Rust MIR に対する情報流解析の型システムの検討

Towards Type-based Information Flow Analysis for Rust MIR

桑原 寛明 *

Summary. This paper proposes a type system for information flow analysis of Rust MIR programs. Since Rust MIR syntax is much simpler than Rust surface syntax, we expect that focusing on MIR makes it possible to clearly formalize ownership and references of Rust in the context of information flow analysis. In this paper, we formalize type-based information flow analysis for Rust MIR including references, but leave an extension for ownership as future work.

1 はじめに

Rust 言語は，Mozilla で Web ブラウザエンジンを実装するために開発が始まったプログラミング言語であり，C/C++言語が主に利用されているシステムプログラミングに適している．メモリ安全性を静的に保証するための言語機能として所有権システムを備えており，二重解放やダングリングポインタの検出，マルチスレッドプログラムにおけるデータ競合の検出，リソースの自動解放などが実現されている．

メモリ安全性の保証はもちろん重要であるが，その他にもプログラムが満たすべき（満たしてほしい）性質が存在する．その一つは，プログラムが処理しているデータを不必要に漏洩させないことであり，その検査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]．これはプログラムが機密データを外部に漏洩させる可能性がないことを静的に検査する手法であり，データの機密度を型として利用し，型付け可能なプログラムが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，機密度の低いデータが機密度の高いデータに直接的および間接的に依存しないことを表し，機密データ自体に加え機密データを推測できる情報も漏洩させないという意味でよい性質である．

Rust プログラムに対する情報流解析が [4] で提案されている．Rust には様々な構文要素があることから Rust のサブセット言語を対象としているが，情報流解析と所有権システムを同時に形式化するため複雑な型システムとなっている．Rust プログラムでは，所有権システムに伴う所有権の借用のために参照を多用する傾向があるが， [4] では参照の意味の定義にバグが残っている．

本稿では，Rust プログラムのコンパイル過程で利用される中間表現である Rust MIR を対象とする型検査に基づく情報流解析を提案する．Rust MIR では，プログラムを制御フローグラフとして表現する．制御フローグラフのノードは基本ブロックであり，基本ブロックは代入文，関数呼び出し，基本ブロック間のジャンプから構成される．構文に着目すると通常の Rust プログラムと比較して非常に簡潔であるため，情報流解析と所有権システムを簡潔に形式化できることが期待される．本稿では，Rust MIR を情報流解析の形式化に必要な要素のみに絞り込んで構文と意味，および情報流解析のための型システムを定義する．なお，所有権システムへの対応は今後の課題として残されている．

2 Rust MIR

2.1 Rust プログラムの中間表現

Rust プログラムのコンパイル過程では，ソースコードから HIR(High-level IR)，MIR(Mid-level IR)，LLVM IR の順に複数の中間表現を経由し，最終的に LLVM を

*Hiroaki Kuwabara, 南山大学理工学部

$$
\begin{array}{rcl}
\eta & ::= & L \mid H \\
\tau & ::= & \eta \mid \&^{\eta}\tau \\
\mathit{MIR} & ::= & F+ \\
F & ::= & \mathsf{fn}\ f(\overline{\_n : \tau}) : \tau\ \{D\ \ \overline{\beta : B}\} \\
D & ::= & \mathsf{vardecl} : \{\overline{V};\ \mathsf{goto}\ \texttt{->}\ \mathsf{bb0};\} \\
V & ::= & \mathsf{let}\ [\mathsf{mut}]\ \_n : \tau \\
\beta & ::= & \mathsf{bb}n \\
B & ::= & \{\overline{S};\ T;\}
\end{array}
\qquad
\begin{array}{rcl}
S & ::= & P = R \\
T & ::= & \mathsf{goto}\ \texttt{->}\ \beta \\
 & \mid & \mathsf{switchInt}(O)\ \texttt{->}\ [\overline{z_i : \beta_i}, \mathsf{otherwise} : \beta] \\
 & \mid & P = f(\overline{O})\ \texttt{->}\ \beta \\
 & \mid & \mathsf{return} \\
P & ::= & \_n \mid *\_n \\
O & ::= & \mathsf{move}\ P \mid \mathsf{copy}\ P \mid \mathsf{const}\ z \\
R & ::= & O \mid \mathit{bop}(O_1, O_2) \mid \mathit{uop}(O) \mid \&^{\eta}[\mathsf{mut}]P
\end{array}
$$

