データ駆動アプローチによるプログラムの線形写像化

石澤 拓哉　小島 健介　末永 幸平

本論文ではプログラムの線形写像表現を計算するアルゴリズムを提案する，プログラム c について，線形写像 A がプログラム c の線形写像表現であるとは，ベクトル $\vec{x}$ で表された任意のプログラムの状態について，$\vec{x}$ を初期状態として c を実行して停止するならば終了状態が $A\vec{x}$ となることをいう．線形写像表現はプログラム検証やプログラムの最適化のために有用である．提案アルゴリズムは以下の操作を繰り返し行うことで線形写像表現を計算する．まずプログラム c の線形写像表現として全ての要素が未定係数である行列 T を生成する．次にテストとなる状態を生成し，そのテストを初期状態として c を実行した結果から T の未定係数を決定する．求めた T が正しい線形写像となるかを外部のソルバを用いて検証する．検証に成功すれば T を出力し，失敗すればソルバが返す反例をテストに加える．本論文では提案アルゴリズムが停止性，健全性，完全性を満たすことを証明する．

We propose an algorithm to compute a linear-mapping representation (LMR) of a program. Given a program c, an LMR of c is a matrix A such that $A\vec{x}$ is identical to the result of executing c with the state expressed as a vector $\vec{x}$. Such representation is useful in program verification and program optimization. Our algorithm computes an LMR of program c by iterating the following steps. First, the algorithm designates a template T of an LMR of c. T is a matrix elements of which are unknowns. Then, the algorithm randomly generates and executes a test to determine the unknowns in T. The algorithm then tries to verify that the estimated matrix is a correct LMR of c using an external solver. If the verification fails and the solver returns a counterexample, the algorithm adds it to the set of tests that are used in the next iteration. We show the soundness, the completeness and the convergence of our algorithm.

1　はじめに

静的検証はプログラムがあらかじめ定められた性質を満たすことをプログラムの実行前に証明する手法である．様々な静的検証手法が研究されており，近年は実用に供されている手法も多く存在する [1,4,6]．

ループを含む命令型プログラムの静的検証においては良い**ループ不変条件**を発見することが重要である [15]．ループ不変条件とは，プログラムの制御がループの先頭に到達したときに必ず成り立つ条件のことである．

例 1.1. 以下のプログラム

$$x := N; sum := 0$$
$$\textbf{while}\, x \neq 0\ \textbf{do}$$
$$(x, sum) := (x - 1, sum + x)$$
$$\textbf{end}$$

において，$N \geq 0$ であれば，1 から x までの総和と sum の値の和はループの先頭に制御が到達した時点では必ず 1 から N までの総和に等しいので，$\frac{x(x+1)}{2} + sum - \frac{N(N+1)}{2} = 0$ はループ不変条件である．このループ不変条件を使えば，ループを抜けたときに $x = 0$ であることと併せて，プログラム終了時点で $sum = \frac{N(N+1)}{2}$ であることが証明できる．

ループ不変条件を自動的に発見する手法は，プログラム検証の研究において重要なテーマであり，保証したい性質に応じて様々な手法がこれまでに提案されて

Computing Linear-Mapping Representations of Programs by Data-Driven Approach
Takuya Ishizawa, 京都大学, Kyoto University.
Kensuke Kojima, 京都大学 / JST CREST, Kyoto University / JST CREST.
Kohei Suenaga, 京都大学 / JST PRESTO, Kyoto University / JST PRESTO.

いる．特にプログラム中に現れる変数 $x_1, \ldots, x_n$ の多項式 $p(x_1, \ldots, x_n)$ を用いて $p(x_1, \ldots, x_n) = 0$ の形に書けるループ不変条件（**代数的不変条件**）の発見手法は近年においても多くの手法が提案されている [2, 3, 9–14] ．

De Oliveira らは代数的不変条件の発見手法として**固有ベクトル法**と呼ばれる手法を提案した [7]．この手法はループ文 **while** b **do** c' のループ不変条件を求めるために，c' の実行を表現した線形写像の表現行列 $A_{c'}$ を計算し，$A_{c'}$ の転置行列 ${A_{c'}}^t$ の固有ベクトルからループ不変条件を求める手法である．†1

固有ベクトル法においては，算術式のベクトル $\mathcal{B}$ をユーザが入力として与え，その算術式の値がループ本体の一回の実行でどのように変化するかを線形写像で表現する．ループ本体の一回の実行が線形写像 A を用いて $\mathcal{B} \mapsto A\mathcal{B}$ と書けるとして，A の転置行列 A^t に固有値 1 の固有ベクトル $\vec{c}$ があったとしよう．このときに $\vec{c}$ と $\mathcal{B}$ の要素ごとの積の和 †2$\vec{c}^t\mathcal{B}$ の値はループ一回の実行で変化しない．実際にループ一回の実行によって算術式 $\vec{c}^t\mathcal{B}$ の値は $\vec{c}^tA\mathcal{B}$ に変化するが，$\vec{c}^tA$ は $(A^t\vec{c})^t$ に等しく，固有ベクトルの定義から $A^t\vec{c}$ は $\vec{c}$ に等しいため，$\vec{c}^tA\mathcal{B}$ の値は $\vec{c}^t\mathcal{B}$ に等しい．従って，初めにこのループの先頭に制御が到達したときの $\vec{c}^t\mathcal{B}$ の値を p とすれば，$\vec{c}^t\mathcal{B} - p = 0$ はループ不変条件となっている．

