセットに写される．プログラム 5–7 行は，`IN(F,F)` が指し示す入力複体 $\mathcal{I}$ のファセット (四辺形の右縦辺) の細分をそれぞれどの出力ファセットに写すかを定義している．細分による 3 つのファセットは，図中の上から順に ordered set partition `[{1};{0}]`, `[{0,1}]`, `[{0};{1}]` が対応し，それぞれ出力ファセットの上辺 `Agree(F)`, 縦辺 `Disagree`, 下辺 `Agree(T)` に写される．

2.2.1 分散プログラムの幾何的整合性条件

プログラムで定義する写像 δ は，以下の幾何的整合性条件を満たしていなければならない．

単体写像の幾何的整合条件 写像 $\delta : V(\mathcal{C}) \to V(\mathcal{O})$ が単体写像として幾何的に整合しているとは，以下の条件を満たすことをいう．

$$\text{任意の } \mathcal{C} \text{ のファセット } \sigma_1, \sigma_2 \text{ について,} \chi(\sigma_1 \cap \sigma_2) \subseteq \chi(\delta(\sigma_1) \cap \delta(\sigma_2)). \tag{2.1}$$

すなわち，ふたつのファセットがある色 a の頂点を共有するとき，写像 δ によるそれらファセットの像もまた色 a の頂点を共有していなければならない．

プログラムが整合条件を満たしているとは限らないため，個別のプログラム毎に幾何的整合条件の検査が必要である．(与えられたプログラムが整合条件をみたしていない場合，定理 1 は適用外となり並行分散システムでの実現可能性が保証されない．上記の分散 quasi-consensus プログラムはもちろん整合条件をみたしているので，実現可能である．)

なお，整合性条件に加えて写像 δ がすべての入力 ($\mathcal{C}$ のすべてのファセット) について定義されていることを保証するため，プログラムで与えられたパターン集合が網羅的である (どの入力もいずれかのパターンにマッチする) ことを保証する必要があるが，この網羅性の検証については本稿では省略する．(網羅性は，幾何的整合性と同様に，集合制約を生成しこれを Boolean Gröbner 基底で解消することで検証可能である．)

3 Boolean Gröbner 基底による単元集合制約解消

第 2 節で導入したプログラミング言語で記述された分散プログラムについて，その幾何的整合性を検査したい．検査手法として，標準色付き細分によるすべてのファセットの組について不整合が起こらないことを総当たりで検証していく素朴な方法が考えられるが，この方法は現実的ではない．なぜなら，$n-1$ 次元単体の標準色付き細分によって生み出されるファセットの数は Fubini 数 (あるいは ordered Bell 数) F_n で与えられることが知られているが，その漸近表示は $F_n = 0.5n!/(\log 2)^{n+1} + O((0.16)^n n!)$ [Wil90] であり，実際 $F_2 = 3$, $F_3 = 13$, $F_4 = 541$, $F_5 = 102247563$, ... のように急増するため素朴な方法はスケールアップしない．

本稿ではより現実的な整合性検査を実現する方法として，Boolean Gröbner 基底を応用した集合制約解消の手法を採用する．第 4 節で詳述するように，分散プログラムの幾何的整合性条件は集合制約として表すことができる．本節では，Boolean Gröbner 基底による集合制約の解消アルゴリズムについて概説する．

3.1 Boolean 多項式による集合制約表現

$(\mathbf{B}, +, \times, \mathbf{0}, \mathbf{1})$ を $\mathbf{B}$ を台集合とするブール環とする．$\mathbf{B}$ に係数を持つ変数 $\vec{X} = X_1, \ldots, X_m$ の多項式環 $\mathbf{B}[\vec{X}]$ について，イデアル $\langle X_1^2 + X_1, \ldots, X_m^2 + X_m \rangle$ による剰余環はまたブール環である．これをブール多項式環といい $\mathbf{B}(\vec{X})$ で表す．イデアル $I \subseteq \mathbf{B}(\vec{X})$ が解を持つとは，$a_1, \ldots, a_m \in \mathbf{B}$ が存在して，任意の $f(\vec{X}) \in I$ について $f(\vec{a}) = \mathbf{0}$ であることをいう．

