情報流解析における Declassifier の配置手法

A Method of Declassifiers Placement in Information Flow Analysis

桑原 寛明* 國枝 義敏†

Summary. In this paper, we propose a method to produce suggestions of declassifiers placement for correcting type errors in information flow analysis. Though the declassification is an useful method to resolve illegal information flow, it can be difficult and time-consuming task to manually place declassifiers. By our method, developers only need to select a declassifiers placement from suggestions for correcting errors. We proved that produced suggestions can resolve type errors and at least one suggestion can be produced, and implemented proposed method as a prototype tool.

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プログラムを静的に解析して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．非干渉性は，代入などの直接的な依存だけでなく，機密度の高いデータに基づく分岐先で機密度の低いデータを変更するといった間接的な依存も認めない．機密データ自体に加えて機密データを推測できる情報も一切漏らさないという意味で非干渉性はよい性質であるが，機密データの内容によって外部から観察可能な動作を変更することが許されないため，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．例えば，一般的なパスワード認証では入力と正解パスワードの比較結果に応じて異なる動作が観察されるため，正解パスワードを機密データとして扱うと非干渉性は満たされない．しかし，正解パスワードを機密データとしたまま，入力と正解の比較結果については機密度を下げるような仕組みがあれば，非干渉性を満たすパスワード認証を実現できる．

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 Declassification（非機密化）と呼ばれるアプローチが多く提案されている [5] [6] [7] [8]．例えば，Andrei らは式 e の機密度を η とみなすことを意味する式レベルの構文要素 `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，機密度の高い側から低い側への情報流を理解して認めていることを型システムに示し，型システムは開発者の理解を踏まえて型検査を行う．Declassification 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が，Andrei らの `declassify` 式のような Declassifiers を手作業で適切に記述することはそれほど容易ではない．一般的に，あるプログラムに含まれる不正な情報流は一通りではなく，複数の不正な情報流それぞれに対処する必要がある．

本稿では，不正な情報流を含むため型検査に失敗するプログラムに対して，型検査を成功させるために必要な Declassifiers の配置箇所の候補を挙げる手法を提案する．本稿では，Declassifier として Andrei らの提案と同等な `declassify` 式を用いる．提案手法により，開発者は挙げられた候補に従って `declassify` 式を記述することで型エラーを解消できる．情報流解析は型システムとして実現されるが，型検査の問題は制約充足問題に帰着させることが可能であり，不正な情報流を含むプロ

*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

$$
\begin{array}{rcl}
\eta & ::= & L \mid H \\
P & ::= & \overline{F} \\
F & ::= & \eta\ f(\overline{\eta\ x})\ B \\
B & ::= & \{\overline{\eta\ x;}\ \overline{S;}\} \\
S & ::= & x = e^l \mid \text{if } (e^l)\ B \text{ else } B \\
e & ::= & x \mid \text{true} \mid \text{false} \mid e^l == e^l \mid f(\overline{e^l}) \mid \texttt{declassify}(e)
\end{array}
$$

図 1　対象言語の文法

グラムから生成される制約集合は充足不能である．本手法は，充足不能な制約集合に対して Minimal Correction Subset(MCS) を求め，MCS に含まれる制約が由来するプログラムの構成要素を declassify 式の記述位置の候補とする．MCS 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である．declassify 式を適切に記述することで，充足可能な制約集合が生成される．

本稿の構成は以下の通りである．2 節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 節で declassify の配置箇所を求める手法を提案し，4 節で実装と簡単な適用例について述べる．5 節で関連研究を挙げ，6 節でまとめる．

2　対象言語

本稿では，[9] の簡単な手続き型言語と型システムの変種に declassify 式を追加して拡張した言語と型システムを対象とする．簡単のために機密度は L と H の二段階とし，$L \sqsubseteq L, L \sqsubseteq H, H \sqsubseteq H$ を満たす機密度束 $(\{L, H\}, \sqsubseteq)$ を仮定する．

対象言語の文法を図 1 に示す．η は機密度を表す．データ型を bool 型のみとすることでデータ型の記述を省略し，変数や関数の返り値の型として機密度のみを記述する．プログラム P は関数定義の並びである．$\overline{A}$ は長さ 0 以上の有限リストを表す略記である．F は関数定義であり，f が関数名を表す．B はブロック，S は文，e は式である．ブロックの先頭でローカル変数を宣言できる．関数の返り値は予約変数 $result$ への代入によって設定する．declassify(e) は e の機密度を L とみなしたいという開発者の意図を表し，e の機密度によらず declassify(e) の機密度は必ず L である．declassify(e) の意味は e と等価である．l は declassify を適用できる式のプログラム中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ものとする．以下ではラベルが重要でない場合は省略する．

型付け規則を図 2 に示す．図中ではラベルは省略されている．PROG 規則がプログラム全体，FDEC 規則が関数定義，BLOCK 規則がブロック，ASSIGN 規則，IF 規則が文，その他が式の型付け規則である．$result$ は関数の返り値を表す変数，$ftype$ は関数のシグネチャを取得する関数である．Δ は型環境であり，変数名からその機密度への関数である．ブロックあるいは文の型判定式 $\Delta \vdash S : \eta$ は，型環境 Δ のもとで S の実行によって変更される変数の機密度が η 以上であることを表す．式の型判定式 $\Delta \vdash e : \eta$ は，型環境 Δ のもとで式 e の機密度が η 以下であることを表す．

