社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

例外処理付きオブジェクト指向言語における情報流の安全性解析

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†

† 名古屋大学大学院情報科学研究科
〒 464–8603 愛知県名古屋市千種区不老町
†† 愛知県立大学情報科学部
〒 480–1198 愛知県愛知郡長久手町大字熊張字茨ヶ廻間 1522–3

あらまし　本稿では，例外処理を含むオブジェクト指向プログラムから機密度の高いデータが外部へ漏洩しないことを検証する型システムを提案する．本型システムは，例外処理を含まないオブジェクト指向プログラムに対する従来の型システム [4] を拡張し，例外処理による制御フローを考慮した情報流解析を行えるようにしたものである．この型システムが安全性に対して健全であること，すなわち型付けできるプログラムは安全であることを示す．
キーワード　セキュリティ検査，情報流解析，型システム，例外処理

Security Analysis of Information Flow for an Object-Oriented Language with Exception Handling

Sho KUROKAWA†, Hiroaki KUWABARA†, Shinichiro YAMAMOTO††, Toshiki SAKABE†,
Masahiko SAKAI†, Keiichirou KUSAKARI†, and Naoki NISHIDA†

† Graduate School of Information Science, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, Aichi, 464–8603 Japan
†† School of Information Science, Aichi Prefectural University
Nagakute-cho, Aichi-gun, Aichi, 480–1198 Japan

Abstract　In this paper, we propose a type system for verifying that secret data don't leak from an object-oriented program with exception handling. Our type system is an extention of the type system [4] which is for a object-oriented language without exception handling. It analyzes information flow taking into account the control flow caused by the exception handling. We show that our type system is sound, that is, all well-typed programs are secure from data leakage.
Key words　Security Checking, Information Flow Analysis, Type System, Exception Handling

1. はじめに

パスワードやクレジット番号など機密性の高いデータを扱うプログラムは，機密データを外部に漏洩しないことが要求される．このような要求を満たすかを検証する手法として情報流解析が提案されている．[1] 情報流解析は機密データの情報がどのように伝播するか静的に解析する手法であり，外部から参照できるデータに機密データの情報が伝播しないか検証することで，機密データが漏洩しないことを確認する．本稿では，機密データを外部に漏洩しないことをプログラムの安全性と呼ぶ．

情報流解析は様々な言語に対して提案されてきた．Volpanoら [2] は簡単な手続き型言語に対する情報流解析を型システムを用いて形式化し，その型システムが非干渉性に対し健全であることを示した．非干渉性とは「機密度の低いデータが機密度の高いデータの情報に依存しない」ことを表す性質である．Myers [3] は Java の拡張言語 JFlow に対し，型システムに基づく情報流解析を提案している．JFlow は例外処理を記述できる等，表現力に豊む言語であるため，従来手法よりも型付けできるプログラムが豊富である．しかしこの型システムが非干渉性に対して健全であることの証明はされていない．Banerjeeら [4]〜[6] は簡単なオブジェクト指向言語に対して型システムに基づく情報流解析を提案し，非干渉性に対し健全であること

を示した．しかし例外処理を考慮していない．

例外処理機構はプログラム中で発生したエラーに対する処理の構造化を可能にする．このため，C++や Java など近年広く用いられているオブジェクト指向言語を初めとして多くのプログラミング言語が例外処理機構をサポートしている．従来の情報流解析で扱ってきた分岐や繰り返し等の制御フローは，単一メソッドに着目するだけで解析できる．しかし，例外処理では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であることも少なくない．このように例外処理の制御フローは複雑であるため，コーディングミスなどによって機密データを漏洩させてしまう可能性が高くなる．

本稿では，例外処理機構を持つオブジェクト指向言語に対し情報流解析を用いてプログラムの安全性を検証する手法を提案する．例外処理を考慮した情報流解析を行う型システムを構築し，型システムが非干渉性に対して健全であることを示す．本型システムで型付けできるプログラムは，機密データを外部に漏洩しない安全なプログラムである．

本稿の構成を述べる．2 節では情報流解析について説明する．3 節では対象言語を定義し，4 節では対象言語に機密度を付加して拡張し，安全な型システムを定義する．5 節で適用例を示し，6 節で健全性を証明する．最後に 7 節でまとめを行う．

2. 情報流解析

情報流解析は，プログラム中のデータフローと制御フローから各データの機密度を算出する．以下に解析の例を示す．x を機密度の高いデータ，y を機密度の低いデータとする．x は外部に漏洩してはならないデータ，y は外部に公開されているデータである．このとき，if (x) $\{y = \text{true};\}$ else $\{y = \text{false};\}$ では y の値から x の値を推測できる．すなわち x の情報が y へ伝わっているため，外部に機密データが漏洩しているとみなす．情報流解析ではこのように機密度の高いデータから機密度の低いデータへ情報がフローしていないかを解析する．

本研究では以下の前提の下で情報流解析を行う．

（1） プログラム中の各データに対し，機密データには高い機密度を与え，公開データ（外部ユーザが参照できるデータ）には低い機密度を与えるものとする．

（2） 外部ユーザはソースコードにアクセスできる．

安全なプログラムは（2）の条件下でも機密データを外部ユーザに漏洩しない．なぜなら，安全なプログラムでは公開データに機密データの情報が含まれないため，外部ユーザはプログラムの構造が分かっていても公開データから機密データを推測することができないからである．

2.1 例外処理を考慮した情報流解析

例外処理を考慮した情報流解析を図 1 を例に説明する．メソッド m と n はクラス O で宣言されているとする．実引数 x を与えて m を呼び出したとき，m の構造より x から例外 E へ情報がフローしていることが分かる．一方，E がスローされるか否かで y の値が変わるため，E から y へ情報がフローしている．ここで x を機密度の高いデータ，y を機密度の低いデータとすると，x から y へ情報が漏洩しているため，このプログラムは安全でない．

```
void m(bool b) throws E{
  if (!b) {
    throw new E();
  }
}
```

```
void n(O o, bool x){
  bool y = true;
  try {o.m(x);}
  catch(E e) {y = false;}
}
```

