Java バイトコードにおける
オブジェクト使用解析のための型システム

浜中 信行
東京大学大学院理学系研究科
hamanaka@yl.is.s.u-tokyo.ac.jp

住井 英二郎
東京大学大学院情報理工学系研究科
sumii@yl.is.s.u-tokyo.ac.jp

小林 直樹
東京工業大学大学院情報理工学研究科
kobayasi@cs.titech.ac.jp

米澤 明憲
東京大学大学院情報理工学系研究科
yonezawa@yl.is.s.u-tokyo.ac.jp

平成 14 年 2 月 15 日

概 要

Java バイトコード (JVML) において，オブジェクトの使用方法を解析・検証する型システムを提案する．ここでオブジェクトの使用方法とは，メソッド起動やフィールド変数アクセスの順序や回数をいう．これにより，たとえば「各々の`File` オブジェクトは，生成されたらまず初期化され，次に`open` されてから`read` または`write` され，最後に`close` される」といった，計算資源の使用方法が静的に推論・検査でき，セキュリティやデバッグの観点から有用であると期待される．

1 序章

一般に，プログラミング言語では，ファイルやデータベース，スタック，キュー等プログラムの外部または内部に存在する計算資源にアクセスを行うが，これら計算資源へのアクセスは往々にして順序が想定されている．例えば，UNIX の file は最初に open され次に read または write が任意回行われ，最後に close されるというアクセスが想定されている．また socket は，bind，listen，connect 等の操作に関して複雑なアクセス順序が想定されており，これらに反するアクセスを行うとエラーとなる．従って，計算資源に対するアクセス順序の制御を行うことは重要であり，特にその静的な検証はデバッグやセキュリティの観点から有用である．

一方，オブジェクト指向言語においては，各種計算資源はオブジェクトとして表現され，それらに対するアクセスはメソッドの起動やフィールド変数の読み書きを通じて行われることが多い．C++や Java など現在主流のクラスベース型付きオブジェクト指向言語では，オブジェクトの属するクラスにしたがって静的型付けを行うことにより，各々のオブジェクトが受理しうるメソッドの集合を管理している．これにより，たとえば「スタックに対して`dequeue` メソッドが呼ばれない」「キューに対して`pop` メソッドが呼ばれない」といった性質が保証され，Smalltalk のようないわゆる “message not understood” エラーが防止できる．しかしそれだけでは，たとえば「`open` されていない`File` オブジェクトが`read` される」「一つの File オブジェクトが`open` され，`close` されないまま再び`open` される」といった，メソッド呼び出しの順序や回数に依存するエラーは防止できない．この問題については，`File` クラスを`UnopenedFile` と`OpenedFile` に細分化するといった対処法もあるが，煩雑すぎて現実にはあまり実践されていない．

そこで本論文では，プログラムを過度に複雑にすることなく上述のような問題を解決するために，オブジェクトが受理するメソッドの列を管理する型システムを提案・定式化する．言語としては，単純かつ現実的なオブジェクト指向言語の一種である Java バイトコード (JVML) [10] のサブセットを対象とする．同様の型システムを Java の表面言語や他のオブジェクト指向言語 (の相当するサブセット) について応用することも可能なはずである．

以下に，我々の型システムの概要を順を追って述べる．

1.1 Stata と Abadi の型システム

我々の型システムは，Stata と Abadi が提案した JVML の型システム [13] に基づいている．JVML プログラムの実行では，メソッド呼び出し毎に局所変数とオペランドスタック (以下ではスタックと呼ぶ) が与えられる．彼らの型システムは，この実行モデルに対しプログラムの各番地毎に局所変数およびスタックに格納されている値の型を定めるために，各番地ごとに型環境を与える．

たとえば，次のような JVML プログラムを考える (説明のために，実際とはやや異なる疑似コードを用いる)．

```
1:   load x
2:   dup
3:   invoke File.open
4:   invoke File.close
```

このプログラムは，まず第 1 番地で変数 x の値を読み込み，その値を第 2 番地で 2 個に複製してスタックに push する．その 1 個目の値を第 3 番地でスタックから pop し，`File` クラスのオブジェクト (への参照) とみなして，open メソッドを呼び出す．さらに，2 個目の値を第 4 番地でスタックから pop し，`File` クラスの`close` メソッドを呼び出す．

Stata と Abadi の型システムは，上のプログラムの第 1 番地から第 4 番地にそれぞれ次のような型環境を与えることができる．

$$
\begin{aligned}
\Gamma_1 &= x : \mathit{File} \\
\Gamma_2 &= s_0 : \mathit{File} \\
\Gamma_3 &= s_0 : \mathit{File},\ s_1 : \mathit{File} \\
\Gamma_4 &= s_0 : \mathit{File}
\end{aligned}
$$

ここで $x : \mathit{File}$ 等は，変数 x 等の値が`File` オブジェクトへの参照であることを表す．s_i はスタックの i 番目を表す特殊な変数名である．(原論文 [13] の定式化ではスタックとローカル変数の型環境が分離されていたが，ここでは以降の説明のためにそれらを統一する．)

1.2 型の拡張

上のように Stata と Abadi の型システムでは，各オブジェクトの型としてそのオブジェクトが属するクラスのみを考える．しかし冒頭で議論したように，それだけでは実際にはオブジェクトが処理できないメソッド呼び出しが発生しうる．例えば上の例で，第 3 番地の`open` メソッド呼び出しと第 4 番地の`close` メソッド呼び出しを入れ替えても，Stata と Abadi の型システムでは (型の上では) 正しいプログラムと判定されるが，実行時には`open` されていない`File` オブジェクトに対して`close` メソッドが呼び出されてしまう．

この問題を解決するために我々の型システムでは，まず各オブジェクトの型を *usage* と呼ばれる式 [14] によって拡張する．一般に usage は型の一部として付加され，その型を持つ値の使用方法を表す [6–9, 14]．本論文では，オブジェクトに対するメソッド呼び出しの順序や回数を表すために usage を用いる．たとえば，「`open` されてから`close` される」というオブジェクトの使用方法は，`open`; `close` という usage によって表される．

usage を用いると，先のプログラムに対する型環境は次のように拡張される．

$$
\begin{aligned}
\Gamma_1 &= x : \mathit{File}/(\mathit{open}; \mathit{close}) \\
\Gamma_2 &= s_0 : \mathit{File}/(\mathit{open}; \mathit{close}) \\
\Gamma_3 &= s_0 : \mathit{File}/\mathit{open},\ s_1 : \mathit{File}/\mathit{close} \\
\Gamma_4 &= s_0 : \mathit{File}/\mathit{close}
\end{aligned}
$$

ここで，$\mathit{File}/(\mathit{open}; \mathit{close})$ 等における $\mathit{open}; \mathit{close}$ の部分が付加された usage である．たとえば，$x : \mathit{File}/(\mathit{open}; \mathit{close})$ は「変数 x の値は，これからまず`open` され，その後に`close` されるような`File` オブジェクトへの参照である」ことを表す．

このような型環境は，次のようにボトムアップに求めることができる．まず，第 4 番地では s_0 に対し `close` メソッドが呼ばれるので，Γ_4 における s_0 の usage は close となる．第 3 番地では s_0 が pop されて `open` メソッドが呼ばれるので，Γ_3 における s_0 の usage は open となり，また $\Gamma_3(s_1)$ は $\Gamma_4(s_0)$ に等しい．第 2 番地では`dup` 命令により s_0 が s_1 に複製されるので，$\Gamma_2(s_0)$ は $\Gamma_3(s_0)$ と $\Gamma_3(s_1)$ を合わせた型になる．最後に，第 1 番地では x の値が s_0 に読み込まれるので，$\Gamma_1 = [x/s_0]\Gamma_2$ となる．

1.3 型環境の拡張

ところが，上のような型環境を求める際に，実際には以下のような問題が発生する．第 2 番地の命令が`dup` であることから $\Gamma_2(s_0)$ は $\Gamma_3(s_0)$ と $\Gamma_3(s_1)$ を合わせた型

になるが，$\Gamma_2(s_0)$ の usage 部分について *open* と *close* の順序はΓ_3 からは知り得ず，*open*; *close* と *close*; *open* のどちらか判断できない．この問題は，第 3 番地における s_0 と s_1 が実は同じオブジェクトを指す別名(alias) であるのに，それらの間のメソッド呼び出しの順序情報が Γ_3 において失われているために起こる．同様の問題は`dup` だけでなく，別名を生成しうるすべての命令 (ローカル変数の読み込みを行う`load` 等) について発生する．

この問題を解決するために我々は，異なる変数もまたがってメソッド呼び出しの順序を扱うよう，$s_0 : \mathit{File}/\mathit{open}, s_1 : \mathit{File}/\mathit{close}$ のような「変数名から型への写像」ではなく，$s_0 : \mathit{File}/\mathit{open}; s_1 : \mathit{File}/\mathit{close}$ のような「各変数の型付けの順序列」に型環境を拡張する．

これにより，たとえば先の例では

$$\Gamma_3 \quad = \quad s_0 : \mathit{File}/\mathit{open};\ s_1 : \mathit{File}/\mathit{close}$$

ととれば，$\Gamma_2(s_0)$ の usage 部分が *open*; *close* となることがわかる．

1.4 usage の拡張

さらに，我々の型システムではメソッド呼び出しだけでなく，フィールド変数に対する読み書きの順序や回数の情報も，便宜的にオブジェクトの使用方法と考えて型に付け加える．その理由は，型システムの健全性に関わるもので，以下の通りである．