型検査は制約充足問題に帰着させることができる．型検査が成功するために満たされるべき制約集合を生成するアルゴリズムを図 3 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．その上で，図 2 の規則に従って機密度に関する制約集合を生成する．$\Delta \Vdash e : \kappa \parallel \mathsf{C}$ は型環境 Δ のもとで式 e が型付け可能であるために満たされるべき制約集合が 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．例えば，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．証明は省略するが，図 3 のアルゴリズムは健全かつ完全である．すなわち，プログラムの構

$$\frac{\vdash F_i \quad i \in \{1,\ldots,n\}}{\vdash F_1 \ \ldots \ F_n}\ [\text{PROG}] \qquad \frac{\overline{x : \eta_x}, result : \eta_r \vdash B : \eta_B}{\vdash \eta_r \ f(\overline{\eta_x \ x}) \ B}\ [\text{FDEC}]$$

$$\frac{\Delta, \overline{x : \eta_x} \vdash S_i : \eta_i \quad \eta \sqsubseteq \eta_i \quad i \in \{1,\ldots,n\}}{\Delta \vdash \{\overline{\eta_x \ x;} \ S_1; \ldots S_n;\} : \eta}\ [\text{BLOCK}]$$

$$\frac{\Delta \vdash e : \eta_e \quad \eta_x = \Delta(x) \quad \eta_e \sqsubseteq \eta_x \quad \eta \sqsubseteq \eta_x}{\Delta \vdash x = e : \eta}\ [\text{ASSIGN}]$$

$$\frac{\Delta \vdash e : \eta_e \quad \Delta \vdash B_t : \eta_t \quad \Delta \vdash B_f : \eta_f \quad \eta_e \sqsubseteq \eta \quad \eta \sqsubseteq \eta_t \quad \eta \sqsubseteq \eta_f}{\Delta \vdash \text{if } (e) \ B_t \text{ else } B_f : \eta}\ [\text{IF}]$$

$$\frac{\eta = \Delta(x)}{\Delta \vdash x : \eta}\ [\text{VAR}] \qquad \frac{\Delta \vdash e_1 : \eta_1 \quad \Delta \vdash e_2 : \eta_2 \quad \eta_1 \sqsubseteq \eta \quad \eta_2 \sqsubseteq \eta}{\Delta \vdash e_1 == e_2 : \eta}\ [\text{COMP}]$$

$$\frac{c \in \{\text{true,false}\}}{\Delta \vdash c : L}\ [\text{CONST}] \qquad \frac{\Delta \vdash e : \eta}{\Delta \vdash \texttt{declassify}(e) : L}\ [\text{DECL}]$$

$$\frac{\Delta \vdash e_i : \eta_{e_i} \quad \eta_{e_i} \sqsubseteq \eta_i \quad i \in \{1,\ldots,n\} \quad y_1 : \eta_1, \ldots, y_n : \eta_n \to \eta_r = \mathit{ftype}(f) \quad \eta_r \sqsubseteq \eta}{\Delta \vdash f(e_1,\ldots,e_n) : \eta}\ [\text{CALL}]$$

図 2　型付け規則

成要素 D に対して，生成される制約集合が充足可能であれば D は型付け可能であり，D が型付け可能であれば生成される制約集合は充足可能である．

3　`declassify` 配置箇所の候補

型検査に失敗したプログラムに対して，`declassify` 式を適切に記述することで型検査を成功させることができる．しかし，一般的に，型検査を失敗させる不正な情報流はプログラム中に複数存在し，さらに不正な情報流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は決まらない．そこで，本稿では `declassify` 式の適切な記述位置の候補を列挙する手法を提案する．実際に記述する箇所は，開発者が候補を比較して決定する．

3.1　準備

充足不能な制約集合には，以下に定義する二種類の部分集合が存在する．

定義 1. 制約集合 C の部分集合 $M \subseteq C$ について，$C \setminus M$ が充足可能であり，かつ，M に含まれる任意の制約 $c \in M$ に対して $C \setminus (M \setminus \{c\})$ が充足不能である時，M は Minimal Correction Subset(MCS) である．

定義 2. 制約集合 C の部分集合 $U \subseteq C$ について，U が充足不能であり，かつ，U に含まれる任意の制約 $c \in U$ に対して $U \setminus \{c\}$ が充足可能である時，U は充足不能な極小部分集合 (Minimal Unsatisfiable Subset, MUS) である．

元の制約集合から MCS に含まれる制約をすべて取り除いて得られる制約集合は充足可能であり，MCS はそのような条件を満たす極小の制約集合である．極小であるので，制約を一つでも削除すると条件は満たされない．MUS は，元の制約集合から充足不能となるように最小限必要な制約を選んで得られる制約集合である．一般的に，充足不能な制約集合に対し MCS と MUS のいずれも複数存在する．

$$\frac{\Vdash F_i \parallel \mathsf{C}_i \quad i \in \{1,\ldots,n\}}{\Vdash F_1 \ldots F_n \parallel \bigcup_i \mathsf{C}_i}\ [\text{C-PROG}] \qquad \frac{\overline{x : \eta_x}, result : \eta_r \Vdash B : \kappa_B \parallel \mathsf{C}}{\Vdash \eta_r\ f(\overline{\eta_x\ x})\ B \parallel \mathsf{C}}\ [\text{C-FDEC}]$$