図 1　例外処理により機密データが漏洩するプログラム

例外を媒介とするデータ間の情報流を解析するため，通常のデータと同様に例外にも機密度を与える．機密データを処理している時に発生する例外には高い機密度を与え，機密度の高い例外に対する処理において公開データを変更しないか検査する．このようにして例外処理を行っても機密データが漏洩しないことを保証する．

3. 対象言語

3.1 構　　文

本稿で対象とする例外処理付きオブジェクト指向言語を図 2 に示す．この言語は [4] の言語に例外処理を加えた言語である．以下では，データの種類を表す型として T，U，クラス名として B，C，D，例外クラス名として E を用いる．f はフィールド名，M はメソッド宣言，m はメソッド名，x は変数名 (ターゲットオブジェクト $this$ と返り値 $result$ を含む)，S は文，e は式をそれぞれ表す．$\overline{T}$ は有限リストを表す略記である．$n \geqq 1$ において $\overline{T\ f}$ は $T_1\ f_1, \ldots, T_n\ f_n$，$\overline{x : T}$ は $x_1 : T_1, \ldots, x_n : T_n$ をそれぞれ表す．

プログラム全体はクラステーブル CT として与えられる．クラステーブルはクラス名からその宣言への関数である．

任意のクラスのスーパークラスとして $Object$ クラスを，任意の例外クラスのスーパークラスとして $Exception$ クラスを定義する．$Exception$ は $Object$ のサブクラスであり，いずれもフィールドやメソッドを持たないクラスである．

関数 $mtype$ と $fields$ をそれぞれ指定されたクラスのメソッド，フィールドの型情報を取得する関数とする．クラス宣言 $CT(C) = \text{class } C \text{ extends } D\ \{\overline{T\ f};\ \overline{M}\}$，メソッド宣言 $M = T\ m(\overline{T\ x})\ \text{throws}\ \overline{E}\ \{S\}$ とすると，$mtype(m, C) = \overline{x : T} \longrightarrow T; \overline{E}$ である．m が D から継承されたメソッドである場合，$mtype(m, C) = mtype(m, D)$ である．本稿では簡単のためメソッドのオーバーロードは扱わない．$fields(C) = dfields(C) \cup fileds(D)$ である．$dfields(C)$ は C で宣言されたフィールドの型情報を表し，$dfields(C) = \overline{T\ f}$ である．メ

T	::=	$\text{bool} \mid \text{unit} \mid C$
CL	::=	$\text{class } C \text{ extends } C\ \{\overline{T\ f}\ ;\ \overline{M}\}$
M	::=	$T\ \text{m}(\overline{T\ x})\ \{S\} \mid T\ \text{m}(\overline{T\ x})\ \text{throws}\ \overline{C}\ \{S\}$
S	::=	$x := e \mid e.f := e \mid x := \text{new } C \mid \text{if } e \text{ then } S \text{ else } S$
	$\mid$	$T\ x := e \text{ in } S \mid x := e.m(\bar{e}) \mid S\ ;\ S \mid \text{throw new } C$
	$\mid$	$\text{try } S\ \text{catch}(C\ e)\ S \ldots \text{catch}(C\ e)\ S$
e	::=	$x \mid \text{null} \mid \text{true} \mid \text{false} \mid \text{it} \mid e.f \mid e == e$
	$\mid$	$e \text{ is } C \mid (C)e$

図 2　例外処理付きオブジェクト指向言語

$$\frac{E \leqq \mathit{Exception}}{\Gamma \vdash \text{throw new } E}$$

$$\frac{\Gamma \vdash S \quad \Gamma, e_i : E_i \vdash S_i \quad E_i \leqq \mathit{Exception} \quad \text{where } i \in [1, \ldots, n]}{\Gamma \vdash \text{try } S \text{ catch}(E_1\ e_1)\ S_1\ \ldots\ \text{catch}(E_n\ e_n)\ S_n}$$

図 3　例外処理文における型推論規則

ソッドと異なりフィールドはオーバーライドできないとする．

型環境 Γ を変数名からその型への関数とする．$\Gamma \vdash e : T$ は式 e が Γ の下で型 T であることを表し，$\Gamma \vdash S$ は Γ の下で S は文であることを表す．

型に対するサブタイプ関係 $\leqq$ は以下を満たす関係である．

（1） bool $\leqq$ bool, unit $\leqq$ unit

（2） $\forall C.\ \Gamma(\text{null}) \leqq C$

（3） $C \leqq D$ iff $C = D$ または C の宣言は $B \leqq D$ なる B について，class C extends B $\{\ldots\}$ を満たす．

図 3 に例外処理に関する文の型推論規則を与える．式とその他の文の型推論規則は [4] を参照されたい．

3.2 意　味　論

プログラムの状態はヒープ h とストア η で表す．h はロケーションからオブジェクトの状態 s への関数，η はローカル変数名や引数名からロケーション，プリミティブ値への関数である．s はフィールド名からロケーション，プリミティブ値への関数である．ロケーションはヒープにあるオブジェクトのアドレスを表し，ロケーションの集合を Loc とする．$\mathit{loctype}$ をオブジェクトのクラスを取得する関数であるとする．$\mathit{loctype}(l) = C$ はロケーション l が指すオブジェクトのクラス名が C であることを表す．