特に台集合 $\mathbf{B}$ をある集合の冪集合としたとき，$+$ を差積 $S+T = (S \backslash T) \cup (T \backslash S)$, $\times$ を集合積 $S \times T = S \cap T$, 零元を空集合 $\mathbf{0} = \emptyset$ および単位元 $\mathbf{1}$ を全集合とすることによって $(\mathbf{B}, +, \times, \mathbf{0}, \mathbf{1})$ はブール環となる．以下本稿では，$\mathbf{B} = 2^{\{0,\ldots,n\}}$ とする．

$\vec{A} = A_1, \ldots, A_l$ を $\vec{X}$ と異なる，単元集合の上を動くパラメータとしたとき，$(\mathbf{B}(\vec{A}))(\vec{X})$ はブール多項式環 $\mathbf{B}(\vec{A})$ に係数を持つ (変数 $\vec{X}$ の) ブール多項式環である．$(\mathbf{B}(\vec{A}))(\vec{X})$ のイデアル I が (パラメータ $\vec{A}$ について) 単元集合解を持つとは

$$\exists S_1, \ldots, S_m \in 2^{\{0,\ldots,n\}}, \exists a_1, \ldots, a_l \in \{\{0\}, \ldots, \{n\}\}, \forall f(\vec{X}, \vec{A}) \in I, f(\vec{S}, \vec{a}) = \mathbf{0}$$

をみたすことをいう．

集合に関する様々な制約を $(\mathbf{B}(\vec{A}))(\vec{X})$ の多項式 (が単元集合解を持つかどうか) で表現することができる．[SMS91, SIS$^+$11]

(等値 $S = T$) $S + T$	(包含 $S \subseteq T$) $ST + S$	(所属 $a \in S$) $\{a\}S + \{a\}$
(非所属 $a \notin S$) $\{a\}S$	(非空 $S \neq \emptyset$) $AS + A$	

3.2 単元集合制約解消アルゴリズム

多項式 $f_1, \ldots, f_s$ によって表された集合制約の解を得るには，$(\mathbf{B}(\vec{A}))(\vec{X})$ の有限生成イデアル $\langle f_1, \ldots, f_s \rangle$ の単元集合解を求めればよい．このような求解は，ブール多項式において拡張定理と零点定理 [SIS$^+$11] が成り立つことを利用して，Boolean Gröbner 基底を用いた以下のようなアルゴリズムで判定することができる．(詳細は [SIS$^+$11, Ino12] を参照せよ．)

入力 多項式の有限集合 $F \subset (\mathbf{B}(\vec{A}))(\vec{X})$

出力 単元集合解の有無 (解有りの場合は解の一例，解無しの場合は $\perp$ を返す)

1: **procedure** SingSolve(F)

2: $G_1 \leftarrow \langle F \rangle$ の Boolean Gröbner 基底 [SIS+11]
3: $G_2 \leftarrow G_1 \cap \mathbf{B}(\vec{A})$
4: **if** G_2 の単元集合解を単元集合解アルゴリズム [Ino12] により発見成功 **then**
5: **return** 単元集合解の一例
6: **else**
7: **return** $\perp$

ここで Inoue によるアルゴリズム [Ino12] は，almost solution と呼ばれる単元集合解の一部 (または全部) を代数的に取得し，この almost solution を加えて Boolean Gröbner 基底の計算を繰り返すことにより単元集合解の求解を行うアルゴリズムである．以下のいずれかの条件を満たすとき，$\mathbf{B}(\vec{A})$ の有限生成イデアル $\langle f_1, \ldots, f_s \rangle$ は almost solution $A_j + \{m\}$ $(A_j \in \vec{A},\ 0 \leq m \leq n)$ を持つという．