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

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

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

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

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

$$\frac{\Delta \Vdash e_i : \kappa_i \parallel \mathsf{C_i} \quad i \in \{1,\ldots,n\} \quad y_1 : \eta_1, \ldots, y_n : \eta_n \to \eta_r = \mathit{ftype}(f)}{\Delta \Vdash f(e_1^{l_1}, \ldots, e_n^{l_n}) : \kappa \parallel \bigcup_i (\mathsf{C_i} \cup \{\kappa_i \sqsubseteq^{l_i} \eta_i\}) \cup \{\eta_r \sqsubseteq \kappa\}}\ [\text{C-CALL}]$$

$$\frac{\Delta \Vdash e : \kappa \parallel \mathsf{C}}{\Delta \Vdash \texttt{declassify}(e) : L \parallel \mathsf{C}}\ [\text{C-DECL}]$$

図 3　制約条件集合生成アルゴリズム

ある制約集合に対する MCS と MUS は hitting set によって結びつけられることが知られている [10] [11].

定義 3. 集合の集合 Ω に対して $D = \bigcup_{S \in \Omega} S$ とする．D の部分集合 $H \subseteq D$ について，Ω に含まれる任意の集合 $S \in \Omega$ に対して $H \cap S \neq \emptyset$ である時，H は Ω の hitting set である.

定理 1. 充足不能な制約集合 C に対して以下が成り立つ.

1. C の部分集合 $M \subseteq C$ が C のすべての MUS に対する極小な hitting set である時，かつその時に限り M は C の MCS である.
2. C の部分集合 $M \subseteq C$ が C のすべての MCS に対する極小な hitting set である時，かつその時に限り M は C の MUS である.

次に，制約の列を定義する．以下，α, α' や添字付きの α_i, α_e などは機密度定数 L, H あるいは機密度変数を表すメタ変数である.

定義 4. 制約の列は以下のように定義される制約の並びである.

1. $\alpha \sqsubseteq \alpha'$ は制約の列である.
2. 制約の列 $\alpha_1 \sqsubseteq \kappa_2, \ldots, \kappa_{n-1} \sqsubseteq \alpha_n$ に対し，α_1 が機密度変数ならば $\alpha_0 \sqsubseteq \alpha_1, \alpha_1 \sqsubseteq \kappa_2, \ldots, \kappa_{n-1} \sqsubseteq \alpha_n$ は制約の列であり，α_n が機密度変数ならば $\alpha_1 \sqsubseteq \kappa_2, \ldots, \kappa_{n-1} \sqsubseteq \alpha_n, \alpha_n \sqsubseteq \alpha_{n+1}$ は制約の列である.
3. 1. および 2. によるものだけが制約の列である.

以下では，制約の列を単に列とも呼び，$\alpha_1 \sqsubseteq \cdots \sqsubseteq \alpha_n$ を $\alpha_1 \sqsubseteq \kappa_2, \kappa_2 \sqsubseteq \kappa_3, \ldots, \kappa_{n-1} \sqsubseteq \alpha_n$ の略記とする．定義より，列 $\alpha_1 \sqsubseteq \cdots \sqsubseteq \alpha_n$ において両端の α_1 と α_n のみ L あるいは H となり得る．例えば，制約集合 $\{L \sqsubseteq \kappa, \kappa \sqsubseteq \kappa', \kappa' \sqsubseteq H\}$

に含まれる制約を用いて，$L \sqsubseteq \kappa \sqsubseteq \kappa'$ や $L \sqsubseteq \kappa \sqsubseteq \kappa' \sqsubseteq H$ といった列を構成できる．

3.2 手順

提案手法は，充足不能な制約集合は MCS を取り除けば充足できることを踏まえ，MCS に含まれる制約が由来する式に `declassify` を適用することで充足不能な制約集合の生成を抑制するというアイデアに基づいている．MCS が複数存在する場合，いずれか一つの MCS が解消されれば十分であるが，適切な MCS を機械的に選択することは難しい．そのため，提案手法では，`declassify` 式の記述によって解消できるすべての MCS について `declassify` 式の記述位置を列挙する．実際に解消する MCS は開発者が決定する．

提案手法は以下の手順からなる．

1. 図 3 のアルゴリズムで生成された制約集合が充足可能であれば不正な情報流は存在しないと判断する．充足不能であれば制約集合のすべての MCS を求める．
2. ラベル付き制約のみを含む MCS を抽出する．
3. 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 e 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．ここで，式 e が `declassify` 式の記述位置である，あるいは式 e の位置に `declassify` 式を記述するとは，式 e を $\texttt{declassify}(e)$ に変更することを指す．

図 2 の PROG 規則から型検査は各関数定義に対して独立に実行できるため，提案手法も関数定義を対象に適用するものとする．

3.3 充足可能性判定

制約 $\kappa_1 \sqsubseteq \kappa_2$ の両辺が取り得る値は束の要素であるため，図 3 のアルゴリズムで得られた制約集合の充足可能性は標準的な制約解消アルゴリズムを用いて判定できる [12]．一部の制約にはラベルが付けられているが，ラベルはプログラムと制約を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，充足可能性の判定はラベルを無視して行う．