各関数における定義域，値域をそれぞれ dom，rng と表記する．図 4 に意味領域を示す．$[\![\Gamma]\!]$ はストアの集合，$[\![\mathit{state}(C)]\!]$ はクラス C のオブジェクトの状態の集合，$[\![\mathit{Heap}]\!]$ はヒープの集合，$[\![C, \overline{x:T} \to T; \overline{E}]\!]$ はクラス C のメソッドの意味の集合，$[\![\mathit{MEnv}]\!]$ はメソッド環境の集合をそれぞれ表す．ヒープ領域にないロケーションを指さないようにするため，$[\![\mathit{Heap} \otimes \Gamma]\!]$ と $[\![\mathit{Heap} \otimes T \otimes T]\!]$ を定義する．意味 θ に対し $d \in [\![\theta_\perp]\!]$ は，$d \in [\![\theta]\!]$ または $d = \perp$ を表す．

ロケーションの割り当ては，以下に定義されるアロケータ fresh を用いる．

定義 1 (Allocator fresh)　インスタンスを生成し，そのロケーションを返すアロケータ fresh は以下の性質を満たす．

1. $\forall C, h.\ (\mathit{loctype}(\mathit{fresh}(C,h)) = C \wedge \mathit{fresh}(C,h) \notin \mathit{dom}(h))$
2. $\mathit{dom}(h_1) \cap \mathit{locs}(C) = \mathit{dom}(h_2) \cap \mathit{locs}(C)$
   $\Rightarrow \mathit{fresh}(C, h_1) = \mathit{fresh}(C, h_2)$

ここで $\mathit{locs}(C) = \{l \mid \mathit{loctype}(l) = C\}$ である．　□

$[\![\Gamma \vdash e : T]\!](h, \eta) = d$ のとき，(h, η) の状態で式 e を評価すると値 d を得る事を意味する．$[\![\Gamma \vdash S]\!]\mu(h, \eta) = (h', \eta')$ のとき，メソッド環境 μ，状態 (h, η) で文 S を実行すると状態が (h', η') になる事を意味する．なお，d も (h', η') もエラー値を表す $\perp$ となりうることに注意する．

$$
\begin{array}{l}
[\![\Gamma \vdash \text{throw new } E]\!]\mu(h,\eta) \\
= \quad \textbf{let } l = \mathit{fresh}(E,h) \textbf{ in} \\
\quad\quad \textbf{let } h_1 = [h \mid l \mapsto [\mathit{fileds}(E) \mapsto \mathit{defaults}]] \textbf{ in} \\
\quad\quad \textbf{let } \eta_1 = [\eta \mid \mathit{exception} \mapsto l] \textbf{ in } (h_1, \eta_1) \\
[\![\Gamma \vdash \text{try } S \text{ catch}(E_1\ e_1)\ S_1 \ldots \text{catch}(E_n\ e_n)\ S_n]\!]\mu(h,\eta) \\
= \quad \textbf{let } (h_1, \eta_1) = [\![\Gamma \vdash S]\!]\mu(h,\eta) \textbf{ in let } l = \eta_1(\mathit{exception}) \textbf{ in} \\
\quad\quad \textbf{if } l = \mathit{nil} \textbf{ then } (h_1, \eta_1) \\
\quad\quad \textbf{else if } \exists i.\ \mathit{loctype}(l) = E_i \textbf{ then} \\
\quad\quad\quad\quad \textbf{let } \eta_2 = [\eta_1 \mid e_i \mapsto l,\ \mathit{exception} \mapsto \mathit{nil}] \textbf{ in} \\
\quad\quad\quad\quad [\![\Gamma, e_i : E_i \vdash S_i]\!]\mu(h_1, \eta_2) \\
\quad\quad\quad\quad \textbf{else } (h_1, \eta_1) \\
[\![\Gamma \vdash x := e.m(\overline{e})]\!]\mu(h,\eta) \\
= \quad \textbf{let } l = [\![\Gamma \vdash e : D]\!](h,\eta) \textbf{ in} \\
\quad\quad \textbf{if } l = \mathit{nil} \textbf{ then } \perp \\
\quad\quad \textbf{else let } \overline{x:T} \to T; \overline{E} = \mathit{mtype}(m, D) \textbf{ in} \\
\quad\quad\quad\quad \textbf{let } \overline{d} = [\![\Gamma \vdash \overline{e:U}]\!](h,\eta) \textbf{ in} \\
\quad\quad\quad\quad \textbf{let } \eta_1 = [\overline{x \mapsto d}, \mathit{this} \mapsto l] \textbf{ in} \\
\quad\quad\quad\quad \textbf{let } (h_1, d, e_0) = \mu(\mathit{loctype}(l), m)(h, \eta_1) \textbf{ in} \\
\quad\quad\quad\quad \textbf{if } e_0 = \mathit{nil} \textbf{ then } (h_1, [\eta \mid x \mapsto d]) \\
\quad\quad\quad\quad \textbf{else } (h_1, [\eta \mid \mathit{exception} \mapsto e_0]) \\
[\![\Gamma \vdash S; S']\!]\mu(h,\eta) \\
= \quad \textbf{let } (h_1, \eta_1) = [\![\Gamma \vdash S]\!]\mu(h,\eta) \textbf{ in} \\
\quad\quad \textbf{if } \eta_1(\mathit{exception}) = \mathit{nil} \textbf{ then } [\![\Gamma \vdash S']\!]\mu(h_1, \eta_1) \textbf{ else } (h_1, \eta_1) \\
[\![M]\!]\mu(h,\eta) \\
= \quad \textbf{let } \eta_1 = [\eta \mid \mathit{result} \mapsto \mathit{default}] \textbf{ in} \\
\quad\quad \textbf{let } (h_1, \eta_2) = [\![\overline{x:T}, \mathit{this} : C, \mathit{result} : T \vdash S]\!]\mu(h, \eta_1) \textbf{ in} \\
\quad\quad (h_1, \eta_2(\mathit{result}), \eta_2(\mathit{exception}))
\end{array}
$$

