型検査に基づく情報流解析における型エラースライシング

Type Error Slicing for Type-based Information Flow Analysis

桑原 寛明*

Summary. In this paper, we propose an algorithm of type error slicing for information flow analysis in object-oriented programs. Our slicing method is based on the existing one for functional programming languages. Type-based information flow analysis is useful to detect invalid information leaks. However, it is sometimes difficult to understand the reasons why type checking or inference fails. Program slicing helps programmers to specify the locations of cause of type error. We implemented our proposed algorithm as a prototype tool and apply to some simple programs.

1 はじめに

プログラムが機密情報を外部に漏らさないことを静的に検査するために，情報流解析と呼ばれるプログラム解析手法が提案，研究されている．著者らも，例外処理付きオブジェクト指向プログラムの情報流解析のための型システムを提案し，非干渉性に対して健全であることを示した [1]．型システムに基づく情報流解析では，データの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を構築する．これにより，情報流解析の問題は型推論の問題へと帰着され，型推論アルゴリズムが定義されればプログラム中に不正な情報流が存在しないか自動的に検査することができる．

型推論によってプログラム中に不正な情報流が存在するかどうか判定することはできるが，型推論に失敗することから不正な情報流の存在が判明した場合に，その情報流がプログラムのどこに存在し原因が何であるか突き止めることは容易ではない．どの型付け規則のどの条件が満たされないために型推論に失敗したのかわかっても，原因までは簡単にわからないことが多い．

本稿では，著者らによる例外処理付きオブジェクト指向プログラムの情報流解析のための型システムを対象として，不正な情報流の原因箇所の特定を支援するための型エラースライシングの手法を提案する．型エラースライシングは，型推論が失敗するために必要なプログラム断片のみをプログラムスライスとして抽出する手法である．型推論を制約条件の充足問題に帰着させる際，各制約条件とプログラムの構成要素の対応付けを行い，充足不能な極小部分集合に含まれる制約条件に対応付けられたプログラムの構成要素をスライスとして抽出する．型エラースライシングによって得られたプログラムスライス内の情報流のみを追跡すればよいため，不正な情報流の原因箇所の特定が行いやすくなる．

本稿の構成は以下の通りである．2 章で対象とする言語と情報流解析のための型システムについて述べる．3 章で型エラースライシングの手法を提案し，4 章で実装と簡単な適用例について述べる．5 章で関連研究を挙げ，6 章でまとめる．

2 対象言語と型システム

本稿では，[1] の例外処理付きオブジェクト指向言語と情報流解析のための型システムを対象とする．型エラースライシングに利用するため，一部の構成要素にプログラム中の位置を表すラベルを付加する．プログラム中に記述されるラベルはすべ

*Hiroaki Kuwabara, 立命館大学情報理工学部

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

図 1　対象言語の文法

て異なるものとし，ラベルの集合を L と書く．以下では，ラベルが重要でない場合は省略する．

対象言語の文法を図 1 に示す．T はデータの型，η はデータの機密度，l はラベルを表す．機密度 η は束 $(\mathcal{H}, \sqsubseteq)$ 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 は変数名，E は例外クラス名を表す．メソッドがスローする例外を表す記述 $\mathrm{throws}\ \overline{E}$ は $\overline{E}$ の長さが 0 であれば省略できる．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 $F \leq D$ なる F が存在して C の宣言が $\mathrm{class}\ C\ \mathrm{extends}\ F\ \{\ldots\}$

すべての例外クラス E は $E \leq Exception$ を満たす．

クラス定義，メソッド定義，文に対する型付け規則を図 2 に示す．図中ではラベルは省略している．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 *result* はメソッドの返り値を表す変数である．*level* はクラスの機密度を取得する関数，*smtype* はメソッドのシグネチャを取得する関数，*sfields* はクラスが持つフィールドの宣言の集合を取得する関数である．

文の型判定式 $\Delta \vdash S : (\eta_s, \eta_h, P)$ は，型環境 Δ のもとで文 S が以下の条件を満たすことを表す．ここで，(η_s, η_h, P) を文の型と呼ぶ．型環境 Δ は変数名からその型 τ への関数である．

1. S で代入されるローカル変数やパラメータの機密度が η_s 以上．
2. S の実行によるヒープエフェクトが η_h 以上．
3. S の実行でスローされ得る例外の集合が P．
4. S 内の情報流が安全である．つまり，x から y への情報流が存在する場合，x の機密度 η_x と y の機密度 η_y は $\eta_x \sqsubseteq \eta_y$ を満たす．

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などによって変更されるヒープ上のデータの機密度のことである．本稿の対象言語の意味論ではオブジェクトはヒープ上に生成される．意味論の詳細は [1,2] を参照されたい．

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 [2] と同様である．

$$\frac{\begin{array}{l} level(D) \sqsubseteq \eta \quad C\ \eta \text{ extends } D \vdash M \text{ for each } M \in \overline{M} \\ level(D) \neq \eta \Rightarrow \text{ every } m \text{ with } smtype(m, D) \text{ defined is overridden in } C \end{array}}{\vdash C\ \eta \text{ extends } D\ \{\overline{\tau\ f};\ \overline{M}\}}\ [\text{CDEC}]$$

$$\frac{\begin{array}{l} \overline{x : (T_x, \eta_x)},\ this : (C, \eta_c),\ result : (T_r, \eta_r) \vdash B : (\eta_s, \eta'_h, \overline{E}) \\ smtype(m, D) \text{ is defined } \Rightarrow smtype(m, D) = \overline{x : (T_x, \eta_x)} \xrightarrow{\eta_h} (T_r, \eta_r); \overline{E} \\ \eta_h \sqsubseteq \eta'_h \end{array}}{C\ \eta_c \text{ extends } D \vdash (T_r, \eta_r)\ m(\overline{(T_x, \eta_x)\ x})\ \eta_h \text{ throws } \overline{E}\ B}\ [\text{MDEC}]$$

$$\frac{\begin{array}{l} \Delta,\ \overline{x : (T_x, \eta_x)} \vdash S_i : (\eta_{s_i}, \eta_{h_i}, P_i) \\ \eta_s \sqsubseteq \eta_{s_i} \quad \eta_h \sqsubseteq \eta_{h_i} \quad \forall E \in P_i.\ \forall j > i.\ level(E) \sqsubseteq \eta_{s_j}, level(E) \sqsubseteq \eta_{h_j} \\ i \in \{1, \ldots, n\} \quad P = \bigcup_i P_i \end{array}}{\Delta \vdash \{\overline{(T_x, \eta_x)\ x};\ S_1; \ldots S_n;\} : (\eta_s, \eta_h, P)}\ [\text{BLOCK}]$$