3.4 MCS と記述位置候補

充足不能な制約集合に対してすべての MCS を求める必要があるが， [13] [10] [11] などで提案されているアルゴリズムを利用すれば求められる．

ラベル付き制約はラベルが付けられた式に由来し，ラベルが付けられた式は `declassify` を適用できる式であるため，MCS に含まれる制約からラベルを抽出すれば `declassify` 式の記述位置が得られる．ただし，制約集合にはラベルのない制約も含まれており，MCS がラベルのない制約を含む可能性を考慮する必要がある．ラベルのない制約は `declassify` 式の記述位置とはなり得ないプログラム中の構成要素に由来するため，ラベルのない制約を含む MCS を `declassify` 式の記述によって解消することはできない．そこで，本手法ではラベル付き制約のみを含む MCS のみを選択する．

ここで，図 3 のアルゴリズムで生成される制約集合が充足不能の場合，ラベル付き制約のみを含む MCS が少なくとも一つは存在することを示す．ここでは，定理 1 より MCS がすべての MUS の極小な hitting set であることに基づき，生成された制約集合のすべての MUS がラベル付き制約を一つ以上含むことを示す．

補題 1. 式 e に対し $\Delta \Vdash e : \alpha \parallel \mathsf{C}$ とする．この時，$\alpha \in \{L, H\}$ ならば $\mathsf{C} = \emptyset$ である．一方，α が機密度変数ならば，C に含まれる制約は $\eta \sqsubseteq \cdots \sqsubseteq \alpha$ なる列を構成できる．また，C に含まれる制約が $\eta' \sqsubseteq \cdots \sqsubseteq \eta''$ なる列を構成できる場合，その列はラベル付き制約を含む．ここで，$\eta, \eta', \eta'' \in \{L, H\}$ である．

証明： e の構造に関する帰納法による．

- x の場合，$\alpha = \Delta(x) \in \{L, H\}$ であり，$\mathsf{C} = \emptyset$.
- $\mathrm{true}, \mathrm{false}$ の場合，$\alpha = L$ であり，$\mathsf{C} = \emptyset$.

- $e_1^{l_1} == e_2^{l_2}$ の場合，$\Delta \Vdash e_1 : \alpha_1 \parallel \mathsf{C_1}$，$\Delta \Vdash e_2 : \alpha_2 \parallel \mathsf{C_2}$ とすると $\mathsf{C} = \mathsf{C_1} \cup \mathsf{C_2} \cup \{\alpha_1 \sqsubseteq^{l_1} \alpha, \alpha_2 \sqsubseteq^{l_2} \alpha\}$．ここで，C-COMP 規則より α は機密度変数である．$\alpha_1 \in \{L, H\}$ ならば $\eta \sqsubseteq^{l_1} \alpha \in \mathsf{C}$ である．α_1 が機密度変数ならば，帰納法の仮定より $\mathsf{C_1}$ に含まれる制約は $\eta \sqsubseteq \cdots \sqsubseteq \alpha_1$ なる列を構成可能であり，$\mathsf{C_1} \subset \mathsf{C}$ かつ $\alpha_1 \sqsubseteq^{l_1} \alpha \in \mathsf{C}$ であるため，C に含まれる制約は $\eta \sqsubseteq \cdots \sqsubseteq \alpha$ なる列を構成できる．C に含まれる制約が $\eta' \sqsubseteq \cdots \sqsubseteq \eta''$ なる列を構成できる場合，$\mathsf{C_1}$ に含まれる制約を構成する機密度変数の集合と $\mathsf{C_2}$ に含まれる制約を構成する機密度変数の集合は互いに素であり，α はいずれにも含まれないため，$\eta' \sqsubseteq \cdots \sqsubseteq \eta''$ なる列は $\mathsf{C_1}$ か $\mathsf{C_2}$ のいずれか一方に含まれる制約のみで構成される．この時，帰納法の仮定よりそのような列はラベル付き制約を含む．

他の場合も同様である． □

補題 2. 文，ブロック D に対し $\Delta \Vdash D : \kappa \parallel \mathsf{C}$ とする．この時，C に含まれる制約は $\kappa \sqsubseteq \cdots \sqsubseteq \eta$ なる列を構成できる．また，$\eta' \sqsubseteq \cdots \sqsubseteq \eta''$ なる列を構成できる場合，その列はラベル付き制約を含む．ここで，$\eta, \eta', \eta'' \in \{L, H\}$ である．

証明： D の構造に関する帰納法による． □

定理 2. 式，文，ブロック D に対し $\Delta \Vdash D : \alpha \parallel \mathsf{C}$ とする．この時，C の MUS はラベル付き制約を含む．

証明： MUS は充足不能であるので，MUS に含まれる制約は $H \sqsubseteq \cdots \sqsubseteq L$ なる列を構成する．そのような列は補題 1，2 よりラベル付き制約を含むため，MUS はラベル付き制約を含む． □

ラベル付き制約のみを含む MCS は複数存在する可能性があるため，ラベル付き制約のみを含むそれぞれの MCS について各制約のラベルを抽出し，それらのラベルが付けられた式の集合を `declassify` 式の記述位置候補の一つとする．例えば，ラベル付き制約のみを含む MCS が二つあり，それぞれから抽出されたラベルの集合が $\{l_1, l_2\}$ および $\{l_3, l_4, l_5\}$ とすると，`declassify` 式の記述位置候補として $\{e_1^{l_1}, e_2^{l_2}\}$ および $\{e_3^{l_3}, e_4^{l_4}, e_5^{l_5}\}$ の二種類の式の集合が得られる．

