社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.
COMP99-49 (1999-11)

多面体分割を用いた有理数プレスブルガー文真偽判定アルゴリズムとその実装

柴田 直樹　岡野 浩三　谷口 健一

大阪大学 大学院基礎工学研究科 情報数理系専攻
〒 560-8531 大阪府 豊中市 待兼山町 1-3
TEL : 06-6850-6608, FAX : 06-6850-6609

E-mail : {n-sibata, okano, taniguchi}@ics.es.osaka-u.ac.jp

あらまし 加算を持つ有理数の理論 (有理数変数, 有理数定数,+, −,=,<,∧,∨,∀,∃ からなる理論) の上の冠頭形の閉論理式 (PRP 文と呼ぶ) の真偽判定ルーチンはプロトコルのテスト, ハードウェアのタイミング検証などに利用される. 著者らは最も速い PRP 文の真偽判定アルゴリズムよりを既に提案している. 本稿では, 筆者らが提案したアルゴリズムの多面体分割を用いた高速化法と, その評価について述べる. 高速化法として, 次の 3 点が挙げられる. 1. 多面体同士を併合し, その結果できる凹多面体も扱えるようにすることで, 処理する多面体の数を最小限にする. 2. 投影の領域を計算する方法を改善し, 真偽判定に必要ない多面体を処理する手間を減らす. 3. 投影の領域を計算する際の凹多面体同士の交差判定の回数を減らす.

キーワード 加算を持つ有理数の理論 真偽判定アルゴリズム 多面体

A decision algorithm for rational Presburger sentences using a division routine of polyhedra

Naoki SHIBATA, Kozo OKANO and Kenichi TANIGUCHI

Division of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University
Machikaneyama 1-3, Toyonaka, Osaka 560-8531, JAPAN
Phone : +81-6-6850-6608, Fax : +81-6-6850-6609
E-mail : {n-sibata, okano, taniguchi}@ics.es.osaka-u.ac.jp

Abstract Formerly, we have proposed the fastest decision algorithm for prenex-normal-form rational Presburger sentences. The algorithm is used in areas of testing of protocols, timing verification of hardware, and so on. In this paper, we present techniques to reduce execution time of the algorithm. The techniques are based on three devices: 1. Making the algorithm merge polyhedra into a concave polyhedron and handle the concave polyhedron in order to minimize the number of polyhedra that express a region, 2. Improving a method to calculate a region of a shadow without processing extra polyhedra that don't affect the result of the decision and 3. Reducing the number of judgements on whether faces are crossing.

keywords Thoery of rationals with addtion, decision procedure and polyhedra

1 はじめに

加算を持つ有理数の理論 (有理数変数, 有理数定数,+, −, =, <, ∧, ∨, ∀, ∃からなる理論) の冠頭形の閉論理式 (以降PRP文と呼ぶ) の真偽判定ルーチンはプロトコルのテスト, ハードウェアのタイミング検証などに利用される [1].

筆者らは文献 [2] で解説されている, 従来知られていた最も速いアルゴリズムである Ferrante と Rackoff のアルゴリズム [3] よりも時間計算量オーダの小さいアルゴリズムを提案した [4]. Ferrante と Rackoff のアルゴリズムの時間計算量は $r5^{\epsilon d}n^{\zeta d(2b+1)^a}$ (n, d, a, b, rはそれぞれ入力のPRP文に含まれる不等式の個数, 変数の個数, 限定子交替数, 同じ限定子が続く最大の個数, 入力の式の各係数と定数の分母, 分子の最大のビット数, ϵ, ζは定数) であり, 筆者らの提案したアルゴリズムの時間計算量は時間計算量は $r\alpha^{\beta d}n^{\gamma d(b+1)^a}(\alpha, \beta, \gamma$は定数) である. 双方のアルゴリズムの計算量で支配的なのは二重指数の部分であり, 提案するアルゴリズムの計算量は Ferrante と Rackoff のアルゴリズムのものに比べてこの部分が改善されている. 本稿では, 筆者らが提案したアルゴリズムに対して, さらに多面体分割を用いて高速化する手法と, その評価について述べる. 高速化法として, 次の3点が挙げられる. 1. 多面体同士を併合し, その結果できる凹多面体も扱えるようにすることで, 処理する多面体の数を最小限にする. 2. 投影の領域を計算する方法を改善し, 真偽判定に必要ない多面体を処理する手間を減らす. 3. 投影の領域を計算する際の凹多面体同士の交差判定の回数を減らす. 以後, 2章では以前提案したアルゴリズムの概要を述べ, 3章では今回提案する高速化手法について述べる. 4章では実装した高速化手法を評価する.

2 基本アルゴリズムの概略

以降, 基本アルゴリズムの真偽判定の概要について述べる. また, 必要な概念を適宜定義していく.

まず, 3次元の場合について必要な概念を定義する. 平面で空間を平面の一方, もう一方, そして平面に含まれる空間の3つに分割することを考える. 複数の平面により空間は, 点, 両端を含まない線分, 外周を含まない多角形, 外面を含まない多面体に分割される. こうして分割されたそれぞれの部分をフェイスと呼ぶ. 点, 両端を含まない線分, 外周を含まない多角形, 外面を含まない多面体をそれぞれ0-フェイス, 1-フェイス, 2-フェイス, 3-フェイスと呼ぶ. 空間全体を平面の集合 Hでフェイスの集合 A に分割したとき, A を Hのアレンジメントと呼ぶ. また, 点を0-フラット, 直線を1-フラット, 平面を2-フラット, 空間全体を3-フラットと呼ぶ. 以上3次元で説明したが一般に d 次元に拡張して $(d-1)$-フェイス, $(d-1)$-フラット, などという用語も使う. d 次元空間における $(d-1)$-フラットを超平面と呼ぶ.

