JVMにおけるロック整合性検証のための新しい型システム

岩間 太[†]　　小林 直樹[††]
Futoshi iwama　　Naoki Kobayashi
[†] 東京大学　　[††] 東京工業大学

プログラミング言語 Java の仮想機械では，実行前にコードの検証を行い，不正なコードを排除している．しかし，現在の検証器では，並行スレッドによるロックの獲得・解放の整合性に関する検証は行っていない．この問題を解決するため，本研究では，Abadi と Stata によるバイトコード検証のため型システムを改良する．新しい型システムでは，各オブジェクトのロックが獲得・解放される順序の情報をオブジェクトの型に付加することにより，ロックの正しい使用を保証する．

1　導入

Java 言語 [1] のプログラムは，Java バイトコードと呼ばれる中間言語のコードにコンパイルされ，その後，Java 仮想機械 (JVM) [6] により解釈，実行される．この際，実行時に致命的なエラーが生じるのを防ぐため，実行の前に，バイトコードを静的にチェックを行うバイトコード検証器と呼ばれるものが定義されている [6]．しかし，現在の検証器は，並行プリミティブ（ロックプリミティブなど）に関する検証をほとんど行っていない．

このような問題に対して，Bigliardi と Laneve[2] は，ロックプリミティブの使用法を検証する型システムを提案した．この型システムは，`load` x, `monitorenter` 命令列と，`load` x, `monitorexit` 命令列が，入れ子構造で出現してクリティカルセクションを構成していること，各クリティカルセクションに対応する例外表の記述があること，などを調べることで検証を行っている．しかし，この型システムは，クリティカルセクションの交差，他のクリティカルセクションへのジャンプを禁止しており，また，例外表の記述も完全に各クリティカルセクションと対応していなければならないなど，制限が多い．さらに，型付け規則が複雑で，その正しさが自明でない．

上の問題に対し，本論文では，ロックプリミティブの使用法を検証するための，より制限が緩く，かつより簡明な型システムを提案する．我々の型システムの重要なアイデアは，各オブジェクトのロックが獲得・解放される順序に関する情報（*usage* と呼ぶ）をオブジェクトの型に付加することである．例えば，最初にロックが獲得され，次に，解放され，その後は獲得も解放もされないオブジェクトの型には $L.\widehat{L}.\mathbf{0}$ という *usage* を付加する．このような *usage* を付加した型を扱うことで，より簡明な型付け規則を与えることができ，また，ロックプリミティブの使用に関する制限を緩めることができる．これにより，Bigliardi らの型システム [2] と異なり，クリティカルセクションの交差やクリティカルセクションからの分岐がある場合においても，次の性質が成り立つことを保証できる．(1) あるメソッドの中で獲得されたロックは，そのメソッドが終了するまでに必ず解放される．(2) 獲得していないロックを解放しようとすることはない．

本論文の以下の構成は次の通り，2 節で対象言語を導入し，3 節において，型システムを定義し，その正当性を示す．4 節で関連研究に触れ，5 節で結論を述べる．ページ制限のため，厳密な定義や型システムの正しさの証明は割愛する．詳細については [3] を参照されたい．

2　対象言語 $JVML_T$

対象言語 $JVML_T$ の構文を Bigriardi と Laneve の枠組み [2] にならい定義する．

2.1　$JVML_T$ の構文

$JVML_T$ は，JAVA のバイトコード言語 JVML の部分言語となる．まず，アドレスの集合 $\mathcal{A}$，自然数の集合 $\mathcal{N}$，変数の集合 $\mathcal{V}$ とする．$\mathcal{V}$ の要素も自然数で表すが、$\mathcal{N}$ の要素とは区別する．

定義 1　命令の集合 **Inst** は以下によって定義される集合である．

$$
\begin{array}{rcl}
I & ::= & \texttt{inc} \mid \texttt{pop} \mid \texttt{push0} \mid \texttt{load}\ x \mid \texttt{store}\ x \\
& & \mid \texttt{if}\ j \mid \texttt{new}\ \tau \mid \texttt{start}\ \sigma \\
& & \mid \texttt{monitorenter}\ x \mid \texttt{monitorexit}\ x \\
& & \mid \texttt{athrow} \mid \texttt{return}
\end{array}
$$