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

$$\frac{\begin{array}{l} \Delta \vdash e_1 : (T_1, \eta_1) \quad \Delta \vdash e_2 : (T_2, \eta_2) \quad (T_f, \eta_f)\ f \in sfields(T_1) \\ T_2 \leq T_f \quad \eta_1 \sqsubseteq \eta_f \quad \eta_2 \sqsubseteq \eta_f \quad \eta_h \sqsubseteq \eta_f \end{array}}{\Delta \vdash e_1.f = e_2 : (\eta_s, \eta_h, \emptyset)}\ [\text{ASSIGN2}]$$

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

$$\frac{\begin{array}{l} \Delta \vdash x : (T_x, \eta_x) \quad \Delta \vdash e : (T_e, \eta_e) \quad \Delta \vdash e_i : (T_{e_i}, \eta_{e_i}) \\ y_1 : (T_{y_1}, \eta_{y_1}), \ldots, y_n : (T_{y_n}, \eta_{y_n}) \xrightarrow{\eta'_h} (T_r, \eta_r); E_1, \ldots, E_k = smtype(m, T_e) \\ \forall i \in \{1, \ldots, n\}.\ T_{e_i} \leq T_{y_i}, \eta_{e_i} \sqsubseteq \eta_{y_i} \quad T_r \leq T_x \quad \eta_e \sqsubseteq \eta_x \quad \eta_r \sqsubseteq \eta_x \quad \eta_s \sqsubseteq \eta_x \\ \eta_e \sqsubseteq \eta'_h \quad \eta_h \sqsubseteq \eta'_h \quad P = \bigcup_i E_i \quad \forall E \in P.\ \eta_x \sqsubseteq level(E) \end{array}}{\Delta \vdash x = e.m(e_1, \ldots, e_n) : (\eta_s, \eta_h, P)}\ [\text{CALL}]$$

$$\frac{E \leq Exception \quad \eta_s \sqsubseteq level(E) \quad \eta_h \sqsubseteq level(E)}{\Delta \vdash \text{throw new } E : (\eta_s, \eta_h, \{E\})}\ [\text{THROW}]$$

$$\frac{\begin{array}{l} \Delta \vdash e : (Bool, \eta_e) \quad \Delta \vdash B : (\eta_1, \eta_2, P) \quad \Delta \vdash B' : (\eta'_1, \eta'_2, P') \\ \eta_e \sqsubseteq \eta_s \quad \eta_e \sqsubseteq \eta_h \quad \eta_s \sqsubseteq \eta_1 \quad \eta_s \sqsubseteq \eta'_1 \quad \eta_h \sqsubseteq \eta_2 \quad \eta_h \sqsubseteq \eta'_2 \end{array}}{\Delta \vdash \text{if } (e)\ B \text{ else } B' : (\eta_s, \eta_h, P \cup P')}\ [\text{IF}]$$

$$\frac{\begin{array}{l} \Delta \vdash B : (\eta'_s, \eta'_h, P') \quad \Delta, x : (E_i, level(E_i)) \vdash B_i : (\eta_{s_i}, \eta_{h_i}, P_i) \\ E_i \leq Exception \quad i \neq j \Rightarrow E_i \neq E_j \\ \eta_s \sqsubseteq \eta'_s \quad \eta_s \sqsubseteq \eta_{s_i} \quad \eta_h \sqsubseteq \eta'_h \quad \eta_h \sqsubseteq \eta_{h_i} \quad level(E_i) \sqsubseteq \eta_{s_i} \quad level(E_i) \sqsubseteq \eta_{h_i} \\ i \in \{1, \ldots, n\} \quad P = (P' - \bigcup_i E_i) \cup \bigcup_i P_i \end{array}}{\Delta \vdash \text{try } B\ \text{catch}(E_1\ x_1)\ B_1\ \ldots \text{catch}(E_n\ x_n)\ B_n : (\eta_s, \eta_h, P)}\ [\text{CATCH}]$$

図 2　型付け規則

3　型エラースライシング

本章では，機密度に関する条件が満たされないために型推論に失敗した場合に，その原因箇所を突き止めるためにプログラムをスライスする手法を提案する．簡単のために，対象のプログラムはデータ型に関しては正しく型付けできるものとする．

このことは図 2 の各規則から機密度に関する条件を除いて得られる型付け規則によって判定できる．また，図 2 の CDEC 規則から型推論は各メソッド定義に対して独立に行うことができるため，型エラースライシングも各メソッド定義ごとに行うものとする．

提案手法は，Haack らによる関数型言語を対象とした型エラースライシングの手法 [3] に基づいており，以下の手順からなる．

1. プログラムが型付け可能であるために満たすべき制約条件の集合を求める．この時，各制約条件に対しその条件が由来するプログラムの構成要素のラベルを与えることで制約条件と構成要素の対応付けを行う．
2. 制約条件集合が充足可能であれば型付け可能と判定する．充足不能であれば充足不能な極小部分集合を求める．
3. 充足不能な極小部分集合に含まれる各制約条件に付けられたラベルに対応する構成要素をプログラムスライスとして抽出する．

3.1 制約条件集合の生成

制約条件集合を生成するアルゴリズムを図 3，図 4 に示す．図中の κ や添字付きの κ_s, κ_h などは機密度を表す変数である．C-CONST 規則に現れる *typeof* はデータ型を取得する関数である．アルゴリズムはプログラム中の各文や式に対して新しく機密度変数を用意し，図 2 の規則に従って機密度に関する制約条件の集合を生成する．$\Delta \vdash S^l : (\kappa_s, \kappa_h, \mathcal{E}) \parallel \mathsf{C}$ は文 S^l が型環境 Δ のもとで型付け可能であるための制約条件の集合が C であることを表す．式についても同様である．メソッド定義に C-MDEC 規則を適用することで得られる制約条件集合 Cs が充足可能であればそのメソッド定義は型付け可能である．

メソッド定義に対する C-MDEC 規則やブロックに対する C-BLOCK 規則では，引数やローカル変数の型情報が追加された型環境のもとで文や式などそれぞれの構成要素に対して制約条件を生成する．この時，型情報の一部としてプログラム中で機密度に付けられたラベルも一緒に記録する．ラベルを変数宣言のプログラム中での位置とみなすことで，変数と機密度と位置情報を結びつけることができる．メソッド定義，ブロック，文，一部の式にもプログラム中でラベルが付けられており，同じラベルを適切な規則により新たに生成される制約条件に付けることで，各制約条件がどの構成要素に由来するか特定することができる．例えば，文 S^l に対する制約条件に $\kappa_1 \overset{l}{\sqsubseteq} \kappa_2$ のようにラベル l を付けることで，この制約条件がラベル l を持つ文 S^l に由来することがわかる．変数参照式に対する C-VAR 規則においてラベル l は変数参照式の位置，l' は変数宣言の位置，η は変数の機密度であり，$\kappa \overset{l}{\sqsubseteq} \eta^{l'}$ のように制約条件に出現する機密度に宣言位置を表すラベル l' が付けられていることで後述するスライシングにおいて変数参照とともに変数宣言も抽出することが可能となる．

