情報量に基づく非機密化プリミティブの記述位置候補の順位付け

Entropy-based Ranking Method for Declassifiers Placement

桑原 寛明*　國枝 義敏†

Summary. This paper proposes a method for ranking the candidates of declassifiers placement. Though the declassification is a useful method to resolve illegal information flow, there may be many candidates of declassifiers placement. Our proposed method calculates the amount of entropy of declassified constructs for each candidate and ranks the candidates in ascending order of calculated entropy. This paper shows how to calculate the Shannon's entropy of each expression in the program and a simple example of our ranking method.

1　はじめに

型検査に基づく情報流解析は，機密情報がプログラムの外部に漏洩しないことを検査する手法である [1] [2] [3] [4]．型としてデータの機密度を利用し，機密度の低いデータが機密度の高いデータに依存しないことを表す非干渉性を満たすように型システムが構築される．非干渉性は，機密データ自体の漏洩だけでなく機密データを推測できる情報の漏洩も存在しないことを表すよい性質である．しかし，外部から観察可能な動作を機密データに基づいて変更することが禁止されるため，非干渉性を満たしながら実用的なプログラムを作成することは困難である．

この問題に対し，本来は不正であるとみなされる機密度の高いデータから低いデータへの情報流を開発者が明示した場合に限って認める非機密化 (Declassification) と呼ばれるアプローチが提案されている [5] [6] [7]．例えば，Sabelfeld らは式 e の機密度を強制的に η とみなすことを意味する式レベルのプリミティブ `declassify`(e, η) を持つプログラミング言語とその型システムを提案している [6]．開発者は，`declassify` 式を適切に記述することで，機密度の高いデータから低いデータへの情報流を容認していることを型システムに示し，型システムは開発者の意図を踏まえて型検査を行う．非機密化は非干渉性の厳しい制約を緩和しながら安全なプログラムを構築するための有益な手法である．

プログラムに含まれる不正な情報流は一通りには限られず，複数の不正な情報流すべてに対処する必要がある．Sabelfeld らの `declassify` 式のような非機密化プリミティブ (Declassifier) を手作業で適切に記述することは容易ではないが，非機密化プリミティブの適切な記述位置を列挙する手法が提案されている [8] [9] [10]．これらの手法を用いると，不正な情報流を解消するために非機密化プリミティブを記述すべきプログラム中の位置を容易に知ることができる．著者らが提案する手法 [10] では，情報流解析のための型システムを制約充足問題に帰着させ，充足不能な制約集合の Minimal Correction Subset(MCS) に含まれる各制約が由来するプログラムの構成要素を非機密化プリミティブの記述位置とする．MCS とは，取り除くことで元の制約集合が充足可能となる要素の極小部分集合を指す．

一般に，充足不能な制約集合の MCS は複数存在するため，非機密化プリミティブの記述位置の組み合わせは一意ではない．本稿では，非機密化プリミティブの記述位置の組み合わせが複数存在する場合に，それらを順位付ける手法を提案する．非機密化プリミティブの記述は不正な情報流の容認を意味するが，そのような情報流は可能な限り少ない方が望ましい．そこで，非機密化プリミティブを記述するこ

*Hiroaki Kuwabara, 立命館大学情報理工学部

†Yoshitoshi Kunieda, 立命館大学情報理工学部

$$
\begin{array}{rcl}
\eta & ::= & L \mid H \\
B & ::= & \{\overline{\eta\ x;}\ \overline{S;}\} \\
S & ::= & x = e^l \mid \text{if } (b^l)\ B \text{ else } B \mid \text{while } (b^l)\ B \\
e & ::= & x \mid n \mid \mathrm{op}(e^l, e^l) \\
b & ::= & \neg b^l \mid \mathrm{cmp}(e^l, e^l)
\end{array}
$$

図 1　対象言語の文法

とで記述位置のプログラム構成要素の情報が流出するとみなし，プログラム構成要素が持つ情報量に基づいて非機密化プリミティブの記述位置の各組み合わせについて発生する不正な情報流の大きさを求める．非機密化プリミティブの記述により発生する不正な情報流の大きさの合計が小さくなる順に記述位置の組み合わせを順位付ける．本稿では，Shannon のエントロピーに基づいて順位付ける方法を示す．