- $\{m\} f_i = \{m\} A_j + \{m\}$ なる f_i が存在する もしくは
- 任意の $m' \in \{0, \ldots, n\} \setminus \{m\}$ について，$\{m'\} f_i = \{m'\} A_j$ なる f_i $(1 \leq i \leq s)$ が存在する．

almost solution の探索は軽量であるため，これによって解の探索を高速化できる場合が多い．ただし，常に almost solution が存在するとは限らず，存在しない場合は単元集合パラメータに関する網羅的探索が必要となるため，必ずしも解探索が高速に行えるとは限らないことに注意が必要である．

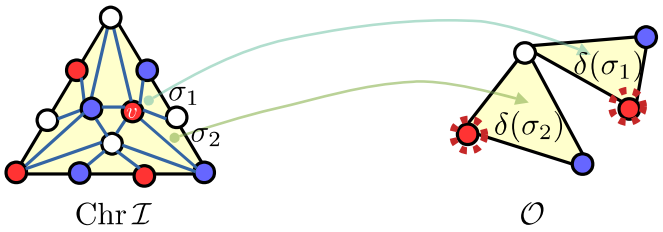
4 幾何的整合性検査のための集合制約生成

プログラム中でファセットからファセットへの関数として定義された単体写像 δ が幾何的整合性条件 (2.1) を満たすかどうかを，整合性条件に対応する集合制約を生成し，その制約に 3 節で示した Boolean Gröbner 基底の計算を適用して検知する手法を示す．

不整合の検知は以下の 2 種類の不整合の可能性に分けて行なう．

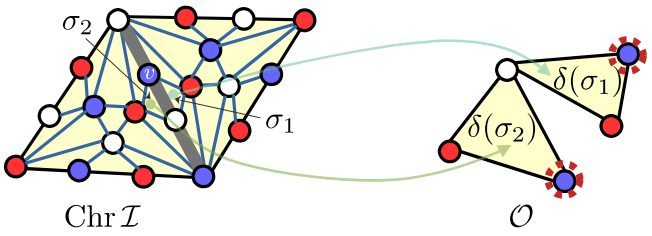
1. 標準色付き細分による細分の内部の頂点に関して不整合が生じる場合

   下図に示すように，入力複体 $\mathcal{I}$ の細分 $\mathrm{Chr}\,\mathcal{I}$ の 2 つのファセット σ_1, σ_2 について，$v \in \sigma_1 \cap \sigma_2$ が存在して，$\chi(v) \notin \chi(\delta(\sigma_1) \cap \delta(\sigma_2))$



2. 標準色付き細分前のファセットが共有する境界上の頂点に関して不整合が生じる場合

   下図に示すように，入力複体 $\mathcal{I}$ の異なる 2 つのファセット τ_1, τ_2 について，$v \in \mathrm{Chr}\,(\tau_1 \cap \tau_2)$ が存在して，ある $\mathrm{Chr}\,\tau_1$ のファセット σ_1 および $\mathrm{Chr}\,\tau_2$ のファセット σ_2 について $v \in \sigma_1 \cap \sigma_2$ だが，$\chi(v) \notin \chi(\delta(\sigma_1) \cap \delta(\tau_2))$



いずれの場合も，プログラム中の全てのパターンの組について以下の手続きを繰り返し行うことで整合性の検査を行う．

1. 集合制約 Φ を，パターンが**不整合であるときおよびそのときに限り充足可能**となるように生成する
2. 集合制約 Φ を $(\mathbf{B}(A))(X)$ の多項式集合 $\{f_1, \ldots, f_s\}$ に翻訳する
3. $\{f_1, \ldots, f_s\}$ が**単元集合解を持たない**ことを，3.2 節の単元集合解アルゴリズム SingSolve で検査する

全てのパターンの組について解が存在しなければプログラムは整合的である．どれかひとつのパターンの組について解が存在すれば，プログラムは不整合であり，その解は不整合の理由を表す．

4.1 パターンがファセットにマッチするための集合制約

上記 2 種類の不整合は，いずれかのパターンにマッチしたファセットの組が幾何的な整合条件をみたさないときに検出されるべきものである．パターンがファセットにマッチすることを，ordered set partition に関する集合制約によって表現する．