以上のことがらを一般次元に拡張して定義すると, 次のようになる. これらの定義は文献 [5] に従う.

定義 1 k-フェイス

一般性を失うことなく v_d軸に垂直でない超平面 h を考える. h 上の任意の点 $x = (x_1, x_2, \cdots, x_d)$ について $x_d = \eta_d + \sum_{i=1}^{d-1} \eta_i x_i$が成り立つというような実数の組$\eta_1, \cdots, \eta_d$が唯一つ存在する. 点 $p = (\pi_1, \cdots, \pi_d)$ に対しπ_dが$\eta_d + \sum_{i=1}^{d-1} \eta_i \pi_i$より大きい, 等しい, 小さい場合にそれぞれ, p は h より上にある, h に乗っている, h より下にあるということにする.h^+は h より上にある点の集合を,h^-は h より下にある点の集合を表す. 超平面の集合 $H = \{h_1, \cdots, h_n\}$ に含まれるどの超平面も全て v_d軸に垂直でないとする. 超平面 h_iおよび点 p に対し $v_i(p)$ を以下のように定める.

$$v_i(p) = \begin{cases} +1 & (p \in h_i^+) \\ 0 & (p \in h_i) \\ -1 & (p \in h_i^-) \end{cases}$$

$(v_1(p), \cdots, v_n(p))$ の値が同じ点 p の集合をフェイスという. k-フラットに含まれて, $(k-1)$-フラットに含まれないフェイスを k-フェイスと呼ぶ. 特に0-フェイスを頂点と呼ぶ. あるフェイスが k-フェイスであるとき, フェイスの次元は kである. □

定義 2 アレンジメント

d 次元ユークリッド空間内の超平面の有限集合 Hは d 次元以下の種々の次元のフェイスに空間を分割する. このフェイスの集合を, Hによって d 次元のユークリッド空間を分割してできる (または単に Hの) アレンジメントと呼ぶ. Hのアレンジメント A に含まれるフェイスを A を構成するフェイスと呼ぶ. ある k-フラット flがアレンジメント A を構成する k-フェイス fを含むとき, flは A から定まるフラットであるという. □

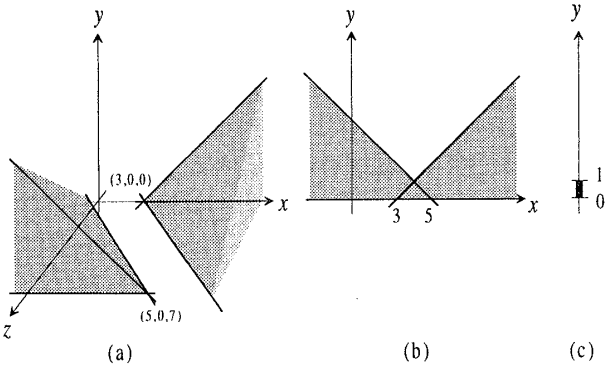

図 1 アルゴリズムの適用によって変化していくアレンジメント

以降, PRP文 $F = \exists y \forall x \exists z \{y \geq 0 \wedge [(z \geq 0 \wedge x - y - z \geq 3) \vee (z \leq 7 \wedge x + y - z \leq -2)]\}$ の真偽判定の様子を例に, 基本アルゴリズムの真偽判定の概要について述べる.

最初に入力のPRP文の母式 (PRP文から限定子を全て取り除いてできる式) に含まれる全ての等式と不等式に

対応する超平面全ての集合から空間を分割するアレンジメントを作る．PRP 文 Fには 3 つの変数が含まれるので，3 次元で考える．Fの母式 $E = y \geq 0 \wedge [(z \geq 0 \wedge x - y - z \geq 3) \vee (z \leq 7 \wedge x + y - z \leq -2)]$ が真になる領域は図 1(a) の灰色の部分で表され，この領域は E中の各不等式の表す平面で 3 次元空間を分割してできるアレンジメントに含まれるフェイスの部分集合である．

例の PRP 式において，次に，Eより得られたアレンジメント A の各フェイス fに，fに含まれる任意の点の座標を母式に代入して得られる真偽値を割り当てる (この操作を A に Eの真偽を割り当てると呼ぶ)．これによって，空間の任意の点 p に対し，p の座標を Eに代入して得られる真偽値と p が含まれるフェイスに割り当てられた真偽値が一致する．図 1(a) 中の灰色のフェイスが，真が割り当てられたフェイスである．

ついで，$\exists z$を消去する操作を行う．これは Eを真にする zが存在する x, yの値の領域を x-y平面上に図示する操作である．これによって得られた図が図 1(b) である．これは，図 1(a) に z軸に並行な光を当ててできる真になる部分の影である．この影の部分を表す真偽を割り当てたアレンジメントを真偽を割り当てたアレンジメントの投影と呼ぶ．

投影のアレンジメントを作る操作は図 1(a) 中の 1-フラット (直線) の x-y平面への投影 (直線) 全ての集合から 2 次元のアレンジメントを作り，図 1(a) の真になる部分の影に含まれるフェイスにのみ真を割り当てることで行う．

