実行時例外に伴う情報流の型検査に基づく解析手法

Type-based Analysis of Information Flow with Runtime Exceptions

桑原 寛明*　國枝 義敏†

Summary. This paper proposes a method of type-based analysis of information flow with runtime exceptions. For information flow analysis, each exception should have a secrecy which means secrecy of information propagated with that exception. Our method calculates the secrecy of exception based on the structure of target program and inferences the type for each program constructs. Proposed type system is sound for noninterference. We show a simple example of illegal information flow with runtime exception.

1　はじめに

プログラムの静的解析により機密情報が外部に漏れないことを検査する手法として，型検査に基づく情報流解析が提案されている [1] [2] [3] [4]．型検査に基づく情報流解析では，データの機密度を型として利用し，型付け可能なプログラムが非干渉性を満たすように型システムを構築する．非干渉性は，機密度の低いデータが機密度の高いデータに直接および間接的に依存しないことを表し，機密データ自体に加え機密データを推測できる情報も漏らさないという意味でよい性質である．

Volpano らは関数のない簡単な手続き型言語を対象として [4]，Pottier らは例外処理機構を持つ関数型言語を対象として [5]，Banerjee らはクラスベースのオブジェクト指向言語を対象として [1]，黒川らは例外処理機構を持つオブジェクト指向言語を対象として [2]，型検査に基づく情報流解析を提案している．いずれの提案についても，非干渉性に対して健全であることが示されている．

例外処理機構は，C++や Java をはじめとして多くのプログラミング言語でサポートされており，プログラム中で発生したエラーに対処する異常系の構造化を可能にする．一方，例外が発生する箇所とその例外を処理する箇所が異なるクラスやメソッドに存在することや，例外処理時にはコールスタックが一度に何段階も巻き戻されることから，例外処理に伴う情報流を直観的に把握することは難しい．黒川らの手法では，文の型付けを工夫することで例外処理に伴う情報流の解析を実現しているが，例外は Java の throw 文のような明示的に例外を発生させる構文によってのみ発生することが前提となっている．Java プログラムにおいてメソッド呼び出しのレシーバオブジェクトが null である時に発生する NullPointerException や，0 除算時に発生する ArithmeticException のように，プログラム中に明示されることなく発生する例外[1]も少なくないが，このような例外を扱うことはできていない．

本稿では，プログラム中に明示されることなく発生する例外（以下，実行時例外と呼ぶ）にも対応した型検査に基づく情報流解析を提案する．提案手法では，文の実行時に発生する可能性がある例外の集合をその文の型の一部に含む点は従来手法と同様であるが，例外の型に加えて例外の機密度を組にして文の型の一部とする．この時，例外の機密度を発生する例外に依存あるいは影響するデータの機密度に基づいて決定する．実行時例外に伴って流れる情報の機密度を文の型の一部として扱うことで，実行時例外に伴う情報流の解析を実現する．本手法により型付け可能なプログラムであれば，実行時例外が発生してもそのことから機密度の高いデータに関する情報は漏洩しない．

*Hiroaki Kuwabara, 南山大学情報センター

†Yoshitoshi Kunieda, 立命館大学情報理工学部

[1] 具体的にどのような例外が存在するかはプログラミング言語による．

$$
\begin{array}{rcl}
T & ::= & \text{Bool} \mid \text{Unit} \mid C \\
\tau & ::= & (T, \eta) \\
CL & ::= & \text{class } C \text{ extends } C\ \{\overline{\tau\ f;}\ \overline{M}\} \\
M & ::= & \tau\ m(\overline{\tau\ x})\ \eta \text{ throws } \overline{(C, \eta)}\ B \\
B & ::= & \{\overline{\tau\ x;}\ \overline{S;}\} \\
S & ::= & B \mid x = e \mid x = e.f \mid e.f = e \mid x = \text{new } C \\
 & \mid & x = e.m(\overline{e}) \mid \text{throw new } C \mid \text{if } (e)\ S \text{ else } S \\
 & \mid & \text{try } S \text{ catch}((C, \eta)\ x)\ S \ldots \text{catch}((C, \eta)\ x)\ S \\
e & ::= & x \mid \text{null} \mid \text{true} \mid \text{false} \mid \text{it} \mid e == e \mid (C)e \mid e \text{ is } C
\end{array}
$$