図 5　変更を加えた文とメソッド宣言における意味論

メソッド環境 μ はクラス名 C とメソッド名 m からメソッドの意味 $[\![C, \overline{x:T} \to T; \overline{E}]\!]$ を返す関数である．以下に定義を与える．

定義 2 (Semantics of complete program)　$n \in \mathbb{N}$ についてメソッド環境 $\mu_n \in [\![\mathit{MEnv}]\!]$ を以下のように定義する．

$$\mu_0(C, m) = \lambda(h, \eta).\ \perp$$

$$\mu_{n+1}(C, m) = \begin{cases} [\![M]\!]\mu_n & \text{if } m \text{ は } M \text{ として } C \text{ で宣言} \\ \mu_{n+1}(B, m) & \text{if } m \text{ は } B \text{ から継承} \end{cases}$$

μ 上の順序関係 $\preceq$ を以下のように定義する．

$$\mu \preceq \mu' \ \textbf{ iff } \ \forall C, m, h, \eta.\ \mu(C,m)(h,\eta) \neq \perp \Rightarrow \mu(C,m)(h,\eta) = \mu'(C,m)(h,\eta)$$

クラステーブル CT の意味論を μ 上の順序関係 $\preceq$ の上限とし $[\![CT]\!] = \mathit{lub}\ \mu$ と定義する．　□

[4] に例外処理を追加したことにより，メソッドコール文，逐次実行文，メソッド宣言に関する意味論を変更した．これらの意味論と例外処理文の意味論を図 5 に与える．ここで，let $d = E_1$ in E_2 は E_1 の値が $\perp$ のとき $\perp$，そうでなければ d を E_1 の値に束縛したときの E_2 の値である．また，$[\eta \mid x \mapsto d]$ は η の更新を表す．

本稿では例外処理の意味論を定義するため，[4] で扱っていた言語の意味論において，予約変数に $\mathit{exception}$ を加えてストアを拡張する．ストアは以下 3 つの予約変数を持つ．

$$
\begin{array}{rl}
[\![\mathrm{bool}]\!] = & \{true, false\} \quad [\![\mathrm{unit}]\!] = \{it\} \quad [\![C]\!] = \{nil\} \cup \{l \mid loctype(l) \leqq C\} \\
[\![\Gamma]\!] = & \{\eta \mid (dom(\eta) - exception = dom(\Gamma)) \wedge \eta(this) \neq nil \wedge \forall x \in (dom(\eta) - exception).\ \eta(x) \in \Gamma(x)\} \\
[\![state(C)]\!] = & \{s \mid dom(s) = dom(fields(C)) \wedge \forall (f:T) \in fields(C).\ s(f) \in [\![T]\!]\} \\
[\![Heap]\!] = & \{h \mid dom(h) \subseteqq Loc \wedge (rng(s) \cap Loc \subseteqq dom(h) \text{ for all } s \in rng(h)) \wedge \forall l \in dom(h).\ h(l) \in [\![state(loctype(l))]\!]\} \\
[\![Heap \otimes \Gamma]\!] = & \{(h, \eta) \mid h \in [\![Heap]\!] \wedge \eta \in [\![\Gamma]\!] \wedge (rng(\eta) \cap Loc \subseteqq dom(h))\} \\
[\![Heap \otimes T \otimes T]\!] = & \{(h, d, e) \mid h \in [\![Heap]\!] \wedge d, e \in [\![T]\!] \wedge (d, e \in Loc \Rightarrow d, e \in dom(h))\} \\
[\![C, \overline{x:T} \to T; \overline{E}]\!] = & [\![Heap \otimes (\overline{x:T}, this : C, result : T)]\!] \to [\![(Heap \otimes T \otimes T)_\perp]\!]\} \\
[\![MEnv]\!] = & \{\mu \mid \forall C, m.\ \mu(C,m) \text{ is defined iff } mtype(m,C) \text{ is defined} \wedge \mu(C,m) \in [\![C, mtype(m,C)]\!]\}
\end{array}
$$

図 4　意味領域

- ターゲットオブジェクトを示す変数 $this$
- メソッドの返り値を示す変数 $result$
- スローされた例外を示す変数 $exception$

文を実行して例外が発生したとき，$exception$ に発生した例外オブジェクトのロケーションを割り当てる．逐次実行文では $exception$ の値が nil でないとき，例外が発生したとして以降の文を実行しない．変数名 $exception$ はプログラム中で出現せず宣言されることは無いとする．

本稿では throw 文と try-catch 文による明示的な例外処理を扱い，レシーバオブジェクトの null 参照等，プログラム実行時に決まる例外は扱わない．

4. 安全型システム

本節では図 2 の言語に機密度を付加する．型 T の宣言を型と機密度 κ の組 (T, κ) の宣言に置き換える．この組を安全型と呼ぶ．本稿では束モデル [1] を用いて機密度を束 $(\{\perp, L, H\}, \sqsubseteq)$ で表す．機密度間の順序は $\perp \sqsubseteq L$，$L \sqsubseteq H$ とする．機密度 κ_1，κ_2 について，これらの上限を $\kappa_1 \sqcup \kappa_2$，下限を $\kappa_1 \sqcap \kappa_2$ と表記する．安全型でラベル付けされた言語を図 6 に示す．

安全型環境 Δ は変数名からその安全型への関数である．$\Delta \vdash S : (\kappa_1, \kappa_2, \kappa_3)$ は文 S において，(1) 値割り当てされるローカル変数やパラメータの機密度が少なくとも κ_1 以上，(2) ヒープエフェクトが少なくとも κ_2 以上，(3) スローされうる例外の機密度が κ_3 であることを表す．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中のデータの機密度の最小値を意味する．