定義 3 点の投影

点 $p = (V_1, V_2, ..., V_d)$ の $(v_1, v_2, ..., v_{d-s})$ 空間への投影は $(V_1, V_2, ..., V_{d-s})$ である． □

フェイスの投影，フラットの投影についても点の投影と同様に定義できる [4]．以降，領域 Rの $(v_1, v_2, ..., v_{d-s})$ 空間への投影を $\mathbf{Pr}_{(v_1, v_2, ..., v_{d-s})} R$と表す．

定義 4 アレンジメントの投影

アレンジメント Bの $(v_1, v_2, ..., v_{d-s})$ 空間への投影は，Bから定まる $(d-1-s)$-フラット ($(v_1, v_2, ..., v_{d-s})$ 空間での超平面) の $(v_1, v_2, ..., v_{d-s})$ 空間への投影 fl'のうち fl'が $(d-1-s)$-フラットとなる fl'の集合のアレンジメントである． □

一般に，あるアレンジメント A に属するフェイス fの投影に対応する集合 $\mathcal{F}$があって，次の条件を満たす: $\mathcal{F}$は A の投影上のフェイスの集合であり，かつ fの投影に含まれる点の集合と $\mathcal{F}$に含まれるフェイスから構成される点の集合は一致する [4]．

さて，入力の論理式の限定子の並びは $\exists x \forall y \exists z$なので，次に $\forall y$を消去して，1 次元の真偽を割り当てたアレンジメントを得る．これは図 1(c) で表される．$\exists$ を消去する操作が，真になる領域に座標軸に並行な光を当ててできる影を真とする領域であるのに対し，$\forall$ を消去する操作は，偽になる領域に座標軸に並行な光を当ててできる影を偽とする領域である．残った $\exists x$ を消去し，0 次元の真偽が割り当てられたアレンジメントを得る．これは真が割り当てられた 1 つの頂点からなる．したがって，式全体は真と判定される．

3 高速化を施したアルゴリズム

3.1 高速化の概略

以降，2章のアルゴリズムに対する多面体分割を用いた高速化法について述べる．高速化の工夫は次の 3 点からなる [6]．1. 多面体同士を併合し，その結果できる凹多面体も扱えるようにすることで，処理する多面体の数を最小限にする．2. 投影の領域を計算する方法を改善し，真偽判定に必要ない多面体を処理する手間を減らす．3. 投影の領域を計算する際の凹多面体同士の交差判定の回数を減らす．

3.1.1 凹多面体のフェイスの処理の導入

真偽判定の処理の中で，フェイスの数が計算オーダの中で最も大きな影響力を持つ．真と偽の領域の境界が表現できれば，たくさんのフェイスを扱わなくとも真偽判定は可能である．そこで，隣り合っている，同じ真偽値が割り当てられたフェイスを併合する．併合した結果，凹多面体のフェイスができるが，これもアルゴリズム中で扱えるようにする．同じ領域を表すために必要な最低限のフェイスの数になるように併合する．凹多面体を扱うことによって，領域を表すためのデータ構造の大きさを最小に保つことができる．Ferrante と Rackoff のアルゴリズムのように不等式の論理結合式を使って表現式上で領域を表す方法では，同じ領域を表す最簡の式を求めることは非常に困難と思われる．

凹多面体を扱うために，フェイス等の概念を拡張する．

定義 5 閉包

集合 $\{x | d(x,p) < \rho\}$ を中心 p，半径ρの開球という (但し $d(x,p)$ は点 x, p 間のユークリッド距離を表す)．Sを点の集合とする．Sの全ての点 p に対し，p を中心として Sに含まれるような開球が存在するならば，Sは開であるという．Sが閉であるとは，Sの補集合が開であることである．点の集合 Sの閉包は，Sを含む最小の閉集合であり，$\mathbf{cl}\ S$ と表す． □

定義 6 サブフェイス，スーパーフェイス

ある k-フェイス fが $(k+1)$-フェイス gの閉包から gを除いた領域に含まれるとき，fは gのサブフェイスであるといい，gは fのスーパーフェイスであるという．以降，フェイス fのスーパーフェイスの集合とサブフェイスの集合を $\mathrm{super}(f)$,$\mathrm{sub}(f)$ で表す． □

定義 7 ancestral フェイス

ancestral フェイスは次のように再帰的に定義される：あるフェイス fのスーパーフェイスは fの ancestral フェイ

スであり，fの ancestral フェイスのスーパーフェイスは f の ancestral フェイスである． □

定義 8 descendant フェイス

descendant フェイスは次のように再帰的に定義される：あるフェイス fのサブフェイスは fの descendant フェイスであり，fの descendant フェイスのサブフェイスは fの descendant フェイスである． □

定義 9 拡張されたフェイス

アレンジメント A とフェイスの集合 S_0と S_1を以下の条件を満たすように任意に選ぶ．S_0の要素は，A に含まれる互いに重ならない k-フェイスであり，これらは共通の k-フラット flに含まれる．S_1の要素は S_0の全ての要素の descendant フェイスの集合の勝手に選んだ部分集合である．

S_0に含まれるフェイスに含まれる点全ての集合を R_0とする．S_2を次のように定義する：$S_2 = \{f | f \in S_1 \wedge f \subseteq (\mathbf{cl}\ R_0) - \mathbf{cl}\ (fl - \mathbf{cl}\ R_0)\}$．$S_0 \cup S_2$の要素に含まれる点全ての集合は拡張されたフェイスである[1]． □