例 1.2. 例 1.1 のプログラムを用いて固有ベクトル法を説明する．ここでは算術式の組として $(x, sum, x^2, 1)$ が与えられたとしよう．ループ本体のコマンド $(x, sum) := (x - 1, sum + x)$ の実行によって，それぞれの算術式は $(x - 1, sum + x, x^2 - 2x + 1, 1)$ に変化する．この算術式の変化は，算術式の組をベクトルと見れば，以下の線形写像として表現することができる．

$$\begin{pmatrix} x \\ sum \\ x^2 \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} 1 & 0 & 0 & -1 \\ 1 & 1 & 0 & 0 \\ -2 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ sum \\ x^2 \\ 1 \end{pmatrix}$$

この行列の転置行列の固有値 1 の固有ベクトルはすべて以下の 3 つのベクトルの線形結合として書ける．

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 2 \\ 1 \\ 0 \end{pmatrix}$$

このうち 3 つ目のベクトル $(1, 2, 1, 0)^t$ と算術式のベクトル $(x, sum, x^2, 1)^t$ との要素ごとの積の和は $x + 2sum + x^2$ となり，この式の値はループ本体一回の実行では変化しない．このループの先頭に初めて制御が到達したときのこの式の値は $N + N^2$ なので，$x + 2sum + x^2 = N + N^2$ はループ不変条件である．

固有ベクトル法は代数的不変条件を生成する有力な手法であるが，ネストしたループを含むプログラムには適用できないという欠点がある．

例 1.3. 以下のプログラムは，例 1.1 のループ本体をさらにループ文を用いて書いたプログラムである．

$x := N; y := 0; sum := 0;$
while $x \neq 0$ **do**
  $y := x;$
  while $y \neq 0$ **do** $\cdots\cdots (*)$
    $(y, sum) := (y - 1, sum + 1);$
  end
  $x := x - 1;$
end

このプログラムは例 1.1 のプログラムと同じ意味を持つが，ネストしたループを含むために固有ベクトル法を適用することができない．

固有ベクトル法をネストしたループを含むプログラムに拡張するにあたっての困難は，ループ文自体の動作を表現する線形写像の計算にある．例えば例 1.3 のプログラムに固有ベクトル法を適用するためには，$(*)$ で示した内側のループ文の実行を線形写像で表現する必要がある．この写像が $(y, sum) \mapsto (0, y)$ であることを計算するのは必ずしも自明ではない．

本論文では，ネストしたループを含むプログラムに

†1 以下では「線形写像」という語をその表現行列を指すために用いることがある．

†2 $\mathcal{B}$ が算術式のベクトルなので，この要素ごとの積の和は算術式である．

固有ベクトル法を拡張することを目指して，与えられたプログラムの線形写像表現を求める手法を提案する．提案手法を用いることで，例 1.3 のようなプログラムにおいてもループ本体の動作を線形写像として表現することができ，固有ベクトル法を拡張することが可能になると期待される．(但し，本論文においてはプログラムの動作の線形写像表現を求める手法のみを扱い，固有ベクトル法の拡張にまでは踏み込まない．)

提案手法の特徴は，与えられたプログラムの線形写像表現を**データ駆動アプローチ**を用いて求める点にある．ここでデータ駆動アプローチとは，プログラムに対するテストを自動生成し，そのテストの実行結果から線形写像表現を推測する手法である．線形写像表現の推測後に，推測された表現が正しいことを検証するフェーズを設けることで，正しい線形写像表現が計算されることを保証する．また，この検証が失敗した際に，検証過程で生じた反例をテストとして用いることで，線形写像表現を決定するのに必要なテストをシステマティックに生成することができる．

本論文では与えられたプログラムの線形写像表現を計算するアルゴリズムを提案し，その正しさを証明する．アルゴリズムの実装と評価，並びに固有ベクトル法の拡張は今後の課題である．

本論文の構成を説明する．2 節では論文中で使用する定義を導入する．3 節におけるアルゴリズムの説明では，まず 3.1 節において動作例を用いてアルゴリズムの直観的な説明を行った後に，3.2 節でフォーマルな定義を述べる．4 節ではアルゴリズムの停止性，健全性，完全性を示す．5 節で関連研究を論じた後に，6 節で結論と今後の課題を述べる．

2 準備

2.1 定義

$\mathbf{PV}$ をプログラム変数の集合，$\mathcal{M}$ をプログラム変数上の単項式全体の集合とする．$\mathbf{PV}$ の要素を表すためにメタ変数 $x, y, z, \ldots$ を用いる．$(-)^t$ はベクトルもしくは行列の転置を表す．

状態とは $\mathbf{PV}$ から $\mathbb{R}$ への部分関数である．メタ変数 σ で状態を表す．本論文ではプログラム変数間に暗黙に全順序をつけることで状態をベクトルとして扱う．例えば，状態 $\{x \mapsto 1, y \mapsto 2, z \mapsto 3\}$ は $x < y < z$ と順序を仮定することで，$(1,2,3)^t$ と表せる．以降では部分関数としての状態とベクトルとしての状態を区別しない．

基底は単項式のベクトル，すなわち $\mathcal{M}^n$ の要素である．基底 $\mathcal{B} \in \mathcal{M}$ 中の各単項式を状態 σ に従って評価した $\mathbb{R}^n$ の要素を $[\![\mathcal{B}]\!]_\sigma$ と書く．例えば，基底 $\mathcal{B}$ を $(x, y, x^2, xy, 1)^t$ とおき，順序 $x < y$ のもとで状態 σ を $(2,3)^t$ とおくと，$[\![\mathcal{B}]\!]_\sigma = (2,3,4,6,1)^t$ となる．

2.2 対象言語

以下の言語を対象とする．

$$
\begin{aligned}
c \quad ::= \quad & \mathbf{skip} \\
\mid \ & (x_1, ..., x_n) := (p_1, ..., p_n) \\
\mid \ & c_1 \,;\, c_2 \\
\mid \ & \mathbf{if}\, b\, \mathbf{then}\, c_1\, \mathbf{else}\, c_2 \\
\mid \ & \mathbf{while}\, b\, \mathbf{do}\, c\, \mathbf{end} \\
\mid \ & \mathbf{assume}\, b
\end{aligned}
$$

