文脈依存資源使用解析のための型システム

仲井間 達也　　五十嵐 淳
京都大学 大学院情報学研究科
{nakaima,igarashi}@kuis.kyoto-u.ac.jp

小林 直樹
東北大学 大学院情報科学研究科
koba@ecei.tohoku.ac.jp

概 要

資源使用解析とは，ファイル・メモリなどの計算資源に対するプログラム実行中のアクセス順序が，与えられた仕様を満たすかどうかを判定するプログラム解析である．Igarashi, Kobayashi は，型付き λ 計算に資源のアクセス順序を表す使用法表現とプログラムの実行順序を反映した型付け規則を導入することで，型システムを用いた解析手法を与えた．しかし，彼等の手法では，変数のエイリアスが生じる場合や，関数の呼び出し先でのアクセスがある場合に解析精度が悪くなることがある．

本研究では，各アクセスがどういった関数呼び出しを経て起こるかといった，アクセスの文脈情報を表せるように拡張した使用法表現に基づいた型システムを形式化する．このような文脈依存性を取り入れることにより，アクセス順序をより正確に捉え, 解析精度を向上させることができる．

1　はじめに

ファイル・メモリなどの計算資源が正しく使われることの保証は, プログラムを検証する上で重要である．計算資源には，通常，その資源に応じたアクセスプリミティブがいくつか用意されており，プログラムはそれらを正しい順番で使うことが要求されている．例えば，ファイルは, 読み書き操作を何度か行った後，プログラムが終了するまでに閉じられること, 一旦閉じられたファイルに対しては読み書きの操作が行われないこと, が要求される．他にも, 共有資源にかけたロックが解放されること, 割り当てたメモリ領域は最終的に開放され, 開放した領域にアクセスしないこと, などが要求される．資源使用解析とは，計算資源に対するアクセス順序が，与えられた仕様を満たすかどうかを判定するプログラム解析であり, 近年盛んに研究されている [4, 7, 2, 1, 11].

Igarashi, Kobayashi は資源使用解析に対する型システムを用いた統一的な解析手法を提案している [4]. 彼等の型システムでは, 通常の型付き λ 計算の型に使用法表現と呼ばれる, 計算資源へのアクセス順序を表す情報が加えられており，型付け規則を工夫することで，型がついたプログラムは実際に資源に与えられた仕様に沿ってアクセスが発生することが保証される．資源使用解析は，このような型システムに対する型推論，つまり，資源に対する使用法表現の推論と仕様との包含関係の判定問題に帰着される．

本研究では，彼らの手法を拡張し，文脈依存な資源使用解析，つまり資源への各アクセスがどのような関数呼び出しを経て発生するかという文脈情報を利用する解析のための型システムを与える．この型システムでは，文脈情報を (実行時の) 関数呼び出しの系列で表し，資源の型には文脈情報付きのアクセス順序を表す拡張された使用法表現を付加する．例えば，次のような OCaml 風のプログラム断片を考える．

$$M \stackrel{\text{def}}{=} \texttt{let f x = read}^{l_1}\texttt{(x); write}^{l_2}\texttt{(y) in}\ \texttt{f @}^{l_3}\ \texttt{y; close}^{l_4}\texttt{(y)}$$

ここで，y はファイルを表す変数，read, write, close はそれぞれファイルの読み込み，書き込み，クローズのための関数，@ は関数適用であり，ファイルアクセス・関数適用にはラベルが付加されている．M を実行すると，y に対して，f の呼び出し内で読み込み・書き込みアクセス，次いで f からリターン後に，クローズ操作が行われる．このようなアクセス情報を，提案する型システムでは使用法表現 $l_3.l_1 \otimes l_3.l_2 \otimes l_4$ として表す．この $\otimes$ で繋がれた各ラベル列 (これを単位使用と呼ぶ) は，一度のアクセスがどこで呼び出さ

れた関数本体内のどのアクセスプリミティブで発生したかを表しており，例えば $l_3.l_1$ は l_3 での関数呼び出し内の l_1 でアクセスされることを意味する．$U_1 \otimes U_2$ は使用法表現 U_1 と U_2 で表される両方のアクセスが発生することを示している．逆に y の使用法 $l_3.l_1 \otimes l_3.l_2 \otimes l_4$ が与えられれば，y に対するアクセスについて次のような情報が得られる．まず，末尾のラベル l_1, l_2, l_4 から `read`, `write`, `close` が一回づつ起きることがわかる．さらに，プログラムの文面と言語の操作的意味から，l_1 が表す `read` が l_2 が表す `write` よりも先に実行されること，l_4 で表される `close` の前に l_3 の呼び出しが完了することがわかる．以上より，y に対するアクセスは `read`, `write`, `close` の順に一度づつ起きることがわかる．(単位使用同士はプログラムのラベル付けを見れば，どちらのアクセスが先に発生するかがわかるので，$\otimes$ はアクセス順序を指定しない，結合的・交換的なオペレータである．) ファイルの型は使用法表現を使って $(\mathbf{File}, U)$ と表され，U に従ってアクセスされるファイルを意味する．

このような拡張した型を反映して，型付け規則も変更される．例えば let 式のための型付け規則は，

$$\frac{\Gamma_1 \vdash M_1 : \tau_1 \qquad \Gamma_2, x : \tau_1 \vdash M_2 : \tau_2}{\Gamma_1 \otimes \Gamma_2 \vdash \texttt{let}\ x{=}M_1\ \texttt{in}\ M_2 : \tau_2}$$

となる．結論の型環境 $\Gamma_1 \otimes \Gamma_2$ は，Γ_1 と Γ_2 中の使用法表現を $\otimes$ で結合して得られる型環境を表す．例えば，$\Gamma_1 = y : (\mathbf{File}, l_3.l_2)$ かつ $\Gamma_2 = y : (\mathbf{File}, l_3.l_1 \otimes l_4)$ であれば，$\Gamma_1 \otimes \Gamma_2 = y : (\mathbf{File}, l_3.l_2 \otimes l_3.l_1 \otimes l_4)$ となる．

資源アクセスの文脈情報を捉えるためには関数型にも使用法表現を付加し，関数抽象・適用の型付け規則で文脈情報を適切に取り扱う必要がある．上の例での関数 `f` の型は，$((\mathbf{File}, l_1) \to \mathbf{unit}, l_3)$ と表される．l_3 が，この関数が (他の関数呼び出しを経由せずに) l_3 で適用されることを示している．また，引数型 $(\mathbf{File}, l_1)$ は，引数のファイルがこの関数本体内のどこでアクセスされるかを，関数本体からの相対的な位置で示している．関数抽象の型付け規則は，

$$\frac{\Gamma, x : \tau_1 \vdash M : \tau_2}{U[\Gamma] \vdash \lambda x.M : (\tau_1 \to \tau_2, U)}$$

となる[1]．$U[\Gamma]$ は Γ 中の使用法表現を U 内の各単位使用の数，つまり呼び出し回数分だけ複製し，それぞれに U 内の各単位使用を prefix として付加したような型環境を表している．これは U が関数の呼び出される回数と場所を示しているので，その本体でアクセスされる資源に文脈情報を付加することに相当している．例えば，$\lambda\mathtt{x}.\mathtt{read}^{\mathtt{l}_1}(\mathtt{x}); \mathtt{write}^{\mathtt{l}_2}(\mathtt{y})$ のための型判断は，

$$\frac{\begin{array}{c}\vdots \\ \mathtt{x} : (\mathbf{File}, l_1), \mathtt{y} : (\mathbf{File}, l_2) \vdash \mathtt{read}^{\mathtt{l}_1}(\mathtt{x}); \mathtt{write}^{\mathtt{l}_2}(\mathtt{y}) : \mathbf{unit}\end{array}}{\mathtt{y} : (\mathbf{File}, l_3.l_2) \vdash \lambda\mathtt{x}.\mathtt{read}^{\mathtt{l}_1}(\mathtt{x}); \mathtt{write}^{\mathtt{l}_2}(\mathtt{y}) : ((\mathbf{File}, l_1) \to \mathbf{unit}, l_3)}$$