式 e を $\texttt{declassify}(e)$ に変更することで生成される制約集合の MCS 群が変化することは直感的には次のように理解できる．図 3 のアルゴリズムにおいて，ラベル l が付けられた式 e^l の機密度を表す変数を κ_e とすると，ラベル l を付けて生成される制約の形は $\kappa_e \sqsubseteq^l \alpha$ である．もし $\kappa_e \sqsubseteq^l \alpha$ が MUS に含まれるならば，その MUS に含まれる制約は $H \sqsubseteq \cdots \sqsubseteq \kappa'_e \sqsubseteq \kappa_e \sqsubseteq^l \alpha \sqsubseteq \cdots \sqsubseteq L$ なる列を構成する．この時，式 e^l が $\texttt{declassify}(e)^l$ に変更されると，C-DECL 規則によって $\kappa_e \sqsubseteq^l \alpha$ の代わりに $L \sqsubseteq^l \alpha$ が生成される．L は最小の機密度であるため，$L \sqsubseteq^l \alpha$ を含む MUS は存在せず，$L \sqsubseteq^l \alpha$ を含む MCS も存在しない．$\kappa_e \sqsubseteq^l \alpha$ を含んでいた MUS は $\kappa_e \sqsubseteq^l \alpha$ の代わりに $L \sqsubseteq^l \alpha$ を含むことになり，$H \sqsubseteq \cdots \sqsubseteq \kappa'_e$ や $L \sqsubseteq^l \alpha \sqsubseteq \cdots \sqsubseteq L$ といった列は構成できるが，$H \sqsubseteq \cdots \sqsubseteq L$ なる列は構成できず MUS ではなくなる．以上の直感的な議論に対し以下で証明を与える．

初めに，式 e^l を $\texttt{declassify}(e)^l$ に変更して制約集合を生成すると，変更前のプログラムから生成された制約集合においてラベル l が付けられた制約の左辺が L に変更された制約集合が生成されることを示す．

補題 3. 式，文，ブロック D に対し $\Delta \Vdash D : \alpha \parallel \mathsf{C}$ とする．この時，D 中に出現する各ラベルについて，そのラベルが付けられた制約は C にただ一つ含まれる．

証明： プログラム中の各ラベルは唯一であり，図 3 の C-ASSIGN 規則，C-IF 規則，C-COMP 規則，C-CALL 規則より明らか． □

補題 4. 式，文，ブロック D と D 中に出現する任意のラベル l に対し $\Delta \Vdash D : \alpha \parallel \mathsf{C} \cup \{\alpha_e \sqsubseteq^l \alpha'_e\}$ とする．この時，D 中の式 e^l を $\texttt{declassify}(e)^l$ に変更したものを D' とすると，$\Delta \Vdash D' : \alpha \parallel \mathsf{C} \cup \{L \sqsubseteq^l \alpha'_e\}$ である．

証明： D の構造に関する帰納法による．

- x, true, false の場合，生成される制約集合は空である．

- $e_1^{l_1} == e_2^{l_2}$ の場合，$\Delta \Vdash e_1 : \alpha_1 \parallel \mathsf{C_1}$，$\Delta \Vdash e_2 : \alpha_2 \parallel \mathsf{C_2}$ とすると $\Delta \Vdash D : \alpha \parallel \mathsf{C_1} \cup \mathsf{C_2} \cup \{\alpha_1 \sqsubseteq^{l_1} \alpha, \alpha_2 \sqsubseteq^{l_2} \alpha\}$．$e_1^{l_1}$ を $\texttt{declassify}(e_1)^{l_1}$ に変更する場合，D' は $\texttt{declassify}(e_1)^{l_1} == e_2^{l_2}$ であり，$\Delta \Vdash e_1 : \alpha_1 \parallel \mathsf{C_1}$ と C-DECL 規則より $\Delta \Vdash \texttt{declassify}(e_1) : L \parallel \mathsf{C_1}$ ゆえ，C-COMP 規則より $\Delta \Vdash D' : \alpha \parallel \mathsf{C_1} \cup \mathsf{C_2} \cup \{L \sqsubseteq^{l_1} \alpha, \alpha_2 \sqsubseteq^{l_2} \alpha\}$．$e_2^{l_2}$ を変更する場合も同様である．e_1 中に出現する e^l を変更する場合，補題 3 よりラベル l を持つ制約は $\mathsf{C_1}$ にただ一つ含まれるため，適当な制約集合 $\mathsf{C_1'}$ が存在して $\mathsf{C_1} = \mathsf{C_1'} \cup \{\alpha_e \sqsubseteq^l \alpha_e'\}$ である．e^l が $\texttt{declassify}(e)^l$ に変更された e_1 に対応する式と e_1' と書くと，帰納法の仮定より $\Delta \Vdash e_1' : \alpha_1 \parallel \mathsf{C_1'} \cup \{L \sqsubseteq^l \alpha_e'\}$ であり，D' は $e_1'^{l_1} == e_2^{l_2}$ であるので C-COMP 規則より $\Delta \Vdash D' : \alpha \parallel \mathsf{C_1'} \cup \{L \sqsubseteq^l \alpha_e'\} \cup \mathsf{C_2} \cup \{\alpha_1 \sqsubseteq^{l_1} \alpha, \alpha_2 \sqsubseteq^{l_2} \alpha\}$ である．e_2 中に出現する e^l を変更する場合も同様である．

他の場合も同様である．□

次に，制約集合 C から制約 $c \in C$ を一つ選んで c の左辺を L に変更した制約集合を C' とすると，C' の MUS の集合は，C の MUS のうち c を含まない MUS の集合に等しいことを示す．