文の型 $(\kappa_s, \kappa_h, \mathcal{E})$ の $\mathcal{E}$ は文の実行中にスローされる可能性のある例外のクラス E とその機密度を表す変数 κ_E の組 (E, κ_E) の集合である．機密度変数の追加は例外をスローする文をスライスとして抽出するためである．例えば，$\{\ldots; S_1^{l_1}; S_2^{l_2}; \ldots\}^{l_0}$ というプログラムがあり $S_1^{l_1}$ が例外をスローする可能性のあるメソッド呼出しであるとする．$S_1^{l_1}$ の型を $(\kappa_{s_1}, \kappa_{h_1}, \{(E, \kappa_E)\})$ とすると $S_1^{l_1}$ に対する制約条件集合に $level(E) \overset{l_1}{\sqsubseteq} \kappa_E$ が含まれる．また，$S_2^{l_2}$ の型を $(\kappa_{s_2}, \kappa_{h_2}, \mathcal{E}_2)$ とすると $\{\ldots\}^{l_0}$ に対する制約条件集合に $\kappa_E \overset{l_0}{\sqsubseteq} \kappa_{h_2}$ が含まれる．この 2 つの制約条件によって，ラベル l_1 を持つメソッド呼出し文もスライスとして抽出することができる．κ_E の値は $level(E)$ と等価であるため，2 つの制約条件の代わりに単に $level(E) \overset{l_0}{\sqsubseteq} \kappa_{h_2}$ だけを用いることも型推論としては可能であるが，この場合はラベル l_1 を持つ制約条件がないため

$$\frac{\overline{x : (T_x, \eta_x^{l_x})},\ this : (C, \eta_c^{l_c}),\ result : (T_r, \eta_r^{l_r}) \vdash B : (\kappa_s, \kappa_h, \mathcal{E}) \parallel \mathsf{C}}{\begin{array}{l} C\ \eta_c^{l_c}\ \text{extends}\ D \vdash (T_r, \eta_r^{l_r})\ m^l(\overline{(T_x, \eta_x^{l_x})\ x})\ \eta_h^{l_h}\ \text{throws}\ \overline{E}\ B \parallel \mathsf{Cs} \\ \text{where}\ \mathsf{Cs} = \mathsf{C} \cup \{\eta_h^{l_h} \overset{l}{\sqsubseteq} \kappa_h\} \end{array}}\ [\text{C-MDEC}]$$

$$\frac{\Delta,\ \overline{x : (T_x, \eta_x^{l_x})} \vdash S_i : (\kappa_{s_i}, \kappa_{h_i}, \mathcal{E}_i) \parallel \mathsf{C}_i \quad i \in \{1, \ldots, n\}}{\begin{array}{l} \Delta \vdash \{\overline{(T_x, \eta_x^{l_x})\ x};\ S_1; \ldots S_n;\}^l : (\kappa_s, \kappa_h, \bigcup_i \mathcal{E}_i) \parallel \mathsf{C} \\ \text{where}\ \mathsf{C} = \bigcup_i \mathsf{C}_i \cup \bigcup_i \{\kappa_s \overset{l}{\sqsubseteq} \kappa_{s_i}, \kappa_h \overset{l}{\sqsubseteq} \kappa_{h_i}\} \\ \qquad \cup \bigcup_i \bigcup_{(E,\kappa) \in \mathcal{E}_i} \bigcup_{j \in \{i+1, \ldots, n\}} \{\kappa \overset{l}{\sqsubseteq} \kappa_{s_j}, \kappa \overset{l}{\sqsubseteq} \kappa_{h_j}\} \end{array}}\ [\text{C-BLOCK}]$$

$$\frac{\Delta \vdash x : (T_x, \kappa_x) \parallel \mathsf{C_x} \quad \Delta \vdash e : (T_e, \kappa_e) \parallel \mathsf{C_e}}{\Delta \vdash x \overset{l}{=} e : (\kappa_s, \kappa_h, \emptyset) \parallel \mathsf{C_x} \cup \mathsf{C_e} \cup \{\kappa_e \overset{l}{\sqsubseteq} \kappa_x, \kappa_s \overset{l}{\sqsubseteq} \kappa_x\}}\ [\text{C-ASSIGN1}]$$

$$\frac{\Delta \vdash e_1 : (T_1, \kappa_1) \parallel \mathsf{C_1} \quad \Delta \vdash e_2 : (T_2, \kappa_2) \parallel \mathsf{C_2} \quad (T_f, \eta_f^{l_f})\ f \in \mathit{sfields}(T_1)}{\Delta \vdash e_1.f \overset{l}{=} e_2 : (\kappa_s, \kappa_h, \emptyset) \parallel \mathsf{C_1} \cup \mathsf{C_2} \cup \{\kappa_1 \overset{l}{\sqsubseteq} \eta_f^{l_f}, \kappa_2 \overset{l}{\sqsubseteq} \eta_f^{l_f}, \kappa_h \overset{l}{\sqsubseteq} \eta_f^{l_f}\}}\ [\text{C-ASSIGN2}]$$

$$\frac{\Delta \vdash x : (T_x, \kappa_x) \parallel \mathsf{C_x}}{\Delta \vdash x \overset{l}{=} \text{new}\ C : (\kappa_s, \kappa_h, \emptyset) \parallel \mathsf{C_x} \cup \{\mathit{level}(C) \overset{l}{\sqsubseteq} \kappa_x, \kappa_s \overset{l}{\sqsubseteq} \kappa_x, \kappa_h \overset{l}{\sqsubseteq} \mathit{level}(C)\}}\ [\text{C-NEW}]$$