ここで，x はプログラム変数，b はブール式，p はプログラム変数上の多項式を表す．

skip は空文を表し，状態を変化させない．$(x_1, ..., x_n) := (p_1, ..., p_n)$ は，$1 \leq i \leq n$ に対して，x_i に p_i の評価結果を全て同時に代入する操作である．$c_1 ; c_2$ は c_1 を実行した直後に c_2 を実行する操作である．$\mathbf{if}\, b\, \mathbf{then}\, c_1\, \mathbf{else}\, c_2$ は b を評価して **true** ならば c_1 を実行し，**false** ならば c_2 を実行する操作である．$\mathbf{while}\, b\, \mathbf{do}\, c\, \mathbf{end}$ は b が **false** になるまで c を繰り返し実行する操作である．$\mathbf{assume}\, b$ は実行時に b が **true** ならば何もせず，**false** ならば無限ループする．

$[\![c]\!]_\sigma$ は，プログラム c を初期状態 σ で実行し，停止した時の終了状態を表す．停止しない時は結果は $\bot$ という特別な値になるものとする．

プログラム c の基底 $\mathcal{B}$ における**線形写像表現**（あるいは単に c の線形写像表現）とは，任意の状態 σ について，$[\![c]\!]_\sigma \neq \bot$ ならば $[\![\mathcal{B}]\!]_{[\![c]\!]_\sigma} = A[\![\mathcal{B}]\!]_\sigma$ となる線形写像 A のことである．また，プログラム c を線形写像表現に変換することを**線形写像化**と呼ぶ．例え

ば，プログラム c が以下のように与えられたとする．

assume $x > 0$
while $x \neq 0$ **do**
$(x, y) := (x - 1, y + 1)$
end

基底 $\mathcal{B}$ を $(x, y, 1)^t$ とすると，c の動作は $(x, y, 1)^t \mapsto (0, x + y, 1)^t$ と表現できる．このとき，$\begin{pmatrix} 0 \\ x+y \\ 1 \end{pmatrix}$ は $\begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ と表現できる．したがって $\begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ は基底 $(x, y, 1)^t$ における c の線形写像表現である．

提案するアルゴリズムにおいては**ホーア論理**において導入される概念を用いる．以下では本論文で必要となる概念を，正確な定義を述べること無しに，簡潔に説明する．詳細な説明は Winskel による教科書 [15] を参照されたい．

アサーションとは **PV** の要素を自由変数として含みうる実数の上の一階述語論理式のことである．アサーションをメタ変数 P または Q で表す．アサーション P 中の自由変数の値を状態 σ に従って解釈すると P が成り立つとき $\sigma \models P$ と書く．任意のアサーション P に対して，$\bot \models P$ であると約束する．任意の σ について $\sigma \models P$ であるとき，$\models P$ と書く．

プログラム c とアサーション P, Q について，$\{P\}c\{Q\}$ を**ホーア三つ組**と呼ぶ．ホーア三つ組は直観的には $\{P\}c\{Q\}$ は条件 P を満たす状態からプログラム c を実行して停止すれば，条件 Q を満たすという意味である．具体的には，$\sigma \models P$ ならば $[\![c]\!]_\sigma \models Q$ であるときにホーア三つ組 $\{P\}c\{Q\}$ が成り立つと言い，$\models \{P\}c\{Q\}$ と書く．ホーア三つ組 $\{P\}c\{Q\}$ の P を c の**事前条件**，Q を c の**事後条件**と呼ぶ．

3 アルゴリズム

この節ではプログラム c を与えられたときに，c を線形写像化するアルゴリズム *GuessAndCheck* を提案する．提案アルゴリズムの入力と出力は以下の通りである．

入力 プログラム c，基底 $\mathcal{B}$，アサーション φ．
出力 c の事前条件が φ であると仮定したときの c の $\mathcal{B}$ 上での線形写像表現，もしくはアルゴリズムが失敗したことを示す **fail.**

3.1 アルゴリズムの動作例

GuessAndCheck の特徴は，与えられたプログラムの線形写像表現をデータ駆動アプローチ，すなわち，与えられたプログラムの初期状態を生成し，その実行結果から線形写像表現を求める点である．このアプローチは 5 節で議論する Sharma ら [11,14] の手法に着想を得ている．

この節では，*GuessAndCheck* をフォーマルに定義する前に，以下のプログラム c_{sum} についてこのアルゴリズムがどう動作するかを説明する．

assume $x > 0$
while $x \neq 0$ **do**
$(x, sum) := (x - 1, sum + x)$
end

GuessAndCheck はプログラム c_{sum} の他に基底 $\mathcal{B}$ を入力として受け取る．†3 ここでは $\mathcal{B}$ として $(x, sum, x^2, 1)^t$ を与えたとしよう．c_{sum} の動作は写像 $(x, sum, x^2, 1)^t \mapsto (0, sum + \frac{1}{2}(x + x^2), 0, 1)^t$ で表現できる．また，この写像の $\mathcal{B}$ 上の線形写像表現は $\begin{pmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{2} & 1 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ である．アルゴリズム *GuessAndCheck* はデータ駆動アプローチを用いてこの線形写像表現を求める．

GuessAndCheck は**テスト** σ と**テンプレート行列** $T := (t_{ij})_{1 \leq i, j \leq |\mathcal{B}|}$ を生成する．テストはプログラムの開始状態を表すベクトルであり，ランダムに生成すればよい．†4 テンプレート行列は行列の各要素が未定

†3 この節では説明の簡単のために，プログラムの事前条件をとして与えられるアサーション φ のことは考えない．
†4 後に示す *GuessAndCheck* の定義においては，プログラム c が満たすべき事前条件を考慮に入れてテストを生成するようになっている．