ここで，x は $\mathcal{V}$ に，j は $\mathcal{A}$ に値域をとるものとする．また，σ は**クラス名**（有限長の文字列）を，τ は 型（3 章を参照）を表す．各命令の意味は次の通り．`inc` は，オペランドスタック (以下スタックという）のトップの値 (整数) を 1 つ増やす．`pop` は，スタックのトップから値を除く．`push0` は，スタックのトップに値 (整数) 0 をのせる．`load` x は，変数 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除き，その値をローカル変数 (以下変数という）x に格納する．`if` j は，スタックのトップの値が 0 (整数) なら，命令アドレス j に分岐，それ以外の整数なら，そのまま次の命令アドレスを実行する．`new` σ は，σ クラスのオブジェクトを新しく生成・初期化し，そのオブジェクトへの参照をスタックのトップにのせる．`start` σ は，スタックのトップにある σ クラスのオブジェクトへの参照があるとき，その参照が指すオブジェクトのメソッドを起動する．`monitorenter` x，`monitorexit` x は，変数 x が指すオブジェクトのロックを獲得，解放する命令である．`athrow` 命令は，強制的に例外を発生させる．`return` 命令は，メソッドの実行を終了する．

上の命令の中で，`new` 命令は，オブジェクトの初期化を失敗したときに例外を発生させるものとする．`new` 命令と `athrow` 命令以外は、例外を発生させないものとする．また，ここでは簡単のため，例外は一種類しかないものとする．

実際の Java のバイトコード [6] では，オブジェクトの生成および初期化は複数の命令列によって行われるがここ

では Abadi と Stata の型システムにならい，アトミックな命令 `new` を仮定する．また，実際のロックの獲得，解放命令は，引数をとらず，スタックのトップにあるオブジェクトのロックの獲得，解放を行うが，ここでは技術上の理由のため，ローカル変数上のオブジェクトのロックに対して獲得，解放がおこるものと仮定する．

メソッド本体とは，$\mathcal{A}$ の有限部分集合から **Inst** への関数であり，メタ変数 P を用いて表す．また，**例外表**とは，$\mathcal{A}$ の有限部分集合から $\mathcal{A}$ への関数であり，メタ変数 E を用いて表す．例外表は，命令アドレスに対し，そこで例外が発生したとき，次に実行すべき命令のアドレスを対応させるものである．メソッド本体 P と例外表 E の組 (P, E) を**メソッド**と呼ぶ．

クラス名 σ とメソッドの集合の組を**クラス定義**と呼び，クラス定義の集合を**プログラム**と呼ぶ．

例 2 以下はメソッド本体の例である．

```
1  monitorenter 0   ;; ローカル変数 0 のオブジェクト
                    ;; のロックを獲得
2  load 1           ;; ローカル変数 1 の値をロード
3  if 6             ;; 値が 0 なら 6 へ分岐
4  monitorexit 0    ;; ローカル変数 0 のオブジェクト
                    ;; のロックを解放
5  return           ;; メソッド終了
6  monitorexit 0    ;; ローカル変数 0 のオブジェクト
                    ;; のロックを解放
7  return           ;; メソッド終了
```

上のバイトコードでは，アドレス 1 で局所変数 0 に格納されているオブジェクト（メソッドが起動されたオブジェクト自身）のロックを獲得した後，アドレス 3 で分岐し，どちらの分岐でもロックの解放を行っているので，ロックプリミティブが正しく使用されている．実際，3 節で導入する我々の型システムでは上のコードを正しいコードとして受理する．一方，Bigliardi らの型システム [2] では `monitorenter` 0 と `monitorexit` 0 が構文上一対一に対応していないため，正しいコードとして受理されない．また，上のコードのアドレス 3 が，誤って `if` 7 と記述されていた場合には我々の型システムではロックの使用法が正しくないと判断される．

