特集●プログラミングおよびプログラミング言語

低レベル言語のための情報流解析の型システム

小林 直樹　白根 慶太

システムソフトウェアなどが扱うデータには，パスワード情報のように特定の者にしか漏らしてはならない機密情報が含まれていることがあるが，プログラムが誤って記述されていればその情報が第 3 者に漏れてしまう．そのような誤りを静的に検出するため，情報流解析と呼ばれるプログラム解析が手続き型言語，関数型言語などを対象として研究されてきた．本研究の目的は，アセンブリ言語や仮想機械言語などの低レベル言語のための情報流解析の手法を確立し，ソースプログラムが存在しないコードの情報流解析を可能とすることである．本論文ではその第一歩として，Java の仮想機械言語の部分言語に対する情報流解析手法を，Stata と Abadi のバイトコード検証のための型システムの拡張として提案する．

1 序　　論

プログラムは様々な内部データを処理するが，その中にはパスワードのように第 3 者に漏らしてはならない機密データが含まれている場合がある．通常，そのような場合にはプログラムの実行権限を制限することによって機密データが漏れる（あるいは改竄される）ことを防ぐが，プログラムそのものに誤りがある場合にはこの対策だけでは不十分である．例えば，パスワードを更新するプログラムが間違っており，変更結果を誰にでも読めるファイルに書き込んでしまえば，そのプログラムの実行権限を管理者のみに制限してもパスワード情報の漏洩を防ぐことはできない．

上記のような問題を解決するため，プログラムの実行前にそのプログラムが機密情報を外部に漏らさないかを検査する，情報流解析 (*information flow analysis*) と呼ばれる静的プログラム解析が提案，研究されてきた[3, 4, 12, 17, 21]．これらは，パスワードなど機密性の高いデータの情報がどのように伝播するかを解析する．例えば

$$non\_secret := \mathbf{if}\ passwd < s\ \mathbf{then}\ 1\ \mathbf{else}\ 2$$

という代入文（$passwd$ はパスワード文字列，s は文字列，2 項演算子 $<$ は文字列の辞書式順序での比較とする）では，代入される値を読むことにより $passwd$ の情報の一部が分かるので，$passwd$ の情報が non_secret という変数に伝播している（流れている）とみなす[†1]．non_secret が誰でも読むことができる変数であれば情報漏えいの可能性があるとして情報流解析によって拒絶（あるいは警告）される．実際，上の文が外部から呼び出すことのできる手続きの一部であり，s がその手続きの引数である場合には，その手続きを複数回呼び出すことによって簡単に $passwd$ の値を知ることができる．（なお，上の基準では login プログラムなども login が成功したかによってパスワードの情報の一部が漏れるので，情報漏えいの可能性があると判断される．login などが安全であるかを判断するには確率的な議論[20] が必要になるが，ここでは扱わない．従って，本論文で提

†1 データフロー解析[14] とは異なることに注意されたい．上の文では $passswd$ そのものが代入されているわけではないので，$passswd$ から non_secret への（データフロー解析の定義上は）データの流れはない．

Naoki Kobayashi, 東京工業大学 情報理工学研究科
Keita Shirane, 東京工業大学 情報理工学研究科

案する解析では他の多くの解析同様，login プログラムも拒絶または警告の対象となる．）

上記のような情報流解析の手法は，これまで主に手続き型言語や関数型言語など比較的高レベルな言語に対して提案されてきた．例えば Denning[3] や Volpano ら[21] は if 文と while 文，代入文などからなる手続き型言語に対する情報流解析の手法を提案した．Heintze と Riecke[4] は λ 計算に対する情報流解析のための型システムを提案した．また，Smith と Volpano[17] や本田ら[5, 6] によって，並行言語に対する情報流解析手法も研究されている．

本研究では，Java の仮想機械言語などのより低レベルな言語のための，型システムに基づく情報流解析の手法を提案し，その正しさを証明する．本手法により，ライブラリなどソースプログラムが存在しないコードに対する情報流解析が可能となる．本質を明確にするため，対象言語として Java の仮想機械言語を単純化した言語を用いる．以下では我々の情報流解析のアイデアについて簡単に説明する．

本論文で提案する情報流解析手法の基本的な枠組みは，Volpano ら[21] の手法や Heintze と Riecke[4] の手法と同じく，型システムに基づくものである．型システムに基づく情報流解析のアイデアは，文字列・整数などデータの種類を表す通常の型に，データの機密度（セキュリティレベル）を表す情報を付加することである．例えば整数の型 **Int** は，「機密度の高い整数」の型 $\mathbf{Int_H}$，「機密度の低い整数」の型 $\mathbf{Int_L}$，にわけることができる．このような型の概念の拡張にあわせて型つけ規則を変更できる†2．例えば if 式の通常の型つけ規則は

$$\frac{\Gamma \vdash b : \mathbf{Bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathbf{if}\ b\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 : \tau}$$

である（**Bool** はブール値の型，τ は任意の型を表すメタ変数，Γ は各自由変数にどのような型の値が格納されているかを表す型環境とする）が，情報流解析のための型システムでは

$$\frac{\Gamma \vdash b : \mathbf{Bool}_{\kappa_1} \qquad \Gamma \vdash e_1 : \tau_{\kappa_2} \qquad \Gamma \vdash e_2 : \tau_{\kappa_3} \qquad \kappa_1, \kappa_2, \kappa_3 \sqsubseteq \kappa_4}{\Gamma \vdash \mathbf{if}\ b\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 : \tau_{\kappa_4}}$$

となる．ここで，τ_κ は機密度が κ のデータの型を表す．$\kappa_1, \kappa_2, \kappa_3 \sqsubseteq \kappa_4$ は if 式の値の機密度が e_1 および e_2 部の値の機密度のみならず，分岐に用いるブール式 b の値の機密度と同等かそれ以上でなければならないことを表す．以上のようにして，「型つけできるプログラムは機密情報を漏らすことはない」という性質を満たす型システムを構築することができる．これにより，情報流解析の問題は型判定あるいは型推論の問題へと帰着される．

本論文で提案する低レベル言語のための情報流解析がこれまでに提案されてきた高レベル言語のための情報流解析の場合と比較して困難な点としては，低レベル言語では（１）同じ変数（レジスタ）にプログラムの場所によって異なる型や機密度を持つ値が格納されること（２）if 文などの分岐命令が構造化されていないこと，があげられる．（１）の問題については，Stata と Abadi のバイトコード検証のための型システム[18] にならい，各変数にプログラムの各場所（命令アドレス）ごとに異なる型を割り当てることによって解決する．（２）の問題を理解するために再び if 式

$$\mathbf{if}\ b\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2$$

を考える．goto 文のような大域脱出がない場合には，分岐に用いる式 b の値の情報が直接伝播するのは if 式全体および e_1, e_2 で代入される変数である．ところが，goto 文がある場合，b の値の情報が伝播する範囲が自明ではない．例えば次のようなプログラムを考える．

$$\begin{array}{ll} 1: & x := 0; \\ 2: & x := x + 1; \\ 3: & \mathbf{if}\ b\ \mathbf{goto}\ 2; \\ 4: & \cdots \end{array}$$

ここで，左の列は行番号を表し，3 行目の命令は b が真なら 2 行目へ飛び，そうでなければ 4 行目へ進む命令とする．この場合，b の情報は 2 行目で代入される変数 x にも伝播すると考えなければならない（x の値によって 2 行目が何回実行されたかがわかり，それに

†2 以下では型システムに関するある程度の知識を仮定する．馴染みのない読者は本段落の残りの部分を読み飛ばしてもかまわない．

よって b の値の情報がわかるため）．本論文では，このような分岐情報が伝播する範囲を「分岐した後の最初の合流点[†3]に至るまでに通過する範囲」とすればよいことを示し，その正しさの証明の概略を与えるとともに，その範囲を求めるためのアルゴリズムも与える．

本研究の新規性は，対象言語が低レベル言語であることに加え，通常よりも強い意味での正しさを保証する点にある．情報流解析の正しさの基準としては，通常，「機密度の低い出力データは機密度の高いデータの情報に依存しない（したがって機密度の低い出力データから機密度の高いデータの情報を推測できない）」ことを表す非干渉性（non-interference）[4, 21] が用いられる．しかしながら，この性質が成り立っても，実行時間の情報から機密度の高いデータの情報漏れが起こる可能性がある．例えば $long\_computation$ が長い計算時間を要した後 1 を返す式であるとし，次のような式を考える．

$$\textbf{if}\ b\ \textbf{then}\ long\_computation\ \textbf{else}\ 1$$

この式の実行結果が 1 であることがわかっても b の値を知ることはできないが，実行時間が分かれば b の値を推測することができる．本論文の情報流解析の型システムでは，このような実行時間からの情報漏れも防ぐことができる．実行時間からの情報漏れを防ぐ方法として，高レベル言語については Agat[2] によってプログラム変換を用いる手法が提案されているが，本論文の手法はプログラム変換の必要はなく，より簡潔である．

本論文の以下の構成は次の通りである．2節で対象言語を導入する．3節において，型システムを定義し，4節で型推論アルゴリズムを説明する．5節では本研究の手法の妥当性について論じる．6節で関連研究に触れ，7節で結論を述べる.

2　対象言語 BL

本論文の情報流解析の対象言語として用いる BL を定義する．BL は Java の仮想機械言語からオブジェクトやそのメソッドの呼び出し，サブルーチンなどを取り除いて単純化した言語である．単純化した言語を用いるのは，提案する解析手法の本質をわかりやすくするためであり，本論文の手法をオブジェクトやメソッド呼び出しなどを含めた言語に拡張することは難しくない．

BL のプログラムは Java のバイトコード同様，仮想機械を操作する命令列からなる．仮想機械はオペランドスタックと局所変数領域から構成されており，各命令はスタック上のデータに関する演算やスタックと局所変数領域の間でのデータの移動を行う．操作するデータとしては整数のみを考える．

本節では 2.1 節において BL の構文を定義し，2.2 節において BL の操作的意味を示す．

2.1　構　　文

以下では，自然数の集合の可算部分集合として，命令アドレス（以下単にアドレスと呼ぶ）の集合 **Adr**，局所変数名の集合 **LVar**，スタックアドレスの集合 **SVar** の存在を仮定する．

定義 2.1　BL の命令 I の構文は以下によって定義される．

$$\begin{array}{rcl} I & ::= & \mathrm{inc} \mid \mathrm{pop} \mid \mathrm{push0} \mid \mathrm{load}\ x \\ & & \mid \mathrm{store}\ x \mid \mathrm{if}\ l \mid \mathrm{goto}\ l \mid \mathrm{return} \end{array}$$

ただしメタ変数 l, x はそれぞれ命令アドレス，局所変数名を表すメタ変数である．命令の集合を $\mathcal{I}$ と書く．

各命令の意味は次の通り．inc はスタックのトップの値（整数）を 1 増やす．pop は，スタックのトップから値を取り出す．push0 は，スタックのトップに整数 0 をのせる．load x は，局所変数（以下変数と呼ぶ）x の値をスタックのトップにのせる．store x は，スタックのトップから値を取り出し，その値を変数 x に格納する．if l は，スタックのトップの値を取り出し，その値が整数 0 なら次のアドレスへ，それ以外の整数なら，アドレス l にとぶ．goto l は，アドレス l にとぶ．return は，スタックのトップの値を取り出し，その値を返り値としてメソッドを終了する．厳密な操作的意味は Stata と Abadi の型システム [18] の対象言語と同様に定義できる．

定義 2.2　セキュリティレベルとは **L** または **H** である．セキュリティレベルの間の 2 項関係 $\sqsubseteq$ は $\mathbf{L} \sqsubseteq \mathbf{H}$ を満たす全順序関係である．

†3　最適化コンパイラなどで用いられている *immediate forward dominators*[11]（あるいは immediate post-dominators）に近い概念だが，若干定義が異なる．詳しくは後述を参照．

セキュリティレベルを表すメタ変数として κ を用いる．$\mathbf{H}$ はデータの機密度が高い（特定の者にしか知られてはならない）ことを，$\mathbf{L}$ はデータの機密度が低いことを表す．本論文で 2 つのセキュリティレベルのみを考えるのは単純化のためであり，一般には任意の束を考えることができる[3]．

セキュリティレベルに関する 2 項演算子 $\sqcup$ を

$$\kappa_1 \sqcup \kappa_2 = \begin{cases} \kappa_1 & \text{if } \kappa_2 \sqsubseteq \kappa_1 \\ \kappa_2 & \text{その他の場合} \end{cases}$$