本稿では，ordered set partition のパターンを以下のいずれかの形で記述する．[2)]

- 固定長 r の ordered set partition にマッチするパターン $[set_1; \ldots; set_r]$ で，各 set_i は前から i 版目の partition にマッチする．
- 可変長の ordered set partition にマッチするパターン $[set_1; \ldots; set_u; \_; set'_1; \ldots; set'_t]$ $(u, t \geq 0)$ で，長さ $r > u + t$ 以上の partition に対して，各 set_i は前部 u 個の partition に，各 set'_j は後部 t 個の partition にマッチする．$\_$ はこれら 2 つに挟まれた長さ 0 以上の partition の部分列にマッチする．

例えばパターン $[\{1\}; \_; \{0\}]$ が長さ 3 の partition $[S_1; S_2; S_3]$ とマッチするための集合制約は，$[S_1; S_2; S_3]$ が ordered set partition であるための制約に対応する多項式

- 各 S_i が空でない — $S_i A_i + A_i$ $(1 \leq i \leq 3,\ A_i$ は fresh$)$
- S_1, S_2, S_3 が互いに素 — $S_i S_j$ $(1 \leq i < j \leq 3)$
- S_1, S_2, S_3 が partition — $S_1 + S_2 + S_3 + \mathbf{1}$

に加えて，各 partition がみたすべき制約に対応する多項式として，第 1 partition S_1 については $S_1 + \{1\}$，第 3 partition S_3 については $S_3 + \{0\}$ を生成する．第 2 partition にはパターン $\_$ がマッチし，その内容は何でもよいのでこれに関しては何も制約を加えない．もし ordered set partition の長さがパターンと合致しない場合は，充足不可能な多項式 $\mathbf{1}$ を生成する．

以下，このようにパターン pat が partition $[S_1; \ldots; S_r]$ とマッチするための集合制約を表す多項式の集合を $\mathsf{Match}(pat, [S_1; \ldots; S_r])$ と表記する．

[2)] 本稿では省略するが，これよりさらに柔軟なパターン記述もブール多項式によって表現可能である．

4.2 細分内部の頂点に関する不整合検出のための集合制約

単体の標準色付き細分による 2 つのファセットが隣接関係にあることは，以下のように組合せ的に特徴づけられることが知られている．

命題 3 ([Koz12])
d 次元単体 σ $(d>0)$ の標準色付き細分 $\mathrm{Chr}\,\sigma$ の相異なる 2 つのファセット τ_1,τ_2 が $d-1$ 次元面を共有して隣接している (すなわち $\#(\tau_1\cap\tau_2)=d$) ことと，これら 2 つのファセットがそれぞれ以下の 2 つの *ordered set partition*

$$[S_1;\cdots;\{b\};S_i;\cdots S_m] \quad \text{および} \quad [S_1;\cdots;\{b\}\cup S_i;\cdots S_m]$$

と対応することは同値である．このとき，$\chi(\tau_1\cap\tau_2)=\chi(\sigma)\setminus\{b\}$ である．

この特徴付けにより，細分内部の不整合を検出する手続きをアルゴリズム 1 のように構成することができる．このアルゴリズムは，どの入力ファセットの標準色付き細分による隣接するファセットの組も不整合を生じないことを保証する手続きを与える．アルゴリズムは，パターンの組，変数 r，変数 i $(1\le i\le r\le n)$ についての 3 重ループ (第 1,4,5 行目) で構成されており，各ループでは，2 つのパターンがそれぞれ命題 3 のようなファセットの隣接関係に対応する ordered set partition の組 $[S_1;\cdots;A+S_i;S_{i+1};\cdots S_m]$ および $[S_1;\cdots;A\cup S_i;S_{i+1};\cdots S_m]$[3] にマッチする (第 6,7 行目) 場合，これらのファセットが共有しない頂点の色集合 A とパターンマッチの出力が共有しない頂点の色集合 Q (第 2 行目) に不整合がないことを検査する．Q が空集合の場合 (第 3 行目) は不整合は起こり得ないので，集合制約の検査は行わない．Q が単元集合 $\{b\}$ のとき (第 9 行目)，$A\cap\{b\}=\emptyset$ を加えた集合制約が解を持つ場合不整合となる．一方，Q が 2 つ以上の要素からなるときはこのような制約を加えなくてもよい．なぜなら，2 つのパターンが命題 3 の意味で隣接す

