論　文

スタック検査機能をもつプログラムに対するセキュリティ検証問題の決定可能性

新田　直也[†]　　高田　喜朗[†]　　関　浩之[†]

Decidability of the Security Verification Problem for Programs with Stack Inspection

Naoya NITTA[†], Yoshiaki TAKATA[†], and Hiroyuki SEKI[†]

あらまし　Java development kit 1.2 のように，プログラム実行時に制御スタックを検査することでアクセス制御を行うようなプログラム環境がある．Jensen らは，プログラム P 及び時相論理式を用いて記述された検証条件 ψ を与えたときに，P の到達可能な状態すべてが ψ を満たすかどうかを決定する問題として検証問題を定義し，相互再帰を含まないプログラムのクラスに対して検証問題が決定可能となることを示した．本論文では，時相論理式よりも真に表現能力の大きい正規言語を用いて検証問題を定義する．そして，プログラムのトレース集合がインデックス言語となることを示し，その系としてプログラムが相互再帰を含む場合も含めて検証問題が一般に決定可能となることを示すことにより，Jensen らの結果を改善する．更に，自明なスタック検査のみを含むプログラムのクラスに対する検証問題が，プログラムサイズに対して多項式時間可解であることを示す．

キーワード　アクセス制御，セキュリティ検証，スタック検査，Java

1. まえがき

計算機ネットワークの急速な発展に伴い，ネットワークアプリケーションにおけるセキュリティ保全の重要性が高まっている．例えば電子商取引アプリケーションにおいては，不特定多数の提供者からのモバイルプロセスが，ユーザの計算機上で実行される．そのため，ユーザのリソースを，悪意あるモバイルプロセスのアクセスから保護するための適切なセキュリティ保全機構が必要となる．Java^{TM} の sandbox モデルはそのようなセキュリティ保全機構の一つであるが，リモートコードがリソースへアクセスすることを強く制限しているため，可用性に欠けることが指摘されている．

これらの問題を解決するため，Java development kit 1.2（JDK1.2）[7] では，あるプロセスが特定のリソースへのアクセス権をもつかどうかを実行時に検査するための，動的なアクセス制御機能を提供している．この機能は，checkPermission と呼ばれるアクセス権を検査するメソッドで実現されている．他のプログラミング言語と同様に，Java の実行環境は，呼び出されたメソッドに関する情報を格納するブロック（フレーム）の列からなる実行時制御スタック（以降単にスタックと呼ぶ）をもつ．あるメソッド m のフレームには，m の用いる実引数，局所変数，呼出し番地とともに，m に割り当てられたアクセス権が記憶されている．$Perm$ をあるアクセス権とする．Java 実行環境は，checkPermission（$Perm$）が呼び出されたとき現在のスタックの内容を，実行中のメソッド m から祖先メソッドに向かって順番に参照する．そして，スタックの底または特権が付与されたメソッドに出会うまでのすべてのメソッドがアクセス権 $Perm$ をもつかどうかを検査する．それらすべてのメソッドが $Perm$ をもつならば実行は継続され，そうでなければ実行は異常終了する．

プログラム全体が満たすべき性質（例えば「制御がローカルリソースを更新するメソッドに到達するのは，$P_{customer}$ または P_{write} のいずれかのアクセス権をもつメソッドのみを通過してきた場合に限る」）を検証条件と呼ぶ．検証条件が成り立つように適切なセキュ

† 奈良先端科学技術大学院大学情報科学研究科，生駒市
Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma-shi, 630–0101 Japan

リティ保全を行うには，保護したいローカルリソースにアクセスするローカルメソッドに，適切なチェック文（checkPermission）を慎重に配置する必要がある．しかし，プログラム全体が巨大かつ複雑になった場合，プログラムが検証条件を満たすか否かを手作業で検証することは困難である．そこで，与えられたプログラム P 及び検証条件 ψ に対して，P のあらゆる実行が ψ を満たすかどうかを自動検証できることが望ましい．

[11] では，スタック検査機能をもつプログラムに対する検証問題を以下のように定式化している．

- プログラムを有向グラフで表現する．ただし，プログラムが扱うデータ内容は捨象し，スタックの内容を，各メソッド呼出しが発生した実行番地に対応する頂点の有限系列で表現する．これをプログラムの状態と呼ぶ．プログラムの実行系列（トレースとも呼ぶ）は，状態の有限系列 $s_0, s_1, \ldots, s_k$（ただし，s_0 は初期状態，s_{i+1} $(0 \leq i < k)$ は s_i から 1 ステップの実行によって到達可能な状態）で表現される．
- スタック検査のためのチェック文を，$check(\phi)$ という形式で表現し，チェック文において検査する性質 ϕ を，線形時相論理 (LTL) 式 [3] を用いて記述する．状態 s における $check(\phi)$ の実行は，クリプケ構造の上で s が ϕ を満たすときかつそのときのみ続行される．検証条件も同様に，LTL 式を用いて記述する．
- プログラム P 及び検証条件 ψ が与えられたとき，P の任意のトレースに含まれる任意の状態において ψ が成り立つかどうかを決定する問題を「P と ψ に対する検証問題」と呼ぶ．

[11] では，以上の定式化のもとで LTL 式に対するモデル検査法 [3] を用い，次の結果を得ている：

「P をプログラム，ψ を検証条件とする．P が相互再帰を含まないならば，P と ψ に対する検証問題は決定可能である．」

本論文でも，[11] と同様に有向グラフを用いてプログラムをモデル化する．ただし，チェック文において検査する性質及び検証条件として，LTL 式の代わりに，頂点集合をアルファベットとする任意の正規言語を指定する．正規言語のクラスは LTL 式によって記述可能な言語クラスを真に包含することが知られているので [6], 本論文のモデルは [11] におけるモデルの拡張となっている．そして，任意のプログラム（相互再帰も許す）に対して検証問題が決定可能であることを示す．これは，[11] の結果の改善となっている．検証問題の決定可能性の証明手順は以下のとおりである．

(i)　任意のプログラム P について，P のトレース集合がインデックス言語 [2]（文脈自由言語の一つの拡張）となることを示す．

(ii)　インデックス言語のクラスが正規言語との集合積演算に関して閉じていること，及び，インデックス言語における空集合問題が決定可能であること [2] より，検証問題も決定可能となる．

また本論文では「自明な」チェック文のみ含むプログラムのクラスに対する検証問題の計算複雑さについても述べる．

本論文の構成は以下のとおりである．まず，**2.** で [11] に基づいて，プログラムモデルを定義する．**3.** で検証問題を形式言語を用いて定義し，検証問題の具体例を示す．**4.** では，プログラムのトレース集合がインデックス言語となることを示し，検証問題が決定可能であることを示す．また **5.** では，プログラムが「自明な」チェック文のみを含む場合，検証問題がプログラムサイズに対して多項式時間可解であることを示す．最後に **6.** で，まとめを述べる．

［関連研究］　[16] では，信念論理の一種である ABLP 論理 [1] の部分クラスを用いて，スタック検査の概念を特定のプログラミング言語に依存しない形で定式化し，与えられたスタック検査の十分条件の判定アルゴリズムを与えている．しかし，[11] や本論文で扱っている，スタック検査を含むプログラムの検証法については，[16] では触れられていない．