$$\frac{\begin{array}{l} y_1 : (T_{y_1}, \eta_{y_1}^{l_{y_1}}), \ldots, y_n : (T_{y_n}, \eta_{y_n}^{l_{y_n}}) \xrightarrow{\eta_h^{l_h}} (T_r, \eta_r^{l_r}); E_1, \ldots, E_k = \mathit{smtype}(m, T_e) \\ \Delta \vdash x : (T_x, \kappa_x) \parallel \mathsf{C_x} \quad \Delta \vdash e : (T_e, \kappa_e) \parallel \mathsf{C_e} \\ \Delta \vdash e_i : (T_{e_i}, \kappa_{e_i}) \parallel \mathsf{C}_i \quad i \in \{1, \ldots, n\} \end{array}}{\begin{array}{l} \Delta \vdash x \overset{l}{=} e.m(e_1, \ldots, e_n) : (\kappa_s, \kappa_h, \{(E_1, \kappa_1), \ldots, (E_k, \kappa_k)\}) \parallel \mathsf{C} \\ \text{where}\ \mathsf{C} = \mathsf{C_x} \cup \mathsf{C_e} \cup \bigcup_{i \in \{1, \ldots, n\}} \mathsf{C}_i \cup \bigcup_{i \in \{1, \ldots, n\}} \{\kappa_{e_i} \overset{l}{\sqsubseteq} \eta_{y_i}^{l_{y_i}}\} \\ \qquad \cup \{\kappa_e \overset{l}{\sqsubseteq} \kappa_x, \eta_r^{l_r} \overset{l}{\sqsubseteq} \kappa_x, \kappa_s \overset{l}{\sqsubseteq} \kappa_x, \kappa_e \overset{l}{\sqsubseteq} \eta_h^{l_h}, \kappa_h \overset{l}{\sqsubseteq} \eta_h^{l_h}\} \\ \qquad \cup \bigcup_{i \in \{1, \ldots, k\}} \{\kappa_x \overset{l}{\sqsubseteq} \kappa_i, \kappa_i \overset{l}{\sqsubseteq} \mathit{level}(E_i), \mathit{level}(E_i) \overset{l}{\sqsubseteq} \kappa_i\} \end{array}}\ [\text{C-CALL}]$$

$$\frac{}{\begin{array}{l} \Delta \vdash \text{throw}^l\ \text{new}\ E : (\kappa_s, \kappa_h, \{(E, \kappa_E)\}) \parallel \mathsf{C} \\ \text{where}\ \mathsf{C} = \{\kappa_s \overset{l}{\sqsubseteq} \kappa_E, \kappa_h \overset{l}{\sqsubseteq} \kappa_E, \kappa_E \overset{l}{\sqsubseteq} \mathit{level}(E), \mathit{level}(E) \overset{l}{\sqsubseteq} \kappa_E\} \end{array}}\ [\text{C-THROW}]$$

$$\frac{\Delta \vdash e : (\text{Bool}, \kappa_e) \parallel \mathsf{C_e} \quad \Delta \vdash B : (\kappa_1, \kappa_2, \mathcal{E}) \parallel \mathsf{C_t} \quad \Delta \vdash B' : (\kappa_1', \kappa_2', \mathcal{E}') \parallel \mathsf{C_f}}{\begin{array}{l} \Delta \vdash \text{if}^l\ (e)\ B\ \text{else}\ B' : (\kappa_s, \kappa_h, \mathcal{E} \cup \mathcal{E}') \parallel \mathsf{C} \\ \text{where}\ \mathsf{C} = \mathsf{C_e} \cup \mathsf{C_t} \cup \mathsf{C_f} \\ \qquad \cup \{\kappa_e \overset{l}{\sqsubseteq} \kappa_s, \kappa_e \overset{l}{\sqsubseteq} \kappa_h, \kappa_s \overset{l}{\sqsubseteq} \kappa_1, \kappa_s \overset{l}{\sqsubseteq} \kappa_1', \kappa_h \overset{l}{\sqsubseteq} \kappa_2, \kappa_h \overset{l}{\sqsubseteq} \kappa_2'\} \end{array}}\ [\text{C-IF}]$$

$$\frac{\begin{array}{l} \Delta \vdash B : (\kappa_s', \kappa_h', \mathcal{E}') \parallel \mathsf{C}' \\ \Delta, x : (E_i, \mathit{level}(E_i)) \vdash B_i : (\kappa_{s_i}, \kappa_{h_i}, \mathcal{E}_i) \parallel \mathsf{C}_i \quad i \in \{1, \ldots, n\} \\ \mathcal{E} = \{(E, \kappa) \mid (E, \kappa) \in \mathcal{E}' \wedge E \notin \{E_1, \ldots, E_n\}\} \cup \bigcup_i \mathcal{E}_i \end{array}}{\begin{array}{l} \Delta \vdash \text{try}^l\ B\ \text{catch}(E_1\ x_1)\ B_1\ \ldots \text{catch}(E_n\ x_n)\ B_n : (\kappa_s, \kappa_h, \mathcal{E}) \parallel \mathsf{C} \\ \text{where}\ \mathsf{C} = \mathsf{C}' \cup \{\kappa_s \overset{l}{\sqsubseteq} \kappa_s', \kappa_h \overset{l}{\sqsubseteq} \kappa_h'\} \cup \bigcup_i \mathsf{C}_i \\ \qquad \cup \bigcup_i \{\kappa_s \overset{l}{\sqsubseteq} \kappa_{s_i}, \kappa_h \overset{l}{\sqsubseteq} \kappa_{h_i}, \mathit{level}(E_i) \overset{l}{\sqsubseteq} \kappa_{s_i}, \mathit{level}(E_i) \overset{l}{\sqsubseteq} \kappa_{h_i}\} \end{array}}\ [\text{C-CATCH}]$$

図 3　メソッド定義，ブロック，文に対する制約条件集合生成アルゴリズム

$$\frac{(T,\eta^{l'}) = \Delta(x)}{\Delta \vdash x^{l} : (T,\kappa) \parallel \{\eta^{l'} \overset{l}{\sqsubseteq} \kappa, \kappa \overset{l}{\sqsubseteq} \eta^{l'}\}}\ \text{[C-VAR]}$$

$$\frac{(T,\eta^{l'}) = \Delta(this)}{\Delta \vdash this^{l} : (T,\kappa) \parallel \{\eta^{l'} \overset{l}{\sqsubseteq} \kappa, \kappa \overset{l}{\sqsubseteq} \eta^{l'}\}}\ \text{[C-THIS]}$$

$$\frac{c \in \{\text{null,true,false,it}\}}{\Delta \vdash c : (\mathit{typeof}(c),\kappa) \parallel \emptyset}\ \text{[C-CONST]}$$

$$\frac{\Delta \vdash e_1 : (T_1,\kappa_1) \parallel \mathsf{C_1} \quad \Delta \vdash e_2 : (T_2,\kappa_2) \parallel \mathsf{C_2}}{\Delta \vdash e_1 \overset{l}{=\!=} e_2 : (\text{Bool},\kappa) \parallel \mathsf{C_1} \cup \mathsf{C_2} \cup \{\kappa_1 \overset{l}{\sqsubseteq} \kappa, \kappa_2 \overset{l}{\sqsubseteq} \kappa\}}\ \text{[C-COMP]}$$

$$\frac{\Delta \vdash e : (T_e,\kappa_e) \parallel \mathsf{C} \quad (T_f,\eta_f^{l'})\ f \in \mathit{sfields}(T_e)}{\Delta \vdash e.f^{l} : (T_f,\kappa) \parallel \mathsf{C} \cup \{\kappa_e \overset{l}{\sqsubseteq} \kappa, \eta_f^{l'} \overset{l}{\sqsubseteq} \kappa\}}\ \text{[C-FIELD]}$$

$$\frac{\Delta \vdash e : (T_e,\kappa_e) \parallel \mathsf{C_e}}{\Delta \vdash (C)e : (C,\kappa_e) \parallel \mathsf{C_e}}\ \text{[C-CAST]}$$

$$\frac{\Delta \vdash e : (T_e,\kappa_e) \parallel \mathsf{C_e}}{\Delta \vdash e \text{ is } C : (\text{Bool},\kappa_e) \parallel \mathsf{C_e}}\ \text{[C-IS]}$$