アルゴリズム 1 細分内部の不整合検知アルゴリズム

入力 パターン集合 $t_1,pat_1\to u_1\mid\cdots\mid t_k,pat_k\to u_k$

出力 細分内部が整合的であれば $\top$，そうでなければ不整合となる一例を返す

```
1: for each j1, j2 ∈ {0,...,n} do
2:     Q ← (t_{j1} = t_{j2} のときの {0,...,n} \ χ(u_{j1} ∩ u_{j2}))      ◁ 出力ファセットが共有しない頂点の色集合
3:     if Q ≠ ∅ then
4:         for r = 1 to n do
5:             for i = 1 to r do
6:                 let S_1,...,S_r, U_i ∈ X⃗, A ∈ A⃗      ◁ fresh な変数
7:                 F ← Match(pat_{j1}, [S_1 ··· ; S_{i-1}; A + S_i; S_{i+1}; ··· ; S_r])
8:                 F ← F ∪ Match(pat_{j2}, [S_1 ··· ; S_{i-1}; A; S_i; S_{i+1}; ··· ; S_r])
9:                 if Q = {b} then F ← F ∪ {{b}A}      ◁ 集合制約 A ∩ {b} = ∅
10:                Sol ← SingSolve(F)
11:                if Sol ≠ ⊥ then return Sol
12:            end for
13:        end for
14: end for
15: return ⊤
```

[3] A が命題 3 における単元集合 $\{b\}$ に対応する．また，$[S_1;\cdots;A;S_i;S_{i+1};\cdots S_m]$ が ordered set partition であることより $A\cap S_i\neq\emptyset$ であるから，$A+S_i$ は $A\cup S_i$ と等しい．

るファセットにマッチするとき，入力で共有する頂点の数が n であるのに対して出力で共有される頂点の数は $n+1-\#Q < n$ となり，常に不整合だからである．

4.3 ファセット境界上の頂点に関する不整合検出のための集合制約

入力複体 $\mathcal{I}$ の 2 つのファセットがその境界上で標準色付き細分による頂点を共有することは，命題 3 に示した隣接条件より，以下のように組合せ的に特徴づけられる．

系 4
2 つの d 次元単体 $(d>0)$ σ_1, σ_2 の標準色付き細分 $\operatorname{Chr}\sigma_1, \operatorname{Chr}\sigma_2$ が共通の頂点を持つ (すなわち $V(\operatorname{Chr}\sigma_1) \cap V(\operatorname{Chr}\sigma_2) \neq \emptyset$) ことと，細分 $\operatorname{Chr}\sigma_1, \operatorname{Chr}\sigma_2$ それぞれのファセット τ_1, τ_2 が存在して，これらが長さ 2 の (同一な)ordered set partition $[S_1; S_2]$ に対応して $S_1 \subseteq \chi(\tau_1 \cap \tau_2)$ であることは同値である．

この組合せ的特徴付けより，ファセット境界上の不整合を検出する手続きをアルゴリズム 2 のように構成することができる．各パターンの組について (第 1 行) P を入力ファセットが共有する頂点の色集合 (第 2 行)，Q を出力ファセットが共有しない頂点の色集合 (第 3 行) とする．入力ファセットの境界上で標準色付き細分による頂点が共有される場合，系 4 によりこれらの頂点の色集合は長さ 2 の partition $[S_1;S_2]$ に対応する標準色付き細分による 2 つのファセットの共通部分，すなわち S_1 に含まれる．よって，両方のパターンが $[S_1;S_2]$ にマッチし (第 5 行) かつ集合制約 $S_1 \subseteq P, Q \cap S_1 \neq \emptyset$ (第 6 行) が解を持つ場合，不整合を検知する．