と導出される．結論の y の使用法表現には l_3 が付いていることに注意．これに対応して，関数適用の型付け規則は，

$$\frac{\Gamma_1 \vdash M_1 : (\tau_1 \to \tau_2, l) \qquad \Gamma_2 \vdash M_2 : l[\tau_1]}{\Gamma_1 \otimes \Gamma_2 \vdash M_1\ @^l\ M_2 : \tau_2} \qquad \text{(T-App)}$$

のように与えられる．上の例からもわかるように，仮引数の型 τ_1 は，関数本体を基点とした相対的な文脈情報によるアクセス情報を表しているため，実引数の型は，仮引数の型に呼び出し位置のラベルを付加し $l[\tau_1]$ という形で表される．例えば，

$$\mathtt{y} : (\mathbf{File}, l_3.l_1), \mathtt{f} : ((\mathbf{File}, l_1) \to \mathbf{unit}, l_3) \vdash \mathtt{f}\ @^{l_3}\ \mathtt{y} : \mathbf{unit}$$

[1] これは，正確には τ_2 が資源や関数型でない場合である．τ_2 が資源・関数型である場合については後述する．

が導出される．これらを総合すると，M に対して，

$$\mathtt{y} : (\mathbf{File}, l_3.l_1 \otimes l_3.l_2 \otimes l_4) \vdash M : \mathbf{unit}$$

という判断が導出できる．単位使用のラベルとプログラムを見ると，l_3 での関数適用が l_4 でのアクセスより先に起こること，l_1 でのアクセスが l_2 でのアクセスより先に起こることがわかるので，最終的に y が，読み込み，書込み，クローズの順でアクセスされることがわかる．

ただし，資源を返すような関数に対しては，もう少し工夫が必要である．例えば，次のようなプログラムを考える．

```
let f x = read^{l1}(x); x in
  write^{l2}(f @^{l3} y); close^{l4}(y)
```

関数 f は，引数のファイルを読み込みそのファイルを返す．このような資源を返す場合の使用法表現として，式の返り値となることを示す ↑ という特別なラベルを導入し，関数 f は

$$\dfrac{\begin{array}{c}\vdots\\ \mathtt{x} : (\mathbf{File}, l_1 \otimes \uparrow.l_2) \vdash \mathtt{read}^{l_1}(\mathtt{x}); \mathtt{x} : (\mathbf{File}, \uparrow.l_2)\end{array}}{\vdash \lambda\mathtt{x}.\mathtt{read}^{l_1}(\mathtt{x}); \mathtt{x} : ((\mathbf{File}, l_1 \otimes \uparrow.l_2) \to (\mathbf{File}, l_2), l_3)}$$

と型付けされる．f の本体内部から見ると，引数 x は，l_1 で使われ，かつ，関数から返った文脈での l_2 で使われるため，$l_1 \otimes \uparrow.l_2$ という使用法表現が与えられ，本体式の型も，ファイルが返されることを示している．結論では，↑ を取り除くことによって，関数の返すファイルが，呼び出し元の文脈での l_2 で使われることを示している．関数呼び出し側では，実引数 y の使用法表現 $l_3[l_1 \otimes \uparrow.l_2]$ を $l_3.l_1 \otimes l_2$ と等しい，つまり，↑ を $l[U]$ をキャンセルする働きをすると考えることにより，正しいアクセス情報を得ることができる．

本論文では, このようなアイデアに基づく型システムを定式化し，その健全性を示す．自動的に解析を行うためには，型推論アルゴリズムと，資源に与えた仕様と，推論された使用法表現の表すアクセス列の包含関係を判定するアルゴリズムを与える必要があるが，それらは将来の課題となっている．以下，2 節で対象言語とその操作的意味論を与えた後，3 節で型システムを形式化し, 型システムの健全性を示す．4 節で関連研究について述べた後，5 節で結論・今後の課題を述べる．なお，紙面の都合上，定理の証明を省略するが，証明概略などを付録としたバージョンが http://www.sato.kuis.kyoto-u.ac.jp/~nakaima/ppl2008-long.pdf からダウンロード可能である．

2　対象言語

本節では，対象言語として，値呼び λ 計算に資源生成/アクセスのための構文を加えた $\lambda^{\mathcal{R}}$ を与える．この言語は基本的には [4] で与えられたものと同じだが，操作的意味論において，関数呼び出しの文脈情報を明示化するためにいくつかの変更が加えられている．

まず，生成される資源の仕様を表現するためのトレース集合を定義してから，項の定義，操作的意味論を与える．

2.1　トレース

$(\mathcal{L}, \sqsubseteq)$ をラベルを要素として持つ (可算無限) 半順序集合とする．ラベルは，資源アクセスの項や関数適用項に付加されて，アクセスや適用の場所を示すために使われる．ラベル上の半順序は，項にラベルを付加するにあたり，実行順序を反映するために使われる．

トレースとは，資源に対して行われるアクセスの順序を表すためのもので，ラベルとプログラムの終了を表す記号 $\downarrow$ の有限列で表現する．正確には，次の集合 $\mathcal{L}^{*,\downarrow}$ の要素を**トレース**と呼ぶ.

$$\mathcal{L}^{*,\downarrow} = \mathcal{L}^* \cup \{s\downarrow \mid s \in \mathcal{L}^*\}$$

ここで $\mathcal{L}^*$ は $\mathcal{L}^*$ の要素の長さ 0 以上の有限列を表す.

トレースは，プログラム実行中のある時点で資源がどのようにアクセスされたかの履歴を示している．トレース $l_1 \ldots l_n$ は資源が $l_1, \ldots, l_n$ でラベル付けされたアクセスによってこの順序でアクセスされたことを示し，トレース $l_1 \ldots l_n \downarrow$ は，プログラムの実行が $l_1, \ldots, l_n$ によるアクセスの発生後に終了したことを示す．資源があるトレース s に従って使用される時には，その資源は s の prefix に従っても使用される．

$\mathcal{L}^{*,\downarrow}$ の部分集合で prefix に関して閉じているもの Φ を**トレース集合**と呼ぶ. prefix に関して閉じているとは, 列 ss' が Φ の要素であるならば, その prefix s も Φ の要素であるときにいう．また，S の prefix に関する閉包を $S^\sharp$ で表す．トレース集合は資源に与えられる仕様と考えることができる．トレース集合を表すための記法は固定しないが，例ではしばしば正則表現を用いる．

例 2.1. 読み書き可能なファイルの仕様を表すトレース集合は，以下の正則表現と prefix 閉包で表される集合, $\Phi = ((R+W)^*C\downarrow)^\sharp$ で表すことができる．ここで，R, W, C はファイルの読み込み, 書き込み, クローズを表す正則表現で，与えられたプログラムから自動的に決めることができる．例えば,

$$\texttt{read}^{l_1}\texttt{(x); write}^{l_2}\texttt{(y); read}^{l_3}\texttt{(x); close}^{l_4}\texttt{(y)}$$

のようなプログラムの場合は $R = l_1 + l_3, W = l_2, C = l_4$ であり,

$$\Phi = \{l_1, l_3, l_2, l_1l_1, l_1l_2, l_2l_1, \ldots, l_1l_1l_4, l_1l_1l_4\downarrow, \ldots\}$$

となる.

2.2 項

次に，ラベルとトレース集合を使って，$\lambda^{\mathcal{R}}$ の項の定義を与える.

定義 2.2 (項, 値)**.** $\lambda^{\mathcal{R}}$ の項 M と値 v は以下の構文で定義される.

$$\begin{aligned} M &::= v \mid M_1 \mathbin{@^l} M_2 \mid \mathbf{acc}^l(M) \mid \mathbf{new}^\Phi() \mid \mathbf{if}\ M_1\ \mathbf{then}\ M_2\ \mathbf{else}\ M_3 \\ v &::= x \mid \mathbf{true} \mid \mathbf{false} \mid \mathbf{fun}(f, x, M) \end{aligned}$$

$\mathbf{fun}(f, x, M)$ は再帰関数であり，x が仮引数，f が再帰的に $\mathbf{fun}(f, x, M)$ を指す．M 中の f, x は束縛変数である．$M_1 \mathbin{@^l} M_2$ は関数適用であり，関数の呼び出し場所を示すためのラベルが付加されている．$\mathbf{new}^\Phi(), \mathbf{acc}^l(M)$ は，それぞれ資源の生成とアクセスを表すプリミティブであり，$\mathbf{new}^\Phi()$ はトレース集合 Φ で与えられる仕様を持つ資源を生成する．$\mathbf{acc}^l(M)$ は M を評価して得られた値の指す資源に対してアクセス l を行う．簡単のため，資源アクセスは真偽値のどちらか，**true** か **false** を (非決定的に) 返すこととする.