2　非機密化プリミティブの記述位置候補の順位付け

非機密化プリミティブの記述位置候補は [10] の手法に基づいて求める．本稿における対象言語の文法を図 1 に示す．η は機密度であり，変数の型として用いる．簡単のために機密度は $L \sqsubset H$ を満たす L, H の 2 段階とする．B はブロックであり，最外のブロックをプログラムとみなす．$\overline{A}$ は長さ 0 以上の有限リストの略記である．ブロックの先頭でローカル変数を宣言できる．S は文，e は整数値の式，b は真偽値の式である．以下，e を整数式，b を真偽式と呼び，双方を区別しない場合は単に式と呼ぶ．x は変数，n は整数定数であり，op は適当な二項演算，cmp は適当な比較演算を表す．非機密化プリミティブとして式に対して記述する `declassify` 式を用いる．l は `declassify` 式を記述可能な式の位置を表すラベルであり，プログラム中のラベルはすべて異なるとする．以下ではラベルが重要でない場合は省略する．

任意のプログラムに対し制約集合を生成するアルゴリズムを図 2 に示す．図中の κ や添字付きの κ_i, κ_e などは機密度を表す変数である．図中の機密度変数 κ はすべてフレッシュである．つまり，それぞれの文や式に対して新しい機密度変数が用意される．$\Delta \Vdash e : \kappa \parallel \mathsf{C}$ は型環境 Δ のもとで式 e に対して生成される制約集合が C であることを表す．文やブロックについても同様である．一部の制約には由来する式を示すラベルを付ける．式 e^l に関する制約に $\kappa_1 \sqsubseteq^l \kappa_2$ のようにラベル l を付けることで，この制約がラベル l が付けられた式 e^l に由来することがわかる．

生成された制約集合が充足不能であれば，対象のプログラムには不正な情報流が存在する．`declassify` 式を適切に記述することで解決できるが，一般に不正な情報流は複数存在し，それぞれについて複数のプログラム構成要素が関係するため，`declassify` 式の記述位置は一意に決まらない．そこで，充足不能な制約集合からラベル付き制約のみを含む MCS をすべて抽出し，抽出された各 MCS について，含まれる各制約に付けられたラベルに対応する式の集合をその MCS を解消するための `declassify` 式の記述位置候補とする．充足不能な制約集合には MCS が存在し，MCS が取り除かれた制約集合は充足可能であることから，MCS に含まれる各制約が由来する式に `declassify` 式を記述すれば充足不能な制約集合は生成されない．MCS に含まれる各制約が由来する式の組み合わせが `declassify` 式の記述位置の組み合わせに対応する．充足不能な制約集合に対し一般に MCS は複数存在するため，`declassify` 式の記述によって解消可能な各 MCS について `declassify` 式の記述位置を求め，それぞれを候補とする．

[10] の手法では, `declassify` 式の記述位置は式に限定される．そこで, `declassify` 式の記述位置の式が持つ情報量に基づいて記述位置候補を順位付ける手法を提案する．実際には，MCS と記述位置候補は一対一に対応するため MCS を順位付ける．

$\mathcal{L}$ をプログラム中のラベルの集合として，プログラム中のラベルと情報量を対応

$$\frac{\Delta, \overline{x:\eta} \Vdash S_i : \kappa_i \parallel \mathsf{C}_i \quad i \in \{1,\ldots,n\}}{\Delta \Vdash \{\overline{\eta\, x;}\ S_1; \ldots S_n;\} : \kappa \parallel \bigcup_i (\mathsf{C}_i \cup \{\kappa \sqsubseteq \kappa_i\})}\ \text{[C-BLOCK]}$$

$$\frac{\Delta \Vdash e : \kappa_e \parallel \mathsf{C_e} \quad \eta_x = \Delta(x)}{\Delta \Vdash x = e^l : \kappa \parallel \mathsf{C_e} \cup \{\kappa_e \sqsubseteq^l \eta_x, \kappa \sqsubseteq \eta_x\}}\ \text{[C-ASSIGN]}$$