以下では，プログラムセキュリティの検証に関する他の関連研究について述べる．[4], [5] は，セキュリティクラスを束でモデル化し，それに基づいてプログラムの情報フローの静的解析法を開拓した．近年，Denning らの手法の形式化，一般化がいくつか提案されている．例えば，[13] では抽象解釈を利用してモデル化を行っている．一方，型推論の手法を応用して，解析法のモデル化を行う手法も提案されている（例えば，[8], [15]）．これらの研究では，セキュリティクラスを型であると解釈し，与えられたプログラム P が型付け可能であるならば，P が望ましくない情報フローを起こさないこと（noninterference）が示されている．それに対し，[11] や本論文では，「局所的なセキュリティ検査を行うプログラム P と，プログラム全体の振舞いを表す条件 ψ が任意に与えられたとき，P が ψ を満たすか」という一般的な検証問題を扱っている．

2. プログラムモデル

2.1 フローグラフ

[11] に従い，プログラムをフローグラフと呼ばれる有向グラフでモデル化する．グラフの頂点は，プログラムの番地に対応する．フローグラフでは，JDK1.2 の checkPermission のように実行時にアクセス権検査を行うチェック文も一つの頂点とする．

フローグラフにおける有向辺は次の 2 種類である．一つ目は transfer edge（tg）と呼ばれ，同一メソッド内の制御の流れを表す．例えば，メソッド呼出し $m_1()$, $m_2()$ に対応する頂点を n_1, n_2 とし，n_1 から n_2 への tg（$n_1 \stackrel{TG}{\rightarrow} n_2$ と表す）が存在するならば，$m_1()$ の実行直後に $m_2()$ が実行可能であることを表す．二つ目は call edge（cg）と呼ばれ，呼出し元から呼出し先へ向かう有向辺である．例えば，頂点 n_1 がメソッド呼出し $m_1()$ を行う番地，n_2 がメソッド m_1 の先頭番地を表すならば，n_1 から n_2 への cg（$n_1 \stackrel{CG}{\rightarrow} n_2$ と表す）がひかれ，n_1 における m_1 の呼出しによって，n_2 に制御が移ることを表す．m_1 の実行が終了したら，$n_1 \stackrel{TG}{\rightarrow} n_3$ を満たす n_3 に制御を移す．

形式的には，プログラム P は以下のような 5 項組で表される有向グラフである．

$P = (NO, IS, IT, TG, CG)$, ここで,

$$\begin{array}{lcl} IS & : & NO \rightarrow \{call, return, check(L_\phi)\} \\ IT & \in & NO \\ TG & \subseteq & NO \times NO \\ CG & \subseteq & NO \times NO \end{array}$$

NO は頂点の有限集合，IT はプログラムの実行開始頂点，TG, CG はそれぞれ transfer edge, call edge の有限集合を表す．

頂点集合 NO は，IS によって次の三つの部分集合に分割される．以下，$n \in NO$ とする．

- $IS(n) = call$: メソッド呼出し．
- $IS(n) = return$: メソッド呼出しからの復帰．
- $IS(n) = check(L_\phi)$: チェック文．現在の状態が言語 L_ϕ で表現される性質を満たしている（L_ϕ に属している）ときに限り実行を継続する．L_ϕ については **2.4** で述べる．

if 文等の条件分岐文，while 文などの繰返し文は，非決定的実行に置き換える．例えば，文の系列 $m_1()$; **if** ... **then** $m_2()$ **else** $m_3()$ を考える．フローグラフにおける $m_1()$, $m_2()$, $m_3()$ の頂点をそれぞれ n_1, n_2, n_3 とすると，n_1 から n_2, n_3 の両方に tg をひく．オブジェクト指向プログラムからフローグラフへの変換法については, [14] などを参照されたい．

2.2 プログラムの状態

一般に，空系列を ϵ で表し，記号集合 Σ 上の有限系列全体の集合を Σ^* と書く．$\Sigma^+ := \Sigma^* - \{\epsilon\}$ とする．プログラム $P = (NO, IS, IT, TG, CG)$ の状態をスタックの内容で定義する．**1.** で述べたように，スタックは呼出し番地を示す頂点の系列で表現される．スタックの先頭(注1)には常に現在実行中の頂点がプッシュされている．すなわち，メソッド呼出しの際は，呼出し先の頂点がスタックにプッシュされる．スタックの先頭の頂点が n_1 であり，$n_1 \stackrel{TG}{\rightarrow} n_2$ であるならば，n_1 の実行後，スタックの先頭を n_1 から n_2 に書き換える．連接演算を記号 : で表す．頂点 n のみからなる系列を $\langle n \rangle$ と表す．文脈から明らかな場合，$\langle n \rangle$ を n と略記する．例えばスタック $n_1 : n_2 : n_3$ は次のことを意味している.「まず，n_1 から n_2 を含むメソッドを呼び出し，そのメソッドの実行中に n_2 に到達した．更に n_2 から n_3 を含むメソッドを呼び出し，現在の実行番地が n_3 である.」スタックの先頭が n であり，$IS(n) = return$ ならば，n をスタックからポップすることによりメソッドからの復帰を行う．また，プログラムの初期状態を，実行開始頂点のみからなるスタック $\langle IT \rangle$ で表す．

2.3 プログラムのトレース

状態間の遷移関係 $\triangleright$ を以下のように定義する．$s_1 \triangleright s_2$ は，プログラムの 1 ステップの実行によって状態が s_1 から s_2 に遷移可能であることを表す．

［定義 1］ [遷移関係]　プログラム $P = (NO, IS, IT, TG, CG)$ の遷移関係 $\triangleright$ は次の三つの規則を満たす最小の関係である．ここで，s は状態 ($\in NO^*$), n, m, n_i, n_j は頂点 ($\in NO$) を表す．

$$\frac{IS(n) = call,\ n \stackrel{CG}{\rightarrow} m}{s : n \ \triangleright \ s : n : m} \quad (1)$$

$$\frac{IS(m) = return,\ n_i \stackrel{TG}{\rightarrow} n_j}{s : n_i : m \ \triangleright \ s : n_j} \quad (2)$$

(注1)：本論文の定義では，スタックを表す頂点系列において「スタックの先頭」は系列の末尾となるので注意.

$$\frac{IS(n) = check(L_\phi),\ s : n \in L_\phi,\ n \overset{TG}{\rightarrow} n_j}{s : n \triangleright s : n_j}. \quad (3)$$

□

以降，頂点系列の連接記号 : はあいまいさのない限り省略する．関係 $\triangleright$ を満たす状態を任意に連接した状態系列（頂点の系列の系列）を遷移系列と呼ぶ．特に，初期状態 $\langle IT \rangle$ から始まる遷移系列を P のトレースと呼ぶ．頂点の連接 : と区別するため，遷移関係を表す記号 $\triangleright$ を流用して，状態間の連接も $\triangleright$ で表す．トレース集合を次のように定義する．

［定義 2］［トレース集合］ プログラム $P = (NO, IS, IT, TG, CG)$ のトレース集合 $[\![P]\!]$ は，

$$[\![P]\!] = \{s_1 \triangleright \cdots \triangleright s_k \mid s_1 = \langle IT \rangle, \\ s_1, \ldots, s_k \in NO^*, \forall i < k, s_i \triangleright s_{i+1}\}. \quad \square$$