図 1　対象言語の文法

$$
\begin{array}{l}
[\![\Delta; \Sigma \vdash x = e.f : (\eta_s, \eta_h, E)]\!]\mu(h, s) = \\
\quad \textbf{let } l = [\![e]\!](h, s) \textbf{ in} \\
\quad\quad \textbf{if } l = \mathit{nil} \textbf{ then} \\
\quad\quad\quad \textbf{let } l_x = \mathit{fresh}(\text{NPE}, h) \textbf{ in } (h[l_x \mapsto \mathit{init}(\text{NPE})], s[\mathit{exception} \mapsto l_x]) \\
\quad\quad \textbf{else } (h, s[x \mapsto h(l)(f)])
\end{array}
$$

図 2　フィールド参照文の意味

2　対象言語

本稿では [1] [2] の拡張として実行時例外を扱うため，これらの対象言語とほぼ同等な言語を対象とする．実行時例外としてフィールドアクセスやメソッド呼び出しのレシーバオブジェクトが null である時に発生する NullPointerException（以下，NPE とする）のみを扱うが，他の種類の実行時例外も同様に考えられる．

対象言語の文法を図 1 に示す．T はデータの型である．η はデータの機密度であり，束 $(\mathcal{H}, \sqsubseteq)$ の元であるとする．τ は対象言語における型を表し，型はデータの型と機密度の組である．CL はクラス定義であり，C はクラス名，f はフィールド名を表す．$\overline{A}$ は長さ 0 以上の有限リストを表す略記である．M はメソッド定義であり，m はメソッド名，x は引数名を表す．throws $\overline{(C, \eta)}$ はメソッドからスローされる各例外の型と機密度がそれぞれ C と η であることを表す．B はブロック，S は文，e は式であり，プログラムは CL の並びである．

データ型のサブタイプ関係 $\leq$ を以下のように定義する．

- $\forall T.\ T \leq T$
- $C \leq D$ iff $C = D$ または $C' \leq D$ なる C' が存在して C の宣言が class C extends C' $\{\ldots\}$

簡単のため，任意のクラスを例外クラスとして扱えるものとする．

言語の意味は，フィールドアクセスおよびメソッド呼び出しのレシーバオブジェクトが null の場合を除いて [1] [2] と同様である．レシーバオブジェクトが null の場合の意味はいずれの構文においても同じであるため，フィールド参照文 $x = e.f$ についてのみ意味の定義を図 2 に示す．文の意味は，メソッド環境 μ およびヒープ h とストア s の組からヒープとストアの組への関数であり，文の実行に伴うヒープとストアの変化として定義される．フィールド参照文では，初めに式 e を評価してレシーバオブジェクトのロケーション l を求める．ロケーションが nil でなければ，ロケーションに対応するオブジェクトをヒープから取り出してフィールド f の値を求め，ストアに記録された変数 x の値を更新する．ロケーションが nil であれば NPE を発生させる．$\mathit{fresh}(\text{NPE}, h)$ によりヒープ上に新しく NPE オブジェクトを生成し，$\mathit{init}(\text{NPE})$ により各フィールドの値を初期化する．その後，生成された NPE オブジェクトのロケーション l_x を特殊変数 $\mathit{exception}$ の値としてストアに記録する．

3 実行時例外の情報流解析

3.1 例外の機密度

機密度の高いデータに依存してスローされた例外に対応する catch 節において機密度の低いデータを変更すると，機密度の低いデータから例外がスローされたか否かが判明し，さらに機密度の高いデータの具体的な値が判明する可能性がある．このような例外処理に伴う不正な情報流の発生を抑制するためには，機密度の高いデータに依存してスローされる例外を機密度の高い例外のみに制限し，かつ機密度の高い例外に対応する catch 節で変更できるデータを機密度の高いデータのみに制限すればよい．この時，スローされる例外の機密度はその例外の発生に対して影響を与えるデータの機密度を反映しているとみなすことができる．

[2] の情報流解析では，例外クラスを含むすべてのクラスに機密度が与えられており，例外クラスの機密度に基づいてスローされる例外の機密度が決定される．開発者は throw 文において，既存のあるいは自身で定義した適切な機密度の例外クラスを指定する．しかし，実行時例外は throw 文によって明示的にスローされるわけではないため，開発者がスローされる例外の機密度を指定することができない．

