論　文

束構造をもつセキュリティクラスに基づく再帰的プログラムに対する情報フロー解析法

國信　茂太[†]　　高田　喜朗[†]　　関　浩之[†]　　井上　克郎[††]

An Information Flow Analysis of Recursive Programs Based on a Lattice Model of Security Classes

Shigeta KUNINOBU[†], Yoshiaki TAKATA[†], Hiroyuki SEKI[†], and Katsuro INOUE[††]

あらまし　本論文では，一般に再帰を含むプログラムに対して情報フローを効率良く解析する手法を示す．この手法において，データの機密度を表すセキュリティレベルは任意の有限分配束によって与えることができる．本論文では，提案する情報フロー解析法の健全性を証明し，このアルゴリズムがプログラムの記述長の 3 乗のオーダで実行できることを示す．更に，引数内の情報を隠すことができるような関数を適切に導入できるように，合同関係を用いてアルゴリズムを拡張する．最後に，試作システムを用いて提案アルゴリズムを実行した結果を紹介する．

キーワード　情報フロー解析，セキュリティクラス（SC），不動点計算

1. まえがき

不特定多数の人々が利用するシステムにおいて，望ましくない情報の漏洩(えい)を防ぐのは重要なことである．望ましくない情報の漏洩を防ぐ手段の一つに Mandatory Access Control（MAC）[18] がある．MAC においては，データ及びユーザ（若しくはプロセス）には，top-secret，confidential，unclassified といったセキュリティレベルを表すラベルが付けられる．データ d に対するラベルはセキュリティクラス（SC）と呼ばれ，$SC(d)$ と書く．また，ユーザ u に対するラベルはクリアランスと呼ばれ，$clear(u)$ と書く．ユーザ u がデータ d を読んでよいのは，$clear(u) \geq SC(d)$ のとき，かつそのときのみである．しかし，クリアランスが $SC(d)$ 以上であるユーザがデータ d を読み込み，このデータ d に基づいてデータ d' を作成しクリアランスが $SC(d)$ より低いユーザが読出し可能な記憶域 d' に書き込むと，データ d' にはデータ d の情報が含まれている可能性があるので，望ましくない情報の漏れが生じ得る．このような情報の漏洩を防ぐためには，プログラムとプログラムへの入力となるデータの SC が与えられたとき，そのプログラムがどの記憶域にどのような SC のデータを出力するのか（情報フロー）を静的に解析できることが望ましい．これまで，束構造をもつ SC に基づくプログラム解析法がいくつか提案されている（後述の関連研究を参照）．しかし，それらの解析法は，解析対象となるプログラムが単純で特に再帰手続きを考慮していない，解析法の健全性の証明が行われていないなどの問題があった．

本論文では，一般に再帰を含む手続き型プログラムに対する情報フロー解析法を提案する．このアルゴリズムは，まず，解析対象プログラム中のすべての実行文について，その文の実行によって起こる情報フローを表すような関係式を定義する．その後，それらの関係式を同時に満たす最小解を最小不動点計算によって求める．本論文では，アルゴリズムを抽象解釈を用いて表現し，これに基づき，アルゴリズムの健全性を証明する．提案アルゴリズムは，与えられたプログラム $Prog$ に対し，$O(N^3)$ の時間計算量で解析を行うこと

† 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, 8916–5 Takayama, Ikoma-shi, 630–1010 Japan
†† 大阪大学大学院情報科学研究科，豊中市
Graduate School of Information Science and Technology, Osaka University, 1–3 Machikaneyama, Toyonaka-shi, 560–8531 Japan

ができる．ただし，N は *Prog* の記述長である．最後に，提案手法に基づいた試作システムにおける実験結果を紹介する．

本論文で紹介するアルゴリズムを含めて，従来の解析手法では，組込み関数 θ（加減算など）に対する解析結果を，θ への入力となる引数の最小上界と定義している．すなわち，θ のすべての引数内の情報が θ の返り値に流れると仮定している．しかし，この仮定は暗号化関数などに対しては適切とはいえない．このような関数においては，関数の結果から引数に含まれる情報を復元することは極めて困難であるからである．本論文では，このような関数を適切にモデル化できるように合同関係を用いてモデルの拡張を行う．

本論文の構成は以下のとおりである．**2.** では，解析対象となるプログラムの構文及び操作的意味を定義する．**3.** では，抽象解釈を用いて提案アルゴリズムを形式的に記述し，アルゴリズムの健全性を証明する．また，提案アルゴリズムの時間計算量を解析する．簡単な例も **3.** で紹介する．**4.** ではモデルの拡張を行い，**5.** で，試作システムを用いて得られた結果を紹介する．

[関連研究]　Denning らは [6], [7] において SC 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を静的に解析する方法を初めて提案した．その後，Denning らの解析手法は様々な形で定式化，拡張されている．例えば，[4] では Hoare の公理論的方法，[17] では抽象解釈，[8], [11], [24] では型理論を用いて定式化を行っている．

型理論を用いた手法では，SC を型とみなし，「あるプログラムが型付け可能ならば，そのプログラムは望ましくない情報の漏洩を起こさない（noninterference, 非干渉性）」が成り立つように型システムを定義する．[24] では，簡単な手続き型プログラムに対して，静的に情報フローを解析する手法が提案されており，その健全性も示されている．しかし，[24] において，解析対象プログラムは再帰を含まないものに限られている．一方，本論文の解析手法は，再帰を含むプログラムに対しても解析を行うことができる．[8] では，SLam 計算と呼ばれる関数型言語に対する，非干渉性を表す型システムが提案されている．しかし，[8] では，代入文などの機能を導入して拡張したモデルに対する解析アルゴリズムの健全性は証明されていない．[22] は，[24] における型システムが分散環境においては健全性を満たしていないことを示し，分散処理機能をもつ言語に対する新しい型システムを提案している．本論文の解析手法をどのようにして分散環境に対応させるかは今後の課題である．

なお，SC が束構造をもつと仮定するのは，二つのデータ d_1 及び d_2 の情報を含むデータ d_3 の SC は，$SC(d_1)$ と $SC(d_2)$ の最小上界と定義するのが自然だからである．一方，現実的なシステムにおいては SC が束構造をもたないこともあり得る．例えば，ユーザ U_1，U_2 及びユーザグループ G_1，G_2 があり，U_1，U_2 の両方が G_1，G_2 の両方に所属していると仮定する．このとき，$SC(G_i) \leq SC(U_j)$ $(i = 1, 2, j = 1, 2)$ である．しかしながら，$SC(G_1)$ と $SC(G_2)$ の最小上界（$SC(G_3)$ と書く）を仮想的に考えると，$SC(G_i) \leq SC(U_j)$ $(i = 1, 2, j = 1, 2)$ と最小上界の定義より，$SC(G_3) \leq SC(U_j)$ $(j = 1, 2)$ が成り立つ．直観的に，$SC(G_1)$ のデータ d_1 と $SC(G_2)$ のデータ d_2 の両方の情報を含むデータの SC は $SC(G_3)$ となる．$SC(G_3) \leq SC(U_j)$ $(j = 1, 2)$ より，このデータは U_1，U_2 両ユーザが読めるデータであることを意味し，現実的にも自然である．このように SC に束構造をもたせることは自然な仮定であり，本論文でも SC は束構造をもつと仮定する．

SC は通常 {top-secret, confidential, unclassified} といった単純な束構造でモデル化されている．[14] は，分散ラベルと呼ばれる SC を提案し，declassification の機能を導入している．分散ラベルのモデルに基づき [13] では情報フローに対する静的な型システムと動的なアクセス制御機能を併せ持つ JFLOW と呼ばれるプログラミング言語が提案されている．しかし，解析法の健全性は示されていない．

分散システムにおける非干渉性はプロセス代数を用いて広く研究されている（例えば，[1], [19], [20]）．[1] 等では，暗号プロトコルの解析を spi と呼ばれるプロセス代数を用いて行っている．

最近，Java development kit 1.2 におけるスタック検査のように，動的なアクセス制御を行うプログラムの制御フロー解析法が研究されている．例えば，[10], [16] では，プログラム P と P が満たすべき性質 ψ が与えられたとき，P のすべての到達可能な状態が ψ を満足するかどうかを検証する方法を提案している．

データ工学の分野でも，分散環境及びオブジェクト指向環境におけるアクセス制御法及び情報フロー解析法の研究が広く行われている（[5] 参照）．例えば，[21] ではオブジェクト指向データベースにおける不正な情報フローを防ぐ情報フロー制御アルゴリズムが提案さ

れている．しかし，これらのアルゴリズムではプログラムの意味を考慮していない．推論攻撃に対する検証問題など，意味論を考慮した手法が [9], [23] で議論されている．