2.4 チェック文において検査する性質

プログラム中のチェック文 $check(L_\phi)$ において検査する性質を表す言語 L_ϕ を，頂点集合 NO 上の正規言語で与える（したがって，$L_\phi \subseteq NO^*$）．規則 (3) より，プログラムの制御が $check(L_\phi)$ に到達したとき，実行は現在の状態 $s : n(\in NO^*)$ が L_ϕ に属するときかつそのときのみ続行される．なお，具体的に L_ϕ をどのような記述法（正規表現，有限オートマトン）で表現するかは自由である．JDK1.2 におけるスタック検査をチェック文として記述した例を次に示す．

［例 1］（JDK1.2 におけるスタック検査） $Perm$ を一つのアクセス権とする．メソッド checkPermission（$Perm$）の呼出しによる検査に通るための条件は，

- スタック内のすべてのフレームがアクセス権 $Perm$ をもつ，または，
- あるフレーム f が特権（privilege）を付与されており，f より新しいフレーム（f を含む）がすべて $Perm$ をもつこと

である．$Priv$ を特権の与えられている頂点集合とし，ϕ をアクセス権 $Perm$ をもつ頂点集合とすると，checkPermission（$Perm$）は，フローグラフにおけるチェック文 $check(JDK(\phi))$ によって表される．ここで，$JDK(\phi)$ は頂点集合 NO をアルファベットとする次の正規表現で表される正規言語である．

$$(NO^*(Priv \cap \phi) \cup \epsilon)\phi^*. \quad (4)$$

ただし，アルファベットの部分集合である $NO, \phi, Priv \cap \phi$ はそれらの集合の要素を $\cup$ で結合した正規表現を略記したものである． □

3. 検証問題

チェック文を含むプログラムに対して，プログラム全体のセキュリティが保たれるかを調べる検証問題を定義する．これは [11] における検証問題の一般化となっている．また，検証問題の具体例を示す．

3.1 検証問題の定義

プログラム全体が満たすべき性質を表す検証条件は，チェック文と同様，正規言語によって与える．

［例 2］（クリティカルセクション）[11] クリティカルセクションに属する頂点集合を *Critical*, 管理者権限をもつ頂点集合を *Manager* とする．

$$(\overline{Critical})^* Manager\ NO^*$$

は，管理者権限を得るまでクリティカルセクションに入らないことを表す検証条件である． □

すべての状態が検証条件 L_ψ を満たすような状態系列集合 $L_{\mathrm{safe}}[\psi]$ は，以下のように表すことができる．

$$L_{\mathrm{safe}}[\psi] := (L_\psi \triangleright)^* L_\psi.$$

プログラム P のトレース集合 $[\![P]\!]$ に属するすべてのトレース中のすべての状態が検証条件を満たすときかつそのときのみ $[\![P]\!]$ が $L_{\mathrm{safe}}[\psi]$ に含まれることから，検証問題を次のように形式的に定義する．

［定義 3］（検証問題）

入力： プログラム P と検証条件 L_ψ.

問題： $[\![P]\!] \subseteq L_{\mathrm{safe}}[\psi]$ かどうかを判定せよ． □

3.2 検証例

電子商取引システムにおける検証問題の例を示す．この例ではプログラムに相互再帰が含まれているため，[11] の検証法によって直接解くことはできないが，**4.** で述べる方法では決定することができる．

顧客に出入金を行う手続きを提供する銀行のオンラインシステムを考える．*System, Provider, Client, Unknown* と呼ばれる四つの保護ドメイン（それぞれ，システム，サービス提供者，顧客，未知ユーザを表す）があるとする．サービス提供者はシステム管理者に信頼されており，read 及び write メソッドを直接呼び出すことが許され，また特権が与えられている．図 1 はこのシステムのプログラムモデルを表す．

実線の矢印は cg を，破線の矢印は tg を表す．丸で囲まれた頂点は特権を付与された頂点を表す．$NO = \{n_i \mid 1 \leq i \leq 25\}$ とする．各頂点はそれぞれ一つの保護ドメインに所属し，各保護ドメインに

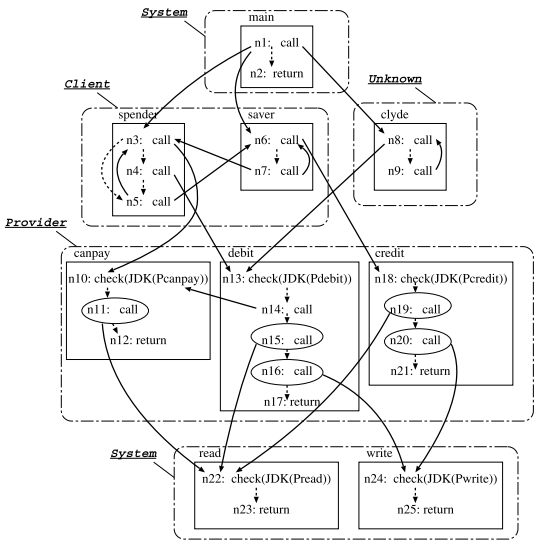


図 1　オンラインシステムのプログラムモデル
Fig. 1　The program model of the online banking system.

表 1　保護ドメインとその所属頂点
Table 1　Nodes and their protection domains.

保護ドメイン	所属頂点
D_{Client}	$n_3,\ n_4,\ n_5,\ n_6,\ n_7$
$D_{Provider}$	$n_{10},\ n_{11}, \ldots, n_{21}$
D_{System}	$n_1,\ n_2,\ n_{22},\ n_{23},\ n_{24},\ n_{25}$
$D_{Unknown}$	$n_8,\ n_9$

表 2　保護ドメインの所有アクセス権
Table 2　Protection domains and their permissions.

保護ドメイン	アクセス権
D_{Client}	$P_{canpay},\ P_{debit},\ P_{credit}$
$D_{Provider}$	$P_{canpay},\ P_{debit},\ P_{credit},\ P_{read},\ P_{write}$
D_{System}	$P_{canpay},\ P_{debit},\ P_{credit},\ P_{read},\ P_{write}$
$D_{Unknown}$	—

対しメソッドへのアクセス権(注2)が与えられる．本例における保護ドメイン及びそれに与えられたアクセス権の関係を表 1，表 2 に示す．なお，D_{dom} は保護ドメイン dom を，P_{meth} はメソッド $meth$ へのアクセス権を表す．頂点 n_{13} に記述された性質 $JDK(P_{debit})$ は，(4) より，

$$JDK(P_{debit}) = (NO^*\ (Priv \cap P_{debit}) \cup \epsilon)(P_{debit})^*$$

となる．頂点 n_{10}，n_{18}，n_{22}，n_{24} に記述された $JDK(P_{canpay})$，$JDK(P_{credit})$，$JDK(P_{read})$，$JDK(P_{write})$ も同様な正規表現で表すことができる．

次の二つの状態系列を考える．

$$\begin{aligned}\alpha_1 = {} & n_1 \triangleright n_1 n_3 \triangleright \underline{n_1 n_3 n_{10}} \triangleright n_1 n_3 n_{11} \triangleright \underline{n_1 n_3 n_{11} n_{22}} \\ & \triangleright n_1 n_3 n_{11} n_{23} \triangleright n_1 n_3 n_{12} \triangleright n_1 n_4 \triangleright \underline{n_1 n_4 n_{13}} \\ & \triangleright n_1 n_4 n_{14} \triangleright \underline{n_1 n_4 n_{14} n_{10}} \triangleright n_1 n_4 n_{14} n_{11} \\ & \triangleright \underline{n_1 n_4 n_{14} n_{11} n_{22}} \triangleright n_1 n_4 n_{14} n_{11} n_{23} \\ & \triangleright n_1 n_4 n_{14} n_{12} \triangleright n_1 n_4 n_{15} \triangleright \underline{n_1 n_4 n_{15} n_{22}} \\ & \triangleright n_1 n_4 n_{15} n_{23} \triangleright n_1 n_4 n_{16} \triangleright \underline{n_1 n_4 n_{16} n_{24}} \\ & \triangleright n_1 n_4 n_{16} n_{25} \triangleright n_1 n_4 n_{17} \triangleright n_1 n_5,\end{aligned}$$

$$\alpha_2 = n_1 \triangleright n_1 n_8 \triangleright \underline{n_1 n_8 n_{13}} \triangleright n_1 n_8 n_{14}.$$