JVML では`putfield` 命令により，オブジェクトのフィールド変数に対して破壊的代入を行うことができる．このような破壊的代入は別名が存在しうる状況では型システムの健全性のために特別な注意を必要とする．(これらの破壊的代入は参照ないしポインタが「指す先」への代入であり，単なる手続き型言語の代入文や JVML の`store` 命令とは異なる．Scheme 言語に例えれば `set!`ではなく `set-car!`や `set-cdr!`に，C 言語で言えば `x = y` 等ではなく `p->x = y` や`*p = z` にあたる．)

まず素朴な方法が困難であることを確認するために，次のようなプログラムを考える．(ここでは簡単のために，バイトコードではなく Java の表面言語風の構文を使用する．)

```
 1:   f = new File();
 2:   g = new File();
 3:   f.open();
 4:   g.open();
 5:   x = new FileContainer();
 6:   y = new FileContainer();
 7:   if (...) then y = x;
 8:   x.contents = f;
 9:   y.contents = g;
10:   x.contents.close();
11:   y.contents.close();
```

これは，二つの`File` オブジェクトを生成し，それぞれ`open` した後コンテナに格納してから`close` するというプログラムである．

このプログラムにおいて，第 8 行目の型環境 Γ_8 としてすべての usage が空であるものをとり，第 10 行目の型環境 Γ_{10} として仮に

$$x.\mathit{contents} : \mathit{File}/\mathit{close};\ y.\mathit{contents} : \mathit{File}/\mathit{close}$$

のようなものをとれば，一見すると型付けが成功してしまう．ここで $x.\mathit{contents} : \mathit{File}/\mathit{close}$ 等は，「変数`x` のフィールド`contents` は，これから`close` されるべき`File` オブジェクトを持つ」等を表すとする．

しかし，第 7 行目より`x` と`y` は同じコンテナを指しているかもしれず，次のことが起こり得る．

- 第 1 行目で生成された`File` オブジェクトが`close` されない
- 第 2 行目で生成された`File` オブジェクトに対し，`close` が二回呼ばれる

よって，このプログラムはむしろ型エラーとなるべきであり，上のようなアプローチはうまくいかない．

この問題を解決するために，我々はフィールド変数の読み書きに関する情報もオブジェクトの型に付加する．具体的には，「今後型 τ に従って使われるべき値をフィールド a から読出す」ことを表す項 $\mathbf{r}(a, \tau)$ と，「今後型 τ に従って使われるべき値をフィールド a に書込む」ことを表す項 $\mathbf{w}(a, \tau)$ を usage に加える．

すると，たとえば上述のプログラムで Γ_{10} は

$$\begin{array}{l} x : \mathit{FileContainer}/\mathbf{r}(\mathit{contents}, \mathit{File}/\mathit{close}); \\ y : \mathit{FileContainer}/\mathbf{r}(\mathit{contents}, \mathit{File}/\mathit{close}) \end{array}$$

となり，Γ_8 は

$$\begin{array}{l} x : \mathit{FileContainer}/\mathbf{w}(\mathit{contents}, \mathit{File}/\mathit{close}); \\ y : \mathit{FileContainer}/\mathbf{w}(\mathit{contents}, \mathit{File}/\mathit{close}); \\ \Gamma_{10} \end{array}$$

のようになる．さらに第 7 行目の型環境 Γ_7 は，「Γ_8 と $[x/y]\Gamma_8$ のどちらか」を表す型環境 $\Gamma_8 \mathbin{\&} [x/y]\Gamma_8$ に等しいと考えられる．ここで，Γ_8 は第 3 行目で `y = x` が実行されない場合に，$[x/y]\Gamma_8$ はそれが実行される場合に相当する型環境である．これらのうち後者の型環境における`x` の usage を見てみると

$$\begin{aligned}&\mathbf{w}(\mathit{contents}, \mathit{File}/\mathit{close});\\&\mathbf{w}(\mathit{contents}, \mathit{File}/\mathit{close});\\&\mathbf{r}(\mathit{contents}, \mathit{File}/\mathit{close});\\&\mathbf{r}(\mathit{contents}, \mathit{File}/\mathit{close})\end{aligned}$$

となっており，

- 最初に`contents` に書き込まれる`File` オブジェクトが，*close* という usage に反して，`close` されないまま上書きされる
- 次に`contents` に書き込まれる`File` オブジェクトに対し，*close* という usage に反して，`close` が二回呼ばれる

ことがわかるので，型エラーとなって型システムの健全性が保たれる．

本論文の以降の構成は次の通りである．まず，2 章で対象言語 JVML_R を定義する．次に，3 章で JVML_R の型システムを与える．4 章で関連研究について議論し，最後に 5 章で結論を述べる．

2　対象言語 $\mathbf{JVML}_R$

対象言語 JVML_R は，Java バイトコード言語 JVML の部分言語である．実行モデルは JVML 同様で，メソッド呼び出しごとにフレームが生成され，フレームごとに局所変数とスタックが与えられる．また，これとは別に動的なデータ領域としてヒープを有する．そして，局所変数，スタックおよびヒープに対する操作として命令が与えられる．

JVML_R を以下で定義する．

2.1　$\mathbf{JVML}_R$ の構文

A をアドレスの集合，N を自然数の集合，LV を局所変数の集合とする．A と LV の要素は実際は自然数である．LV の要素については，N の要素と明示的に区別するため f_n と表記する．

定義 1 (命令)　JVML_R の命令 I の構文は，以下で定義される．

$$\begin{array}{rcl} I & ::= & \texttt{inc} \mid \texttt{push0} \mid \texttt{pop} \mid \texttt{if}\ j \\ & \mid & \texttt{load}\ f_n \mid \texttt{store}\ f_n \mid \texttt{dup} \\ & \mid & \texttt{putfield}\ \sigma.a \mid \texttt{getfield}\ \sigma.a \\ & \mid & \texttt{new}\ \sigma \mid \texttt{invoke}\ \sigma.m \mid \texttt{return} \end{array}$$

ここで，f_n，j はそれぞれ LV，A の要素を値にとる変数である．また，σ，m，a はそれぞれクラス名，メソッド名，フィールド名を表す．σ，m，a に対し $\sigma.m$，$\sigma.a$ をそれぞれ絶対メソッド名，絶対フィールド名と呼ぶ．I の集合を**Inst** と書く．

各命令の意味は次の通りである．`inc` は，スタックトップの整数を 1 増やす．`push0` は，スタックトップに整数 0 を置く．`pop` は，スタックトップから値を取り除く．`if` j は，スタックトップの整数を取り除き，それが 0 ならば次のアドレスに移り，そうでなければアドレス j に分岐する．`load` f_n は，局所変数 f_n の値をスタックに積む．`store` f_n は，スタックトップから値を除き，その値を局所変数 f_n に格納する．`dup` は，スタックトップの値を複製してスタックトップに置く．`putfield` $\sigma.a$ は，スタックの 2 番目の値が指すクラス σ のオブジェクトのフィールド a に，スタックトップの値を書き込む (両値は取り除く)．`getfield` $\sigma.a$ は，スタックトップの値を取り除き，その値の指すオブジェクトのフィールド a の値をスタックに積む．`new` σ は，クラス σ のオブジェクトを生成しスタックに積む．`invoke` $\sigma.m$ は，$n = \mathit{numargs}(\sigma.m)$ とすると，スタックトップから n 番目までの値を引数として，$n+1$ 番目の値の指すクラス σ のオブジェクトのメソッドを呼び出し ($n+1$ 番目までスタックから取り除く)，返り値をスタックに積む．`return` はスタックトップを返り値として現在実行中のメソッドを終了する．

メソッド本体は，A から**Inst** への部分関数であり，メタ変数 P を用いて表す．メソッド定義は，メソッド名の集合から P の集合への部分関数であり，メタ変数 MD を用いて表す．また，フィールド定義は，フィールド名の集合から基本型の集合への部分関数で，メタ変数 FD を用いて表す．ただし，基本型は **Int** または クラス名 σ で表され，それぞれそのフィールドに格納される値が整数値，クラス σ のオブジェクトへの参照であることを表す．

クラス定義は，組 (MD, FD) であり，メタ変数 Σ を用いて表す．JVML_R のプログラムは，クラス名の集

合から Σ の集合への部分関数であり，メタ変数 Q を用いて表す．プログラム中にはただ一つのメインメソッドが存在し，実行はメインメソッドから始まるものとする．プログラム Q のメインメソッドの絶対メソッド名を Q_{main} と表す．

クラス定義 $\Sigma = (MD, FD)$ に対して，Mtd と Fld を，$Mtd(\Sigma) = MD$，$Fld(\Sigma) = FD$ と定義する．また，クラス名 σ，メソッド名 m，プログラム Q に対して，メソッド本体 $Mtd(Q(\sigma))(m)$ を簡便のため絶対メソッド名を用いて $Q(\sigma.m)$ とも書くことにする．同様に，$Fld(Q(\sigma))(a)$ を $Q(\sigma.a)$ と書く．

また，絶対メソッド名 $\sigma.m$ に対して，引数の数を $numargs(\sigma.m)$，i 番目の引数の基本型を $argtyp(\sigma.m, i)$，返り値の基本型を $rettyp(\sigma.m)$ と書く．

2.2 操作的意味

本節では，JVML_R に対して操作的意味を定義する．まず，JVML_R の実行モデルをマシンステートの遷移として定式化する．次に，プログラムの実行過程で各オブジェクトが正しく使用されることを検証するため，オブジェクト使用に関する安全性の概念を定義しこれを操作的意味に組み込む．