クラス C の宣言を $CT(C) = \text{class } C\ \kappa_1 \text{ extends } D\ \{\overline{\tau\ f};\ \overline{M}\}$，メソッド M の宣言を $M = (T, \kappa)\ m(\overline{(T, \kappa)\ x})\ \kappa_h \text{ throws } \overline{(E, \kappa_e)}\{S\}$ とする．この時，クラス C のインスタンスが持つ機密度は κ である．各引数の機密度が $\overline{\kappa}$ 以下で m がコールされた時，レシーバオブジェクトに対するヒープエフェクトが κ_h 以上，返り値の機密度が κ 以下，スローされうる各例外の機密度が $\overline{\kappa_e}$ であることを意味する．

$$
\begin{array}{rcl}
\kappa & ::= & \mathrm{L} \mid \mathrm{H} \\
\tau & ::= & (T, \kappa) \\
CL & ::= & \text{class } C\ \kappa \text{ extends } C\ \{\overline{\tau\ f}\ ;\ \overline{M}\} \\
M & ::= & \tau\ \mathrm{m}(\overline{\tau\ x})\ \kappa\ \{S\} \mid \tau\ \mathrm{m}(\overline{\tau\ x})\ \kappa \text{ throws } \overline{\tau}\ \{S\} \\
S & ::= & \ldots \mid \tau\ x := e \text{ in } S \mid \ldots \\
 & \mid & \text{try } S \text{ catch}(\tau\ e)\ S \ldots \text{ catch}(\tau\ e)\ S \mid \ldots
\end{array}
$$

図 6　安全型を持つ例外処理付きオブジェクト指向言語

関数 $mtype$，$fields$，$dfields$ を安全型を用いるように拡張した関数をそれぞれ $smtype$，$sfields$，$sdfields$ とする．$smtype(m, C) = \overline{x : (T, \kappa)} \xrightarrow{\kappa_h} (T, \kappa); \overline{(E, \kappa_e)}$，$sfields(C) = sdfields(C) \cup sfileds(D)$，$sdfields(C) = \overline{\tau\ f}$ である．

クラス名から機密度を取得する関数を $level$ とする．$level(C) = \kappa_1$ である．任意のクラスのスーパークラス $Object$ に対し $level(Object) = L$ とする．ロケーション l に対し $level(l) = level(loctype(l))$ とする．

以下では，安全型から型へと変換するシンボル † を用いる．$(T, \kappa)^\dagger = T$ のように安全型から機密度を削除する．型環境，文，メソッド宣言に対しても同様に用いることができる．

図 7 に安全型の型推論規則を示す．MDEC2 規則の条件 $throw(S)$ は S でスローされうる例外クラスのリストを表す．式における安全型の型推論規則は [4] を参照されたい．CATCH 規則について説明する．条件 $\kappa_i \sqsubseteq \kappa_{1i} \sqcap \kappa_{2i}$ は，キャッチ節内で値割り当てされる変数やフィールドがキャッチした例外の機密度以上でなければならないことを表す．キャッチした例外からキャッチ節内の変数やフィールドに情報がフローしているためである．

5. 適　用　例

図 1 のプログラムを図 6 の言語で記述したプログラムを図 8 に示す．E は機密度が H の例外クラスであり，プリミティブ（$true$ 等）の機密度は L であるとする．例外処理を持つこのプログラムに対し，本型システムを適用する．

メソッド m と n にそれぞれ MDEC2 規則，MDEC1 規則をそれぞれ適用すると，m は型付けに成功するが n は型付けに失敗する．機密度が H である例外 E のキャッチ節内で機密度が L である変数 y の値を変更しているため，CATCH 規則の条件 $\kappa_i \sqsubseteq \kappa_{1i} \sqcap \kappa_{2i}$ を満たさないからである．n を実行後，y の値から機密度が H である x の値を特定できるため安全でない．

以上より図 8 のプログラムは，本型システムで型付けできない．y の機密度を H に変更し，y は外部に公開されない機密データとすれば，図 8 のプログラムは本型システムで型付けできる安全なプログラムになる．

6. 型システムの健全性

本節では，提案した型システムが非干渉性について健全であることを示す．健全性の証明に必要となる定義と補題を以

ASSIGN1

$$\frac{\Delta, x:(T,\kappa) \vdash e:(T',\kappa') \quad x \neq this \quad T' \leqq T \quad \kappa' \sqcup \kappa_1 \sqsubseteq \kappa}{\Delta, x:(T,\kappa) \vdash x := e : (\kappa_1, \kappa_2, \bot)}$$

ASSIGN2

$$\frac{\Delta \vdash e_1:(T,\kappa) \quad \Delta \vdash e_2:(T',\kappa') \quad T' \leqq U \quad (U,\kappa_f)\ f \in sdfields(\Delta(this)) \quad \kappa \sqcup \kappa' \sqcup \kappa_2 \sqsubseteq \kappa_f}{\Delta, x:(T,\kappa) \vdash e_1.f := e_2 : (\kappa_1, \kappa_2, \bot)}$$

NEW

$$\frac{x \neq this \quad B \leqq D \quad level(B) \sqcup \kappa_1 \sqsubseteq \kappa \quad \kappa_2 \sqsubseteq level(B)}{\Delta, x:(D,\kappa) \vdash x := \text{new } B : (\kappa_1, \kappa_2, \bot)}$$

THROW

$$\frac{E \leqq Exception \quad \kappa_1 \sqcup \kappa_2 \sqsubseteq level(E)}{\Delta \vdash \text{throw new } E : (\kappa_1, \kappa_2, level(E))}$$