2. 諸 定 義

2.1 プログラムの構文

この節では，解析対象となるプログラムの構文と形式的意味を定義する．このプログラミング言語は C 言語に類似した手続き型言語である．

プログラム *Prog* は関数定義の集合である．関数定義は以下の形式で与えられる．

$$f(x_1,\ldots,x_n) \text{ local } y_1,\ldots,y_m \ \{P_f\}$$

ただし，f は関数名，$x_1,\ldots,x_n$ は f の仮引数，$y_1,\ldots,y_m$ は局所変数，P_f は関数本体である．P_f の構文は以下の BNF 式で与えられる．ただし，c は定数，x は局所変数または仮引数，f はプログラム中で定義された関数，θ は加減算などの組込み演算を表す．$cseq$ から生成される任意の文字列が P_f となり得る．

$$\begin{aligned}
cseq &::= cmd \mid cmd_1; cseq\\
cmd &::= \text{if } exp \text{ then } cseq \text{ else } cseq \text{ fi}\\
&\quad \mid \text{return } exp\\
cseq_1 &::= cmd_1 \mid cmd_1; cseq_1\\
cmd_1 &::= x := exp\\
&\quad \mid \text{if } exp \text{ then } cseq_1 \text{ else } cseq_1 \text{ fi}\\
&\quad \mid \text{while } exp \text{ do } cseq_1 \text{ od}\\
exp &::= c \mid x \mid f(exp,\ldots,exp) \mid \theta(exp,\ldots,exp)
\end{aligned}$$

exp，cmd 及び cmd_1，$cseq$ 及び $cseq_1$ から生成される文字列をそれぞれ，式，文，文の系列と呼ぶ．プログラム *Prog* の実行は，*Prog* の要素のうち関数名が *main* である関数を評価することで行われる．この関数を main 関数と呼ぶ．*Prog* の入力は main 関数に与える実引数，出力は main 関数の返り値である．

2.2 プログラムの意味

プログラムの意味を定義するため，次の型を仮定する．ただし，$\times$ で直積，$+$ で分離和を表す．

「*val* 型（値を表す型）」 プログラムに現れる n 引数基本演算子 θ に対し，n 引数関数 $\theta_{\mathcal{I}} : val\times\cdots\times val \to val$ が定義されているとする．プログラム内で生成若しくは計算された値は，すべて *val* 型であるとする．

「*store* 型（記憶域を表す型）」二つの関数

$$\begin{aligned}
&lookup : store \times var \to val\\
&update : store \times var \times val \to store
\end{aligned}$$

があり，以下の公理を満たすとする．

$$\begin{aligned}
&lookup(update(\sigma,x,v),y)\\
&\quad = \text{ if } x = y \text{ then } v \text{ else} lookup(\sigma,y)
\end{aligned}$$

読みやすさのため，$\sigma(x) \equiv lookup(\sigma,x)$，$\sigma[x := v] \equiv update(\sigma,x,v)$ と略記する．また，$\bot_{store}$ で，すべての変数の値が未定義の記憶域を表す．

プログラムの意味を与える意味関数を定義する．この関数は，記憶域,（式，文，文の系列のいずれか）を引数にとり，記憶域または値を返す．

$$\begin{aligned}
&\models : (store \to exp \to val)\\
&\quad +(store \to cmd \to (store + val))\\
&\quad +(store \to cseq \to (store + val))
\end{aligned}$$

- 「$\sigma \models M \Rightarrow v$」は，記憶域 σ のもとで式 M を評価した値が v であることを表す．
- 「$\sigma \models C \Rightarrow \sigma'$」は，記憶域 σ のもとで文 C を実行した直後の記憶域が σ' であることを表す．
- 「$\sigma \models C \Rightarrow v$」は，記憶域 σ のもとで文 C を実行した直後，値 v が返されることを表す．これは，C が return M の場合のみ当てはまる．
- 文の系列に対しても同様．

以下に意味関数を定義する公理と推論規則を与える．ここで，メタ変数として以下を用いる．

$x, x_1, \ldots, y_1, \ldots : var$　　$M, M_1, \ldots : exp$
$C : cmd$ 及び cmd_1　　$\sigma, \sigma', \sigma'' : store$
$P, P_1, P_2 : cseq$ 及び $cseq_1$

$$\text{(CONST)}\quad \sigma \models c \Rightarrow c_{\mathcal{I}}$$

$$\text{(VAR)}\quad \sigma \models x \Rightarrow \sigma(x)$$

$$\text{(PRIM)}\quad \frac{\sigma \models M_i \Rightarrow v_i \ \ (1 \le i \le n)}{\sigma \models \theta(M_1,\ldots,M_n) \Rightarrow \theta_{\mathcal{I}}(v_1,\ldots,v_n)}$$

$f(x_1,\ldots,x_n)$ local $y_1,\ldots,y_m$ $\{P_f\}$,
$\sigma' = \bot_{store}[x_1 := v_1]\cdots[x_n := v_n]$ のとき，

$$\text{(CALL)}\quad \frac{\sigma \models M_i \Rightarrow v_i \ \ (1 \le i \le n) \qquad \sigma' \models P_f \Rightarrow v}{\sigma \models f(M_1,\ldots,M_n) \Rightarrow v}$$

$$\text{(ASSIGN)}\quad \frac{\sigma \models M \Rightarrow v}{\sigma \models x := M \Rightarrow \sigma[x := v]}$$

$$\text{(IF1)}\quad \frac{\sigma \models M \Rightarrow \text{true} \quad \sigma \models P_1 \Rightarrow \sigma'\ (rsp.\ v)}{\sigma \models \text{if } M \text{ then } P_1 \text{ else } P_2 \text{ fi} \Rightarrow \sigma'\ (rsp.\ v)}$$

$$\text{(IF2)}\quad \frac{\sigma \models M \Rightarrow \text{false} \quad \sigma \models P_2 \Rightarrow \sigma'\ (rsp.\ v)}{\sigma \models \text{if } M \text{ then } P_1 \text{ else } P_2 \text{ fi} \Rightarrow \sigma'\ (rsp.\ v)}$$

$$\text{(WHILE1)}\quad \frac{\sigma \models M \Rightarrow \text{true} \quad \sigma \models P \Rightarrow \sigma' \quad \sigma' \models \text{while } M \text{ do } P \text{ od} \Rightarrow \sigma''}{\sigma \models \text{while } M \text{ do } P \text{ od} \Rightarrow \sigma''}$$

$$\text{(WHILE2)}\quad \frac{\sigma \models M \Rightarrow \text{false}}{\sigma \models \text{while } M \text{ do } P \text{ od} \Rightarrow \sigma}$$

$$\text{(RETURN)}\quad \frac{\sigma \models M \Rightarrow v}{\sigma \models \text{return } M \Rightarrow v}$$

$$\text{(CONCAT)}\quad \frac{\sigma \models C \Rightarrow \sigma' \quad \sigma' \models P \Rightarrow \sigma''\ (rsp.\ v)}{\sigma \models C; P \Rightarrow \sigma''\ (rsp.\ v)}$$

3. 解析アルゴリズム

以下では，有限集合 $SCset$ が与えられていると仮定する．$SCset$ の要素をセキュリティクラス（SC）と呼ぶ．SC は値の機密度を表す．$SCset$ 上には半順序 $\sqsubseteq$ が定義され，$SCset$ は $\sqsubseteq$ に関して分配束をなすと仮定する．特に断らない限り最小上界演算を $\sqcup$，$SCset$ の最小元を $\bot$ と表す．直観的に，$\tau_1 \sqsubseteq \tau_2$ は，τ_2 が τ_1 よりも機密度が高いか等しいこと，クリアランスが τ_2 であるユーザは，SC が τ_1 である値にアクセスできることを表す．以下は単純な $SCset$ の例である．

$$SCset = \{low, high\}, \quad low \sqsubseteq high.$$

解析の目的は，プログラムとプログラムへの入力となる各値の SC が与えられたとき，出力値の SC を知ることである．厳密には，**3.3** で述べる健全性を満たすような解析結果を求めることが目的である．

SC が「高い（$high$）/低い（low）」とするアプローチを利用したセキュリティ保全法として，**1.** で述べたようにデータベースの主要なアクセス制御法である MAC [18] が挙げられる．MAC では，以下の二つの規則に基づいてアクセス制御を行う．

- ユーザ u がデータ d を読んでよいのは，$SC(d) \sqsubseteq clear(u)$ のとき，かつそのときのみ．
- ユーザ u がデータ（ストア）d に書いてよいのは，$clear(u) \sqsubseteq SC(d)$ のとき，かつそのときのみ．

データベース内のデータを一つ以上引数として読み込み，新たなデータ d' を作り出すようなプログラムがある場合，プログラムの入力となるデータの SC から，d' の SC を解析することことになる．その結果に応じて d' をデータベースの特定のストアに書き戻してよいかの解析に利用できる．

3.1 で解析アルゴリズムを記述し，**3.4** でアルゴリズムの健全性を示す．

3.1 アルゴリズム

アルゴリズムを記述するために，以下の型を用いる．
「sc 型（セキュリティクラス）」 $SCset$ そのもの．
「$fname$ 型（関数名）」$Prog$ 内で定義されている関数の名前の集合．
「$\underline{store}$ 型（記憶域の SC）」

$$update : \underline{store} \times var \times sc \to \underline{store}$$
$$lookup : \underline{store} \times var \to sc$$