補題 5. 制約集合 $C \cup \{\alpha \sqsubseteq \alpha'\}$ の MUS の中で，$\alpha \sqsubseteq \alpha'$ を含まない MUS の集合を U_e，含む MUS の集合を U_i とする．この時，制約集合 $C \cup \{L \sqsubseteq \alpha'\}$ の MUS の集合は U_e である．

証明： U_e に含まれる MUS は $\alpha \sqsubseteq \alpha'$ を含まないため，C の MUS である．一方，U_i に含まれる MUS は極小であり $\alpha \sqsubseteq \alpha'$ を含むため，C の MUS ではない．ここで，$L \sqsubseteq \alpha'$ は α' が L, H のいずれでも成り立ち，α' が機密度変数の場合は L, H のいずれを割り当てても成り立つため，$C \cup \{L \sqsubseteq \alpha'\}$ の MUS の集合は U_e である．□

以上より，生成された制約集合の MCS の中からラベル付き制約のみを含む MCS を選び，その MCS に含まれる各制約のラベルに対応する式それぞれに declassify を適用することで，不正な情報流を解消できることがわかる．

定理 3. 関数定義 F に対し $\Vdash F \parallel \mathsf{C}$ とする．C が充足不能の場合，C のラベル付き制約のみを含む MCS に含まれる制約のラベルを $\{l_1, \ldots, l_n\}$ とし，F 中の $e_i^{l_i}$ を $\texttt{declassify}(e_i)^{l_i}$ に変更した関数定義 F' について $\Vdash F' \parallel \mathsf{C'}$ とすると，$\mathsf{C'}$ は充足可能である．

証明： MCS を $M = \{\alpha_1 \sqsubseteq^{l_1} \alpha_1', \ldots, \alpha_n \sqsubseteq^{l_n} \alpha_n'\}$ とおくと，適当な $\mathsf{C_0}$ が存在して $\mathsf{C} = \mathsf{C_0} \cup M$ であり，M は C の MCS であるため $\mathsf{C_0}$ は充足可能である．$M' = \{L \sqsubseteq^{l_1} \alpha_1', \ldots, L \sqsubseteq^{l_n} \alpha_n'\}$ とすると，補題 4 より $\mathsf{C'} = \mathsf{C_0} \cup M'$ である．この時，$\mathsf{C_0}$ は充足可能であるため補題 5 より $\mathsf{C'}$ の MUS は存在せず，$\mathsf{C'}$ は充足可能である．□

4 実装と例

Java 言語を用いて提案手法を実装した．充足可能性判定には制約解消系 Cream [14] を利用し，MCS の集合は [10] のアルゴリズムによって求めている．今回の実装では，プログラム中の式と生成される制約を対応付けるラベルをプログラム中に記述するのではなく，構文解析時に自動的に与えることとした．

図 3 のアルゴリズムで生成された制約集合の解を Cream が発見できれば制約集合は充足可能であると判定する．解が発見できない場合は充足不能であると判定し，制約集合の MCS をすべて求めてからラベル付き制約のみを含む MCS を抽出し，各制約に付けられたラベルに基づいて declassify 式の記述位置候補を表示する．

例として，以下に示す関数 f の定義に対して提案手法を適用する．

```
L f() {
   H x; H y; L z;
   if (x5 ==7 true6) {
```

```
      y = true10;  z = true13;  z = g(z18)17;
   } else {
      x = true22;
   }
}
```

ここで，$ftype(g) = a : L \to H$ とする．5, 6, 7, 10, 13, 17, 18, 22 といった数はラベルである[1]．このプログラムでは，機密度の高い変数 x を条件とする if 文内での機密度の低い変数 z への代入（2 箇所）と，機密度の高い関数 g の返り値の機密度の低い変数 z への代入（1 箇所）が，型付け規則に反する不正な情報流となっている．

図 3 のアルゴリズムによって以下の制約集合が得られる．

$$\{H \sqsubseteq^5 \kappa_7, L \sqsubseteq^6 \kappa_7, L \sqsubseteq^{10} H, \kappa_{11} \sqsubseteq H, \kappa_8 \sqsubseteq \kappa_{11}, L \sqsubseteq^{13} L, \kappa_{14} \sqsubseteq L,$$
$$\kappa_8 \sqsubseteq \kappa_{14}, H \sqsubseteq \kappa_{17}, L \sqsubseteq^{18} L, \kappa_{17} \sqsubseteq^{17} L, \kappa_{19} \sqsubseteq L, \kappa_8 \sqsubseteq \kappa_{19}, L \sqsubseteq^{22} H,$$
$$\kappa_{23} \sqsubseteq H, \kappa_{20} \sqsubseteq \kappa_{23}, \kappa_7 \sqsubseteq^7 \kappa_{24}, \kappa_{24} \sqsubseteq \kappa_8, \kappa_{24} \sqsubseteq \kappa_{20}, \kappa_1 \sqsubseteq \kappa_{24}\}$$