図 4　式に対する制約条件集合生成アルゴリズム

メソッド呼出し文をスライスして抽出できない．

3.2　充足可能性判定と充足不能集合の極小化

制約条件 $\kappa_1 \sqsubseteq \kappa_2$ の両辺が取りうる値は束 $(\mathcal{H}, \sqsubseteq)$ の要素であるため，図 3，図 4 のアルゴリズムで得られた制約条件集合の充足可能性は標準的な制約解消アルゴリズムによって判定することができる [4,5]．

もし制約条件集合が充足不能であれば充足不能な極小部分集合を求める．これにより型付けに失敗する原因となった制約条件のみを抽出し無関係な条件を除去できる．極小部分集合に含まれる制約条件に対応する構成要素が型付けの失敗，つまり不正な情報流に関係しているとみなす．制約条件集合の充足不能な極小部分集合は，[3] や [6] で提案されているアルゴリズムによって求めることができる．

図 3，図 4 のアルゴリズムで得られる制約条件集合内の各制約条件と一部の制約条件に含まれる機密度はラベルを持つが，ラベルは制約条件や機密度をプログラムの構成要素と関連付けるためのものであり機密度の大きさや順序関係には影響しない．そのため，充足可能性の判定や充足不能な極小部分集合を求める際には各ラベルを無視して行う．

3.3　スライシング

充足不能な極小部分集合に含まれるそれぞれの制約条件と機密度に付けられたラベルを抽出し，抽出されたラベルの集合を利用してプログラムをスライスする．

プログラムをスライスするアルゴリズムを図 5 に示す．図の上から順にプログラム全体，クラス定義，メソッド定義，ブロック，変数宣言それぞれに対し適用され，残りは各種の文に適用される．式はスライスの対象としない．アルゴリズムの入力

$$
\begin{array}{l}
\mathit{Slice}(CL_1\ \ldots\ CL_n, L_c) \\
\quad = \ \textbf{let}\ CL'_1 = \mathit{Slice}(CL_1, L_c)\ \textbf{in}\ \ldots\ \textbf{let}\ CL'_n = \mathit{Slice}(CL_n, L_c)\ \textbf{in} \\
\qquad CL'_1\ \ldots\ CL'_n
\end{array}
$$

$$
\begin{array}{l}
\mathit{Slice}(\text{class } C\ \eta^l \text{ extends } D\ \{\tau_1\ f_1; \ldots \tau_m\ f_m;\ M_1\ \ldots\ M_n\}, L_c) \\
\quad = \ \textbf{let}\ L_s = \mathit{labels}(\text{class } C\ \eta^l \text{ extends } D\ \{\tau_1\ f_1; \ldots \tau_m\ f_m;\ M_1\ \ldots\ M_n\})\ \textbf{in} \\
\qquad \textbf{if}\ L_c \cap L_s \neq \emptyset\ \textbf{then} \\
\qquad\quad \textbf{let}\ F'_1 = \mathit{Slice}(\tau_1\ f_1, L_c)\ \textbf{in}\ \ldots\ \textbf{let}\ F'_m = \mathit{Slice}(\tau_m\ f_m, L_c)\ \textbf{in} \\
\qquad\quad \textbf{let}\ M'_1 = \mathit{Slice}(M_1, L_c)\ \textbf{in}\ \ldots\ \textbf{let}\ M'_n = \mathit{Slice}(M_n, L_c)\ \textbf{in} \\
\qquad\quad \text{class } C\ \eta \text{ extends } D\ \{F'_1; \ldots; F'_m; M_1\ \ldots\ M_n\}
\end{array}
$$

$$
\begin{array}{l}
\mathit{Slice}(\tau\ m^l(\tau_1\ x_1, \ldots, \tau_n\ x_n)\ \eta^{l'} \text{ throws } \overline{E}\ B, L_c) \\
\quad = \ \textbf{let}\ L_s = \mathit{labels}(\tau\ m^l(\tau_1\ x_1, \ldots, \tau_n,\ x_n)\ \eta^{l'} \text{ throws } \overline{E}\ B)\ \textbf{in} \\
\qquad \textbf{if}\ L_c \cap L_s \neq \emptyset\ \textbf{then} \\
\qquad\quad \textbf{let}\ V'_1 = \mathit{Slice}(\tau_1\ x_1, L_c)\ \textbf{in}\ \ldots\ \textbf{let}\ V'_n = \mathit{Slice}(\tau_n\ x_n, L_c)\ \textbf{in} \\
\qquad\quad \textbf{let}\ B' = \mathit{Slice}(B, L_c)\ \textbf{in} \\
\qquad\quad \tau\ m(V_1, \ldots, V_n)\ \eta \text{ throws } \overline{E}\ B'
\end{array}
$$

$$
\begin{array}{l}
\mathit{Slice}(\{\tau_1\ x_1; \ldots \tau_m\ x_m; S_1; \ldots S_n;\}^l, L_c) \\
\quad = \ \textbf{let}\ V'_1 = \mathit{Slice}(\tau_1\ x_1, L_c)\ \textbf{in}\ \ldots\ \textbf{let}\ V'_m = \mathit{Slice}(\tau_m\ x_m, L_c)\ \textbf{in} \\
\qquad \textbf{let}\ S'_1 = \mathit{Slice}(S_1, L_c)\ \textbf{in}\ \ldots\ \textbf{let}\ S'_n = \mathit{Slice}(S_n, L_c)\ \textbf{in} \\
\qquad \{V'_1; \ldots; V'_m; S'_1; \ldots; S'_n\}
\end{array}
$$

$$
\mathit{Slice}((T, \eta^l)\ x, L_c) \ = \ \textbf{if}\ l \in L_c\ \textbf{then}\ (T, \eta)\ x
$$

$$
\mathit{Slice}(x \overset{l}{=} e, L_c) \ = \ \textbf{if}\ l \in L_c\ \textbf{then}\ x = e
$$

$$
\mathit{Slice}(e_1.f \overset{l}{=} e_2, L_c) \ = \ \textbf{if}\ l \in L_c\ \textbf{then}\ e_1.f = e_2
$$

$$
\mathit{Slice}(x \overset{l}{=} \text{new } C, L_c) \ = \ \textbf{if}\ l \in L_c\ \textbf{then}\ x = \text{new } C
$$