O をオブジェクトへの参照の集合とする．O は，無効な参照を表す値 $null$ を含むものとする．値は，集合 $N \cup O$ の要素である．値の集合を VAL とする．また，オブジェクト値とは，フィールド名の集合から VAL への部分関数である．オブジェクト値の集合を R とする．フィールド定義はフィールド名へ基本型を割り当てるのに対し，オブジェクト値はフィールド名へ値を割り当てる．

なお，以下では関数 f について，f の定義域および値域をそれぞれ $dom(f)$，$codom(f)$ で表す．

定義 2 (使用，使用集合)　オブジェクトの使用とは，そのオブジェクトのメソッド名の有限列または無限列である (空列を含む)．使用集合とは，使用の集合でありメタ変数 Φ を用いて表す．

使用は，オブジェクトの使われ方を表すもので，どの順序でどのメソッドが呼ばれるかを表す．有限列の使用はプログラム終了までに行われるメソッド呼び出しを表し，無限列の使用はサーバプログラムのような停止しないプログラムの実行におけるメソッド呼び出しを表す．使用集合 Φ は，プログラムの実行においてオブジェクトに対する使用を規定し，その時点で Φ に含まれる使用の表すメソッド呼び出しのみ行うことができる．例えば，$\Phi = \{open \cdot close, open \cdot read \cdot close, open \cdot write \cdot close\}$ という使用集合が与えられているとき，そのオブジェクトに対して実行可能なメソッド呼び出しは，「まず open し，次に高々1 回 read または write を行い最後に close する」となる．

本論文では，使用集合の与え方は特に定めない．正則表現や文脈自由文法で表現してもよいし，3 章で定義する usage (型環境) を用いてもよい．

なお，JVML_R では使用集合はクラス毎に既定とし，クラス σ に対する使用集合を Φ_σ と表す．また，Φ^{-m} を $\{\ s \mid ms \in \Phi\ \}$ と定義する．これは，使用集合が Φ であるオブジェクトに対しメソッド m を呼んだ後の使用集合を表す．例えば，先の Φ に対して，$\Phi^{-open} = \{close, read \cdot close, write \cdot close\}$，$(\Phi^{-open})^{-read} = \{close\}$ となる．

次に実行モデルの定式化を行う．

組 $\langle \sigma.m, pc, f, s \rangle$ をフレームと呼ぶ．$\sigma.m$ は現在実行中のメソッドの絶対メソッド名である．pc は A の要素を値とする変数で，現在の実行アドレスを表す．f は LV から VAL への部分関数で，局所変数への値の割り当てを表す．s は値の列で，スタックの状態を表す．例えば，絶対メソッド名 $\sigma.m$ のメソッドの実行開始時点でのフレームは $\langle \sigma.m, 1, f, \epsilon \rangle$ となる (ϵ は空列で，空のスタックを表す)．フレームスタックとは，フレームの列であり，メソッドの呼び出し，終了ごとにフレームが生成，破棄されることを表す．メタ変数 FS を用いて表す．空のフレームスタックは空列 ϵ で表す．

ヒープ値とは，R の要素 ρ と使用集合の組 (ρ, Φ) である．ヒープとは，O からヒープ値の集合への部分関数であり，メタ変数 H を用いて表す．オブジェクトの使用を規定する使用集合 Φ は，実際の使用に伴い動的に変化するもので，オブジェクト値 ρ と合わせてヒープ値とする．ただし，Φ は後で定義するオブジェクト使用に関する安全性の定義のために用いるものであって，現実の実行においてはヒープに保持されない．

組 (FS, H) を，マシンステートとする．プログラム Q においてある命令を実行したときにマシンステートが (FS, H) から (FS', H') に移る関係を

$$Q \vdash (FS, H) \rightarrow (FS', H')$$

で表す．JVML_R の操作的意味は，マシンステートの遷移関係として与えられる．

定義 3 マシンステートの遷移関係 $Q \vdash (FS, H) \rightarrow (FS', H')$ とは，図 1 の規則を満たす最小の関係である．

図 1 の各規則において，n，o，v，ρ はそれぞれ N，O，VAL，R の要素を値にとる変数である．関数 f について $f[f_n \mapsto v]$ は，$f(f_n)$ の値を v に変更して得られる関数とする．ρ，H についても同様である．また，マシンステート (FS, H) に対して，図 1 のうち適用できる規則がない場合，$Q \vdash (FS, H) \rightarrow$ **Error** と書く．また，この遷移関係の反射的推移的閉包を $Q \vdash (FS, H) \rightarrow^* (FS', H')$ と書く．

それぞれの遷移規則の説明は論文 [5] に譲り，ここでは代表的な規則のみ説明する．

規則 (E-New): 左の前提は，アドレス pc の命令が `new` σ' であることを表す．命令の実行によりクラス σ' のオブジェクトが生成され，使用集合 $\Phi_{\sigma'}$ と共にヒープに割り当てられる．矢印の右のマシンステートが，命令実行後の状態を表し，スタックトップは生成されたオブジェクトへの参照となる．

規則 (E-Invoke):矢印の左のマシンステートは，命令実行前のスタックに引数 $\tilde{v}$ とメソッドを呼び出すオブジェクトへの参照 o が積まれていることを表す．メソッド呼び出しは o の使用集合 Φ によって想定されている必要があり，これは前提 $\Phi^{-m'} \neq \emptyset$ が表している．命令実行後，スタックトップに返り値が積まれ，o の使用集合は $\Phi^{-m'}$ となる．実行後のマシンステートのスタックフレームの延びが，メソッド呼び出しに伴うフレームの生成を表す．

最後に，オブジェクト使用の安全性の定義を行う．プログラム Q がオブジェクト使用に関して安全であるとは，Q の実行において各オブジェクトのメソッド呼び出しが正しく行われ，かつ実行の終了時にすべてのオブジェクトが正しく使い尽くされていることを言う．

定義 4 プログラム Q に対して，次の条件が成り立つときかつその時に限り，Q はオブジェクト使用に関して安全であるという．(1) $Q \vdash (\langle Q_{main}, 1, f_0, \epsilon \rangle, H_0) \not\rightarrow^*$ **Error** かつ (2) $Q \vdash (\langle Q_{main}, 1, f_0, \epsilon \rangle, H_0) \rightarrow^* (\epsilon, H)$ ならば，$h \in dom(H)$ であるすべての h に対してある ρ，Φ が存在し $H(h) = (\rho, \Phi)$ かつ $\epsilon \in \Phi$．ただし，H_0 は $dom(H_0) = \emptyset$ であるようなヒープとする．

次節で，プログラム Q のオブジェクト使用の安全性を判定する型システムを定義する．

3 型システム

3.1 型，usage，型環境

序章で述べたように，通常のオブジェクトの型にそのオブジェクトが今後どのように使われるかを表す式 usage を付加する．また，異なる変数間の順序を保存するよう型環境を拡張する．その上で，使用に関する型環境の意味を定義する．

3.1.1 型，usage，型環境の定義

スタックの n $(\in N)$ 番目の要素をスタック変数 s_n で表す．スタック変数の集合を SV とする．変数とは，$LV \cup SV$ の要素であり，以下の型環境で型の割り当てを行う変数である．変数の集合を V とする．

定義 5 (型，usage，型環境) 型 τ，usage U，型環境 γ を以下で定義する．

$$
\begin{array}{rcl}
\tau & ::= & \mathbf{Int} \mid \mathbf{Top} \mid \sigma/U \\
U & ::= & \mathbf{0} \mid m \mid \mathbf{w}(a,\gamma) \mid \mathbf{r}(a,\gamma) \\
\gamma & ::= & \emptyset \mid \alpha \mid x:\tau \mid \mu\alpha.\gamma \\
& \mid & \gamma_1 \,;\, \gamma_2 \mid \gamma_1 \,\&\, \gamma_2 \mid \Diamond\gamma \\
& \mid & [+_n]\gamma \mid [-_n]\gamma \mid [x_2/x_1]\gamma \\
& \mid & \gamma\downarrow_x \mid \gamma\uparrow_x \\
x & ::= & f_n \mid s_n
\end{array}
$$

型 τ について，**Int** は整数値の型である．**Top** は Java 仮想機械の仕様上必要なもので，今後使われない値の型を表す．σ/U は今後 U に従って使用されるようなクラス σ のオブジェクトの型である．

usage U の意味は次の通りである．$\mathbf{0}$ は，今後使われないという使用を表す．m は，メソッド m がちょうど一回呼ばれる使用を表す．$\mathbf{w}(a,\gamma)$，$\mathbf{r}(a,\gamma)$ はフィールドに対する読み書きを表し，それぞれフィールド a に対して γ に従って使われる値を書込むあるいは読出すという使用を表す．序章で示した ; や & のような usage の構成子は説明のためのものであり，ここでは定義しない．同様の構成子が型環境に定義されているからである．また，フィールド操作 **w**,**r** についても，型ではなく型環境を保持するような定義となっている．

型環境について，$\emptyset$ は，空の型環境を表す．α は，型環境を値にとる変数であり $\mu\alpha.$ によって束縛される．$x:\tau$ は，変数 x の値は型 τ に従って使用されること

を表す．ここで x は V の要素を値にとる変数であり，f_n $(\in LV)$ が局所変数，s_n $(\in SV)$ がスタック変数を表す．$\mu\alpha.\gamma$ は，再帰型環境で，γ 中の α を $\mu\alpha.\gamma$ 全体で置き換えて得られる型環境を表す．γ_1 ; γ_2 は，γ_1 に従ってプログラムが実行された後，γ_2 に従って実行されることを表し，逐次実行に対応する．γ_1 & γ_2 は，γ_1 または γ_2 に従ってプログラムが実行されることを表し if 命令などの分岐に対応する．$\Diamond\gamma$ は，プログラムが型環境の構成子 “; ” に関して今あるいは後で γ に従って実行されることを表すもので，例えば $(\Diamond\gamma_1);\gamma_2$ は，プログラムが γ_1 に従って実行された後 γ_2 に従って実行される，あるいは γ_2 に従って実行された後 γ_1 に従って実行されることを表す．$\Diamond$ は，putfield や return のように値がエスケープし得る場合を扱うためのものである．

