OUKA
Osaka University Knowledge Archive

Title	束構造のセキュリティモデルに基づくプログラムの情報フロー解析
Author(s)	國信, 茂太; 高田, 喜朗; 関, 浩之 他
Citation	電子情報通信学会技術研究報告. SS, ソフトウェアサイエンス. 2000, 100(472), p. 25-32
Version Type	VoR
URL	https://hdl.handle.net/11094/26697
rights	Copyright © 2000 IEICE
Note	

Osaka University Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

Osaka University

Institute of Electronics, Information, and Communication Engineers

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.
SS 2000-30 (2000-11)

束構造のセキュリティモデルに基づくプログラムの情報フロー解析

國信 茂太† 高田 喜朗† 関 浩之† 井上 克郎‡

† 奈良先端科学技術大学院大学 情報科学研究科
〒 630-0101 生駒市高山町 8916-5
‡ 大阪大学大学院 基礎工学研究科 情報数理専攻
〒 560-8531 豊中市待兼山町 1-3
E-mail: {shige-ku,y-takata,seki}@is.aist-nara.ac.jp, inoue@ics.es.osaka-u.ac.jp

あらまし プログラムの実行によって, 機密度の高い情報が, 不特定多数のユーザにアクセスされるような記憶域に書き出されないことを保証するためには, プログラムにある機密度の情報が入力されたとき, どのような機密度の情報が出力されるか解析できればよい. 従来の研究において, 機密度 (セキュリティクラス) を束でモデル化し, それに基づいてプログラムの情報フローを静的に解析する方法が提案されているが, 解析対象となるプログラムが単純で, 特に再帰手続きを考慮していなかった. そこで, 本稿では, 一般に再帰を含む手続き型プログラムの情報フローを静的に解析するアルゴリズムを提案する. また, 提案するアルゴリズムの健全性を抽象解釈を用いて証明する.

キーワード 情報フロー解析, セキュリティクラス, 健全性

An Information Flow Analysis of Programs based on a Lattice Model

Shigeta Kuninobu†, Yoshiaki Takata†, Hiroyuki Seki† and Katsuro Inoue‡

†Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, 630-1010, Japan
‡Graduate School of Engineering Science, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan
E-mail: {shige-ku,y-takata,seki}@is.aist-nara.ac.jp, inoue@ics.es.osaka-u.ac.jp

Abstract In order to guarantee that a given program never leaks secrecy information to insecure storage, we should conduct a program analysis which provides the security class of each output of the program when the security class of each input is given. Several program analyses based on a lattice model of security classes were proposed; however, the programs they can analyze are relatively simple and specifically do not contain a recursive call. In this paper, we propose an algorithm which can analyze programs containing recursive calls. Soundness of the algorithm is proved using abstract interpretation.

key words information flow analysis, security class, soundness

NII-Electronic Library Service

Institute of Electronics, Information, and Communication Engineers

1 まえがき

不特定多数の人が利用するシステムにおいては, 情報の漏洩を防ぐことは重要なことである. 情報の漏洩を防ぐ手段として以下のようなアクセス制御法がよく用いられる (Mandatory Access Control と呼ばれる [P94]): 「各データに, そのデータの機密度を表すセキュリティクラス (SC と略記, 例えば, topsecret, confidential, unclassified) を割り当てる. データ d の SC を $SC(d)$ と表す. 同様に, ユーザ (またはプロセス) には, どの機密度のデータまでアクセスできるかを表すクリアランスを割り当てる. ユーザ u のクリアランスを $clear(u)$ と表す. $clear(u) \geq SC(d)$ のときかつそのときのみ, ユーザ u はデータ d を読むことができる. 」

しかし, クリアランスが $SC(d)$ 以上のユーザプログラムがデータ d を読み込み, それを, 故意にまたは過失によって, クリアランスが $SC(d)$ より小さいユーザにもアクセスできる記憶域に書き出してしまうと, 望ましくない情報の漏洩が生じる.

このような情報漏洩を防ぐためには, プログラム P と, P への入力となるデータの SC が与えられたとき, P がどの記憶域にどのような SC のデータを出力するのかを解析できることが望ましい.

Denning らは [D76][DD77] において, SC を束でモデル化し, それに基づいてプログラムの情報フローを静的に解析する方法を初めて提案した. [BBM94][O95][VS97] などでは, [DD77] の手法を理論的に再検討し, 解析法の一般化を試みている. [O95] では抽象解釈を利用してモデル化を行なっている. 一方, [VS97] などでは, 型推論の手法を応用して, 解析法を形式的に記述し, 健全性の証明を行なっている. [O95] では, 通常のプログラムの意味論に情報フローという概念を直接追加した instrumented semantics (作為的意味論) を定義し, その意味論の下で解析法の健全性を示している. [VS97] は, [O95] における健全性の定義が必ずしも自然でないことを指摘し, 解析法の健全性を次のように定義している: 「解析アルゴリズムが『解析対象のプログラムにおける手続き f の SC は τ である』と答えたと仮定する. このとき, SC が τ 以下であるような f の実引数値 (入力) がすべて等しいならば, それ以外の入力が変化しても f の実行結果は不変である.」

しかし, これらの研究では, 解析対象のプログラムが単純であり, 特に, 再帰手続きを考慮していなかった. また, SC の集合も, $\{low, high\}$ など単純なものに限定されていた.

本稿では, 一般に再帰を含む手続き型プログラムの情報フローを解析する方法を提案する. SC の集合は任意の有限分配束によって与えることができる. 本方法では, 解析対象プログラム中のすべての実行文について, その文の実行前後の各変数の SC 間で成り立つべき再帰的な関係式を定義する. この関係式に基づき, プログラムの各手続きの実行結果の SC を解析する. 解析アルゴリズムは, 上述の再帰的な関係式を生成した後, それらの関係式を同時に満たす最小解を不動点計算により求める.

本稿の構成は以下の通りである. まず, 2 節で解析対象となるプログラムを定義する. プログラムは, 代入文, if 文, while 文, 再帰的関数呼出しの機能をもつ通常の手続き型プログラムである. 3 節では, 提案するセキュリティ解析アルゴリズムの概要を説明し, 解析例も示す. 4 節では, 抽象解釈を用いて解析アルゴリズムを形式的に記述する. そして, 本アルゴリズムの [VS97] の意味での健全性を証明する.

2 諸定義

2.1 プログラムの構文

解析対象とするプログラムの構文を以下のように定義する. このプログラミング言語は C 言語に類似した手続き型言語で, 大域変数がない等, 単純なものになっている. プログラムの形式的意味は 4.1節で定義する.

プログラム $Prog$ は関数定義の集合である. 関数定義は以下の形式で与えられる.

$$f(x_1, \ldots, x_n) \text{ local } y_1, \ldots, y_m \ \{P\}$$

ただし, f は関数名, $x_1, \ldots, x_n$ は仮引数, $y_1, \ldots, y_m$ は局所変数, P は関数本体である.