$$
\mathit{Slice}(x \overset{l}{=} e.m(\overline{e}), L_c) \ = \ \textbf{if}\ l \in L_c\ \textbf{then}\ x = e.m(\overline{e})
$$

$$
\mathit{Slice}(\text{throw}^l \text{ new } E, L_c) \ = \ \textbf{if}\ l \in L_c\ \textbf{then}\ \text{throw new } E
$$

$$
\begin{array}{l}
\mathit{Slice}(\text{if}^l\ (e)\ B_t \text{ else } B_f, L_c) \\
\quad = \ \textbf{let}\ L_s = \mathit{labels}(\text{if}^l\ (e)\ B_t \text{ else } B_f)\ \textbf{in} \\
\qquad \textbf{if}\ L_c \cap L_s \neq \emptyset\ \textbf{then} \\
\qquad\quad \textbf{let}\ B'_t = \mathit{Slice}(B_t, L_c)\ \textbf{in let}\ B'_f = \mathit{Slice}(B_f, L_c)\ \textbf{in} \\
\qquad\quad \text{if } (e)\ B'_t \text{ else } B'_f
\end{array}
$$

$$
\begin{array}{l}
\mathit{Slice}(\text{try}^l\ B\ \text{catch}(E_1\ x_1)\ B_1 \ldots \text{catch}(E_n\ x_n)\ B_n, L_c) \\
\quad = \ \textbf{let}\ L_s = \mathit{labels}(\text{try}^l\ B\ \text{catch}(E_1\ x_1)\ B_1 \ldots \text{catch}(E_n\ x_n)\ B_n)\ \textbf{in} \\
\qquad \textbf{if}\ L_c \cap L_s \neq \emptyset\ \textbf{then} \\
\qquad\quad \textbf{let}\ B' = \mathit{Slice}(B, L_c)\ \textbf{in} \\
\qquad\quad \textbf{let}\ B'_1 = \mathit{Slice}(B_1, L_c)\ \textbf{in}\ \ldots\ \textbf{let}\ B'_n = \mathit{Slice}(B_n, L_c)\ \textbf{in} \\
\qquad\quad \text{try } B'\ \text{catch}(E_1\ x_1)\ B'_1 \ldots \text{catch}(E_n\ x_n)\ B'_n
\end{array}
$$

図 5　スライシングのアルゴリズム

はプログラム $\overline{CL}$ とラベル集合 L_c であり，出力としてスライスされたプログラムが得られる．スライスされたプログラムには機密度に関する条件を満たさない原因となった部分が残される．図中の *labels* はプログラム断片中のすべてのラベルを抽出する関数である．

フィールド，仮引数，ローカル変数の宣言と代入文，throw 文はラベルを一つだ

け持っており，そのラベルが L_c に含まれていれば残す．if 文と try 文についてはその一部を構成するすべての文に対してスライスを適用する．ただし，L_c に含まれるラベルが if 文あるいは try 文内に全く出現しない場合は if 文あるいは try 文全体を除去する．プロックに関しても同様である．

3.4 例

以下のプログラム断片に対するスライシングの例を示す．

```
{
   (Bool,H1) x; (Bool,H2) y; (Bool,L3) z;
   if4 (x6 ==5 true) {
      y9 =8 true;  z11 =10 true;
   }7 else {
      x14 =13 true;
   }12
}0
```

ここで，L と H は $L \sqsubseteq H$ なる機密度であり，0 から 14 までの数はラベルである．このプログラムは，機密度が高い変数 x を条件とする if 文の then 節で機密度の低い変数 z に代入が行われており，if 文の型付け規則に違反している．スライシングによって z への代入に関連する箇所が切り出されることを確認する．

if 文に対する制約条件集合は型環境

$$\Delta = x : (\mathrm{Bool}, H^1),\ y : (\mathrm{Bool}, H^2),\ z : (\mathrm{Bool}, L^3)$$

のもとで生成される．ラベル 11 の変数 z に対して，C-VAR 規則から z の型を $(\mathrm{Bool}, \kappa_{11})$ として $\Delta(z) = (\mathrm{Bool}, L^3)$ であるので制約条件集合

$$\mathsf{C}_{11} = \{\kappa_{11} \overset{11}{\sqsubseteq} L^3, L^3 \overset{11}{\sqsubseteq} \kappa_{11}\}$$

が得られる．ラベル 10 の代入に対して，C-ASSIGN1 規則から代入文の型を $(\kappa_{s_{10}}, \kappa_{h_{10}}, \emptyset)$ として

$$\mathsf{C}_{10} = \mathsf{C}_{11} \cup \{\kappa_{t_2} \overset{10}{\sqsubseteq} \kappa_{11}, \kappa_{s_{10}} \overset{10}{\sqsubseteq} \kappa_{11}\}$$

が得られる．ここで κ_{t_2} は右辺の定数 true に対して C-CONST 規則によって与えられる機密度変数とする．ラベル 8 の代入文に対しても同様に制約条件集合を生成できて，その集合を C_8 とする．ラベル 7 のブロックに対して，C-BLOCK 規則からブロックの型を $(\kappa_{s_7}, \kappa_{h_7}, \emptyset)$ として

$$\mathsf{C}_7 = \mathsf{C}_8 \cup \mathsf{C}_{10} \cup \{\kappa_{s_7} \overset{7}{\sqsubseteq} \kappa_{s_8}, \kappa_{h_7} \overset{7}{\sqsubseteq} \kappa_{h_8}, \kappa_{s_7} \overset{7}{\sqsubseteq} \kappa_{s_{10}}, \kappa_{h_7} \overset{7}{\sqsubseteq} \kappa_{h_{10}}\}$$

が得られる．ラベル 12 のブロックに対しても同様にして制約条件集合を生成できて，その集合を C_{12} とする．ラベル 5 の比較式に対しては，左辺に対する制約条件集合が x の機密度変数を κ_6 とすると $\{\kappa_6 \overset{6}{\sqsubseteq} H^1, H^1 \overset{6}{\sqsubseteq} \kappa_6\}$ であるので，C-COMP 規則から式の機密度変数を κ_5 として

$$\mathsf{C}_5 = \{\kappa_6 \overset{6}{\sqsubseteq} H^1, H^1 \overset{6}{\sqsubseteq} \kappa_6\} \cup \{\kappa_6 \overset{5}{\sqsubseteq} \kappa_5, \kappa_{t_4} \overset{5}{\sqsubseteq} \kappa_5\}$$