そこで本稿の体系では，[1] [2] と異なりクラスの機密度を採用しない．ただし，例外の機密度を決定することは必要であるため，スローされる例外の機密度として例外の発生に影響を与えるデータの機密度を利用する．具体的には，throw 文，フィールドアクセス文，メソッド呼び出し文が制御依存する各式の機密度の結びをスローされる例外の機密度とする．この機密度を，コンテキスト機密度と呼ぶ．最外の文のコンテキスト機密度は最小の機密度とする．例えば，$L \sqsubseteq H$ のもとで変数 `l` の機密度を L，`h` の機密度を H とすると，右図のプログラムにおいて (1) の位置ではコンテキスト機密度は L，(2) では $L \sqcup L = L$，(3) と (4) では $L \sqcup L \sqcup H = H$ である．

```
// (1)
if (l) {
  // (2)
  if (h) {
    // (3)
  } else {
    // (4)
  }
}
...
```

3.2 型付け規則

クラス定義，メソッド定義，文に対する型付け規則を図 3 に示す．CDEC 規則がクラス定義，MDEC 規則がメソッド定義，その他の規則が文の型付け規則である．図中の $\eta_\perp$ は機密度束の最小元，*result* はメソッドの返り値を表す変数，*method* はメソッドのシグネチャを返す関数，*field* はクラスが持つフィールドの型を返す関数である．例外クラス X と機密度 σ の組 (X, σ) の集合 E_1, E_2 に対し $E_1 \uplus E_2$ を以下のように定義する．

$$\begin{aligned} E_1 \uplus E_2 = \quad & \{(X,\sigma) \mid (X,\sigma) \in E_1 \wedge (X,\sigma) \notin E_2\} \\ & \cup \{(X,\sigma) \mid (X,\sigma) \notin E_1 \wedge (X,\sigma) \in E_2\} \\ & \cup \{(X,\sigma_1 \sqcup \sigma_2) \mid (X,\sigma_1) \in E_1 \wedge (X,\sigma_2) \in E_2\} \end{aligned}$$

文の型判定式 $\Delta; \Sigma \vdash S : (\eta_s, \eta_h, E)$ は，型環境 Δ およびコンテキスト機密度のスタック Σ のもとで文 S の型が (η_s, η_h, E) であることを表す．型環境 Δ は変数名からその型への関数である．コンテキスト機密度のスタック Σ は文 S を囲む if 文のネストに対応してコンテキスト機密度を記録しており，スタックのトップが S のコンテキスト機密度を指す．トップが σ のスタックを Σ, σ のように表す．文の型 (η_s, η_h, E) は，文において代入されるローカル変数や引数の機密度が η_s 以上，文の実行によるヒープエフェクトが η_h 以上，文の実行によりスローされ得る例外と例外の機密度の組の集合が E であることを示す．ヒープエフェクトとは，フィールドへの代入やインスタンスの生成によって変更されるヒープ上のデータの機密度を指す．

ASSIGN2 規則は，フィールド参照文 $x = e.f$ の型付け規則である．レシーバオブジェクトが null の時に NPE がスローされるため，スローされ得る例外として NPE が含まれる．x の値からフィールド f とレシーバオブジェクト e の値がわかる可能

$$\frac{C \text{ extends } D;\ \overline{f:\tau} \vdash M \text{ for each } M \in \overline{M}}{\vdash C \text{ extends } D\ \{\overline{\tau\ f};\ \overline{M}\}}\ [\text{CDEC}]$$

$$\frac{\begin{array}{l}\overline{f:\tau},\ \overline{x:(T_x,\eta_x)},\ this:(C,\eta_\perp),\ result:(T_r,\eta_r);\ \eta_\perp \vdash B:(\eta_s,\eta_h',E)\\ \eta_h \sqsubseteq \eta_h' \quad \forall(E_i,\eta_i)\in E.\ \exists j.\ E_i \le X_j \wedge \eta_i \sqsubseteq \eta_{X_j}\\ method(m,D) \text{ is defined } \Rightarrow\\ \quad method(m,D) = \overline{x':(T_x,\eta_x)} \xrightarrow{\eta_h} (T_r,\eta_r);\overline{(X,\eta_X)}\end{array}}{C \text{ extends } D;\ \overline{f:\tau} \vdash (T_r,\eta_r)\ m(\overline{(T_x,\eta_x)\ x})\ \eta_h \text{ throws } \overline{(X,\eta_X)}\ B}\ [\text{MDEC}]$$