$$\frac{\Delta \Vdash b : \kappa_b \parallel \mathsf{C_b} \quad \Delta \Vdash B_t : \kappa_t \parallel \mathsf{C_t} \quad \Delta \Vdash B_f : \kappa_f \parallel \mathsf{C_f}}{\Delta \Vdash \text{if } (b^l)\ B_t \text{ else } B_f : \kappa \parallel \mathsf{C_b} \cup \mathsf{C_t} \cup \mathsf{C_f} \cup \{\kappa_b \sqsubseteq^l \kappa, \kappa \sqsubseteq \kappa_t, \kappa \sqsubseteq \kappa_f\}}\ \text{[C-IF]}$$

$$\frac{\Delta \Vdash b : \kappa_b \parallel \mathsf{C_b} \quad \Delta \Vdash B : \kappa_B \parallel \mathsf{C_B}}{\Delta \Vdash \text{while } (b^l)\ B : \kappa \parallel \mathsf{C_b} \cup \mathsf{C_B} \cup \{\kappa_b \sqsubseteq^l \kappa, \kappa \sqsubseteq \kappa_B\}}\ \text{[C-WHILE]}$$

$$\frac{\eta = \Delta(x)}{\Delta \Vdash x : \eta \parallel \emptyset}\ \text{[C-VAR]} \qquad \frac{}{\Delta \Vdash n : L \parallel \emptyset}\ \text{[C-CONST]}$$

$$\frac{\Delta \Vdash e_1 : \kappa_1 \parallel \mathsf{C_1} \quad \Delta \Vdash e_2 : \kappa_2 \parallel \mathsf{C_2}}{\Delta \Vdash \text{op}(e_1^{l_1}, e_2^{l_2}) : \kappa \parallel \mathsf{C_1} \cup \mathsf{C_2} \cup \{\kappa_1 \sqsubseteq^{l_1} \kappa, \kappa_2 \sqsubseteq^{l_2} \kappa\}}\ \text{[C-OP]}$$

$$\frac{\Delta \Vdash b : \kappa_b \parallel \mathsf{C_b}}{\Delta \Vdash \neg b^l : \kappa \parallel \mathsf{C_b} \cup \{\kappa_b \sqsubseteq^l \kappa\}}\ \text{[C-NOT]}$$

$$\frac{\Delta \Vdash e_1 : \kappa_1 \parallel \mathsf{C_1} \quad \Delta \Vdash e_2 : \kappa_2 \parallel \mathsf{C_2}}{\Delta \Vdash \text{cmp}(e_1^{l_1}, e_2^{l_2}) : \kappa \parallel \mathsf{C_1} \cup \mathsf{C_2} \cup \{\kappa_1 \sqsubseteq^{l_1} \kappa, \kappa_2 \sqsubseteq^{l_2} \kappa\}}\ \text{[C-CMP]}$$

図 2　制約集合生成アルゴリズム

付ける関数 $E : \mathcal{L} \to \mathbb{R}^+$ が存在する，すなわちプログラム中のそれぞれの式が持つ情報量は既知であると仮定する．ここで，式が持つ情報量の具体的な求め方は問わない．ラベル付き制約のみを含む MCS を M，M に含まれる各制約に付けられたラベルの集合を $\mathcal{L}_M$ とすると，M に基づいて `declassify` 式が記述される各式の情報量の合計は $\sum_{l \in \mathcal{L}_M} E(l)$ である．この値が小さい順に MCS を順位付ける．

3　式の情報量

実際に MCS を順位付けるためには，`declassify` 式が記述され得る式の具体的な情報量を求める必要がある．情報量に着目した情報流解析である量的情報流については様々な研究 [11] [12] [13] [14] [15] [16] が行われており，本研究では量的情報流における情報量を応用する．量的情報流における情報量として，未知の値に関する情報量の期待値である Shannon のエントロピー，未知の値の特定に必要な問い合わせ (guess) の平均回数である guessing エントロピー，未知の値を 1 回の問い合わせで特定できる確率に基づく min-entropy などが利用される．本稿では Shannon のエントロピー（以下，単にエントロピーという）を利用する．