3 型システム

本節では，Abadi と Stata のバイトコード検証のための型システム [9] にならい，ロックプリミティブの使用法を検証するための型システムを与える．

3.1 Usage と 型

1 節で述べたように，通常のオブジェクトの型をロックの獲得，解放の順序に関する情報（*usage* と呼ぶ）を付加することにより拡張する．Usage の集合は次の構文で与えらる．

定義 3 (usages)

$$U ::= \mathbf{0} \mid \alpha \mid L.U \mid \widehat{L}.U \mid U_1 \otimes U_2 \mid U_1 \& U_2 \mid \mu\alpha.U$$

$\mathbf{0}$ は，ロックが獲得も解放もされないことを表す．α は，usage を表す変数であり，$\mu\alpha.$ により束縛される．$L.U$ は，獲得された後，U にしたがって使用されるロックの usage を表す．$\widehat{L}.U$ は，解放された後 U に従って使用されるロックの usage を表す．$U_1 \otimes U_2$ は，U_1 に従う獲得・解放と U_2 に従う獲得・解放がインターリーブして起るおこることを表す．$U_1 \& U_2$ は，U_1 または U_2 のいずれかにしたがって獲得・解放が起ることを表す．$\mu\alpha.U$ は再帰 usage で，$\mu\alpha.U = [\alpha \mapsto \mu\alpha.U]U$ の形に再帰的に展開して得られる usage に従って使用され得ることを表す．

以下では，$L.\mathbf{0}$，$\widehat{L}.\mathbf{0}$ をしばしば L, $\widehat{L}$ と略記する．Usage 中の有限個の部分式を穴 [] でおきかえたものを **Usage の文脈**と呼び，メタ変数 C で表す．Usage の文脈 C の穴を左から順に $U_1, \ldots, U_n$ で置換して得られる usage を $C[U_1, \ldots, U_n]$ で表す．例えば $C = [] \otimes []$ ならば，$C[U_1, U_2] = U_1 \otimes U_2$ である．

定義 4 Usage の 2 項関係 $\equiv$ は，$\otimes$ および & に関する交換率と結合律，および $U \otimes \mathbf{0} \equiv U$，$\mu\alpha.U \equiv [\alpha \mapsto \mu\alpha.U]U$ を満たす最小の合同関係である．

定義 5 部分 usage 関係 $\leq$ は以下の規則を満たす最小の前順序関係である

$$\frac{U_1 \equiv U_2}{U_1 \leq U_2} \qquad \frac{U_1 \& U_2}{U_1} \qquad \frac{U_i \leq U_i'}{C[U_1, \ldots, U_n] \leq C[U_1', \ldots, U_n']}$$

上の usage を用い，**型**の集合 **Types** を以下の構文によって定義する．

定義 6 (型)

$$\tau \quad ::= \quad \mathbf{Int} \mid \mathbf{Top} \mid \sigma/U$$

Int は整数の型，**Top** は全く使われない値の型を表す．σ/U は，U に従ってロックが獲得・解放されるようなクラス σ のオブジェクトの型である．

例 7 $\texttt{Counter}/L.\widehat{L}$ は，ロックが獲得された後，その後解放される，`Counter` クラスのオブジェクトの型を表す．$\texttt{Account}/L.(\widehat{L}\&\mathbf{0})$ は，最初にロックが獲得された後，解放されるか，または何もされずに終わる `Account` クラスのオブジェクトの型を表す．

上の例から分かるように，オブジェクトの型中の usage を見ればそのオブジェクトに対して正しくロックの獲得・解放が起るかどうかが分かる．例えば上の `Counter` クラスのオブジェクトの usage $L.\widehat{L}$ は正しいが，`Account` クラスのオブジェクトの usage $L.(\widehat{L}\&\mathbf{0})$ は正しくない（ロックを解放しないままで終わる可能性がある）．そこで，Usage U が正しいロックの使用法を表している時，すなわち U 中の $L.U_1$ の形の各部分式に対して対応する $\widehat{L}.U_2$ の形の部分式が必ず U 中に存在する時，U は**正当である**と言い，$rel(U)$ と書く．$rel(U)$ の厳密な定義は論文 [3] に譲る．