図 1　MIR プログラムの構文

用いてコード生成が行われる．HIR は，構文糖衣の展開などが行われた抽象構文木相当の中間表現である．MIR は，基本ブロックをノードとする制御フローグラフ相当の中間表現である [5]．基本ブロックの中に分岐はなく，入れ子の式は一時変数を用いて展開される．MIR を用いて借用の検査や最適化が行われる．

2.2　構文

MIR の構文を図 1 に示す．MIR プログラムを対象とする情報流解析の形式化のため，Rust MIR のサブセットに型としての機密度の記述を加えている．

機密度定数 η は L と H の 2 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．τ は型としての機密度である．$\&^{\eta}\tau$ は参照の型であり，η は参照自体の機密度，τ は参照先の機密度を表す．$\sqsubseteq$ を次のように τ に拡張する．すなわち，$\&^{\eta_1}\tau_1 \sqsubseteq \&^{\eta_2}\tau_2$ **iff** $\eta_1 \sqsubseteq \eta_2 \wedge (\tau_1 = \tau_2 \vee (\tau_1 = \&^{\eta_1'}\tau_1' \wedge \tau_2 = \&^{\eta_2'}\tau_2' \wedge \tau_1 \sqsubseteq \tau_2))$ とする．MIR プログラムは関数宣言 F の並びである．関数宣言中の $\_n$ は仮引数名であり，ここで n は 0 以上の自然数である．MIR プログラム中に出現する変数は $\_0, \_1, \ldots$ であり，$\_0$ は返り値のために予約されている．関数の本体は，関数内に出現するローカル変数の宣言 D と，ラベル付き基本ブロック B のリスト $\overline{\beta : B}$ からなる．D は vardecl でラベル付けされたブロックで，0 個以上の変数を宣言してラベルが bb0 の基本ブロックにジャンプする．変数宣言では，変数 $\_n$ の機密度が τ であることを宣言する．基本ブロックは bb$n$ でラベル付けされており，0 個以上の代入文 S と次に実行する基本ブロックへのジャンプ文 T を含む．代入の左辺 P は変数 $\_n$ か参照解決 $*\_n$ であり，右辺 R は変数，参照解決，定数 z，2 項演算 bop，単項演算 uop，参照生成である．z は整数であり，プログラムが扱うデータを整数型に制限する．move P は値とともに所有権が移動すること，copy P は値を複製して所有権は移動しないことを表す．ジャンプ文 T について，goto は無条件ジャンプ，switchInt は O の値に基づく条件分岐，$P = f(\overline{O})$ -> β は関数呼び出しとジャンプ，return は関数からのリターンを表す．関数呼び出しの正常終了と異常終了とで分岐先が異なる可能性があるが，ここでは簡単のために異常終了の場合を考慮しない．

実用に際して，プログラム中の機密度は開発者が与えるが，開発者は MIR プログラムを直接編集しない．Rust プログラムに注釈やコメントとして機密度を記述し，それを MIR プログラムに移植するなどの方法を検討する必要がある．

2.3　意味

図 1 の構文に従う MIR プログラムの意味を定義する．変数環境 $\mathcal{E}$ を変数名からロケーションへの関数，メモリ $\mathcal{M}$ をロケーションから値への関数とする．変数環境は関数呼び出しごとに独立である．MIR プログラムには定数と演算に加えて参照生成があるため，値は整数またはロケーションである．

図 2 に式 P, O, R を評価する関数 eval の定義を示す．本稿では所有権システムに対応しないため，move P と copy P の意味は同一である．bop，uop はそれぞれ 2 項演算 bop と単項演算 uop を計算する関数である．図 3 に変数宣言ブロック，基本ブロック，代入文，ジャンプ文の意味の定義を示す．fresh は新しいロケーション

$$eval(P,\mathcal{E},\mathcal{M}) = \begin{cases} \mathcal{M}(\mathcal{E}(\_n)) & \text{if } P = \_n \\ \mathcal{M}(\mathcal{M}(\mathcal{E}(\_n))) & \text{if } P = *\_n \end{cases}$$

$$eval(O,\mathcal{E},\mathcal{M}) = \begin{cases} z & \text{if } O = \mathsf{const}\ z \\ eval(P,\mathcal{E},\mathcal{M}) & \text{otherwise} \end{cases}$$