この制約集合は充足不能であり，MCS を求めると以下に挙げる 14 通りの MCS が得られる．

$\{H \sqsubseteq \kappa_{17}, H \sqsubseteq^5 \kappa_7\}$　　$\{H \sqsubseteq \kappa_{17}, \kappa_7 \sqsubseteq^7 \kappa_{24}\}$
$\{H \sqsubseteq \kappa_{17}, \kappa_{24} \sqsubseteq \kappa_8\}$　　$\{H \sqsubseteq \kappa_{17}, \kappa_8 \sqsubseteq \kappa_{14}, \kappa_8 \sqsubseteq \kappa_{19}\}$
$\{H \sqsubseteq \kappa_{17}, \kappa_8 \sqsubseteq \kappa_{14}, \kappa_{19} \sqsubseteq L\}$　　$\{H \sqsubseteq \kappa_{17}, \kappa_{14} \sqsubseteq L, \kappa_8 \sqsubseteq \kappa_{19}\}$
$\{H \sqsubseteq \kappa_{17}, \kappa_{14} \sqsubseteq L, \kappa_{19} \sqsubseteq L\}$　　$\{\kappa_{17} \sqsubseteq^{17} L, H \sqsubseteq^5 \kappa_7\}$
$\{\kappa_{17} \sqsubseteq^{17} L, \kappa_7 \sqsubseteq^7 \kappa_{24}\}$　　$\{\kappa_{17} \sqsubseteq^{17} L, \kappa_{24} \sqsubseteq \kappa_8\}$
$\{\kappa_{17} \sqsubseteq^{17} L, \kappa_8 \sqsubseteq \kappa_{14}, \kappa_8 \sqsubseteq \kappa_{19}\}$　　$\{\kappa_{17} \sqsubseteq^{17} L, \kappa_8 \sqsubseteq \kappa_{14}, \kappa_{19} \sqsubseteq L\}$
$\{\kappa_{17} \sqsubseteq^{17} L, \kappa_{14} \sqsubseteq L, \kappa_8 \sqsubseteq \kappa_{19}\}$　　$\{\kappa_{17} \sqsubseteq^{17} L, \kappa_{14} \sqsubseteq L, \kappa_{19} \sqsubseteq L\}$

このうち，ラベル付き制約のみを含む MCS は $\{\kappa_{17} \sqsubseteq^{17} L, H \sqsubseteq^5 \kappa_7\}$ と $\{\kappa_{17} \sqsubseteq^{17} L, \kappa_7 \sqsubseteq^7 \kappa_{24}\}$ の二種類であり，含まれるラベルの集合はそれぞれ $\{5, 17\}$，$\{7, 17\}$ である．よって，declassify 式の記述位置候補として

1. if 文の条件における変数 x の参照式，および関数 g の呼び出し式
2. if 文の条件における比較式，および関数 g の呼び出し式

の二通りが挙げられる．

後者の候補に基づいて declassify 式を記述すると，以下の関数定義が得られる．

```
L f() {
   H x; H y; L z;
   if (declassify(x5 == true6)7) {
      y = true10;  z = true13;  z = declassify(g(z18))17;
   } else {
      x = true22;
   }
}
```

制約集合を生成すると

$$\{H \sqsubseteq^5 \kappa_7, L \sqsubseteq^6 \kappa_7, L \sqsubseteq^{10} H, \kappa_{11} \sqsubseteq H, \kappa_8 \sqsubseteq \kappa_{11}, L \sqsubseteq^{13} L, \kappa_{14} \sqsubseteq L,$$
$$\kappa_8 \sqsubseteq \kappa_{14}, H \sqsubseteq \kappa_{17}, L \sqsubseteq^{18} L, \boxed{L \sqsubseteq^{17} L}, \kappa_{19} \sqsubseteq L, \kappa_8 \sqsubseteq \kappa_{19}, L \sqsubseteq^{22} H,$$
$$\kappa_{23} \sqsubseteq H, \kappa_{20} \sqsubseteq \kappa_{23}, \boxed{L \sqsubseteq^7 \kappa_{24}}, \kappa_{24} \sqsubseteq \kappa_8, \kappa_{24} \sqsubseteq \kappa_{20}, \kappa_1 \sqsubseteq \kappa_{24}\}$$

が得られる．ここで，四角形で囲まれた二つの制約は，declassify 式を記述する前の関数定義に対する制約集合から変更があった制約を表している．この制約集合は充足可能である．元のプログラムには 3 種類の不正な情報流が存在していたが，

[1] 7 は比較式全体に対するラベルである．

そのうちの 2 種類は同一の if 文内における機密度の低い変数 z への代入が関係しており，条件式に対する declassify の適用によって同時に解決されている．

ここで挙げた適用例は，プロトタイプ実装を実際に動作させた結果を示している．Mac OS X 10.7.5，Core i7 1.8GHz，メモリ 4GB，Java1.6.0_51 の環境において，制約集合の生成に約 0.3 ミリ秒，充足可能性判定に約 0.4 ミリ秒，MCS の列挙に約 75 ミリ秒を要した．MCS の列挙が所要時間の大半を占めているが，プログラムの規模に対するスケーラビリティについては今後検討する必要がある．

5 関連研究

本稿と同様に，King らも不正な情報流が存在するプログラムに対して Declassifiers の記述位置を求める手法を提案している [15] [16]．彼らの手法では，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，制約中に出現する変数や定数を節として制約の左辺から右辺への有向辺をもつ有向グラフを制約集合から構築する．有向グラフに高い機密度から低い機密度へのパスが存在すればそれが不正な情報流に対応するため，そのようなパスの切断点に対応する式に対して Declassfier を記述する．本稿の提案手法では単に制約集合の MCS を求めればよく，グラフを構成する必要はない．Declassifiers の記述位置を求める能力については，高い機密度から低い機密度へのパスすべての切断点を求めることが制約集合の MCS を求めることに相当するため同等であると思われる．ただし，King らの手法で記述位置が必ず一つ以上見つかるかは明確ではない．