以降，単にフェイスと書いた場合は拡張されたフェイスを指す．また，スーパーフェイス，サブフェイス，ancestral フェイス，descendant フェイスは拡張されたフェイスに対しても同様に定義される．

定義 10 拡張されたアレンジメント

有限個のフェイスの和集合が空間全体であるとき，これらのフェイス全ての集合を拡張されたアレンジメントと呼ぶ．拡張されたアレンジメントに含まれるどの 2 つのフェイスの組合わせについても重なりがない場合，この拡張されたアレンジメントを「フェイスの重なりのない拡張されたアレンジメント」と呼ぶ． □

以降，単にアレンジメントと書いた場合はフェイスの重なりのない拡張されたアレンジメントを指す．

3.1.2 フェイスの投影からの投影のアレンジメントの直接作成

高速化を施す前のアルゴリズムでは，定義通りに投影を取る前のアレンジメントを構成する線分を延長してできるフラットの投影全ての集合を求め，それらで空間を細分化することで投影のアレンジメントを作っていた．この方法では細分化する必要のない空間まで細分化するので，投影のアレンジメントに含まれるフェイスの数が莫大になり，計算時間がかかってしまう．高速化を施したアルゴリズムでは，投影のアレンジメントに含まれるフェイスの数をなるべく減らすことを考え，また，投影を取る前のアレンジメントの持つ情報をできるだけ活かして投影の処理を行う．具体的には，まず投影を取る前のアレンジメントの各フェイスの投影の集合を求める (図 2(a))．この状態ではフェイス同士が重なっているので，重なりあったフェイスを分割して重ならないようにする (∀ を消去するためには，偽が割り当てられたフェイスに含まれる領域を明確にする必要があり，また，∃ を消去するためには，真が割り当てられたフェイスに含まれる領域を明確にする必要がある．フェイスが重なったままでは真及び偽が割り当てられたフェイスに含まれる領域が不明確であるので，フェイスを分割して重ならないようにすることで，それらの領域を明確にする処理は不可欠である)．これは，アレンジメント中の 2 つのフェイス (図 2(a) の斜線のフェイス) に注目し，図 2(b) のように分割する．

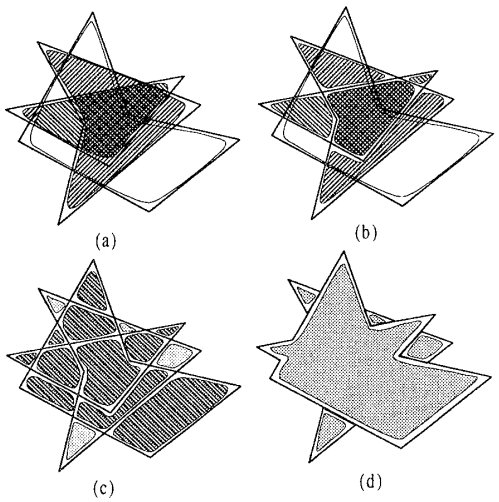

図 2 フェイス同士の重なりがなくなるようにする過程

全てのフェイスの組合わせに対し分割を行うと，図 2(c) のようになる．その後，同じ真偽値を持つフェイス (図 2(c) の濃い色のフェイス) を併合すると，図 2(d) のようになる．

3.1.3 フェイスの交差判定回数の削減

上で述べた高速化を施す上で，フェイス同士が重なっているか調べる処理を，全ての 2 つのフェイスの組合わせの数だけ行う必要が生じる．この回数をなるべく減らすために次のような高速化を施す．空間を 1 つの超平面の片側 (上側) ともう一方 (下側) の 2 つに分割し，それぞれのフェイスに対し「上側にのみ交わる[2]」,「下側にのみ交わる」,「上側と下側の両方に交わる」のいずれであるかあらかじめ調べておく．上側にのみ交わるフェイスと下側にのみ交わるフェイスは交差しないので，これらのフェイス同士の交差判定は省略できる．空間をさらに細かく分割することで，最良の場合フェイス同士の交差判定の回数はフェイスの数 n に対して $O(n \log n)$ 程度にまで少なくできる．また，最悪の場合でもフェイス同士の交差判定の回数は変わらない [6]．

また，上記の方法はフェイスの数が多い場合に明らかに交わっていないフェイスの組合わせの交差判定を省略する方法であるが，2 つの凹多面体のフェイスの交差判定の

[1] 拡張されたフェイスは直観的には凹のフェイスである．ここで，S_2 は R_0の周辺の (勝手に選んだ) フェイスの集合と見なせる．

[2] 正確には上側の部分空間とのみ共有点を持つ．

前処理として，それぞれのフェイスを含む最小の直方体を求め，それらが交わっているかを調べる [7]．これらの直方体が交わってなければ，2つのフェイスは交わっていない．

3.2 高速化を施したアルゴリズムの詳細

最初にいくつかの用語の定義を与えた後，アレンジメントのデータ構造について述べ，アルゴリズムの詳細について述べる．

以下では，アレンジメントのデータ構造について述べる．

図3のアレンジメントのデータ構造は図4で示されるグラフであり，図4の各ノードが図3の各フェイスに対応している．あるフェイス f に対応するノードは，f のスーパーフェイス群，サブフェイス群に対応するノード群へのエッジ群，次元が1つ下のフェイスへのエッジ群で表される．各ノードには対応するフェイスに割り当てられた真偽値などが含まれる (以降フェイス f に割り当てられた真偽値を truth(f) で表す)．また，頂点に対応するノードは上記に加えて任意精度の多倍長の有理数で表された座標を含む．なお有界でないフェイス (図3のフェイス AG 等) については十分大きな値を頂点の座標として用いる (例えば A の座標として半直線 GA 上の十分 G から離れた点をとる)．