関数適用 $\mathbf{fun}(f, x, M_2) \mathbin{@^l} M_1$ を，ラベルを省略して，$\mathbf{let}\ x = M_1\ \mathbf{in}\ M_2$ と表記することがある．ただし, f は M_2 の自由変数ではない．さらに，x が M_2 の自由変数でないときには $M_1; M_2$ と書く．式中の複数の変数への値の capture-avoiding な同時代入を $[v_1/x_1, \cdots, v_n/x_n]M$ と書く.

ラベルの半順序関係には条件が必要である. 項に付いたラベルを使って文脈情報を表し, 文脈同士の順序はラベルの順序を利用するので, 得られた文脈情報と実行順序とがうまく対応してなければならない. 例えば, 次のようなプログラムの場合,

$$\texttt{close}^{l_1}\texttt{((read}^{l_2}\texttt{(y); } \lambda\ \texttt{x. write}^{l_3}\texttt{(y); x) @}^{l_4}\ \texttt{(read}^{l_5}\texttt{(y); y))}$$

実行すると $l_2, l_5, l_4.l_3, l_1$ という四つのアクセスが順に起こるので, プログラムに付けるラベルには $l_2 \sqsubset l_5 \sqsubset l_4 \sqsubset l_1$ という順序が付いているものを用いる. 関数適用 $M_1 \ @^l \ M_2$ なら, M_1, M_2, 関数呼び出しの順に実行されるので,

$$M_1\text{中のラベル} \sqsubset M_2\text{中のラベル} \sqsubset l$$

という条件が必要である. ただし, 「M 中のラベル」から, 関数抽象されている項に出現するものは除く. これは, 関数抽象の中にあるラベルが表すアクセスは, 関数が適用される文脈で起こるためである. この直観に従い, 項 M に対して, ラベルの集合 $Lab(M)$ と項に付くラベルの条件を定義する.

$$\begin{aligned}
Lab(v) &= \emptyset \\
Lab(\mathbf{new}^{\Phi}()) &= \emptyset \\
Lab(\mathbf{acc}^l(M)) &= \{l\} \cup Lab(M) \\
Lab(M_1 \ @^l \ M_2) &= \{l\} \cup Lab(M_1) \cup Lab(M_2) \\
Lab(\mathbf{if}\ M_1\ \mathbf{then}\ M_2\ \mathbf{else}\ M_3) &= Lab(M_1) \cup Lab(M_2) \cup Lab(M_3)
\end{aligned}$$

項 M に対するラベル付けは以下の条件を満たすものとする. ここで $l_1 \sqsubset l_2$ は $l_1 \sqsubseteq l_2$ かつ $l_1 \neq l_2$ を示す.

- M の任意の部分項 $\mathbf{acc}^l(M_1)$ に対し $\forall l_1 \in Lab(M_1).l_1 \sqsubset l$
- M の任意の部分項 $M_1 \ @^l \ M_2$ に対し $\forall l_1 \in Lab(M_1), \forall l_2 \in Lab(M_2).l_1 \sqsubset l_2 \wedge l_2 \sqsubset l$
- M の任意の部分項 $\mathbf{if}\ M_1\ \mathbf{then}\ M_2\ \mathbf{else}\ M_3$ に対し $\forall l_1 \in Lab(M_1), \forall l \in Lab(M_2) \cup Lab(M_3).l_1 \sqsubset l$

2.3 操作的意味論

次に $\lambda^{\mathcal{R}}$ の操作的意味論を与える. 操作的意味論は資源の状態を表すヒープと項の書き換え (簡約) で定義される.

定義 2.3 (ヒープ)**.** ヒープ H は変数の有限集合からトレース集合への写像である. $dom(H) = \{x_1, \ldots, x_n\}$ かつ $\forall x_i.H(x_i) = \Phi_i$ であるようなヒープを $\{x_1 \mapsto \Phi_1, \ldots, x_n \mapsto \Phi_n\}$ と表記する. ヒープ H_1, H_2 の合成 $H = H_1 \uplus H_2$ は $dom(H_1) \cap dom(H_2) = \emptyset$ のときに限り定義され, $dom(H) = dom(H_1) \cup dom(H_2), H(x) = H_i(x)(x \in dom(H_i))$ である.

簡約関係を定義する前に, 項に $\langle M \rangle_l$ を加えて拡張する. この項 $\langle M \rangle_l$ は, 計算の途中でのみ表れる中間的な項で, l でラベル付けされた関数適用での関数呼び出しが発生した状態を表現している.

定義 2.4 (拡張された項)**.** 対象言語の式 M の構文を以下のように拡張する. それに伴い, ラベルの集合の定義も拡張される.

$$\begin{aligned}
M &::= \cdots \mid \langle M \rangle_l \\
Lab(\langle M \rangle_l) &= \{l\} \cup Lab(M)
\end{aligned}$$

簡約関係は, 評価文脈を使って以下のように定義される.

定義 2.5 (評価文脈)**.** 評価文脈 $\mathcal{E}$ は次で定義される.

$$\mathcal{E} ::= [\,] \mid \mathcal{E} \ @^l \ M \mid v \ @^l \ \mathcal{E} \mid \mathbf{acc}^l(\mathcal{E}) \mid \mathbf{if}\ \mathcal{E}\ \mathbf{then}\ M_1\ \mathbf{else}\ M_2 \mid \langle \mathcal{E} \rangle_l$$

$\mathcal{E}[M]$ は $\mathcal{E}$ 中の $[\,]$ を M で置き換えた項を示す.

$$\frac{z \text{ fresh}}{(H, \mathcal{E}[\mathbf{new}^{\Phi}()]) \rightsquigarrow (H \uplus \{z \mapsto \Phi\}, \mathcal{E}[z])} \quad \text{(R-New)}$$

$$\frac{b = \mathbf{true} \text{ or } \mathbf{false} \qquad \Phi^{-l} \neq \emptyset}{(H \uplus \{x \mapsto \Phi\}, \mathcal{E}[\mathbf{acc}^{l}(x)]) \rightsquigarrow (H \uplus \{x \mapsto \Phi^{-l}\}, \mathcal{E}[b])} \quad \text{(R-Acc)}$$

$$\frac{\Phi^{-l} = \emptyset}{(H \uplus \{x \mapsto \Phi\}, \mathcal{E}[\mathbf{acc}^{l}(x)]) \rightsquigarrow \mathbf{Error}} \quad \text{(R-AccErr)}$$

$$(H, \mathcal{E}[\mathbf{fun}(f, x, M) \ @^{l} \ v]) \rightsquigarrow (H, \mathcal{E}[\langle [\mathbf{fun}(f, x, M)/f, v/x]M \rangle_{l}]) \quad \text{(R-App)}$$

$$(H, \mathcal{E}[\mathbf{if\ true\ then}\ M_1\ \mathbf{else}\ M_2]) \rightsquigarrow (H, \mathcal{E}[M_1]) \quad \text{(R-IfT)}$$

$$(H, \mathcal{E}[\mathbf{if\ false\ then}\ M_1\ \mathbf{else}\ M_2]) \rightsquigarrow (H, \mathcal{E}[M_2]) \quad \text{(R-IfF)}$$

$$(H, \mathcal{E}[\langle v \rangle_{l}]) \rightsquigarrow (H, \mathcal{E}[v]) \quad \text{(R-Return)}$$

図 1: 操作的意味論

定義 2.6 (簡約関係)**.** 簡約関係 $(H, M) \rightsquigarrow P$ (ただし P は (H', M') もしくは **Error**) は図 1 の規則で閉じている最小の関係である．ここで，Φ^{-l} は $\{s \mid ls \in \Phi\}$ である．

規則 R-Acc の条件 $\Phi^{-l} \neq \emptyset$ はトレース集合 Φ に l で始まるトレースが含まれていること，つまり l でアクセスしてよいことを示している．この条件が満たされない場合はこのアクセスは不正なものなので **Error** になる (規則 R-AccErr)．l でラベル付けされた関数適用で関数が呼び出されると，関数本体の式には $\langle\cdot\rangle_l$ が付く．関数呼び出しの計算が終了して本体が値に簡約されると $\langle\cdot\rangle_l$ が外れる．