例 8 $L.L.(\widehat{L} \otimes \widehat{L})$，$(L.\widehat{L})\&(L.\widehat{L})$，$L.\mu\alpha.((\widehat{L}.L.\alpha)\&\widehat{L})$ は正当な usage である．一方，$L.(L \otimes \widehat{L})$，$L.\widehat{L}.\widehat{L}$ は正当でない．

定義 9 部分型関係 $\leq$ は以下の規則を満たす最小の前順序関係である．

$$\mathbf{Int} \leq \mathbf{Top} \qquad \frac{U \leq \mathbf{0}}{\sigma/U \leq \mathbf{Top}} \qquad \frac{U_1 \leq U_2}{\sigma/U_1 \leq \sigma/U_2}$$

型に関する述語 $rel_t(\tau)$ を以下の規則を満たす最小の関係として定義する．

$$\frac{}{rel_t(\mathbf{Int})} \qquad \frac{}{rel_t(\mathbf{Top})} \qquad \frac{rel(U)}{rel_t(\sigma/U)}$$

3.2 型環境

ローカル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，ローカル変数 x に格納されている値の型を表す．**スタックの型**は自然数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．ϵ は，任意の変数について **Top** を対応づける写像であり，空のスタックの型を表す．$\tau \cdot S$ 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ローカル変数の型への写像を**ローカル変数の型環境**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．$\mathcal{F}[i]$，$\mathcal{S}[i]$ はアドレス i を実行する際のローカル変数の型，スタックの型をそれぞれ表す．

記述を簡略にするため，以下のように usage の構成子を型やローカル変数の型に関する演算子に拡張する．

定義 10 $*$ は $\otimes$ または $\&$，$\dot{L}.$ は $L.$ または $\widehat{L}.$ を表すものとする．$\tau_1 * \tau_2$，$\dot{L}.\tau$ を以下によって定義する（下にあてはまらない場合は未定義）．

$$
\begin{array}{lcl}
\mathbf{Top} * \mathbf{Top} & = & \mathbf{Top} \\
\mathbf{Int} * \mathbf{Int} & = & \mathbf{Int} \\
(\sigma / U_1) * (\sigma / U_2) & = & \sigma / (U_1 * U_2) \\
\dot{L}.(\sigma / U) & = & \sigma / (\dot{L}.U)
\end{array}
$$

$F_1 * F_2$ は $dom(F_1) = dom(F_2)$（$dom(f)$ は関数 f の定義域を表す）の時のみ定義され，$dom(F_1 * F_2) = dom(F_1)$，$\forall x \in dom(F_1).(F_1 * F_2)(x) = (F_1(x)) * (F_2(x))$ によって定義される．$S_1 * S_2$ についても同様に定義する．

3.3 型付け規則

型環境 $\mathcal{F}, \mathcal{S}$ のもとでの，メソッド (P, E) の型付けを定める判定を定義する．

定義 11 (メソッドの各命令アドレスでの型判定)
$\mathcal{F}, \mathcal{S}, j \vdash (P, E)$ は，図 1 の規則を満たす最小の関係

この判定は，メソッド (P, E) が，命令アドレス j において，型環境 $\mathcal{F}, \mathcal{S}$ のもと，型付けされていることを述べる．

図 1 の中では，$\mathcal{F}[j], \mathcal{S}[j]$ を F_j, S_j と記す．

以下で，代表的な規則を説明する．

規則 (MEntr) は，`monitorenter x` 命令に対する規則．最初の行は，型付けしようとしているアドレスの命令が `monitorenter x` 命令であることを示している．次の行は，次の命令が存在する（`monitorenter x` 命令が最後の命令ではない）ことを述べている．3,4,5 行目は，アドレス j でのローカル変数 x のなかの値は，ロックされた後，アドレス $j+1$ でのローカル変数 x の値として使用され，アドレス j でのそれ以外の変数の値 と スタックの値は，アドレス $j+1$ と同じように使用されることを述べている．

規則 (If) は，`if j'` 命令に対する規則．最初の行は，型付けしようとしているアドレスの命令が `if j'` 命令であることを，次の行は，アドレス $j', j+1$ に命令があることを述べている．3,4 行目は，アドレス j でのローカル変数の値，トップ以外のスタックの値は，アドレス j' での値として使用されるか，もしくは，アドレス $j+1$ での値として使用されるということを述べている．また 4 行目では，アドレス j でのスタックのトップの値は，分岐判定のための整数値であることも述べている．