$$eval(R,\mathcal{E},\mathcal{M}) = \begin{cases} eval(O,\mathcal{E},\mathcal{M}) & \text{if } R = O \\ \mathsf{bop}(eval(O_1,\mathcal{E},\mathcal{M}), eval(O_2,\mathcal{E},\mathcal{M})) & \text{if } R = bop(O_1,O_2) \\ \mathsf{uop}(eval(O,\mathcal{E},\mathcal{M})) & \text{if } R = uop(O) \\ \mathcal{E}(\_n) & \text{if } R = \&^{\eta}[\mathsf{mut}]\_n \\ \mathcal{M}(\mathcal{E}(\_n)) & \text{if } R = \&^{\eta}[\mathsf{mut}]*\_n \end{cases}$$

図 2　関数 $eval$ の定義

$$\frac{\mathcal{M}' = \mathcal{M}[\mathcal{E}(\_n) \mapsto eval(R,\mathcal{E},\mathcal{M})]}{\langle \_n = R,\mathcal{E},\mathcal{M}\rangle \to_{\mathsf{S}} \mathcal{M}'} \qquad \frac{\mathcal{M}' = \mathcal{M}[\mathcal{M}(\mathcal{E}(\_n)) \mapsto eval(R,\mathcal{E},\mathcal{M})]}{\langle *\_n = R,\mathcal{E},\mathcal{M}\rangle \to_{\mathsf{S}} \mathcal{M}'}$$

$$\frac{\alpha_i \text{ fresh} \quad i \in \{1,\ldots,l\} \quad \mathcal{E}' = \mathcal{E}[\ldots,\_i \mapsto \alpha_i,\ldots] \quad \mathcal{M}' = \mathcal{M}[\ldots,\alpha_i \mapsto \bot,\ldots]}{\begin{array}{c}\langle\{\overline{\mathsf{let}\ [\mathsf{mut}]\ \_n};\ \mathsf{goto}\ \texttt{->}\ \mathsf{bb0};\},(\beta,P,\mathcal{E},f)::cs,\mathcal{M}\rangle \\ \to_{\mathsf{B}} \langle\{\mathsf{goto}\ \texttt{->}\ \mathsf{bb0};\},(\beta,P,\mathcal{E}',f)::cs,\mathcal{M}'\rangle\end{array}}$$

$$\frac{\langle P = R,\mathcal{E},\mathcal{M}\rangle \to_{\mathsf{S}} \mathcal{M}'}{\langle\{P = R;\overline{S};\ T;\},(\beta,P,\mathcal{E},f)::cs,\mathcal{M}\rangle \to_{\mathsf{B}} \langle\{\overline{S};\ T;\},(\beta,P,\mathcal{E},f)::cs,\mathcal{M}'\rangle}$$

$$\frac{}{\langle\{\mathsf{goto}\ \texttt{->}\ \beta';\},(\beta,P,\mathcal{E},f)::cs,\mathcal{M}\rangle \to_{\mathsf{B}} \langle block_f(\beta'),(\beta,P,\mathcal{E},f)::cs,\mathcal{M}\rangle}$$

$$\frac{\exists i.z_i == eval(O,\mathcal{E},\mathcal{M})}{\begin{array}{c}\langle\mathsf{switchInt}(O)\ \texttt{->}\ [\overline{z_i : \beta_i},\mathsf{otherwise} : \beta'];,(\beta,P,\mathcal{E},f)::cs,\mathcal{M}\rangle \\ \to_{\mathsf{B}} \langle block_f(\beta_i),(\beta,P,\mathcal{E},f)::cs,\mathcal{M}\rangle\end{array}}$$

$$\frac{\forall i.z_i \neq eval(O,\mathcal{E},\mathcal{M})}{\begin{array}{c}\langle\mathsf{switchInt}(O)\ \texttt{->}\ [\overline{z_i : \beta_i},\mathsf{otherwise} : \beta'];,(\beta,P,\mathcal{E},f)::cs,\mathcal{M}\rangle \\ \to_{\mathsf{B}} \langle block_f(\beta'),(\beta,P,\mathcal{E},f)::cs,\mathcal{M}\rangle\end{array}}$$