$[+_n]\gamma$ および $[-_n]\gamma$ は，γ 中のすべてのスタック変数 $s_{n'}$ をそれぞれ $s_{n'+n}$ あるいは $s_{n'-n}$ で置き換えて得られる型環境である．ただし，$n'-n<0$ である $s_{n'-n}:\tau$ は $\emptyset$ で置き換えられる．これは，命令の実行に伴いスタックが変動する (添字がずれる) ことに対応する．$[x_2/x_1]\gamma$ は γ 中の x_1 をすべて x_2 に置き換えて得られる型環境である．load，store，dup の実行時に別名が発生することに対応する．

$\gamma\downarrow_x$ は，γ から変数 x に関する束縛のみを取出した型環境である (x 以外の束縛は $\emptyset$ で置換する)．$\gamma\uparrow_x$ は，逆に x 以外の変数に関する束縛を γ から取出した環境となる．$\gamma\downarrow_x$ および $\gamma\uparrow_x$ の形式的な定義は論文 [5] に譲る．

なお，型環境の構成子の結合について，二項構成子より単項構成子の結合が強く，また ; より & の結合が強いものとする．すなわち，$\gamma_1;\Diamond\gamma_2\&\gamma_3$ は $\gamma_1;((\Diamond\gamma_2)\&\gamma_3)$ を意味する．

例 6 $s_0:File/open; s_1:File/read\&s_1:File/write$ は，変数 s_0 の指すFile オブジェクトのメソッドopen を 1 回呼び，次に変数 s_1 の指すFile オブジェクトのメソッドread あるいはwrite を 1 回呼ぶプログラムの型環境である．また，$x:\mathbf{Int};\mu\alpha.\ ((y:File/read;\alpha)\&\emptyset)$ は，x の値を整数として使用した後，y の指す File オブジェクトのread メソッドを任意回数呼ぶプログラムの型環境である．

例 7 次のプログラムは，Socket オブジェクトを UNIX の socket の様に使用する例である．ただし，この使い方は説明のためのものであり Java 標準 API の定義とは異なる．

```
Socket s = new Socket();
if (server==1) {   // server side
    s.bind();  s.listen();
    while(x>=0) s.accept();
} else {           // client side
    s.connect();
    while (y==0) if (z<0) s.read();
                 else s.write();
}
s.close();
```

アドレス 2 は，次の型環境 γ で型付けされる．

$$\begin{aligned}
\gamma &= server:\mathbf{Int};\ \gamma_s\&\gamma_c;\ s:Socket/close\\
\gamma_s &= (s:Socket/bind; s:Socket/listen;\\
&\qquad \mu\alpha_s.(x:\mathbf{Int};(s:Socket/accept;\alpha_s)\&\emptyset))\\
\gamma_c &= (s:Socket/connect;\\
&\qquad \mu\alpha_c.(y:\mathbf{Int};\\
&\qquad\qquad (z:\mathbf{Int}; s:Socket/read\ \&\\
&\qquad\qquad\qquad s:Socket/write;\alpha_c)\ \&\ \emptyset))
\end{aligned}$$

例 8 型環境 $\gamma = x:F/open;\ y:F/read;\ (y:F/read; x:F/close)\ \&\ y:F/write$ に対して

$$\begin{aligned}
&\gamma\downarrow_x = x:F/open;\ x:F/close\ \&\ \emptyset\\
&\gamma\downarrow_y = y:F/read;\ y:F/read\ \&\ y:F/write
\end{aligned}$$

型環境 γ に対する操作 $base(\gamma)$ を定義する．定義の詳細は論文 [5] に譲るが，直観的には，γ をある一つの変数に対する型環境とみなしその変数の値の基本型 (あるいは$\mathbf{Top}$) を返す操作である．例えば，$base(x:File/open; x:File/close\&x:\mathbf{Top}) = File$ であり，$base(\mu\alpha.(y:\mathbf{Int}\&\emptyset;\alpha)) = \mathbf{Int}$ である．一方，$base(x:File/read; x:\mathbf{Int})$ は定義されない．

また，拡張した型環境 γ に対し，定義域$dom(\gamma)$ を通常の型環境の定義域と同様に定義する (定義の詳細は論文 [5] 参照)．例えば，$dom(s_0:File/open; s_1:\mathbf{Int}; s_0:File/\mathbf{0}) = \{s_0, s_1\}$ であり，$dom(x:\mathbf{Int};\mu\alpha.(y:Socket/accept)) = \{x, y\}$ である．

次に，部分型関係 $\tau_1 \leq \tau_2$ および部分型環境関係 $\gamma_1 \leq \gamma_2$ を定義する．直観的には，$\tau_1 \leq \tau_2$ は，型が τ_1 である値を型が τ_2 である値として用いてよいことを表す．$\gamma_1 \leq \gamma_2$ は，γ_2 に従って実行されるプログラムを γ_1 に従って実行されるとみなしてよいことを表す．

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．

$$\mathbf{Int} \leq \mathbf{Top} \qquad \sigma/\mathbf{0} \leq \mathbf{Top}$$

注: 本論文では，通常の部分型関係である部分クラス関係 $\sigma_1 \leq \sigma_2$ は扱わない．

型環境の 2 項関係 $\equiv$ を，& に関する交換律と結合律，; に関する結合律および $\gamma; \emptyset \equiv \emptyset; \gamma \equiv \gamma$，$\mu\alpha.\gamma \equiv [\alpha \mapsto \mu\alpha.\gamma]\gamma$ を満たす最小の合同関係であるとする．

定義 10 部分型環境関係 $\leq$ は，型環境の 2 項関係 $\equiv$ を含み，以下の規則を満たす最小の前順序関係である．

$$\gamma_1 \& \gamma_2 \leq \gamma_i \quad (i = 1, 2) \qquad x : \mathbf{Top} \leq \emptyset$$

$$\Diamond\gamma \leq \gamma \quad \Diamond\gamma_1; \gamma_2; \gamma_3 \leq \gamma_2; \Diamond\gamma_1; \gamma_3 \quad \frac{\tau_1 \leq \tau_2}{x : \tau_1 \leq x : \tau_2}$$

$$\frac{\gamma_1 \leq \gamma_1' \quad \gamma_2 \leq \gamma_2'}{\gamma_1; \gamma_2 \leq \gamma_1'; \gamma_2'} \qquad \frac{\gamma_1 \leq \gamma_1' \quad \gamma_2 \leq \gamma_2'}{\gamma_1 \& \gamma_2 \leq \gamma_1' \& \gamma_2'}$$

$$\frac{\gamma_1 \leq \gamma_2}{\Diamond\gamma_1 \leq \Diamond\gamma_2} \qquad \frac{\gamma_1 \leq \gamma_2}{\gamma_1 \downarrow_x \leq \gamma_2 \downarrow_x} \qquad \frac{\gamma_1 \leq \gamma_2}{\gamma_1 \uparrow_x \leq \gamma_2 \uparrow_x}$$

$$\frac{\gamma_1 \leq \gamma_2}{[\pm_n]\gamma_1 \leq [\pm_n]\gamma_2} \qquad \frac{\gamma_1 \leq \gamma_2}{[x_2/x_1]\gamma_1 \leq [x_2/x_1]\gamma_2}$$

$$\frac{\gamma_1 \leq \gamma_2}{x : \sigma/\mathbf{w}(a, \gamma_2) \leq x : \sigma/\mathbf{w}(a, \gamma_1)}$$

$$\frac{\gamma_1 \leq \gamma_2}{x : \sigma/\mathbf{r}(a, \gamma_1) \leq x : \sigma/\mathbf{r}(a, \gamma_2)}$$

下から 2 つ目の規則について，$x : \sigma/\mathbf{w}(a, \gamma)$ の関係は γ の関係と逆になっていることに注意されたい．

3.1.2 usage に関する型環境の意味

型環境 γ 中の usage の表す使用が対応する使用集合 Φ によって想定されたものであるとき，γ は Φ に対して妥当であるという．

以下で，usage に関する型環境の簡約規則を定義し，これを用いて型環境 γ 中の usage が表す使用の集合 $[\![\gamma]\!]$ を定義する．そして，$[\![\gamma]\!]$ と Φ の関係をみることで γ が Φ に対して妥当であるか判定する．

メソッド名の集合に空列 ϵ を加えたものを MS とし，MS の要素をメタ変数 M で表す．

定義 11 usage に関する型環境の簡約規則 $\gamma \xrightarrow{m} \gamma'$ は以下を満たす最小の関係である．

$$x : \sigma/m \xrightarrow{m} \emptyset \qquad x : \sigma/\mathbf{w}(a, \gamma) \xrightarrow{\epsilon} \emptyset$$

$$x : \sigma/\mathbf{r}(a, \gamma) \xrightarrow{\epsilon} \emptyset \qquad \mu\alpha.\gamma \xrightarrow{\epsilon} [\alpha \mapsto \mu\alpha.\gamma]\gamma$$

$$\frac{\gamma_1 \xrightarrow{M} \gamma_1'}{\gamma_1; \gamma_2 \xrightarrow{M} \gamma_1'; \gamma_2} \qquad \frac{\gamma_1 \leq \gamma_1' \quad \gamma_1' \xrightarrow{M} \gamma_2' \quad \gamma_2' \leq \gamma_2}{\gamma_1 \xrightarrow{M} \gamma_2}$$