プログラム中の式のエントロピーを求めるためには，式を確率変数であるとみなし，その確率分布（取り得る値とその確率）が得られればよい．例えば，式 e の取り得る値の集合が $\{v_1, \ldots, v_n\}$，値が v_i となる確率が p_i であるとすると，e のエントロピーは $\mathcal{H} = -\sum_{i=1}^{n} p_i \log p_i$ のように求められる．

図 1 の言語の確率分布に基づく意味を図 3 に示す．各変数の初期値に関する確率分布を与えてこの意味に従ってプログラムを実行すれば各式の確率分布が得られる．

$$\frac{\Gamma_i \vdash S_i \rhd \Gamma'_i \quad i \in \{1,\ldots,n\} \quad \Gamma_i = \begin{cases} \Gamma, \overline{x : \mu} & \text{if } i = 1 \\ \Gamma'_{i-1} & \text{otherwise} \end{cases}}{\Gamma \vdash \{\overline{\eta\ x;}\ S_1; \ldots S_n;\} \rhd \Gamma'_n \downarrow \overline{x}}\ \text{[D-BLOCK]}$$

$$\frac{\Gamma \vdash b : \mu \mid \Gamma_t; \Gamma_f \quad \Gamma_t \vdash B_t \rhd \Gamma'_t \quad \Gamma_f \vdash B_f \rhd \Gamma'_f}{\Gamma \vdash \text{if } (b)\ B_t \text{ else } B_f \rhd \mu(\mathtt{tt}) \cdot \Gamma'_t + \mu(\mathtt{ff}) \cdot \Gamma'_f}\ \text{[D-IF]}$$

$$\frac{\Gamma \vdash b : \mu \mid \Gamma_t; \Gamma_f \quad \mu(\mathtt{ff}) < 1 \quad \Gamma_t \vdash B \rhd \Gamma' \quad \Gamma' \vdash \text{while } (b)\ B \rhd \Gamma''}{\Gamma \vdash \text{while } (b)\ B \rhd \mu(\mathtt{tt}) \cdot \Gamma'' + \mu(\mathtt{ff}) \cdot \Gamma_f}\ \text{[D-WHILE-T]}$$

$$\frac{\Gamma \vdash b : \mu \mid \Gamma_t; \Gamma_f \quad \mu(\mathtt{ff}) = 1}{\Gamma \vdash \text{while } (b)\ B \rhd \Gamma_f}\ \text{[D-WHILE-F]} \qquad \frac{\Gamma \vdash e : \mu}{\Gamma \vdash x = e \rhd \Gamma[x : \mu]}\ \text{[D-ASSIGN]}$$

$$\frac{\mu = \Gamma(x)}{\Gamma \vdash x : \mu}\ \text{[D-VAR]} \qquad \frac{}{\Gamma \vdash n : [n \mapsto 1]}\ \text{[D-CONST]}$$

$$\frac{\Gamma \vdash e_1 : \mu_1 \quad \Gamma \vdash e_2 : \mu_2}{\Gamma \vdash \mathrm{op}(e_1, e_2) : \mu}\ \text{[D-OP]}$$

where $\mu(v) = \sum_{(x,y) \in \mathrm{dom}(\mu_1) \times \mathrm{dom}(\mu_2), [\![\mathrm{op}]\!](x,y) = v} \mu_1(x)\mu_2(y)$

$$\frac{\Gamma \vdash b : \mu \mid \Gamma_t; \Gamma_f}{\Gamma \vdash \neg b : [\mathtt{tt} \mapsto \mu(\mathtt{ff}), \mathtt{ff} \mapsto \mu(\mathtt{tt})] \mid \Gamma_f; \Gamma_t}\ \text{[D-NOT]}$$

$$\frac{\Gamma \vdash e_1 : \mu_1 \quad \Gamma \vdash e_2 : \mu_2}{\Gamma \vdash \mathrm{cmp}(e_1, e_2) : \mu \mid \Gamma_t; \Gamma_f}\ \text{[D-CMP]}$$

where $\mu(v) = \sum_{(x,y) \in \mathrm{dom}(\mu_1) \times \mathrm{dom}(\mu_2), [\![\mathrm{cmp}]\!](x,y) = v} \mu_1(x)\mu_2(y)$,
$b = \mathrm{cmp}(e_1, e_2)$,