$$\frac{\begin{array}{l}\Delta,\ \overline{x:(T_x,\eta_x)};\Sigma \vdash S_i:(\eta_{s_i},\eta_{h_i},E_i) \quad i\in\{1,\dots,n\}\\ \eta_s \sqsubseteq \eta_{s_i} \quad \eta_h \sqsubseteq \eta_{h_i} \quad \forall(X,\eta_X)\in E_i.\ \forall j>i.\ \eta_X \sqsubseteq \eta_{s_j} \wedge \eta_X \sqsubseteq \eta_{h_j}\end{array}}{\Delta;\Sigma \vdash \{\overline{(T_x,\eta_x)\ x};\ S_1;\dots S_n;\}:(\eta_s,\eta_h,\uplus_i E_i)}\ [\text{BLOCK}]$$

$$\frac{\Delta \vdash x:(T_x,\eta_x) \quad \Delta \vdash e:(T_e,\eta_e) \quad T_e \le T_x \quad \eta_e \sqsubseteq \eta_x \quad \eta_s \sqsubseteq \eta_x}{\Delta;\Sigma \vdash x = e:(\eta_s,\eta_h,\emptyset)}\ [\text{ASSIGN1}]$$

$$\frac{\begin{array}{l}\Delta \vdash x:(T_x,\eta_x) \quad \Delta \vdash e:(T_e,\eta_e) \quad (T_f,\eta_f) = field(T_e,f)\\ T_f \le T_x \quad \eta_f \sqsubseteq \eta_x \quad \eta_e \sqsubseteq \eta_x \quad \eta_s \sqsubseteq \eta_x \quad \eta_s \sqsubseteq \sigma\end{array}}{\Delta;\Sigma,\sigma \vdash x = e.f:(\eta_s,\eta_h,\{(\text{NPE},\eta_e \sqcup \sigma)\})}\ [\text{ASSIGN2}]$$

$$\frac{\begin{array}{l}\Delta \vdash e_1:(T_1,\eta_1) \quad \Delta \vdash e_2:(T_2,\eta_2) \quad (T_f,\eta_f) = field(T_1,f)\\ T_2 \le T_f \quad \eta_1 \sqsubseteq \eta_f \quad \eta_2 \sqsubseteq \eta_f \quad \eta_h \sqsubseteq \eta_f \quad \eta_s \sqsubseteq \sigma\end{array}}{\Delta;\Sigma,\sigma \vdash e_1.f = e_2:(\eta_s,\eta_h,\{(\text{NPE},\eta_1 \sqcup \sigma)\})}\ [\text{ASSIGN3}]$$

$$\frac{\Delta \vdash x:(T_x,\eta_x) \quad C \le T_x \quad \eta_s \sqsubseteq \eta_x}{\Delta;\Sigma \vdash x = \text{new } C:(\eta_s,\eta_h,\emptyset)}\ [\text{NEW}]$$

$$\frac{\begin{array}{l}\Delta \vdash x:(T_x,\eta_x) \quad \Delta \vdash e:(T_e,\eta_e) \quad \Delta \vdash e_i:(T_{e_i},\eta_{e_i})\\ y_1:(T_{y_1},\eta_{y_1}),\dots,y_n:(T_{y_n},\eta_{y_n}) \xrightarrow{\eta_{he}} (T_r,\eta_r);\overline{(X,\eta_X)} = method(T_e,m)\\ \forall i\in\{1,\dots,n\}.\ T_{e_i} \le T_{y_i} \wedge \eta_{e_i} \sqsubseteq \eta_{y_i} \quad T_r \le T_x \quad \eta_r \sqsubseteq \eta_x \quad \eta_e \sqsubseteq \eta_x\\ \eta_s \sqsubseteq \eta_x \quad \eta_e \sqsubseteq \eta_{he} \quad \eta_h \sqsubseteq \eta_{he} \quad \eta_s \sqsubseteq \sigma \quad E = \overline{(X,\eta_X)} \uplus \{(\text{NPE},\eta_e \sqcup \sigma)\}\end{array}}{\Delta;\Sigma,\sigma \vdash x = e.m(e_1,\dots,e_n):(\eta_s,\eta_h,E)}\ [\text{CALL}]$$