$\underline{store}$ 型でも，$store$ 型と同様の略記法を用いる．$\underline{\sigma}$ を $\underline{store}$ 型の元とするとき，$\underline{\sigma}(x)$ は，変数 x の SC の解析（途中）結果を表す．$SCset$ で定義された半順序 $\sqsubseteq$ を型 $\underline{store}$ 上に次のように拡張することにより，$\underline{store}$ に束の構造を与えることができる．$\underline{\sigma}$，$\underline{\sigma}'$ を $\underline{store}$ 型の値とする．

$$\underline{\sigma} \sqsubseteq \underline{\sigma}' \ \Leftrightarrow \forall x \in var.\ \underline{\sigma}(x) \sqsubseteq \underline{\sigma}'(x)$$

$\underline{store}$ の最小元は $\forall x \in var.\ \underline{\sigma}(x) = \bot$ であるような $\underline{\sigma}$ であり，これを $\bot_{\underline{store}}$ と表す．
「$\underline{fun}$ 型（関数の SC）」 $\underline{store}$ 型と同様，次の関数が定義されている．

$$lookup : \underline{fun} \times fname \to (sc \times \cdots \times sc \to sc)$$
$$update : \underline{fun} \times fname \times (sc \times \cdots \times sc \to sc) \to \underline{fun}$$

また，$F \in \underline{fun}$，$f \in fname$，$\psi : sc \times \cdots \times sc \to sc$ に対して，次の略記を用いる．

$$F[f] \equiv lookup(F, f)$$
$$F[f := \psi] \equiv update(F, f, \psi)$$

$F[f](\tau_1, \ldots, \tau_n)$ は引数の SC が $\tau_1, \ldots, \tau_n$ であるときの n 引数関数 f の SC の解析（途中）結果を表す．

sc の束構造を $\underline{store}$ 型に拡張したのと同様に，$\underline{fun}$ 型にも束構造を導入する．最小元を $\bot_{\underline{fun}}$ と表す．
「cv-$\underline{fun}$ 型（covariant $\underline{fun}$ 型）」 $\underline{fun}$ 型の値 F のうち，条件

$\tau_i \sqsubseteq \tau_i'\ (1 \le i \le n)$ ならば $F[f](\tau_1, \ldots, \tau_n) \sqsubseteq F[f](\tau_1', \ldots, \tau_n')$

を満たすものすべてからなる．
メタ変数として，**2.2** のものに加えて以下を用いる．

$\underline{\sigma}, \underline{\sigma}', \underline{\sigma}'' : \underline{store} \quad F, F_1, F_2 : \underline{fun}$

以下，情報フローを解析する関数 $\mathcal{A}[\![\cdot]\!]$ を定義する．次のような if 文を考える．

if $x = 0$ then $y := 0$ else $y := 1$ fi

この if 文内で，y への代入文の右辺には x は現れていないが，この if 文の実行直後において y の値が 0 か 1 かを調べることにより，x の値が 0 かどうかを知ることができる．すなわち，変数 x の情報が変数 y に流れたと考えることができる．このように，if 文の条件部の情報が then 及び else 以下の文に流れること，及び while 文の条件部の情報が do 以下の文に流れることを implicit flow と呼ぶ．各文または文の系列について，それが属する if 文及び while 文の条件部の SC の最小上界を，その文（文の系列）での implicit flow の SC と定義する．$\mathcal{A}$ の計算においては，各文（の系列）が，どの if 文・while 文に属するかを記憶しておく代わりに，implicit flow の SC の解析結果を $\mathcal{A}$ の第 4 引数を用いて受け渡す．

$$\begin{aligned}\mathcal{A} : &(exp \times \underline{fun} \times \underline{store} \to sc)\\ &+ (cmd \times \underline{fun} \times \underline{store} \times sc \to \underline{store})\\ &+ (cseq \times \underline{fun} \times \underline{store} \times sc \to \underline{store})\end{aligned}$$

- 「$\mathcal{A}[\![M]\!](F, \underline{\sigma}) = \tau$」は，各関数の SC の解析結果 F, 記憶域の SC $\underline{\sigma}$ の下で，式 M の SC を τ と解析することを表す．
- 「$\mathcal{A}[\![C]\!](F, \underline{\sigma}, \nu) = \underline{\sigma}'$」は，各関数の SC の解析結果 F, 記憶域の SC $\underline{\sigma}$, implicit flow の SC ν の下で，文 C を実行した直後の記憶域の SC を $\underline{\sigma}'$ と解析することを表す．
- 文の系列に対しても同様．

以下に $\mathcal{A}$ の定義を与える．

(CONST) $\mathcal{A}[\![c]\!](F, \underline{\sigma}) = \bot$

(VAR) $\mathcal{A}[\![x]\!](F, \underline{\sigma}) = \underline{\sigma}(x)$

(PRIM) $\mathcal{A}[\![\theta(M_1, \ldots, M_n)]\!](F, \underline{\sigma}) = \bigsqcup_{1 \le i \le n} \mathcal{A}[\![M_i]\!](F, \underline{\sigma})$

(CALL) $\mathcal{A}[\![f(M_1, \ldots, M_n)]\!](F, \underline{\sigma}) = F[f](\mathcal{A}[\![M_1]\!](F, \underline{\sigma}), \ldots, \mathcal{A}[\![M_n]\!](F, \underline{\sigma}))$

(ASSIGN) $\mathcal{A}[\![x := M]\!](F, \underline{\sigma}, \nu) = \underline{\sigma}[x := \mathcal{A}[\![M]\!](F, \underline{\sigma}) \sqcup \nu]$

(IF) $\mathcal{A}[\![\text{if } M \text{ then } P_1 \text{ else } P_2 \text{ fi}]\!](F, \underline{\sigma}, \nu) = \mathcal{A}[\![P_1]\!](F, \underline{\sigma}, \nu \sqcup \tau) \sqcup \mathcal{A}[\![P_2]\!](F, \underline{\sigma}, \nu \sqcup \tau)$
ただし，$\tau = \mathcal{A}[\![M]\!](F, \underline{\sigma})$

(WHILE) $\mathcal{A}[\![\text{while } M \text{ do } P \text{ od}]\!](F, \underline{\sigma}, \nu) = fix(\lambda X.\underline{\sigma} \sqcup \mathcal{A}[\![P]\!](F, \underline{\sigma} \sqcup X, \nu \sqcup \mathcal{A}[\![M]\!](F, \underline{\sigma} \sqcup X)))$

(RETURN) ret を関数の返り値を記憶する特別な局所変数とする．
$\mathcal{A}[\![\text{return } M]\!](F, \underline{\sigma}, \nu) = \underline{\sigma}[ret := \mathcal{A}[\![M]\!](F, \underline{\sigma}) \sqcup \nu]$

(CONCAT) $\mathcal{A}[\![C; P]\!](F, \underline{\sigma}, \nu) = \mathcal{A}[\![P]\!](F, \mathcal{A}[\![C]\!](F, \underline{\sigma}, \nu), \nu)$

プログラム $Prog$ を入力として $Prog$ で定義された各関数の情報フローを解析する関数 $\mathcal{A}[\![\cdot]\!] : program \to \underline{fun} \to \underline{fun}$ を次のように定義する：

$Prog \equiv \{f(x_1, \ldots, x_n) \text{ local } y_1, \ldots, y_m \ \ \{P_f\}, \ldots\}$

とするとき，

$$\begin{aligned}&\mathcal{A}[\![Prog]\!](F)\\ &= F[f := \lambda\tau_1 \ldots \tau_n.(\mathcal{A}[\![P_f]\!](F, \bot_{\underline{store}}[x_1 := \tau_1] \cdots [x_n := \tau_n], \bot)(ret)) \mid Prog \text{ で定義された各 } n \text{ 引数関数 } f] \qquad (1)\end{aligned}$$

束 $(S, \preceq)$ における関数 $f : S \to S$ に対して，f の最小不動点を $fix(f)$ で表す．プログラム $Prog$ に対して，$\mathcal{A}[\![Prog]\!]$ の最小不動点を与える関数

$$\mathcal{A}^*[\![Prog]\!] = fix(\lambda F.\mathcal{A}[\![Prog]\!](F)) \qquad (2)$$

が解析アルゴリズムである．後述の補題 3.2 より，$\mathcal{A}[\![Prog]\!]$ は有限の束 cv-$\underline{fun}$ 上の単調関数となるので，

$$\mathcal{A}^*[\![Prog]\!] = \bigsqcup_{i \ge 0} \mathcal{A}[\![Prog]\!]^i(\bot_{\underline{fun}}) \qquad (3)$$

が成り立つ[12]．ここで，$f^0(x) = x$，$f^{i+1}(x) = f(f^i(x))$．すなわち，$\mathcal{A}^*[\![Prog]\!]$ は，すべての内容が最小元であるような関数の SC から開始して，$\mathcal{A}[\![Prog]\!]$ を，関数の SC が変化しなくなるまで繰り返し適用することにより計算できる．