IF

$$\frac{\Delta \vdash e:(bool,\kappa) \quad \Delta \vdash S:(\kappa_1,\kappa_2,\kappa_3) \quad \Delta \vdash S':(\kappa_1',\kappa_2',\kappa_3') \quad \kappa \sqsubseteq \kappa_4 \sqcap \kappa_5 \quad \kappa_4 \sqsubseteq \kappa_1 \sqcap \kappa_{1'} \quad \kappa_5 \sqsubseteq \kappa_2 \sqcap \kappa_{2'}}{\Delta \vdash \text{if } e\ S \text{ else } S' : (\kappa_4, \kappa_5, \kappa_3 \sqcup \kappa_3')}$$

VAR

$$\frac{\Delta \vdash e:(T',\kappa') \quad \Delta, x:(T,\kappa) \vdash S:(\kappa_1,\kappa_2,\kappa_3) \quad T' \leqq T \quad \kappa' \sqsubseteq \kappa \quad \kappa_4 \sqsubseteq \kappa_1 \quad \kappa_5 \sqsubseteq \kappa_2}{\Delta \vdash (T,\kappa)\ x := e \text{ in } S : (\kappa_4, \kappa_5, \kappa_3)}$$

SEQ

$$\frac{\Delta \vdash S:(\kappa_1,\kappa_2,\kappa_3) \qquad \Delta \vdash S':(\kappa_1',\kappa_2',\kappa_3') \quad \kappa_4 \sqsubseteq \kappa_1 \sqcap \kappa_1' \quad \kappa_5 \sqsubseteq \kappa_2 \sqcap \kappa_2' \quad \kappa_3 \sqsubseteq \kappa_1' \sqcap \kappa_2'}{\Delta \vdash S; S' : (\kappa_4, \kappa_5, \kappa_3 \sqcup \kappa_3')}$$

CALL

$$\frac{\Delta, x:(U,\kappa) \vdash e:(D,\kappa_1) \quad \Delta, x:(U,\kappa) \vdash \overline{e:(U,\kappa')} \quad \overline{U} \leqq \overline{T} \quad T \leqq U \quad \overline{\kappa'} \sqsubseteq \overline{\kappa} \quad \kappa_1 \sqcup \kappa_3 \sqsubseteq \kappa_h \quad \kappa_1 \sqcup \kappa' \sqcup \kappa_2 \sqsubseteq \kappa \quad \kappa = H \wedge \overline{(E,\kappa_e)} \neq \emptyset \Rightarrow \bigsqcap \overline{\kappa_e} = H \quad \text{where } \overline{x:(T,\kappa)} \xrightarrow{\kappa_h} (T,\kappa'); \overline{(E,\kappa_e)} := smtype(m,D)}{\Delta, x:(U,\kappa) \vdash x := e.m(\overline{e}) : (\kappa_2, \kappa_3, \bigsqcup \overline{\kappa_e})}$$

CATCH

$$\frac{\Delta \vdash S:(\kappa_1,\kappa_2,\kappa_3) \quad \Delta, e_i:(E_i,\kappa_i) \vdash S_i:(\kappa_{1i},\kappa_{2i},\kappa_{3i}) \quad E_i \leqq Exception \quad level(E_i) = \kappa_i \quad \kappa_i \sqsubseteq \kappa_{1i} \sqcap \kappa_{2i} \quad \kappa_4 \sqsubseteq \kappa_1 \sqcap \bigsqcap_i \kappa_{1i} \quad \kappa_5 \sqsubseteq \kappa_2 \sqcap \bigsqcap_i \kappa_{2i} \quad \kappa_6 = \kappa_3 \sqcup \bigsqcup_i \kappa_{3i} \quad \text{where } i \in [1,\ldots,n]}{\Delta \vdash \text{ try } S \text{ catch}((E_1,\kappa_1)\ e_1)\ S_1\ \ldots\ \text{catch}((E_n,\kappa_n)\ e_n)\ S_n : (\kappa_4, \kappa_5, \kappa_6)}$$

MDEC1

$$\frac{\overline{x:(T,\kappa)},\ this:(C,\kappa_c),\ result:(T,\kappa) \vdash S:(\kappa_1,\kappa_h,\bot) \quad smtype(m,D) \text{ is defined} \Rightarrow smtype(m,D) \text{ equals } \overline{x:(T,\kappa)} \xrightarrow{\kappa_h} (T,\kappa);}{C\ \kappa_c \text{ extends } D \vdash (T,\kappa)\ m(\overline{\tau\ x})\ \kappa_h\ \{S\}}$$

MDEC2

$$\frac{\overline{x:(T,\kappa)},\ this:(C,\kappa_c),\ result:(T,\kappa) \vdash S:(\kappa_1,\kappa_h,\bigsqcup\overline{\kappa_e}) \quad throws(S) = \overline{E} \quad level(\overline{E}) = \overline{\kappa_e} \quad \kappa = L \Rightarrow \bigsqcup\overline{\kappa_e} = L \quad smtype(m,D) \text{ is defined} \Rightarrow smtype(m,D) \text{ equals } \overline{x:(T,\kappa)} \xrightarrow{\kappa_h} (T,\kappa); \overline{(E,\kappa_e)}}{C\ \kappa_c \text{ extends } D \vdash (T,\kappa)\ m(\overline{\tau\ x})\ \kappa_h \text{ throws } \overline{(E,\kappa_e)}\ \{S\}}$$

CDEC

$$\frac{level(D) \leqq \kappa \quad C\ \kappa \text{ extends } D \vdash M \text{ for each } M \in \overline{M} \quad level(D) \neq \kappa \Rightarrow \text{every } m \text{ with } smtype(m,D) \text{ defined is overridden in } C \text{ by some } M \in \overline{M}}{\vdash \text{class } C\ \kappa \text{ extends } D\{\overline{\tau\ f};\ \overline{M}\}}$$