関数本体 P の構文は以下の BNF 式で与えられる. ただし, c は定数, x は局所変数または仮引数, f はプログラム中で定義された関数, θ は加減算などの組み込み演算を表す.

$$\begin{aligned} P &::= C \mid \text{return } M \mid C; P \\ C &::= x := M \mid \text{if } M \text{ then } P \text{ else } P \text{ fi} \\ &\quad \mid \text{while } M \text{ do } P \text{ od} \\ &\quad \mid \text{read}(infile, x) \mid \text{write}(outfile, M) \\ M &::= c \mid x \mid f(M, \ldots, M) \mid \theta(M, \ldots, M) \end{aligned}$$

read, write 文はそれぞれファイルからの読み込み・ファイルへの書き出しを表す命令文である. $infile$, $outfile$ は読み込み・書き出し対象を表すファイル指定子である. プログラム中で使用できるファイル指定子の集合は別途与えられるとする. また簡単のため, $infile$ として使用できる指定子の集合と $outfile$ として使用できる指定子の集合は互いに素と仮定する. すなわち, 入力元でありかつ出力先であるようなファイルは存在しない. 以降, 読み込み対象であるファイルを入力ファイル, 書き出し対象であるファイルを出力ファイルと呼ぶ.

プログラム $Prog$ の実行は, $Prog$ の要素のうち関数名が $main$ である関数を評価することで行われる. この関数を main 関数と呼ぶ. $Prog$ の入力は main 関数に与える実引数および入力ファイルから読み込まれる値, 出力は main 関数の返り値および出力ファイルに書き出される値である.

2.2 値とセキュリティクラス

プログラム実行時にデータとして扱われる値 (定数, 変数が格納する値, 実引数, 関数の返り値など) の型は区別せず, すべて val 型に属すとする.

解析時には, プログラムの入力となる値にそれぞれ, その機密度を表すセキュリティクラス (SC) が与えられる. SC の集合 SCs 上には半順序 $\sqsubseteq$ が定義され, SCs は $\sqsubseteq$ に関して分配束をなすと仮定する. SCs の最小元を $\bot$, 最小上界演算を $\sqcup$ と表す. 直観的には, $s \sqsubseteq t$ であるとは, クリアランスが t であるユーザは SC が s である値にアクセスできることを表す. 以下は最も単純な SCs の例である.

$$SCs = \{low, high\}, \quad low \sqsubseteq high.$$

NII-Electronic Library Service

Institute of Electronics, Information, and Communication Engineers

解析の目的は，入力となる各値のSCがわかっていると仮定した上で，出力となる各値のSCがどのようになるか調べることである．厳密には，4.2節で述べる健全性を満たすような解析結果を求めることが目的である．

ここで，解析の際，入出力ファイルに関しては，1ファイル内の要素を個別に扱うことはせず，1ファイルにつき1つのSCで，全要素のSCを代表して表す．すなわち，入力ファイルについては，1ファイル内の要素はすべて等しいSCを持つと仮定する．出力ファイルについては，各ファイルに書き出される値のSCの最小上界のみ解析結果として求める．

3 解析アルゴリズムの概要

3.1 アルゴリズムの概略

この解析アルゴリズムは，解析対象のプログラム $Prog$ が与えられたとき，SCに関する写像 $\mathcal{A}^*[\![Prog]\!]$ を出力する(4.2節参照)．$Prog$ のmain関数の仮引数を $x_1,\ldots,x_i$，入力ファイルを $infile_1,\ldots,infile_j$ とし，main関数の返り値を y，出力ファイルを $outfile_1,\ldots,outfile_k$ とする．このとき，$\mathcal{A}^*[\![Prog]\!]$ は任意の $i+j$ 個のSCの組を，$1+k$ 個のSCに写像する．これは $Prog$ の入力である実引数および入力ファイルのSCを与えたとき，$Prog$ の出力である返り値および出力ファイルのSCの上界を返すものである．

写像 $\mathcal{A}^*[\![Prog]\!]$ を得るために，プログラムの各関数ごとにSCに関する連立方程式(SC方程式)を作成し，それらの方程式を同時に満たす最小解を繰り返し計算により求める．この節では，SC方程式の作成法と繰り返し計算により解を求める方法を説明する．

3.2 SC方程式の生成

解析アルゴリズムはまず，プログラム中の各命令文に基づいて，SCに関する連立方程式(SC方程式)を生成する．

準備として，変数への値の代入が行われる命令(代入文およびread文)および関数外への値の出力が起こる命令(return文およびwrite文)を同一視し，すべて代入文のように扱う．すなわち，return文は ret という特別な変数への代入文，write文はファイル指定子を変数名とする特別な変数への代入文と考える．また，read文はファイル指定子を変数名とする変数を右辺とする代入文と考える．

SC方程式に現れる変数をSC変数と呼ぶ．各SC変数は，プログラムに現れる変数(局所変数，仮引数，ret，ファイル指定子に対応する変数)に対応している．プログラム中の変数 x に対応するSC変数を $\underline{x}_{(0)},\ldots,\underline{x}_{(n)}$ のように書く．各 $\underline{x}_{(i)}$ はそれぞれプログラム中のある命令文に対応している．各 $\underline{x}_{(i)}$ は，その命令文の実行直後において，x に格納されている値のSCを表すために用いられる．

解析アルゴリズムは，1個の代入文に対し1個の方程式を生成する．方程式の左辺は被代入変数に対応するSC変数である．方程式の右辺は，代入される値が持つSCの上界を表すような式である．代入される値のSCは，代入文の右辺から直接決まるSC(explicit flowに対するSC)と，代入文が属するif文やwhile文の条件部から決まるSC(implicit flowに対するSC)の最小上界となる．explicit flowとは，代入文の右辺から左辺への情報の流れを指す．代入文 $x := M$ におけるexplicit flowに対するSCを $\langle M\rangle$ と表し，以下のように定義する．ただし，以下の $x_{(i)\cdots(j)}$ は，後述のステップ2で変数 x にラベルが付けられたものである．

$$
\begin{aligned}
\langle c\rangle &:= \bot\\
\langle infile\rangle &:= \underline{infile}\\
\langle x_{(i)\cdots(j)}\rangle &:= \underline{x}_{(i)}\sqcup\cdots\sqcup\underline{x}_{(j)}\\
\langle f(M_1,\ldots,M_k)\rangle &:= \underline{f}(\langle M_1\rangle,\ldots,\langle M_k\rangle)\\
\langle\theta(M_1,\ldots,M_k)\rangle &:= \langle M_1\rangle\sqcup\cdots\sqcup\langle M_k\rangle
\end{aligned}
$$

implicit flow[D76]とは，代入文がif文のthen, else以下またはwhile文のdo以下にあるときに，if文またはwhile文の条件部から代入文の左辺への情報の流れを指す．代入文 $x := M$ に対する方程式は，explicit flowとimplicit flowの両方を考慮し，以下のようになる．

$$\underline{x} = \langle M\rangle \sqcup \bigsqcup_{s\in imp}\langle s\rangle$$

ただし，imp はこの代入文におけるimplicit flowを表す式の集合，すなわち代入文が属するif文・while文の条件部の集合である．

SC方程式の生成は以下の4ステップから構成される．

ステップ1 プログラム内に現れる変数(ただし，ファイル指定子に対応する変数を除く)を関数名でラベル付けし，他の関数の変数と区別する．次に，代入文の左辺に現れる各変数に順番に番号(1,2...)を割り当てる．ただし，異なる変数には同じ番号を割り当ててもよい．また，仮引数宣言部の各仮引数にも番号0番を割り当てる(下図)．

```
f(x)                          f(x^f_(0))
{                             {
  read(infile, y);              y^f_(1) := infile;
  while x > 0 do                while x^f > 0 do
    y := y + 1;        ⇒          y^f_(2) := y^f + 1;
    x := x - 1                    x^f_(1) := x^f - 1
  od;                           od;
  return y                      ret^f_(1) := y^f
}                             }
```