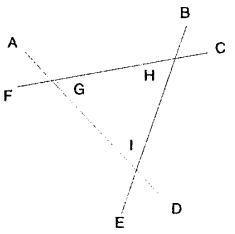

図 3 アレンジメントの例

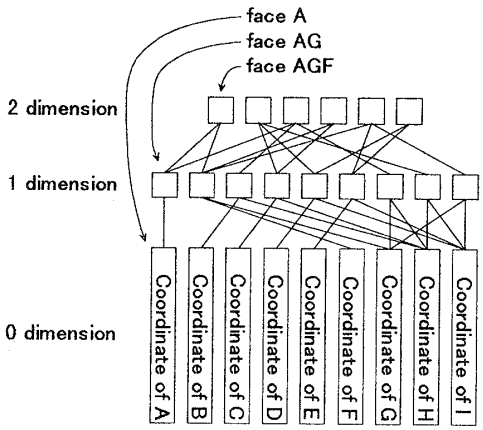

図 4 図3のアレンジメントのデータ構造

定義 11 隣り合うフェイス

2つのフェイスが隣り合うとは，2つのフェイスが共通のサブフェイスを持つことを言う． □

定義 12 連続

ある領域のどのような2点を取っても，その2点を結ぶ，その領域に含まれる曲線が存在するときその領域は連続であるという． □

以下では，まずメインルーチンである MAIN について述べ，その後サブルーチンとなる PROJECT, SIMPLIFY, DIVIDE について順に述べる．ここで，
PROJECT では真偽を割り当てたアレンジメントの投影を作り，SIMPLIFY はアレンジメント中の，隣り合っていて同じ真偽値が割り当てられたフェイスを全て併合する．DIVIDE は重なったフェイスを，一方のフェイスのみに含まれた部分，他方のみに含まれた部分，両方に含まれた部分に分割する．

MAIN は高速化を施したアルゴリズムのメインルーチンで，与えられた PRP 文の真偽を判定する．

Algorithm MAIN
◇ 入力：PRP 文 F
◇ 出力：F の真偽値
1 入力の PRP 文の限定子の並びを $Q_1, ..., Q_d$ とする．それぞれの限定子が束縛する変数を $v_1, ..., v_d$ とする．
2 $H := F$ に含まれる不等式に対応する超平面全ての集合;
3 $A :=$ ARRANGE(H);
4 $A :=$ ASSIGN(A,Fの母式);
5 $A :=$ SIMPLIFY(A);
▷ i は残っている限定子の数
6 $i := d$;
7 **while** $i \geq 1$ **do**
8 $s :=$ 最も内側の連続した Q_i の個数;
9 $A :=$ PROJECT(A,s,Q_i);
10 DIVIDE(A,Q_i);
11 SIMPLIFY(A);
12 $i := i - s$;
13 **endwhile**
▷ A は 1 つの点だけで構成される
14 **return** A の原点の真偽値;
15 **end**

Algorithm PROJECT
◇ 入力：アレンジメント A, 整数 s, 限定子 q
◇ 出力：A に含まれるフェイス全てを $(v_1, ..., v_{d-s})$ 空間へ投影してできるアレンジメント
1 A の次元を d とする．
2 **for** i := d **to** 1 **do**
3 **for each** i-フェイス $f \in A$ **do**
4 **if** $\mathbf{Pr}_{(v_1,v_2,...,v_{d-s})} f$ が i-フェイスではない **then**
5 **for each** フェイス g s.t. f のサブフェイスで $v_{d-s+1}...v_d$ 軸のいずれにも平行でない **do**
6 **if** $q =$ '∃' **then**
g の真偽値 := g の真偽値 ∨ f の真偽値
else g の真偽値 := g の真偽値 ∧ f の真偽値
endif
next
7 f を A から削除
8 **endif**
9 **next**

```
10      next
11      for each 頂点 v ∈ A do
12          v := Pr_(v_1,v_2,...,v_{d-s}) v;
13      next
14  end
```

Algorithm SIMPLIFY

◇ 入力：アレンジメント A
◇ 出力：A に含まれる隣り合っていてかつ同じ真偽値が割り当てられたフェイスを全て併合してできるアレンジメント

```
1       for each f ∈ A, 次元の低い順に do
            ▷ fのスーパーフェイス全てに割り当てられた真偽値がfのと同じでなければ continue
2           if ∃g, g ∈ super(f) ∧ truth(g) ≠ truth(f) then continue ;
            ▷ fのスーパーフェイスのスーパーフェイスが全て同じではなければ continue
3           if ∃g∃h, g ∈ super(f) ∧ h ∈ super(f) ∧ super(g) ≠ super(h) then continue ;
            ▷ if fのスーパーフェイスのスーパーフェイスが全て同じ同次元のフラット上になければ continue
4           if ∃g∃h∃i∃j, g ∈ super(f) ∧ h ∈ super(g) ∧ i ∈ super(f) ∧ j ∈ super(i) ∧ flatIncludingFace(h) ≠ flatIncludingFace(j) then continue ;
5           fと fのスーパーフェイス全てを併合する
6       next
7   end
```