まず，状態系列 α_1 の中で，チェック文に到達したときの状態は下線を引いた 7 箇所である．これらの状態はすべて，検査する性質を表す言語に属していることがわかる．また，系列中の隣接する 2 状態はすべて遷移関係を満たしているので，$\alpha_1 \in [\![P]\!]$ である．一方，α_2 では，$n_1 n_8 n_{13} \notin JDK(P_{debit})$ であるので，$\alpha_2 \notin [\![P]\!]$ である．今，検証条件を L_ψ を，

$$L_\psi = ((\overline{E_{write}})^* \cup (P_{debit})^* E_{write} (\overline{E_{write}})^*) \cap ((\overline{E_{read}})^* \cup (P_{canpay})^* E_{read} (\overline{E_{read}})^*)$$

とする．E_{meth} はメソッド $meth$ に属する頂点を表す．$L_{\rm safe}[\psi] = (L_\psi \triangleright)^* L_\psi$ は「P_{debit} (P_{canpay}) をもつメソッドのみを通ったときに限り，$E_{write}(E_{read})$ に到達することができる」を表す．$[\![P]\!] \subseteq L_{\rm safe}[\psi]$ が満たされるので，プログラム P は L_ψ を満たす．

4. 検証問題の決定可能性

ここでは，**2.3** で定義したプログラムのトレース集合を形式文法で表現する方法を考える．

［補題 1］ トレース集合 $[\![P]\!]$ が文脈自由言語（cfl）にならないようなプログラム P が存在する．

（証明） $P = (NO, IS, IT, TG, CG)$ を次のようなプログラムとする．

- $NO = \{n\}$. $IT = n$.
- transfer edge は存在しない．
- call edge は $n \overset{CG}{\rightarrow} n$ のみである．

このプログラムのトレース集合 $[\![P]\!]$ は

$$[\![P]\!] = \{n,\ n \triangleright nn,\ n \triangleright nn \triangleright nnn, \cdots\}.$$

ここで，$h(n) = n$, $h(\triangleright) = \epsilon$ なる準同型 h に対

(注2)：Java においては，ファイル等のリソースへのアクセスは必ずそのリソースがもつメソッドを介して行われるので，リソースへのアクセス権はメソッドへのアクセス権と考えて支障ない．

して，

$$h([\![P]\!]) = \{n,\ nnn,\ nnnnnn,\ nnnnnnnnnn, \cdots\} = \{n^{i(i+1)/2} \mid i \geq 1\}$$

となる．この $h([\![P]\!])$ が cfl でないことは，cfl に対する反復補題 [9] を用いて容易に証明できる．cfl のクラスは準同型について閉じている [9] ので，$[\![P]\!]$ も cfl ではない． □

上の補題より，任意のプログラムのトレース集合を生成するには，文脈自由文法（cfg）より生成能力が大きい文法クラスが必要である．cfg より生成能力の大きい文法クラスの一つとして，インデックス文法（ig）[2] が知られている．ここでは，ig を用い，以下の方針に従ってトレース集合を表現することを考える．

(a) ある状態系列 α が，その中のすべてのチェック文で検査が成功すると仮定するとトレースとなるとき，α を非検査トレースと呼ぶ．厳密には，(3) を，

$$\frac{IS(n) = check(L_\phi),\ n \overset{TG}{\rightarrow} n_j}{s : n \triangleright s : n_j}$$

に置き換えたとき，状態系列 α がプログラム P のトレースとなるならば，α を P の非検査トレースと呼ぶ．プログラム P が与えられたとき，P の非検査トレース全体からなる集合を生成する ig を構成する．

(b) 非検査トレース α 中のすべてのチェック文について，それを実行するときの状態が検査する性質を満たしているときかつそのときのみ，α はトレースとなる．そこで，任意の 2 状態が隣接し得るが，ただしチェック文においては検査する性質を満たしているような状態系列を表す言語を定義する．

以上の (a)，(b) により定義された二つの言語の共通集合として，プログラム P のトレース集合 $[\![P]\!]$ を表す．

4.1 インデックス文法

インデックス文法とは，5 項組 $G = (N, T, I, R, S)$ である [2]．各成分はそれぞれ次のとおりである．

(a) N は非終端記号の有限集合．
(b) T は終端記号の有限集合．
(c) I はインデックスと呼ばれる記号の有限集合．
(d) $S \in N$ は開始記号．
(e) R は次のいずれかの形の生成規則の有限集合．
$(1)A \to \alpha$, $(2)A \to Bf$, $(3)Af \to \alpha$.
ここで，$A, B \in N$, $f \in I$, $\alpha \in (N \cup T)^*$. ig の規則の適用は，インデックスの列を操作することを除けば，cfg の規則の適用とほとんど同じである．形式的には，$(NI^* \cup T)^*$ に属する文字列の集合上に関係 $\to_G$ を次のように定義する．以下，$\beta, \gamma \in (NI^* \cup T)^*$, $\xi \in I^*$, $X_i \in N \cup T$ とする．

1. $A \to X_1X_2 \cdots X_k \in R$ をタイプ (1) の生成規則とする．このとき，

$$\beta A \xi \gamma \to_G \beta X_1 \xi_1 X_2 \xi_2 \cdots X_k \xi_k \gamma$$

である．ただし，$X_i \in N$ のとき $\xi_i = \xi$ であり，$X_i \in T$ のとき $\xi_i = \epsilon$ である．

2. $A \to Bf \in R$ をタイプ (2) の生成規則とする．このとき，$\beta A \xi \gamma \to_G \beta B f \xi \gamma$.

3. $Af \to X_1X_2 \cdots X_k \in R$ をタイプ (3) の生成規則とする．このとき，

$$\beta A f \xi \gamma \to_G \beta X_1 \xi_1 X_2 \xi_2 \cdots X_k \xi_k \gamma.$$

ただし，$X_i \in N$ のとき $\xi_i = \xi$ であり，$X_i \in T$ のとき $\xi_i = \epsilon$ である．

1. では規則の右辺のすべての非終端記号にインデックス列が分配される．2. ではインデックス列の先頭にインデックスを追加する．3. ではインデックス列の先頭が削除され，残りのインデックス列が規則の右辺のすべての非終端記号に分配される．

$\to_G$ の反射推移閉包を $\overset{*}{\to}_G$ と表す．文脈から文法 G が明らかな場合は，関係 $\to_G$ を単に $\to$ と表す．ig $G = (N, T, I, R, S)$ によって生成される言語を $L(G) = \{w \in T^* \mid S \overset{*}{\to}_G w\}$ と定義する．

4.2 非検査トレース集合

以下，補題 4 までプログラム $P = (NO, IS, IT, TG, CG)$ を固定する．P の非検査トレースの集合を生成するインデックス文法 $G_{P,\mathrm{T}} = (N, T, I, R, S)$ を次のように定義する．