によって定義する。

定義 2.3 メソッド本体とは，$\mathbf{Adr}$ の有限部分集合から $\mathcal{I}$ への関数であり，メタ変数 B を用いて表す．メソッドとは，セキュリティレベルの組とメソッド本体の組 $((\kappa_1, ..., \kappa_n, \kappa_r), B)$ であり，メタ変数 M を用いて表す．

メソッド中のセキュリティレベルの組 $(\kappa_1, ..., \kappa_n, \kappa_r)$ は，引数の個数が n 個で，（局所変数 i に格納される）i 番目の引数のセキュリティレベルが κ_i であり，返り値の値のセキュリティレベルが κ_r であることを表す．$\kappa_r = \mathbf{H}$ である場合には，$\mathbf{L}$ 以下のセキュリティレベルのデータしか読むことの許されない者にはメソッドの実行が終了したかどうかの情報も隠すと仮定する[†4]．

例えば，メソッド $((\mathbf{H}, \mathbf{L}, \mathbf{L}), B)$ は 1 番目の引数として機密度の高いデータを，2 番目のデータとして機密度の低いデータを受け取り，機密度の低いデータを返すメソッドを表す．（従って，このメソッドは 1 番目の引数の値の情報が含まれる値を返り値として返すことはない．）本論文ではプログラムとしてただ 1 つのメソッドのみからなるものを考える．したがってメソッドとプログラムとを同一視する．

例 2.1 図 1のメソッド本体 B をもつメソッド $((\mathbf{H}, \mathbf{L}), B)$ を考える．アドレス 2 においては，引数 1 の値が 0 であるかどうかを判定し，0 ならばアドレス 3 ~ 4 において変数 2 に整数 0 を格納し，0 でなければアドレス 6 ~ 8 において変数 2 に 1 を格納する．アドレス 11 で変数 2 に格納されている値を返り値としているので，返り値を読むことによって，引数の情報（0 であるかどうか）が分かる．ところが引数のセキュリティレベルが $\mathbf{H}$，返り値のセキュリティレベルが $\mathbf{L}$ なので，上のメソッドは（秘密の情報を漏らしているという点で）誤りである．一方，B を上のメソッド本体として，メソッド $((\mathbf{H}, \mathbf{H}), B)$ および $((\mathbf{L}, \mathbf{L}), B)$ は正しいプログラムである．3 節では，このような情報の漏洩の観点からプログラムが正しいかどうかを判定するための型システムを提案する．

l	$B(l)$	説明
1	load 1	引数 1 の値をロード
2	if 6	ロードした値が 0 でない時 アドレス 6 へ分岐
3	push0	0 をスタックにプッシュ
4	store 2	0 を変数 2 に格納
5	goto 10	アドレス 10 へとぶ
6	push0	0 をスタックにプッシュ
7	inc	プッシュした 0 を 1 増やす
8	store 2	1 を変数 2 に格納
9	goto 10	アドレス 10 へとぶ
10	load 2	変数 2 の値をロード
11	return	ロードした値を返し， メソッドの実行終了

図 1 メソッド本体の例

記法 2.1 以下では，集合 S_1 から集合 S_2 への部分関数の集合を $A \rightharpoonup B$ と書き，関数 f の定義域を $dom(f)$ と書く．また，関数 f の定義域から x を取り除いて得られる関数を $f \backslash x$ と書き，定義域が空集合である関数を $\emptyset$ と書く．$f\{x \mapsto v\}$ は，以下によって定義される関数を表す．

$$dom(f\{x \mapsto v\}) = dom(f) \cup \{x\}$$

$$(f\{x \mapsto v\})(y) = \begin{cases} v & y = x \text{ のとき} \\ f(y) & \text{それ以外のとき} \end{cases}$$

2.2 操作的意味

メソッドの実行状態を命令アドレス $pc \in \mathbf{Adr}$, 局所変数名の有限集合から自然数の集合 $\mathbf{Nat}$ への写像 $f \in \mathbf{LVar} \rightharpoonup \mathbf{Nat}$, スタックアドレスの部分集合 $\{i \mid 0 \leq i < n\}$ から $\mathbf{Nat}$ への写像 $s \in \mathbf{SVar} \rightharpoonup \mathbf{Nat}$ の組 (pc, f, s) として表す．pc は現在実行しようとして

†4 メソッドの実行が終了したかどうかの情報を公開するのであれば，セキュリティレベル $\mathbf{L}$ の値しか分岐に用いることができないように 3節の型システムを変更する必要がある．

いる命令アドレスを，$f(x)$ は局所変数 x に格納されている値を，$s(i)$ はスタックの i 番目に格納されている値を表す．以下では，pc をプログラムカウンタ，f を局所変数の状態（あるいは環境），s をスタックの状態（あるいは単にスタック），と呼ぶ．

実行の 1 ステップを表す関係

$$(pc, f, s) \longrightarrow_B (pc', f', s')$$

を図 2の規則によって定義する．これは，プログラムカウンタが pc，環境が f，スタックが s のとき，アドレス pc の命令 $B(pc)$ を実行すると，プログラムカウンタが pc'，環境が f'，スタックが s' となることを表す．図中の $v \cdot s$ は，$(v \cdot s)(0) = v, (v \cdot s)(n+1) = s(n)$ によって定義されるスタックを表す．規則には表されていないが，return 命令が実行された時には，スタックのトップの値が結果として返され，残りのスタックや局所変数中の値は捨てられて外部に漏れないものとする．

$\longrightarrow_B$ の反射的推移的閉包を $\longrightarrow_B^*$ と書く．また，n ステップの実行を表す関係 $\longrightarrow_B^n$ を以下の規則によって帰納的に定義する．

$$\frac{}{(pc, f, s) \longrightarrow_B^0 (pc, f, s)}$$

$$\frac{(pc, f, s) \longrightarrow_B (pc'', f'', s'') \quad (pc'', f'', s'') \longrightarrow_B^n (pc', f', s')}{(pc, f, s) \longrightarrow_B^{n+1} (pc', f', s')}$$

さらに，n 以下のある自然数 i について $(pc, f, s) \longrightarrow_B^i (pc', f', s')$ が成り立つとき，

$$(pc, f, s) \longrightarrow_B^{\leq n} (pc', f', s')$$

と書く．

3 型システム

本節では，プログラムが秘密の情報を漏らしていないかを判定するための型システムを与える．我々の型システムで正しく型つけできるプログラムは情報漏洩をしないことが保証される．

1節で述べたように，我々の型システムの基本的なアイデアは，通常の型にデータの機密度を表す情報（セキュリティレベル）を加えて拡張することである．Stata と Abadi の型システム[18] にならい，プログラムの各命令アドレスごとに，その命令を実行する時にスタックおよび局所変数に格納されているデータの型を不整合なく割り当てることができるかどうかを判定する．ただし，我々の型システムにおいては，（他の情報流解析のための型システム[15, 21, 22] と同様に）各命令アドレスに「そのアドレスの命令を実行したかどうかの情報が漏れてもよいか」を表すセキュリティレベルも割り当てる．例えば例 2.1では，アドレス 6 を実行したことが分かると，アドレス 2 で検査した引数 1 の値が 0 でないことが分かってしまうので，アドレス 6 のセキュリティレベルとして $\mathbf{H}$ を割り当てる．これにより，アドレス 6 においてスタックに格納した値のセキュリティレベルも $\mathbf{H}$ でなければならない（スタックに格納された値の情報がセキュリティレベル $\mathbf{L}$ の返り値に伝播するとアドレス 6 が実行されることが分かってしまうので）ことが分かり，結局返り値のセキュリティレベルも $\mathbf{H}$ でなければならないことが分かる．このようにして，「アドレスのセキュリティレベル」の概念を導入することにより，分岐による情報漏れの解析を上手く定式化することができる．

以下では，まず 3.1節において値およびスタック，ローカル変数の型を導入し，3.2節においてアドレスのセキュリティレベルと分岐に用いる値のセキュリティレベルとの関係を導く上で重要となる概念を導入する．3.3節において型つけ規則を与え，3.4節において型システムの健全性を示す．与えられたメソッドが情報を漏洩していないかを実際に判定するには型推論を行う必要があるが，これについては 4節に譲る．

3.1 型と型環境

定義 3.1 型は集合 $\{\mathbf{Int_L}, \mathbf{Int_H}, \top\}$ の要素である．

$\mathbf{Int_L}$ は，機密度の低い整数の型，$\mathbf{Int_H}$ は，機密度の高い整数の型，$\top$ は全く使われない値（言い換えれば，誰も見ることのできない極限の機密度をもつ値）の型を表す．

以下で定義する関係 $\tau_1 \leq \tau_2$ は，τ_1 型の値を τ_2 型の値として用いてもよいことを表す．

定義 3.2 部分型関係 $\leq$ は以下の条件を満たす全順序関係である．

$$\mathbf{Int_L} \leq \mathbf{Int_H} \leq \top$$

$\tau_1 \leq \tau_2$ が成り立つとき，τ_1 を τ_2 の部分型と呼ぶ．

$$\frac{B(pc) = \mathrm{inc}}{(pc, f, v \cdot s) \longrightarrow_B (pc+1, f, (v+1) \cdot s)} \quad \text{(R-Inc)} \qquad \frac{B(pc) = \mathrm{store}\ x}{(pc, f, v \cdot s) \longrightarrow_B (pc+1, f\{x \mapsto v\}, s)} \quad \text{(R-Str)}$$

$$\frac{B(pc) = \mathrm{pop}}{(pc, f, v \cdot s) \longrightarrow_B (pc+1, f, s)} \quad \text{(R-Pop)} \qquad \frac{B(pc) = \mathrm{if}\ l'}{(pc, f, 0 \cdot s) \longrightarrow_B (pc+1, f, s)} \quad \text{(R-IfT)}$$

$$\frac{B(pc) = \mathrm{push0}}{(pc, f, s) \longrightarrow_B (pc+1, f, 0 \cdot s)} \quad \text{(R-Push)} \qquad \frac{B(pc) = \mathrm{if}\ l' \qquad v \neq 0}{(pc, f, v \cdot s) \longrightarrow_B (l', f, s)} \quad \text{(R-IfF)}$$

$$\frac{B(pc) = \mathrm{load}\ x}{(pc, f, s) \longrightarrow_B (pc+1, f, f(x) \cdot s)} \quad \text{(R-Load)} \qquad \frac{B(pc) = \mathrm{goto}\ l'}{(pc, f, s) \longrightarrow_B (l', f, s)} \quad \text{(R-Goto)}$$

図 2 操作的意味

局所変数の型は変数の有限集合から型への写像であり，メタ変数 F を用いて表す．$F(x)$ は，局所変数 x に格納されている値の型を表す．スタックの型は $\{i \mid 0 \leq i < n\}$(n は任意) の形のスタックアドレスの有限集合から型への写像であり，メタ変数 S を用いて表す．$S(k)$ は，スタックの k 番目（0 番目がトップ）に格納されている値の型を表す．$\tau \cdot S$ は，$(\tau \cdot S)(0) = \tau, (\tau \cdot S)(n+1) = S(n)$ によって定義されるスタックの型を表す．アドレスの有限集合から局所変数の型への写像を局所変数の型環境と呼び，メタ変数 $\mathcal{F}$ を用いて表す．アドレスの有限集合からスタックの型への写像をスタックの型環境と呼び，メタ変数 $\mathcal{S}$ を用いて表す．アドレスの有限集合からセキュリティレベルの集合への写像をアドレスの型環境と呼び，メタ変数 $\mathcal{A}$ を用いて表す．$\mathcal{F}(l)$，$\mathcal{S}(l)$ はそれぞれアドレス l を実行する際の局所変数の型，スタックの型を表す．$\mathcal{A}(l)$ はアドレス l の命令を実行したかどうかの情報の機密度を表す．例えば，$\mathcal{A}(l) = \mathbf{H}$ ならばアドレス l の命令を実行したかどうかの情報の機密性が高いことを表す．

F_1，F_2 が以下の関係を満たすとき，F_1 は F_2 の部分型であると言い，$F_1 \leq F_2$ とかく．

$$dom(F_1) = dom(F_2) \wedge \forall x \in dom(F_1).(F_1(x) \leq F_2(x)).$$

スタックの型に関する部分型関係 $S_1 \leq S_2$ も同様に定義する．

3.2 制御情報の伝播範囲 $depend(l)$

各命令アドレスのセキュリティレベル，すなわち各命令を実行したかどうかの情報の機密度は，それ以前に経由した分岐結果の機密度に依存する．例えば例 2.1のプログラムでは，アドレス 3 から 9 のどれを実行したことが分かるとアドレス 2 の分岐結果が分かってしまうので，アドレス 3 から 9 のセキュリティレベルはアドレス 2 の分岐結果の機密度，すなわち分岐に用いた値のセキュリティレベル $\mathbf{H}$ 以上に保たなければならない．一方，アドレス 1 や 10 はアドレス 2 での分岐結果にかかわらず実行されるので，それらのセキュリティレベルはアドレス 2 の分岐結果の機密度の影響を受けない．上のようにアドレス l での分岐結果の機密度の影響をうけるアドレスの集合を l の制御情報の伝播範囲と呼び，$depend(l)$ と書く．上の例では $depend(2) = \{3, 4, 5, 6, 7, 8, 9\}$ である．以下では，$depend(l)$ の非形式的な定義を述べ，厳密な定義は付録 Aに譲る．