Algorithm ARRANGE

◇ 入力 超平面の集合 H
◇ 出力 Hのアレンジメント

アルゴリズムは文献 [5] 第 7 章参照

Algorithm ASSIGN

◇ 入力 与えられた入力論理式の母式 E,Eの不等式の表す超平面集合を含む超平面集合のアレンジメント A
◇ 出力 A に Eの真偽を割り当てたアレンジメント

アルゴリズムは文献 [4] 参照

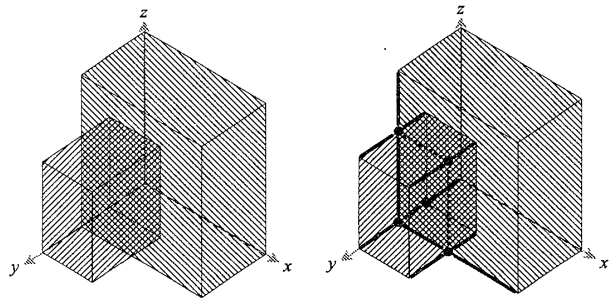

図 5 1 次元以下のフェイスを構成する

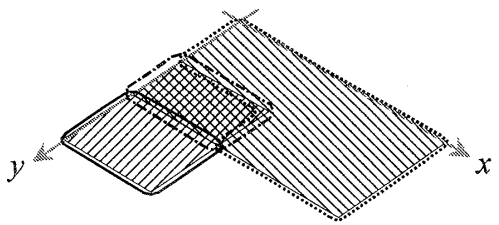

図 6 2 次元以上のフェイスを構成する過程

サブルーチン DIVIDE は, 引数として与えられたアレンジメント中の 2 つのフェイスの組合わせ全てについて, 2 つの重なったフェイスを, 一方のフェイスのみに含まれた部分, 他方のみに含まれた部分, 両方に含まれた部分に分割する. なお, ここに示したアルゴリズムは簡易版であり, 実際に実装したものはこのアルゴリズムにさらなる高速化を施したものである.

サブルーチン DIVIDE の動作は多少複雑であるので, 以下に動作の概要を述べる. 図 5(a) に示す, 2 つの重なった直方体の分割を例に, 多面体分割アルゴリズムの概要について述べる.

最初に, 2 つのフェイスの交点として求まる頂点を作る (サブルーチン PHASE_A の 1行). 図 5(b) の黒い点が新しくできる頂点である[3]. また, この頂点が 1-フェイス上にある場合はそれらの 1-フェイスを分割する. この処理によって分割された辺は, 図 5(b) の太い実線で示された辺である. 次に, 平面同士が交差する場所に 1-フェイスを作る. 入力のフェイスが 3 次元空間にある場合は平面同士が交差する場所のみを考えれば良いが, それ以上の次元になると, 例えば 5 次元では 2-フェイスと 4-フェイスの交わりにも, 3-フェイスと 3-フェイスの交わりにも 1-フェイスができる. このため, 20行から 28行までの処理が必要になる. このようにして新しくできた辺が, 図 5(b) の太い点線で示された辺である.

次に, サブルーチン PHASE_B では 2 次元以上のフェイスを順次作っていく. 入力のフェイスの隣り合う 2 本の 1 次元の descendant フェイスに注目し, それらを含む平面を考える. 以下, この平面が $z = 0$ である場合について述べる. 図 5の領域の閉包を平面 $z = 0$ で切断した領域が図 6である. 図 6の実線で囲まれた部分, 点線で囲まれた部分がそれぞれ一方, 他方のフェイスの閉包にのみ含まれる部分で, 鎖線で囲まれた部分は両方のフェイスの閉包に囲まれる部分である. これらの 3 つの領域の内いずれか一つに選んだ 2 つの 1-フェイスが両方とも含まれている場合, その領域内の隣り合う 1 次元の descendant フェイスを順に辿って全て集める (9行から 16行まで). 次に集めたフェイスをサブフェイスとする 2 次元のフェイスを新しく作る. 全ての 1 次元の descendant フェイスの組合わせに対してこの処理を行うと, 入力のフェイスの descendant フェイスである 2 次元のフェイスに関しては分割後のフェイスが全て出来る. もともとあった入力のフェイスの descendant フェイスである 2 次元のフェイスは不要であるので全て削除する (21行).

以下に用いる関数 flatIncludingFace はフェイスを, そのフェイスを含む同次元のフラットに変換するものである.

Algorithm DIVIDE

◇ 入力 拡張されたアレンジメント A, 限定子 Q
◇ 出力 アレンジメント

```
1       Aから重複するフェイスを取り除く;
2       PHASE_A(A,Q);
```

[3] 必ずしも 1-フェイス上にないことに注意. *ex.* 3 平面の交点など

```
3     for A内の全てのフェイスの組合わせ f1, f2 do
          ▷ 実際に実装したルーチンではここに 3.1.3で述べた
          工夫がなされている
4         if cl f1 ∩ cl f2 ≠ ∅ then
5             PHASE_B(f1, f2,Q);
6             PHASE_B(f2, f1,Q);
7         endif ;
8     next
9     Aから重複するフェイスを取り除く;
10    return A;
11 end
```