$$\frac{\eta_s \sqsubseteq \sigma}{\Delta;\Sigma,\sigma \vdash \text{throw new } C:(\eta_s,\eta_h,\{(C,\sigma)\})}\ [\text{THROW}]$$

$$\frac{\begin{array}{l}\Delta \vdash e:(Bool,\eta_e)\\ \Delta;\Sigma,\sigma,\sigma \sqcup \eta_e \vdash S_t:(\eta_{s_t},\eta_{h_t},E_t) \quad \Delta;\Sigma,\sigma,\sigma \sqcup \eta_e \vdash S_f:(\eta_{s_f},\eta_{h_f},E_f)\\ \eta_e \sqsubseteq \eta_s \quad \eta_e \sqsubseteq \eta_h \quad \eta_s \sqsubseteq \eta_{s_t} \quad \eta_s \sqsubseteq \eta_{s_f} \quad \eta_h \sqsubseteq \eta_{h_t} \quad \eta_h \sqsubseteq \eta_{h_f}\end{array}}{\Delta;\Sigma,\sigma \vdash \text{if } (e)\ S_t \text{ else } S_f:(\eta_s,\eta_h,E_t \uplus E_f)}\ [\text{IF}]$$

$$\frac{\begin{array}{l}\Delta;\Sigma \vdash S_0:(\eta_{s_0},\eta_{h_0},E_0) \quad \Delta,x:(X_i,\eta_{X_i});\Sigma \vdash S_i:(\eta_{s_i},\eta_{h_i},E_i)\\ \eta_s \sqsubseteq \eta_{s_i} \quad \eta_h \sqsubseteq \eta_{h_i} \quad \eta_{X_i} \sqsubseteq \eta_{s_i} \quad \eta_{X_i} \sqsubseteq \eta_{h_i} \quad i\in\{1,\dots,n\}\\ i \ne j \Rightarrow X_i \ne X_j \quad \forall(X,\eta_X)\in E_0.\ X \le X_i \Rightarrow \eta_X \sqsubseteq \eta_{X_i}\\ \eta_s \sqsubseteq \eta_{s_0} \quad \eta_h \sqsubseteq \eta_{h_0} \quad E = \{(X,\eta_X)\in E_0 \mid X \notin \bigcup_i X_i\} \uplus \biguplus_i E_i\end{array}}{\Delta;\Sigma \vdash \text{try } S_0\ \text{catch}((X_1,\eta_{X_1})\ x_1)\ S_1\ \dots \text{catch}((X_n,\eta_{X_n})\ x_n)\ S_n:(\eta_s,\eta_h,E)}\ [\text{CATCH}]$$

図 3　型付け規則

```
class O extends Object {
  (C, L) c;
  (Bool, H) h;
  (Unit, L) m1((Bool, H) h) L throws (NPE, H), (E, H) {
    (Unit, H) tmp;
    if (h) tmp = c.m3(true);
    else throw new E;
  }
  (Unit, L) m2((Bool, H) h) L throws (E, H) {
    (Bool, L) l;
    (Unit, L) tmp;
    l = true;
    try
      tmp = this.m1(h);
    catch ((NPE, H) e)
      l = false;
  }
}
```

図 4　例題プログラム

性があるため，x の機密度はいずれの機密度に対しても同等以上でなければならない．NPE がスローされたことが判明すると，コンテキスト機密度に加えて，レシーバオブジェクトが null であることも判明するため，NPE の機密度はコンテキスト機密度とレシーバオブジェクトの機密度の結びとする．フィールド代入文に対する ASSIGN3 規則とメソッド呼び出しに対する CALL 規則においても同様に，レシーバオブジェクトが null の可能性があるため，コンテキスト機密度とレシーバオブジェクトの機密度の結びを機密度として NPE をスローされうる例外に加える．

式の型判定式 $\Delta \vdash e : \tau$ は，型環境 Δ のもとで式 e の型が τ であることを示す．式の型付け規則は [1] と同様である．

型付け規則は非干渉性に対して健全である．つまり，型付け可能なプログラムには機密度の低いデータが機密度の高いデータに依存する不正な情報流が存在しないことが保証される．証明は [2] と同様に，機密度の高いデータのみが異なる初期状態からプログラムを実行した結果を比較すると，機密度の低いデータはすべて同じであることを示せばよい．