例 2.7 (簡約の例)**.** $\Phi = ((R + W)^{*} C \downarrow)^{\sharp}, R = l_3, C = l_1$ とする．

$$\begin{aligned} M &\stackrel{\text{def}}{=} \mathbf{let}\ y\ =\ \mathbf{new}^{\Phi}()\ \mathbf{in}\ M_1 \\ M_1 &\stackrel{\text{def}}{=} \mathbf{let}\ f\ =\ \mathbf{fun}(f, x, \mathbf{close}^{l_1}(x))\ \mathbf{in} \\ &\qquad f\ @^{l_2}\ y;\ \mathbf{read}^{l_3}(y) \end{aligned}$$

は関数呼び出しの中でファイルを閉じた後に読み込みを試みて失敗する．M は次のように簡約される．z は生成された資源に束縛される新しい変数である．

$$\begin{aligned} & (\{\}, M) \\ \rightsquigarrow\ & (\{z \mapsto ((R + W)^{*} C \downarrow)^{\sharp}\}, \mathbf{let}\ y\ =\ z\ \mathbf{in}\ M_1) \\ \rightsquigarrow\ & (\{z \mapsto ((R + W)^{*} C \downarrow)^{\sharp}\}, \mathbf{fun}(f, x, \mathbf{close}^{l_1}(x))\ @^{l_2}\ z;\ \mathbf{read}^{l_3}(z)) \\ \rightsquigarrow\ & (\{z \mapsto ((R + W)^{*} C \downarrow)^{\sharp}\}, \langle \mathbf{close}^{l_1}(z) \rangle_{l_2};\ \mathbf{read}^{l_3}(z)) \\ \rightsquigarrow\ & (\{z \mapsto \downarrow^{\sharp}\}, \langle b \rangle_{l_2};\ \mathbf{read}^{l_3}(z)) \\ \rightsquigarrow\ & (\{z \mapsto \downarrow^{\sharp}\}, b;\ \mathbf{read}^{l_3}(z)) \\ \rightsquigarrow\ & (\{z \mapsto \downarrow^{\sharp}\}, \mathbf{read}^{l_3}(z)) \\ \rightsquigarrow\ & \mathbf{Error} \end{aligned}$$

3　型システム

ここでは資源使用解析のための型システムを定義する．この型システムの下でプログラムに型が付けば，実行の際には仕様に従った安全な資源アクセスしか起こらないことが保証される．以下では，まず使用法表現を定義し，その意味を与える．そして型と型環境を与え，型判断を導出する推論規則を定義する．

3.1　使用法表現

3.1.1　使用法表現の構文

まず，使用法表現の構文を定義する前に，アクセス一回分の情報を表す単位使用を定義する．単位使用は，関数のリターンを表す特別な記号 $\uparrow$ が先頭に 0 個以上ついた空でないラベルの有限列である．

定義 3.1 (単位使用)**.** 単位使用 u の集合 $\mathcal{U}$ を

$$\mathcal{U} \quad = \quad \{\underbrace{\uparrow.\cdots.\uparrow}_{k}.l_1.\cdots.l_n \mid n,k \geq 0 \wedge l_i \in \mathcal{L}\} \setminus \{\varepsilon\}$$

とする．ε は空列である．

ラベル上の順序から，二つの単位使用のどちらが先に発生するアクセスかを示す順序が定義される．

定義 3.2 (単位使用上の順序 $u_1 \sqsubset u_2$)**.** 単位使用上の順序 $u_1 \sqsubset u_2$ は $\mathcal{L} \cup \{\uparrow\}$ 上の順序 $\sqsubset \cup \{(l,\uparrow) \mid l \in \mathcal{L}\}$ に基づく辞書式順序である．

定義 3.3 (使用法表現)**.** 使用法表現 U は次の文法で定義される．

$$U \quad ::= \quad \mathbf{0} \mid \alpha \mid u \mid U_1 \,\&\, U_2 \mid \mu\alpha.\, U \mid U_1 \otimes U_2 \mid U_1[U_2]$$

$\mathbf{0}$ は空のアクセス，単位使用 u はアクセス一回を表す．$\mu\alpha.\, U$ は U 中の使用変数 α が $\mu\alpha.\, U$ 全体を指している再帰的な使用 $\mu\alpha.\, U$ である．$U_1 \,\&\, U_2$ は U_1 と U_2 のどちらかのアクセスが起こることを表す．$U_1 \otimes U_2$ は U_1 と U_2 の両方のアクセスが，単位使用の順序に従って起こることを表す．$U_1[U_2]$ は，ある関数の呼び出しが使用法表現 U_1 で表され，その各呼び出しの中で U_2 というアクセスが起きるときに，全体のアクセスを表すのに用いられる．直感的には，U_1 の各単位使用と U_2 の各単位使用について，全ての組み合わせについて連結したような意味になる．単位使用の連結に関して $\uparrow$ は左側のラベルをキャンセルする．例えば $(l_1 \otimes l_2)[l_3\& \uparrow .l_4]$ は，関数呼び出しが l_1 と l_2 の二つのプログラムポイントで起こり，各呼び出しごとに $l_3\& \uparrow .l_4$ で表されるアクセスが起きることを表すので，$l_1[l_3 \,\&\, \uparrow .l_4] \otimes l_2[l_3 \,\&\, \uparrow .l_4]$ と同じ意味である．更に上で説明した $\uparrow$ の作用により，$(l_1.l_3\&l_4) \otimes (l_2.l_3\&l_4)$ と同じ意味である．

3.1.2　使用法表現の意味論

使用法表現の意味をラベル遷移系を用いて定義する．使用法表現の意味は可能な遷移列の集合として与えられる．更に定義した意味に基づいて部分使用関係 $U_1 \leq U_2$ を定義する．部分使用関係は U_2 よりも U_1 の方が一般的な使用法であること，つまり，U_1 の使用法に従ってアクセスできる資源は U_2 に従ってアクセスしてもよいことを表す．まずは，意味論を定義するために必要になる補助的な述語，関係を定義する．

定義 3.4 (関係 $\preceq$)**.** 関係 $\preceq$ は図 2 の規則で閉じている最小の関係である．ここで $U_1 \equiv U_2$ は $U_1 \preceq U_2$ かつ $U_2 \preceq U_1$ を示す．

$$U_1 \,\&\, U_2 \preceq U_1 \qquad U_1 \,\&\, U_2 \preceq U_2 \qquad \mu\alpha.\, U \equiv [\mu\alpha.\, U/\alpha]U$$

$$\frac{U_1 \equiv U_1' \qquad U_2 \equiv U_2'}{U_1 \otimes U_2 \equiv U_1' \otimes U_2'} \qquad \frac{U_1 \equiv U_1' \qquad U_2 \equiv U_2'}{U_1[U_2] \equiv U_1'[U_2']}$$

$$(U_{11} \otimes U_{12})[U_2] \equiv (U_{11}[U_2]) \otimes (U_{12}[U_2]) \qquad (U_{11} \,\&\, U_{12})[U_2] \equiv (U_{11}[U_2]) \,\&\, (U_{12}[U_2])$$

$$u[U_1 \otimes U_2] \equiv (u[U_1]) \otimes (u[U_2]) \qquad u[U_1 \,\&\, U_2] \equiv (u[U_1]) \,\&\, (u[U_2])$$

$$\mathbf{0}[U_2] \equiv \mathbf{0} \qquad U[\mathbf{0}] \equiv \mathbf{0} \qquad U_1[U_2[U_3]] \equiv (U_1[U_2])[U_3]$$

$$\frac{l \sqsubset l'}{u_1.l[\uparrow .l'.u_2] \equiv u_1.l'.u_2} \qquad u_1.l[\uparrow . \uparrow .u_2] \equiv u_1[\uparrow .u_2] \qquad u_1[l.u_2] \equiv u_1.l.u_2 \qquad (\uparrow . \cdots . \uparrow)[u] \equiv \uparrow . \cdots . \uparrow .u$$