ステップ2 各代入文の右辺およびif文・while文の条件部について以下を行う．対象である式を M とする．M に現れる各変数 y について，y がどの代入文によって計算された値を取り得るかを調べ，ステップ1でその代入文に付けられた番号を M 中の y にラベル付けする．y が実引数の値を取り得る場合は，仮引数宣言部における番号，すなわち0をラベル付けする．一般に，y が取り得る値を計算する代入文は複数あるので，y には複数の番号がラベル付けされる．(下図)

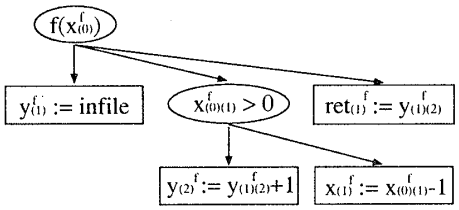
```
f(x^f_(0))                    f(x^f_(0))
{                             {
  y^f_(1) := infile;            y^f_(1) := infile;
  while x^f > 0 do              while x^f_(0)(1) > 0 do
    y^f_(2) := y^f + 1;  ⇒        y^f_(2) := y^f_(1)(2) + 1;
    x^f_(1) := x^f - 1            x^f_(1) := x^f_(0)(1) - 1
  od;                           od;
  ret^f_(1) := y^f              ret^f_(1) := y^f_(1)(2)
}                             }
```


NII-Electronic Library Service

Institute of Electronics, Information, and Communication Engineers

M 中の y がどの代入文によって計算された値を取り得るかを調べるためには，プログラムにおける制御依存グラフ [ASU86] を利用する．$x^f_{(0)(1)} > 0$ となる．

ステップ 3 プログラム内の各代入文に対して，方程式を生成する．すなわち，前述のように，代入文 $x_{(i)} := M$ に対して方程式 $\underline{x}_{(i)} = \langle M \rangle \sqcup \bigsqcup_{s \in imp} \langle s \rangle$ を出力する．ただし，代入文の左辺が出力ファイル $outfile$ の場合は $\underline{outfile} = \langle M \rangle \sqcup \bigsqcup_{s \in imp} \langle s \rangle \sqcup \underline{outfile}$ を出力する．ここで，implicit flow を表す式の集合 imp は，プログラムの構文木を用いて調べる．imp は，構文木上でその代入文の祖先となる if 文および while 文の条件部すべてからなる集合とする．



例えば，代入文 $y^f_{(2)} := y^f_{(1)(2)} + 1$ に対する imp は，上の構文木より，$imp = \{x^f_{(0)(1)} > 0\}$ となる．

ステップ 4 以上の 3 ステップによって各関数ごとに SC 方程式が生成される．その各 SC 方程式に，関数の返り値の SC を与える方程式を加える．例えば，n 引数関数 f に return 文が i 個含まれている場合は，次の方程式を加える．

$$\underline{f}(\underline{x}^f_{(0)}, \ldots, \underline{x}^f_{(n)}) = \underline{ret}^f_{(1)} \sqcup \cdots \sqcup \underline{ret}^f_{(i)}$$

以上，4 ステップによって各関数 f に対して生成された SC 方程式を $Eqs(f)$ と表す．

3.3 SC 方程式の最小不動点の計算

ここでは，プログラム $Prog$ の main 関数の実引数の SC を $s_1, \ldots, s_i$，入力ファイルの SC を $s'_1, \ldots, s'_j$ としたとき，main 関数の返り値および出力ファイルの SC の上界を計算する方法を示す．$\mathcal{A}^*[\![Prog]\!]$ を完全に求めるためには，$s_1, \ldots, s_i$, $s'_1, \ldots, s'_j$ の任意の組み合わせについて以下の手続きを行う必要がある．

準備として，前節で求めた $Eqs(f)$ に対して，$Eqs'(f; t_1, \ldots, t_n; t'_1, \ldots, t'_j)$ を定義する．ただし，f は n 引数関数とし，$t_1, \ldots, t_n, t'_1, \ldots, t'_j$ はそれぞれ任意の SC とする．$Eqs'(f; t_1, \ldots, t_n; t'_1, \ldots, t'_j)$ は，$Eqs(f)$ 中の各方程式に対して，(1) 仮引数に対する SC 変数 $\underline{x_1}_{(0)}, \ldots, \underline{x_n}_{(0)}$ をそれぞれ $t_1, \ldots, t_n$ で置き換え，(2) 入力ファイルに対する SC 変数 $\underline{infile_1}, \ldots, \underline{infile_j}$ をそれぞれ $t'_1, \ldots, t'_j$ で置き換え，(3) 関数名 f がラベル付けされている SC 変数のラベルにさらに $t_1, \ldots, t_n$ を加えて得られる方程式の集合である．例えば，次のようなプログラムを考える．

```
main(x)
{
    read(infile, y);
    z = x + y;
    return z
}
```

このプログラムに関する SC 方程式 $Eqs(main)$ および $Eqs'(main; s_1; s'_1)$ は次のようになる．

$Eqs(main)$

$$\underline{y}^{main}_{(1)} = \underline{infile}$$
$$\underline{z}^{main}_{(1)} = \underline{x}^{main}_{(0)} \sqcup \underline{y}^{main}_{(1)}$$
$$\underline{ret}^{main}_{(1)} = \underline{z}^{main}_{(1)}$$
$$\underline{main}(\underline{x}^{main}_{(0)}) = \underline{ret}^{main}_{(1)}$$

$Eqs'(main; s_1; s'_1)$

$$\underline{y}^{main,s_1}_{(1)} = s'_1$$
$$\underline{z}^{main,s_1}_{(1)} = s_1 \sqcup \underline{y}^{main,s_1}_{(1)}$$
$$\underline{ret}^{main,s_1}_{(1)} = \underline{z}^{main,s_1}_{(1)}$$
$$\underline{main}(s_1) = \underline{ret}^{main,s_1}_{(1)}$$

SC 方程式の解の計算は以下の手順で行われる．

ステップ 1 $Eqs'(main; s_1, \ldots, s_i; s'_1, \ldots, s'_j)$ の方程式の左辺に現れる全変数について記憶領域を割り当て，初期値を $\bot$ とする．このとき，関数の返り値の SC を表す式 $\underline{main}(s_1, \ldots, s_i)$ も変数のように扱い，記憶領域を割り当てる．

ステップ 2 現在の記憶領域の値に基づいて各方程式の右辺を評価し，その値で記憶領域を更新する．右辺に関数呼び出し $\underline{f}(t_1, \ldots, t_n)$ が含まれる場合，$\underline{f}(t_1, \ldots, t_n)$ に対する記憶領域がまだ作られていなければ，$Eqs'(f; t_1, \ldots, t_n; s'_1, \ldots, s'_j)$ に対してステップ 1 を行って記憶領域を新しく割り当てる．

ステップ 3 記憶領域中の全ての値が変化しなくなるまでステップ 2 を繰り返す．すなわち，$m-1$ 回目にステップ 2 を実行した直後の記憶領域と m 回目の直後の記憶領域が完全に等しいとき手続きは終了する．このときの記憶領域中の値が解となる．繰り返し計算中，各変数の SC が下がらないことと，SC の集合およびプログラムに含まれる関数の個数が有限であることから，この手続きは必ず停止する．