3.2 解 析 例

この節では，次のような 2 関数からなるプログラムを例に，どのように解析が行われるのかを示す．ここで，SC の集合は $\{low, high\}$ で，$low \sqsubseteq high$ とする．

```
main(x, y) {              f(x) {
    while x > 0 do            if x > 0 then
        y := x + 1;               return x * f(x − 1)
        x := y − 4            else
    od;                           return 0
    return f(x) + y           fi
}                         }
```

このプログラムを解析するためには，次の関係式を用いて F が変化しなくなるまで，F の更新を繰り返せばよい．

$$
\begin{aligned}
&F = \\
&F[main := \\
&\lambda\tau_1\tau_2.(\mathcal{A}[\![P_{main}]\!](F, \bot_{\underline{store}}[x := \tau_1][y := \tau_2], \bot)(ret))] \\
&\quad [f := \lambda\tau_1.(\mathcal{A}[\![P_f]\!](F, \bot_{\underline{store}}[x := \tau_1], \bot)(ret))]
\end{aligned}
$$

次の表は，F の更新過程を示したものである．第 i 列の SC は第 $i-1$ 列の各値を用いて計算された値を表している．

	0	1
$F[main]$	$\lambda\tau_1\tau_2.\bot$	$\lambda\tau_1\tau_2.\tau_2$
$F[f]$	$\lambda\tau_1.\bot$	$\lambda\tau_1.\tau_1$

2	3
$\lambda\tau_1\tau_2.\tau_1 \sqcup \tau_2$	$\lambda\tau_1\tau_2.\tau_1 \sqcup \tau_2$
$\lambda\tau_1.\tau_1$	$\lambda\tau_1.\tau_1$

この表から，$\mathcal{A}^*[\![Prog]\!][main](\tau_1, \tau_2) = \tau_1 \sqcup \tau_2$ であることがわかる．すなわち，実引数の SC が二つとも low のとき main 関数の返り値の SC は low，それ以外のとき main 関数の返り値の SC は $high$ となり得ることがわかる．

3.3 健全性の定義

一般に，次の型をもつ関数 $\mathcal{Z}$ を，解析アルゴリズムという．

$$\mathcal{Z}^*[\![\cdot]\!] : program \to fname \to (sc \times \cdots \times sc \to sc)$$

$\mathcal{Z}^*[\![Prog]\!][f](\tau_1, \ldots, \tau_n) = \tau$ であるとき，$\mathcal{Z}^*[\![\cdot]\!]$ は，$Prog$ で定義された n 引数関数 f に対して，引数の SC が $\tau_1, \ldots, \tau_n$ であるとき，f の SC を τ と解析したことを表す．

［定義 3.1］ 以下の条件が成り立つとき，解析アルゴリズム $\mathcal{Z}^*[\![\cdot]\!]$ は健全性（soundness）を満たすという：

$Prog$ を任意のプログラム，$main$ を $Prog$ の main 関数とする．

$$
\begin{aligned}
&\mathcal{Z}^*[\![Prog]\!][main](\tau_1, \ldots, \tau_n) = \tau \\
&\bot_{store} \models main(v_1, \ldots, v_n) \Rightarrow v \\
&\bot_{store} \models main(v'_1, \ldots, v'_n) \Rightarrow v' \\
&\forall i\,(1 \le i \le n) : \tau_i \sqsubseteq \tau.\ v_i = v'_i
\end{aligned}
$$

であると仮定する．このとき，$v = v'$ が成り立つ．

□

上の定義で，解析アルゴリズムが健全性を満たすとは，「解析結果が τ であると仮定する．このとき，SC が τ 以下であるような実引数の値がすべて等しいならば，それ以外の実引数が変化しても実行結果の値は不変である」ことを表す．つまり，解析アルゴリズムが「関数 f の SC は τ」と答えたときは，τ 以下でない実引数の情報は，実行結果に流れ出ないと保証されることを表す．

3.4 提案アルゴリズムの健全性

以降，**3.1** で定義したアルゴリズム $\mathcal{A}$（及び $\mathcal{A}^*$）の健全性を示す．

次の補題は関係式 (3) の正当性を保証する．

［補題 3.2］ (a) F が cv-$\underline{fun}$ 型ならば，$\mathcal{A}[\![Prog]\!](F)$ も cv-$\underline{fun}$ 型である．

(b)（単調性） F_1, F_2 は cv-$\underline{fun}$ 型であると仮定する．このとき，$F_1 \sqsubseteq F_2$ ならば $\mathcal{A}[\![Prog]\!](F_1) \sqsubseteq \mathcal{A}[\![Prog]\!](F_2)$．

（証明の方針） $Prog$ の構造に関する帰納法により示せる． □

次の二つの補題は補題 3.5 の証明に用いられる．

［補題 3.3］（implicit flow の性質） $\mathcal{A}[\![P]\!](F, \underline{\sigma}, \nu) = \underline{\sigma}', \sigma \models P \Rightarrow \sigma', \nu \not\sqsubseteq \underline{\sigma}'(y)$ ならば，$\sigma(y) = \sigma'(y)$．

（証明の方針） $\sigma \models P \Rightarrow \sigma'$ を導出するのに用いた推論規則の適用回数に関する帰納法により示せる． □

［補題 3.4］（implicit flow の性質） $\mathcal{A}[\![P]\!](F,\underline{\sigma},\nu)=\underline{\sigma}'$, $\sigma\models$ while M do P od $\Rightarrow\sigma'$, $\nu\not\sqsubseteq\underline{\sigma}'(y)$ ならば，$\sigma(y)=\sigma'(y)$.
（証明の方針） 補題 3.3 を用い，$\sigma\models \mathit{while}\mathrm{M}\mathit{do}\mathrm{P}\mathit{od}\Rightarrow\sigma'$ を導出するのに用いた推論規則の適用回数に関する帰納法により示せる． □

次の補題は，提案アルゴリズムによる解析結果と非干渉性との関係を表している．
［補題 3.5］ $F=\mathcal{A}^*[\![Prog]\!]$ とおく．

(a) $\mathcal{A}[\![M]\!](F,\underline{\sigma})=\tau$, $\sigma_1\models M\Rightarrow v_1$, $\sigma_2\models M\Rightarrow v_2$, $\forall x:\underline{\sigma}(x)\sqsubseteq\tau.\ \sigma_1(x)=\sigma_2(x)$ ならば，$v_1=v_2$.

(b) $\mathcal{A}[\![P]\!](F,\underline{\sigma},\nu)=\underline{\sigma}'$, $\sigma_1\models P\Rightarrow\sigma_1'$, $\sigma_2\models P\Rightarrow\sigma_2'$, $\underline{\sigma}'(y)=\tau$, $\forall x:\underline{\sigma}(x)\sqsubseteq\tau.\ \sigma_1(x)=\sigma_2(x)$ ならば，$\sigma_1'(y)=\sigma_2'(y)$.

(c) $\mathcal{A}[\![P]\!](F,\underline{\sigma},\nu)=\underline{\sigma}'$, $\sigma_1\models P\Rightarrow v_1$, $\sigma_2\models P\Rightarrow v_2$, $\forall x:\underline{\sigma}(x)\sqsubseteq\underline{\sigma}'(ret).\ \sigma_1(x)=\sigma_2(x)$ ならば，$v_1=v_2$.

（証明） $\mathcal{A}[\![\cdot]\!]$ に対する推論規則の適用回数に関する帰納法により示す．
（(a) の証明）(CALL)

$$\tau=\mathcal{A}[\![f(M_1,\ldots,M_n)]\!](F,\underline{\sigma})$$
$$=F[f,\mathcal{A}[\![M_1]\!](F,\underline{\sigma}),\ldots,\mathcal{A}[\![M_n]\!](F,\underline{\sigma})] \quad (4)$$
$$\frac{\sigma_k\models M_i\Rightarrow u_{ki}\ \ (1\le i\le n)\qquad \sigma_k'\models P_f\Rightarrow v_k}{\sigma_k\models f(M_1,\ldots,M_n)\Rightarrow v_k}\ (k=1,2) \quad (5)$$
$$\sigma_k'=\bot_{store}[x_1:=u_{k1}]\cdots[x_n:=u_{kn}]\ (k=1,2) \quad (6)$$
$$\forall x:\underline{\sigma}(x)\sqsubseteq\tau.\ \sigma_1(x)=\sigma_2(x) \quad (7)$$

と仮定する．$F=\mathcal{A}[\![Prog]\!](F)$ であるから，(1) と (4) より，

$$\tau=\mathcal{A}[\![f(M_1,\ldots,M_n)]\!](F,\underline{\sigma})$$
$$=\mathcal{A}[\![P_f]\!](F,\bot_{\underline{store}}[x_1:=\mathcal{A}[\![M_1]\!](F,\underline{\sigma})]$$
$$\cdots[x_n:=\mathcal{A}[\![M_n]\!](F,\underline{\sigma})],\bot)(ret) \quad (8)$$