命令アドレス l からの完全実行経路とは，l 以降に順に実行されうるアドレスの列（最初の l は含まない）のうち，無限列あるいは末尾のアドレスの命令が return である有限列のことである．命令アドレス l からの合流点とは，命令アドレス l からのすべての完全実行経路に含まれるアドレスのことである．命令アドレス l からの最初の合流点とは，命令アドレス l からの合流点のうち，いずれかの完全実行経路に最初に現れるアドレスのことである．ただしそのような点が複数存在する場合には[†5]，その内最も小さいアドレスとする．命令アドレス l の制御情報の伝播範囲 $depend(l)$ とは，合流点が

†5 実は，完全経路中に有限列が含まれていれば（すなわちいずれかの経路で return 命令に到達する場合には），上の条件を満たす点は高々一つしか存在しない．したがって，複数存在する場合は停止しないプログラムなので実は最初の合流点をどのように選んでもかまわない．詳細は付録 Aを参照

存在する場合にはいずれかの完全経路の最初の合流点以前に登場する点の集合であり，合流点が存在しない場合にはいずれかの完全経路に含まれる点の集合である．

例えば例 2.1のメソッドでは，2 からの完全実行経路は 3.4.5.10.11 および 6.7.8.9.10.11 であり，合流点の集合は $\{10, 11\}$，最初の合流点は 10 である．したがって，$depend(2) = \{3, 4, 5, 6, 7, 8, 9\}$ である．それ以外のアドレス l については $depend(l) = \emptyset$ となる．

直観的には，アドレス l の命令が分岐命令である時，分岐後にすべての実行経路が合流するアドレスがあれば，それ以降は分岐の影響をうけないので，アドレス l から最初の合流点に至る経路上にあるアドレスを l の制御情報の伝播範囲とみなしている，と理解すればよい．上の定義で，完全実行経路から無限列を除けば最適化コンパイラなどで用いられている “immediate foward dominator” [11]（immediate post-dominator とも呼ばれる）に一致する．我々の解析ではプログラムが停止するかどうかで情報が漏れてしまう可能性があるので，完全実行経路に無限列を含むことは必要不可欠である．

3.3 型判定規則

本節では「メソッド M の各アドレス l での局所変数，スタックの型が $\mathcal{F}(l)$，$\mathcal{S}(l)$ であり，かつアドレスのセキュリティレベルが $\mathcal{A}(l)$ であれば M を正しく実行できる」ことを表す関係 $\mathcal{F}, \mathcal{S}, \mathcal{A} \vdash M$ を定義する．前準備として，まず各行での型の整合性を表す関係 $\mathcal{F}, \mathcal{S}, \mathcal{A}, l \vdash M$ を定義する．この関係は，「各アドレス k での局所変数の型，スタックの型，アドレスのセキュリティレベルがそれぞれ $\mathcal{F}(k), \mathcal{S}(k), \mathcal{A}(k)$ ならば，アドレス l の命令を正しく実行できる」ことを表す．

定義 3.3 $\mathcal{F}, \mathcal{S}, \mathcal{A}, l \vdash M$ は，図 3の規則を満たす最小の関係である．

図 3の中で，$\mathcal{F}(l)$，$\mathcal{S}(l)$ を $\mathcal{F}_l$，$\mathcal{S}_l$ と略す．以下で，それぞれの規則について説明する．

規則 (T-Inc)： 1 行目は，型付けしようとしているアドレス l の命令が inc であることを表している．inc 命令はスタックのトップの整数に 1 を足すので，命令実行前のスタックのトップの型 $\mathcal{S}(l)(0)$ は整数型でなければならない．命令実行後のスタックのトップの型も整数型または $\top$ でなければならず，そのセキュリティレベルは元の値のセキュリティレベル以上でなければならない．これらは 2 行目の条件によって保証される．inc 命令はそれ以外のスタックの値および局所変数の値にはアクセスしないので，3 行目の制約に示されるように，$\mathcal{F}(l)$ は $\mathcal{F}(l+1)$ の部分型でなければならない．さらに，スタックのトップの値のセキュリティレベルはアドレス l のセキュリティレベル以上でなければならない．これらの制約を表しているのが 3 行目である．4 行目は次のアドレスの命令が存在することを表す．

規則 (T-Pop)： pop 命令はスタックのトップの値を取り出し，次のアドレスへ進む．局所変数は命令実行前と後で同じなので，$\mathcal{F}(l)$ は $\mathcal{F}(l+1)$ の部分型となる．スタックはトップの値が取られ，残りの部分は命令実行前と後で同じなので，$\mathcal{S}(l)$ は $\tau \cdot \mathcal{S}(l+1)$ の部分型となる．これらの条件を表しているのが 2 行目である．

規則 (T-Push)： push0 命令はスタックのトップに整数 0 をのせ，次のアドレスへ進む．局所変数は命令実行前と後で同じなので，$\mathcal{F}(l)$ は $\mathcal{F}(l+1)$ の部分型となる．スタックのトップにのせる値のセキュリティレベルはアドレス l のセキュリティレベル $\mathcal{A}(l)$ 以上でなければならず，残りの部分は命令実行前と後で同じなので，$\mathbf{Int}_{\mathcal{A}(l)} \cdot \mathcal{S}(l)$ は $\mathcal{S}(l+1)$ の部分型となる．これらの条件を表しているのが 2 行目である．

規則 (T-Load)： load x 命令は，局所変数は変更しないので，2 行目に示されるように $\mathcal{F}(l)$ は $\mathcal{F}(l+1)$ の部分型でなければならない．スタックには局所変数 x の値が積まれるので，$\mathcal{F}(l)(x) \cdot \mathcal{S}(l) \leq \mathcal{S}(l+1)$ が成り立たなければならない．また，積まれた値のセキュリティレベルはアドレス $l+1$ のセキュリティレベル以上でなければならない．この条件を表しているのが 4 行目の条件である．

規則 (T-Str)： 2 行目は，局所変数 x が，存在することを表している．store x 命令は，スタックのトップの値を変数 x に格納するので，$\mathcal{S}(l)$ は $\mathcal{F}(l+1)(x) \cdot \mathcal{S}(l+1)$ の部分型でなければならない．さらに，格納された値のセキュリティレベルはアドレス l のそれ以上でなければならない．これらの制約を表しているのが 2 行目である．残りの部分は命令実行前と後で同じなので，3 行目が示すように $\mathcal{F}(l)$ は $\mathcal{F}(l+1)$ の部分型でな

$$\frac{B(l) = \mathrm{inc} \quad \mathcal{S}_l(0) = \mathbf{Int}_\kappa \quad \mathcal{S}_l \leq \mathcal{S}_{l+1} \quad \mathcal{F}_l \leq \mathcal{F}_{l+1} \quad \mathbf{Int}_{\mathcal{A}(l)} \leq \mathcal{S}_{l+1}(0) \quad l+1 \in dom(B)}{\mathcal{F}, \mathcal{S}, \mathcal{A}, l \vdash ((\kappa_1, \ldots, \kappa_n, \kappa_r), B)} \quad (\text{T-Inc})$$

$$\frac{B(l) = \mathrm{pop} \quad \mathcal{F}_l \leq \mathcal{F}_{l+1} \quad \mathcal{S}_l \leq \tau \cdot \mathcal{S}_{l+1} \quad l+1 \in dom(B)}{\mathcal{F}, \mathcal{S}, \mathcal{A}, l \vdash ((\kappa_1, \ldots, \kappa_n, \kappa_r), B)} \quad (\text{T-Pop})$$

$$\frac{B(l) = \mathrm{push0} \quad \mathcal{F}_l \leq \mathcal{F}_{l+1} \quad \mathbf{Int}_{\mathcal{A}(l)} \cdot \mathcal{S}_l \leq \mathcal{S}_{l+1} \quad l+1 \in dom(B)}{\mathcal{F}, \mathcal{S}, \mathcal{A}, l \vdash ((\kappa_1, \ldots, \kappa_n, \kappa_r), B)} \quad (\text{T-Push})$$

$$\frac{B(l) = \mathrm{load}\ x \quad \mathcal{F}_l \leq \mathcal{F}_{l+1} \quad \mathcal{F}_l(x) \cdot \mathcal{S}_l \leq \mathcal{S}_{l+1} \quad \mathbf{Int}_{\mathcal{A}(l)} \leq \mathcal{S}_{l+1}(0) \quad l+1 \in dom(B)}{\mathcal{F}, \mathcal{S}, \mathcal{A}, l \vdash ((\kappa_1, \ldots, \kappa_n, \kappa_r), B)} \quad (\text{T-Load})$$

$$\frac{B(l) = \mathrm{store}\ x \quad S_l \leq \mathcal{F}_{l+1}(x) \cdot S_{l+1} \quad \mathbf{Int}_{\mathcal{A}(l)} \leq \mathcal{F}_{l+1}(x) \quad \mathcal{F}_l \backslash x \leq \mathcal{F}_{l+1} \backslash x \quad l+1 \in dom(B)}{\mathcal{F}, \mathcal{S}, \mathcal{A}, l \vdash ((\kappa_1, \ldots, \kappa_n, \kappa_r), B)} \quad (\text{T-Str})$$

$$\frac{B(l) = \mathrm{if}\ l' \quad \mathcal{F}_l \leq \mathcal{F}_{l+1} \quad \mathcal{F}_l \leq \mathcal{F}_{l'} \quad \mathcal{S}_l \leq \mathbf{Int}_\kappa \cdot \mathcal{S}_{l+1} \quad \mathcal{S}_l \leq \mathbf{Int}_\kappa \cdot \mathcal{S}_{l'} \quad \forall k \in depend(l).\kappa \sqsubseteq \mathcal{A}(k) \quad l+1, l' \in dom(B)}{\mathcal{F}, \mathcal{S}, \mathcal{A}, l \vdash ((\kappa_1, \ldots, \kappa_n, \kappa_r), B)} \quad (\text{T-If})$$

$$\frac{B(l) = \mathrm{goto}\ l' \quad \mathcal{F}_l \leq \mathcal{F}_{l'} \quad \mathcal{S}_l \leq \mathcal{S}_{l'} \quad l' \in dom(B)}{\mathcal{F}, \mathcal{S}, \mathcal{A}, l \vdash ((\kappa_1, \ldots, \kappa_n, \kappa_r), B)} \quad (\text{T-Goto})$$

$$\frac{B(l) = \mathrm{return} \quad \mathcal{S}_l(0) \leq \mathbf{Int}_{\kappa_r} \quad \mathcal{A}(l) \sqsubseteq \kappa_r}{\mathcal{F}, \mathcal{S}, \mathcal{A}, l \vdash ((\kappa_1, \ldots, \kappa_n, \kappa_r), B)} \quad (\text{T-Ret})$$

図 3　型判定規則

ければならない．

規則 (T-If)：if　l' 命令は，スタックトップの値によって次のアドレスもしくはアドレス l' へ進むので，2 ~ 3 行目が示すように $\mathcal{F}(l)$ は $\mathcal{F}(l+1)$ および $\mathcal{F}(l')$ の部分型でなければらない．$\mathcal{S}(l)$ は $\mathbf{Int}_\kappa \cdot \mathcal{S}(l+1)$ および $\mathbf{Int}_\kappa \cdot \mathcal{S}(l')$ の部分型でなければらない．さらに 4 行目が示すように，分岐結果に依存する範囲 $depend(l)$ のアドレスのセキュリティレベルは分岐に用いられる値のセキュリティレベル κ 以上でなければらない．

規則 (T-Goto)：goto　l' 命令はアドレス l' に進む．局所変数およびスタックは命令実行前と後で同じなので，$\mathcal{F}(l)$ は $\mathcal{F}(l')$ の部分型，$\mathcal{S}(l)$ は $\mathcal{S}(l')$ の部分型でなければならない．

規則 (T-Ret)：return 命令はスタックのトップの値の整数をメソッドの返り値とするので，スタックトップの値の型は $\mathbf{Int}_{\kappa_r}$ の部分型でなければならない．さらに，返り値のセキュリティレベル κ_r はアドレスのセキュリティレベル以上でなければらない．これらの条件を表しているのが 2 ~ 3 行目である．