3.4 解析例

この節では，次のような 2 関数からなるプログラムを例に，どのように解析が行なわれるのかを示す．ここで，SC の集合は $\{low, high\}$ で，$low \sqsubseteq high$ とする．

```
main(x)                    f(x)
{                          {
  read(infile, y);           if x > 0 then
  while x > 0 do                 return x * f(x - 1)
      y := y + 1;            else
      x := x - 1                 return 0
  od;                        fi
  write(outfile, f(y));    }
  return x
}
```

3.2 節に説明した手順により，SC 方程式 $Eqs(main)$ および $Eqs(f)$ は次のようになる．

$Eqs(main)$

$$\underline{y}^{main}_{(1)} = \underline{infile}$$
$$\underline{y}^{main}_{(2)} = \underline{y}^{main}_{(1)} \sqcup \underline{y}^{main}_{(2)} \sqcup \bot \sqcup \underline{x}^{main}_{(0)} \sqcup \underline{x}^{main}_{(1)} \sqcup \bot$$
$$\underline{x}^{main}_{(1)} = \underline{x}^{main}_{(0)} \sqcup \underline{x}^{main}_{(1)} \sqcup \bot \sqcup \underline{x}^{main}_{(0)} \sqcup \underline{x}^{main}_{(1)} \sqcup \bot$$
$$\underline{outfile} = \underline{f}(\underline{y}^{main}_{(1)} \sqcup \underline{y}^{main}_{(1)}) \sqcup \underline{outfile}$$
$$\underline{ret}^{main}_{(1)} = \underline{x}^{main}_{(0)} \sqcup \underline{x}^{main}_{(1)}$$
$$\underline{main}(\underline{x}^{main}_{(0)}) = \underline{ret}^{main}_{(1)}$$

$Eqs(f)$

$$\underline{ret}^f_{(1)} = \underline{x}^f_{(0)} \sqcup \underline{f}(\underline{x}^f_{(0)} \sqcup \bot) \sqcup \underline{x}^f_{(0)}$$
$$\underline{ret}^f_{(2)} = \bot \sqcup \underline{x}^f_{(0)}$$
$$\underline{f}(\underline{x}^f_{(0)}) = \underline{ret}^f_{(1)} \sqcup \underline{ret}^f_{(2)}$$

NII-Electronic Library Service

ここでは, main 関数の実引数の SC が *high*, *infile* の SC が *low* であるときの解析過程を示す.
まず $Eqs'(main; high; low)$ を考えると, 次のようになる.

$$Eqs'(main; high; low)$$
$$\underline{y}_{(1)}^{main,high} = low$$
$$\underline{y}_{(2)}^{main,high} = \underline{y}_{(1)}^{main,high} \sqcup \underline{y}_{(2)}^{main,high} \sqcup \bot \sqcup high \sqcup \underline{x}_{(1)}^{main,high} \sqcup \bot$$
$$\underline{x}_{(1)}^{main,high} = high \sqcup \underline{x}_{(1)}^{main,high} \sqcup \bot \sqcup high \sqcup \underline{x}_{(1)}^{main,high} \sqcup \bot$$
$$\underline{outfile} = \underline{f}(\underline{y}_{(1)}^{main,high} \sqcup \underline{y}_{(2)}^{main,high}) \sqcup \underline{outfile}$$
$$\underline{ret}_{(1)}^{main,high} = high \sqcup \underline{x}_{(1)}^{main,high}$$
$$\underline{main}(high) = \underline{ret}_{(1)}^{main,high}$$

そして, $Eqs'(main; high; low)$ の方程式の左辺に現れる全変数について記憶領域を割り当て, 初期値を $\bot$ とする. この初期値を用いて方程式の右辺を評価し記憶領域を更新するのだが, 1 回目の計算で $\underline{f}(low)$ の値が必要となる. そこで, $Eqs'(f; low; low)$ を考えると, 次のようになる.

$$Eqs'(f; low; low)$$
$$\underline{ret}_{(1)}^{f,low} = low \sqcup \underline{f}(low \sqcup \bot) \sqcup low$$
$$\underline{ret}_{(2)}^{f,low} = low \sqcup low$$
$$\underline{f}(low) = \underline{ret}_{(1)}^{f,low} \sqcup \underline{ret}_{(2)}^{f,low}$$

そして, 先ほどと同様に $Eqs'(f; low; low)$ の方程式の左辺に現れる全変数について記憶領域を割り当て, 初期値を $\bot$ とする. これにより, $\underline{f}(low)$ の値が利用できるようになるので, 繰り返し計算を続行する. また, 2 回目の計算では, $\underline{f}(high)$ の値が必要となるので, 同様に $Eqs'(f; high; low)$ の方程式の各左辺に記憶領域を割り当て初期値を $\bot$ とする. 以下の表は $Eqs'(main; high; low)$, $Eqs'(f; low; low)$ および $Eqs'(f; high; low)$ の方程式の左辺に現れる全変数が, SC の初期値 $\bot = low$ から始まり, 繰り返し計算により収束するまでの過程を示したものである. ただし, 表中の L, H はそれぞれ, *low*, *high* を表す. また, 第 i 列の SC は第 $i-1$ 列の各値を用いて計算された値を表している.

	0	1	2	3	4	5
$\underline{y}_{(1)}^{main,high}$	L	L	L	L	L	L
$\underline{y}_{(1)}^{main,high}$	L	H	H	H	H	H
$\underline{x}_{(1)}^{main,high}$	L	H	H	H	H	H
$\underline{outfile}$	L	L	L	L	H	H
$\underline{ret}_{(1)}^{main,high}$	L	H	H	H	H	H
$\underline{main}(high)$	L	L	H	H	H	H
$\underline{ret}_{(1)}^{f,low}$	L	L	L	L	L	L
$\underline{ret}_{(2)}^{f,low}$	L	L	L	L	L	L
$\underline{f}(low)$	L	L	L	L	L	L
$\underline{ret}_{(1)}^{f,high}$	-	L	H	H	H	H
$\underline{ret}_{(2)}^{f,high}$	-	L	H	H	H	H
$\underline{f}(high)$	-	L	L	H	H	H

この表より, main 関数の実引数の SC が *high* で入力ファイル (*infile*) の SC が *low* であるとき, main 関数の返り値の SC は *high* となり得ることがわかる ($\underline{main}(high)$ の行を見ればよい). また, このとき, 出力ファイル (*outfile*) の SC も *high* となり得ることがわかる.

4 解析アルゴリズムの健全性

本節では 3節で述べた解析アルゴリズムを形式的に定義し, その健全性を証明する. ただし, 簡単のため, ファイル操作を含まないプログラムのみを考える.

4.1 プログラムの意味

次のような型を仮定する. $\times$ で直積, $+$ で分離和を表す.

val 型 (値を表す型) プログラムに現れる n 引数基本演算子 θ に対し, n 引数関数 $\theta_{\mathcal{I}} : val \times \cdots \times val \to val$ が定義されているとする.

store 型 (記憶域を表す型) 2 つの関数

$$update : store \times var \times val \to store$$
$$lookup : store \times var \to val$$

があり, 以下の公理を満たすとする.

$$lookup(update(\sigma, x, v), y) = \text{if } x = y \text{ then } v \text{ else } lookup(\sigma, y)$$

読み易さのため, $\sigma(x) \equiv lookup(\sigma, x)$, $\sigma[x := v] \equiv update(\sigma, x, v)$ と略記する. また, $\bot_{store}$ で, すべての変数の値が未定義の記憶域を表す.

sc 型 (セキュリティクラス (SC) の型) .