3.3　例

図 4 のプログラムを用いて型付けの例を示す．L, H は $L \sqsubseteq H$ を満たす機密度であり，$method(C, m3) = (\mathrm{Bool}, L) \xrightarrow{H} (\mathrm{Unit}, L); \emptyset$ とする．このプログラムには不正な情報流が存在する．メソッド `m2` から抜ける直前において，変数 l の値から例外 NPE がスローされたか否かがわかる．例外 NPE をスローする可能性があるのはメソッド `m1` の呼び出しであるが，メソッド `m1` における例外 NPE のスローは変数 h に依存する．つまり，機密度 L の変数 l の値から機密度 H の変数 h の値を推測することが可能であり，これは不正な情報流である．

メソッド `m1` は型付け可能である．if 文の条件式の機密度が H であることからメソッド呼び出し文と throw 文のコンテキスト機密度は H であり，throws 節にあるようにスローされ得る例外 NPE と E の機密度も H となっている．

一方，メソッド `m2` は型付け可能ではない．catch 節において例外が代入される変数 e の機密度が H であるが，機密度 L の変数 l に対して代入が行われており，これが CATCH 規則の条件に反している．変数 e の機密度が H であるということはスローされる例外の機密度が H であることを表しており，例外の機密度が H であるということはその例外が機密度が H のデータに依存してスローされることを表す．よって，変数 l の値から例外 NPE のスローに影響を与える機密度 H のデータを推

測することが可能であり，これが不正な情報流として検出される．

3.4 throws 節

図 4 のメソッド m1 からわかるように，本稿の対象言語ではメソッド宣言の throws 節に実行時例外である NPE も列挙する必要がある．しかし，スローされる可能性のある実行時例外をすべて把握して catch 節あるいは throws 節を記述することは開発者の負荷を大きくする．Java では throws 節に実行時例外を記述する必要はなく，C#には throws 節が存在しない．以下では，実行時例外を throws 節から省略する可能性について検討する．ここでは実行時例外として NPE のみを考えるが，他の実行時例外についてもスローされる箇所が異なるのみで同様である．

図 3 の CALL 規則から，メソッド呼び出し文の型付けにおいてはメソッド呼び出しによってスローされる可能性のある例外がすべて必要となる．そのため，throws 節に NPE を記述しない場合，呼び出されるメソッドから NPE がスローされるか否かと，スローされるならばその機密度を求める必要がある．この時，多態性により呼び出される可能性のあるメソッドの定義それぞれに対して，フィールドアクセスおよびメソッド呼び出しの有無と，コンテキスト機密度およびレシーバオブジェクトの機密度に基づく NPE の機密度を求めることになる．メソッド定義については，レシーバオブジェクトのクラスと継承関係から特定できる．フィールドアクセスとメソッド呼び出しの有無および NPE の機密度は，特定されたメソッド定義に対して図 3 の MDEC 規則を適用すれば得られる．以上より，実行時例外を throws 節から省略することは可能であり，定式化することは今後の課題である．

4 おわりに

本稿では，実行時例外に対応した型検査に基づく情報流解析を提案した．提案する型付け規則では，その文でスローされる可能性のある例外を文の型に含む．さらに，例外のスローに影響を与えるデータの機密度とレシーバオブジェクトの機密度を例外の機密度としてメソッドの呼び出し元に伝えることで，例外を媒介とする不正な情報流を検出する．

本稿の体系では解析を容易にするために，メソッド定義の throws 節に実行時例外を記述する必要がある．しかし，メソッドの外にスローされる可能性のある実行時例外を開発者が常に把握しなければならず望ましくない．この点を解決することは今後の課題である．本稿では実行時例外としてレシーバオブジェクトが null の時にスローされる NPE のみを対象としたが，他の実行時例外を扱うことと，そのための枠組みを検討することは今後の課題である．

謝辞 本研究の一部は 2016 年度南山大学パッヘ研究奨励金 I-A-2 および JSPS 科研費 15K00112 の助成による．

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. IEEE Computer Society Press, 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, No. 3, 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] François Pottier and Vincent Simonet. Information Flow Inference for ML. *ACM Trans. Program. Lang. Syst.*, Vol. 25, No. 1, pp. 117–158, 2003.