以上を用い，メソッドの型判定を以下で定義する．

定義 12 (メソッドの型判定)

$$
\frac{\begin{array}{c}\forall\, x \in Dom(F_1).\ rel_t(F_1(x)) \\ S_1 = \epsilon \\ \forall j \in Dom(P).\ \mathcal{F}, \mathcal{S}, j \vdash (P, E)\end{array}}{\mathcal{F}, \mathcal{S} \vdash (P, E)}
$$

この判定は，メソッド (P, E) が，型環境 $\mathcal{F}, \mathcal{S}$ のもと，型付けされていることを述べる．ここで，アドレスの 1 は，メソッド本体の最初の命令アドレス．1 行目は，すべてのオブジェクトのロックが正しく使われることを述べている．2 行目は，最初のアドレスでスタックが空であることを述べ，最後の行は，各アドレスで，メソッドが型づけされることを述べている．

型付けされたプログラムは以下の関係で $\mathcal{F}, \mathcal{S} \vdash Q$ 定義される．

定義 13 (型付けされたプログラム) Q の中のすべてのクラスで、クラスの中のすべてのメソッド (P, E) について，$\mathcal{F}, \mathcal{S} \vdash (P, E)$ が成り立つならば，関係 $\mathcal{F}, \mathcal{S} \vdash Q$ が成り立つ．

この関係は，型環境 $\mathcal{F}, \mathcal{S}$ のもとで，プログラム Q が型付けされていること示し，Q のすべてのクラスに対して，そのすべてのメソッドが型づけされれば，プログラム Q が型付けされる，ということを述べている．

3.4 型システムの健全性

上の型システムでプログラムが型付けされるのであれば，プログラム実行中にロックは必ず正しく使用されること，すなわち以下の性質が成り立つことが保証される．

すべてのメソッドの任意の実行状態において，

(1) 次の命令が `return` なら，このメソッドの中で獲得されたオブジェクトのロックはすべて解放されている．

(2) 次の命令が `monitorexit` x なら，その状態において，このメソッドの中で，x が指すオブジェクトのロックが 1 回以上獲得されている．

この性質は，通常の型システムの健全性の証明と同様，（1）実行の過程で型付けが保存されること，および（2）型付け可能なプログラムはすぐに誤ったロックの使い方をしないこと，から導くことができる．上の性質の厳密な記述およびその証明は論文 [3] に譲る．

3.5 型推論アルゴリズム

ロックプリミティブの使用法を検証するためには，型付け規則に基づいて型推論を行い，プログラムが型付けが可能かどうかを判定すればよい．各メソッドについての型推論は次のように行えばよい．

1. 図 1 の規則に基づき，バイトコードの各行から型や usage 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．

詳細は論文 [3] に譲り，以下では，例 2 のメソッドに対する型推論の概略を示す．簡単のため，usage を除いた通常の型情報はすでに得られているものとし，1 行目ではローカル変数の 0 にクラス σ のオブジェクト，1 に整数が格納されているものとする．従って，ローカル変数の型環境 F は $F_i(0) = \sigma/\alpha_i, F_i(1) = \mathbf{Int}$ という形で表すことができ

$$(INC)\quad \frac{P[j] = \texttt{inc} \quad j+1 \in Dom(P) \quad F_j \leq F_{j+1} \quad S_{j+1}(0) = S_j(0) \leq \mathbf{Int} \quad S_{j+1} = S_j}{\mathcal{F}, \mathcal{S}, j \vdash (P, E)}$$

$$(PUSH)\quad \frac{P[j] = \texttt{push0} \quad j+1 \in Dom(P) \quad F_j \leq F_{j+1} \quad S_{j+1} \leq \mathbf{Int} \cdot S_j}{\mathcal{F}, \mathcal{S}, j \vdash (P, E)}$$

$$(POP)\quad \frac{P[j] = \texttt{pop} \quad j+1 \in Dom(P) \quad F_j \leq F_{j+1} \quad S_j(0) \leq \mathbf{Top}}{\mathcal{F}, S, j \vdash (P, E)}$$