本論文では，妥当性を考える際の使用の対象はメソッド呼び出しのみであり，フィールド操作は無視される (ϵ 簡約)．

定義 12 型環境 γ に対して，集合 $[\![\gamma]\!]$ を次のように定義する．ただし，$n \geq 0$ とする．

$$\begin{aligned}
&\{s = M_1 \cdots M_n \mid \exists\gamma_1, \cdots, \gamma_{n-1}. \\
&\quad (\gamma \xrightarrow{M_1} \gamma_1 \xrightarrow{M_2} \cdots \xrightarrow{M_{n-1}} \gamma_{n-1} \xrightarrow{M_n} \emptyset),\ s \in MS^*\} \\
\cup\ &\{s = M_1 \cdots M_n \cdots \mid \exists\gamma_1, \cdots, \gamma_n, \cdots. \\
&\quad (\gamma \xrightarrow{M_1} \gamma_1 \xrightarrow{M_2} \cdots \xrightarrow{M_{n-1}} \gamma_{n-1} \xrightarrow{M_n} \gamma_n \xrightarrow{M_{n+1}} \cdots), \\
&\quad s \in MS^\omega\}
\end{aligned}$$

上の集合がプログラム終了までに行われる使用に対応し，下の集合が停止しないプログラムにおいて行われる使用に対応している．

例 13

$$\begin{aligned}
&[\![s_0 : \mathit{File}/\mathit{open}; s_0 : \mathit{File}/\mathit{read} \& s_0 : \mathit{File}/\mathit{write}]\!] \\
= &\{\ \mathit{open},\ \mathit{open} \cdot \mathit{read},\ \mathit{open} \cdot \mathit{write}\ \} \\
&[\![x : \mathit{File}/\mathit{open}; \mu\alpha.((x : \mathit{File}/\mathit{read}; \alpha) \& \emptyset)]\!] \\
= &\{\ \mathit{open},\ \mathit{open} \cdot \mathit{read},\ \mathit{open} \cdot \mathit{read} \cdot \mathit{read},\ \ldots\ \} \\
&[\![x : \mathit{Socket}/\mathit{open}; \mu\alpha.(x : \mathit{Scoket}/\mathit{accept}; \alpha)]\!] \\
= &\{\ \mathit{open} \cdot \mathit{accept} \cdot \mathit{accept} \cdot \mathit{accept} \cdot \mathit{accept} \cdots\ \}
\end{aligned}$$

定義 14 型環境 γ と使用集合 Φ について，以下が成り立つとき γ は Φ に対して妥当であるという．

$$[\![\gamma]\!] \subseteq \Phi$$

型推論では最終的に型環境の妥当性の判定を行うが，Φ の与え方によっては妥当性の判定は決定不能になる．これについては，3.4 節で述べる．

3.1.3 フィールド操作に関する型環境の正当性

型環境 γ 中の usage に現れる式 $\mathbf{w}(a,\gamma_w)$，$\mathbf{r}(a,\gamma_r)$ について，フィールド a に対する $\mathbf{w}(a,\gamma_w)$ と $\mathbf{r}(a,\gamma_r)$ の整合がとれているとき，γ はフィールド a の読み書きに関して正当であるといい，$rel(\gamma,a)$ と書く．ここでの整合性とは，直観的には，書込んだ値の usage は，次の書込みまでに行われる 0 回以上の読出しによって読出され使い尽くされる性質である．$rel(\gamma,a)$ の形式的な定義は以下で与える．

定義 15 フィールド名と基本型の組に対する型環境の組の遷移関係 $(\gamma,\gamma_{rst}) \xrightarrow{a}_{rel} (\gamma',\gamma'_{rst})$ は以下を満たす最小の関係である．

$$\frac{}{(x:\sigma/m;\gamma,\gamma_{rst}) \xrightarrow{a}_{rel} (\gamma,\gamma_{rst})}$$

$$\frac{\gamma_{rst} \leq \emptyset}{(x:\sigma/\mathbf{w}(a,\gamma_w);\gamma,\gamma_{rst}) \xrightarrow{a}_{rel} (\gamma,\gamma_w)}$$

$$\frac{\gamma_{rst} \leq \Diamond\gamma_r;\gamma'_{rst}}{(x:\sigma/\mathbf{r}(a,\gamma_r);\gamma,\gamma_{rst}) \xrightarrow{a}_{rel} (\gamma,\gamma'_{rst})}$$

$$\frac{b \neq a}{(x:\sigma/\mathbf{op}(b,\gamma);\gamma,\gamma_{rst}) \xrightarrow{a}_{rel} (\gamma,\gamma_{rst})} \quad (\mathbf{op}=\mathbf{w},\mathbf{r})$$

$$\frac{\gamma_1 \leq \gamma'_1 \qquad (\gamma'_1,\gamma_{rst}) \xrightarrow{a}_{rel} (\gamma_2,\gamma'_{rst})}{(\gamma_1,\gamma_{rst}) \xrightarrow{a}_{rel} (\gamma_2,\gamma'_{rst})}$$

上の規則中の (γ,γ_{rst}) について，γ は正当性を現在検証中の型環境を表し，γ_{rst} はフィールドに書込まれた値の今後読出されるべき型環境を表す．

定義 16 以下が成り立つとき，型環境 γ はフィールド a の読み書きに関して正当であるといい，$rel(\gamma,a)$ と書く．

$$\forall\gamma',\gamma'_{rst}.\ \ (\gamma,\emptyset) \xrightarrow{a}_{rel}{}^{*} (\gamma',\gamma'_{rst}) \stackrel{a}{\not\to}_{rel} \implies \gamma' \leq \emptyset \wedge \gamma'_{rst} \leq \emptyset$$

例 17 以下は，フィールドの読み書きに関して正当でない型環境と正当な型環境の例である．

- 正当でない型環境
  $x:C/\mathbf{w}(a,s_0:F/open);x:\mathbf{Int}$
  $y:C/\mathbf{w}(a,s_0:F/open);y:C/\mathbf{r}(a,s_0:\mathbf{Int})$
  $z:C/\mathbf{w}(a,s_0:F/open);z:C/\mathbf{r}(a,s_0:F/close)$
  $x:C/\mathbf{w}(a,s_0:F/read);x:C/\mathbf{w}(a,s_0:F/write)$
- 正当な型環境
  $x:C/\mathbf{w}(a,s_0:F/\mathbf{0});x:C/\mathbf{w}(a,s_0:F/open);$
  $x:C/\mathbf{r}(a,s_0:F/open)$
  $y:C/\mathbf{w}(a,s_0:F/open;s_0:F/close);$
  $y:C/\mathbf{r}(a,s_0:F/open;s_0:F/close)$

クラス σ のフィールドの読み書きに関する型環境の正当性 $rel(\gamma,\sigma)$ は以下で定義される．

定義 18 プログラム Q のもとでクラス σ に対し型環境 γ が次を満たすとき，γ はクラス σ のフィールドの読み書きに関して正当であるといい，$rel(\gamma,\sigma)$ と書く．

$$base(\gamma)=\sigma \wedge \forall a \in Fld(Q(\sigma)).rel(\gamma,a)$$

プログラム中の各オブジェクトに対する型環境は，そのオブジェクトが属するクラスのフィールドの読み書きに関して正当でなければならない．

3.2 型判定規則

本節では，プログラムの型判定を定義する．メソッドの型判定規則から順に定義する．

型環境表とは，A から型環境の集合への部分関数であり，メタ変数 Γ を用いて表す．また，メソッド表とは，絶対メソッド名の集合から型環境の組 (γ_p,γ_r) の集合への部分関数であり，メタ変数 Δ を用いて表す．

定義 19 $\Delta,\Gamma,\sigma.m,j \vdash Q$ は図 2 の規則を満たす最小の関係である．

図 2 中では，$\Gamma(j)$ を Γ_j と記している．判定 $\Delta,\Gamma,\sigma.m,j \vdash Q$ は，メソッド本体 $Q(\sigma.m)$ の命令アドレス j(およびそれ以降) の実行において局所変数およびスタックが $\Gamma(j)$ に従って用いられることを表す．メソッド表 Δ は，命令`invoke` $\sigma.m$ の実行時に，メソッド $\sigma.m$ の引数および返り値の使われ方を表す型環境を与えるものである．$\sigma.m$ は型付けを行うメソッドの絶対メソッド名である．

以下は各規則の説明である．

規則 (INC):1 行目は型付けしようとしているアドレスの命令がincであることを表している．2 行目は，次の命令が存在することを述べている．inc 命令はスタックトップの整数をインクリメントするが，よって，アドレス j ではスタックトップの値が整数として使用されるだけで，それ以外の局所変数およびスタック変数の値はアドレス $j+1$ 以降で使われる．これらの事柄を表しているのが 3，4 行目である．規則 (PUSH)，(POP) についても同様である．

規則 (IF): if j' 命令は，スタックトップの整数値に応じてアドレス $j+1$ あるいは j' に分岐する．よって，アドレス j ではスタックトップの値は整数として使用されるだけで，それ以外の局所変数およびスタック変数の値はアドレス $j+1$ 以降あるいはアドレス j' 以降で使われる．これらを表しているのが，3 行目である．

規則 (LOAD): load f_n 命令は，局所変数 f_n の値をスタックトップにのせるだけで値の使用は行ない．すなわち，アドレス $j+1$ でのスタックトップの値はアドレス j での f_n の値となり，また各変数の値はアドレス $j+1$ 以降で使われる．3 行目がこれらの事柄を表している．規則 (STORE)，(DUP) についても同様である．