図 2: 関係 $\preceq$

$void(U)$:

$$void(\mathbf{0}) \qquad \frac{void(U_1)}{void(U_1[U_2])} \qquad \frac{void(U_2)}{void(U_1[U_2])} \qquad \frac{void([\mu\alpha.\, U/\alpha]U)}{void(\mu\alpha.\, U)}$$

$$\frac{\mathbf{op} = \,\&\text{ or } \otimes \qquad void(U_1) \qquad void(U_2)}{void(U_1 \mathbf{op} U_2)}$$

$U^{\downarrow}$:

$$\frac{U \preceq U' \qquad void(U')}{U^{\downarrow}}$$

$U^{\Downarrow u}$:

$$\frac{u_1 \not\sqsubseteq u_2}{{u_1}^{\Downarrow u_2}} \qquad \frac{void(U)}{U^{\Downarrow u}} \qquad \frac{U^{\Downarrow u_2}}{(u_1[U])^{\Downarrow u_1.u_2}} \qquad \frac{{U_1}^{\Downarrow u}}{(U_1[U_2])^{\Downarrow u}}$$

$$\frac{([\mu\alpha.\, U/\alpha]U)^{\Downarrow u}}{(\mu\alpha.\, U)^{\Downarrow u}} \qquad \frac{\mathbf{op} = \,\&\text{ or } \otimes \qquad {U_1}^{\Downarrow u} \qquad {U_2}^{\Downarrow u}}{(U_1 \mathbf{op} U_2)^{\Downarrow u}}$$

図 3: 述語 $void(U)$, $U^{\downarrow}$, $U^{\Downarrow u}$

$\preceq$ は先ほど説明した，& や μ，$U_1[U_2]$ といった使用法表現の演算子の意味を与える関係である．例えば, $\uparrow$ は $l[U]$ の l と相殺することで関数呼び出し l の外側で起こるアクセスであることを表すが, この意味は $l[\uparrow .l'.u] \preceq l'.u$ という関係で与えられる．l は関数呼び出しを, l' は返り値がどこでアクセスされるかを表す．この関係を導く規則の前提部分の $l \sqsubset l'$ は, 関数呼び出しよりも先に返り値のアクセスが起こらないということを反映した条件で, 型システムの中で意味的に矛盾した使用法表現が現れるのを防ぐためである．

定義 3.5 (使用法表現の述語 $void(U)$, $U^{\downarrow}$, $U^{\Downarrow u}$)**.** 使用法表現上の述語 $void(U)$, $U^{\downarrow}$, $U^{\Downarrow u}$ は図 3 の規則に関して閉じている最小の関係である．

$void(U)$ は使用法 U がアクセスを起こさないことを表している．$U^{\downarrow}$ は U で表される使用法の中に何のアクセスも起こさないものが含まれていることを表す．

$U^{\Downarrow u}$ は, u が使用法 U で起こるどのアクセスよりも先に起こることを表す．$U_1[U_2]$ に関する 3 番目，4 番目の規則は以下のような意味である．まず，$U_1[U_2]$ の U_1 は U_2 で表されるアクセスが起こる文脈を表しており，文脈 U_1 に対応する関数呼び出しよりも先に u が起こることがわかれば $({U_1}^{\Downarrow u})$, その関数の中で起こるどのアクセスよりも u が先であること $((U_1[U_2]^{\Downarrow u})$ がわかる．また，U より u_2 が先に発生するなら

$$u \xrightarrow{u} \mathbf{0} \qquad \frac{U_1 \xrightarrow{u} U_1' \qquad U_2{}^{\Downarrow u}}{U_1 \otimes U_2 \xrightarrow{u} U_1' \otimes U_2} \qquad \frac{U_2 \xrightarrow{u} U_2' \qquad U_1{}^{\Downarrow u}}{U_1 \otimes U_2 \xrightarrow{u} U_1 \otimes U_2'} \qquad \frac{U \preceq U'' \qquad U'' \xrightarrow{u} U'}{U \xrightarrow{u} U'}$$

図 4: 使用法表現の遷移規則

$(U^{\Downarrow u_2})$, 文脈 u_1 の中でもその関係が保存される $(u_1[U]^{\Downarrow u_1.u_2})$.

定義 3.6 (使用法表現の遷移)**.** 使用法表現上の遷移関係 $U \xrightarrow{u} U'$ は図 4 の規則で閉じている最小の関係である.

$U \xrightarrow{u} U'$ は, U が単位使用 u の表すアクセスの後 U' に遷移することを表す. ラベルの順序から一番最初に起こるアクセス u を選ばれて遷移が起こる. $U = u$ のときは, u が選ばれて $\mathbf{0}$ になる. 二番目の規則では, $U = U_1 \otimes U_2$ の場合に, 前提条件 $(U_2{}^{\Downarrow u})$ から, U_1 が一番最初に起こすアクセス u が先に実行されることがわかるので, U_1 の方が選ばれる. 三番目も同様である. 最後の規則は, $\preceq$ の表す, 再帰の展開や演算子の結合法則に従った変換などを行った後に遷移を行う規則である.

例 3.7. $l_1 \sqsubset l$ とする. $\mu\alpha.\,(l_1 \otimes l[\alpha])$ は次のように遷移する.

$$\begin{array}{rl}
 & \mu\alpha.\,(l_1 \otimes l[\alpha]) \\
\preceq & l_1 \otimes l[\mu\alpha.\,(l_1 \otimes l[\alpha])] \\
\xrightarrow{l_1} & \mathbf{0} \otimes l[\mu\alpha.\,(l_1 \otimes l[\alpha])] \\
\preceq & \mathbf{0} \otimes l[l_1 \otimes l[\mu\alpha.\,(l_1 \otimes l[\alpha])]] \\
\preceq & \mathbf{0} \otimes (l[l_1] \otimes l[l[\mu\alpha.\,(l_1 \otimes l[\alpha])]]) \\
\preceq & \mathbf{0} \otimes (l.l_1 \otimes l[l[\mu\alpha.\,(l_1 \otimes l[\alpha])]]) \\
\xrightarrow{l.l_1} & \mathbf{0} \otimes (\mathbf{0} \otimes l[l[\mu\alpha.\,(l_1 \otimes l[\alpha])]]) \\
\cdots &
\end{array}$$

定義 3.8 (使用法表現のトレース集合)**.** 使用法表現 U に対して集合 $[\![U]\!]$ を次で定義する.

$$\begin{array}{rcl}
[\![U]\!] & = & \{l_1 \cdots l_n \mid \exists U_1, \ldots, U_n.(U \xrightarrow{u_1} U_1 \xrightarrow{u_2} \cdots \xrightarrow{u_n} U_n) \wedge u_i = \cdots.l_i (1 \le i \le n)\} \\
 & & \cup \{l_1 \cdots l_n \downarrow \mid \exists U_1, \ldots, U_n.(U \xrightarrow{u_1} U_1 \xrightarrow{u_2} \cdots \xrightarrow{u_n} U_n) \wedge U_n^{\downarrow} \wedge u_i = \cdots.l_i (1 \le i \le n)\}
\end{array}$$

定義より $[\![U]\!]$ は prefix に関して閉じているのでトレース集合であるといえる. $[\![U]\!]$ は使用法表現 U が表すアクセス系列の集合である. u_i の頭に付いているラベル列は文脈情報で, 一番右側が読み書き等の具体的なアクセスを表している. 文脈情報はアクセス順序を知るために必要なだけなので, アクセス系列そのものを表すのには不要である.

例 3.9.

$$[\![\mu\alpha.\,((\mathbf{0} \,\&\, l_1) \otimes l[\alpha])]\!] \quad = \quad \{\downarrow, l_1 \downarrow, l_1 l_1 \downarrow, \ldots\}^{\sharp}$$

3.1.3 部分使用関係

使用文脈 $\mathcal{C}$ を穴 [] を一つだけ含むような使用法表現とし, 使用文脈 $\mathcal{C}$ 中の穴を使用法表現 U で置き換えたものを $\mathcal{C}[U]$ と表わす. 部分使用関係 $U_1 \le U_2$ は, U_2 が U_1 よりも一般的な使用法表現であることを表し, 以下のように余帰納的に定義される.

定義 3.10 (部分使用関係)**.** 部分使用関係 $U_1 \le U_2$ を次の条件を満たす最大の関係とする.

(1) 任意の使用文脈 $\mathcal{C}$ に対して $\mathcal{C}[U_1] \leq \mathcal{C}[U_2]$.