(a) $N = \{S, W, A, B, C\} \cup \{N_i, N_i', N_i'' \mid n_i \in NO\}$.
(b) $T = NO \cup \{\triangleright\}$.
(c) $I = \{ni \mid n_i \in NO\} \cup \{\$\}$.
(d) R は以下を含む．

$$S \to W\$ \tag{5}$$
$$W \to Ani_1 \quad \text{for } n_1 = IT \tag{6}$$
$$A \to B \triangleright C \tag{7}$$
$$A \to B \tag{8}$$
$$Bni \to Bn_i \quad \text{for } \forall n_i \in NO \tag{9}$$

$$B\$ \to \epsilon \tag{10}$$

$$C\dot{n_i} \to N_i \quad \text{for } \forall n_i \in NO \tag{11}$$

$$N_j'' \to A\dot{n_k} \quad \text{for } \forall n_j, n_k.\ n_j \overset{TG}{\to} n_k \tag{12}$$

また，頂点の種類によって次の規則も含む．

(1) $IS(n_i) = call$ のとき．

$$N_i \to N_i' \dot{n_i} \tag{13}$$

$$N_i' \to A\dot{n_j} \quad \text{for } \forall n_j.\ n_i \overset{CG}{\to} n_j \tag{14}$$

(2) $IS(n_i) = return$ のとき．

$$N_i \dot{n_j} \to N_j'' \quad \text{for } \forall n_j \in NO \tag{15}$$

(3) $IS(n_i) = check(L_\phi)$ のとき．

$$N_i \to A\dot{n_j} \quad \text{for } \forall n_j.\ n_i \overset{TG}{\to} n_j \tag{16}$$

なお，(16) の規則は，すべての $check(L_\phi)$ での検査結果が真となると仮定することに対応する．

4.3 チェック文における検査を満たす状態系列集合

s をプログラム P の任意の状態とする．s に対して，次の条件が満たされるときかつそのときのみ，実行を継続することができる：「現在の状態（スタック）s の先頭が n_i かつ $IS(n_i) = check(L_{\phi_i})$ ならば，$s \in L_{\phi_i}$ が成り立つ．」n_i を固定したとき，この性質を満たす言語 $L_{P,\mathrm{C}}^{(i)}$ は次の正規表現によって表すことができる．

$$NO^*(NO - \{n_i\}) \cup \epsilon \cup L_{\phi_i}.$$

したがって，P 中のチェック文の集合を $\{n_1, \cdots, n_k\}$ とするとき，すべてのチェック文における検査を満たす状態系列の集合 $L_{P,\mathrm{C}}$ は次のように表現できる．

$$L_{P,\mathrm{C}} := (X\triangleright)^* NO^*.$$

ここで $X := L_{P,\mathrm{C}}^{(1)} \cap L_{P,\mathrm{C}}^{(2)} \cap \ldots \cap L_{P,\mathrm{C}}^{(k)}$．ただしここでの $\cap$ は，言語間の積集合演算を表す．

4.4 トレース集合

4.2 で定義した言語 $L(G_{P,\mathrm{T}})$ と **4.3** で定義した言語 $L_{P,\mathrm{C}}$ に対し，$L(G_{P,\mathrm{T}}) \cap L_{P,\mathrm{C}}$ が P のトレース集合 $[\![P]\!]$ と一致することを示す．次の二つの補題は $G_{P,\mathrm{T}}$ の定義より容易に示せる．

［補題 2］ 任意の $\alpha \in L(G_{P,\mathrm{T}})$ は $\alpha = s_1 \triangleright \cdots \triangleright s_n$ ($s_i \in NO^*$ $(1 \le i \le n)$, $n \ge 1$) と書くことができ，かつ α に対して以下の形の導出が存在する．ただし，各行の右端の数字は適用される規則の番号である．また，$\delta_i \in (I - \{\$\})^+$ $(1 \le i \le n)$ である．

$$\begin{aligned}
S &\overset{*}{\to} A\delta_1\$ && (5),\ (6)\\
&\to B\delta_1\$ \triangleright C\delta_1\$ && (7)\\
&\overset{*}{\to} B\delta_1\$ \triangleright A\delta_2\$ && (11) - (16)\\
&\to B\delta_1\$ \triangleright B\delta_2\$ \triangleright C\delta_2\$ && (7)\\
&\overset{*}{\to} B\delta_1\$ \triangleright \cdots \triangleright B\delta_{n-1}\$ \triangleright A\delta_n\$ &&\\
&\to B\delta_1\$ \triangleright \cdots \triangleright B\delta_{n-1}\$ \triangleright B\delta_n\$ && (8)\\
&\overset{*}{\to} s_1 \triangleright \cdots \triangleright s_n. && (9),\ (10)
\end{aligned}$$

□

［補題 3］ 文法 $G_{P,\mathrm{T}}$ において，任意の $\delta \in (I - \{\$\})^*$, $s \in T^*$ について，

$B\delta\$ \overset{*}{\to} s$ であるときかつそのときのみ，$s \in NO^*$ かつ $\delta = \sigma(s)$ である．

ここで，$\sigma(n_{i_1} \cdots n_{i_n}) = \dot{n_{i_n}} \cdots \dot{n_{i_1}}$．□

［補題 4］ $s_1, \cdots, s_n \in NO^*$ とする．$s_1 \triangleright \cdots \triangleright s_n \in [\![P]\!]$ であるときかつそのときのみ，文法 $G_{P,\mathrm{T}}$ において $S \overset{*}{\to} B\delta_1\$ \triangleright \cdots \triangleright B\delta_{n-1}\$ \triangleright A\delta_n\$$ かつ $s_1 \triangleright \cdots \triangleright s_n \in L_{P,\mathrm{C}}$ である．ただし，$\delta_i = \sigma(s_i)$ $(1 \le i \le n)$.

（証明） $(\Rightarrow)$ n に関する帰納法で証明する（逆も補題 2 を用いて同様にして示せる）．$IT = n_1$ とする．

（基底段） $s_1 \in [\![P]\!]$ とする．定義から，$s_1 = \langle IT \rangle = n_1$ である．文法 $G_{P,\mathrm{T}}$ の規則 (5), (6) から，$S \to W\$ \to A\dot{n_1}\$$. また $\dot{n_1} = \sigma(n_1)$, $n_1 \in (X\triangleright)^* NO^* = L_{P,\mathrm{C}}$ も成り立つ．

（帰納段） 帰納法の仮定として，「$s_1 \triangleright \cdots \triangleright s_n \in [\![P]\!]$ であるならば，$S \overset{*}{\to} B\delta_1\$ \triangleright \cdots \triangleright B\delta_{n-1}\$ \triangleright A\delta_n\$$, $\delta_i = \sigma(s_i)(1 \le i \le n)$, かつ $s_1 \triangleright \cdots \triangleright s_n \in L_{P,\mathrm{C}}$」が成り立つとする．