規則 (PUTFIELD): putfield $\sigma'.a$ 命令は，アドレス j でのスタックの 2 番目 (s_1) の値の指すオブジェクトのフィールド a に書込みを行うので，s_1 の値の型の usage 部分は $\mathbf{w}(a, \gamma)$ となる．また，スタックトップにあった値はアドレス $j+1$ 以降で使われることになるが，putfield により値がエスケープしてしまい今後正確にいつ使用されるか分からなくなる (アドレス $j+1$ 以降，としか知り得ない)．そのため，その値の使われ方を表す型環境に $\diamond$ が付けられる．4，5 行目がこれらの事柄を表している．3 行目は，書込む値の基本型とフィールドの型の一致を表す．

規則 (GETFIELD): getfield $\sigma'.a$ 命令は，スタックトップの値が指すオブジェクトのフィールド a の値をスタックトップにのせる．すなわち，アドレス j でのスタックトップの値の型の usage 部分は $\mathbf{r}(a, \gamma)$ となる．また，読み出された値はアドレス $j+1$ 以降で使われる．4 行目がこれらの情報を表している．3 行目は，読み出す値の基本型とフィールドの型の一致を表す．

規則 (NEW): new σ' 命令は，クラス σ' のオブジェクトを生成し，それへの参照をスタックトップにのせる．よって，アドレス j では値の使用は行われず，また生成したオブジェクトはアドレス $j+1$ 以降で使用される．3，4 行目がこれを表している．生成したオブジェクトのアドレス $j+1$ 以降での使用は正しいものでなければならないが，それを表しているのが 5，6 行目である．5 行目はフィールドの読み書きに関する正当性を保証し，6 行目は対応する使用集合 $\Phi_{\sigma'}$ に対し妥当性を保証する．

規則 (INVOKE): invoke $\sigma'.m'$ 命令は，スタックトップから n 個の値を引数とし，$n+1$ 番目の値の指す σ' オブジェクトに対してメソッド m' を呼ぶ．アドレス j においてメソッド呼び出しが起るのでスタックの $n+1$ 番目の値が m' という使用をされる．また，引数として渡した値はメソッド内で γ_p に従って使用される．5 行目がこれらの事柄を表している．6 行目は，メソッドの返り値がアドレス $j+1$ においてスタックトップにのっており，アドレス $j+1$ 以降で使用されることを述べている．なお，型環境の関係式中の $[+_n]$ について，invoke により $n+1$ 個の値 (引数) がスタックから取り除かれ 1 個の値 (返り値) が置かれるためこのようになる．

規則 (RETURN): return は，メソッドの実行を終了するので次の命令の存在は要求しない．また，2 行目で返り値の型環境に $\diamond$ がついているが，これは putfield と同様の理由からである．すなわち，値がエスケープしてしまい正確な使用の順序が分からなくためである．

定義 20 (メソッドの型判定)

$$\frac{\begin{array}{c}\Delta(\sigma.m) = (\gamma_p, \gamma_r) \\ \mathit{numargs}(\sigma.m) = n \\ \gamma_p \leq [s_{n-1-i}/f_i]\Gamma_1 \quad (i = 0, \ldots, n-1) \\ SV \cap \mathit{dom}(\Gamma_1) = \emptyset \\ \mathit{base}(\gamma_r) = \mathit{rettyp}(\sigma.m) \\ \mathit{base}(\Gamma_1 \downarrow_{f_i}) = \mathit{argtyp}(\sigma.m, i) \quad (i = 0, \ldots, n-1) \\ \forall j \in \mathit{dom}(Q(\sigma.m)).\ \Delta, \Gamma, \sigma.m, j \vdash Q\end{array}}{\Delta, \Gamma, \sigma.m \vdash Q}$$

アドレス 1 がメソッド本体の最初の命令アドレスであるとする．3 行目は，γ_p は，Γ_1 のスタック変数を，添字を逆にした局所変数で置換した型環境であることを表し，すなわちメソッドの実行がメソッド表が与える型環境に従って始まることを表す．4 行目は，$\Gamma(1)$ にスタック変数が含まれていない，すなわちメソッドの開始時点でスタックが空であることを表す．5 行目は，メソッドの返り値の基本型が $\mathit{rettyp}(\sigma.m)$ であること

を表し，6 行目はメソッドの i 番目の引数の基本型が $argtyp(\sigma.m, i)$ であることを表す．7 行目は．メソッド本体の各アドレスでメソッドが型付けされていることを表す．

定義 21 (プログラムの型判定)

$$\frac{\begin{array}{c}\forall \sigma \in dom(Q).\ \forall m \in dom(Mtd(Q(\sigma))).\\ \exists \Gamma.\ \ \Delta, \Gamma, \sigma.m \vdash Q\\ \Delta(Q_{main}) = (\gamma_p, s_0 : \mathbf{Top})\end{array}}{\Delta \vdash Q}$$

1，2 行目は，プログラム中のすべてのメソッドが型付けされることを表す．3 行目は，メインメソッドの返り値が **Top** として使われる，すなわちプログラムの終了時にすべてのオブジェクトが正しく使い切られていることを表す．

プログラム Q に対し $\Delta \vdash Q$ を満たすメソッド表 Δ が存在するとき，Q は型付け可能であるという．

3.3 型システムの健全性

健全性が保証される見込みであり，次の命題が成り立つと思われる．

命題 22 (型システムの健全性) プログラム Q が型付け可能であるならば，Q はオブジェクト使用に関して安全である．

厳密な証明は得られていないが，小林らの資源使用解析 [7] と同様の手法で証明できる見込みである．

3.4 型推論アルゴリズム

前節の型システムの健全性が成り立つと仮定すると，プログラム Q の実行において各オブジェクトが正しく使用されることを保証するには，Q に対して型推論を行い定義 21 の関係 $\Delta \vdash Q$ を満たす Δ が存在することを示せば十分である．型推論は，[6, 7] の制約解消に基づく型推論と同様に行うことができる．すなわち，1). 型判定規則に基付き，各メソッド本体のアドレスから型環境に関する制約を生成し，2). 生成された制約を簡約し，充足可能性を調べる．アルゴリズムの詳細は論文 [5] に譲り，以下では例を通して型推論の概略を述べる．

```
1:  new File             ; File オブジェクト生成
2:  dup                  ; 参照を複製
3:  invoke File.open     ; メソッド open 呼び出し
4:  load f1              ; 局所変数 f1 の値をロード
5:  if 8                 ; 0 以外ならアドレス 8 へ
6:  invoke File.close    ; メソッド close 呼び出し
7:  return               ; メソッド終了
8:  invoke File.close    ; メソッド close 呼び出し
9:  return               ; メソッド終了
```

上はメソッド本体の例である．ただし，`File` クラスのメソッド `open`，`close` は引数も返り値もないとする．また，$\Phi_{File} = \{open \cdot close\}$ とする．

上の例では，アドレス 1 で `File` クラスのオブジェクトを生成し，そのオブジェクトに対しアドレス 3 でメソッド `open` を呼び出した後アドレス 5 で分岐し，どちらの分岐でもメソッド `close` を呼んだ後メソッド本体を終了している．よって，`File` オブジェクトは正しく使用されている．また，アドレス 5 の命令が誤って `if 9` となっていた場合，分岐によってはメソッド `close` が呼ばれず，`File` オブジェクトは正しく使用されない．以下で，我々の型システムがどのように両者の判定を行うかを示す．

アドレス j を型付けする型環境を α_j とすると，図 2 の型付け規則から次の制約を得る．ただし，`open`，`close` メソッドには返り値がないため，規則 (INVOKE) および (RETURN) による制約は実際のものとは若干異なる．

1.	$\alpha_1 \leq [-_1]\alpha_2$	(アドレス 1 より)
2.	$\alpha_F \leq \alpha_2 \downarrow_{s_0}$	(アドレス 1 より)
3.	$rel(\alpha_F, File)$	(アドレス 1 より)
4.	$[\![\alpha_F]\!] \leq \Phi_{File}$	(アドレス 1 より)
5.	$\alpha_2 \leq [-_1]([s_1/s_0]\alpha_3)$	(アドレス 2 より)
6.	$\alpha_3 \leq s_0 : File/open; [+_1]\alpha_4$	(アドレス 3 より)
7.	$\alpha_4 \leq [-_1]([f_1/s_0]\alpha_5)$	(アドレス 4 より)
8.	$\alpha_5 \leq s_0 : \mathbf{Int}; [+_1](\alpha_6 \& \alpha_8)$	(アドレス 5 より)
9.	$\alpha_6 \leq s_0 : File/close; \alpha_7$	(アドレス 6 より)
10.	$\alpha_7 \leq \emptyset$	(アドレス 7 より)
11.	$\alpha_8 \leq s_0 : File/close; \alpha_9$	(アドレス 8 より)
12.	$\alpha_9 \leq \emptyset$	(アドレス 9 より)

これらの制約より，$\alpha_1 \leq f_1 : \mathbf{Int}$，$\alpha_2 \leq s_0 : File/open; y : \mathbf{Int}; s_0 : File/close$ が求まるが，α_1 は定義 18 の 4 行目を満たす．α_2 について，$\alpha_2 \downarrow_{s_0} = s_0 : File/open; s_0 : File/close$ であるから，制約 2，3，4 より制約

$$\begin{array}{l} rel(s_0 : File/open; s_0 : File/close,\ File)\\ [\![s_0 : File/open; s_0 : File/close]\!] \subseteq \Phi_{File}\end{array}$$