メタ変数として以下を用いる.

$x, x_1, \ldots, y_1, \ldots : var$ $\quad$ $M, M_1, \ldots : expression$
$C : command$ $\quad$ $P, P_1, P_2 : cseq$
$\sigma, \sigma', \sigma'' : store$

プログラムの意味を与える関数 $\cdot \models \cdot \Rightarrow \cdot$ を定義する. この関数は, 記憶域, (式, 文, 文の系列のいずれか) を引数に取り, 記憶域または値を返す.

$$\models : (store \to expression \to val) + (store \to command \to store) + (store \to cseq \to (store + val))$$

- 「$\sigma \models M \Rightarrow v$」は, 記憶域 σ の下で式 M を評価した値が v であることを表す.
- 「$\sigma \models C \Rightarrow \sigma'$」は, 記憶域 σ の下で文 C を実行した直後の記憶域が σ' であることを表す.
- 文の系列に対しても同様.
- 「$\sigma \models P \Rightarrow v$」は, 記憶域 σ の下で文の系列 P を実行した直後, 関数値 v が返されることを表す. これは, P の末尾 が return M の場合のみあてはまる.

以下に関数 $\cdot \models \cdot \Rightarrow \cdot$ を定義する公理と推論規則を与える.

(CONST) $\sigma \models c \Rightarrow c_{\mathcal{I}}$

(VAR) $\sigma \models x \Rightarrow \sigma(x)$

$$\textbf{(PRIM)}\ \frac{\sigma \models M_i \Rightarrow v_i \quad (1 \le i \le n)}{\sigma \models \theta(M_1, \ldots, M_n) \Rightarrow \theta_{\mathcal{A}}(v_1, \ldots, v_n)}$$

$$\textbf{(CALL)}\ \frac{\sigma \models M_i \Rightarrow v_i \quad (1 \le i \le n) \quad \sigma' \models P_f \Rightarrow v}{\sigma \models f(M_1, \ldots, M_n) \Rightarrow v}$$
$$\left(\begin{array}{l} f(x_1, \ldots, x_n) \text{ local } y_1, \ldots, y_m \quad P_f \\ \sigma' = \bot_{store}[x_1 := v_1] \cdots [x_n := v_n] \end{array} \right)$$

NII-Electronic Library Service

Institute of Electronics, Information, and Communication Engineers

$$\textbf{(ASSIGN)}\ \frac{\sigma \models M \Rightarrow v}{\sigma \models x := M \Rightarrow \sigma[x := v]}$$

$$\textbf{(IF1)}\ \frac{\sigma \models M \Rightarrow \text{true} \quad \sigma \models P_1 \Rightarrow \sigma'}{\sigma \models \text{if } M \text{ then } P_1 \text{ else } P_2 \text{ fi} \Rightarrow \sigma'}$$

$$\textbf{(IF2)}\ \frac{\sigma \models M \Rightarrow \text{false} \quad \sigma \models P_2 \Rightarrow \sigma'}{\sigma \models \text{if } M \text{ then } P_1 \text{ else } P_2 \text{ fi} \Rightarrow \sigma'}$$

$$\textbf{(WHILE1)}\ \frac{\sigma \models M \Rightarrow \text{true} \quad \sigma \models P \Rightarrow \sigma' \quad \sigma' \models \text{while } M \text{ do } P \text{ od} \Rightarrow \sigma''}{\sigma \models \text{while } M \text{ do } P \text{ od} \Rightarrow \sigma''}$$

$$\textbf{(WHILE2)}\ \frac{\sigma \models M \Rightarrow \text{false}}{\sigma \models \text{while } M \text{ do } P \text{ od} \Rightarrow \sigma}$$

$$\textbf{(RETURN)}\ \frac{\sigma \models M \Rightarrow v}{\sigma \models \text{return } M \Rightarrow v}$$

$$\textbf{(CONCAT)}\ \frac{\sigma \models C \Rightarrow \sigma' \quad \sigma' \models P \Rightarrow \sigma''}{\sigma \models C; P \Rightarrow \sigma''}$$

$$\frac{\sigma \models C \Rightarrow \sigma' \quad \sigma' \models P \Rightarrow v}{\sigma \models C; P \Rightarrow v}$$

4.2 健全性の定義

解析アルゴリズムとは，
$\mathcal{A}^*[\![\cdot]\!] : program \to (fname \times sc \cdots \times sc) \to sc$ という型をもつ関数である．$\mathcal{A}^*[\![Prog]\!](f, \tau_1, \ldots, \tau_n) = \tau$ であるとき，$\mathcal{A}^*[\![\cdot]\!]$ は，$Prog$ で定義された n 引数関数 f に対して，引数の SC が $\tau_1, \ldots, \tau_n$ であるとき，f の SC を τ と解析したことを表す．

定義 4.1 以下の条件が成り立つとき，解析アルゴリズム $\mathcal{A}^*[\![\cdot]\!]$ は**健全性 (soundness)** を満たすという：

$Prog$ を任意のプログラム，$main$ を $Prog$ の main 関数とする．

$$\mathcal{A}^*[\![Prog]\!](main, \tau_1, \ldots, \tau_n) = \tau$$

であり，また，

$$\bot_{store}[x_1 := v_1] \cdots [x_n := v_n] \models main(x_1, \ldots, x_n) \Rightarrow v$$
$$\bot_{store}[x_1 := v'_1] \cdots [x_n := v'_n] \models main(x_1, \ldots, x_n) \Rightarrow v'$$
$$\forall i (1 \le i \le n) : \tau_i \sqsubseteq \tau.\ v_i = v'_i$$

であると仮定する．このとき，$v = v'$ が成り立つ． □

上の定義で，解析アルゴリズムが健全性を満たすとは，「解析結果が τ であると仮定する．このとき，SC が τ 以下であるような実引数の値がすべて等しいならば，それ以外の実引数が変化しても実行結果の値は不変である」ことを表す．つまり，解析アルゴリズムが「関数 f の SC は τ」と答えたときは，τ 以下でない実引数の情報は，実行結果に流れ出ないと保証されることを表す．

4.3 情報フロー解析関数

4.1節で導入した型に加えて以下の型を用いる．

$\underline{store}$ 型 **(記憶域の SC)**

$$update : \underline{store} \times var \times sc \to \underline{store}$$
$$lookup : \underline{store} \times var \to sc$$

$\underline{store}$ 型でも，$store$ 型と同様の略記法を用いる．$\underline{\sigma}$ を $\underline{store}$ 型の元とするとき，$\underline{\sigma}(x)$ は，変数 x の SC の解析 (途中) 結果を表す．sc で定義された半順序 $\sqsubseteq$ を型 $\underline{store}$ 上に次のように拡張することにより，$\underline{store}$ に束の構造を与えることができる．$\underline{\sigma}$, $\underline{\sigma'}$ を $\underline{store}$ 型の値とする．

$$\underline{\sigma} \sqsubseteq \underline{\sigma'} \Leftrightarrow \forall x \in var.\ \underline{\sigma}(x) \sqsubseteq \underline{\sigma'}(x)$$