$$\Gamma_t(x) = \begin{cases} \mu(\mathtt{tt})^{-1} \cdot (\Gamma(x) \uparrow tt(b,x)) & \text{if } x \in d(b) \wedge \mu(\mathtt{tt}) \neq 0 \\ \Gamma(x) & \text{otherwise} \end{cases},$$

$$\Gamma_f(x) = \begin{cases} \mu(\mathtt{ff})^{-1} \cdot (\Gamma(x) \uparrow ff(b,x)) & \text{if } x \in d(b) \wedge \mu(\mathtt{ff}) \neq 0 \\ \Gamma(x) & \text{otherwise} \end{cases}$$

図 3　対象言語の意味

ただし，プログラムが停止しない初期値の組み合わせが含まれている場合は得られない．初期値に関する確率分布の作り方は任意であり，開発者が手作業で作成する他に，プログラム実行時の入力をログに記録して作成することも可能である．

確率分布 $\mu = [v_1 \mapsto p_1, \ldots, v_n \mapsto p_n]$ は，値が v_i となる確率が p_i であることを示し，$\mathrm{dom}(\mu) = \{v_1, \ldots, v_n\}$ および $\mu(v_i) = p_i$ とする．$v \notin \mathrm{dom}(\mu)$ ならば $\mu(v) = 0$ である．実数 r に対し $r \cdot \mu = [v_1 \mapsto r \cdot p_1, \ldots, v_n \mapsto r \cdot p_n]$ とするが，$r \cdot p_i$ が 1 より大きい場合は 1 とする．$\mu_1 + \mu_2$ を $(\mu_1 + \mu_2)(v) = \mu_1(v) + \mu_2(v)$ のように定義する．値の集合 V に対し，$\mu \uparrow V$ は μ から V に含まれない値を取り除いた確率分布を表す．$\Gamma = x_1 : \mu_1, \ldots, x_m : \mu_m$ はプログラム中の変数と確率分布の対応を記録する環境であり，変数 x_i の確率分布が μ_i であることを示す．この時，$\mathrm{dom}(\Gamma) = \{x_1, \ldots, x_m\}$ および $\Gamma(x_i) = \mu_i$ であり，$x \notin \mathrm{dom}(\Gamma)$ に対し $\Gamma(x) = []$ とする．$[]$ は空の確率分布であり，任意の値 v に対して $[](v) = 0$ である．実数 r に対し $r \cdot \Gamma = x_1 : r \cdot \mu_1, \ldots, x_m : r \cdot \mu_m$ とする．$\Gamma_1 + \Gamma_2$ を $(\Gamma_1 + \Gamma_2)(x) = \Gamma_1(x) + \Gamma_2(x)$ のように定義する．Γ に含まれる x の確率分布の更新を $\Gamma[x : \mu]$ のように表し，$\Gamma[x : \mu](x) =$

μ および $y \neq x$ なる y に対し $\Gamma[x:\mu](y) = \Gamma(y)$ とする．$\Gamma \downarrow \overline{x}$ は Γ から $\overline{x}$ 中の変数のエントリを除いた環境である．`tt` は真の値，`ff` は偽の値を表す．真偽式 b に対し，$d(b)$ は b の値が直接的に依存する変数の集合を表す．$tt(b,x)$ は b の値が真となる時に変数 x が取り得る値の集合，$ff(b,x)$ は b の値が偽となる時に変数 x が取り得る値の集合である．一般に $tt(b,x)$ と $ff(b,x)$ は互いに素ではない．

ブロックあるいは文 S に対する規則 $\Gamma \vdash S \rhd \Gamma'$ は，環境 Γ のもとで S が実行されると Γ が Γ' に変化することを表す．つまり，プログラムの実行に伴う各変数の確率分布の変化が得られる．ブロックの先頭では変数を宣言できるが，宣言された変数 x の初期値に関する確率分布 μ として任意の確率分布を与えられるとする．