(2) $U_2 \xrightarrow{u} U_2'$ ならば, ある U_1' が存在して, $U_1 \xrightarrow{u} U_1'$, $U_1' \leq U_2'$.

(3) $U_2^{\downarrow}$ ならば $U_1^{\downarrow}$.

例 3.11. $U_1 \,\&\, U_2 \leq U_1$, $\mathbf{0} \otimes U_1 \leq U_1$, $(l_1 \otimes l_2)[u] \leq l_1.u \otimes l_2.u$ などが成り立つ.

3.2 型，型環境

定義 3.12 (型)**.** 型 τ を以下の文法により定義する.

$$\tau \quad ::= \quad \mathbf{bool} \mid (\mathbf{R}, U) \mid (\tau_1 \to \tau_2, U)$$

bool は真偽値の型，$(\mathbf{R}, U)$ は U に従ってアクセスされる資源の型，$(\tau_1 \to \tau_2, U)$ は U に従って呼び出される関数の型である．τ_1 は引数が関数呼び出し内部でどうアクセスされるか，τ_2 は関数の返り値が呼び出し側でどうアクセスされるかの情報を示す．型から使用法表現を抽出する関数 $Use(U)$ を $Use(\mathbf{bool}) = \mathbf{0}$, $Use((\tau_1 \to \tau_2, U)) = U$, $Use((\mathbf{R}, U)) = U$ で定義する.

次に，部分使用関係を使って部分型関係を定義する.

定義 3.13 (部分型関係)**.** 型上の部分型関係 $\leq$ は次の規則で閉じている最小の関係である.

$$\mathbf{bool} \leq \mathbf{bool} \qquad \frac{U \leq U'}{(\tau_1 \to \tau_2, U) \leq (\tau_1 \to \tau_2, U')} \qquad \frac{U \leq U'}{(\mathbf{R}, U) \leq (\mathbf{R}, U')}$$

型環境 Γ は変数 x から型 τ への写像である．$\Gamma \setminus x$ は Γ の定義域から x を除いた写像を表す．型及び型環境上の演算を次で定義する.

定義 3.14 (型と型環境上の演算)**.** 型と型環境に対する使用文脈の適用を次で定義する.

$$\begin{aligned} \mathcal{C}[\mathbf{bool}] &= \mathbf{bool} \\ \mathcal{C}[(\tau_1 \to \tau_2, U)] &= (\tau_1 \to \tau_2, \mathcal{C}[U]) \\ \mathcal{C}[(\mathbf{R}, U)] &= (\mathbf{R}, \mathcal{C}[U]) \end{aligned} \qquad \mathcal{C}[\Gamma](x) = \mathcal{C}[\Gamma(x)]$$

型と型環境に対する使用法表現の演算子の適用を次で定義する．ここで $\mathbf{op} = \otimes \text{or } \&$ である.

$$\begin{aligned} \mathbf{bool}\ \mathbf{op}\ \mathbf{bool} &= \mathbf{bool} \\ (\tau_1 \to \tau_2, U_1)\ \mathbf{op}\ (\tau_1 \to \tau_2, U_2) &= (\tau_1 \to \tau_2, U_1\ \mathbf{op}\ U_2) \\ (\mathbf{R}, U_1)\ \mathbf{op}\ (\mathbf{R}, U_2) &= (\mathbf{R}, U_1\ \mathbf{op}\ U_2) \end{aligned}$$

$$\begin{aligned} dom(\Gamma_1\ \mathbf{op}\ \Gamma_2) &= dom(\Gamma_1) \cup dom(\Gamma_2) \\ (\Gamma_1\ \mathbf{op}\ \Gamma_2)(x) &= \begin{cases} \Gamma_1(x)\ \mathbf{op}\ \Gamma_2(x) & \text{if } x \in dom(\Gamma_1) \cap dom(\Gamma_1) \\ \Gamma_1(x) & \text{if } x \in dom(\Gamma_1) \setminus dom(\Gamma_2) \\ \Gamma_2(x) & \text{if } x \in dom(\Gamma_2) \setminus dom(\Gamma_1) \end{cases} \end{aligned}$$

使用法表現上の述語 $U^{\Downarrow u}$ も同様に型の上の述語 $\tau^{\Downarrow u}$ に拡張する．また，表記 $\Gamma_1 \leq \Gamma_2$ は $dom(\Gamma_1) \supseteq dom(\Gamma_2)$, $\forall x \in dom(\Gamma_2).(\Gamma_1(x) \leq \Gamma_2(x))$, $\forall x \in dom(\Gamma_1) \setminus dom(\Gamma_2).Use(\Gamma_1(x)) \leq \mathbf{0}$ であることを表わす.

$$x : \tau \vdash x : \tau \qquad \text{(T-Var)}$$

$$\frac{c = \textbf{true} \text{ or } \textbf{false}}{\vdash c : \textbf{bool}} \qquad \text{(T-Const)}$$

$$\frac{\Gamma, f : (\tau_1 \to \tau_2, U_1), x : \tau_1 \vdash M : \uparrow [\tau_2]}{U_2[\mu\alpha.\ (\Gamma \otimes U_1[\alpha])] \vdash \textbf{fun}(f, x, M) : (\tau_1 \to \tau_2, U_2)} \qquad \text{(T-Fun)}$$

$$\frac{\Gamma_1 \vdash M_1 : (\tau_1 \to \tau_2, l) \qquad \Gamma_2 \vdash M_2 : l[\tau_1] \qquad \tau_2 \le \tau_2' \qquad {\tau_2'}^{\Downarrow l}}{\Gamma_1 \otimes \Gamma_2 \vdash M_1 \ @^l \ M_2 : \tau_2'} \qquad \text{(T-App)}$$

$$\frac{[\![U]\!] \subseteq \Phi}{\vdash \textbf{new}^{\Phi}() : (\mathbf{R}, U)} \qquad \text{(T-New)}$$

$$\frac{\Gamma \vdash M : (\mathbf{R}, l)}{\Gamma \vdash \textbf{acc}^l(M) : \textbf{bool}} \qquad \text{(T-Acc)}$$

$$\frac{\Gamma_1 \vdash M_1 : \textbf{bool} \qquad \Gamma_2 \vdash M_2 : \tau \qquad \Gamma_3 \vdash M_3 : \tau}{\Gamma_1 \otimes (\Gamma_2 \ \& \ \Gamma_3) \vdash \textbf{if } M_1 \textbf{ then } M_2 \textbf{ else } M_3 : \textbf{bool}} \qquad \text{(T-If)}$$

$$\frac{\Gamma \vdash M : \tau \qquad \tau^{\Downarrow(\uparrow . l)}}{l[\Gamma] \vdash \langle M \rangle_l : l[\tau]} \qquad \text{(T-Return)}$$

$$\frac{\Gamma \vdash M : \tau \qquad \Gamma' \le \Gamma \qquad \tau \le \tau'}{\Gamma' \vdash M : \tau'} \qquad \text{(T-Sub)}$$

図 5: 型付け規則 $\Gamma \vdash M : \tau$

3.3 型付け規則

型判断 $\Gamma \vdash M : \tau$ を導出する推論規則を定義する．型判断の直観的な意味は, 項 M を評価する際に Γ で表されるアクセスが起こり, 結果の値が型 τ を持つことである．

定義 3.15 (型判断)**.** 型判断 $\Gamma \vdash M : \tau$ は, 図 5 の型付け規則に関して閉じている最小の関係である．

型判断 $\Gamma \vdash M : \tau$ の型環境 Γ は M の評価中に資源がどう使われるかを示しているため，複数の部分項を持つ項の型付け規則を考える際には，それぞれの型環境を $\otimes$ や $\&$ で結合する．例えば (T-If) 規則では，M_1 の評価中の資源へのアクセスが Γ_1，その後に M_2 もしくは M_3 が評価されるため，全体の型環境は $\Gamma_1 \otimes (\Gamma_2 \& \Gamma_3)$ となる．(T-New) 規則は，使用法表現が仕様に従っていなければならないことを，また，(T-Acc) 規則は，アクセスされる資源 M の型にある使用法表現が **acc** に付けられたラベルと一致していなければならない，ということを示している．