係数である行列であり，*GuessAndCheck* は生成されたテストの実行結果からテンプレート行列中の未定係数の間の制約を生成し，その制約の解から線形写像化を徐々に求める．

テストとして $(x, sum)^t$ 上の状態 $\sigma_1 := (1,0)^t$ が生成されたとしよう．*GuessAndCheck* は終了状態 $[\![c_{sum}]\!]_{\sigma_1}$ を求める．†5 ここでは終了状態は $[\![c_{sum}]\!]_{\sigma_1} = (0,1)^t$ となる．

T が c_{sum} の線形写像表現であるためには，c_{sum} が停止する任意の状態 σ について，$[\![\mathcal{B}]\!]_{[\![c_{sum}]\!]\sigma} = T[\![\mathcal{B}]\!]_\sigma$ が成り立つ必要がある．この制約は，テンプレート行列中の未定係数の間の連立一次方程式に帰着できるので，SMT ソルバ等を用いて容易に解くことができる．σ を上記の σ_1 としてこの制約を解き，その解を T に代入すると $T = \begin{pmatrix} t_{11} & t_{12} & t_{13} & -t_{11}-t_{13} \\ t_{21} & t_{22} & t_{23} & 1-t_{21}-t_{23} \\ t_{31} & t_{32} & t_{33} & -t_{31}-t_{33} \\ t_{41} & t_{42} & t_{43} & 1-t_{41}-t_{43} \end{pmatrix}$ となる．

次にここで求めた T が c_{sum} の線形写像表現になっているかどうかを検証する．これには，任意の状態 $\vec{a}$ と任意の t_{ij} で $\models \{\mathcal{B} = [\![\mathcal{B}]\!]_{\vec{a}}\} c_{sum} \{\mathcal{B} = T[\![\mathcal{B}]\!]_{\vec{a}}\}$ が成り立つかをプログラム検証器や SMT ソルバを用いて調べればよい．しかし，T 中に未解決の未定係数が多く存在するために，これらのソフトウェアを用いてこの三つ組の成否を判定することは多くの場合難しい．

そこで *GuessAndCheck* では，T 中に残る未定係数に 0 を代入して得られる行列 T'（この行列を**インスタンス化された行列**と呼ぶ）が条件 $\models \{\mathcal{B} = [\![\mathcal{B}]\!]_{\vec{a}}\} c_{sum} \{\mathcal{B} = T'[\![\mathcal{B}]\!]_{\vec{a}}\}$ を満たすかどうかを検証する．†6 この条件を以下で C と呼ぶ．もし T' が条件 C を満たすならば，T' は c_{sum} の正しい線形写像表現になっているので，T' を出力すればよい．

†5 実際には c_{sum} が σ_1 を初期状態にとると停止しない場合も考えられるので，あらかじめ定めた実行ステップ数を超えても c_{sum} が停止しない場合は，アルゴリズム自体を異常終了することにする．

†6 テンプレート行列をインスタンス化する際に未定係数に代入する値は任意の値でよく，0 でなくともアルゴリズムの正しさには影響はない．ここでは簡単のために 0 を代入するものとして説明する．

もし C が成り立たない場合は，C の反例となる $\vec{a}$ が存在するはずなので，$\vec{a}$ をテスト，T をテンプレート行列として，再びアルゴリズムを動作させる．なお，求まった行列が正しい線形写像表現になっているかどうかの検証の際に，テンプレート行列 T を用いずにインスタンス化された行列 T' を用いても，最終的には線形写像表現が（存在するならば）求まることを定理 4.3 で述べる．

この例では $T' = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ は c_{sum} の動作を正しく表してはいない．なぜならば，$\sigma_2 := (2,0)^t$ とすれば，$[\![c_{sum}]\!]_{\sigma_2} := (0,3)^t$ となり，$[\![\mathcal{B}]\!]_{[\![c_{sum}]\!]_{\sigma_2}} = T'[\![\mathcal{B}]\!]_{\sigma_2}$ を満たさないからである．よって，反例 σ_2 をテスト，テンプレート行列を T として再びアルゴリズムを動作させる．

始めの反復と同様に制約 $[\![\mathcal{B}]\!]_{[\![c_{sum}]\!]_{\sigma_2}} = T[\![\mathcal{B}]\!]_{\sigma_2}$ を解くと $T = \begin{pmatrix} -3t_{13} & t_{12} & t_{13} & 2t_{13} \\ 2-3t_{23} & t_{22} & t_{23} & -1+2t_{23} \\ -3t_{33} & t_{32} & t_{33} & 2t_{33} \\ -3t_{43} & t_{42} & t_{43} & 1+2t_{43} \end{pmatrix}$ が解として得られる．この T のインスタンス化された行列は $T' = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ となるが，これもまた c_{sum} の動作を正しく表してはいない．反例としては $\sigma_3 = (3,0)^t$ がとれる．よって，σ_3 をテスト，テンプレート行列を T として再びアルゴリズムを動作させる．

反例 σ_3 のもとで制約を解くと，$T = \begin{pmatrix} 0 & t_{12} & 0 & 0 \\ \frac{1}{2} & t_{22} & \frac{1}{2} & 0 \\ 0 & t_{32} & 0 & 0 \\ 0 & t_{42} & 0 & 0 \end{pmatrix}$ が解として得られる．インスタンス化された行列は $T' = \begin{pmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$ となるが，反例として $\sigma_4 = (1,1)^t$ が存在する．σ_4 をテ

スト，T をテンプレート行列として再びアルゴリズムを動作させ，制約を解くと $T = \begin{pmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{2} & 1 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ が得られる．T' は条件 C を満たすため，c_{sum} の動作を正しく表している．よって，*GuessAndCheck* は T' を出力する，この T' が c_{sum} の線形写像表現となる．