が得られる．κ_{t_4} は右辺の true の機密度変数である．ラベル 4 の if 文に対しては C-IF 規則から if 文の型を $(\kappa_{s_4}, \kappa_{h_4}, \emptyset)$ として

$$\begin{aligned}\mathsf{C}_4 \ = \ & \mathsf{C}_5 \cup \mathsf{C}_7 \cup \mathsf{C}_{12} \cup \\ & \{\kappa_5 \overset{4}{\sqsubseteq} \kappa_{s_4}, \kappa_5 \overset{4}{\sqsubseteq} \kappa_{h_4}, \kappa_{s_4} \overset{4}{\sqsubseteq} \kappa_{s_7}, \kappa_{s_4} \overset{4}{\sqsubseteq} \kappa_{s_{12}}, \kappa_{h_4} \overset{4}{\sqsubseteq} \kappa_{h_7}, \kappa_{h_4} \overset{4}{\sqsubseteq} \kappa_{h_{12}}\}\end{aligned}$$

が得られる．

ラベル 0 のブロックに対しても同様にして C-BLOCK 規則から制約条件集合が得られる．この集合はラベル 4 の if 文に対する制約条件集合 C_4 を含むが，C_4 は充足

不能である[1]．充足不能な極小部分集合を計算すると次の集合が得られる．

$$\{H^1 \overset{6}{\sqsubseteq} \kappa_6, \kappa_6 \overset{5}{\sqsubseteq} \kappa_5, \kappa_5 \overset{4}{\sqsubseteq} \kappa_{s_4}, \kappa_{s_4} \overset{4}{\sqsubseteq} \kappa_{s_7}, \kappa_{s_7} \overset{7}{\sqsubseteq} \kappa_{s_{10}}, \kappa_{s_{10}} \overset{10}{\sqsubseteq} \kappa_{11}, \kappa_{11} \overset{11}{\sqsubseteq} L^3\}$$

この充足不能な極小部分集合に含まれるラベルは $\{1, 3, 4, 5, 6, 7, 10, 11\}$ であり，これらのラベルを持つ構成要素を抽出して以下のようなスライスが得られる．

```
{
   (Bool,H) x;              (Bool,L) z;
   if (x == true) {
                  z = true;
   } else {
   }
}
```

ここから，if 文の then 節における変数 z への代入が型付けに失敗する原因に関連すること，変数 z への代入自体は正当なため if 文の条件式が関連すること，変数 z の機密度が変数 x の機密度より低いことがわかり，そのために if 文の型付け規則に違反することが理解できる．

4 実装

提案手法を実装し簡単なプログラムに適用した．制約条件集合の生成およびスライシングの実装には CASE ツール・プラットフォーム **Sapid** を利用した．充足可能性の判定には既存の制約解消系 Cream [7] を利用し，充足不能な極小部分集合は [6] のアルゴリズムを利用して求めている．ユーザインタフェースには Web ブラウザを利用した．

対象言語は機密度を除けば Java 言語のサブセットとなっている．Java 言語として **Sapid** を用いて解析するため機密度はコメントとして特定の位置に記述することとした．また，**Sapid** の解析器によって各構成要素に与えられる ID を制約条件と構成要素を対応付けるためのラベルとして用いる．制約条件集合の生成とスライシングは図 3，図 4，図 5 のアルゴリズムをそのまま実装した．

制約条件集合の充足可能性は，制約解消系 Cream が解を発見できたか否かによって判定する．解が発見された場合は充足可能と判定して得られた解を出力する．この解は型推論の結果になっている．解が発見されない場合は充足不能と判定し，充足不能な極小部分集合を求めてスライシングを実行する．Cream は制約プログラミングのためのライブラリであり，数種類の制約解消のアルゴリズムを抽象化して提供している．標準で用意されているライブラリは整数上の制約条件のみを対象としているが，簡単な拡張によって機密度束上の制約条件を扱うことができる．

今回の実装では，制約条件集合の充足不能かつ極小の部分集合を以下のアルゴリズム [6] によって求める．入力は制約条件集合 $\{c_1, \ldots, c_n\}$，出力は充足不能な極小部分集合 C である．

$c_{n+1} \leftarrow \mathit{false}$; $C \leftarrow \emptyset$;
while C is consistent **do**
  $i \leftarrow 0$; $D \leftarrow C$;
  while D is consistent **do**
    $i \leftarrow i + 1$; $D \leftarrow D \cup \{c_i\}$;
  end while;
  $C \leftarrow C \cup \{c_i\}$;
end while;
if $i = n + 1$ **then** $C \leftarrow \emptyset$ **fi**

[1] ここでは C_8 と C_{12} の要素を明示していないが $\mathsf{C}_4 - (\mathsf{C}_8 \cup \mathsf{C}_{12})$ が充足不能である．

図 6　適用例

ある制約条件集合に対して充足不能な極小部分集合が複数存在する可能性があるが，今回の実装ではそのうちの一つのみを求めている．

簡単なプログラムに提案手法を適用した例を図 6 に示す．この例では機密度は $L \sqsubseteq H$ をなす L, H の 2 段階とし，図の Echo クラスの echo メソッドに適用した．read は外部からの入力を読み込むメソッドであるが実装は省略した．print は引数を出力するメソッドであり，ここでは機密度が L のフィールド output への代入を出力とみなして実装した．Echo 以外のクラスの実装も省略している．図の左側がプログラムの内容とスライスの結果である．プログラム中の影が付けられている箇所がスライスによって除去された箇所を示している．図では省略されているが他に String クラスと LogWriter クラスの定義があり，いずれもスライスによって除去されている．図の右側は制約条件の充足不能な極小部分集合であり，一行で一つの制約条件を表している．制約条件中の下線付きの先頭に小文字の “s” が付けられた 10 桁の数字は **Sapid** の解析器がプログラムの構成要素に付加する ID を表しており，マウスポインタをのせると対応する構成要素がハイライトされる．

制約条件は基本的には $\{ID_0\}([v_1, ID_1, sort_1] < [v_2, ID_2, sort_2])$ の形で表示される．ID_0 はこの制約条件を持つ構成要素の ID である．v_1 は制約条件の左辺であり，L あるいは H ならば定数，$*$ ならば変数である．ID_1 は左辺の機密度が導入される構成要素の ID を表し，$sort_1$ は左辺の機密度の種別を表す．制約条件の右辺も同様である．種別は，DECLARED はプログラム中で宣言されていること，HEAP はヒープエフェクト，STORE はストアへのエフェクト，EXCEPTION はスローされる例外の機密度であることを意味している．EXCEPTION の場合，例外は複数スローされる可能性があるのでそれらを区別するためにさらにインデックスが付けられる．例えば，図 6 の一番上の制約条件`{s5947523081}([H, s5888802820,`

DECLARED] < [*, s5947523088, HEAP]) は，ID がs5947523081 の構成要素の制約条件であり，機密度変数 κ に対して $H \sqsubseteq \kappa$ であり，左辺の機密度は ID がs5888802820 の構成要素で宣言されており，右辺の機密度変数は ID がs5947523088 の構成要素のヒープエフェクトであることを表している．s5947523081 は try 文，s5888802820 は例外クラス IOException の宣言，s5947523088 は catch ブロックの ID である．制約条件の連続する 2 行は上の行の右辺と下の行の左辺が同じ機密度変数となるように並べられており，制約条件の依存関係の追跡を容易にしている．