を得る．1 つ目の制約は充足され，また，$\Phi_{File} = \{open \cdot close\}$ であるから 2 つ目の制約も充足される．従って，

上の例では File オブジェクトの使用は正しいと判定される．

一方，アドレス 5 が `if 9` であったとすると，$\alpha_2 \downarrow_{s_0} = s_0 : \mathit{File}/\mathit{open}; (s_0 : \mathit{File}/\mathit{close} \& s_0 : \mathit{File}/\mathbf{0})$ となり，制約 $[\![s_0 : \mathit{File}/\mathit{open}; (s_0 : \mathit{File}/\mathit{close} \& s_0 : \mathit{File}/\mathbf{0})]\!] \subseteq \Phi_{\mathit{File}}$ を得るがこれは充足不可能である．よって，File オブジェクトの使用は正しくないと判定される．

制限　我々の型推論アルゴリズムは，現在次の 2 点において制限がある．

1. $\mathit{rel}(\gamma, \sigma)$ の制約を解消する一般的方法は未解決．
2. $[\![\gamma]\!] \subseteq \Phi$ の判定は一般には決定不能．

1 について，γ は一般には無限の状態を取り得るが，現在のところ制約 $\mathit{rel}(\gamma, \sigma)$ を有限の制約集合に変換できていない．今後可能になるかもしれないし，そうでない場合は近似が必要となる．

2 の制限は，型環境は構成子 $\mu\alpha.$ と $\diamond$ を持つため文脈自由文法以上の表現力を有するが (例：$\mu\alpha.(\diamond(x : \mathbf{Int}); \alpha)$)，文脈自由言語の包含関係の判定問題は決定不能であることから生じる．本論文では $[\![\gamma]\!] \subseteq \Phi$ を判定するアルゴリズムの存在を仮定している．Φ を正則言語に限れば $[\![\gamma]\!]$ を文脈自由言語あるいは正則言語に近似することにより近似解を求めることは可能と思われるが，これは今後の課題である．

4　関連研究

序章で述べた本研究の型システムにおける 3 つのアイデアは，五十嵐と小林による資源使用解析 [7] および π 計算における一般型システム [6] から取り入れた．[7] では，資源生成，アクセスのプリミティヴを追加した λ 計算に対して，資源使用の解析を行う型システムを与えている．資源および関数の型に使用に関する情報を付加するが，我々の usage はこれにならったものである．なお，本研究は別名，ヒープも扱っており，彼らの型システムの拡張になっている．[6] の型システムでは，型環境は各チャネルについて送受信する値の型情報および送信，受信のどちらを行うかの情報だけでなく，送受信を行なう順序や送受信される値の使われ方の情報も保持する．本研究での型環境の拡張は，チャネルの送受信の順序の解析にならったものである．また，`putfield` と `getfield` のオブジェクトの使用としての扱いは，π 計算でのチャネルに対する送信，受信からの類推である．

DeLine と Fahndrich による Vault 言語 [1] も，型システムを利用して資源の使用解析を行う．Vault 言語では，資源に対してその状態を表す key が与えられ，key の状態によりその時点で許可される資源の操作が決まる．型システムにより各資源の key を追跡することで使用解析を行なうが，この点において key は我々の型システムの usage に相当する．主な相違点として，我々の型システムは型推論を行うのに対し，Vault 言語ではプログラマが明示的に key を書く必要がある．また，オートマトンにより資源の状態を表す点も我々の型システムと異なる．

Puntigam と Peter の型システム [11] は，受理可能なメソッドが動的に変化する並行オブジェクトに対して，受理不可能なメソッド呼び出しが行われないことを保証する．オブジェクトへの参照を通してメソッド呼び出しを行うが，参照に実行可能なメソッドの列を付加することで解析を行う．彼らの型システムでは，参照の複製時に実行可能なメソッドの列が新旧両参照で分配され，それらの順序情報は失われるが，我々の型システムでは別名間のメソッド呼び出しの順序も解析可能である．また，本研究ではヒープを扱っている点も相違点である．

動的に計算資源のアクセス制御を行うものとして，Monitor に対する Path 式 [3] や Schneider のセキュリティ・オートマトン [12] がある．[3] では，Monitor へのアクセスが，付加された正則表現風の式で記述されたアクセス順序に従って実際に行われることを動的に検証する．また，[12] は，資源の状態を表すオートマトンの状態ごとに実行可能なメソッドの集合を定義し，メソッド呼び出しごとにオートマトンを状態遷移させることによりアクセス制御を行う．これらの研究の主眼は資源へのアクセス順序の記述および動的検査であるのに対し，本研究の主眼はプログラムが実際にその記述に従うことを静的に解析することである．また，Walker の型システム [15] は，セキュリティ・オートマトンの無駄なチェックを省くが，すべての検査を静的に行うわけではなく，本研究の型システムとは利用目的が異なる．

本研究の型システムは，Stata と Abadi の型システム [13] の拡張として定義した．彼らの型システムおよび萩谷と戸澤の型システム [4] はサブルーチンを扱うが，我々の型システムでもサブルーチンを扱うよう変更することは容易と思われる．

JVML に対する他の型システムとして，Freund らの初期化解析 [2]，岩間と小林のロック解析 [16] がある．初期化メソッドやロックの取得，解放もメソッド呼び出しの一つとみなすことにより，我々の型システムはこれらの解析も包含する．

5 結論

我々は，Java バイトコードにおいてオブジェクトの使用解析を行う型システムを提案した．これにより，プログラムの実行において各オブジェクトが想定された順序に従って使われることが静的に保証される．

今後の課題としては，まず型推論における制約解消の近似解法の考案がある．その上で，型推論アルゴリズムを実装し現実的なプログラムに対する我々の型システムの実用性を検証する予定である．また，我々の型システムをより現実的なものにするため，型システムの拡張も重要な課題である．例外 (`athrow`) およびサブルーチン (`jsr`) 命令は現在対象外であるが，これらの導入は容易であると思われる．JVML 本来の部分型関係である部分クラス関係も本研究の部分型関係に組み込む必要がある．

参考文献

[1] Robert DeLine and Manuel Fahndrich. Enforcing high-level protocols in low-level software. In *Proceedings of ACM SIGPLAN Conference on Programming Languages Design and Implementation*, pp. 59–69, 2001.

[2] Stephen N. Freund and John C. Mitchell. A type system for object initialization in the Java bytecode language. In *Proceedings of the conference on Object-oriented programming, systems, languages, and applications*, pp. 310–328, October 1998.

[3] A. N. Habermann and R. H. Campbell. The specification of process synchronization by path expressions. In *Proceedingsof an International Symposium*, Vol. 16 of *Lecture Notes in Computer Science*, pp. 89–102. Springer-Verlag, April 1974.

[4] Masami Hagiya and Akihiko Tozawa. On a new method for dataflow analysis of Java virtual machine subroutines. In *Proceedings of Static Analysis Symposium*, 1998.

[5] Nobuyuki Hamanaka. A type system for object usage analysis in Java bytecode. Master's thesis, University of Tokyo, February 2002. To appear at `http://www.yl.is.s.u-tokyo.ac.jp/~hamanaka/pub/`.

[6] Atsushi Igarashi and Naoki Kobayashi. A generic type system for the pi-calculus. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages*, pp. 128–141, January 2001.

[7] Atsushi Igarashi and Naoki Kobayashi. Resource usage analysis. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages*, pp. 331–342, 2002.

[8] Naoki Kobayashi. A type system for lock-free processes. *Information and Computation*, to appear. A preliminary version appeared in Proceedings of TCS2000, Springer LNCS 1872, pp.365-389, under the title "Type Systems for Concurrent Processes: From Deadlock-Freedom to Livelock-Freedom, Time-Boundedness".

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

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

[11] Franz Puntigam and Christof Peter. Changeable interfaces and promised messages for concurrent components. In *Proceedingsof Symposium on Applied Computing (SAC'99)*, pp. 141–145, 1999.

[12] Fred B. Schneider. Enforceable security policies. Technical Report TR98-1664, Cornell University, 1998.

[13] Raymie Stata and Martin Abadi. A type system for Java bytecode subroutines. *ACM Transactions on Programming Languages and Systems*, Vol. 21, No. 1, pp. 90–137, 1999.

[14] Eijiro Sumii and Naoki Kobayashi. A generalized deadlock-free process calculus. In *Proceedings of Workshop on High-Level Concurrent Language (HLCL'98)*, Vol. 16(3) of *Electronic Notes in Theoretical Computer Science*, pp. 55–77, 1998.

[15] David Walker. A type system for expressive security policies. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages*, pp. 254–267, January 2000.