規則 (T-App) は，$\tau_2 \leq \tau_2'$ などの追加された条件を除けば，1 節で説明した通りである．(追加された条件については後述する．)

規則 (T-Fun) も基本的には 1 節で説明した通りであるが，再帰関数を扱うために複雑になっている．まず，Γ は関数呼び出しが一度起こった時に発生する資源へのアクセスを示している．前提にある U_1 は，この関数がどのように再帰呼び出しされるかを示している．そのため，この関数は一度呼び出される度に $\Gamma \otimes U_1[\Gamma \otimes U_1[\Gamma \otimes \cdots]]$ つまり，$\mu\alpha.\Gamma \otimes U_1[\alpha]$ というアクセスが発生する．結論にある U_2 は，この再帰関数が外側からどう呼び出されるかを示しているので，全体として，資源へのアクセスは $U_2[\mu\alpha.\Gamma \otimes U_1[\alpha]]$ と表現される．M の型 $\uparrow[\tau_2]$ は，M の評価結果が，関数からリターンした文脈のもとで τ_2 に従って使われることを示している．

規則 (T-Return) は，l での関数呼び出しの中での M の評価中のアクセス情報は M の型環境に $l[\cdots]$ を適用することによって得られることを示している．規則 (T-Sub) は，いわゆる subsumption の規則である．

次に (T-App) の $\tau_2 \leq \tau_2'$ と ${\tau_2'}^{\Downarrow l}$ と (T-Return) の $\tau^{\Downarrow(\uparrow.l)}$ について説明する．

${\tau_2'}^{\Downarrow l}$ と $\tau^{\Downarrow(\uparrow.l)}$ は，関数の返り値に対するアクセスは関数呼び出しよりも後に起こる, という事実を反映しており，型判断の導出の中に意味を持たない使用法表現が現れないようにするための条件である．例えば，これらの条件がないとして，関数 M_1 の返り値の型 τ_2 が $(\mathbf{R}, l_1)$ で $l_1 \sqsubset l$ の場合を考える．返り値に対するアクセスを表す l_1 の方が関数呼び出しを表す l よりも先の順序になっている．$M_1 \ @^l \ M_2$ の簡約が進むと M_1 は $\mathbf{fun}(f, x, M_0)$ という形になる．M_0 の型は $(\mathbf{R}, \uparrow[l_1])$ である．ここで関数が呼び出されると，$M_1 \ @^l \ M_2$ は $\langle[\mathbf{fun}(f, x, M_0)/f, v/x]M_0\rangle_l$ という形になる．(T-Return) の規則より，この式には $(\mathbf{R}, l[\uparrow[l_1]])$ という型が付く．しかし, この使用法表現は $void(U)$ でないにも関わらず, $l[\uparrow[l_1]] \preceq l_1$ が成立しないため遷移を持たないという，意味のない使用法表現である．

また，(T-App) の $\tau_2 \leq \tau_2'$ は，関数から返る資源に対するアクセスの可能性のうち，今考えている文脈でのアクセスのみに対して $(\cdot)^{\Downarrow l}$ の条件を課せるように導入されている．例えば，$l_2 \sqsubset l_3 \sqsubset l_4 \sqsubset l_5$ の時，

$$\begin{array}{l} \mathbf{let}\ f\ =\ \mathbf{fun}(f, x, \mathbf{read}^{l_1}(x);\ x)\ \mathbf{in} \\ \quad \mathbf{write}^{l_3}(f\ @^{l_2}\ y);\ \mathbf{close}^{l_5}(f\ @^{l_4}\ y) \end{array}$$

の $\mathbf{fun}(f, x, \cdots)$ は $((\mathbf{R}, l_1 \otimes (\uparrow[l_3 \& l_5])) \to (\mathbf{R}, l_3\ \&\ l_5), l_2 \otimes l_4)$ という型を持つ．つまり，この関数が返す資源は l_3 もしくは l_5 で一度使ってよいことを示している．ここで，$(l_3 \& l_5)^{\Downarrow l_4}$ は成立しないため，$l_3 \& l_5 \leq l_5$ かつ ${l_5}^{\Downarrow l_4}$ であることから $\tau_2 \leq \tau_2'$ の条件を使って $f\ @^{l_4}\ y$ を型付けすることになる．

例 3.16. $\Phi = (R + W)^* C, R = l_1, C = l_4, l_1 \sqsubset l_2, l_3 \sqsubset l_4$ とする．

$$M \stackrel{\text{def}}{=} \begin{array}{l} \mathbf{let}\ y\ =\ \mathbf{new}^{\Phi}()\ \mathbf{in} \\ \mathbf{let}\ f\ =\ \mathbf{fun}(f, x, \mathbf{read}^{l_1}(x);\ f\ @^{l_2}\ x)\ \mathbf{in} \\ \quad f\ @^{l_3}\ y;\ \mathbf{close}^{l_4}(y) \end{array}$$

の型判断の導出を考える.

$\Gamma_1 = f : ((\mathbf{R}, \mu\alpha.\ l_1 \otimes l_2[\alpha]) \to \mathbf{bool}, l_3)$, $\Gamma_2 = y : (\mathbf{R}, l_3[\mu\alpha.\ l_1 \otimes l_2[\alpha]])$, $\Gamma_3 = y : (\mathbf{R}, l_4)$ とすると,

$$D_1 \quad = \quad \frac{\Gamma_1 \vdash f : ((\mathbf{R}, \mu\alpha.\ l_1 \otimes l_2[\alpha]) \to \mathbf{bool}, l_3) \qquad \Gamma_2 \vdash y : (\mathbf{R}, l_3[\mu\alpha.\ l_1 \otimes l_2[\alpha]])}{\Gamma_1, \Gamma_2 \vdash f\ @^{l_3}\ y : \mathbf{bool}}$$

$$D_2 \quad = \quad \frac{\Gamma_3 \vdash y : (\mathbf{R}, l_4)}{\Gamma_3 \vdash \mathbf{close}^{l_4}(y)}$$

より $\Gamma_4 = y : (\mathbf{R}, (l_3[\mu\alpha.\ l_1 \otimes l_2[\alpha]]) \otimes l_4) (= \Gamma_2 \otimes \Gamma_3)$ として,

$$D_3 \quad = \quad \frac{D_1 \qquad D_2}{\Gamma_1, \Gamma_4 \vdash f\ @^{l_3}\ y;\ \mathbf{close}^{l_4}(y) : \mathbf{bool}}$$

が得られる. ここで,

$$M_1 \stackrel{\text{def}}{=} \begin{array}{l} \mathbf{let}\ f\ =\ \mathbf{fun}(f, x, \mathbf{read}^{l_1}(x);\ f\ @^{l_2}\ x)\ \mathbf{in} \\ \qquad f\ @^{l_3}\ y;\ \mathbf{close}^{l_4}(y) : \mathbf{bool} \end{array}$$

とすると,

$$D_4 \quad = \quad \emptyset \vdash \mathbf{fun}(f, x, \mathbf{read}^{l_1}(x);\ f\ @^{l_2}\ x) : ((\mathbf{R}, \mu\alpha.\ l_1 \otimes l_2[\alpha]) \to \mathbf{bool}, l_3)$$

$$D_5 \quad = \quad \frac{D_4 \qquad D_3}{\Gamma_4 \vdash M_1 : \mathbf{bool}}$$

$$D_6 \quad = \quad \frac{[\![(l_3[\mu\alpha.\ l_1 \otimes l_2[\alpha]]) \otimes l_4]\!] \subseteq \Phi}{\emptyset \vdash \mathbf{new}^{\Phi}() : (\mathbf{R}, (l_3[\mu\alpha.\ l_1 \otimes l_2[\alpha]]) \otimes l_4)}$$

より,

$$\frac{D_6 \qquad D_5}{\emptyset \vdash M : \mathbf{bool}}$$

が導出される.

3.4 型健全性

これまでに定義した型システムの型健全性について述べる. 型健全性定理は, プログラムに型が付けば, 実行した際に資源へのアクセスエラーを起きないことと, 実行が終了したときに, 生成された資源はこれ以上の処理を必要としない状態 (ファイルは閉じられている等) であることを保証する定理である.

定理 3.17 (型健全性)**.** $\emptyset \vdash M : \tau$, $Use(\tau) \leq \mathbf{0}$ のとき, 次が満たされる.