$\underline{store}$ の最小元は $\forall x \in var.\ \underline{\sigma}(x) = \bot$ であるような $\underline{\sigma}$ であり，これを $\bot_{\underline{store}}$ と表す．

$\underline{fun}$ 型 **(関数の SC)** $\underline{store}$ 型と同様，次の関数が定義されている．

$$update : \underline{fun} \times fname \times sc \cdots \times sc \times sc \to \underline{fun}$$
$$lookup : \underline{fun} \times fname \times sc \cdots \times sc \to sc$$

また，

$$F[f, \tau_1, \ldots, \tau_n] \equiv lookup(F, f, \tau_1, \ldots, \tau_n)$$
$$F[\langle f, \tau_1, \ldots, \tau_n \rangle := \tau] \equiv update(F, f, \tau_1, \ldots, \tau_n, \tau)$$

と略記する．$F[f, \tau_1, \ldots, \tau_n]$ は引数の SC が $\tau_1, \ldots, \tau_n$ であるときの関数 f の SC の解析 (途中) 結果を表す．sc の束構造を $\underline{store}$ 型に拡張したのと同様に，$\underline{fun}$ 型にも束構造を導入する．最小元を $\bot_{\underline{fun}}$ と表す．

cv-$\underline{fun}$ 型 (covariant $\underline{fun}$ 型) $\underline{fun}$ 型の値 F のうち，条件

$$(\tau_1, \ldots, \tau_n) \sqsubseteq (\tau'_1, \ldots, \tau'_n) \text{ ならば}$$
$$F[f, \tau_1, \ldots, \tau_n] \sqsubseteq F[f, \tau'_1, \ldots, \tau'_n]$$

を満たすものすべてからなる．

メタ変数として，4.1節のものに加えて以下を用いる．

$$\underline{\sigma}, \underline{\sigma'}, \underline{\sigma''} : \underline{store} \qquad F, F_1, F_2 : \underline{fun}$$

情報フローを解析する関数 $\mathcal{A}[\![\cdot]\!]$ を定義する．$\mathcal{A}$ は，式，関数の SC，記憶域の SC を引数に取って SC を返し，文または文の系列，関数の SC，記憶域の SC，implicit flow の SC を引数に取って SC または記憶域の SC を返す．

$$\begin{aligned}\mathcal{A} : &(expression \times \underline{fun} \times \underline{store} \to sc)\\ &+ (command \times \underline{fun} \times \underline{store} \times sc \to \underline{store})\\ &+ (cseq \times \underline{fun} \times \underline{store} \times sc \to (\underline{store} + sc))\end{aligned}$$

- 「$\mathcal{A}[\![M]\!](F, \underline{\sigma}) = \tau$」は，各関数の SC の解析結果 F，記憶域の SC $\underline{\sigma}$ の下で，式 M の SC を τ と解析することを表す．
- 「$\mathcal{A}[\![C]\!](F, \underline{\sigma}, \nu) = \underline{\sigma'}$」は，各関数の SC の解析結果 F，記憶域の SC $\underline{\sigma}$，implicit flow の SC ν の下で，文 C を実行した直後の記憶域の SC を $\underline{\sigma'}$ と解析することを表す．
- 文の系列に対しても同様．
- 「$\mathcal{A}[\![P]\!](F, \underline{\sigma}, \nu) = \tau$」は，各関数の SC の解析結果 F，記憶域の SC $\underline{\sigma}$，implicit flow の SC ν の下で，文の系列 P を実行した直後に返される関数値の SC を τ と解析することを表す．これは，P の末尾が return M の場合のみ定義される．

NII-Electronic Library Service

Institute of Electronics, Information, and Communication Engineers

以下に $\mathcal{A}$ の定義を与える．これは 3 節で述べた解析アルゴリズムの形式的定義である．

(CONST) $\mathcal{A}[\![c]\!](F,\underline{\sigma}) = \bot$

(VAR) $\mathcal{A}[\![x]\!](F,\underline{\sigma}) = \underline{\sigma}(x)$

(PRIM)
$\mathcal{A}[\![\theta(M_1,\ldots,M_n)]\!](F,\underline{\sigma}) = \bigsqcup_{1\le i\le n} \mathcal{A}[\![M_i]\!](F,\underline{\sigma})$

(CALL) $\mathcal{A}[\![f(M_1,\ldots,M_n)]\!](F,\underline{\sigma})$
$= F[f, \mathcal{A}[\![M_1]\!](F,\underline{\sigma}),\ldots,\mathcal{A}[\![M_n]\!](F,\underline{\sigma})]$

(ASSIGN)
$\mathcal{A}[\![x := M]\!](F,\underline{\sigma},\nu) = \underline{\sigma}[x := \mathcal{A}[\![M]\!](F,\underline{\sigma}) \sqcup \nu]$

(IF) $\mathcal{A}[\![\text{if } M \text{ then } P_1 \text{ else } P_2 \text{ fi}]\!](F,\underline{\sigma},\nu)$
$= \mathcal{A}[\![P_1]\!](F,\underline{\sigma},\nu\sqcup\tau) \sqcup \mathcal{A}[\![P_2]\!](F,\underline{\sigma},\nu\sqcup\tau)$
ただし，$\tau = \mathcal{A}[\![M]\!](F,\underline{\sigma})$

(WHILE) $\mathcal{A}[\![\text{while } M \text{ do } P \text{ od}]\!](F,\underline{\sigma},\nu)$
$= \mathcal{A}[\![P]\!](F,\underline{\sigma},\nu \sqcup \mathcal{A}[\![M]\!](F,\underline{\sigma})) \sqcup \underline{\sigma}$

(RETURN) $\mathcal{A}[\![\text{return } M]\!](F,\underline{\sigma},\nu) = \mathcal{A}[\![M]\!](F,\underline{\sigma}) \sqcup \nu$

(CONCAT)
$\mathcal{A}[\![C; P]\!](F,\underline{\sigma},\nu) = \mathcal{A}[\![P]\!](F, \mathcal{A}[\![C]\!](F,\underline{\sigma},\nu),\nu)$

プログラム $Prog$ を入力として $Prog$ で定義された各関数の情報フローを解析する関数 $\mathcal{A}[\![\cdot]\!] : program \to \underline{fun} \to \underline{fun}$ を次のように定義する:

$Prog \equiv \{f(x_1,\ldots,x_n) \text{ local } y_1,\ldots,y_m \quad P_f,\ldots\}$ とするとき，

$$\begin{aligned}&\mathcal{A}[\![Prog]\!](F) = \\ &\quad F[\langle f,\tau_1,\ldots,\tau_n\rangle := \\ &\quad \mathcal{A}[\![P_f]\!](F, \bot_{\underline{store}}[x_1 := \tau_1]\cdots[x_n := \tau_n], \bot) \\ &\quad \mid Prog \text{ で定義された各 } n \text{ 引数関数 } f \text{ と} \\ &\quad \text{任意の SC } \tau_1,\ldots,\tau_n] \end{aligned} \tag{1}$$