$\tau_i=\mathcal{A}[\![M_i]\!](F,\underline{\sigma})$ $(1\le i\le n)$ とし，$\tau_i\sqsubseteq\tau$ と仮定すると，(7) より，$\forall x:\underline{\sigma}(x)\sqsubseteq\tau_i.\ \sigma_1(x)=\sigma_2(x)$．帰納法の仮定 (a) より，$u_{1i}=u_{2i}$．すなわち，

$$\forall i(1\le i\le n):\mathcal{A}[\![M_i]\!](F,\underline{\sigma})\sqsubseteq\tau.\ u_{1i}=u_{2i}.$$

よって，(6) より，

$$\forall x:\bot_{\underline{store}}[x_1:=\mathcal{A}[\![M_1]\!](F,\underline{\sigma})]$$
$$\cdots[x_n:=\mathcal{A}[\![M_n]\!](F,\underline{\sigma})](x)$$
$$\sqsubseteq\tau.\ \sigma_1'(x)=\sigma_2'(x) \quad (9)$$

(8)，(5)，(9) と帰納法の仮定 (c) より，$v_1=v_2$.
他の場合は容易．
（(b) の証明）(ASSIGN)

$$\underline{\sigma}'=\mathcal{A}[\![x:=M]\!](F,\underline{\sigma},\nu)$$
$$=\underline{\sigma}[x:=\mathcal{A}[\![M]\!](F,\underline{\sigma})\sqcup\nu] \quad (10)$$
$$\frac{\sigma_k\models M\Rightarrow v_k}{\sigma_k\models x:=M\Rightarrow\sigma_k'}\ (k=1,2) \quad (11)$$
$$\sigma_k'=\sigma[x:=v_k]\ \ (k=1,2) \quad (12)$$
$$\tau=\underline{\sigma}'(y) \quad (13)$$
$$\forall z:\underline{\sigma}(z)\sqsubseteq\tau.\ \sigma_1(z)=\sigma_2(z) \quad (14)$$

と仮定．$x\neq y$ のとき，(10) より $\underline{\sigma}'(y)=\underline{\sigma}(y)$. これと (13)，(14) より，$\sigma_1(y)=\sigma_2(y)$. 更にこれと (12) より，$\sigma_1'(y)=\sigma_2'(y)$. $x=y$ のとき，(10)，(13) より，$\underline{\sigma}'(y)=\mathcal{A}[\![M]\!](F,\underline{\sigma})\sqcup\nu=\tau$. よって，$\mathcal{A}[\![M]\!](F,\underline{\sigma})\sqsubseteq\tau$. これと (14) より，

$$\forall z:\underline{\sigma}(z)\sqsubseteq\mathcal{A}[\![M]\!](F,\underline{\sigma}).\ \sigma_1(z)=\sigma_2(z). \quad (15)$$

(11)，(15) と帰納法の仮定 (a) より，$v_1=v_2$，すなわち，$\sigma_1'(y)=\sigma_2'(y)$.
(WHILE)

$$\underline{\sigma}''=\mathcal{A}[\![\text{while } M \text{ do } P \text{ od}]\!](F,\underline{\sigma},\nu)$$
$$=\mathit{fix}(\lambda X.\underline{\sigma}\sqcup\mathcal{A}[\![P]\!](F,\underline{\sigma}\sqcup X,$$
$$\nu\sqcup\mathcal{A}[\![M]\!](F,\underline{\sigma}\sqcup X))) \quad (16)$$
$$\tau'=\underline{\sigma}''(y) \quad (17)$$
$$\forall x:\underline{\sigma}(x)\sqsubseteq\tau'.\ \sigma_1(x)=\sigma_2(x) \quad (18)$$

と仮定し，

$$\underline{\rho}=\mathcal{A}[\![P]\!](F,\underline{\sigma},\nu\sqcup\tau) \quad (19)$$
$$\tau=\mathcal{A}[\![M]\!](F,\underline{\sigma}) \quad (20)$$

とおく．(WHILE1) の適用回数に関する帰納法を用いる．

$$\underline{\sigma}'=\underline{\sigma}\sqcup\underline{\rho} \quad (21)$$

とおくと，不動点の性質と (16)，(19) より，

$$\underline{\sigma}''=\mathcal{A}[\![\text{while } M \text{ do } P \text{ od}]\!](F,\underline{\sigma}',\nu) \quad (22)$$

(i) $$\frac{\sigma_k \models M \Rightarrow \text{false}}{\sigma_k \models \text{while } M \text{ do } P \text{ od} \Rightarrow \sigma_k} \quad (k = 1, 2)$$

のとき．(16) より $\underline{\sigma} \sqsubseteq \underline{\sigma}''$ だから，(17) より，$\underline{\sigma}(y) \sqsubseteq \underline{\sigma}''(y) = \tau'$．(18) より，$\sigma_1(y) = \sigma_2(y)$．

(ii)

$$\frac{\sigma_1 \models M \Rightarrow \text{true} \quad \sigma_1 \models P \Rightarrow \sigma_1' \quad \sigma_1' \models \text{while } M \text{ do } P \text{ od} \Rightarrow \sigma_1''}{\sigma_1 \models \text{while } M \text{ do } P \text{ od} \Rightarrow \sigma_1''} \tag{23}$$

$$\frac{\sigma_2 \models M \Rightarrow \text{false}}{\sigma_2 \models \text{while } M \text{ do } P \text{ od} \Rightarrow \sigma_2} \tag{24}$$

のとき．$\tau \sqsubseteq \tau'$ と仮定すると，(20)，(18)，(23)，(24) と帰納法の仮定 *(a)* より，true=false となり矛盾．したがって，$\tau \not\sqsubseteq \tau'$．もし $\nu \sqcup \tau \sqsubseteq \underline{\rho}(y)$ なら，(21)，(22)，(17) より，$\tau \sqsubseteq \nu \sqcup \nu \sqsubseteq \underline{\rho}(y) \sqsubseteq \underline{\sigma}''(y) = \tau'$ となり矛盾．よって，

$$\nu \sqcup \tau \not\sqsubseteq \underline{\rho}(y). \tag{25}$$

(19)，$\sigma_1 \models P \Rightarrow \sigma_1'$，(25) と補題 3.3 より，$\sigma_1(y) = \sigma_1'(y)$．

(19)，$\sigma_1' \models \text{while } M \text{ do } P \text{ od} \Rightarrow \sigma_1''$，(25) と補題 3.4 より，$\sigma_1'(y) = \sigma_1''(y)$．一方 (i) の場合と同様にして，$\sigma_1(y) = \sigma_2(y)$．以上より，$\sigma_1''(y) = \sigma_2(y)$．

(iii)

$$\frac{\sigma_k \models M \Rightarrow \text{true} \quad \sigma_k \models P \Rightarrow \sigma_k' \quad \sigma_k' \models \text{while } M \text{ do } P \text{ od} \Rightarrow \sigma_k''}{\sigma_k \models \text{while } M \text{ do } P \text{ od} \Rightarrow \sigma_k''} (k = 1, 2) \tag{26}$$

のとき．$\underline{\rho}(z) \sqsubseteq \tau'$ と仮定すると，(19)，(18)，(26) と帰納法の仮定 (b) より，$\sigma_1'(z) = \sigma_2'(z)$．すなわち，

$$\forall z : \underline{\rho}(z) \sqsubseteq \tau'.\ \sigma_1'(z) = \sigma_2'(z). \tag{27}$$

(22)，(17)，(26), (27) と (WHILE1) の適用回数に関する帰納法の仮定 (b) より，$\sigma_1''(y) = \sigma_2''(y)$．

(CONCAT)

$$\underline{\sigma}'' = \mathcal{A}[\![C; P]\!](F, \underline{\sigma}, \nu) = \mathcal{A}[\![P]\!](F, \underline{\sigma}', \nu) \tag{28}$$

$$\underline{\sigma}' = \mathcal{A}[\![C]\!](F, \underline{\sigma}, \nu) \tag{29}$$

$$\frac{\sigma_k \models C \Rightarrow \sigma_k' \quad \sigma_k' \models P \Rightarrow \sigma_k''}{\sigma_k \models C; P \Rightarrow \sigma_k''} (k = 1, 2) \tag{30}$$

$$\tau = \underline{\sigma}''(y) \tag{31}$$

$$\forall x : \underline{\sigma}(x) \sqsubseteq \tau.\ \sigma_1(x) = \sigma_2(x) \tag{32}$$

と仮定．$\underline{\sigma}'(x) \sqsubseteq \tau$ と仮定すると，(29)，(30)，(32) と帰納法の仮定 (b) より，$\sigma_1'(x) = \sigma_2'(x)$．つまり，

$$\forall x : \underline{\sigma}'(x) \sqsubseteq \tau.\ \sigma_1'(x) = \sigma_2'(x). \tag{33}$$