[16] 岩間太, 小林直樹. JVM におけるロックの整合性検証のための新しい型システム. 日本ソフトウェア科学会第 18 回大会 論文集, 2001.

$$\frac{Q(\sigma.m)(pc) = \texttt{inc}}{Q \vdash (\langle \sigma.m, pc, f, n \cdot s\rangle \cdot FS, H) \rightarrow (\langle \sigma.m, pc+1, f, (n+1) \cdot s\rangle \cdot FS, H)} \quad \text{(E-Inc)}$$

$$\frac{Q(\sigma.m)(pc) = \texttt{push0}}{Q \vdash (\langle \sigma.m, pc, f, s\rangle \cdot FS, H) \rightarrow (\langle \sigma.m, pc+1, f, 0 \cdot s\rangle \cdot FS, H)} \quad \text{(E-Push)}$$

$$\frac{Q(\sigma.m)(pc) = \texttt{pop}}{Q \vdash (\langle \sigma.m, pc, f, v \cdot s\rangle \cdot FS, H) \rightarrow (\langle \sigma.m, pc+1, f, s\rangle \cdot FS, H)} \quad \text{(E-Pop)}$$

$$\frac{Q(\sigma.m)(pc) = \texttt{if}\ j}{Q \vdash (\langle \sigma.m, pc, f, 0 \cdot s\rangle \cdot FS, H) \rightarrow (\langle \sigma.m, pc+1, f, s\rangle \cdot FS, H)} \quad \text{(E-If0)}$$

$$\frac{Q(\sigma.m)(pc) = \texttt{if}\ j \qquad n \neq 0}{Q \vdash (\langle \sigma.m, pc, f, n \cdot s\rangle \cdot FS, H) \rightarrow (\langle \sigma.m, j, f, s\rangle \cdot FS, H)} \quad \text{(E-Ifn)}$$

$$\frac{Q(\sigma.m)(pc) = \texttt{load}\ f_n}{Q \vdash (\langle \sigma.m, pc, f, s\rangle \cdot FS, H) \rightarrow (\langle \sigma.m, pc+1, f, f(f_n) \cdot s\rangle \cdot FS, H)} \quad \text{(E-Load)}$$

$$\frac{Q(\sigma.m)(pc) = \texttt{store}\ f_n}{Q \vdash (\langle \sigma.m, pc, f, v \cdot s\rangle \cdot FS, H) \rightarrow (\langle \sigma.m, pc+1, f[f_n \mapsto v], s\rangle \cdot FS, H)} \quad \text{(E-Store)}$$

$$\frac{Q(\sigma.m)(pc) = \texttt{dup}}{Q \vdash (\langle \sigma.m, pc, f, v \cdot s\rangle \cdot FS, H) \rightarrow (\langle \sigma.m, pc+1, f, v \cdot v \cdot s\rangle \cdot FS, H)} \quad \text{(E-Dup)}$$

$$\frac{Q(\sigma.m)(pc) = \texttt{putfield}\ \sigma'.a \qquad H(o) = (\rho, \Phi)}{Q \vdash (\langle \sigma.m, pc, f, v \cdot o \cdot s\rangle \cdot FS, H) \rightarrow (\langle \sigma.m, pc+1, f, s\rangle \cdot FS, H[o \mapsto (\rho[a \mapsto v], \Phi)])} \quad \text{(E-Putfield)}$$

$$\frac{Q(\sigma.m)(pc) = \texttt{getfield}\ \sigma'.a \qquad H(o) = (\rho, \Phi) \qquad \rho(a) = v}{Q \vdash (\langle \sigma.m, pc, f, o \cdot s\rangle \cdot FS, H) \rightarrow (\langle \sigma.m, pc+1, f, v \cdot s\rangle \cdot FS, H)} \quad \text{(E-Getfield)}$$

$$\frac{Q(\sigma.m)(pc) = \texttt{new}\ \sigma' \quad o \notin Dom(H) \quad o \neq null \quad dom(\rho) = dom(Fld(Q(\sigma'))) \quad \forall a \in dom(\rho).\rho(a) = 0 \text{ or } null}{Q \vdash (\langle \sigma.m, pc, f, s\rangle \cdot FS, H) \rightarrow (\langle \sigma.m, pc+1, f, o \cdot s\rangle \cdot FS, H[o \mapsto (\rho, \Phi_{\sigma'})])} \quad \text{(E-New)}$$

$$\frac{Q(\sigma.m)(pc) = \texttt{invoke}\ \sigma'.m' \qquad H(o) = (\rho, \Phi) \qquad \Phi^{-m'} \neq \emptyset}{Q \vdash (\langle \sigma.m, pc, f, \tilde{v} \cdot o \cdot s\rangle \cdot FS, H) \rightarrow (\langle \sigma'.m', 1, f', \epsilon\rangle \cdot \langle \sigma.m, pc+1, f, s\rangle \cdot FS, H[o \mapsto (\rho, \Phi^{-m'})])} \quad \text{(E-Invoke)}$$

$$\frac{Q(\sigma.m)(pc) = \texttt{return}}{Q \vdash (\langle \sigma.m, pc, f, v \cdot s\rangle \cdot \langle \sigma'.m', pc', f', s'\rangle \cdot FS, H) \rightarrow (\langle \sigma'.m', pc', f', v \cdot s'\rangle \cdot FS, H)} \quad \text{(E-Return)}$$

図 1: マシンステートの遷移規則

(INC)
$$\frac{Q(\sigma.m)(j) = \texttt{inc} \quad j+1 \in \mathit{dom}(Q(\sigma.m)) \quad \Gamma_j \leq \Gamma_{j+1} \quad \Gamma_j \downarrow_{s_0} \leq s_0 : \mathbf{Int}}{\Delta, \Gamma, \sigma.m, j \vdash Q}$$

(PUSH)
$$\frac{Q(\sigma.m)(j) = \texttt{push0} \quad j+1 \in \mathit{dom}(Q(\sigma.m)) \quad \Gamma_j \leq [-_1]\Gamma_{j+1} \quad s_0 : \mathbf{Int} \leq \Gamma_{j+1} \downarrow_{s_0}}{\Delta, \Gamma, \sigma.m, j \vdash Q}$$

(POP)
$$\frac{Q(\sigma.m)(j) = \texttt{pop} \quad j+1 \in \mathit{dom}(Q(\sigma.m)) \quad \Gamma_j \leq s_0 : \mathbf{Top}; [+_1]\Gamma_{j+1}}{\Delta, \Gamma, \sigma.m, j \vdash Q}$$

(IF)
$$\frac{Q(\sigma.m)(j) = \texttt{if}\ j' \quad j+1, j' \in \mathit{dom}(Q(\sigma.m)) \quad \Gamma_j \leq s_0 : \mathbf{Int}; [+_1](\Gamma_{j+1} \& \Gamma_{j'})}{\Delta, \Gamma, \sigma.m, j \vdash Q}$$

(LOAD)
$$\frac{Q(\sigma.m)(j) = \texttt{load}\ f_n \quad j+1 \in \mathit{dom}(Q(\sigma.m)) \quad \Gamma_j \leq [-_1]([f_n/s_0]\Gamma_{j+1})}{\Delta, \Gamma, \sigma.m, j \vdash Q}$$

(STORE)
$$\frac{Q(\sigma.m)(j) = \texttt{store}\ f_n \quad j+1 \in \mathit{dom}(Q(\sigma.m)) \quad \Gamma_j \leq [s_0/f_n]([+_1]\Gamma_{j+1})}{\Delta, \Gamma, \sigma.m, j \vdash Q}$$

(DUP)
$$\frac{Q(\sigma.m)(j) = \texttt{dup} \quad j+1 \in \mathit{dom}(Q(\sigma.m)) \quad \Gamma_j \leq [-_1]([s_1/s_0]\Gamma_{j+1})}{\Delta, \Gamma, \sigma.m, j \vdash Q}$$

(PUTFIELD)
$$\frac{Q(\sigma.m)(j) = \texttt{putfield}\ \sigma'.a \quad j+1 \in \mathit{dom}(Q(\sigma.m)) \quad \mathit{base}(\gamma) = Q(\sigma'.a) \quad \Gamma_j \leq s_1 : \sigma'/\mathbf{w}(a,\gamma); \Diamond\gamma; [+_2]\Gamma_{j+1} \quad \mathit{dom}(\gamma) = \{s_0\}}{\Delta, \Gamma, \sigma.m, j \vdash Q}$$

(GETFIELD)
$$\frac{Q(\sigma.m)(j) = \texttt{getfield}\ \sigma'.a \quad j+1 \in \mathit{dom}(Q(\sigma.m)) \quad \mathit{base}(\Gamma_{j+1} \downarrow_{s_0}) = Q(\sigma'.a) \quad \Gamma_j \leq s_0 : \sigma'/\mathbf{r}(a, \Gamma_{j+1} \downarrow_{s_0}); \Gamma_{j+1} \uparrow_{s_0}}{\Delta, \Gamma, \sigma.m, j \vdash Q}$$

(NEW)
$$\frac{Q(\sigma.m)(j) = \texttt{new}\ \sigma' \quad j+1 \in \mathit{dom}(Q(\sigma.m)) \quad \Gamma_j \leq [-_1]\Gamma_{j+1} \quad \gamma \leq \Gamma_{j+1} \downarrow_{s_0} \quad \mathit{rel}(\gamma, \sigma') \quad [\![\gamma]\!] \subseteq \Phi_{\sigma'}}{\Delta, \Gamma, \sigma.m, j \vdash Q}$$

(INVOKE)
$$\frac{Q(\sigma.m)(j) = \texttt{invoke}\ \sigma'.m' \quad j+1 \in \mathit{dom}(Q(\sigma.m)) \quad \Delta(\sigma'.m') = (\gamma_p, \gamma_r) \quad n = \mathit{numargs}(\sigma'.m') \quad \Gamma_j \leq s_n : \sigma'/m'; \gamma_p; [+_n](\Gamma_{j+1} \uparrow_{s_0}) \quad \gamma_r \leq \Gamma_{j+1} \downarrow_{s_0}}{\Delta, \Gamma, \sigma.m, j \vdash Q}$$

(RETURN)
$$\frac{Q(\sigma.m)(j) = \texttt{return} \quad \Delta(\sigma.m) = (\gamma_p, \gamma_r) \quad \Gamma_j \leq \Diamond\gamma_r \quad \mathit{dom}(\gamma_r) = \{s_0\}}{\Delta, \Gamma, \sigma.m, j \vdash Q}$$

図 2: 型判定規則