整数式 e に対する規則 $\Gamma \vdash e : \mu$ は，環境 Γ のもとで e の確率分布が μ であることを表す．変数の確率分布は環境に記録されており，定数 n については確率 1 で値が n である．D-OP 規則から，$\mathrm{op}(e_1, e_2)$ の値が v となる確率は $[\![\mathrm{op}]\!](e_1, e_2) = v$ となるように e_1 と e_2 が適切な値を取る確率の合計である．ここで，$[\![\mathrm{op}]\!]$ は二項演算の意味を表す．例えば，$x : [1 \mapsto \frac{1}{2}, 2 \mapsto \frac{1}{2}], y : [1 \mapsto \frac{1}{3}, 2 \mapsto \frac{1}{3}, 3 \mapsto \frac{1}{3}]$ の時，$x + y : [2 \mapsto \frac{1}{6}, 3 \mapsto \frac{1}{3}, 4 \mapsto \frac{1}{3}, 5 \mapsto \frac{1}{6}]$ である．

真偽式 b に対する規則 $\Gamma \vdash b : \mu \mid \Gamma_t; \Gamma_f$ について，μ は環境 Γ のもとでの b の確率分布である．Γ_t は b の値が真，Γ_f は b の値が偽であるという条件付きで各変数が取り得る値の確率を表す環境であり，b の値に直接的に影響を与える変数の確率分布が Γ から更新される．D-CMP 規則の $[\![\mathrm{cmp}]\!]$ は比較演算の意味を表す．

4 例

簡単な例として，以下のプログラムに対し `declassify` 式の記述位置候補の順位付けを示す．

```
{
  L l; H h; if (l <2 h1) { l = 1; } else { l = 0; }
}
```

ここで，ラベル 1 は変数 `h` の参照，ラベル 2 は比較式 `l` $<$ `h` を指す．このプログラムは，機密度の高い変数 `h` の値を条件とする分岐先で機密度の低い変数への代入が行われており，不正な情報流が存在する．図 2 のアルゴリズムによって生成される制約集合は充足不能であり，この制約集合のラベル付き制約のみを含む MCS から，ラベル 1 の変数参照式とラベル 2 の比較式が `declassify` 式の記述位置候補となる．

2 つの式のエントロピーを求めるために，それぞれの確率分布を求める．`l` と `h` の初期値に関する確率分布を環境 $\Gamma = l : [0 \mapsto 1], h : [0 \mapsto \frac{1}{3}, 1 \mapsto \frac{1}{3}, 2 \mapsto \frac{1}{3}]$ のように定める．この時，ラベル 1 の変数参照式の確率分布は $\mu_1 = \Gamma(h) = [0 \mapsto \frac{1}{3}, 1 \mapsto \frac{1}{3}, 2 \mapsto \frac{1}{3}]$ である．ラベル 2 の比較式の確率分布は図 3 の D-CMP 規則より以下のように求められる．

$$\frac{\Gamma \vdash l : [0 \mapsto 1] \qquad \Gamma \vdash h : [0 \mapsto \frac{1}{3}, 1 \mapsto \frac{1}{3}, 2 \mapsto \frac{1}{3}]}{\Gamma \vdash l < h : \mu_2 \mid \Gamma_t; \Gamma_f}$$

ここで，$\mu_2 = [\mathtt{tt} \mapsto \frac{2}{3}, \mathtt{ff} \mapsto \frac{1}{3}]$，$\Gamma_t = l : [0 \mapsto 1], h : [1 \mapsto \frac{1}{2}, 2 \mapsto \frac{1}{2}]$，$\Gamma_f = l : [0 \mapsto 1], h : [0 \mapsto 1]$ である．以上より，ラベル i の式のエントロピー $\mathcal{H}_i$ は μ_i より

$$\mathcal{H}_1 = 3 \cdot \frac{1}{3} \log 3 = \log 3$$
$$\mathcal{H}_2 = \frac{2}{3} \log \frac{3}{2} + \frac{1}{3} \log 3 = \log 3 - \frac{2}{3} \log 2$$