上の関係を用いて関係 $\mathcal{F}, \mathcal{S}, \mathcal{A} \vdash M$ を以下のように定義できる．

定義 3.4メソッドの型判定関係 $\mathcal{F}, \mathcal{S}, \mathcal{A} \vdash M$ は次の規則を満たす最小の関係である．

$$\frac{\{1, \ldots, n\} \subseteq dom(\mathcal{F}(1)) \quad \forall j \in \{1, \ldots, n\}.\mathcal{F}(1)(j) = \mathbf{Int}_{\kappa_j} \quad \forall j \in dom(\mathcal{F}(1)) \backslash \{1, \ldots, n\}.\mathcal{F}(1)(j) = \top \quad \mathcal{S}(1) = \emptyset \quad \forall l \in dom(B).\mathcal{F}, \mathcal{S}, \mathcal{A}, l \vdash ((\kappa_1, \ldots, \kappa_n, \kappa_r), B)}{\mathcal{F}, \mathcal{S}, \mathcal{A} \vdash ((\kappa_1, \ldots, \kappa_n, \kappa_r), B)}$$

$\mathcal{F}, \mathcal{S}, \mathcal{A} \vdash M$ を満たす $\mathcal{F}, \mathcal{S}, \mathcal{A}$ が存在するとき，メ

ソッド M は型つけ可能であると言う．

1 ~ 3 行目は，メソッドの先頭であるアドレス 1 では，局所変数の型のセキュリティレベルが引数のセキュリティレベル $\kappa_1, \ldots, \kappa_n$ と合致していることを表す．4 行目は，最初のアドレスにおいて，スタックが空であることを表す．5 行目は，各アドレスにおいて型の整合性がとれていることを表している．

3.4 型システムの健全性

本節では，型システムの健全性として，型付けできるプログラムは以下の条件を満たすことを示す．

- プログラムの入力中の機密度が高いデータを変更しても，実行結果中のそれよりも機密度が低いデータには影響を与えない．
- 実行結果に機密度が低いデータが含まれるとき，プログラムの入力中の機密度が高いデータを変更しても，実行時間は定数倍でしか変化しない．

1 番目の性質は，通常の情報流解析の健全性の基準として採用されている非干渉性 (non-interference)[4] と呼ばれる性質である．この性質が成り立っても，プログラムの実行時間の観測によって機密度の高いデータの情報が漏れる場合がある．例えば $long\_computation$ が長い計算時間を

$$\textbf{if } b \textbf{ then } long\_computation \textbf{ else } 1$$

この式の実行結果が 1 であることがわかっても b の値を知ることはできないが，実行時間が分かれば b の値を推測することはできる．後で述べるように，2 番目の性質を用いると，このようなプログラムの実行時間からの機密情報の漏れを防ぐことができる．

以下に上の健全性の基準のより厳密な定義および証明を与える．まず値および状態の間の等価性を定義する．これは，通常の等価性と異なり，各値の機密度によって決まる．

定義 3.5 整数 v_1, v_2 とセキュリティレベル κ との間の 3 項関係 $v_1 \sim_\kappa v_2$ を以下によって定義する．

$$v_1 \sim_\kappa v_2 \Leftrightarrow v_1 = v_2 \vee \kappa = \mathbf{H}$$

さらに整数 v_1, v_2 と型 τ との間の 3 項関係 $v_1 \sim_\tau v_2$ を以下によって定義する．

$$v_1 \sim_\tau v_2 \Leftrightarrow (\tau = \top) \vee \exists\kappa.(\tau = \mathbf{Int}_\kappa \wedge v_1 \sim_\kappa v_2)$$

上の関係 $v_1 \sim_\kappa v_2$ は，v_1, v_2 のセキュリティレベルが κ であるとき，セキュリティレベルが $\mathbf{L}$ のデータしか観測できない者にとっては v_1 と v_2 が区別できない（言い換えれば，v_1, v_2 のセキュリティレベルが $\mathbf{L}$ であれば v_1 と v_2 は同一の値でなければならない）ことを表す†6．

定義 3.6 f, f' を局所変数の状態，s, s' をスタックの状態とし，F，S をそれぞれ局所変数の型，スタックの型とする．以下のすべての条件が成り立つとき，$(f, s) \sim_{F,S} (f', s')$ と書く．また条件 (i)，(iii) が成り立つとき，$f \sim_F f'$，条件 (ii)，(iv) が成り立つとき，$s \sim_S s'$ と書く．

(i) $dom(f) \cap dom(f') \supseteq dom(F)$

(ii) $dom(s) \cap dom(s') \supseteq dom(S)$

(iii) $\forall x \in dom(F).f(x) \sim_{F(x)} f'(x)$

(iv) $\forall i \in dom(S).s(i) \sim_{S(i)} s'(i)$

上の関係 $(f, s) \sim_{F,S} (f', s')$ は各スタックアドレス i および局所変数 x に格納されている値がそれぞれ型 $S(i), F(x)$ を持つと仮定すると，セキュリティレベルが $\mathbf{L}$ のデータしか見ることのできない者にとっては (f, s) と (f', s') とが区別できないことを表す．なお，定義より，$\sim_{(F,S)}$ は部分同値関係（partial equivalence relation）である．すなわち推移律と対称律を満たす．

以上の関係を用いて，型システムの健全性を次のように述べることができる．

定理 3.1 (型システムの健全性)

メソッド $((\kappa_1, \ldots, \kappa_n, \kappa_r), B)$ が型付け可能であるとする．すると，（B によって定まる）ある定数 c が存在し，$(1, f_1, \emptyset) \longrightarrow_B^m (pc_1, f_1', s_1')$，$\kappa_r = \mathbf{L}$ かつ $B(pc_1) = \mathrm{return}$ ならば $\forall i \in \{1, \ldots, n\}.f_1(i) \sim_{\kappa_i} f_2(i)$ を満たす任意の局所変数の状態 f_2 について，ある pc_2, f_2', s_2' が存在して以下の条件が成り立つ．

---

†6 セキュリティレベルが一般の束構造をなす場合には，観測者のセキュリティレベルとして $\mathbf{L}$ 以外のものを考える必要が在るので，上の関係 $v_1 \sim_\kappa v_2$, $v_1 \sim_\tau v_2$ を「セキュリティレベルが κ_o 以下のデータしか観測できない者にとっては v_1 と v_2 を区別できない」ことを表す $v_1 \sim_\kappa^{\kappa_o} v_2$, $v_1 \sim_\tau^{\kappa_o} v_2$ で置き換える必要がある．さらに下の定義 3.6も同様に変更し，定理 3.1は $\kappa_r = \mathbf{L}$ という条件を取り除き，観測者のセキュリティレベルを κ_r とする（$\sim_{\_}$ を $\sim_{\_}^{\kappa_r}$ で置き換える）必要がある．

(i) $(1, f_2, \emptyset) \longrightarrow_B^{\leq cm} (pc_2, f'_2, s'_2)$

(ii) $B(pc_2) = \mathrm{return}$

(iii) $s'_1(0) \sim_{\kappa_r} s'_2(0)$

直観的には，(セキュリティレベルが $\mathbf{L}$ のデータしか見ることができない者にとって) 区別不能な入力が与えられれば結果として返される値 (上の $s'_1(0)$ と $s'_2(0)$) も区別不能であり，実行時間は高々 c 倍しか異ならないことを表す．

上の事実を用いると，次のようにしてメソッドの実行時間からの情報漏れを防ぐことができる．(なお，実行の各ステップにかかる時間は定数であるとする．実際の計算機でこれを保証するには必要に応じて各ステップごとに遅延を挿入すれば良い．) 返り値の機密度が $\mathbf{L}$ であるメソッドが起動された場合，

(1) まず機密度が $\mathbf{H}$ のデータをすべて 0 にした上でメソッドを実行する．そのときの実行ステップを n とする．

(2) 次に機密度が $\mathbf{H}$ のデータを正しい値に戻した上でメソッドを実行する．そのときの実行ステップを m とする．

(3) $cn - m$ ステップ分の時間待った後に，実行結果を返す．(定理より，$cn - m$ は非負整数であることに注意．)

このようにすれば，実行結果が返るまでの時間は，機密度が $\mathbf{H}$ のデータにかかわらず，常に $(c+1)n$ である．従って，実行結果を受け取った者は実行時間を観測しても機密度が $\mathbf{H}$ のデータの値を推測することはできない[†7]．

以下に定理 3.1の証明を示す．まずいくつかの補題を証明する．次の補題は，互いに区別できない状態から命令を 1 つ実行した場合，その命令が機密度が高い値に関する分岐命令でないかぎり，実行後の状態も区別できないことを表す．

補題 3.1 以下の条件 (i) ~ (iv) が成り立つと仮定する．

(i) $\mathcal{F}, \mathcal{S}, \mathcal{A} \vdash ((\kappa_1, \ldots, \kappa_n, \kappa_r), B)$

(ii) $(pc, f_1, s_1) \longrightarrow_B (pc', f'_1, s'_1)$

(iii) $B(pc) = \mathrm{if}\ l$ ならば $\mathcal{S}(pc)(0) = \mathbf{Int_L}$

(iv) $(f_1, s_1) \sim_{(\mathcal{F}(pc), \mathcal{S}(pc))} (f_2, s_2)$

すると，以下の条件 (v) および (vi) を満たす f'_2, s'_2 が存在する．

(v) $(pc, f_2, s_2) \longrightarrow_B (pc', f'_2, s'_2)$

(vi) $(f'_1, s'_1) \sim_{(\mathcal{F}(pc'), \mathcal{S}(pc'))} (f'_2, s'_2)$

証明 付録 B.1を参照．□

命令が機密度の高い値に関する分岐命令の場合には，(セキュリティレベルが $\mathbf{L}$ 以下のデータしか観測できない者にとって) 等価な状態であっても分岐結果が異なる場合があるので上の補題は成り立たない．しかし，その場合には合流点に達するまではアドレスのセキュリティレベルが $\mathbf{H}$ のままであるので，セキュリティレベルが $\mathbf{L}$ 以下のデータしか観測できない者にとっては，スタックの形を除いては状態は変化しない．これを示すのが以下の補題である．

補題 3.2 ある自然数 n について以下の条件 (i) ~ (iv) が成り立つと仮定する．

(i) $\mathcal{F}, \mathcal{S}, \mathcal{A} \vdash ((\kappa_1, \ldots, \kappa_n, \kappa_r), B)$.

(ii) $\forall i \in \{0, \ldots, m-1\}.$
$$(pc_i, f_i, s_i) \longrightarrow_B (pc_{i+1}, f_{i+1}, s_{i+1})$$

(iii) $\forall i \in \{0, \ldots, m-1\} \mathcal{A}(pc_i) = \mathbf{H}$.

(iv) $(f_0, s_0) \sim_{(\mathcal{F}(pc_0), \mathcal{S}(pc_0))} (f'_0, s'_0)$.