(28)，(30)，(31)，(33) と帰納法の仮定 (b) より，$\sigma_1''(y) = \sigma_2''(y)$．

(IF) の場合は容易．

(c) の証明は容易なので省略． □

［定理 3.6］ 解析アルゴリズム $\mathcal{A}^*[\![\cdot]\!]$ は健全性を満たす．

（証明） 定理 3.5 (c) より明らか． □

提案アルゴリズムは健全性を満たすので，あるプログラムに対し解析アルゴリズムが「このプログラムは入力情報を出力に漏らさない」と答えたとすると，入力情報が漏れることはない．しかし逆に，提案アルゴリズムが「このプログラムは入力情報を漏洩する可能性がある」と答えたとしても，必ずしもこのプログラムに情報の漏れがあるとは限らない．すなわち，完全性は満たさない．直観的に，解析アルゴリズムが完全性を満たすためには，if 文・while 文の条件部等に現れる式の意味を正確に解析しなければならないが，これは計算不可能な問題であるため，完全性を満たすアルゴリズムは存在し得ない．しかしながら，提案アルゴリズムは，組込み演算の意味定義（解釈）まで自由に与えられるという前提で「解析アルゴリズムが検出した情報フローを実際に引き起こすような組込み演算の解釈とプログラムへの入力が存在する」という意味での弱い完全性 [25] は満たすと考えられる．

3.5 時間計算量

この節では，**3.1** で述べた解析アルゴリズム $\mathcal{A}^*[\![\cdot]\!]$ の時間計算量について述べる．

3.1 のアルゴリズム $\mathcal{A}^*[\![\cdot]\!]$ は，式 (WHILE) 及び式 (2) に不動点関数を含んでおり，単純に繰返し演算を使って実装すると計算量が大きくなる可能性がある．これに対し，以下で述べる方法では，図 1 のような解析対象プログラムに対して図 2 のような有向グラフを作成し，このグラフ内を探索することで，**3.1** のアルゴリズムと同じ解析結果を得る．

図 2 中，薄い色の楕円で囲まれた各部分は，プログラム中の各文に対してアルゴリズム $\mathcal{A}$ を適用した結果を表している．例えば，図 1 の行 6 である代入文 $y := y + g(y, x)$ は $y := y \sqcup g(y, x) \sqcup \nu_2$ に変換され（ν_2 は implicit flow を表すために追加した変数），右辺である項が木の形で表現されている（図 2 の楕

```
    main(x, y) local z
    {
 1:    if y < 0 then
 2:       return 0
       else
 3:       y := 0;
 4:       while y < 10 do
 5:          z := y;
 6:          y := y + g(y, x)
          od;
 7:       return z
       fi
    }

    g(x, y)
    {
 8:    if y <= 0 then
 9:       return 0
       else
10:       return g(x, y-1) + 1
       fi
    }
```

図 1　解析対象プログラム例
Fig. 1　A program to analyze.

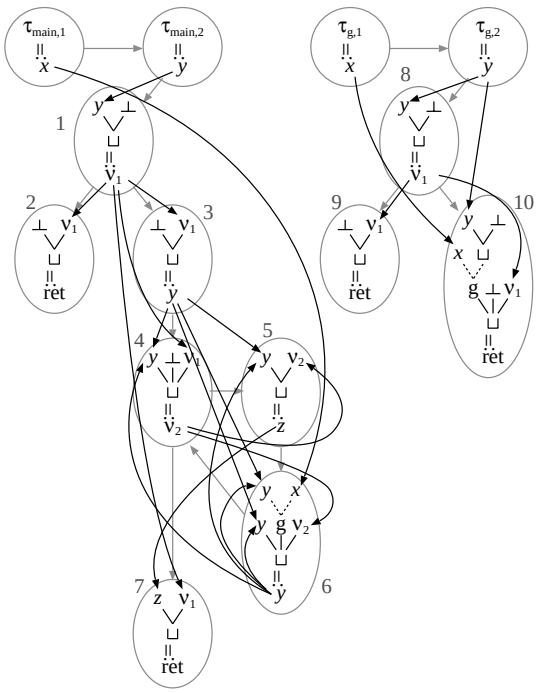

図 2　情報フローを表すグラフ
Fig. 2　An information flow graph.

円 6）．この変換では，**3.1** の場合と違い，変数や関数は項の構成要素としてそのまま変換結果に残す．また図 2 では，各 n 引数関数 f に対して f の実引数の SC を表す定数 $\tau_{f,1},\ldots,\tau_{f,n}$ を用意し，仮引数にそれらを代入する文を加えることが行われている．図 2 の楕円間にある矢印は，変数の定義・使用の関係を表している．例えば楕円 5 の := の右辺に現れる y は，$\mathcal{A}[\![Prog]\!]$ の計算において，楕円 3 または楕円 6 で定義される y の値を使用する可能性がある．このことを，楕円 3 及び楕円 6 の := の左辺から楕円 5 の y への矢印で表している．

3.1 における $\mathcal{A}[\![Prog]\!]$ の計算は，図 2 において，実引数の SC を表す各定数 $\tau_{f,1},\ldots,\tau_{f,n}$ から ret までのパスが存在するかどうか調べることと等価である．すなわち

$$\mathcal{A}[\![Prog]\!](F)[f](\tau_1,\ldots,\tau_n) = \tau_{i_1} \sqcup \cdots \sqcup \tau_{i_m}$$

とするとき（ただし $\{i_1,\ldots,i_m\} \subseteq \{1,\ldots,n\}$．解析アルゴリズムに現れる演算は $\bot$ と $\sqcup$ だけなので，必ずこのように書ける），グラフ上で $\tau_{f,j}$ から ret までのパスが存在することと $j \in \{i_1,\ldots,i_m\}$ であることとが等しい．ここで，関数の SC を表す F は，グラフ中の関数呼出しにあたる頂点とその子との間の辺（図 2 中の破線で書かれた辺）の状態に対応する．例えば $F[g](\tau_1,\tau_2) = \tau_2$ であることと，g の呼出しに対応する頂点（2 個あるうちの両方）とその子との間の辺のうち，第 1 引数に対応する辺は使用不可能（第 1 引数の情報を関数値に流さない），かつ第 2 引数に対応する辺は使用可能（第 2 引数の情報を関数値に流し得る）であることとが対応する．そこで，①まずこれら破線で書かれた辺すべてに使用不可能という印を付け，②関数 f が第 i 引数の情報を関数値に流し得る（$\tau_{f,i}$ から ret へのパスが存在する）とわかった時点で，関数 f の呼出しの第 i 引数に対応する辺を使用可能にしていくことで，$\mathcal{A}^*[\![Prog]\!]$ の計算が可能である．これは，各 ret 頂点から有向辺を逆向きに探索することで行える．ただし実引数の SC を表す定数 $\tau_{f,i}$ に到達した場合は，関数 f の各呼出しの第 i 引数に対応する辺を使用可能に変え，f の呼出しに対応する頂点が既に訪問済みであれば，第 i 引数に対応する頂点を探索開始位置として処理待ちリストに追加すればよい．

このグラフの頂点数を V, 辺数を E とすると，グラフの探索は $O(V+E)$ 時間でできる．解析対象プログラムの記述長を N とすると，$V = O(N)$，$E = O(N^2)$ である．一方，このグラフを作成するために，定義・使用の関係を求める必要がある．これは到達する定義（reaching definition）[2] を計算することで得られる．解析対象プログラムにおける while 文の入れ子の深さの最大値を d とすると，代入文の部分集合に関する集

合演算を $O(N \cdot d)$ 回行うことで，各文での到達する定義が求められる．1 回の集合演算には $O(N)$ の時間が必要であり，$d = O(N)$ であることから，到達する定義の計算時間は合計 $O(N^3)$ となる．ただし，現実のプログラムでは，while 文の入れ子の深さは定数以下と考えてよいことが多く，またビットベクトル [2], [3] などの技術を用いて集合演算を高速化できるため，多くの場合，計算時間は $O(N)$ ないし $O(N^2)$ で済む [3].