3.2 アルゴリズム *GuessAndCheck*

input ：プログラム c，基底 $\mathcal{B}$，アサーション φ
output: c の $\mathcal{B}$ 上での線形写像表現

1 $\mathcal{T} \leftarrow \mathbb{R}^{|\mathcal{B}|\times|\mathcal{B}|}$;
2 $\sigma \leftarrow GenTest_{\mathcal{B}}(\varphi)$;
3 **while true do**
4 　**try** $\sigma' \leftarrow Execute(c, \sigma)$ **with** $timeout \rightarrow$ **fail**;
5 　$\mathcal{T}' \leftarrow \{T \in \mathcal{T} \mid [\![\mathcal{B}]\!]_{[\![c]\!]_\sigma} = T[\![\mathcal{B}]\!]_\sigma\}$;
6 　**if** $\mathcal{T}' = \emptyset$ **then**
7 　　**fail**;
8 　**end**
9 　$\mathcal{T} \leftarrow \mathcal{T}'$;
10 　$T' \leftarrow Choose_{\mathcal{B}}(\mathcal{T})$;
11 　**if** $\forall \vec{a}.\varphi[\vec{a}/\vec{x}] \implies \models \{\mathcal{B} = [\![\mathcal{B}]\!]_{\vec{a}}\}c\{\mathcal{B} = T'[\![\mathcal{B}]\!]_{\vec{a}}\}$ **が恒真 then**
12 　　**return** T'
13 　**end**
14 　σ を $\varphi[\sigma/\vec{x}] \wedge \not\models \{\mathcal{B} = [\![\mathcal{B}]\!]_\sigma\}c\{\mathcal{B} = T'[\![\mathcal{B}]\!]_\sigma\}$ を満たす状態とする．
15 **end**

Algorithm 1: アルゴリズム *GuessAndCheck*.

GuessAndCheck の定義をアルゴリズム 1 に示す．このアルゴリズムはプログラム c と基底 $\mathcal{B}$ とアサーション φ を入力とし，c の $\mathcal{B}$ 上での線形写像表現を出力する．条件 φ は c の事前条件である．条件 φ は *GuessAndCheck* 中で事前条件を満たすテストのみが生成されるようにするために用いられる．一般には線形写像化できないプログラムであっても，事前条件を限定することで線形写像化できる場合があり，条件 φ によってこのようなプログラムも扱えるようにする．†7

アルゴリズム 1 中では以下の補助的な手続きを用いている．

- *Choose*
  集合を入力としてその集合の要素を任意に一つ選んで返す手続き．
- *GenTest*
  アサーション φ を入力として φ を満たす状態を任意に一つ返す手続き．
- *Execute*
  プログラム c と状態 σ を入力として，c を入力 σ のもとで実行した後の状態 $[\![c]\!]_\sigma$ を返す手続き．事前に設定した計算ステップ数を超えても c が停止しなければ，例外 *timeout* を発生させる．

以下ではアルゴリズム 1 中で重要な箇所について説明する．

- 3.1 節の説明においてはテンプレート行列 T を用いて線形写像表現を求めたが，アルゴリズム 1 中ではテンプレート行列を明示的には扱わず，代わりにテンプレート行列のインスタンスの集合 $\mathcal{T}$ を扱っている．これは，4 節でのアルゴリズムの性質の議論を簡潔にするためである．1 行目は 3.1 節のテンプレート行列の生成，5 行目はテンプレート行列 T について $[\![\mathcal{B}]\!]_{[\![c]\!]_\sigma} = T[\![\mathcal{B}]\!]_\sigma$ を解くことにそれぞれ対応している．
- 11 行目においては $\forall \vec{a}.\varphi[\vec{a}/\vec{x}] \implies \models \{\mathcal{B} = [\![\mathcal{B}]\!]_{\vec{a}}\}c\{\mathcal{B} = T'[\![\mathcal{B}]\!]_{\vec{a}}\}$ が恒真であるかを判定する必要がある．この条件が満たされれば，φ を満たす状態については，T' は c の線形写像表現になる．この判定には SMT ソルバ [6] を用いることを想定している．

4 性質

この章では 3 章で示したアルゴリズム *GuessAndCheck* の性質を示す．補題の証明は付録 A

†7 実際に 3.1 節の動作例においては，事前条件 $x > 0$ を考慮しなければ線形写像化することができない．

に述べる．

4.1 停止性

定理 4.1 (停止性)．*GuessAndCheck* は任意の入力に対して停止して線形写像表現を返すか **fail** する．

これを示すためにいくつかの補題を示す．

定義 4.1. ベクトル空間 V の空でない部分集合 A が *affine subspece* であるとは，ある $a \in A$ について $A - a = \{b - a | a \in A\}$ が V の線形部分空間となることである．

Affine subspace の次元を以下のように定義する．

定義 4.2. A が affine subspace のとき，$a \in A$ について $\dim(A - a)$ を affine subspace の**次元**と言い，$\dim A$ と書く．

以下の補題は，affine subspace の次元が well-defined であることを保証する．

補題 4.1. A がベクトル空間 V の affine subspace のとき，$A - a$ は $a \in A$ の取り方によらず同じ集合となる．

以下の補題は，Algorithm 1 の 5 において $\mathcal{T}'$ が必ず affine subspace になることを保証する．

補題 4.2. A が行列の成す線形空間 $R^{m \times n}$ の affine subspace であるとき，$u \in R^m, v \in R^n$ について $A'_{u,v} := \{T \in A | u = Tv\}$ が空でないとする．このとき，$A'_{u,v}$ は $R^{m \times n}$ の affine subspace である．