である．$\mathcal{H}_2$ の方が小さいためラベル 2 の比較式に `declassify` 式を記述する方が望ましいと判断できる．

5 おわりに

本稿では，情報流解析のための型検査に失敗する不正な情報流を含むプログラムに対し，型検査を成功させるための `declassify` 式の記述位置の候補を順位付ける手法を提案した．`declassify` 式の記述位置であるそれぞれの式について情報量を求め，情報量の合計が小さい候補の順位を高くする．プログラム中の各式について取り得る値とその確率を表す確率分布を変数の初期値に関する確率分布から求める手法を示し，Shannon のエントロピーに基づいて `declassify` 式の記述位置候補を順位付ける例を示した．

今後の課題として，Shannon のエントロピー以外の情報量に基づく順位付けを検討すること，機密度の低い変数の値は既知であるとの条件付きエントロピーを用いること，`declassify` 式の記述位置である式の情報量とその式が依存する機密度の高い変数の関係を明らかにすること，セキュリティに関する性質と情報量の関係を明らかにすることが挙げられる．

参考文献

[1] Anindya Banerjee and David A. Naumann. Secure Information Flow and Pointer Confinement in a Java-like Language. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pp. 253–267, 2002.

[2] 黒川翔, 桑原寛明, 山本晋一郎, 坂部俊樹, 酒井正彦, 草刈圭一朗, 西田直樹. 例外処理付きオブジェクト指向プログラムにおける情報流の安全性解析のための型システム. 電子情報通信学会論文誌 D, Vol. J91-D, pp. 757–770, 2008.

[3] Andrei Sabelfeld and Andrew C. Myers. Language-Based Information-Flow Security. *IEEE Journal on Selected Areas in Communications*, Vol. 21, No. 1, pp. 5–19, 2003.

[4] Dennis Volpano, Geoffrey Smith, and Cynthia Irvine. A Sound Type System for Secure Flow Analysis. *Journal of Computer Security*, Vol. 4, No. 2, pp. 167–187, 1996.

[5] Andrei Sabelfeld and David Sands. Dimensions and Principles of Declassification. In *Proceedings of the 18th IEEE Computer Security Foundations Workshop*, pp. 255–269, 2005.

[6] Andrei Sabelfeld and Andrew C. Myers. A Model for Delimited Information Release. In *Software Security - Theories and Systems*, Vol. 3233 of *LNCS*, pp. 174–191. 2004.

[7] Gilles Barthe, Salvador Cavadini, and Tamara Rezk. Tractable Enforcement of Declassification Policies. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium*, pp. 83–97, 2008.

[8] Dave King, Susmit Jha, Trent Jaeger, Somesh Jha, and Sanjit A. Seshia. On Automatic Placement of Declassifiers for Information-Flow Security. Technical Report NAS-TR-0083-2007, Network and Security Research Center, 2007.

[9] Dave King, Susmit Jha, Divya Muthukumaran, Trent Jaeger, Somesh Jha, and Sanjit A. Seshia. Automating Security Mediation Placement. In *Programming Languages and Systems*, Vol. 6012 of *LNCS*, pp. 327–344. 2010.

[10] 桑原寛明, 國枝義敏. 情報流解析における Declassifier の配置手法. コンピュータソフトウェア, Vol. 32, No. 1, pp. 136–146, 2015.

[11] David Clark, Sebastian Hunt, and Pasquale Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, Vol. 15, No. 3, pp. 321–371, 2007.

[12] Pasquale Malacaria. Assessing Security Threats of Looping Constructs. In *Proceedings of the 34th ACM Symposium on Principles of Programming Languages*, pp. 225–235, 2007.

[13] M. Backes, B. Kopf, and A. Rybalchenko. Automatic Discovery and Quantification of Information Leaks. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, pp. 141–153, 2009.

[14] Chunyan Mu and David Clark. Quantitative Analysis of Secure Information Flow via Probabilistic Semantics. In *International Conference on Availability, Reliability and Security*, pp. 49–57, 2009.

[15] Geoffrey Smith. On the Foundations of Quantitative Information Flow. In *Foundations of Software Science and Computational Structures*, Vol. 5504 of *LNCS*, pp. 288–302. 2009.

[16] Pasquale Malacaria and Jonathan Heusser. Information Theory and Security: Quantitative Information Flow. In *Formal Methods for Quantitative Aspects of Programming Languages*, Vol. 6154 of *LNCS*, pp. 87–134. 2010.