すると，以下の条件 (v) が成り立つ．

(v) $$(\uparrow_{\mathcal{F}(pc_m)}(f'_0), \uparrow_{\mathcal{S}(pc_0)}^{\mathcal{S}(pc_m)}(s'_0)) \sim_{(\mathcal{F}(pc_m), \mathcal{S}(pc_m))} (f_m, s_m)$$

ただし，$dom(S) = \{0, \ldots, n-1\}, dom(S') = \{0, \ldots, n'-1\}$ のとき，$\uparrow_F(f)$ および $\uparrow_S^{S'}(s)$ は以下によって定義する．

$$dom(\uparrow_F(f)) = dom(F)$$
$$(\uparrow_F(f))(x) = \begin{cases} f(x) & \text{if } x \in dom(f) \\ \star & \text{if } x \in dom(F) \backslash dom(f) \end{cases}$$
$$dom(\uparrow_S^{S'}(s)) = dom(S')$$
$$\uparrow_S^{S'}(s)(i) = \begin{cases} s(i + n' - n) & \\ \quad \text{if } i + n' - n \in dom(s) & \\ \star \quad \text{if } i + n' - n \notin dom(s) & \end{cases}$$

†7 なお，このような隠蔽が可能であるためには，実行時間が定数倍で押さえられることは必須ではなく，一般にはある関数 g が存在して $g(n)$ で押さえられれば十分である．しかし $g(n)$ ステップの遅延を挿入しなければならないことを考慮すると，実用上は本論文の場合のように g が線形関数であることが望ましいと思われる．

$\star$ は実際の計算には出現しない特別な値であり，関係 $\sim_\kappa$ を以下によって $\mathbf{Nat} \cup \{\star\}$ 上の同値関係に拡張する．

$$\star \sim_\kappa v \Leftrightarrow v \sim_\kappa \star \Leftrightarrow (\kappa = \mathbf{H} \vee v = \star)$$

証明 付録 B.2を参照．□

上の 2 つの補題を用いて以下の補題を示すことができる．これは互いに等価な状態から実行を行って命令 return に至った場合，両者の返り値は等価であることを表す．定理 3.1はこの補題の系である．

補題 3.3 以下の条件 (i) ~ (iv) が成り立つと仮定する．

(i) $\mathcal{F}, \mathcal{S}, \mathcal{A} \vdash ((\kappa_1, \ldots, \kappa_n, \mathbf{L}), B)$

(ii) $(pc, f_1, s_1) \longrightarrow_B^m (pc', f_1', s_1')$

(iii) $B(pc') = \mathrm{return}$

(iv) $(f_1, s_1) \sim_{(\mathcal{F}(pc), \mathcal{S}(pc))} (f_2, s_2)$

B の定義域 $dom(B)$ の要素数を c とすれば，以下の条件 (v) および (vi) を満たす f_2', s_2' が存在する．

(v) $(pc, f_2, s_2) \longrightarrow_B^{\leq cm} (pc', f_2', s_2')$

(vi) $s_1'(0) \sim_\mathbf{L} s_2'(0)$

証明 m に関する数学的帰納法により示す．

$m = 0$ の場合は，(i) ~ (iv) および return 命令の型つけ規則より，$pc' = pc,\ s_1'(0) = s_1(0) = s_2(0)$ が成り立つ．したがって，$f_2' = f_2, s_2' = s_2$ とすれば (v) および (vi) が成り立つ．

$m \geq 1$ の場合，$B(pc) = \mathrm{if}\ l \wedge \mathcal{S}(pc)(0) = \mathbf{Int_H}$ であるかどうかによる場合分けを行う．

$B(pc) = \mathrm{if}\ l \wedge \mathcal{S}(pc)(0) = \mathbf{Int_H}$ でない場合，

$$(pc, f_1, s_1) \longrightarrow_B (pc'', f_1'', s_1'') \longrightarrow_B^{m-1} (pc', f_1', s_1')$$

を満たす pc'', f_1'', s_1'' が存在する．補題 3.1より，次の条件を満たす f_2'', s_2'' が存在する．

$$(pc, f_2, s_2) \longrightarrow_B (pc'', f_2'', s_2'')$$
$$(f_1'', s_1'') \sim_{(\mathcal{F}(pc''), \mathcal{S}(pc''))} (f_2'', s_2'')$$

よって，帰納法の仮定より，

$$(pc'', f_2'', s_2'') \longrightarrow_B^{\leq c(m-1)} (pc', f_2', s_2')$$
$$s_1'(0) \sim_\mathbf{L} s_2'(0)$$

を満たす f_2', s_2' が存在する．$1 + c(m-1) \leq cm$ より，$(pc, f_2, s_2) \longrightarrow_B^{\leq cm} (pc', f_2', s_2')$ が成り立つ．

$B(pc) = \mathrm{if}\ l \wedge \mathcal{S}(pc)(0) = \mathbf{Int_H}$ である場合．返り値のセキュリティレベルが $\mathbf{L}$ なので，型付け規則より $\mathcal{A}(pc') = \mathbf{L}$ でなければならない．したがって，pc からの最初の合流点 l' が存在し，

$$(pc, f_1, s_1) \longrightarrow_B^{m_1} (l', f_1'', s_1'') \longrightarrow_B^{m-m_1} (pc', f_1', s_1')$$
$$(pc, f_2, s_2) \longrightarrow_B^{m_2} (l', f_2'', s_2'')$$

を満たす $f_1'', s_1'', f_2'', s_2''$ が存在する．ここで，「最初の合流点」の定義から $1 \leq m_1, m_2 \leq c$ が成り立つので，$m_2 \leq cm_1$ が成り立つ．さらに規則 (T-IF) より，実行ステップ $(pc, f_1, s_1) \longrightarrow_B^{m_1} (l', f_1'', s_1'')$ および $(pc, f_2, s_2) \longrightarrow_B^{m_2} (l', f_2'', s_2'')$ の中間状態のプログラムカウンタのセキュリティレベルはすべて $\mathbf{H}$ である．したがって，補題 3.2により，

$$\begin{array}{ll} & (f_1'', s_1'') \\ \sim_{(\mathcal{F}(l'), \mathcal{S}(l'))} & (\uparrow_{\mathcal{F}(l')}(f_2), \uparrow_{\mathcal{S}(pc)}^{\mathcal{S}(l')}(s_2)) \\ \sim_{(\mathcal{F}(l'), \mathcal{S}(l'))} & (f_2'', s_2'') \end{array}$$

が成り立つ．よって帰納法の仮定より，

$$(l', f_2'', s_2'') \longrightarrow_B^{\leq c(m-m_1)} (pc', f_2', s_2')$$
$$s_1'(0) \sim_\mathbf{L} s_2'(0)$$

を満たす f_2', s_2' が存在する．$m_2 + c(m - m_1) \leq cm_1 + c(m - m_1) = cm$ より，$(pc, f_2, s_2) \longrightarrow_B^{\leq cm} (pc', f_2', s_2')$ が成り立つ．□

4 型推論アルゴリズム

定理 3.1で述べたように，前節の型システムで型付けできるプログラムは，秘密の情報を漏らさない．したがって，プログラムが秘密の情報を漏らさないことを確認するためには，与えられたプログラム（メソッド）M に対して型推論を行い，定義 3.4の関係 $\mathcal{F}, \mathcal{S}, \mathcal{A} \vdash M$ を満たす $\mathcal{F}, \mathcal{S}, \mathcal{A}$ が存在することを確認すれば十分である．型推論は通常の制約に基づく型推論[7, 9] と同様に，次のステップで行うことができる．

(1) 型判定規則に基づき，メソッド本体の各アドレスから型やセキュリティレベルに関する制約を生成する．

(2) 生成された制約を簡約し，充足可能性を調べる．

以下では，例 2.1のプログラムを通して型推論の概略を示す．単純化のため，セキュリティレベルを除いた通常の型情報は（Stata と Abadi の型システム[18] など

を用いて)すでに得られているものとする．表 1に，例 2.1のメソッド $((\mathbf{H},\mathbf{L}),B)$ に対する型環境 $\mathcal{F},\mathcal{S},\mathcal{A}$ のうち，セキュリティレベル以外の型情報を示す．セキュリティレベルについては未知として変数 $\alpha_l,\beta_l,\gamma_l,\delta_l$ を割り当てている．図 3の規則から最右列に示す制約を得ることができる．例えば，4 行目の store 2 命令に関する制約は，図 3の規則によれば

$$S_4 \leq \mathcal{F}_5(2)\cdot S_5$$
$$\mathbf{Int}_{\mathcal{A}(4)} \leq \mathcal{F}_5(2)$$
$$\mathcal{F}_4\backslash 2 \leq \mathcal{F}_5\backslash 2$$

である．これに表 1の型を代入すると

$$\mathbf{Int}_{\alpha_4}\cdot\emptyset \leq \mathbf{Int}_{\beta_5}\cdot\emptyset$$
$$\mathbf{Int}_{\delta_4} \leq \mathbf{Int}_{\beta_5}$$
$$\{1\mapsto \mathbf{Int}_{\alpha_4}\} \leq \{1\mapsto \mathbf{Int}_{\alpha_5}\}$$

が得られる．これらの制約を 3.1節の部分型関係の定義に従って簡約すると，表 1の最右列に示したような制約

$$\alpha_4 \sqsubseteq \beta_5, \delta_4 \sqsubseteq \beta_5, \alpha_4 \sqsubseteq \alpha_5$$

が得られる．

変数を $\vec{\rho} = (\alpha_2,\alpha_3,\ldots,\delta_{11})$ とおくと，表 1の制約は $\vec{g}(\vec{\rho}) \sqsubseteq \vec{\rho},\ (\gamma_{11},\delta_{11}) \sqsubseteq (\mathbf{L},\mathbf{L})$ の形にまとめることができる．ここで，$\sqsubseteq$ はセキュリティレベルの間の関係をセキュリティレベルの組の間の関係に自然に拡張をしてえられる関係であり，$\vec{g}$ はその関係に関して単調な関数である．したがって，ある自然数 n について，$\vec{\rho} = \vec{g}^n(\mathbf{L},\ldots,\mathbf{L})$ が $\vec{g}(\vec{\rho}) \sqsubseteq \vec{\rho}$ の最小解となる．上の例では実際に解を求めると($n = 0$ から順に $\vec{g}^n(\mathbf{L},\ldots,\mathbf{L})$ を計算していき，$\vec{g}^{n+1}(\mathbf{L},\ldots,\mathbf{L}) = \vec{g}^n(\mathbf{L},\ldots,\mathbf{L})$ となる n を探せばよい) $\gamma_{11} = \mathbf{H}$ となり，条件 $\gamma_{11} \sqsubseteq \mathbf{L}$ に反する．したがって，例 2.1のメソッド $((\mathbf{H},\mathbf{L}),B)$ は型つけ可能でないと判定できる．一方，$((\mathbf{H},\mathbf{H}),B)$ であれば型つけ可能である(よって秘密の情報を漏らさない)ことが同様に判定できる．

一般の場合も上と同様にして，$\vec{g}(\vec{\rho})\sqsubseteq\vec{\rho}, \vec{g}'(\vec{\rho})\sqsubseteq\vec{\kappa}$ の形の制約が得られるので，前者の最小解をまず求め，それが後者を満たすかを検査することによって型判定が行える．メソッドのサイズ(命令数)を n とするとき，各命令から生成される制約の数が $O(n)$ なので，制約の個数は $O(n^2)$ である．したがって，Rehof らの線形時間の制約解消アルゴリズム[16] を用いれば，型推論の計算時間は最悪でも $O(n^2)$ である．

5 議 論

5.1 他の低レベル言語への適用可能性について

本研究では低レベル言語として Java の仮想機械言語(の部分言語)を用いたが，対象の低レベル言語によっては，本研究の型システムをさらに拡張する必要がある．Java の仮想機械言語は，各命令アドレスにおけるスタックの深さが一定，ジャンプ命令の飛び先が決まっている，などの点で低レベル言語としては比較的「構造化」されたものであり，そのために本論文の型システムも比較的単純なものになっている．各命令アドレスにおけるスタックの深さや中身の型が定まらない(動的に変化する)などの問題については(Stata と Abadi[18] がサブルーチンの問題に対処したのと同様の考え方で)多相型などを用いてある程度対処できると思われる．しかしながら，実際の機械語のようにアドレスに関する演算を行ったのちにそのアドレスにジャンプする，などの命令がある場合には，制御構造自体が動的に変化するので，本研究のように型システムによって自動的に検証するという手法を適用するのは難しいと思われる．そのような場合には，型付アセンブリ言語[10] や Proof-Carrying Code[13] のように検証のための付加情報が必要となると思われる．

5.2 型システムの(不)完全性について

型つけできるプログラムは秘密の情報を漏らさないという意味で本研究の型システムは健全ではあるが，当然のことながらその逆は言えない．すなわち，秘密の情報を漏らさないにもかかわらず，型つけできないプログラムが存在するという意味で本研究の型システムは完全ではない．一般に任意のプログラムが与えられたときにそのプログラムが情報を漏らすかどうかを判定することは決定不能である(停止性問題に容易に還元できる)ので，完全性が成り立たないのは当然であるが，以下では本研究の型システムによる制限の妥当性について議論する．以下の議論は低レベル言語であることには依存しないので，わかりやすさのために手続き型言語や関数型言語の構文を用いる．