束 $(S,\preceq)$ における関数 $f : S \to S$ に対して，f の最小不動点を $fix(f)$ で表す．プログラム $Prog$ に対して，$\mathcal{A}[\![Prog]\!]$ の最小不動点を与える関数

$$\mathcal{A}^*[\![Prog]\!] = fix(\lambda F.\mathcal{A}[\![Prog]\!](F))$$

が解析アルゴリズムである．後述の補題 4.2より，$\mathcal{A}[\![Prog]\!]$ は有限の束 cv-$\underline{fun}$ 上の単調関数となるので，

$$\mathcal{A}^*[\![Prog]\!] = \bigsqcup_{i\ge 0} \mathcal{A}[\![Prog]\!]^i(\bot_{\underline{fun}})$$

が成り立つ [M96]．ここで，$f^0(x) = x$, $f^{i+1}(x) = f(f^i(x))$．すなわち，$\mathcal{A}^*[\![Prog]\!]$ は，すべての内容が最小元であるような関数の SC から開始して，$\mathcal{A}[\![Prog]\!]$ を，関数の SC が変化しなくなるまで繰り返し適用することにより計算できる．

4.4 健全性

補題 4.2 *(a)* F が cv-$\underline{fun}$ 型ならば，$\mathcal{A}[\![Prog]\!](F)$ も cv-$\underline{fun}$ 型である．

(b) (単調性) F_1, F_2 は cv-$\underline{fun}$ 型であると仮定する．
$F_1 \sqsubseteq F_2$ ならば $\mathcal{A}[\![Prog]\!](F_1) \sqsubseteq \mathcal{A}[\![Prog]\!](F_2)$.

(証明の方針) $Prog$ の構造に関する帰納法により示せる．□

補題 4.3 $\underline{\sigma'} = \mathcal{A}[\![P]\!](F,\underline{\sigma},\nu)$ とおく．任意の変数 x に対し，$\underline{\sigma}(x) = \underline{\sigma'}(x)$ または $\nu \sqsubseteq \underline{\sigma'}(x)$.

(証明の方針) P の構造的帰納法により示せる．□

補題 4.4 (*implicit flow* の性質) $\mathcal{A}[\![P]\!](F,\underline{\sigma},\nu) = \underline{\sigma'}$, $\sigma \models P \Rightarrow \sigma'$, $\nu \not\sqsubseteq \underline{\sigma'}(y)$ ならば，$\sigma(y) = \sigma'(y)$.

(証明の方針) 補題 4.3 を用い，$\sigma \models P \Rightarrow \sigma'$ を導出するのに用いた推論規則の適用回数に関する帰納法により示せる．

補題 4.5 (*implicit flow* の性質) $\mathcal{A}[\![P]\!](F,\underline{\sigma},\nu) = \underline{\sigma'}$, $\sigma \models$ *while* M *do* P *od* $\Rightarrow \sigma'$, $\nu \not\sqsubseteq \underline{\sigma'}(y)$ ならば，$\sigma(y) = \sigma'(y)$.

(証明の方針) 補題 4.4を用い，推論規則の適用回数に関する帰納法により示せる．□

定理 4.6 $F = \mathcal{A}^*[\![Prog]\!]$ とおく．

(a) $\mathcal{A}[\![M]\!](F,\underline{\sigma}) = \tau$, $\sigma_1 \models M \Rightarrow v_1$, $\sigma_2 \models M \Rightarrow v_2$, $\forall x : \underline{\sigma}(x) \sqsubseteq \tau.\ \sigma_1(x) = \sigma_2(x)$ ならば，$v_1 = v_2$.

(b) $\mathcal{A}[\![P]\!](F,\underline{\sigma},\nu) = \underline{\sigma'}$, $\sigma_1 \models P \Rightarrow \sigma_1'$, $\sigma_2 \models P \Rightarrow \sigma_2'$, $\underline{\sigma'}(y) = \tau$, $\forall x : \underline{\sigma}(x) \sqsubseteq \tau.\ \sigma_1(x) = \sigma_2(x)$ ならば，$\sigma_1'(y) = \sigma_2'(y)$.

(c) $\mathcal{A}[\![P]\!](F,\underline{\sigma},\nu) = \tau$, $\sigma_1 \models P \Rightarrow v_1$, $\sigma_2 \models P \Rightarrow v_2$, $\forall x : \underline{\sigma}(x) \sqsubseteq \tau.\ \sigma_1(x) = \sigma_2(x)$ ならば，$v_1 = v_2$.

(証明) 推論規則の適用回数に関する帰納法により示す．
((a) の証明) (CALL)

$$\tau = \mathcal{A}[\![f(M_1,\ldots,M_n)]\!](F,\underline{\sigma})$$
$$= F[f, \mathcal{A}[\![M_1]\!](F,\underline{\sigma}),\ldots,\mathcal{A}[\![M_n]\!](F,\underline{\sigma})] \tag{2}$$
$$\frac{\sigma_k \models M_i \Rightarrow u_{ki} \quad (1\le i\le n) \quad \sigma_k' \models P_f \Rightarrow v_k}{\sigma_k \models f(M_1,\ldots,M_n) \Rightarrow v_k} \tag{3}$$
$$\sigma_k' = \bot_{store}[x_1 := u_{k1}]\cdots[x_n := u_{kn}] \quad (k = 1,2) \tag{4}$$
$$\forall x : \underline{\sigma}(x) \sqsubseteq \tau.\ \sigma_1(x) = \sigma_2(x) \tag{5}$$

と仮定する．$F = \mathcal{A}[\![Prog]\!](F)$ であるから，(1)と (2)より，

$$\begin{aligned}&\tau = \mathcal{A}[\![f(M_1,\ldots,M_n)]\!](F,\underline{\sigma}) \\ &= \mathcal{A}[\![P_f]\!](F, \bot_{\underline{store}}[x_1 := \mathcal{A}[\![M_1]\!](F,\underline{\sigma})] \\ &\quad \cdots[x_n := \mathcal{A}[\![M_n]\!](F,\underline{\sigma})], \bot) \end{aligned} \tag{6}$$

$\tau_i = \mathcal{A}[\![M_i]\!](F,\underline{\sigma}) \sqsubseteq \tau$ $(1\le i\le n)$ と仮定すると，(5)より，$\forall x : \underline{\sigma}(x) \sqsubseteq \tau_i.\ \sigma_1(x) = \sigma_2(x)$. 帰納法の仮定 (a) より，$u_{1i} = u_{2i}$. すなわち，

$$\forall i(1\le i\le n) : \mathcal{A}[\![M_i]\!](F,\underline{\sigma}) \sqsubseteq \tau.\ u_{1i} = u_{2i}.$$

よって，(4)より，

$$\begin{aligned}&\forall x : \bot_{\underline{store}}[x_1 := \mathcal{A}[\![M_1]\!](F,\underline{\sigma})]\cdots \\ &[x_n := \mathcal{A}[\![M_n]\!](F,\underline{\sigma})](x) \sqsubseteq \tau.\ \sigma_1'(x) = \sigma_2'(x). \end{aligned} \tag{7}$$