補題 4.3. A, B が affine subspace で，$B \subseteq A$ とする．このとき $\dim B \leq \dim A$ となる．また，等号が成り立つのは $A = B$ のときに限る．

補題 4.4. 任意の基底 $\mathcal{B}$ と affine subspace $\mathcal{T} \subseteq R^{|\mathcal{B}| \times |\mathcal{B}|}$ と $T' \in \mathcal{T}$ について，σ を $\varphi[\sigma/\vec{x}] \wedge \not\models \{\mathcal{B} = [\![\mathcal{B}]\!]_\sigma\} c \{\mathcal{B} = T'[\![\mathcal{B}]\!]_\sigma\}$ を満たす任意の状態ベクトルとする．ここで，$\mathcal{T}' := \{T \in \mathcal{T} | [\![\mathcal{B}]\!]_{[\![c]\!]_\sigma} = T[\![\mathcal{B}]\!]_\sigma\}$ とおくと，$\mathcal{T}' \subsetneq \mathcal{T}$．

(証明) (定理 4.1) 補題 4.4 より，Algorithm 1 の 5 行目の $\mathcal{T}'$ はループを実行するたびに真に小さくなることが分かる．これと補題 4.3 より，affine subspace $\mathcal{T}'$ の次元がループのたびに真に小さくなることが分かる．$\mathcal{T}'$ の次元は最大で $|\mathcal{B}|^2$ なので，*GuessAndCheck* は高々$|\mathcal{B}|^2$ 回の反復で線形写像表現を出力するか，$\mathcal{T}'$ が空集合となり **fail** する．

4.2 健全性

定理 4.2 (健全性)．任意のプログラム c，基底 $\mathcal{B}$，事前条件 φ について，$T := \mathit{GuessAndCheck}(c, \mathcal{B}, \varphi)$ が **fail** でなかったとする．任意の状態 $\vec{a}$ について $\varphi[\vec{a}/\vec{x}] \implies \{\mathcal{B} = [\![\mathcal{B}]\!]_{\vec{a}}\} c \{\mathcal{B} = T[\![\mathcal{B}]\!]_{\vec{a}}\}$ が成り立つ．

(証明) $\mathit{GuessAndCheck}(c, \mathcal{B}, \varphi)$ が **fail** 以外を返すのは 11 行目の検証条件が真となった場合のみであることから明らか．

4.3 完全性

定理 4.3 (完全性)．プログラム c が任意の初期状態について停止すると仮定し，$\mathcal{S} := \{T \in \mathbb{R}^{|\mathcal{B}| \times |\mathcal{B}|} | \forall \vec{a}. \varphi[\vec{a}/\vec{x}] \implies \models \{\mathcal{B} = [\![\mathcal{B}]\!]_{\vec{a}}\} c \{\mathcal{B} = T[\![\mathcal{B}]\!]_{\vec{a}}\}\}$ とする．$\mathcal{S}$ が空でなければ，$\mathit{GuessAndCheck}(c, \mathcal{B}, \varphi)$ は $\mathcal{S}$ の元を返す．

定理 4.3 は，プログラム c がすべての初期状態について停止することを要求する．状態 σ を初期状態とすると c が停止しない場合において，*GenTest* が σ を生成すると，4 行目でアルゴリズムが失敗するため，実際には c の線形写像表現が存在する場合においても，それを返すことができないためである．

補題 4.5. $\mathcal{S}$ を定理 4.3 中のとおりに定義する．$\mathcal{T}'_0 := \mathbb{R}^{|\mathcal{B}| \times |\mathcal{B}|}$ とし，n 回目にアルゴリズム 1 中の 5 行目が実行されたときの $\mathcal{T}'$ の値を $\mathcal{T}'_n$ と書く．任意の n で $\mathcal{S} \subseteq \mathcal{T}'_n$．

(証明) (定理 4.3) 補題 4.5 より，5 行目において必ず $\mathcal{S} \subseteq \mathcal{T}'$ となり，$\mathcal{S}$ が空でないとき，$\mathcal{T}'$ も空でない（すなわちアルゴリズムは **fail** を返さない．）よって定理 4.1 より，*GuessAndCheck* は線形写像 T' を出力する．検証条件より $T' \in \mathcal{S}$ となることは明らか．

5 関連研究

本論文で提案したアルゴリズムは Sharma ら [14] の手法に着想を得ている．彼らはプログラムの実行から得られるデータから不変条件を推測し検証する手法を提案した．彼らは後に基底を自動的に求めながら

プログラムの事前条件を計算するアルゴリズムも提案している [11]．本論文ではデータ駆動アプローチをプログラムの線形写像化の問題に適用した．彼らの手法との主な差分は，線形化の候補が正しいことの検証を行う際の効率を向上するために，我々の手法では推定された線形写像に残存する未定係数に 0 を代入した上で検証を行う点である．これがアルゴリズムの停止性と完全性を損なわないことが保証されている．彼らの基底の自動生成へのアプローチの我々の手法への適用は重要な将来の課題である．

Sharma らの手法の他にも代数的不変条件を自動生成する研究はここ十数年研究が進んでいる．これらの研究は主に多項式代数を用いて代数的不変条件を生成する．Sankaranarayanan ら [13] はグレブナー基底を用いて代数的不変条件を自動生成する手法を提案した．彼らのアルゴリズムは，ループ文の代数的不変条件を与える多項式の集合のグレブナー基底を求めることで代数的不変条件を計算する．Müller-Olm ら [10] は抽象解釈 [5] を用いた手法を提案している．Rodríguez-Carbonell ら [12] らはある限られたクラスの命令型プログラムについては，任意のループ文の代数的不変条件の基底をなすグレブナー基底を求める問題は決定可能であることを示した．Cachera ら [2,3] は代数的不変条件の計算をグレブナー基底を用いずに行うアルゴリズムを提案した．彼らは未定係数を含むテンプレート多項式 p について，$p = 0$ がループ不変条件となるための条件を多項式に対する演算でエンコードし，未定係数の間に成り立つべき制約を求め，それを解くことで代数的不変条件を求める．小島ら [9] はテンプレート多項式に含まれる必要のない単項式を次元型 [8] を用いたプログラム解析で求め，テンプレート多項式のサイズを小さくすることで，彼らの手法を高速化する方法を提案した．データ駆動アプローチを用いた手法は，多項式代数を用いた手法に比べて，多くの場合軽量であり，プログラムのソースコードが必ずしも無くても適用が可能であるという利点がある．