$s_1 \triangleright \cdots \triangleright s_n \triangleright s_{n+1} \in [\![P]\!]$ かつ $s_n = s' : n_v$ ($s' \in NO^*$, $n_v \in NO$) とする．（$[\![P]\!]$ の定義より $s_n = \epsilon$ はあり得ない．）帰納法の仮定における記号列の末尾 $A\delta_n\$$ に対し規則 (7) より，

$$A\delta_n\$ \to B\delta_n\$ \triangleright C\delta_n\$. \tag{17}$$

- $IS(n_v) = call$ のとき．

$s_n \triangleright s_{n+1}$ であるから，$n_v \overset{CG}{\to} n_j$ である頂点 n_j が存在して，$s_{n+1} = s' : n_v : n_j$ である．式 (17) の $C\delta_n\$$ に規則 (11), (13), (14) が適用できて，$\delta_n = \sigma(s_n)$, $s_n = s' : n_v$ より，$C\delta_n\$ = C\dot{n_v}\sigma(s')\$ \to N_v\sigma(s')\$ \to N_v'\dot{n_v}\sigma(s')\$ \to A\dot{n_j}\dot{n_v}\sigma(s')\$ = A\sigma(s_{n+1})\$$. したがって，$S \overset{*}{\to} B\delta_1\$ \triangleright B\delta_2\$ \triangleright \cdots \triangleright B\delta_n\$ \triangleright A\delta_{n+1}\$$, $\delta_i = \sigma(s_i)$ $(1 \le i \le n+1)$ が成り立つ．

次に，$s_1 \triangleright s_2 \triangleright \cdots \triangleright s_n \triangleright s_{n+1} \in L_{P,\mathrm{C}}$ を示す．帰納法の仮定 $s_1 \triangleright s_2 \triangleright \cdots \triangleright s_n \in L_{P,\mathrm{C}}$ より，$s_i \in X$ $(1 \le i < n)$ だから，$s_n \in X$ かつ $s_{n+1} \in NO^*$ を示せばよい．後

者は明らか．$s_n = s' : n_v$ かつ $IS(n_v) = call$ だから（チェック文ではないから），$L_{P,\mathrm{C}}$ の定義より，$s_n \in X$.

- $IS(n_v) = check(L_{\phi_v})$ のとき，

$s_n \triangleright s_{n+1}$ より $s_n \in L_{\phi_v}$ かつ，$n_v \overset{TG}{\to} n_j$ である頂点 n_j が存在して $s_{n+1} = s' : n_j$ である．式 (17) の $C\delta_n\$$ に規則 (11)，(16) が適用できて，

$$C\delta_n\$ = Cn_v\sigma(s')\$ \to N_v\sigma(s')\$ \to An_j\sigma(s')\$ = A\sigma(s_{n+1})\$.$$

したがって，$S \overset{*}{\to} B\delta_1\$ \triangleright B\delta_2\$ \triangleright \cdots \triangleright B\delta_n\$ \triangleright A\delta_{n+1}\$$, $\delta_i = \sigma(s_i)$ $(1 \le i \le n+1)$ が成り立つ．

$IS(n_v) = call$ の場合と同様，$s_1 \triangleright s_2 \triangleright \cdots \triangleright s_n \triangleright s_{n+1} \in L_{P,\mathrm{C}}$ を示すためには，$s_n \in X$ を示せばよい．$s_n = s' : n_v$ かつ $IS(n_v) = check(L_{\phi_v})$ かつ $s_n \in L_{\phi_v}$ だから，$s_n \in L^{(v)}_{P,C}$．よって，$s_n \in X$.

- $IS(n_v) = return$ のときも同様． □

［定理 1］ 任意のプログラム P に対して，$[\![P]\!] = L(G_{P,\mathrm{T}}) \cap L_{P,\mathrm{C}}$.

（証明） $[\![P]\!] \subseteq L(G_{P,\mathrm{T}}) \cap L_{P,\mathrm{C}}$ を示す（逆も同様に示せる）．$s_1, \ldots, s_n \in NO^*$ とし，$s_1 \triangleright s_2 \triangleright \cdots \triangleright s_n \in [\![P]\!]$ と仮定する．補題 4 より，文法 $G_{P,\mathrm{T}}$ において $S \overset{*}{\to} B\delta_1\$ \triangleright \cdots \triangleright B\delta_{n-1}\$ \triangleright A\delta_n\$$, $\delta_i = \sigma(s_i)$ $(1 \le i \le n)$ である．このとき，上記の $A\delta_n\$$ に規則 (8) を適用すると，$B\delta_1\$ \triangleright \cdots \triangleright B\delta_{n-1}\$ \triangleright A\delta_n\$ \to B\delta_1\$ \triangleright \cdots \triangleright B\delta_{n-1}\$ \triangleright B\delta_n\$$. ここで $\delta_i = \sigma(s_i)$ $(1 \le i \le n)$ だから，各 $B\delta_i\$$ に対し補題 3 を用いると，$B\delta_1\$ \triangleright \cdots \triangleright B\delta_n\$ \overset{*}{\to} s_1 \triangleright \cdots \triangleright s_n$ となり，$s_1 \triangleright \cdots \triangleright s_n \in L(G_{P,\mathrm{T}})$ が成り立つ．補題 4 から $s_1 \triangleright \cdots \triangleright s_n \in L_{P,\mathrm{C}}$ も成り立つので，$s_1 \triangleright s_2 \triangleright \cdots \triangleright s_n \in L(G_{P,\mathrm{T}}) \cap L_{P,\mathrm{C}}$．以上より，$[\![P]\!] \subseteq L(G_{P,\mathrm{T}}) \cap L_{P,\mathrm{C}}$ が成り立つ． □

［補題 5］[2] インデックス言語のクラスは，正規言語との集合積について閉じている．また，インデックス言語の空集合問題は決定可能である． □

［定理 2］ プログラム P と検証条件 L_ψ が与えられたとき，検証問題 $[\![P]\!] \subseteq L_{\mathrm{safe}}[\psi]$ は決定可能である．

（証明） $[\![P]\!] \subseteq L_{\mathrm{safe}}[\psi] \Longleftrightarrow [\![P]\!] \cap \overline{L_{\mathrm{safe}}[\psi]} = \emptyset$ であることと，定理 1，補題 5 より明らか． □

5. 自明なチェック文のみを含む場合

補題 1 の証明からわかるように，プログラム P がチェック文を含まない場合でも，$[\![P]\!]$ は cfl になるとは限らない．

一方，P において初期状態から到達可能な状態全体の集合 $[P]$ を以下のように定義する．

$$[P] = \{s \in NO^* \mid \exists s_1 \triangleright s_2 \triangleright \cdots \triangleright s_i \in [\![P]\!],\ s = s_i\}.$$

プログラム P と検証条件 L_ψ に対する検証問題 $[\![P]\!] \subseteq (L_\psi \triangleright)^* L_\psi$ を判定することは，$[P] \subseteq L_\psi$ を判定することと等価である．

本章では，$P = (NO, IS, IT, TG, CG)$ を，チェック文として $check(NO^*)$（任意の状態に対し，検査結果が真となるチェック文）のみを含むプログラムとする．以下では $[P]$ が正規言語となることを示し，その系として，検証問題がプログラムサイズ（定理 4 の直前で定義）の多項式時間で解けることを示す．各頂点を呼び出したときに，その呼出しから復帰できる実行が存在するかどうかを表す述語 CR（can return）: $NO \to \{true,\ false\}$ を次の推論規則により定義する．

$$\frac{IS(n_1) = call,\ n_1 \overset{CG}{\to} n_2,\ n_1 \overset{TG}{\to} n_3,\ \mathrm{CR}(n_2) = \mathrm{CR}(n_3) = true}{\mathrm{CR}(n_1) = true} \tag{18}$$

$$\frac{IS(n) = return}{\mathrm{CR}(n) = true} \tag{19}$$

$$\frac{IS(n_1) = check(NO^*),\ n_1 \overset{TG}{\to} n_2,\ \mathrm{CR}(n_2) = true}{\mathrm{CR}(n_1) = true} \tag{20}$$