$$\frac{\alpha_i \text{ fresh} \quad \mathcal{E}' = [\ldots,\_i \mapsto \alpha_i,\ldots] \quad \mathcal{M}' = \mathcal{M}[\ldots,\alpha_i \mapsto eval(O_i,\mathcal{E},\mathcal{M}),\ldots] \quad i \in \{1,\ldots,l\}}{\begin{array}{c}\langle\{P' = f(O_1,\ldots,O_l)\ \texttt{->}\ \beta';\},(\beta,P,\mathcal{E},e)::cs,\mathcal{M}\rangle \\ \to_{\mathsf{B}} \langle block_f(\mathsf{vardecl}),(\beta',P',\mathcal{E}',f)::(\beta,P,\mathcal{E},e)::cs,\mathcal{M}'\rangle\end{array}}$$

$$\frac{\mathcal{M}' = \mathcal{M}[\alpha \mapsto eval(\_0,\mathcal{E}',\mathcal{M})] \quad \alpha = \begin{cases} \mathcal{E}(P') & \text{if } P = \_n \\ \mathcal{M}(\mathcal{E}(P')) & \text{if } P = *\_n \end{cases}}{\langle\{\mathsf{return};\},(\beta',P',\mathcal{E}',g)::(\beta,P,\mathcal{E},f)::cs,\mathcal{M}\rangle \to_{\mathsf{B}} \langle block_f(\beta'),(\beta,P,\mathcal{E},f)::cs,\mathcal{M}'\rangle}$$

図 3　MIR プログラムの意味

を表す．$\mathcal{E}[\ldots,\_i \mapsto \alpha_i,\ldots]$ は $\mathcal{E}$ について $\mathcal{E}(\_i)$ の値を α_i に更新することを表し，$\mathcal{M}[\ldots]$ も同様である．$\bot$ は値が未割り当てであることを表す．$block_f(\beta)$ は関数 f 内でラベルが β のブロックを表す．$\langle S,\mathcal{E},\mathcal{M}\rangle \to_{\mathsf{S}} \mathcal{M}'$ は代入文 S の意味であり，変数環境 $\mathcal{E}$ およびメモリ $\mathcal{M}$ の下で S を実行するとメモリが $\mathcal{M}'$ に変化することを表す．$\langle B,cs,\mathcal{M}\rangle \to_{\mathsf{B}} \langle B',cs',\mathcal{M}'\rangle$ は基本ブロック B の意味であり，コールスタック cs とメモリ $\mathcal{M}$ の下で B を 1 ステップ実行すると，基本ブロック，コールスタック，メモリがそれぞれ B'，cs'，$\mathcal{M}'$ に変化することを表す．コールスタックは関数呼び

出し順を追跡するためのデータ構造であり，関数から返る先の基本ブロックのラベル β，関数の返り値の代入先 P，仮引数とローカル変数の値を記録する変数環境 $\mathcal{E}$，実行中の関数の名前 f の 4 項組 $(\beta, P, \mathcal{E}, f)$ がスタックフレームである．

各ジャンプ文の意味は以下の通りである．goto -> β ではラベル β の基本ブロックに制御を移す．switchInt(O) -> $[\overline{z_i : \beta_i}, \mathsf{otherwise} : \beta']$ では O の値がいずれかの z_i と等しければラベル β_i，すべての z_i と等しくなければラベル β' の基本ブロックに制御を移す．$P' = f(\overline{O})$ -> β' では，関数 f の各仮引数に対応する新しいロケーション α_i を用いて，仮引数（とローカル変数）を格納する新しい変数環境 $\mathcal{E}'$ と実引数の値を格納したメモリ $\mathcal{M}'$ を準備し，呼び出す関数のためのスタックフレームをコールスタックに積んでから f のローカル変数を宣言するブロックに制御を移す．return では関数の返り値を指定された P' に代入し，コールスタックの先頭のスタックフレームを破棄して，返り先であるラベル β' の基本ブロックに制御を移す．

3　情報流解析のための型システム

情報流解析では，代入に伴う直接的な情報流 (explicit flow) と，条件分岐に伴う間接的な情報流 (implicit flow) を検査する．代入では，代入先の値から代入元の値がわかるため，代入元から代入先への情報流が存在し，代入元の機密度が代入先の機密度以下であればよい．関数呼び出しにおける引数の受け渡しも実引数から仮引数への代入とみなす．条件分岐では，分岐先において代入される変数の値を追跡することで分岐先，すなわち分岐の条件の値が判明する可能性がある．つまり，分岐の条件から分岐先で代入される変数への情報流が存在し，分岐の条件の機密度が分岐先で代入される変数の機密度以下であればよい．