図 7　安全型に対する型推論規則

```
class O extends Object {
  (unit,H) m((bool,H) b) H throws (E,H) {
    if b == false
      throw new E;
    else
      result := it;
  }
  (unit,H) n((O,L) o, (bool,H) x) L {
    (bool,L) y := true in
    try
      result := o.m(x);
    catch((E,H) e)
      y := false;
  }
}
```

図 8　サンプルプログラム

下に与える．各補題の証明は http://www.sakabe.i.is.nagoya-u.ac.jp/~kurokawa/proof.pdf を参照されたい．

定義 3 (Noninterference)　例外を考慮した非干渉性の定義を以下に与える．

- 機密度が H のデータを変更してプログラムを実行しても，機密度が L のデータは変化しない．
- プログラムの実行で機密度が H の例外がスローされても，機密度が L のデータは変化しない．　□

定義 4 (Indistinguishable by L)　オブジェクトの状態，ヒープ，ストアそれぞれに対し，L 区別不可能であることを表す関係 $\sim$ を定義する．L 区別不可能とは機密度が L である全てのデータの値が等しいことを表す．$LLoc = \{l \mid level(l) = L, l \in Loc\}$ とすると

- $s, s' \in [\![\ state(C)\ ]\!]$ について $s \sim s'$ **iff**
  $\forall (f:(T,\kappa)) \in sfields(C).\ \kappa = L \Rightarrow s(f) = s'(f)$
- $h, h' \in [\![Heap]\!]$ について $h \sim h'$ **iff**

(1) $dom(h) \cap LLoc = dom(h') \cap LLoc$, かつ

(2) $\forall l \in dom(h) \cap LLoc.\ h(l) \sim h'(l)$

- $\eta, \eta' \in [\![\Delta^\dagger]\!]$ について $\eta \sim_\Delta \eta'$ **iff**

(1) $\forall x \in dom(\Delta).\ \Delta(x) = (T,L) \ \Rightarrow\ \eta(x) = \eta'(x)$, かつ

(2) $\eta(exception) \in LLoc$ or $\eta'(exception) \in LLoc \ \Rightarrow$
$\eta(exception) = \eta'(exception)$

□

定義 5 (H-confined method environment)　メソッド環境 μ によるメソッドの実行前後でヒープが L 区別不可能であるとき，$Hconf\ \mu$ と表記する．

$Hconf\ \mu$ **iff** $\mu(C,m)(h,\eta) \neq \bot \ \Rightarrow\ h \sim h_0$

ただし，$smtype(m,C) = \overline{x:\tau} \xrightarrow{H} \tau; \overline{\tau_e}$ である任意の C，m について，$(h_0,d,e) = \mu(C,m)(h,\eta)$ である．　□

補題 6 (H-confinement of commands)　$Hconf\ \mu$ のとき，文の実行前後におけるヒープとストアは以下を満たす．

$\forall \mu, h, \eta.Hconf\ \mu,\ \Delta \vdash S:(\kappa_1,\kappa_2,\kappa_3)$,
$(h_0,\eta_0) = [\![\Delta^\dagger \vdash S^\dagger]\!]\mu(h,\eta),\ (h_0,\eta_0) \neq \bot \Rightarrow$

(1) $\kappa_1 = H \Rightarrow \eta \sim_\Delta \eta_0$, かつ
(2) $\kappa_2 = H \Rightarrow h \sim h_0$ □

補題 7 (H-confinement of method environments) 型付けできるクラステーブルの意味 $lub\ \mu$ は $Hconf(lub\ \mu)$ を満たす．また $lub\ \mu$ を構成する各 μ_i について $Hconf\ \mu_i$． □

補題 8 (H-exception) $\forall \mu, h, \eta.\ \Delta \vdash S : (\kappa_1, \kappa_2, \kappa_3)$, $(h_0, \eta_0) = [\![\Delta^\dagger \vdash S^\dagger]\!]\mu(h, \eta)$, $level(\eta_0(exception)) = H \Rightarrow \kappa_3 = H$ □

プログラム中の各メソッド実行の前後でヒープとストアが L 区別不可能のままであるとき，そのプログラムは定義 3 を満たす．

定義 9 (Safe method environment) 以下を満たすときメソッド環境 μ が安全であるといい $safe\ \mu$ と表記する．

$safe\ \mu$ **iff** $\forall C, m, h, h', \eta, \eta'$.
$h \sim h'$, $\eta \sim_\Delta \eta'$, $(h_0, d, e) \neq\!\perp\!\neq (h'_0, d', e') \Rightarrow$
(1) $h_0 \sim h'_0$，かつ
(2) $\kappa = L \Rightarrow d = d' \wedge \bigsqcup \overline{\kappa_e} = L$，かつ
(3) $level(e) = L$ または $level(e') = L \Rightarrow e = e'$

ただし
$$\begin{cases} \overline{x : (T, \kappa)} \xrightarrow{\kappa_h} (T, \kappa); \overline{(E, \kappa_e)} = smtype(m, C) \\ \overline{x : (T, \kappa)}, this : (C, level(C)) = \Delta \\ (h_0, d, e) = \mu(C, m)(h, \eta) \\ (h'_0, d', e') = \mu(C, m)(h', \eta') \end{cases}$$
□

補題 10 (Safe expressions) L の式の値は一意である．
$\forall h, h', \eta, \eta'.\Delta \vdash e : (T, L)$, $h \sim h'$, $\eta \sim_\Delta \eta'$,
$d \neq\!\perp\!\neq d' \Rightarrow d = d'$
ただし，$d = [\![\Delta^\dagger \vdash e : T]\!](h, \eta)$, $d' = [\![\Delta^\dagger \vdash e : T]\!](h', \eta')$ である． □