NII-Electronic Library Service

Institute of Electronics, Information, and Communication Engineers

(6), (3), (7)と帰納法の仮定 (c) より, $v_1 = v_2$.
他の場合は容易.
((b) の証明) (ASSIGN)

$$\begin{aligned}\underline{\sigma}' &= \mathcal{A}[\![x := M]\!](F, \underline{\sigma}, \nu) \\ &= \underline{\sigma}[x := \mathcal{A}[\![M]\!](F, \underline{\sigma}) \sqcup \nu]\end{aligned} \tag{8}$$

$$\frac{\sigma_k \models M \Rightarrow v_k}{\sigma_k \models x := M \Rightarrow \sigma_k[x := v_k]} \quad (k = 1, 2) \tag{9}$$

$$\tau = \underline{\sigma}'(y) \tag{10}$$

$$\forall z : \underline{\sigma}(z) \sqsubseteq \tau.\ \sigma_1(z) = \sigma_2(z) \tag{11}$$

と仮定. $x \neq y$ のとき, (8)より $\underline{\sigma}'(y) = \underline{\sigma}(y)$. これと (10), (11)より, $\sigma_1(y) = \sigma_2(y)$. さらにこれと (9)より, $\sigma_1'(y) = \sigma_2'(y)$. $x = y$ のとき, (8), (10)より, $\underline{\sigma}'(y) = \mathcal{A}[\![M]\!](F, \underline{\sigma}) \sqcup \nu = \tau$. よって, $\mathcal{A}[\![M]\!](F, \underline{\sigma}) \sqsubseteq \tau$. これと (11)より,

$$\forall z : \underline{\sigma}(z) \sqsubseteq \mathcal{A}[\![M]\!](F, \underline{\sigma}).\ \sigma_1(z) = \sigma_2(z). \tag{12}$$

(9), (12)と帰納法の仮定 (a) より, $v_1 = v_2$, すなわち, $\sigma_1'(y) = \sigma_2'(y)$.
(WHILE)

$$\underline{\sigma}'' = \underline{\sigma}' \sqcup \underline{\sigma} \tag{13}$$

$$\underline{\sigma}' = \mathcal{A}[\![P]\!](F, \underline{\sigma}, \nu \sqcup \tau) \tag{14}$$

$$\tau = \mathcal{A}[\![M]\!](F, \underline{\sigma}) \tag{15}$$

$$\tau' = \underline{\sigma}''(y) \tag{16}$$

$$\forall x : \underline{\sigma}(x) \sqsubseteq \tau'.\ \sigma_1(x) = \sigma_2(x) \tag{17}$$

と仮定する.
(i) $\dfrac{\sigma_k \models M \Rightarrow \text{false}}{\sigma_k \models \text{while } M \text{ do } P \text{ od} \Rightarrow \sigma_k}$ のとき. (13), (16)より, $\underline{\sigma}(y) \sqsubseteq \underline{\sigma}''(y) = \tau'$. (17)より, $\sigma_1(y) = \sigma_2(y)$.
(ii)

$$\frac{\sigma_1 \models M \Rightarrow \text{true} \quad \sigma_1 \models P \Rightarrow \sigma_1' \quad \sigma_1' \models \text{while } M \text{ do } P \text{ od} \Rightarrow \sigma_1''}{\sigma_1 \models \text{while } M \text{ do } P \text{ od} \Rightarrow \sigma_1''} \tag{18}$$

$$\frac{\sigma_2 \models M \Rightarrow \text{false}}{\sigma_2 \models \text{while } M \text{ do } P \text{ od} \Rightarrow \sigma_2} \tag{19}$$

のとき. $\tau \sqsubseteq \tau'$ と仮定すると, (15), (17), (18), (19)と帰納法の仮定 (a) より, true=false となり矛盾. 従って, $\tau \not\sqsubseteq \tau'$. これと (13), (16)より,

$$\nu \sqcup \tau \not\sqsubseteq \underline{\sigma}'(y). \tag{20}$$

(14), $\sigma_1 \models P \Rightarrow \sigma_1'$, (20)と補題 4.4より, $\sigma_1(y) = \sigma_1'(y)$. (14), $\sigma_1' \models$ while M do P od $\Rightarrow \sigma_1''$, (20)と補題 4.5より, $\sigma_1'(y) = \sigma_1''(y)$. 一方 (i) の場合と同様にして, $\sigma_1(y) = \sigma_2(y)$. 以上より, $\sigma_1''(y) = \sigma_2(y)$.
(iii)

$$\frac{\sigma_k \models M \Rightarrow \text{true} \quad \sigma_k \models P \Rightarrow \sigma_k' \quad \sigma_k' \models \text{while } M \text{ do } P \text{ od} \Rightarrow \sigma_k''}{\sigma_k \models \text{while } M \text{ do } P \text{ od} \Rightarrow \sigma_k''} \quad (k = 1, 2) \tag{21}$$

$\underline{\sigma}'(x) \sqsubseteq \tau'$ と仮定すると, (14), (17), (21)と帰納法の仮定 (b) より, $\sigma_1'(x) = \sigma_2'(x)$. すなわち,

$$\forall x : \underline{\sigma}'(x) \sqsubseteq \tau'.\ \sigma_1'(x) = \sigma_2'(x). \tag{22}$$

(14), (16), (21), (22)と再び帰納法の仮定 (b) より, $\sigma_1''(y) = \sigma_2''(y)$.

(CONCAT), (IF) の場合は容易.
(c) の証明は容易なので省略. □

系 4.7 解析アルゴリズム $\mathcal{A}^*[\![\cdot]\!]$ は健全性を満たす.

(証明) 定理 4.6(c) より明らか. □

5 あとがき

本稿では, 一般に再帰を含む手続き型プログラムの情報フローを静的に解析するアルゴリズムを提案した. このアルゴリズムでは, SC の集合として任意の有限分配束を与えることができる. また, 抽象解釈を用いて解析アルゴリズムを形式的に記述し, その健全性を証明した. [Y00] では, このアルゴリズムの実装が行なわれている.

今後は, 大域変数を含むような, より複雑なプログラムに対しても解析が行えるようにアルゴリズムを拡張することを考える. また, 解析アルゴリズムの効率化についても考えていく予定である.

参考文献

[ASU86] A. V. Aho, R. Sethi and J. D. Ullman: "Compilers : Principles, Techniques, and Tools," Addison-Wesley, 1986.

[BBM94] J. Banâtre, C. Bryce and D. Le Métayer: "Compile-Time Detection of Information Flow in Sequential Programs," Proc. 3rd ESORICS, LNCS 875, pp.55–73, 1994.

[D76] D. E. Denning: "A Lattice Model of Secure Information Flow," Communications of the ACM, Vol.19, No.5, pp.236–243, 1976.

[DD77] D. E. Denning and P. J. Denning: "Certifications of Programs for Secure Information Flow," Communications of the ACM, Vol.20, No.7, pp.504–513, 1977.

[M96] J. Mitchell: "Foundations of Programming Languages," The MIT Press, 1996.

[O95] P. Ørbæk: "Can You Trust Your Data?" Proc. TAPSOFT '95, LNCS 915, pp.575–589, 1995.

[P94] G. Purnul: "Database Security," Advances in Computers (M. Yovits Ed.), Vol.38, pp.1–72, 1994.

[VS97] D. Volpano and G. Smith: "A Type-Based Approach to Program Security," Proc. TAPSOFT '97, LNCS 1214, pp.607–621, 1997.

[Y00] 横森, 大畑, 高田, 関, 井上: "セキュリティ解析アルゴリズムの実現とオブジェクト指向言語への適用に関する一考察," 電子情報通信学会ソフトウェアサイエンス研究会, 2000 年 11 月.

NII-Electronic Library Service