Rust などの高水準言語では条件分岐は構造化されており，条件分岐に伴う情報流の到達範囲は構文木から特定できるため，条件分岐に対する型付け規則の制約は素直に記述できる．一方，MIR プログラムの場合，構文木としては基本ブロックがフラットに並んでいるだけで構造化されていない．条件分岐に伴う情報流の到達範囲の特定には制御フローグラフの解析が必要である．以上から，MIR プログラムに対する情報流解析のための型システムにおける型付け規則を図 4 のように定義する．

FDEC 規則が関数宣言，FBODY 規則が関数本体，BLOCK 規則が基本ブロック，ASSIGN 規則が代入文，GOTO 規則，RET 規則，SWITCH 規則，CALL 規則がジャンプ文，その他が式に対する規則である．$\eta^\dagger = \eta$，$(\&^\eta \tau)^\dagger = \eta$ であり，*ftype* は関数のシグネチャを取得する関数である．Γ は型環境であり，変数名からその機密度への関数である．関数に含まれるすべての基本ブロックが型付け可能であれば関数本体と関数宣言が型付け可能である．基本ブロック，代入文，ジャンプ文の型判定式 $\Gamma \vdash A : \tau$ は，型環境 Γ の下で A の実行に伴って代入される変数の機密度が τ 以上であることを表す．式の型判定式 $\Gamma \vdash e : \tau$ は，型環境 Γ の下で e の機密度が τ 以下であることを表す．

ASSIGN 規則では，代入元の機密度が代入先の機密度以下であることが制約である．関数呼び出しに対する CALL 規則では，実引数の機密度が仮引数の機密度以下であることと，返り値の機密度が返り値の代入先の機密度以下であることが制約である．これらの規則では，代入先の機密度を文の機密度とする．GOTO 規則，RET 規則，SWITCH 規則では，対象の文で代入が発生しないため文の機密度を H とする．

SWITCH 規則では，分岐の条件 O の機密度と分岐後に発生する代入における代入先の機密度を比較するために，O に関する情報流が到達する範囲を特定する必要がある．到達範囲は分岐先の基本ブロックに留まらず，それ以降の基本ブロックまで及ぶ可能性がある．例えば，ラベル bb1 の基本ブロックで bb2 と bb3 に分岐し，一方は bb2，bb4，bb6，bb7 の順に，他方は bb3，bb5，bb6，bb7 の順に基本ブロックを実行するとする．この場合 bb6 以降は双方で実行されるため区別できず，その手前の bb2，bb4，bb3，bb5 の基本ブロックが O に関する情報流の到達範囲となる．

$$\frac{\_0 : \tau, \overline{\_n : \tau} \vdash \{D\ \ \overline{\beta : B}\}}{\vdash \mathsf{fn}\ f(\overline{\_n : \tau}) : \tau\ \{D\ \ \overline{\beta : B}\}}\ [\mathrm{FDEC}]$$

$$\frac{\Gamma, \overline{\_n : \tau} \vdash B_i : \eta_i \quad i \in \{1, \ldots, l\}}{\Gamma \vdash \{\mathsf{vardecl} : \{\overline{\mathsf{let}\ [\mathsf{mut}]\ \_n : \tau;}\ \mathsf{goto}\ \texttt{->}\ \mathsf{bb0};\}\ \beta_1 : B_1\ \ldots\ \beta_l : B_l\}}\ [\mathrm{FBODY}]$$

$$\frac{\Gamma \vdash S_i : \eta_i \quad \Gamma \vdash T : \eta_t \quad i \in \{1, \ldots, l\}}{\Gamma \vdash \{S_1;\ \ldots;\ S_l;\ T;\} : \bigsqcap_i \eta_i \sqcap \eta_t}\ [\mathrm{BLOCK]} \qquad \frac{}{\Gamma \vdash \mathsf{goto}\ \texttt{->}\ \beta : H}\ [\mathrm{GOTO}]$$

$$\frac{\Gamma \vdash P : \tau_p \quad \Gamma \vdash R : \tau_r \quad \tau_r \sqsubseteq \tau_p}{\Gamma \vdash P = R : {\tau_p}^{\dagger}}\ [\mathrm{ASSIGN}] \qquad \frac{}{\Gamma \vdash \mathsf{return} : H}\ [\mathrm{RET}]$$