本研究の型システムによる制限で一番問題となると思われるのは，繰り返しや再帰における条件分岐に関する

l	$B(l)$	$\mathcal{F}_l(1)$	$\mathcal{F}_l(2)$	$\mathcal{S}_l$	$\mathcal{A}(l)$	制約
1	load 1	$\mathbf{Int}_{\mathbf{H}}$	$\top$	$\emptyset$	δ_1	$\mathbf{H} \sqsubseteq \alpha_2, \mathbf{H} \sqsubseteq \gamma_2, \delta_1 \sqsubseteq \gamma_2$
2	if 6	$\mathbf{Int}_{\alpha_2}$	$\top$	$\mathbf{Int}_{\gamma_2} \cdot \emptyset$	δ_2	$\alpha_2 \sqsubseteq \alpha_3, \alpha_2 \sqsubseteq \alpha_6, \forall k \in \{3,4,5,6,7,8,9\}.\gamma_2 \sqsubseteq \delta_k$
3	push0	$\mathbf{Int}_{\alpha_3}$	$\top$	$\emptyset$	δ_3	$\alpha_3 \sqsubseteq \alpha_4, \delta_3 \sqsubseteq \gamma_4$
4	store 2	$\mathbf{Int}_{\alpha_4}$	$\top$	$\mathbf{Int}_{\gamma_4} \cdot \emptyset$	δ_4	$\gamma_4 \sqsubseteq \beta_5, \delta_4 \sqsubseteq \beta_5, \alpha_4 \sqsubseteq \alpha_5$
5	goto 10	$\mathbf{Int}_{\alpha_5}$	$\mathbf{Int}_{\beta_5}$	$\emptyset$	δ_5	$\alpha_5 \sqsubseteq \alpha_{10}, \beta_5 \sqsubseteq \beta_{10}$
6	push0	$\mathbf{Int}_{\alpha_6}$	$\top$	$\emptyset$	δ_6	$\alpha_6 \sqsubseteq \alpha_7, \delta_6 \sqsubseteq \gamma_7$
7	inc	$\mathbf{Int}_{\alpha_7}$	$\top$	$\mathbf{Int}_{\gamma_7} \cdot \emptyset$	δ_7	$\gamma_7 \sqsubseteq \gamma_8, \delta_7 \sqsubseteq \gamma_8, \alpha_7 \sqsubseteq \alpha_8$
8	store 2	$\mathbf{Int}_{\alpha_8}$	$\top$	$\mathbf{Int}_{\gamma_8} \cdot \emptyset$	δ_8	$\gamma_8 \sqsubseteq \beta_9, \delta_8 \sqsubseteq \beta_9, \alpha_8 \sqsubseteq \alpha_9$
9	goto 10	$\mathbf{Int}_{\alpha_9}$	$\mathbf{Int}_{\beta_9}$	$\emptyset$	δ_9	$\alpha_9 \sqsubseteq \alpha_{10}, \beta_9 \sqsubseteq \beta_{10}$
10	load 2	$\mathbf{Int}_{\alpha_{10}}$	$\mathbf{Int}_{\beta_{10}}$	$\emptyset$	δ_{10}	$\alpha_{10} \sqsubseteq \alpha_{11}, \beta_{10} \sqsubseteq \beta_{11}, \beta_{10} \sqsubseteq \gamma_{11}, \delta_{10} \sqsubseteq \gamma_{11}$
11	reurn	$\mathbf{Int}_{\alpha_{11}}$	$\mathbf{Int}_{\beta_{11}}$	$\mathbf{Int}_{\gamma_{11}} \cdot \emptyset$	δ_{11}	$\gamma_{11} \sqsubseteq \mathbf{L}, \delta_{11} \sqsubseteq \mathbf{L}$

表 1 例 2.1のメソッドに対するセキュリティレベル以外の型づけおよび制約

制約である．我々の型システムでは

(i) 機密度が高いデータに関する条件を用いた繰り返しや再帰 (**while** $secret\_data > 0$ **do** $\cdots$ など)

(ii) 機密度が高いデータに依存する条件による分岐の中での（繰り返しの条件で用いるデータの機密度にかかわらず）繰り返しや再帰

（**if** $secret\_data > 0$ **then while** b **do** $\cdots$ **else** $\cdots$ など）

は許されない．しかしながら，機密度の低いデータに関する繰り返しや再帰は許されるので，有用なプログラムを書くことは可能である．例えば，他人の銀行口座に振込みを行う場合を考えると，振込み処理をする関数 $transfer$ は，口座番号，振込み金額，銀行口座のデータベースの組をとって，振込みが成功したかおよび更新されたデータベースを返す，図 4に示すような関数として表すことができる．ここで，銀行口座のデータベースは口座番号と預金額の組のリストとして表現し，預金額は振り込んだ者に知られてはならないのでそのセキュリティレベルは $\mathbf{H}$ である．セキュリティレベルは $\{\}$ 内に記述されている．このプログラムには再帰呼び出しが含まれているが，条件分岐に用いているデータのセキュリティレベルは $\mathbf{L}$ なので我々の型システムと同様の制約の下でも型つけすることができる．

一方，最近の情報流解析のための型システムの多くの研究[15, 22] では，上のような（繰り返しや再帰の条件分岐に関する）制限は厳しすぎるとして，そのような制限は課さず，プログラムの停止性が機密度の高いデータに依存するのをあえて許している．しかしこれにより，プログラムの 1 回の実行のたびに平均して 1 ビットの情報が漏れてしまう．より現実的で安全な改善方法としては，繰り返しや再帰の条件分岐に関する制限を緩め，その代わりとしてプログラムの停止性を別の方法によって解析することが考えられる．停止性が保証されかつ情報流解析の型検査を通るプログラムのみを受理すれば，プログラムの停止性の問題による情報漏れは起きない．ただし，我々の型システムのように実行時間からの情報漏れをも防ぐには，停止性のみでなく実行時間に関する解析も行う必要がある．

我々の情報流解析の型システムでは機密情報を全く漏らさないことを保証することを目的としているが，現実には 1節で触れた login プログラムのように，ある程度の情報漏れは許容したい場合も多い．そのような場合は，機密度の高いデータから低いデータを生成することを許す特殊なプリミティブを用意することによって対処することができる．例えば，暗号化されたデータは元のデータの情報に依存しているが，暗号システムが十分堅牢であれば，暗号化されたデータから元のデータを推測することは現実的には不可能である．そこで，そのような暗号システムの暗号化関数 $encode$ を次のような型を持つプリミティブとして言語に追加することができる．

$$\mathbf{String}_{\mathbf{H}} \times \mathbf{Key}_{\mathbf{H}} \rightarrow \mathbf{String}_{\mathbf{L}}$$

ただし，そのようなプログラムの安全性を厳密に検証す

```
fun transfer(account:int{L}, amount:int{L}, database:(int{L}*int{H}) list{L})
                                          : bool{L} * (int{L}*int{H}) list{L}) =
  Case database of
        [] => (false, database)
      | (accunt',money)::database' =>
          if account=account' then (true, (account', amount+money))::database'
          else
            let val (result, new_database)=transfer(account,amount,database')
            in (result, (account',money)::new_database) end
```

図4 型つけできるプログラム例

るには確率的議論[20] が必要である．

6 関連研究

1節に述べたように，情報流解析の問題についてはすでに，手続き型言語[3, 21]，関数型言語[4, 15, 22]，オブジェクト指向言語[12] 並列言語[5, 17] などを対象として多くの研究がなされている．しかしながら，我々の知る限りにおいては本研究で扱ったような低レベル言語に対して明確な解析方法を与え，かつ実行時間による情報漏れにも対処したのは本研究が初めてである．

Denning[3] は，手続き型言語のための情報流解析の拡張として，goto 文の扱いについても簡単にふれている．しかしながら，そこで示されている解析方法では，3.2節で我々が導入した「最初の合流点」の代わりに，immediate forward dominator（無限の実行経路を無視したもの）[11] が用いられており，少なくとも我々の情報漏れの基準では正しくない（実際，彼らはその解析方法の正しさを証明していない）．例えば以下のメソッド本体 B を持つメソッド $((\mathbf{H}, \mathbf{L}), B)$ を考える．

l	$B(l)$	説明
1	load 1	引数 1 の値をロード
2	if 1	ロードした値が 0 でない時 1 へ分岐
3	push0	
4	return	

immediate forward dominator は 3 であるので，3 行目以降を実行したという情報が漏れてもよいことになり，彼らの解析方法では正しいプログラムとなる．しかしながら，上のプログラムは実行が終了する（返り値が存在する）かどうかを観測することにより引数の情報を知ることができてしまう．

Myers [22] の情報流解析の型システムでは関数型言語に継続 (continuation) を導入することにより大域ジャンプを扱っているが，この型システムでも同様にプログラムの停止したかの情報が漏れてしまい，我々の基準では正しくない．また，彼らの型システムでは継続の使い方の制限が強いので，本研究が対象とするような低レベル言語のプログラムに出現するジャンプ命令がそのまま扱えるのかどうかは自明ではない．

実行時間からの情報漏れを保証する方法，Volpano と Smith [19] や Agat [2] によって研究されている．Volpano と Smith [19] は，実行時間が機密度の高いデータの値に全く影響を受けない型システムを示しているが，この型システムでは機密度の高いデータに関する条件分岐を禁じているという点で制限がきつすぎる．Agat [2] はプログラム変換によって実行時間の情報を隠蔽する方法を提案している．これは，機密度の高いデータの値に関する if 文において，then 部と else 部の実行時間が同一になるように，then 部には else 部のダミーの式を，else 部には then 部のダミーを挿入する方法である．本研究の遅延を挿入する方法と異なり，この方法では無駄な計算コストがかかる．また，本研究で扱ったような if 文が構造化されていない低レベル言語に同じ方法を適用するのは難しいと思われる．

本研究の型システムは，Stata と Abadi による Java の仮想機械言語の型システム[18] にセキュリティレベルを加えて拡張したものである．彼らの型システムではサブルーチンについても扱っているが，本研究の型システムでも同様にサブルーチンを扱うように変更すること

は容易であると思われる．

Abadi らの依存性解析の研究[1] により，情報流解析も部分評価のための束縛時解析も依存性解析の一種としてとらえることができ，両者が密接に関係することが指摘されている．したがって，本研究の情報流解析の型システムと増原らによる Java バイトコードの束縛時解析のための型システム[8] にも密接な関連があるはずである．実際，本研究の情報流解析と同様， Java バイトコードの束縛時解析においても分岐の扱いに関して（高レベル言語のための解析と比較して）微妙な問題が生じる．この問題に関して増原ら[8] はややアドホックな方法で解決しているが，本研究の方法を取り入れることで，より明快な束縛時解析のための型システムが得られることが期待される．両者の詳細な比較は今後の課題である．

7 結　論

Java の仮想機械言語の部分言語を対象として，プログラムが秘密の情報を漏らさないかを判定するための型システムを構築した．これにより， Java の通常のバイトコード検証と同様にして，ソースプログラムではなくバイトコードについて直接情報流解析が行うことが可能になる．

今後の課題として，本論文で扱っていない命令（特にメソッド呼び出しを含むオブジェクトに関する命令，サブルーチン，スレッドプリミティブ）を追加して型システムを拡張し，それに基づいて現実のプログラムが秘密の情報を漏らしていないかを判定する検証器を実装することが挙げられる．また，型付アセンブリ言語[10] などの他の低レベル言語に対する情報流解析の型システムを構築することも今後の課題である．

謝　辞

本研究の内容に関して一緒にご議論いただいた東京大学米澤研究室および東京工業大学小林研究室の「プログラミング言語の基礎」グループのメンバーの方々，有益なコメントをくださった査読者および編集委員の方々に感謝いたします．

参考文献

[1] Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Rieck. A core calculus of dependency. In *Proc. of POPL*, pages 147–169, 1999.

[2] Johan Agat. Transforming out timing leaks. In *Proc. of POPL*, pages 40–53, 2000.

[3] Dorothy E. Denning and Peter J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, 1977.

[4] Nevin Heintze and Jon Riecke. The slam calculus: programming with secrecy and integrity. In *Proc. of POPL*, pages 365–377, 1998.

[5] Kohei Honda, Vasco Vasconcelos, and Nobuko Yoshida. Secure information flow as typed process behaviour. In *Proc. of European Symposium on Programming (ESOP) 2000*, volume 1782 of *LNCS*, pages 180–199. Springer-Verlag, 2000.

[6] Kohei Honda and Nobuko Yoshida. A uniform type structure for secure information flow. In *Proc. of POPL*, pages 81–92, 2002.

[7] Atsushi Igarashi and Naoki Kobayashi. Type reconstruction for linear pi-calculus with I/O subtyping. *Info. Comput.*, 161:1–44, 2000.

[8] Hidehiko Masuhara and Akinori Yonezawa. A portable approach to dynamic optimization in runtime specialization. *New Generation Computing*, 20(1):101–124, 2002.

[9] Torben Mogensen. Types for 0, 1 or many uses. In *Implementation of Functional Languages*, volume 1467 of *LNCS*, pages 112–122, 1998.

[10] J. Gregory Morrisett, David Walker, Karl Crary, and Neal Glew. From system F to typed assembly language. *ACM Trans. Prog. Lang. Syst.*, 21(3):527–568, 1999.

[11] Steven S. Muchnick. *Advanced Compiler Design Implementation.* Morgan Kaufmann Publishers, 1997.

[12] Andrew C. Myers. JFlow: Practical mostly-static information flow control. In *Proc. of POPL*, pages 228–241, 1999.

[13] George C. Necula. Proof-carrying code. In *Proc. of POPL*, pages 106–119, 1997.