$$(IF)\quad \frac{P[j] = \texttt{if } j' \quad j', j+1 \in Dom(P) \quad F_j \leq F_{j+1} \& F_{j'} \quad S_j \leq \mathbf{Int} \cdot (S_{j+1} \& S_{j'})}{\mathcal{F}, \mathcal{S}, j \vdash (P; E)}$$

$$(LOAD)\quad \frac{P[j] = \texttt{load } x \quad j+1 \in Dom(P) \quad x \in Dom(F_j) \quad F_j(y) \leq F_{j+1}(y) \ \ (y \neq x) \quad F_j(x) \leq F_{j+1}(x) \otimes S_{j+1}(0) \quad S_{j+1} \leq S_{j+1}(0) \cdot S_j}{\mathcal{F}, \mathcal{S}, j \vdash (P, E)}$$

$$(STORE)\quad \frac{P[j] = \texttt{store } x \quad j+1 \in Dom(P) \quad x \in Dom(F_j) \quad F_j(y) \leq F_{j+1}(y) \ \ (y \neq x) \quad F_j(x) \leq \mathbf{Top} \quad S_j \leq F_{j+1}(x) \cdot S_{j+1}}{\mathcal{F}, \mathcal{S}, j \vdash (P, E)}$$

$$(NEW)\quad \frac{P[j] = \texttt{new } \sigma \quad j+1 \in Dom(P) \quad F_j \leq F_{j+1} \& F_{E[j]} \quad rel(U) \quad S_{j+1} \leq \sigma / U \cdot S_j}{\mathcal{F}, \mathcal{S}, j \vdash (P, E)}$$

$$(START)\quad \frac{P[j] = \texttt{start } \sigma \quad j+1 \in Dom(P) \quad F_j \leq F_{j+1} \& F_{E[j]} \quad S_j \leq \sigma / \mathbf{0} \cdot S_{j+1}}{\mathcal{F}, \mathcal{S}, j \vdash (P, E)}$$

$$(RETURN)\quad \frac{P[j] = \texttt{return} \quad \forall x.\ F_j(x) \leq \mathbf{Top} \quad \forall n.\ S_j(n) \leq \mathbf{Top}}{\mathcal{F}, \mathcal{S}, j \vdash (P, E)}$$

$$(MEntr)\quad \frac{p[j] = \texttt{monitorenter } x \quad j+1 \in Dom(P) \quad F_j(x) \leq L.F_{j+1}(x) \quad F_j(y) \leq F_{j+1}(y)\ (y \neq x) \quad S_j \leq S_{j+1}}{\mathcal{F}, \mathcal{S}, j \vdash (P, E)}$$

$$(MExt)\quad \frac{P[j] = \texttt{monitorexit } x \quad j+1 \in Dom(P) \quad F_j(x) \leq \widehat{L}.F_{j+1}(x) \quad F_j(y) \leq F_{j+1}(y)\ (y \neq x) \quad S_j \leq S_{j+1}}{\mathcal{F}, \mathcal{S}, j \vdash (P, E)}$$

$$(ATHROW)\quad \frac{P[j] = \texttt{athrow} \quad i \in Dom(E) \Rightarrow F_j(x) \leq F_{E[i]}(x) \quad i \notin Dom(E) \Rightarrow F_j(x) \leq \mathbf{Top}}{\mathcal{F}, \mathcal{S}, j \vdash (P, E)}$$

図 1: The static semantics of $JVML_T$

る．Usage 変数 α_i はアドレス i 以降でローカル変数 0 に格納されているオブジェクトがどのように使用されるかを表す．スタックの型環境 S は，$S_3 = \mathbf{Int} \cdot \epsilon$，3 以外の i については $S_i = \epsilon$ である．