$$\frac{\Gamma \vdash O : \eta_o \quad \Gamma \vdash \mathit{block}_f(\beta_i) : \tau_i \quad \eta_o \sqsubseteq \tau_i \quad \beta_i \in \mathit{depend}(\overline{\beta}, \beta)}{\Gamma \vdash \mathsf{switchInt}(O)\ \texttt{->}\ [\overline{z : \beta}, \mathsf{otherwise} : \beta] : H}\ [\mathrm{SWITCH}]$$

$$\frac{\begin{array}{l}\tau_1, \ldots, \tau_l \to \tau_r = \mathit{ftype}(f) \quad \Gamma \vdash P : \tau_p \quad \tau_r \sqsubseteq \tau_p \\ \Gamma \vdash O_i : \tau_{o_i} \quad \tau_{o_i} \sqsubseteq \tau_i \quad i \in \{1, \ldots, l\}\end{array}}{\Gamma \vdash P = f(O_1, \ldots, O_l)\ \texttt{->}\ \beta : {\tau_p}^{\dagger}}\ [\mathrm{CALL}]$$

$$\frac{}{\Gamma \vdash \_n : \Gamma(\_n)}\ [\mathrm{VAR}] \qquad \frac{\Gamma \vdash \_n : \&^{\eta}\tau}{\Gamma \vdash *\_n : \tau}\ [\mathrm{DEREF}] \qquad \frac{\Gamma \vdash P : \tau}{\Gamma \vdash \&^{\eta}[\mathsf{mut}]P : \&^{\eta}\tau}\ [\mathrm{REF}]$$

$$\frac{\Gamma \vdash P : \tau}{\Gamma \vdash \mathsf{move}\ P : \tau}\ [\mathrm{MOVE}] \qquad \frac{\Gamma \vdash P : \tau}{\Gamma \vdash \mathsf{copy}\ P : \tau}\ [\mathrm{COPY}] \qquad \frac{}{\Gamma \vdash \mathsf{const}\ P : L}\ [\mathrm{CONST}]$$

$$\frac{\Gamma \vdash O_1 : \eta_1 \quad \Gamma \vdash O_2 : \eta_2}{\Gamma \vdash \mathit{bop}(O_1, O_2) : \eta_1 \sqcup \eta_2}\ [\mathrm{BOP}] \qquad \frac{\Gamma \vdash O : \eta}{\Gamma \vdash \mathit{uop}(O) : \eta}\ [\mathrm{UOP}]$$

図 4　型付け規則

SWITCH 規則における *depend* は，分岐の条件の情報流が到達する範囲の基本ブロック集合を求める関数である．

depend の定義を以下に示す．着目している関数 f の基本ブロックをノードとする制御フローグラフを G とし，G のノードを基本ブロックのラベル β により表す．ラベルが β の基本ブロック $\{\overline{S};\ T;\}$ に対し，

$$\mathit{next}(\beta) = \begin{cases} \{\beta'\} & \text{if } T = \mathsf{goto}\ \texttt{->}\ \beta' \text{ or } T = P = f(\overline{O})\ \texttt{->}\ \beta' \\ \{\overline{\beta}, \beta'\} & \text{if } T = \mathsf{switchInt}(O)\ \texttt{->}\ [\overline{z : \beta}, \mathsf{otherwise} : \beta'] \\ \emptyset & \text{if } T = \mathsf{return} \end{cases}$$

とする．β を始点とする有限の実行経路 $p = \beta_0\beta_1 \ldots \beta_l$ は，$\beta_0 = \beta$，$\mathit{block}_f(\beta_l) = \{\overline{S};\ \mathsf{return};\}$，$\forall i \in \{0, \ldots, l-1\}.\beta_{i+1} \in \mathit{next}(\beta_i)$ を満たす．実行経路 p の i 番目のラベルを $p(i)$，$\overline{\beta}$ に含まれる各ラベルを始点とする有限の実行経路の集合を $\mathit{paths}(\overline{\beta})$ と書く．$\overline{p}$ に含まれるすべての実行経路に出現するラベルを $\overline{p}$ の合流点と呼び，$\mathit{paths}(\overline{\beta})$ の合流点の集合を $\mathit{merge}(\overline{\beta}) = \{\beta' \mid \forall p \in \mathit{paths}(\overline{\beta}).\exists i.p(i) = \beta'\}$ と定義し，$\mathit{paths}(\overline{\beta})$ の最初の合流点の集合を $\mathit{fmerge}(\overline{\beta}) = \{\beta' \mid \forall p \in \mathit{paths}(\overline{\beta}).\exists i.p(i) = \beta' \wedge \forall j < i.p(j) \notin \mathit{merge}(\overline{\beta})\}$ と定義する．以上より，