6 結論

本論文ではデータ駆動アプローチを用いてプログラムの線形写像表現を求めるアルゴリズムを提案した．提案した手法では，プログラムの初期状態を順次生成し，生成された状態から始めてプログラムを実行して終了状態を計算し，初期状態と終了状態の間の関係から線形写像表現の候補を求める．求められた候補が正しい線形写像表現になっているかどうかをプログラム検証器で検証し，もし正しくないという結果が得られた場合は，得られた反例を新たに初期状態として再び線形写像表現を求める．以上のループを繰り返すことで，停止するプログラムについては，線形写像表現が存在するならば有限時間でそれが求まることを証明した．

本アルゴリズムが正しく動作することを小さな例で確認しているが，より大きなプログラムを用いた評価はまだ行っていない．アルゴリズムを実装しベンチマークで評価することは重要な課題である．

提案手法が使う基底は，現在人手で与えることを仮定しているが，もし与えられた基底が線形写像表現を構成するために不十分であった場合は，提案するアルゴリズムは失敗する．アルゴリズム 1 の 7 行目に到達した場合がこのケースに相当するが，この場合にこれまでに生成されたテストの集合を解析することで，基底として足りない単項式を生成することができる場合がある．Sharma ら [11] の基底の自動生成アルゴリズムは，実際にこのようにして新しい基底を学習しており，提案手法においても同様のアプローチが適用できる可能性がある．

謝辞

小島は JST CREST の助成を，末永は JST さきがけ (No. JPMJPR15E5) と科研費 (15KT0012) の助成を受けて本研究を行った．

参 考 文 献

[1] Beyer, D. and Keremoglu, M. E.: CPAchecker: A Tool for Configurable Software Verification, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, 2011, pp. 184–190.

[2] Cachera, D., Jensen, T., Jobin, A., and Kirchner, F.: Fast inference of polynomial invariants for imperative programs, Technical Report RR-7627, INRIA, 2011.
[3] Cachera, D., Jensen, T. P., Jobin, A., and Kirchner, F.: Inference of polynomial invariants for imperative programs: A farewell to Gröbner bases, *Sci. Comput. Program.*, Vol. 93(2014), pp. 89–109.
[4] Calcagno, C., Distefano, D., Dubreil, J., Gabi, D., Hooimeijer, P., Luca, M., O'Hearn, P. W., Papakonstantinou, I., Purbrick, J., and Rodriguez, D.: Moving Fast with Software Verification, *NASA Formal Methods - 7th International Symposium, NFM 2015, Pasadena, CA, USA, April 27-29, 2015, Proceedings*, 2015, pp. 3–11.
[5] Cousot, P. and Cousot, R.: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints, *POPL 1977*, 1977, pp. 238–252.
[6] de Moura, L. M. and Bjørner, N.: Z3: An Efficient SMT Solver, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, 2008, pp. 337–340.
[7] de Oliveira, S., Bensalem, S., and Prevosto, V.: Polynomial Invariants by Linear Algebra, *Automated Technology for Verification and Analysis - 14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings*, 2016, pp. 479–494.
[8] Kennedy, A.: Dimension Types, *Programming Languages and Systems - ESOP'94, 5th European Symposium on Programming, Edinburgh, U.K., April 11-13, 1994, Proceedings*, 1994, pp. 348–362.
[9] Kojima, K., Kinoshita, M., and Suenaga, K.: Generalized Homogeneous Polynomials for Efficient Template-Based Nonlinear Invariant Synthesis, *Static Analysis - 23rd International Symposium, SAS 2016, Edinburgh, UK, September 8-10, 2016, Proceedings*, 2016, pp. 278–299.
[10] Müller-Olm, M. and Seidl, H.: Computing polynomial program invariants, *Inf. Process. Lett.*, Vol. 91, No. 5(2004), pp. 233–244.
[11] Padhi, S., Sharma, R., and Millstein, T. D.: Data-driven precondition inference with learned features, *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016*, 2016, pp. 42–56.
[12] Rodríguez-Carbonell, E. and Kapur, D.: Generating all polynomial invariants in simple loops, *J. Symb. Comput.*, Vol. 42, No. 4(2007), pp. 443–476.
[13] Sankaranarayanan, S., Sipma, H., and Manna, Z.: Non-linear loop invariant generation using Gröbner bases, *POPL 2004*, 2004, pp. 318–329.
[14] Sharma, R., Gupta, S., Hariharan, B., Aiken, A., Liang, P., and Nori, A. V.: A Data Driven Approach for Algebraic Loop Invariants, *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, 2013, pp. 574–592.
[15] Winskel, G.: *The Formal Semantics of Programming Languages: An Introduction*, MIT Press, Cambridge, MA, USA, 1993.