[14] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis.* Springer-Verlag, 1999.

[15] François Pottier and Vincent Simonet. Information flow inference for ML. In *Proc. of POPL*, pages 319–330, 2002.

[16] Jakob Rehof and Torben Mogensen. Tractable constraints in finite semilattices. *Science of Computer Programming*, 35(2):191–221, 1999.

[17] Geoffrey Smith and Dennis Volpano. Secure information flow in a multi-threaded imperative language. In *Proc. of POPL*, pages 355–364, 1998.

[18] Raymie Stata and Martín Abadi. A type system for java bytecode subroutines. *ACM Trans. Prog. Lang. Syst.*, 21(1):90–137, 1999.

[19] Dennis Volpano and Geoffrey Smith. Eliminat-

ing covert flows with minimum typings. In *Proc. 10th IEEE Computer Security Foundations Workshop*, pages 156–168, 1997.

[20] Dennis Volpano and Geoffrey Smith. Verifying secrets and relative secrecy. In *Proc. of POPL*, pages 268–276, 2000.

[21] Dennis Volpano, Geoffrey Smith, and Cynthia Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.

[22] Steve Zdancewic and Andrew C. Myers. Secure information flow via linear continuations. *Higher-Order and Symbolic Computation*, 2002. To appear. A preliminary version appeared in Proceedings of ESOP 2001, Springer LNCS 2028, pp.46–61.

A 制御情報の伝播範囲 $depend(l)$ の定義

本節では，3.2節で非形式的に導入した制御情報の伝播範囲 $depend(l)$ の厳密な定義およびそれを求めるためのアルゴリズムを与える．

まず各命令アドレスの「次の」アドレスの集合を与える関数 $next$ を定義する．

定義 A.1 B をメソッド本体，l を $dom(B)$ の要素とする．このとき，$next_B(l)$ は以下によって定義されるアドレスの集合である．

$$next_B(l) = \begin{cases} \{l+1, l'\} & \text{if } B(l) = \text{if } l' \\ \{l'\} & \text{if } B(l) = \text{goto } l' \\ \emptyset & \text{if } B(n) = \text{return} \\ \{l+1\} & \text{それ以外の時} \end{cases}$$

例えば，例 2.1のメソッド本体 B について $next_B(2) = \{3, 6\}$ である．

続いて「完全実行経路」を定義する．

定義 A.2 (完全実行経路)B をメソッド本体，アドレス l を $dom(B)$ の要素とする．B における l からの完全実行経路とはアドレスの列 $p(\in \mathbf{Adr}^* \cup \mathbf{Adr}^\omega)$ で以下の条件のいずれかを満たすものである．

(i) p は無限列 $l_0 l_1 l_2 \cdots$(すなわち $p \in \mathbf{Adr}^\omega$) であり，$l_0 \in next_B(l)$ かつ任意の自然数 i について $l_{i+1} \in next_B(l_i)$．

(ii) p は有限列 $l_0 l_1 \cdots l_n$(すなわち $p \in \mathbf{Adr}^*$) で $l_0 \in next_B(l),\ B(l_n) = \text{return}$ かつ任意の $i(\in \{0, \ldots, n-1\})$ について $l_{i+1} \in next_B(l_i)$．

(iii) p は空列 ϵ で $B(l) = \text{return}$．

B における l からの完全実行経路の集合を $full\_path_B(l)$ と書く．アドレスの列 p の i 番目の要素を $p(i)$ と書く．

定義 A.3 (合流点) B をメソッド本体，$l \in dom(B)$ とする．B における l からの合流点の集合 $merge_B(l)$ を以下の等式により定義する．

$$merge_B(l) = \{l' \mid \forall p \in full\_path_B(l).\exists i.(p(i) = l')\}$$

定義 A.4 (最初の合流点) B をメソッド本体，アドレス l を $dom(B)$ の要素とする．B における l からの最初の合流点 $fmerge_B(l)$ とは，以下の集合 ($fmrg_B(l)$ と書く) の要素で最も (アドレスの大小関係について) 小さいものである．

$$\begin{aligned} \{l' \mid l' \in merge_B(l) \wedge & \\ \exists p \in full\_path_B(l).\exists i.(p(i) = l' \wedge & \\ \forall j.(j < i \Rightarrow p(j) \notin merge_B(l)))\} & \end{aligned}$$

集合 $fmrg_B(l)$ が空の場合は $fmerge_B(l)$ を未定義とする．

定義 A.5 (制御情報の伝播範囲) B をメソッド本体，$l \in dom(B)$ とする．B における l の制御情報の伝播範囲 $depend_B(l)$ とは以下に与えられるアドレスの集合である．

$$\begin{aligned} \{l' \mid \exists p \in full\_path_B(l).\exists i.(p(i) = l' \wedge & \\ \forall j.(j < i \Rightarrow p(j) \neq fmerge_B(l))\} & \end{aligned}$$

なお，$fmerge_B(l)$ が未定義の時は l から到達可能なすべての点の集合，すなわち，

$$\{l' \mid \exists p \in full\_path_B(l).\exists i.p(i) = l'\}$$

とする．

以下では，$depend_B(l)$ を求めるアルゴリズムを与える．まず合流点の集合 $merge_B(l)$ は，以下の等式の最小解として求めることができる．

$$merge_B(l) = \begin{cases} \bigcap_{l' \in next_B(l)} (\{l'\} \cup merge_B(l')) & \\ & \text{if } next_B(l) \neq \emptyset \\ \emptyset & \text{if } next_B(l) = \emptyset \end{cases}$$

最初の合流点の候補集合 $fmrg_B(l)$ は次の等式の最小解 $first_B(l, S)$ を用いて $first_B(l, merge_B(l))$ とすればよい．

$$\begin{aligned} first_B(l, S) = & (next_B(l) \cap S) \\ & \cup (\textstyle\bigcup_{l' \in next_B(l) \setminus S} first_B(l', S)) \end{aligned}$$

最後に $depend_B(l)$ は，次の等式の最小解 $btwn_B(l_1, l_2)$ を用いて $btwn_B(l, fmerge_B(l))$ とすればよい．

$$btwn_B(l_1, l_2) = \bigcup_{l'_1 \in next_B(l_1) \setminus \{l_2\}} (\{l'_1\} \cup btwn_B(l'_1, l_2))$$

なお，3.2節の脚注に述べたように，任意の命令アドレス l について，命令アドレス l からのいずれかの完全

実行経路が有限列であれば，l からの最初の合流点の候補（上の $fmrg_B(l)$ の要素）は高々 1 つであることが証明できる．

今，l からの合流点および有限の完全実行経路が存在するとする．任意の完全実行経路は，合流点 n，合流点を含まない有限列 s_0，有限または無限列 s_1 を用いて $s_0 n s_1$ の形に表すことができる．有限の完全実行経路の存在から，s_1 が有限列であるものが存在する．そこで、s_1 の部分が最短の完全実行経路 $s_0 n s_1$ を考える．仮に n とは異なる最初の合流点の候補 n' を持つ完全実行経路 $s'_0 n' s'_1$ が存在すると仮定する．n' は $s_0 n s_1$ にも含まれなければならないので，$s_1 = s_2 n' s_3$ を満たす列 s_2, s_3 が存在する．すると，$s'_0 n' s_3$ も完全実行経路となるが，s_3 は s_1 よりも短いので経路 $s_0 n s_1$ の選び方に反する．以上から，最初の合流点の候補は高々 1 つしか存在しない．

有限の完全実行経路が存在しない場合（すなわちどのような分岐をしても return 命令に辿りつけない袋小路となっている場合）には，最初の合流点の候補は必ずしも 1 つではない．例えば以下のようなプログラムの断片では，1 からの完全実行経路は $2 \cdot 3 \cdot 4 \cdot 2 \cdot \ldots$ および $3 \cdot 4 \cdot 2 \cdot 3 \cdot \ldots$ なので，最初の合流点の候補は 2 および 3 となる．

l	$B(l)$
1	if 3
2	inc
3	inc
4	goto 2

B　補題の証明

B.1　補題 3.1の証明

補題 B.1 $v \sim_\kappa v'$ かつ $\kappa \sqsubseteq \kappa'$ ならば，$v \sim_{\kappa'} v'$ が成り立つ．

証明　$v \sim_\kappa v'$ の定義より明らか．□

系 B.1

(1)　$(v \sim_\tau v' \wedge \tau \leq \tau') \Rightarrow v \sim_{\tau'} v'$.

(2)　$(f \sim_F f' \wedge F \leq F') \Rightarrow f \sim_{F'} f'$.

(3)　$(s \sim_S s' \wedge S \leq S') \Rightarrow s \sim_{S'} s'$.

証明 補題 B.1および定義より明らか．□

補題 B.2任意の自然数 v, v' および型 τ について，$v \sim_\tau v'$ が成り立つならば $v + 1 \sim_\tau v' + 1$ が成り立つ．

証明　定義より明らか．□

証明（補題 **3.1**）　以下の条件 (i) ~ (iv) が成り立つと仮定する．

(i)　$\mathcal{F}, \mathcal{S}, \mathcal{A} \vdash ((\kappa_1, \ldots, \kappa_n, \kappa_r), B)$

(ii)　$(pc, f_1, s_1) \longrightarrow_B (pc', f'_1, s'_1)$

(iii)　$B(pc) = $ if l ならば $\mathcal{S}(pc)(0) = \mathbf{Int_L}$

(iv)　$(f_1, s_1) \sim_{(\mathcal{F}(pc), \mathcal{S}(pc))} (f_2, s_2)$

$(pc, f_1, s_1) \longrightarrow_B (pc', f'_1, s'_1)$ を導いた規則による場合分けにより，以下の条件 (v) および (vi) を満たす f'_2, s'_2 が存在することを示す．

(v)　$(pc, f_2, s_2) \longrightarrow_B (pc', f'_2, s'_2)$

(vi)　$(f'_1, s'_1) \sim_{(\mathcal{F}(pc'), \mathcal{S}(pc'))} (f'_2, s'_2)$

以下では規則 (R-Inc) および (R-IfT) の場合についてのみ示す．他の場合も同様である．

- 規則 (R-Inc) の時．規則より，

$$
\begin{array}{rcl}
B(pc) & = & \text{inc} \\
pc' & = & pc + 1 \\
f'_1 & = & f_1 \\
s'_1(i) & = & \begin{cases} s_1(i) + 1 & \text{if } i = 0 \\ s_1(i) & \text{if } i \in dom(s_1) \setminus \{0\} \end{cases}
\end{array}
$$

が成り立つ．

f'_2, s'_2 を

$$
\begin{array}{rcl}
f'_2 & = & f_2 \\
s'_2(i) & = & \begin{cases} s_2(i) + 1 & \text{if } i = 0 \\ s_2(i) & \text{if } i \in dom(s_2) \setminus \{0\} \end{cases}
\end{array}
$$

によって定義すると，条件 (v) および (vi) が満たされることを示す．条件 (v) は規則 (R-Inc) より明らか．

条件 (iv) より

$$dom(f_1) \cap dom(f_2) \supseteq dom(\mathcal{F}(pc))$$

$$dom(s_1) \cap dom(s_2) \supseteq dom(\mathcal{S}(pc))$$

が成り立つ．最初の条件より，

$$
\begin{array}{rcl}
dom(f'_1) \cap dom(f'_2) & = & dom(f_1) \cap dom(f_2) \\
& \supseteq & dom(\mathcal{F}(pc)) \\
& = & dom(\mathcal{F}(pc'))
\end{array}
$$

が成り立つ．同様に

$$dom(s_1') \cap dom(s_2') \supseteq dom(\mathcal{S}(pc'))$$

も成り立つ．

さらに，条件 (iv) および型付け規則 (T-INC) より，

$$\forall x \in dom(\mathcal{F}(pc)).f_1(x) \sim_{\mathcal{F}(pc)(x)} f_2(x)$$
$$\forall i \in dom(\mathcal{S}(pc)).s_1(i) \sim_{\mathcal{S}(pc)(i)} s_2(i)$$
$$\mathcal{F}(pc) \leq \mathcal{F}(pc')$$
$$\mathcal{S}(pc) \leq \mathcal{S}(pc')$$

が成り立つので，系 B.1および補題 B.2より，

$$\forall x \in dom(\mathcal{F}(pc')).f_1'(x) \sim_{\mathcal{F}(pc')(x)} f_2'(x)$$
$$\forall i \in dom(\mathcal{S}(pc')).s_1'(i) \sim_{\mathcal{S}(pc')(i)} s_2'(i)$$