$$\mathit{depend}(\overline{\beta}) = \{\beta' \mid \exists p \in \mathit{paths}(\overline{\beta}).\exists i.p(i) \in \mathit{fmerge}(\overline{\beta}) \Rightarrow \forall j < i.p(j) = \beta'\}$$

と定義する．直観的には，分岐したすべての実行経路が初めて合流する基本ブロッ

```
fn f(b : &bool) -> i32 {
  let i;
  if *b {
    i = 1;
  } else {
    i = 0;
  }
  i
}
```

図 5　Rust プログラムの例

```
fn f(_1 : &H) : L {
  vardecl : {
    let mut _0 : L; let mut _2 : H;
    goto -> bb0; }
  bb0 : { _2 = *_1;
    switchInt(move _2) ->
      [0 : bb2, otherwise : bb1]; }
  bb1 : { _0 = const 1; goto -> bb3; }
  bb2 : { _0 = const 0; goto -> bb3; }
  bb3 : { return; }
}
```

図 6　図 5 に対応する MIR プログラム

クが最初の合流点であり，各実行経路について分岐先から最初の合流点の直前までのすべての基本ブロックの集合が到達範囲である．

4　例

例として図 5 の Rust プログラムに対応する図 6 の MIR プログラムを考える．紙幅の都合で詳細は省略するが，f の呼び出しを図 3 に従って実行すると適切な返り値が得られる．図 6 では，機密度が H の変数 _2 に基づいて分岐した先で機密度が L の変数 _0 に代入しており，不正な情報流が存在するため型付け不能である．switchInt 文に対して図 4 の SWITCH 規則を適用すると，$\mathit{depend}(\mathsf{bb1}, \mathsf{bb2}) = \{\mathsf{bb1}, \mathsf{bb2}\}$ であり，いずれのラベルの基本ブロックも BLOCK 規則から型が L である．一方，分岐の条件 move _2 の機密度は H であるため条件を満たさず，型付けできない．

5　おわりに

本稿では，Rust プログラムの中間表現である MIR を対象とする情報流解析のための型システムを提案した．MIR プログラムの意味定義は [6] を，SWITCH 規則の *depend* は [7] を参考にした． [6] は Rust MIR に対するテイント解析を， [7] は JVM を簡略化したアセンブリ言語向けの情報流解析のための型システムを提案している．

今後の課題として，非干渉性に対する健全性の証明，可変性 (mutability) への対応，ムーブセマンティクス（所有権の移動）への対応と情報流解析との関係の明確化，機密度の指定方法の検討，検査器の実装などが挙げられる．

謝辞　本研究の一部は 2022 年度南山大学パッヘ研究奨励金 I-A-2 の助成による．

参考文献

[1] 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.

[2] 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.

[3] 黒川翔, 桑原寛明, 山本晋一郎, 坂部俊樹, 酒井正彦, 草刈圭一朗, 西田直樹. 例外処理付きオブジェクト指向プログラムにおける情報流の安全性解析のための型システム. 電子情報通信学会論文誌 D, Vol. J91-D, No. 3, pp. 757–770, 2008.

[4] 長谷川健太, 桑原寛明, 國枝義敏. Rust プログラムの情報流解析のための型システム. 信学技報, Vol.119, No.452, SS2019-53, pp. 73–78, 2020.

[5] Guide to Rustc Development. `https://rustc-dev-guide.rust-lang.org/mir/index.html`. 2022/09/13 参照.

[6] Emil Jørgensen Njor and Hilmar Gústafsson. Static Taint Analysis in Rust: Using Rusts Ownership System for Precise Static Analysis. Master's thesis, Department of Computer Science, Aalborg University, 2021.

[7] 小林直樹, 白根慶太. 低レベル言語のための情報流解析の型システム. コンピュータソフトウェア, Vol. 20, No. 2, pp. 118–137, 2003.