定義 12 より，先頭アドレス 1 について以下の制約が生成される．

$$rel(\alpha_1)$$

さらに，図 1 の規則より各アドレスについて以下の制約が生成される．

$$\begin{array}{ll}
\alpha_1 \leq L.\alpha_2 & (\text{アドレス 1 より}) \\
\alpha_2 \leq \alpha_3 & (\text{アドレス 2 より}) \\
\alpha_3 \leq \alpha_4 \& \alpha_6 & (\text{アドレス 3 より}) \\
\alpha_4 \leq \widehat{L}.\alpha_5 & (\text{アドレス 4 より}) \\
\alpha_5 \leq \mathbf{0} & (\text{アドレス 5 より}) \\
\alpha_6 \leq \widehat{L}.\alpha_7 & (\text{アドレス 6 より}) \\
\alpha_7 \leq \mathbf{0} & (\text{アドレス 7 より})
\end{array}$$

これらの制約より，$\alpha_1 = L.((\widehat{L}.\mathbf{0})\&(\widehat{L}.\mathbf{0}))$ が求まり，これは最初の制約 $rel(\alpha_1)$ を満たす．従って，例 2 のメソッドは正しくロックプリミティブを使用していることが分かる．

なお，仮にアドレス 3 が `if` 7 だった場合には上の制約 $\alpha_3 \leq \alpha_4 \& \alpha_6$ の代りに $\alpha_3 \leq \alpha_4 \& \alpha_7$ が生成される．この場合は $\alpha_1 = L.((\widehat{L}.\mathbf{0})\&\mathbf{0})$ となり，$rel(\alpha_1)$ が成り立たないので，ロックプリミティブの使用法が間違っていることが検出できる．

4 関連研究

本研究の型システムは，Abadi と Stata による JVM の型システム [9] に usage 情報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．1 節で述べたように，Bigliardi と Laneve[2] は，ロックプリミティブの使用法を検証する型システムを提案しているが，受理できるコードに関する制限がきつく，かつ型システムが複雑である．型にその値がどのような順序で使用されるかの情報を加えて拡張するという考え方は，小林ら [5, 4] による π 計算 [7, 8] におけるデッドロックの解析などのために導入されている．本研究の usage がオブジェクトのロックが獲得・解放される順序を表すのに用いられているのに対し，彼らの型システムでは通信チャネルが送信・受信に用いられる順序を表現するのに使用されている．

5 結論

JAVA のバイトコード言語 [6] の部分言語 に対して，ロックプリミティブの使用法を検証するための新しい型システムを提案した．

今後の課題としては，本論文で扱っていない命令（オブジェクトのフィールドへのアクセス，メソッド呼び出し，サブルーチン，スレッド関連の命令）を追加して型システムを拡張し，それに基づくバイトコード検証器を実装することが挙げられる．また，現在の型システムでは解析の精度が十分でなく，実際には正しいコードであっても拒絶する場合があるので，その改善も今後の課題である．

参考文献

[1] K. Arnold and J. Gosling. *The Java programming Language*. Addison Wesley, 1996.

[2] Gaetano Bigliardi and Cosimo Laneve. A type system for JVM threads. In *Proceedings of The Third ACM SIGPLAN Workshop on Types in Compilation (TIC 2000)*, 2000.

[3] Futoshi Iwama and Naoki Kobayashi. A new type system for JVM lock primitives. In preparation, 2001.

[4] Naoki Kobayashi. Type systems for concurrent processes: From deadlock-freedom to livelock-freedom, time-boundedness. In *Proceedings of IFIP International Conference on Theoretical Computer Science (TCS2000)*, volume 1872 of *Lecture Notes in Computer Science*, pages 365–389, August 2000. Invited Talk.

[5] Naoki Kobayashi, Eijiro Sumii, and Shin Saito. An implicitly-typed deadlock-free process calculus. In *Proceedings of CONCUR2000*, volume 1877 of *Lecture Notes in Computer Science*, pages 489–503. Springer-Verlag, August 2000. The full version is available as technical report TR00-01, Dept. Info. Sci., Univ. Tokyo.

[6] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification (2nd edition)*. Addison Wesley, 1999.

[7] Robin Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1999.

[8] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I, II. *Information and Computation*, 100:1–77, September 1992.

[9] Raymie Stata and Martín Abadi. A type system for java bytecode subroutines. *ACM Transactions on Programming Languages and Systems*, 21(1):90–137, 1999.