が得られる．以上より，条件 (vi) が成り立つ．

- 規則 (R-IFT) の時．規則および条件 (iii) より，

$$\begin{array}{rcl} B(pc) & = & \text{if } l' \\ pc' & = & pc+1 \\ f_1' & = & f_1 \\ s_1 & = & 0 \cdot s_1' \\ \mathcal{S}(pc)(0) & = & \mathbf{Int_L} \end{array}$$

が成り立つ．f_2', s_2' を

$$\begin{array}{rcl} f_2' & = & f_2 \\ s_2'(i) & = & s_2(i+1) \text{ if } i+1 \in dom(s_2) \end{array}$$

によって定義すると，条件 (v) および (vi) が満たされることを示す．

条件 (iv) および $\mathcal{S}(pc)(0) = \mathbf{Int_L}$ より，$s_2(0) = s_1(0) = 0$．よって，規則 (R-IF) より (v) が成り立つ．

条件 (iv) より

$$dom(s_1) \cap dom(s_2) \supseteq dom(\mathcal{S}(pc))$$

が成り立つので，

$$\begin{array}{rl} & dom(s_1') \cap dom(s_2') \\ = & \{i \mid i+1 \in dom(s_1)\} \\ & \cap\{i \mid i+1 \in dom(s_2)\} \\ = & \{i \mid i+1 \in dom(s_1) \cap dom(s_2)\} \\ \supseteq & \{i \mid i+1 \in dom(\mathcal{S}(pc))\} \\ = & dom(\mathcal{S}(pc')) \end{array}$$

が得られる．同様に，

$$dom(f_1') \cap dom(f_2') \supseteq dom(\mathcal{F}(pc'))$$

も成り立つ．

さらに，条件 (iv) および型付け規則 (T-IF) より，

$$\forall x \in dom(\mathcal{F}(pc)).f_1(x) \sim_{\mathcal{F}(pc)(x)} f_2(x)$$
$$\forall i \in dom(\mathcal{S}(pc)).s_1(i) \sim_{\mathcal{S}(pc)(i)} s_2(i)$$
$$\forall x \in dom(\mathcal{F}(pc)).\mathcal{F}(pc)(x) \leq \mathcal{F}(pc')(x)$$
$$\forall i \in dom(\mathcal{S}(pc'))\mathcal{S}(pc)(i+1) \leq \mathcal{S}(pc')(i)$$

が成り立つので，系 B.1より，

$$\forall x \in dom(\mathcal{F}(pc')).f_1'(x) \sim_{\mathcal{F}(pc')(x)} f_2'(x)$$
$$\forall i \in dom(\mathcal{S}(pc')).s_1'(i) \sim_{\mathcal{S}(pc')(i)} s_2'(i)$$

が得られる．以上より，条件 (vi) が成り立つ．

□

B.2 補題 3.2の証明

補題 B.3 以下の条件 (i) ~ (iv) が成り立つと仮定する．

(i) $\mathcal{F}, \mathcal{S}, \mathcal{A} \vdash ((\kappa_1, \ldots, \kappa_n, \kappa_r), B)$

(ii) $(pc, f, s) \longrightarrow_B (pc', f', s')$

(iii) $\mathcal{A}(pc) = \mathbf{H}$

(iv) $(dom(f) \supseteq dom(\mathcal{F}(pc)))$
$\wedge(dom(s) \supseteq dom(\mathcal{S}(pc)))$

すると，以下の条件 (v) が成り立つ．

(v) $(\uparrow_{\mathcal{F}(pc')}(f), \uparrow_{\mathcal{S}(pc)}^{\mathcal{S}(pc')}(s)) \sim_{(\mathcal{F}(pc'),\mathcal{S}(pc'))} (f', s')$

証明 補題の (i) ~ (iv) を仮定する．(ii) の導出規則に関する場合分けにより，(v) が成り立つことを示す．以下では規則 (R-PUSH) および (R-IFT) の場合についてのみ示す．他の場合も同様である．

- 規則 (R-PUSH) の時．規則より，

$$\begin{array}{rcl} B(pc) & = & \text{push0} \\ pc' & = & pc+1 \\ f' & = & f \\ s' & = & 0 \cdot s \end{array}$$

が成り立つ．また，型付け規則 (T-PUSH) より，

$$\mathcal{F}(pc) \leq \mathcal{F}(pc')$$
$$\mathbf{Int_H} \cdot \mathcal{S}(pc) \leq \mathcal{S}(pc')$$

が成り立つ．従って，

$$\uparrow_{\mathcal{F}(pc')}(f) = f'$$
$$\uparrow_{\mathcal{S}(pc)}^{\mathcal{S}(pc')}(s)(i) = \begin{cases} \star & \text{if } i = 0 \\ s'(i) & \text{if } i \in dom(s')\backslash\{0\} \end{cases}$$

が成り立つ．これらと $\mathbf{Int_H} \leq \mathcal{S}(pc')(0)$ より，条件 (v) が成り立つ．

- 規則 (R-IfT) の時．規則より，

$$\begin{aligned} B(pc) &= \text{if } l' \\ pc' &= pc+1 \\ f'_1 &= f_1 \\ s_1 &= 0 \cdot s'_1 \end{aligned}$$

が成り立つ．また，型付け規則 (T-If) より，

$$\mathcal{F}(pc) \le \mathcal{F}(pc')$$
$$\mathcal{S}(pc) \le \mathbf{Int}_\kappa \cdot \mathcal{S}(pc')$$

が成り立つ．従って，

$$\uparrow_{\mathcal{F}(pc')}(f) = f'$$
$$\uparrow^{\mathcal{S}(pc')}_{\mathcal{S}(pc)}(s)(i) = s'$$

が成り立つ．よって，条件 (v) が成り立つ．

□

補題 **B.4** $(f,s) \sim_{(F,S)} (f',s')$ が成り立つとする．さらに，局所変数およびスタックの型 F', S' は次の条件を満たすものとする．

(i) $\forall x \in dom(F')\backslash\{y \mid F(y) = \mathbf{Int_L}\}.(\mathbf{Int_H} \le F'(x))$

(ii) $dom(S) = \{0,\ldots,n-1\}$，$dom(S') = \{0,\ldots,n'-1\}$ なる n, n' について，

$$\forall i \in \{j \mid S(j+n-n') = \mathbf{Int_L}\}. (\mathbf{Int_H} \le S'(i)).$$

すると，

$$(\uparrow_{F'}(f), \uparrow^{S'}_{S}(s)) \sim_{(F',S')} (\uparrow_{F'}(f'), \uparrow^{S'}_{S}(s'))$$

が成り立つ．

証明　$\uparrow_{F'}(f)(x)$ と $\uparrow_{F'}(f')(x)$ が異なるのは，$x \in dom(F')\backslash(dom(f) \cap dom(f')) \subseteq dom(F')\backslash dom(F)$，または $f(x) \ne f(x')$ かつ $\mathbf{Int_H} \le F(x)$ のときのみである．いずれの場合も F' の条件より $\mathbf{Int_H} \le F'(x)$ が成り立つので，$\uparrow_{F'}(f)(x) \sim_{F'(x)} \uparrow_{F'}(f')(x)$ が得られる．したがって，$\uparrow_{F'}(f) \sim_{F'} \uparrow_{F'}(f')$ が成り立つ．同様に $\uparrow^{S'}_{S}(s) \sim_{S'} \uparrow^{S'}_{S}(s')$ も成り立つ．□

証明（補題 **3.2**）　m に関する数学的帰納法により，(i) ~ (iv) が成り立つならば (v) および $dom(f_m) \supseteq dom(\mathcal{F}(pc_m))$，$dom(s_m) \supseteq dom(\mathcal{S}(pc_m))$ が成り立つことを示す．

$m=0$ のときは，$\uparrow_{\mathcal{F}(pc_0)}(f_0) = f_0$ および $\uparrow^{\mathcal{S}(pc_m)}_{\mathcal{S}(pc_0)}(s_0) = s_0$ より明らか．

$m \ge 1$ で，(i) ~ (iv) が成り立つと仮定する．帰納法の仮定より，

$$(\uparrow_{\mathcal{F}(pc_{m-1})}(f'_0), \uparrow^{\mathcal{S}(pc_{m-1})}_{\mathcal{S}(pc_0)}(s'_0)) \sim_{(\mathcal{F}(pc_{m-1}),\mathcal{S}(pc_{m-1}))} (f_{m-1}, s_{m-1})$$
$$dom(f_{m-1}) \supseteq dom(\mathcal{F}(pc_{m-1}))$$
$$dom(s_{m-1}) \supseteq dom(\mathcal{S}(pc_{m-1}))$$

が成り立つ．2 番目と 3 番目の条件および型付け規則，操作的意味論より明らかに

$$dom(f_m) \supseteq dom(\mathcal{F}(pc_m))$$
$$dom(s_m) \supseteq dom(\mathcal{S}(pc_m))$$

が成り立つ．

1 番目の条件および補題 B.4より，

$$\begin{aligned}&(\uparrow_{\mathcal{F}(pc_m)}(\uparrow_{\mathcal{F}(pc_{m-1})}(f'_0)), \uparrow^{\mathcal{S}(pc_m)}_{\mathcal{S}(pc_{m-1})}(\uparrow^{\mathcal{S}(pc_{m-1})}_{\mathcal{S}(pc_0)}(s'_0))) \\ &\sim_{(\mathcal{F}(pc_m),\mathcal{S}(pc_m))} (\uparrow_{\mathcal{F}(pc_m)}(f_{m-1}), \uparrow^{\mathcal{S}(pc_m)}_{\mathcal{S}(pc_{m-1})}(s_{m-1}))\end{aligned}$$

が成り立つ（$\mathcal{F}(pc_m), \mathcal{S}(pc_m)$ が補題 B.4の条件を満たすことは型付け規則より明らか）．

また，補題 B.3より，

$$(\uparrow_{\mathcal{F}(pc_m)}(f_{m-1}), \uparrow^{\mathcal{S}(pc_m)}_{\mathcal{S}(pc_{m-1})}(s_{m-1})) \sim_{(\mathcal{F}(pc_m),\mathcal{S}(pc_m))} (f_m, s_m)$$

が成り立つ．従って，条件 (v) を示すには，

$$\begin{aligned}&(\uparrow_{\mathcal{F}(pc_m)}(\uparrow_{\mathcal{F}(pc_{m-1})}(f'_0)), \uparrow^{\mathcal{S}(pc_m)}_{\mathcal{S}(pc_{m-1})}(\uparrow^{\mathcal{S}(pc_{m-1})}_{\mathcal{S}(pc_0)}(s'_0))) \\ &\sim_{(\mathcal{F}(pc_m),\mathcal{S}(pc_m))} (\uparrow_{\mathcal{F}(pc_m)}(f'_0), \uparrow^{\mathcal{S}(pc_m)}_{\mathcal{S}(pc_0)}(s'_0))\end{aligned}$$

が成り立つことを示せば十分である．

ここで，$\uparrow_{\mathcal{F}(pc_m)}(\uparrow_{\mathcal{F}(pc_{m-1})}(f'_0))(x)$ と $\uparrow_{\mathcal{F}(pc_m)}(f'_0)(x)$ とが異なるのは，$\uparrow_{\mathcal{F}(pc_{m-1})}(f'_0)(x) = \star$ であるような x についてのみである．

$$\uparrow_{\mathcal{F}(pc_{m-1})}(f'_0) \sim_{\mathcal{F}(pc_{m-1})} f_{m-1}$$

より，そのような x について $\mathbf{Int_H} \le \mathcal{F}(pc_{m-1})(x)$ が成り立つ．型付け規則より，$\mathbf{Int_H} \le \mathcal{F}(pc_m)(x)$ が成り立つので，

$$\uparrow_{\mathcal{F}(pc_m)}(\uparrow_{\mathcal{F}(pc_{m-1})}(f'_0))(x) \sim_{\mathcal{F}(pc_m)(x)} \uparrow_{\mathcal{F}(pc_m)}(f'_0)(x)$$

が得られる．従って，

$$\uparrow_{\mathcal{F}(pc_m)}(\uparrow_{\mathcal{F}(pc_{m-1})}(f'_0)) \sim_{\mathcal{F}(pc_m)} \uparrow_{\mathcal{F}(pc_m)}(f'_0)$$

が成り立つ．同様に

$$\uparrow^{\mathcal{S}(pc_m)}_{\mathcal{S}(pc_{m-1})}(\uparrow^{\mathcal{S}(pc_{m-1})}_{\mathcal{S}(pc_0)}(s'_0))) \sim_{\mathcal{S}(pc_m)} \uparrow^{\mathcal{S}(pc_m)}_{\mathcal{S}(pc_0)}(s'_0)$$

も成り立つ．よって十分条件が成り立つ．□