［補題 6］ $P = (NO, IS, IT, TG, CG)$ を，チェック文として $check(NO^*)$ のみを含むプログラムとする．P における任意の頂点 $n \in NO$ について，

$\mathrm{CR}(n) = true \Leftrightarrow IS(n') = return$
なるある頂点 n'とある遷移系列 $n \triangleright \cdots \triangleright n'$が存在．

ここで，$T = s_1 \triangleright \cdots \triangleright s_k$ が遷移系列であるとは，T が $s_i \triangleright s_{i+1} (1 \le i < k)$ を満たす状態系列であることをいう（**2.3**）．

（証明） ($\Rightarrow$) 推論規則 (18)〜(20) の適用回数に関する帰納法で証明できる．($\Leftarrow$) 遷移系列 $T_1 = s_1 \triangleright \cdots \triangleright s_l$ において，$s_1 = n$, $s_l = n'$, $IS(n') = return$ とし，l に関する帰納法で証明する．$l = 1$ のときは $IS(n) = return$ であることに注意．

- $IS(n) = return$ のとき．(19) より，$\mathrm{CR}(n) = true$.
- $IS(n) = check(NO^*)$ のとき．$n \overset{TG}{\to} n_2$ なる頂点 n_2 が存在し，$s_2 = n_2$ である．すなわち，$s_2 \triangleright \cdots \triangleright s_l$

は補題の条件を満たし T_1 より短い遷移系列であるから，l に関する帰納法の仮定より，$\mathrm{CR}(n_2) = \mathit{true}$. ゆえに，(20) より $\mathrm{CR}(n) = \mathit{true}$.

- $IS(n) = \mathit{call}$ のとき．遷移系列 T_1 は，$T_1 = n \triangleright nm \triangleright ns'_1 \triangleright \cdots \triangleright ns'_k \triangleright nm' \triangleright n_2 \triangleright \cdots \triangleright n'$, $s_1 = n, s'_i \in NO^+ (1 \leq i \leq k), s_l = n'$ と書けて，$IS(m') = \mathit{return}, n \overset{CG}{\rightarrow} m, n \overset{TG}{\rightarrow} n_2$. T_1 の部分系列 $nm \triangleright ns'_1 \triangleright \cdots \triangleright ns'_k \triangleright nm'$ における各状態の左端 n を取り除いた状態系列 $m \triangleright s'_1 \triangleright \cdots \triangleright s'_k \triangleright m'$ は補題の条件を満たし T_1 より短い遷移系列であるから，帰納法の仮定より，$\mathrm{CR}(m) = \mathit{true}$. 同様に，$T_1$ の部分系列 $n_2 \triangleright \cdots \triangleright n'$ について，帰納法の仮定より，$\mathrm{CR}(n_2) = \mathit{true}$. ゆえに，(18) より，$\mathrm{CR}(n) = \mathit{true}$.

□

述語 CR を用いて，$[P]$ を生成する正規文法 $G_{P,\mathrm{S}} = (N, T, R, S)$ を，次のように定義する．

(a) N は非終端記号の集合で，$N = \{S\} \cup \{N_i \mid n_i \in NO\}$.

(b) T は終端記号の集合で，$T = NO$.

(c) S は開始記号.

(d) R は生成規則の集合で，以下を含む.

$$S \rightarrow N_1 \quad \text{for } n_1 = IT \tag{21}$$

$$N_i \rightarrow n_i \quad \text{for } \forall n_i \in NO \tag{22}$$

また，頂点の種類によって次の規則を含む.

(1) $IS(n_i) = \mathit{call}$ のとき.

$$N_i \rightarrow n_i N_j \quad \text{for } \forall n_j.\ n_i \overset{CG}{\rightarrow} n_j \tag{23}$$

$\exists n_j.\ n_i \overset{CG}{\rightarrow} n_j, \mathrm{CR}(n_j) = \mathit{true}$ ならば，次を含む.

$$N_i \rightarrow N_k \quad \text{for } \forall n_k.\ n_i \overset{TG}{\rightarrow} n_k \tag{24}$$

(2) $IS(n_i) = \mathit{check}(NO^*)$ のとき.

$$N_i \rightarrow N_j \quad \text{for } \forall n_j.\ n_i \overset{TG}{\rightarrow} n_j \tag{25}$$

［定理 3］ $P = (NO, IS, IT, TG, CG)$ を，チェック文として $\mathit{check}(NO^*)$ のみを含むプログラムとする．このとき，$L(G_{P,\mathrm{S}}) = [P]$.

（証明） $s \in NO^*$ について $S \overset{*}{\rightarrow}_{G_{P,\mathrm{S}}} s \quad \Leftrightarrow \quad IT \triangleright \cdots \triangleright s \in [\![P]\!]$ を示せばよい．補題 6 を用いることにより，$(\Rightarrow)$ は $S \overset{*}{\rightarrow}_{G_{P,\mathrm{S}}} s$ における生成規則 (23)〜(25) の適用回数の和に関する帰納法で証明でき，$(\Leftarrow)$ はトレース $IT \triangleright \cdots \triangleright s$ の長さに関する帰納法で証明できる． □

集合 $\mathcal{S}$ に対して，$|\mathcal{S}|$ で $\mathcal{S}$ の要素数を表す．有限オートマトン $\mathcal{A}$ の状態数を $\sharp\mathcal{A}$ とする．チェック文として $\mathit{check}(NO^*)$ のみを含むプログラム $P = (NO, IS, IT, TG, CG)$ のサイズ $||P||$ を $||P|| = |NO| + |TG| + |CG|$ と定義する．また，$L(\mathcal{A})$ で有限オートマトン $\mathcal{A}$ によって受理される言語を表す．

［定理 4］ $P = (NO, IS, IT, TG, CG)$ を，チェック文として $\mathit{check}(NO^*)$ のみを含むプログラムとする．このとき，P と，決定性有限オートマトン $\mathcal{A}_\psi$ によって記述された検証条件に対する検証問題は，$O(||P|| \cdot \sharp\mathcal{A}_\psi)$ 時間可解である．

（証明） 検証問題は $[P] \subseteq L(\mathcal{A}_\psi)$ を判定することと等価．更に，定理 3 より，これは

$$L(G_{P,\mathrm{S}}) \cap \overline{L(\mathcal{A}_\psi)} = \emptyset \tag{26}$$

を判定することと等価である．$L(G_{P,\mathrm{S}})$, $L(\mathcal{A}_\psi)$ が正規言語であること及び，正規言語のクラスが補集合，積集合演算について閉じていること [9] より，条件 (26) 左辺は正規言語である．正規文法で与えられた正規言語の空集合問題は，文法の規則数の線形時間で決定可能である．また，文法 $G_{P,\mathrm{S}}$ の規則数は $O(|NO| + |TG| + |CG|)$ で押さえられる．よって，$L(G_{P,\mathrm{S}}) \cap \overline{L(\mathcal{A}_\psi)}$ を生成する正規文法の規則数は，$O((|NO| + |TG| + |CG|) \cdot \sharp\mathcal{A}_\psi)$.