補題 11 (Safe commands) 各文の実行において L 区別不可能なヒープとストアの状態は保存される．
$\forall \mu, h, h', \eta, \eta'.safe\ \mu$, $Hconf\ \mu$, $\Delta \vdash S : (\kappa_1, \kappa_2, \kappa_3)$,
$h \sim h'$, $\eta \sim_\Delta \eta'$, $(h_0, \eta_0) \neq \perp \neq (h'_0, \eta'_0) \Rightarrow$
$h_0 \sim h'_0$, かつ $\eta_0 \sim_\Delta \eta'_0$
ただし，$(h_0, \eta_0) = [\![\Delta^\dagger \vdash S^\dagger]\!]\mu(h, \eta)$, $(h'_0, \eta'_0) = [\![\Delta^\dagger \vdash S^\dagger]\!]\mu(h', \eta')$ である． □

プログラムはクラステーブルから与えられる．以下の定理を満たすとき，プログラムは定義 3 を満たす．

定理 12 (Noninterfering programs) 型付けできるクラステーブルの意味 $lub\ \mu$ は $safe\ (lub\ \mu)$ を満たす．

証明 定義 2 より $\exists j.\ \mu_0 \preceq \mu_1 \preceq \cdots \preceq \mu_j = \mu_{j+1} = \cdots = lub\ \mu$ である．よって $\forall i.\ safe\ \mu_i$ を i に関する帰納法で示す．
• $i = 0$ のとき
定義 2 より $\mu_0(C, m)(h, \eta) = \perp$ である．よって $safe\ \mu_0$.
• $i = j$ のとき
m が継承されているか否かで場合分けを行う．本稿では m が C で宣言されている場合についてのみ示す．$\mu_{j+1}(C, m) = [\![M]\!]\mu_j$, $M = (T, \kappa)\ m(\overline{(T, \kappa)\ x})\ \kappa_h$ throws $\overline{(E, \kappa_e)}$ $\{S\}$, $\Delta = \overline{x : (T, \kappa)}$, $this : (C, level(C))$, $result : (T, \kappa)$, $h \sim h'$, $\eta \sim_\Delta \eta'$, $\mu_{j+1}(C, m)(h, \eta) \neq\!\perp\!\neq \mu_{j+1}(C, m)(h', \eta')$ とする．
(1) $h_0 \sim h'_0$ を示す．
メソッド宣言の意味論より $\eta_1 = [\eta \mid result \mapsto default]$,
$(h_0, \eta_0) = [\![\Delta^\dagger \vdash S^\dagger]\!]\mu_j(h, \eta_1)$ とおける．よって
$\mu_{j+1}(C, m)(h, \eta) = (h_0, \eta_0(result), \eta_0(exception))$
$\mu_{j+1}(C, m)(h', \eta') = (h'_0, \eta'_0(result), \eta'_0(exception))$.
$\eta \sim_\Delta \eta'$ より $\eta_1 \sim_\Delta \eta'_1$ ($\because default$ の値は一意)．よって $Hconf\ \mu_j$, $safe\ \mu_j$ から補題 11 より $h_0 \sim h'_0$, $\eta_0 \sim_\Delta \eta'_0$
(2) $\kappa = L \Rightarrow d = d' \wedge \bigsqcup \overline{\kappa_e} = L$ を示す．
$\kappa = L$ とすると $\eta_0 \sim_\Delta \eta'_0$ より $d = \eta_0(result) = \eta'_0(result) = d'$．このとき MDEC2 規則から $\bigsqcup \overline{\kappa_e} = L$ を満たす。
(3) $level(e) = L$ or $level(e') = L \Rightarrow e = e'$ を示す．
$e \in LLoc$, または $e' \in LLoc$ より $\eta_0 \sim_\Delta \eta'_0$ から $e = e'$.
以上より $safe\ \mu_{j+1}$ □

7. おわりに

本稿では例外処理を持つオブジェクト指向言語を対象とした情報流解析に基づく安全型システムを提案した．また，提案した型システムが非干渉性について健全であることを証明した．本型システムで型付けできるプログラムは，機密データを外部に漏洩しない安全なプログラムである．

しかし本型システムは制約が強い．例えば，メソッド実行するレシーバオブジェクトの機密度が H の時，そのメソッド実行でスローする例外の機密度は CALL 規則より H でなければならない．そのため有用でかつ型付けできるプログラムを記述することは容易ではない．今後の課題は型システムの非干渉性に対する健全性を保ちつつ制約の緩和を行うことである．

謝辞 本研究は一部，科研費 ♯16300005 の補助を受けている．

文　献

[1] D.E.Denning, “A Lattice Model of Secure Information Flow”, Communications of the ACM, Vol.19, No.5, pp.236–243, 1976.
[2] Dennis Volpano, Geoffrey Smith, and Cynthia Irvine, “A Sound Type System for Secure Flow Analysis”, Journal of Computer Security, Vol.4, No.3, pp.167–187, 1996.
[3] Andrew C. Myers, “JFlow:Practical Mostly-Sftatic Information Flow Control”, In Proceedings of the 26th ACM Symposium on Principles of Programming Languages, pp.228–241, 1999.
[4] Anindya Banerjee, David A. Naumann, “Secure Information Flow and Pointer Confinement in a Java-like Language”, IEEE Computer Security Foundations Workshop, pp.253–270, 2002.
[5] Anindya Banerjee, David A. Naumann, “Using Access Control for Secure Information Flow in a Java-like Language”, IEEE Computer Security Foundations Workshop, pp.155–169, 2003.
[6] Anindya Banerjee, David A. Naumann, “Stack-based Access Control for Secure Information Flow”, Journal of Functional Programming, Vol.15, No.2, pp.131–177, 2005.