4. 拡張モデル

4.1 組込み関数に対する解析関数の拡張

一般に静的にプログラムの情報フローを解析する方法及び，**3.** までで提案している方法では，組込み演算に対して，引数内のすべての情報が演算結果に流れると仮定し，演算結果の SC をすべての引数の SC の最小上界と定義している．この定義では，実際には SC を上げなくてもよい組込み演算に対しても SC を上げることになり，現実的に不自然な点が生じる．そこで，解析方針に応じて任意の組込み演算の出力の SC を設定できるようにモデルを拡張する．

3.1 の解析アルゴリズム $\mathcal{A}$ において，組込み関数 θ に対する定義は以下のとおりであった：

$$\underline{\text{(PRIM)}} \quad \mathcal{A}[\![\theta(M_1, \ldots, M_n)]\!](F, \underline{\sigma}) = \bigsqcup_{1 \le i \le n} \mathcal{A}[\![M_i]\!](F, \underline{\sigma}).$$

すなわち，すべての引数内の情報が $\theta_{\mathcal{I}}$ の結果に流れ込むと仮定していた．しかし，これは特定の演算に対しては強すぎる仮定である．例えば，$\theta_{\mathcal{I}}(x, y) = x$ ならば，明らかに第 2 引数の情報は演算結果に流れない．別の例として，暗号化がある．平文 d と暗号化鍵 k を引数とする暗号化関数を $E(d, k)$ で表す．E が安全であるならば，E の引数 x，y の SC がともに $high$ であっても，$E(x, y)$ の SC は low であると考えることができる．

そこで，組込み関数 θ に対する定義を次のように一般化する：

$$\underline{\text{(PRIM)}} \quad \mathcal{A}[\![\theta(M_1, \ldots, M_n)]\!](F, \underline{\sigma}) = \mathcal{B}[\![\theta]\!](\mathcal{A}[\![M_1]\!](F, \underline{\sigma}), \ldots, \mathcal{A}[\![M_n]\!](F, \underline{\sigma}))$$

ただし，$\mathcal{B}[\![\theta]\!]$ は sc 上のある単調な関数である：

$$\mathcal{B}[\![\theta]\!] : sc \times \cdots \times sc \to sc.$$

本節の拡張は，解析方針に応じ，任意の組込み関数 θ に対して $\mathcal{B}[\![\theta]\!]$ を定義できるようにすることである．ここで，$\mathcal{B}[\![\theta]\!]$ を $\mathcal{B}[\![\theta]\!](\tau_1, \ldots, \tau_n) = \bigsqcup_{1 \le i \le n} \tau_i$ と定義すると，もとの $\mathcal{A}$ と同じ定義になる．

以下に，拡張モデルを用いることにより，組込み関数の解析結果を現実世界の組込み関数の振舞いに近づけられる例を三つ示す．

［例 4.1］（nonstrict 関数） $\theta_{\mathcal{I}}(x, y) = x$ という組込み関数がある場合，組込み関数の結果 x から引数 y の情報を得ることはできないので，$\mathcal{B}[\![\theta]\!](\tau_1, \tau_2) = \tau_1$ と定義することができる． □

［例 4.2］（暗号化関数） E を平文と暗号化鍵を引数とし，暗号化を行う関数とする．不正な攻撃者が暗号文に対してどのような操作を行っても，平文及び暗号化鍵に含まれる情報が得られないと仮定すると，$\mathcal{B}[\![E]\!](high, high) = low$ と定義することができる． □

例 4.2 は，解析方針として「不正な攻撃者は復号化鍵を用いない限り，プログラムの出力に対してどのような操作を行っても，平文及び暗号化鍵に関する情報を得ることができない」かどうかを解析するときに有意である．

［例 4.3］（平均値計算関数） 平均値計算関数 Ave を i 個の整数 $n_1, \ldots, n_i$ を引数として，平均値を計算する関数とする．平均値からどのような操作を行っても，引数に含まれる情報が得られないと仮定すると，$\mathcal{B}[\![Ave]\!](\tau_1, \ldots, \tau_i) = low$ と定義することができる． □

平均値計算関数に対して $\mathcal{B}[\![Ave]\!](\tau_1, \ldots, \tau_i) = low$ と定義するのが妥当なのは，Ave 関数の引数の数 i が十分大きい場合である．例えば，$i = 1$ のとき $Ave_{\mathcal{I}}(n_1) = n_1$ であるから，「平均値から，引数に含まれる情報が得られない」という仮定は成り立たず，$\mathcal{B}[\![Ave]\!](\tau_1) = low$ と定義するのは明らかに不自然である．いくつ以上の引数が与えられれば「平均値に対してどのような操作を行っても，引数に含まれる情報が得られない」と仮定してよいかは，解析方針に依存する．解析対象プログラム内で Ave 関数に多数の引数が与えられるかどうかがわからない場合，$\mathcal{B}[\![Ave]\!](\tau_1, \ldots, \tau_i) = low$ と定義するべきではない．

さて，このように一般化されたアルゴリズムは，定義 3.1 の意味での健全性を満たさない．$\mathcal{B}[\![E]\!](\tau_1, \tau_2) = low$ と定義したと仮定し，次のプログラムを考える．

$$Prog = \{ main(x, y) \ \{ \text{ return } E(x, y) \ \} \}$$

ここで，$\mathcal{A}^*[\![Prog]\!][main](high, high) = low$ となるが，d_1，$d_2 (d_1 \neq d_2)$ 及び k に対して，$E_{\mathcal{I}}(d_1, k) \neq E_{\mathcal{I}}(d_2, k)$ となる．したがって，$\mathcal{A}^*[\![\cdot]\!]$ は健全ではな

い．直観的に，$E(x,y)$ の SC を low であると解析することは，暗号文から引数 x，y 内の情報を得られないことを意味する．すなわち，暗号文 $E_{\mathcal{I}}(d_1,k)$ 及び $E_{\mathcal{I}}(d_2,k)$ からそれぞれの引数内の情報を区別できない．この区別不能性を定義するために，次の関係を用いる．

型 val 上の同値関係 R が以下の条件を満たすとき，R を合同関係と呼ぶ：

各 n 引数組込み関数 θ に対して，もし $c_i\ R\ c_i'(1 \le i \le n)$ ならば，$\theta_{\mathcal{I}}(c_1,\ldots,c_n)\ R\ \theta_{\mathcal{I}}(c_1',\ldots,c_n')$.

以下，一つの合同関係〜が与えられていると仮定する．型 val の値 v, v' に対して，もし v〜v' ならば v と v' は区別不能であるという．定義から，もし v_i と v_i' $(1 \le i \le n)$ が区別不能ならば任意の組込み関数 θ に対して，$\theta_{\mathcal{I}}(c_1,\ldots,c_n)$ と $\theta_{\mathcal{I}}(c_1',\ldots,c_n')$ も区別不能である．このことは，v と v' が一度区別不能になると，どのような操作を行ったとしても v と v' を区別するような情報を得られないことを意味する．次に，$\mathcal{B}[\![\cdot]\!]$ に関する次の条件を考える．

［条件 4.4］ n 引数関数 θ に対して $\mathcal{B}[\![\theta]\!](\tau_1,\ldots,\tau_n) = \tau$ と仮定し，c_i, c_i' の型を val 型とする $(1 \le i \le n)$．このとき，$\tau_j \sqsubseteq \tau$ を満たすようなすべての j $(1 \le j \le n)$ に対して c_j〜c_j' が成り立つならば，$\theta_{\mathcal{I}}(c_1,\ldots,c_n)$〜$\theta_{\mathcal{I}}(c_1',\ldots,c_n')$. □

この条件は以下のことを意味する：

$\mathcal{B}[\![\theta]\!](\tau_1,\ldots,\tau_n) = \tau$ とする．θ の引数が $c_1,\ldots,c_n$ から $c_1',\ldots,c_n'$ に変化したと仮定する．このとき，$\tau_j \sqsubseteq \tau$ を満たす第 j 引数 c_j 及び c_j' が区別不能ならば，$\theta_{\mathcal{I}}(c_1,\ldots,c_n)$ と $\theta_{\mathcal{I}}(c_1',\ldots,c_n')$ も区別不能である．

例 4.1 において，条件 4.4 は任意の値 c_1, c_1'，c_2, c_2' に対して，$\theta_{\mathcal{I}}(c_1,c_2)$〜$\theta_{\mathcal{I}}(c_1',c_2')$ であることを要求する．c_1〜c_1' ならば $\theta_{\mathcal{I}}(c_1,c_2) = c_1$〜$c_1' = \theta_{\mathcal{I}}(c_1',c_2')$ なので，任意の合同関係〜は条件 4.4 を満たす．

例 4.2 では，条件 4.4 は任意の平文 d，d' 及び任意の鍵 k, k' に対して，$E_{\mathcal{I}}(d,k)$〜$E_{\mathcal{I}}(d',k')$ であることを要求する．これは「暗号文から平文内及び鍵に関する情報を得られない」ことを意味する．

例 4.3 では，条件 4.4 は任意の整数 $n_1,\ldots,n_i$，$n_1',\ldots,n_i'$ に対して，$Ave_{\mathcal{I}}(n_1,\ldots,n_i)$〜$Ave_{\mathcal{I}}(n_1',\ldots,n_i')$ であることを要求する．これは「平均値からどのような操作を行っても，引数に含まれる情報が得られない」ことを意味する．

区別不能性を用いて，以下のように健全性を定義する．

［定義 4.5］（健全性） 〜を合同関係とする．次の条件が成り立つとき，解析アルゴリズム $\mathcal{A}^*[\![\cdot]\!]$ は〜に関して健全であるという：

$$\mathcal{A}^*[\![Prog]\!][main](\tau_1,\ldots,\tau_n) = \tau,$$
$$\bot_{store} \models main(v_1,\ldots,v_n) \Rightarrow v,$$
$$\bot_{store} \models main(v_1',\ldots,v_n') \Rightarrow v',$$
$$\forall i\,(1 \le i \le n) : \tau_i \sqsubseteq \tau.\, v_i \sim v_i'$$

ならば v〜v' が成り立つ． □

［定理 4.6］ もし，条件 4.4 が成り立つならば，一般化された解析アルゴリズム $\mathcal{A}^*[\![\cdot]\!]$ は定義 4.5 の意味で健全である． □