A 補題の証明

(証明) (補題 4.1)

まず $a \in A$ について $A - a$ が線形空間になるとき，任意の $b \in A$ について $A - b$ が線形空間になることを示す，任意に $a', b' \in A - b$ を選ぶ．

- $a' + b' \in A - b$ であること: $a' + b \in A$ かつ $b' + b \in A$ なので $a' + b - a, b' + b - a \in A - a$. $b - a \in A - a$ であることと $A - a$ が線形空間であることより，$a' + b' + b - a = (a' + b - a) + (b' + b - a) - (b - a) \in A - a$. よって，$a' + b' \in A - b$.
- 任意の $k \in \mathbb{R}$ について $ka' \in A - b$ であること: $a' + b - a, b - a \in A - a$ であることと $A - a$ が線形空間であることより，$ka' + b - a = k(a' + b - a) - (k - 1)(b - a) \in A - a$. よって $ka' \in A - b$.

よって，$A - b$ は線形空間である．

次に，任意の $a, b \in A$ について，$A - a = A - b$ となることを示す．任意の $a' \in A - b$ について，$a' + b - a \in A - a$ となり，$b - a \in A - a$ であることと $A - a$ が線形空間であることより，$a' \in A - a$. よって，$A - b \subseteq A - a$. 同様にして $A - a \subseteq A - b$ が示せるので，$A - a = A - b$.

(証明) (補題 4.2) $A'_{u,v}$ が空でないので $T' \in A'_{u,v}$ を取れる．定義より $A'_{u,v} - T' = \{T - T' | u = Tv \wedge T \in A\}$ であるが，$u = T'v$ であるから，この集合は $\mathcal{A} := \{T - T' | (T - T')v = 0 \wedge T \in A\} = v^{\perp} \cap (A - T')$ に等しい．よって $\mathcal{A}$ は線形空間であり，$A'_{u,v}$ は affine subspace である．

(証明) (補題 4.3) $B \subseteq A$ のとき，$b \in B$ について

$B-b \subseteq A-b$ であるが，$B-b$ と $A-b$ は線形空間なので，$B-b$ は $A-b$ の線形部分空間である．よって $\dim(B-b) \leq \dim(A-b)$ であるから $\dim B \leq \dim A$ となる．$\dim A = \dim B$ のときは，$\dim(A-b) = \dim(B-b)$ となるが，$B-b$ は $A-b$ の線形部分空間だから $A-b = B-b$ である．よって $A = B$.

(証明) (補題 4.4)

$\mathcal{T}' \subseteq \mathcal{T}$ であることは明らか．ここで，$\mathcal{T} = \mathcal{T}'$ と仮定すると，任意の $T \in \mathcal{T}$ で $[\![\mathcal{B}]\!]_{[\![c]\!]_\sigma} = T[\![\mathcal{B}]\!]_\sigma$ となるため，$[\![\mathcal{B}]\!]_{[\![c]\!]_\sigma} = T'[\![\mathcal{B}]\!]_\sigma$ となる．よって，$\{\mathcal{B} = [\![\mathcal{B}]\!]_{[\![c]\!]_\sigma}\}c\{\mathcal{B} = T'[\![\mathcal{B}]\!]_\sigma\}$ となるが，これは σ の定義に矛盾する．以上より，$\mathcal{T} \neq \mathcal{T}'$ となるため，$\mathcal{T}' \subsetneq \mathcal{T}$.

(証明) (補題 4.5) n に関する数学的帰納法で証明する．$\mathcal{S} \subseteq \mathcal{T}'_0$ は明らかである．$\mathcal{S} \subseteq \mathcal{T}'_k$ を仮定する．T' を 10 行目で選ばれた $\mathcal{T}'_k$ の元とし，σ を 14 行目で選ばれた $\varphi[\sigma/\vec{x}] \wedge \not\models \{\mathcal{B} = [\![\mathcal{B}]\!]_\sigma\}c\{\mathcal{B} = T'[\![\mathcal{B}]\!]_\sigma\}$ を満たす状態，すなわち $\sigma \models \varphi$ かつ $\not\models \{\mathcal{B} = [\![\mathcal{B}]\!]_\sigma\}c\{\mathcal{B} = T'[\![\mathcal{B}]\!]_\sigma\}$ なる状態とする．アルゴリズムの定義から，$\mathcal{T}'_{k+1} = \mathcal{T}'_k \cap \{T \mid [\![\mathcal{B}]\!]_{[\![c]\!]_\sigma} = T[\![\mathcal{B}]\!]_\sigma\}$．帰納法の仮定 $\mathcal{S} \subseteq \mathcal{T}'_k$ より，$\mathcal{S} \subseteq \mathcal{T}'_{k+1}$ を示すためには，$\mathcal{S} \subseteq \{T \mid [\![\mathcal{B}]\!]_{[\![c]\!]_\sigma} = T[\![\mathcal{B}]\!]_\sigma\}$ を示せば十分である．任意に $S \in \mathcal{S}$ を選ぶ．$\mathcal{S}$ の定義より，$\forall \vec{a}.\varphi[\vec{a}/\vec{x}] \implies \models \{\mathcal{B} = [\![\mathcal{B}]\!]_{\vec{a}}\}c\{\mathcal{B} = S[\![\mathcal{B}]\!]_{\vec{a}}\}$．よって，$\sigma \models \varphi$ ならば，$\models \{\mathcal{B} = [\![\mathcal{B}]\!]_\sigma\}c\{\mathcal{B} = S[\![\mathcal{B}]\!]_\sigma\}$．仮定より実際に $\sigma \models \varphi$ なので，$\models \{\mathcal{B} = [\![\mathcal{B}]\!]_\sigma\}c\{\mathcal{B} = S[\![\mathcal{B}]\!]_\sigma\}$．よって $\models \{A\}c\{B\}$ の定義と $\sigma \models \mathcal{B} = [\![\mathcal{B}]\!]_\sigma$ が成り立つことから $[\![c]\!]_\sigma \models \mathcal{B} = S[\![\mathcal{B}]\!]_\sigma$．したがって，$[\![\mathcal{B}]\!]_{[\![c]\!]_\sigma} = S[\![\mathcal{B}]\!]_\sigma$.