一方，文法 $G_{P,\mathrm{S}}$ の構成に先立って，述語 $\mathrm{CR}(\cdot)$ の値を，以下のアルゴリズムに従って求める．まず各頂点 n に以下の情報を関連づける．

- CR; $\mathrm{CR}(n)$ の計算途中結果（初期値は $IS(n) = \mathit{return}$ を満たす頂点のみ true, それ以外は false）
- $TGok$; $IS(n) = \mathit{call}$ のときのみ使用し，$n \overset{TG}{\rightarrow} n'$ かつ $\mathrm{CR}(n') = \mathit{true}$ である頂点 n' が一つでも見つかれば，true になるフラグ（初期値は false）.
- $CGok$; $IS(n) = \mathit{call}$ のときのみ使用し，$n \overset{CG}{\rightarrow} n'$ かつ $\mathrm{CR}(n') = \mathit{true}$ である頂点 n' が一つでも見つかれば，true になるフラグ（初期値は false）.
- $n' \overset{TG}{\rightarrow} n$ を満たす頂点 n' のリスト.
- $n' \overset{CG}{\rightarrow} n$ を満たす頂点 n' のリスト.

この初期化は，$O(|NO| + |TG| + |CG|)$ 時間で実行可能である．また，CR の値が true であるとわかった後，次の処理を待つ頂点を記憶するリスト Work を用意する．Work の初期値は，フローグラフ中の $IS(n) = \mathit{return}$ を満たす頂点 n からなるリストであ

る．*Work* が空になるまで，以下の (1)，(2) を繰り返す．

(1) *Work* から任意の頂点 n を一つ選び，n を *Work* から削除する．

(2.a) $n' \overset{TG}{\rightarrow} n$, $IS(n') = check(NO^*)$ かつ，$\mathrm{CR}(n') = false$ である頂点 n' について，$\mathrm{CR}(n') := true$ とし，n' を *Work* に追加する．

(2.b) $n' \overset{TG}{\rightarrow} n$, $IS(n') = call$ かつ，$\mathrm{CR}(n') = false$ である頂点 n' について，$CGok(n') = true$ なら，$\mathrm{CR}(n') := true$ として，n' を *Work* に追加する．$CGok(n') = false$ なら，$TGok(n') := true$ とする．

(2.c) $n' \overset{CG}{\rightarrow} n$, $IS(n') = call$ かつ，$\mathrm{CR}(n') = false$ である頂点 n' について，$TGok(n') = true$ なら，$\mathrm{CR}(n') := true$ として，n' を *Work* に追加する．$TGok(n') = false$ なら，$CGok(n') := true$ とする．

このアルゴリズムで，各頂点はたかだか 1 回しか *Work* に含まれない．また各頂点が *Work* から削除されたとき，その頂点に向かう各有向辺に対して，(2) の処理がたかだか 1 回しか実行されない．したがって，このアルゴリズムは全体で $O(|NO| + |TG| + |CG|)$ 時間で実行可能である．

以上より，本問題は，$O((|NO| + |TG| + |CG|) \cdot \sharp\mathcal{A}_\psi) = O(||P|| \cdot \sharp\mathcal{A}_\psi)$ 時間可解となる． □

6. むすび

スタック検査機能をもつプログラムに対する検証問題が決定可能であること，自明なチェック文のみを含む場合，プログラムサイズの多項式時間で可解であることを示した．後者の部分クラス自体は実用上の意義が薄いが，JDK1.2 の checkPermission 等をモデル化できる実用的な部分クラスに対する検証問題を，この部分クラスの問題に帰着して効率良く解くことが可能である．これについては，検証問題の計算複雑さ [12] とともに別途報告する予定である．

謝辞 時相論理式と正規言語の関係について御教示頂いた大阪大学大学院基礎工学研究科浜口清治助教授，トレース集合の所属する言語クラスに関して貴重な御指摘を頂いた同研究科石原靖哲講師，本研究全般にわたり御協力頂いた伊加田恵志氏（現在沖電気工業（株））に感謝致します．

文　献

[1] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin, “A calculus for access control in distributed systems,” ACM Trans. Prog. Lang. and Systems, vol.15, no.4, pp.706–734, 1993.

[2] A.V. Aho, “Indexed grammars — An extension of context-free grammars,” J. Assoc. Comput. Mach., vol.15, no.4, pp.647–671, 1968.

[3] E.M. Clarke, Jr., O. Grumberg, and D.A. Peled, Model Checking, The MIT Press, 1999.

[4] D.E. Denning, “A lattice model of secure information flow,” Commun. ACM, vol.19, no.5, pp.236–243, 1976.

[5] D. Denning and P. Denning, “Certification of programs for secure information flow,” Commun. ACM, vol.20, no.7, pp.504–513, 1977.

[6] E.A. Emerson, “Temporal and Modal Logic,” Handbook of Theoretical Computer Science, pp.1023–1024, Elsevier, 1990.

[7] L. Gong, M. Mueller, H. Prafullchandra, and R. Schemers, “Going beyond the sandbox: An overview of the new security architecture in the Java^{TM} development kit 1.2,” USENIX Symp. on Internet Technologies and Systems, pp.103–112, 1997.

[8] N. Heintze and J.G. Riecke, “The SLam calculus: Programming with secrecy and integrity,” 25th ACM Symp. on Principles of Programming Languages, pp.365–377, 1998.

[9] J.E. Hopcroft and J.D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, 1979.

[10] 伊加田恵志，新田直也，高田喜朗，関　浩之，“スタック検査機能を持つプログラムの制御フロー解析に基づくセキュリティ検証法，” 信学技報，ISEC2000-78, 2000.

[11] T. Jensen, D. Le Métayer, and T. Thorn, “Verification of control flow based security properties,” 1999 IEEE Symp. on Security and Privacy, pp.89–103, 1999.

[12] N. Nitta, Y. Takata, and H. Seki, “Complexity of the security verification problem for programs with stack inspection,” ソフトウェア科学会第 3 回プログラミング及びプログラミング言語ワークショップ (PPL2001) 論文集，pp.53–60, 2001.

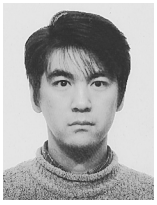
[13] P. Ørbæk, “Can you trust your data?,” TAPSOFT ’95, LNCS 915, pp.575–589, 1995.

[14] J. Palsberg and M.I. Schwartzbach, Object-Oriented Type Systems, John Wiley & Sons, 1994.

[15] D. Volpano and G. Smith, “A type-based approach to program security,” TAPSOFT ’97, LNCS 1214, pp.607–621, 1997.

[16] D.S. Wallach and E.W. Felten, “Understanding Java stack inspection,” 1998 IEEE Symp. Security and Privacy, pp.52–63, 1998.

（平成 13 年 4 月 16 日受付，8 月 24 日再受付）



新田　直也

平 3 阪大・工・原子力卒．平 4〜10（株）ダイナウェアに勤務．平 10 奈良先端大・情報博士前期課程入学．平 11 同課程了，現在，同大博士後期課程在学中．形式論理，ソフトウェア基礎理論に関する研究に従事．

高田　喜朗　（正員）

平 4 阪大・基礎工・情報卒．平 9 同大大学院博士後期課程了．同年奈良先端大・情報助手．現在に至る．博士（工学）．インタラクティブシステムの設計法，情報検索に関する研究に従事．

関　　浩之　（正員）

昭 57 阪大・基礎工・情報卒．昭 62 同大大学院博士後期課程了．同年同大・基礎工・情報助手．同講師，同助教授を経て，平 6 奈良先端大・情報助教授，平 8 同教授，現在に至る．工博．形式言語理論，ソフトウェア基礎理論に関する研究に従事．