図 6 のプログラムの echo メソッドは，read メソッドで読み込んだ入力を print メソッドで出力する．この時，入力文字列長が 0 であればそのことをログに出力するが，ログ出力を秘密にすることを意図して関連するクラスなどの機密度を H としている．しかし，ログ出力において例外 IOException が発生した際に print メソッドを呼び出してしまっており，その出力から例外の発生がわかるため秘密になっていない．この例では出力は機密度が L のフィールドへの代入であるので，機密度が H である IOException の発生という情報が機密度が L のフィールドへ流れていることになり，この情報流は不正である．プログラムスライスとして，例外クラスの定義，catch 節，print メソッドの呼び出し，print メソッドの定義が抽出されており，無関係な try 節の中や変数は除去されている．また，print メソッドの定義についても引数は無関係であるため除去されている．

5 関連研究

型推論の失敗に対してその原因箇所の特定や修正に有用なエラーレポートの生成に関する研究は多数行われている．特に情報流解析に関する研究として，King らは Java 言語に情報流制御とアクセス制御を追加して拡張した Jif [8] を対象として不正な情報流に対してエラーレポートを生成する手法を提案している [5]．彼らの手法では [4] のアルゴリズムを拡張し，制約解消を行う際にそれぞれの制約条件がどのように他の制約条件に波及していくか記録をとる．充足不能な場合に成立しなかった制約条件を起点に記録をさかのぼることで充足不能に関係している制約条件を特定する．King らと同様の手法を手続き型言語における情報流解析に適用した研究として [9] がある．本稿の提案手法では，制約条件集合に対して充足不能な極小部分集合を求めることで原因箇所を特定する．充足不能な極小部分集合は既存手法で求めることが可能であり，制約解消のアルゴリズムを変更する必要がないため既存の制約解消系が利用できる．そのため全体としての実現が容易になっている．また，King らの手法では充足不能な制約条件の元となったコード断片を単に列挙するだけであるが，提案手法ではプログラムスライシングによってエラーの原因箇所を抽出する．抽出された結果は元のプログラムの構造を保っており，関連する変数の宣言箇所も含むためエラーの原因がわかりやすい．

型エラースライシングに関しては，手続き型言語のデータ型を対象とした [10] や関数型言語のデータ型を対象とした [3] の他に，並行プログラムのデッドロック解析のための型システムを対象とした研究 [11] も行われている．本稿では，オブジェクト指向プログラムにおける情報流解析のための型システムを対象とした型エラースライシングの手法を提案した．提案手法は例外処理に伴う情報流解析にも対応しており，そのような情報流が不正な場合に例外を考慮したスライシングを行うことができる．

6 おわりに

本稿では，例外処理付きオブジェクト指向言語を対象とする情報流解析のための型システムにおいて，型推論に失敗した場合，つまり不正な情報流が見つかった場合にその原因箇所の特定を支援するための型エラースライシングの手法を提案した．型推論を機密度に関する制約条件集合の制約解消として行い，充足不能な場合に充

足不能な極小部分集合を求める．制約条件集合を求める際に各制約条件とプログラムの構成要素を対応付けておくことで，充足不能な極小部分集合に含まれる制約条件に対応する構成要素を不正な情報流の原因を示すプログラムスライスとして抽出する．これによりプログラム中の何が不正な情報流の原因になっているか絞り込むことができる．

今後の課題として，スライシングをより細かい粒度で行うことが挙げられる．提案手法では文の粒度でスライスしているが，式の粒度でスライスすることで不要なプログラム断片をさらに除去できると考えられる．その際，提案手法のみによるのではなく，従来の制御依存やデータ依存によるスライシング手法と組み合わせてスライスすることも考えられる．

本稿では，制約条件集合の解として得られた機密度変数の値割り当てによって元のプログラムが型付けできること，型付け可能なプログラムから得られた制約条件集合が解を持つこと，充足不能な極小部分集合に基づいて得られたプログラムスライスが元のプログラムと同じ理由で型付け不能であることを示していない．これらの証明を与えることは今後の課題である．

提案手法のプロトタイプ実装ではスライシングの結果と制約条件の充足不能な極小部分集合を単純に表示するだけであるが，極小部分集合に含まれる各制約条件がどのような意味かといった情報もエラーの原因を理解するためには重要である．エラーレポート生成の技術と組み合わせて，わかりやすいユーザインタフェースを提供することも今後の課題である．Eclipse ベースの Jif 開発環境である Jifclipse [12] のようなユーザインタフェースが望ましいと考えている．

謝辞 有益なコメントを頂いた査読者の皆様および名古屋大学大学院情報科学研究科 小林隆志准教授，今井敬吾氏に感謝いたします．

参考文献

[1] 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.

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

[3] Christian Haack and J.B. Wells. Type Error Slicing in Implicitly Typed Higher-Order Languages. *Science of Computer Programming*, Vol. 50, No. 1-3, pp. 189–224, 2004.

[4] Jakob Rehof and Torben Æ. Mogensen. Tractable Constraints in Finite Semilattices. *Science of Computer Programming*, Vol. 35, No. 2, pp. 191–221, 1999.

[5] Dave King, Trent Jaeger, Somesh Jha, and Sanjit A. Seshia. Effective Blame for Information-Flow Violations. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 250–260. ACM Press, 2008.

[6] Yasuhiro Ajiro and Kazunori Ueda. Kima: an Automated Error Correction System for Concurrent Logic Programs. *Automated Software Engineering*, Vol. 9, No. 1, pp. 67–94, 2002.

[7] Cream: Class Library for Constraint Programming in Java. `http://bach.istc.kobe-u.ac.jp/cream/`.

[8] Jif: Java + information flow. `http://www.cs.cornell.edu/jif/`.

[9] Zhenyue Deng and Geoffrey Smith. Type Inference and Informative Error Reporting for Secure Information Flow. In *ACM-SE 44*, pp. 543–548. ACM Press, 2006.

[10] F. Tip and T.B. Dinesh. A Slicing-Based Approach for Locating Type Errors. *ACM TOSEM*, Vol. 10, No. 1, pp. 5–55, 2001.

[11] 飯村枝里, 小林直樹, 末永幸平. 型エラースライシングによるデッドロックの原因特定. 情報処理学会 プログラミング, Vol. 1, No. 2, pp. 71–84, 2008.

[12] Boniface Hicks, Dave King, and Patrick McDaniel. Jifclipse: Development Tools for Security-Typed Applications. In *PLAS'07*, pp. 1–10. ACM Press, 2007.