アルゴリズム 2 ファセット境界上の不整合検知アルゴリズム

入力 パターン集合 $t_1, pat_1 \to u_1 \mid \cdots \mid t_k, pat_k \to u_k$
出力 ファセット境界上に関して整合的であれば $\top$，そうでなければ不整合となる一例を返す

1: **for each** $j_1, j_2 \in \{0, \ldots, n\}$ **do**
2: $\quad P \leftarrow \chi(t_{j_1} \cap t_{j_2})$ ◁ 入力ファセットが共有する頂点の色集合
3: $\quad Q \leftarrow \{0, \ldots, n\} \setminus \chi(u_{j_1} \cap u_{j_1})$ ◁ 出力ファセットが共有しない頂点の色集合
4: $\quad$ **let** $S_1, S_2 \in \vec{X}$, $A \in \vec{A}$ ◁ fresh な変数
5: $\quad F \leftarrow \mathsf{Match}(pat_{j_1}, [S_1; S_2]) \cup \mathsf{Match}(pat_{j_2}, [S_1; S_2])$
6: $\quad F \leftarrow F \cup \{PS_1 + S_1, QS_1A + A\}$ ◁ 集合制約 $S_1 \subseteq P, Q \cap S_1 \neq \emptyset$
7: $\quad Sol \leftarrow \mathsf{SingSolve}(F)$
8: $\quad$ **if** $Sol \neq \bot$ **then return** Sol
9: **end for**
10: **return** $\top$

5 整合性検査器の実装

前節で示した手法に基づいて，並行分散プログラムの整合性検査器のプロトタイプ実装を行った．実装言語は，静的型付き関数型言語 OCaml[OCa] である．

実際に実装を行ってみると，前節で示した手法を単純にそのまま実装しただけでは満足いく効率は得られないことがわかった．(初期バージョンのプロトタイプでは，2.2 節で示した 2 プロセスの 2 値 quasi-consensus 問題のプログラムに対する整合性検査にも 1 分程度を要した．(実行環境: MacOS/Apple M1)) 検査器の実行状況を精査したところ，以下のような要因が効率的な検査の妨げとなっていた．

- 多項式集合に含まれる変数や単元集合パラメータの個数が多い．

  特に単元集合パラメータの個数が増えるほど，Inoue アルゴリズム [Ino12] では almost solution が得られない場合の網羅的探索が行われる機会が多くなり，実行効率の低下に結びつきやすいと考えられる．

- 生成する集合制約に重複がある．

  集合制約の生成を前節の通りに行うと，全く同じ制約や論理的に重複した制約が生成されてしまうことがあり，これらが実行効率に大きく影響することがある．

- Boolean Gröbner 基底および単元集合解のアルゴリズムをアルゴリズム 1 や 2 のループ 1 回毎に必ず 1 度実行すること自体が効率低下の原因となっている．

以上の観察より，以下のような検査器の最適化を行った．

- Boolean Gröbner 基底による集合制約の解消について，明らかに解がないとわかるような対象についての検査はスキップすることで，Gröbner 基底の計算を本当に必要な時に限定して行うようにした．すなわち，パターンが ordered set partition にマッチしない (パターンの長さと partition の長さが合致しない・集合の要素数が合致しない等) ときは Gröbner 基底の計算を省略するようにした．

- 重複した集合制約を生成しないように最適化を行なった．例えば，アルゴリズム 1 で生成される集合制約 $\mathsf{Match}(pat_{j_1}, [S_1 \cdots ; S_{i-1}; A + S_i; S_{i+1}; \cdots ; S_r])$ と $\mathsf{Match}(pat_{j_2}, [S_1 \cdots ; S_{i-1}; A; S_i; S_{i+1}; \cdots ; S_r])$ にはそれぞれ $[S_1 \cdots ; S_{i-1}; A + S_i; S_{i+1}; \cdots ; S_r]$ と $[S_1 \cdots ; S_{i-1}; A; S_i; S_{i+1}; \cdots ; S_r]$ が ordered set partition であるという制約が含まれるが，前者に関するこの制約は後者のそれから導かれるので割愛することができる．