Algorithm PHASE_A

◇ 入力 拡張されたアレンジメント A, 限定子 Q
◇ 出力 A の 1 次元以下のフェイスを分割した，拡張されたアレンジメント

```
1  分割によってできる 0-フェイス v を全て作る．また，Q 等によって適切な真偽値を v に割り当てる．また，その 0-フェイスが 1-フェイス上にある場合はその 1-フェイスを分割する
2  for A 内のフェイスの組合わせ (f1, f2) 全てに対して do
       ▷ 実際に実装したルーチンではここに 3.1.3で述べた工夫がなされている
3      if flatIncludingFace(f1) と flatIncludingFace(f2) の交わりが直線にならない then continue ;
4      f1, f2 の descendant フェイスである頂点全ての集合を V1, V2 とする.
5      if V1 ∩ V2 = ∅ then continue ;
6      V := V1 ∩ V2;
7      V から勝手に 2 つの異なる要素を選び，それらの座標を e1, e2 とする.
8      V のある要素の座標を v としたとき，(e1 − e2) · (v − e2) の値の昇順に V の要素全てを整列し，これをリスト L とする.
9      for each e1' ∈ L, 但し最後の要素を除く do
10         e2' := e1' の次の要素
11         if e1' と e2' の中点の座標が f1 と f2 の両方に含まれていない then continue ; endif
12         if e1' と e2' をサブフェイスとする 1-フェイスがない then
13             サブフェイスを e1' と e2' とするフェイスを作り，そのポインタを ls に代入する.
14             Q と f1, f2 に割り当てられた真偽値から ls に適切な真偽値を割り当てる.
15         else
16             e1' と e2' をサブフェイスとする 1-フェイスへのポインタを ls に代入する.
17         endif
18         f1' := f1;
19         while true do
20             if f1' のサブフェイス fs のなかで，
               flatIncludingFace(f) と f2 の交わりが
               flatIncludingFace(f1) と
               flatIncludingFace(f2) の交わりと等しく，かつ fs が e1' と e2' の中点を含む，という条件を満たす fs がある then
21                 f1' := fs;
22             else
23                 break ;
               endif
24         endwhile
25         while true do
26             if f2' のサブフェイス fs のなかで，
               flatIncludingFace(f) と f1 の交わりが
               flatIncludingFace(f1) と
               flatIncludingFace(f2) の交わりと等しく，かつ fs が e1' と e2' の中点を含む，という条件を満たす fs がある then
27                 f2' := fs;
28             else
29                 break ;
               endif
30         endwhile
31         ls のスーパーフェイスに f1', f2' を加える.
32     next
33 next
34 end
```

Algorithm PHASE_B

◇ 入力 1 次元以下のフェイスの重なりのない，拡張されたアレンジメント A, A に含まれるフェイスへのポインタ f_1, f_2
◇ 出力 f_1 と f_2 およびそれらの descendant フェイス同士の重なりのない，拡張されたアレンジメント

```
1  for each f1 の descendant フェイスで，かつ隣り合う 2 つの (dim-1)-フェイス f, g do
2      if flatIncludingFace(f) と，flatIncludingFace(g) が等しい then continue ;
3      if f と g の両方をサブフェイスとする NEW フラグのついた dim-フェイスが存在する then continue ;
4      f と g の両方を含む (dim)-フラット fl を求める.
       ▷ f1 と fl の領域の積を f1' とする.
       ▷ f2 と fl の領域の積を f2' とする.
       ▷ 領域 Rg1 を f1' にのみ含まれる領域全ての閉包とする.
       ▷ 領域 Rg2 を f2' にのみ含まれる領域全ての閉包とする.
       ▷ 領域 RgB を f1' と f2' の両方に含まれる領域全ての閉包とする.
5      if f ∈ Rg1 ∧ g ∈ Rg1 then rg := Rg1;
6      else if f ∈ Rg2 ∧ g ∈ Rg2 then rg := Rg2;
7      else if f ∈ RgB ∧ g ∈ RgB then rg := RgB;
8      else continue ; endif ;
9      リスト S = {f,g},S' = ∅;
10     for each t ∈ S 但し，最初の要素は除く，リストの要素順に do
11         for each t の隣の (dim-1)-フェイス u do
12             if u ∉ S ∧ u ⊆ cl rg then
13                 S の最後に u を加える;
14             endif
15         next ;
16     next ;
17     S の要素をサブフェイスとする新しい dim-フェイス n を作り，n に NEW フラグを設定する;
18     Q と truth(f1), truth(f2) 等から n に適切な真偽値を割り当てる.
19 next ;
20 f1 または f2 の dim-次元の descendant フェイスで NEW フラグのないものを削除.
21 NEW フラグのついたフェイスは NEW フラグを解除し，そのフェイスが連続でない場合，連続な複数の dim-フェイスに分割する.
22 end
```

4 評価実験と考察

3.1で述べた高速化を施した判定ルーチンと行ってないルーチンを実装し，評価実験を行った.

入力式に含まれる変数が 2 つの場合に，不等式の数を 1 から 20 まで，変数が 3 つの場合に，不等式の数を 1 から 9 まで，変化させ，各係数をランダムな数として，3 回ずつ

測定した．各測定はSunOS 5.6の動作するワークステーション(UltraSPARC-II 248MHz, メモリ512Mバイト, スワップ領域1Gバイト)上で行った．なお，式の形を変化させても判定時間はあまり変わらなかった．判定時間を図7，図8に示す．変数の数が2個, 3個の場合のFerranteとRackoffのアルゴリズムはそれぞれ不等式の数が6個, 3個を越える場合判定時間が600秒を越えたので，判定時間の計測を打ち切り，結果はのせていない．