定理 4.6 の証明は，定理 3.6 の証明と同様に行うことができる．

4.2 拡張モデルの適用例

以下のような，正規のユーザが指定するデータに対してのみ電子署名を付けるプログラムを考える．

```
main(id, pass, d) {
    if (verify(id, pass)) then
        return sign(d)
    else
        return d
    fi
}
```

このプログラムの入力は，ユーザ ID（id），ユーザパスワード（$pass$），及び電子署名を付けてほしいデータ（d）である．ここで，$verify$ はユーザ ID とユーザパスワードを引数とし，正規のユーザと確認できれば “true”，そうでなければ “false” を返す組込み関数とし，$sign$ はデータを引数とし，そのデータに署名を付け，署名付きのデータを返す組込み関数とする．ここで，ユーザ ID 及びユーザパスワードの SC は最高値の $\top$，入力されるデータの SC は τ であると仮定する．拡張モデルを用いずに（$\mathcal{A}[\![verify]\!](\tau_1,\tau_2) = \tau_1 \sqcup \tau_2$）このプログラムの解析を行うと，implicit flow により if 文の条件部の情報が return 文に流れ「出力となるデータの SC は $\top$ である」と解析する．この結果は，もしデータ d の所有者であるユーザが SC τ のデータしか読めないとすると，自分のデータに電子署名を付けてもらうとそのデータが読めなくなる可能性があることになり不自然な解析結果である．

ここで自然な仮定として $verify$ 関数の出力

(true/false) に対してどのような操作を行っても，ユーザ ID 及びユーザパスワードの情報は得られないと考え，$\mathcal{B}[\![verify]\!](\top,\top) = \bot$ と定義する．そうすると，このプログラムの解析結果は「出力となるデータの SC は τ である」となる．これは，入力されたデータと同じ SC の電子署名付きデータが出力されることを表しており，自然な解析結果である．また，この解析は，(条件 4.4)「任意の c_1，c_1'，c_2，c_2' に対して，$verify_{\mathcal{I}}(c_1,c_2) \sim verify_{\mathcal{I}}(c_1',c_2')$」すなわち，「*verify* 関数の出力 (true/false) に対してどのような操作を行っても，ユーザ ID 及びユーザパスワードの情報は得られない」という仮定のもとで，出力である電子署名付きデータに，ユーザ ID 及びユーザパスワードの情報が漏れないことを保証している．

5. むすび

本論文では，一般に再帰を含む手続き型プログラムの情報フローを静的に解析するアルゴリズムを提案し，提案アルゴリズムの健全性を証明した．また，この解析アルゴリズムがプログラムの記述長に対する多項式時間で実行可能であることを示した．更に，任意の組込み演算に対する定義を一般化し，解析方針に応じて，任意の組込み演算の結果の SC を設定できるようにモデルを拡張した．最後に拡張モデルに対しても健全性を証明した．[26] では，提案アルゴリズムの実装が行われており，チケット予約システム等の，いくつかの例プログラムについて解析が行われている．この試作システムでは，複数の出力をもつプログラムの解析も行えるように提案アルゴリズムが拡張されており，大域変数を含むプログラムも解析できる．ただし，SC は $\{high, low\}$ の 2 値のみを扱える ($low \sqsubseteq high$)．

例プログラムに対する解析時間を表 1 に示す (CPU: Pentium4 1.5 GHz, 主記憶：512MB)．

表 1 のチケット予約システムには，クレジットカード番号の認証モジュールが含まれており，入力となるクレジットカード番号の SC を *high* と仮定した．その結果，36 ある出力文のうち 13 の出力で SC が *high* となり得ると解析された (SC が *high* となる 13 の出力のうち，七つは認証モジュール内に存在)．次に，拡張モデルの適用例として，認証モジュールを組込み演算であると考え，演算結果の SC は *low* であると仮定すると，出力のうち SC が *high* となるのは，認証モジュール内の七つの出力のみで，認証モジュール外の残り六つの出力の SC は *low* となった．これは，認証モジュールから SC が *high* のデータが漏れないかぎり，出力にはクレジットカード番号は漏れないことを保証している．また，解析アルゴリズムが安全でない (SC が *high* となり得る) と答えた出力に関して，実際に実行時のプログラムの流れを追ってみたところ，これらすべての出力に対して，クレジットカード番号の情報が使われていることが確認できた．

表 1 解析時間
Table 1 Analysis time.

プログラム	行数	平均解析時間 (s)
チケット予約システム	419	0.040
ソートアルゴリズム	825	0.073
ライブラリ群	2471	1.261

ポインタを含む言語及びオブジェクト指向言語に対しても解析が行えるようにアルゴリズムを拡張することは今後の課題である．

謝辞 共同研究を通して日ごろより御討論頂き，また，解析時間に関するデータを提供して頂きました，大阪大学大学院情報科学研究科の大畑文明氏，横森励士氏に深謝致します．

文献

[1] M. Abadi, C. Fournet, and G. Gonthier, "Secure communications processing for distributed languages," 1999 IEEE Symp. on Security and Privacy, pp.74–88, 1999.

[2] A.V. Aho, R. Sethi, and J.D. Ullman, Compilers, Principles, Techniques, and Tools, Addison-Wesley, 1986.

[3] A.W. Appel, Modern Compiler Implementation in Java, Cambridge University Press, 1998.

[4] J. Banâtre, C. Bryce, and D. Le Métayer, "Compile-time detection of information flow in sequential programs," 3rd ESORICS, LNCS 875, pp.55–73, 1994.

[5] E. Bertino and H. Weigand, "An approach to authorization modeling in object-oriented database systems," Data & Knowledge Engineering, vol.12, pp.1–29, 1994.

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

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

[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] Y. Ishihara, T. Morita, and M. Ito, "The security problem against inference attacks on object-oriented databases," in Research Advances in Database and

Information Systems Security, pp.303–316, Kluwer Academic, 1999.
[10] 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.
[11] X. Leroy and F. Rouaix, “Security properties of typed applets,” 25th ACM Symp. on Principles of Programming Languages, pp.391–403, 1998.
[12] J. Mitchell, Foundations of Programming Languages, MIT Press, 1996.
[13] A.C. Myers, “JFLOW: Practical mostly-static information flow control,” 26th ACM Symp. on Principles of Progmming Languages, pp.228–241, 1999.
[14] A.C. Myers and B. Liskov, “Complete, safe information flow with decentralized labels,” 1998 IEEE Symp. on Security and Privacy, pp.186–197, 1998.
[15] 中田育男，コンパイラ，産業図書，1981.
[16] N. Nitta, Y. Takata, and H. Seki, “Security verification of programs with stack inspection,” 6th ACM Symp. on Access Control Models and Technologies, pp.31–40, 2001.
[17] P. Ørbæk, “Can you trust your data?” TAPSOFT ’95, LNCS 915, pp.575–589.
[18] G. Purnul, “Database security,” in Advances in Computers, ed. M. Yovits, vol.38, pp.1–72, Academic Press, 1994.
[19] A.W. Roscoe, J.C. Woodcock, and L. Wulf, “Noninterference through determinism,” 3rd ESORICS, LNCS 875, pp.33–53, 1994.
[20] P.Y.A. Ryan and S.A. Shneider, “Process algebra and non-interference,” 12th IEEE Computer Security Foundations Workshop, pp.214–227, 1999.
[21] P. Samarati, E. Bertino, A. Ciampichetti, and S. Jajodia, “Information flow control in object-oriented systems,” IEEE Trans. Knowledge and Data Engneering, vol.9, no.4, pp.524–538, 1997.
[22] G. Smith and D. Volpano, “Secure information flow in a muti-threaded imperative language,” 25th ACM Symp. on Principles of Programming Languages, pp.355–364, 1998.
[23] K. Tajima, “Static detection of security flaws in object-oriented databases,” 1996 ACM SIGMOD Intl. Conf. on Management of Data, pp.341–352, 1996.
[24] D. Volpano and G. Smith, “A type-based approach to program security,” TAPSOFT ’97, LNCS 1214, pp.607–621.
[25] R. バード（著），土井範久（訳），プログラム理論入門，情報処理シリーズ 3, 2.3 節，培風館.
[26] 横森励士，“オブジェクト指向プログラムにおけるセキュリティ解析アルゴリズムの提案と実現,” 修士学位論文，大阪大学，2001.

（平成 13 年 9 月 6 日受付，14 年 4 月 5 日再受付）

國信　茂太　（学生員）

平 11 同志社大・電子卒．平 13 奈良先端科学技術大学院大学情報科学研究科博士前期課程了．同大学院博士後期課程に在学中．

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．

井上　克郎　（正員）

昭 54 阪大・基礎工・情報卒．昭 59 同大大学院博士課程了．同年同大・基礎工・情報・助手．昭 59〜61 ハワイ大マノア校・情報工学科・助教授．平元阪大・基礎工・情報・講師．平 3 同学科・助教授．平 7 同学科・教授．工博．ソフトウェア工学の研究に従事．