情報流解析を対象とする型エラースライシングが提案されている [9] [17]．これらの手法では，制約集合の MUS に基づいて不正な情報流に関係する構成要素をプログラムスライスとして抽出する．そこで，抽出されたスライスに対して Declassifiers の配置箇所を検討するアプローチも考えられるが，その場合は複数の不正な情報流について個別に対処することになる．本稿の提案手法では制約集合の MCS に着目することで，複数の不正な情報流に一括して対処できる．

制約集合の MCS に着目する修正手法は，制約充足問題に帰着された型システムだけでなく，VLSI や SoC などの論理回路設計のデバッグにも応用されている [18] [19]．これらの応用はいずれも，制約集合の MCS は何らかの間違いを表しており，MCS を解消することが間違いの修正に対応するという発想に基づいている．

6 おわりに

本稿では，不正な情報流を含むために情報流解析のための型検査に失敗するプログラムに対し，型検査が成功するように declassify 式の記述位置候補を挙げる手法を提案した．型検査を制約充足問題に帰着させ，充足不能な制約集合に対して MCS を求める．制約集合を生成する際に declassify を適用できる式と制約を対応付けておくことで，MCS に含まれる制約から declassify 式の記述対象を決定できる．不正な情報流が存在する場合，提案手法により候補が少なくとも一組は挙げられること，候補に従って declassify 式を記述することで不正な情報流が解消されることを示した．提案手法により，開発者は declassify 式の適切な記述位置を試行錯誤して探すのではなく，候補から選択することで決定できるようになる．

今後の課題として，一般的な機密度束に対応することが挙げられる．本稿では機密度を高いか低いかの二段階としたため，単に declassify 式の記述位置候補を求めるだけでよかった．しかし，一般的な機密度束を前提とする場合，declassify 式を $\texttt{declassify}(e, \eta)$ のように拡張して式 e の機密度を η に指定する仕組みと，declassify 式を挿入する際に適切な η の値を求める手段が必要がある．なお，記述位置候補は本稿の手法のままで求められると予想している．

本稿では，関数を利用できる簡単な手続き型言語を対象としたが，対象言語を拡張してオブジェクトや例外処理などに対応することは今後の課題である．本稿の提

案手法を適用する場合，開発者は提案手法が挙げる候補を比較検討する必要があるが，候補の提示方法を検討することも今後の課題である．複数の候補の具体的な内容と差異や得失を開発者が理解しやすい形で示すことが重要である．

謝辞 本研究の一部は JSPS 科研費 24700036 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, 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. IEEE Computer Society Press, 2005.

[6] Andrei Sabelfeld and Andrew C. Myers. A Model for Delimited Information Release. In *Software Security - Theories and Systems*, Vol. 3233 of *Lecture Notes in Computer Science*, pp. 174–191. Springer Berlin Heidelberg, 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. IEEE Computer Society Press, 2008.

[8] Aslan Askarov and Andrei Sabelfeld. Localized Delimited Release: Combining the What and Where Dimensions of Information Release. In *Proceedings of the 2007 workshop on Programming languages and analysis for security*, PLAS '07, pp. 53–60. ACM, 2007.

[9] 桑原寛明. 型検査に基づく手続き型言語向け情報流解析における型エラースライシング. コンピュータソフトウェア, Vol. 27, No. 4, pp. 221–227, 2010.

[10] James Bailey and Peter J. Stuckey. Discovery of Minimal Unsatisfiable Subsets of Constraints Using Hitting Set Dualization. In *Practical Aspects of Declarative Languages*, Vol. 3350 of *Lecture Notes in Computer Science*, pp. 174–186. Springer Berlin Heidelberg, 2005.

[11] Mark H. Liffiton and Karem A. Sakallah. Algorithms for Computing Minimal Unsatisfiable Subsets of Constraints. *Journal of Automated Reasoning*, Vol. 40, No. 1, pp. 1–33, 2008.

[12] Jakob Rehof and Torben Æ. Mogensen. Tractable constraints in finite semilattices. *Science of Computer Programming*, Vol. 35, No. 2–3, pp. 191 – 221, 1999.

[13] Maria Garcia de la Banda, Peter J. Stuckey, and Jeremy Wazny. Finding All Minimal Unsatisfiable Subsets. In *Proceedings of the 5th ACM SIGPLAN international conference on Principles and practice of declaritive programming*, pp. 32–43. ACM, 2003.

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

[15] 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.

[16] 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 *Lecture Notes in Computer Science*, pp. 327–344. Springer Berlin Heidelberg, 2010.

[17] Jeroen Weijers, Jurriaan Hage, and Stefan Holdermans. Security Type Error Diagnosis for Higher-Order, Polymorphic Languages. In *Proceedings of the ACM SIGPLAN 2013 workshop on Partial evaluation and program manipulation*, PEPM '13, pp. 3–12. ACM, 2013.

[18] Sean Safarpour, Hratch Mangassarian, Andreas Veneris, Mark H. Liffiton, and Karem A. Sakallah. Improved Design Debugging Using Maximum Satisfiability. In *Formal Methods in Computer Aided Design 2007*, pp. 13–19, 2007.

[19] Brian Keng and Andreas Veneris. Automated Debugging of Missing Input Constraints in a Formal Verification Environment. In *Formal Methods in Computer Aided Design 2012*, pp. 101–105, 2012.