また，アレンジメントの形が最も複雑になると思われる，次のような例に対しても判定時間を計測した．ランダムな数の組合わせ $r_1, r_2, ..., r_{80}, r'_1, r'_2, ..., r'_{80}$(ただし $r'_1, ..., r'_{80}$ は0以上20未満の数)に対し，以下の式の判定を行った．

$$\forall x \exists y \bigvee_{1 \le k \le 80} (x > r_k \wedge y > r'_k \wedge x + y < r_k + r'_k + 1)$$

判定時間の平均値は582秒であり，図7に示した結果とあまり変わらなかった．

提案する高速化によって，時間計算量オーダは変わらないが，実行上の高速化は果たせているといえる．

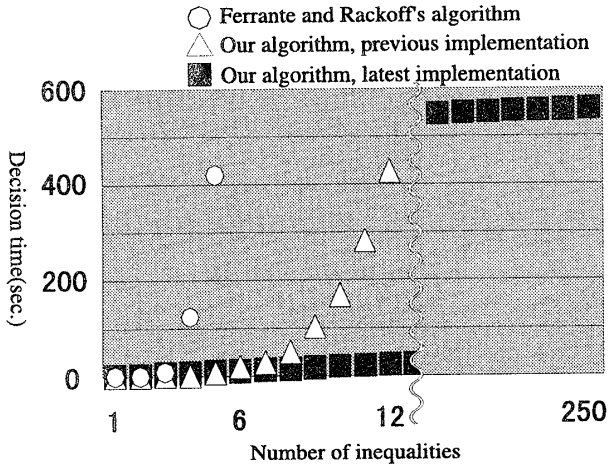

図 7 高速化を施した場合と施さない場合の判定時間
変数の数が2個の場合

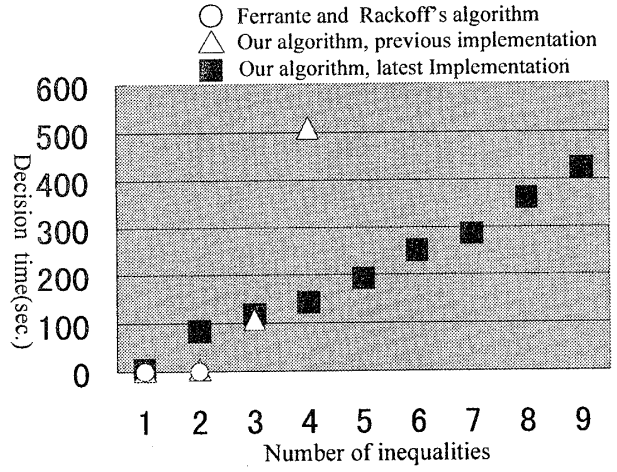

図 8 高速化を施した場合と施さない場合の判定時間
変数の数が3個の場合

また，3.1.3で述べた工夫を施した場合と施さない場合について判定時間の比較を行ったが，わずかしか(1/50程度)差はなかった．この工夫は前回の実装でフェイスの数が計測した範囲において数万といった数になることから考えたものであるが，今回の実装ではフェイスの数が計測した範囲において多くても数百程度にしかならず，効果が少なかったといえる．今後判定アルゴリズムの更なる高速化によってフェイスの数が増えた場合にも判定できるようになれば，この工夫の効果が現れると思われる．

式の形を変化させても判定時間が余り変わらない理由として，以下の事柄が考えられる．現在の真偽判定ルーチンの制限としてアレンジメントに含まれるフェイス全ての分割が終わってからでないと簡単化を行えないことがあり，アレンジメントに含まれるフェイス全てを分割する過程の中でアレンジメントに含まれるフェイスの数はかなり増える．判定時間はこの数に主に依存し，式の複雑さには依存しない．ルーチンを改良し，頻繁に簡単化を行うようにすることで，式の複雑さと判定時間の関係はさらに明確になると思われる．

5 あとがき

本稿では著者らが以前提案したPRP文真偽判定アルゴリズムに対するさらなる高速化手法とその評価について述べた．

今後の課題として，今回提案した手法を使った真偽判定ルーチンの判定時間についてさらに詳しく調べ，ルーチンを実用的な例題に適用し，評価を行うことなどが挙げられる．

参考文献

[1] 東野輝夫, 北道淳司, 谷口健一：“整数上の線形制約の処理と応用”, コンピュータソフトウェア, Vol.9, **6**, 31-39 (1992).
[2] Hopcroft, J.E. and Ullmann, J.D.: “Introduction to Automata Theory, Languages and Computation,” Addison-Wesley (1979).
[3] Ferrante. J and Rackoff. C : “A decision procedure for the first order theory of real addition with order,” SIAM J. Comput. **4**, pp.69-76 (1975).
[4] 柴田直樹, 岡野浩三, 東野輝夫, 谷口健一：“冠頭標準形有理数プレスブルガー文の真偽判定アルゴリズムの提案”, 信学会論文誌 Vol. J82-D-I **6**, pp.691-700 (1999).
[5] Edelsbrunner, H.: “Algorithms in Combinatorial Geometry,” Springer-Verlag Berlin Heidelberg (1987).
[6] 柴田直樹, 岡野浩三, 東野輝夫, 谷口健一：“組合わせ幾何を用いた有理数プレスブルガー文真偽判定アルゴリズムにおける投影操作の高速化”, 第57回情処全大予稿集 3C−08 (1998).
[7] Suri. S , Hubbard. P and Hughes. J : “Collision Detection in Aspect and Scale Bounded Polyhedra.”, Proc. of 9th Annual Symposium on Discrete Algorithms, San Francisco (1998).