- さらに，論理的に重複した集合制約も削除するように最適化を行なった．例えば，集合制約が $X \neq \emptyset$, $2 \in X$ を含むとしよう．このとき，$X \neq \emptyset$ はもう一方の制約 $2 \in X$ から導かれるので削除できる．特にこの場合，$X \neq \emptyset$ は fresh な単元集合パラメータ A を含む多項式 $AX + A$ に対応することから，単元集合パラメータを 1 つ減らすことができ，上述の Inoue アルゴリズム [Ino12] における網羅的探索の機会を減らすことができるという点で有効である．

これらの最適化により大幅な効率改善を達成することができた．具体的には，2.2 節の 2 値 quasi-consensus 問題の 3 プロセス拡張版に対する検査が 1 秒未満で終了するようになった．このことは，Boolean Gröbner 基底が，並行分散計算の整合性検査に必要な集合制約解消の手法として有効であることを示している．

6 まとめと将来課題

並行分散計算プログラムの幾何的整合性問題を集合制約解消問題として表し，Boolean Gröbner 基底を用いた単元集合制約アルゴリズムを適用することで検証する手法を提案した．この手法にいくつかの最適化を施すことによって，現実的な時間内で整合性検査を行うことができた．このことは，Boolean Gröbner 基底が並行分散計算の整合性検査に有効な手法であることを示している．

本研究は，Boolean Gröbner 基底の手法が並行分散計算の効率的な整合性検査に適用可能であることを示すものであるが，これがプロセス数が大きい場合等，より現実的なプログラムに対してもどのくらい効率的に適用可能であるかを調査することは今後の課題である．期待したほどの効率性が得られない場合は，さらなる最適化 (Boolean Gröbner 基底アルゴリズム自体の最適化も含む) について検討する必要があるだろう．また，今回の並行分散プログラム記述は 1 回の標準色付き細分のみを扱っているが，これを複数回の細分を許すようにした場合の本手法の有効性についても検証が必要である．

謝　辞

本研究は JSPS 科研費 20K11678 の助成を受けたものです.

参 考 文 献

[GK18] Rachid Guerraoui and Petr Kuznetsov. *Algorithms for Concurrent Systems*. EPFL press, 2018.

[GLR21] Éric Goubault, Jérémy Ledent, and Sergio Rajsbaum. A simplicial complex model for dynamic epistemic logic to study distributed task computability. *Information and Computation*, 278:104597, 2021. An earlier version appeared in Proc. of 9th International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2018.

[HKR13] Maurice Herlihy, Dmitry N. Kozlov, and Sergio Rajsbaum. *Distributed Computing Through Combinatorial Topology*. Morgan Kaufmann, 2013.

[HS99] Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *Journal of the ACM*, 46(6):858–923, 1999.

[Ino12] Shutaro Inoue. Efficient singleton set constraint solving by boolean Gröbner bases. *Communications of JSSAC*, 1:27–37, 2012.

[Koz12] Dmitry N. Kozlov. Chromatic subdivision of a simplicial complex. *Homology, Homotopy and Applications*, 14(2):197–209, 2012.

[OCa] OCaml homepage. `https://ocaml.org/`.

[SIS+11] Yosuke Sato, Shutaro Inoue, Akira Suzuki, Katsusuke Nabeshima, and Ko Sakai. Boolean Gröbner bases. *Journal of Symbolic Computation*, 46:622–632, 2011.

[SMS91] Yosuke Sato, Satoshi Menju, and Ko Sakai. Solving constraints over sets by boolean Gröbner bases. ICOT Tehcnical Report TR-680, Institute for New Generation Computer Technology, 1991.

[Wil90] Herbert S. Wilf. *generatingfunctionology*. Academic Press, 1990. `https://www2.math.upenn.edu/~wilf/DownldGF.html`.