(1) $(\{\}, M) \not\rightsquigarrow^* \mathbf{Error}$

(2) $(\{\}, M) \rightsquigarrow^* (H, v)$ ならば $\forall x \in dom(H)$ に対して $\downarrow \in H(x)$.

この定理は, 簡約が型付けを保存するという型保存定理から証明できる. 型保存定理を述べるために, 型環境 Γ とヒープ H の対応を与える. Γ と H が $dom(\Gamma) = dom(H)$, $\forall x \in dom(\Gamma).H(x) \supseteq [\![\Gamma(x)]\!]$ を満たすとき, 型環境 Γ とヒープ H が対応しているといい, $\mathcal{H}(\Gamma, H)$ と書く.

定理 3.18 (型保存定理)**.** $\Gamma \vdash M : \tau$, $\mathcal{H}(\Gamma, H)$, $(H, M) \rightsquigarrow (H', M')$ ならば, ある Γ' が存在して $\Gamma' \vdash M' : \tau$, $\mathcal{H}(\Gamma', H')$.

現段階では部分型関係に関する補題の証明など一部がまだ完成していないが, 証明概略は付録付バージョンを参照のこと.

4 関連研究

Igarashi, Kobayashi[4] は使用法表現を用いて資源使用解析を行う型システムを与えた．この型システムでは，各使用同士の順序関係は ";" という使用法表現の構成子によって表されていた．従って，異なる変数 x, y が同一の資源に対して束縛されている場合，変数 x を介した使用と変数 y を介した使用とは型環境において独立に保持され，結果として,「x を介した読み込みが y を介したクローズよりも先に起こる」といった情報を得ることができない．それに対し，本論文の型システムでは，各アクセスがいつ起きるかという情報を，ラベルの順序関係と関数の呼び出し系列によって表現しているため，資源がどの変数からアクセスされるかによらずアクセス間の順序関係をより正確に表現できる．例えば，どちらの型システムでも型判断

$$\mathtt{x} : (\mathbf{File}, l_1), \mathtt{y} : (\mathbf{File}, l_2) \vdash \mathtt{read}^{l_1}(\mathtt{x});\ \mathtt{close}^{l_2}(\mathtt{y}) : \mathbf{bool}$$

と，そこから

$$\mathtt{x} : (\mathbf{File}, l_1 \otimes l_2) \vdash \mathtt{let\ y\ =\ x\ in\ read}^{l_1}(\mathtt{x});\ \mathtt{close}^{l_2}(\mathtt{y}) : \mathbf{bool}$$

が導出できるが，Igarashi, Kobayashi の型システムでは，l_1, l_2 の関係は何も仮定されていないので，x を介した読み込みとクローズのどちらが先に起こることがわからない．また，$\mathtt{x} : (\mathbf{File}, l_1; l_2)$ の下で型付けすることもできない．

エフェクトシステムを用いた資源使用解析として [7, 2, 1, 3, 11] がある．[2] はリージョンを用いて資源を表し, エイリアス解析とエフェクトの推論を組み合わせて解析を行う．エイリアスがある場合に複数の資源が同じリージョンに割り当てられるため精度が落ちる．[1] ではリージョンに多相性を持たせることと, 型に資源の状態変化を書くことで文脈情報の解析精度を上げている．[3, 7] も同様の特徴を持つ．また, これらの解析 [7, 2, 1, 3] では代入を扱うことができる．

[6] では [4] の線形型システムとエフェクトシステムを組み合わせることで, 文脈依存な解析を実現している．リージョンを用いてアクセスが起こる時刻を表すことで, アクセスの起こるタイミングをより正確に表現できる．そのため, 本研究の手法と同じく [4] では不可能であった関数呼び出しの中で起こるアクセスの順序を解析することができる．しかし, 同じ関数が複数回呼ばれている場合に, 各々の呼び出しの中で起こるアクセス同士の順序がわからない．多相的なリージョンを用いて彼等の体系を拡張すれば可能であるが, 型システムが複雑になることに加え, 再帰呼び出しの中で起こるアクセス同士の順序を解析しようとすれば, 再帰多相な型システムとなるため型推論が決定不能になる．

関連する文脈依存なプログラム解析として, 制御フロー解析 (CFA) が挙げられる. 制御フロー解析とは, プログラムの各部分が実行中にどの関数値になり得るかを調べる解析である．それらの解析の中で k-CFA[5, 8, 10] と呼ばれるものは, (予め与えられた) 深さ k までの関数呼び出し系列とその文脈情報を利用することにより文脈依存な解析を実現しており，関数の呼び出し系列の情報が反映されている点で我々の型システムと共通している．本研究の型システムでは，再帰的な使用法表現 $\mu\alpha.U$ を用いて，任意の長さの文脈情報を扱うことができる．制御フロー解析とエフェクトシステムとの類似性は広く認識されており [9], 三つのプログラム解析を比較することは今後の課題である．

5 おわりに

本研究では Igarashi, Kobayashi の資源使用解析 [4] をもとに, 文脈依存な資源使用解析のための型システムを形式化した．使用法表現にアクセスの文脈情報を取り入れることにより，関数内に自由変数として含まれる資源へのアクセスがある場合など，資源へのエイリアスが発生している場合の解析結果が改善される．型健全性に関しては, 現段階では部分型関係に関する補題の証明など一部がまだ完成していない．

解析を自動的に行うためには次の二つのアルゴリズムを与える必要がある．一つはプログラムから使用法表現を抽出するアルゴリズムである．これは [4] と同様な型推論アルゴリズムが与えられると予想している．もう一つは使用法表現が仕様を満たすかどうか ($[\![U]\!] \subseteq \Phi$) を判定するためのアルゴリズムである．

将来の課題として，例外処理や代入機構，データ構造といった，現実の言語にある機構を扱えるよう解析を拡張することがあげられる．このうち，組やレコードのようなデータ構造に関しては, 特別な仕組みを新たに加えることなく型システムを拡張できる．次のような推論規則

$$\frac{\Gamma_1 \vdash M_1 : \tau_1 \qquad \Gamma_2 \vdash M_2 : \tau_2}{\Gamma_1 \otimes \Gamma_2 \vdash (M_1,\ M_2) : \tau_1 \star \tau_2}(\text{T-PAIR}) \qquad \frac{\Gamma \vdash M : \tau_1 \star \tau_2}{\Gamma \vdash \mathbf{fst}(M) : \tau_1}(\text{T-FST}) \qquad \frac{\Gamma \vdash M : \tau_1 \star \tau_2}{\Gamma \vdash \mathbf{snd}(M) : \tau_2}(\text{T-SND})$$

を加えて拡張すると，下のプログラム

```
let p = let y = newFile in ( fun x -> write^l1(y), y) in
   (fst p) @^l2 (); read^l3(snd y) ; (fst p) @^l4 ()
```

の p の型は $(\mathbf{unit} \to \mathbf{bool}, l_2 \otimes l_4) \star (\mathbf{File}, l_3)$，newFile の型は $(\mathbf{File}, (l_2 \otimes l_4)[l_1] \otimes l_3)$ つまり $(\mathbf{File}, l_2.l_1 \otimes l_4.l_1 \otimes l_3)$ となる．

参考文献

[1] Manuel Fähndrich and Robert DeLine. Adoption and focus: practical linear types for imperative programming. In *Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation (PLDI'02)*, pages 13–24, Berlin, Germany, June 2002.

[2] Jeffrey S. Foster, Tachio Terauchi, and Alex Aiken. Flow-sensitive type qualifiers. In *Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation (PLDI'02)*, pages 1–12, Berlin, Germany, June 2002.

[3] Fritz Henglein, Henning Makholm, and Henning Niss. A direct approach to control-flow sensitive region-based memory management. In *Proceedings of the 3rd ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP'01)*, pages 175–186, Florence, Italy, September 2001.

[4] Atsushi Igarashi and Naoki Kobayashi. Resource usage analysis. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 27(2):264–313, 2005.

[5] Suresh Jagannathan and Stephen Weeks. A unified treatment of flow analysis in higher-order languages. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'95)*, pages 393–407, San Francisco, California, January 1995.

[6] Naoki Kobayashi. Time regions and effects for resource usage analysis. In *Proceedings of ACM SIGPLAN International Workshop on Types in Languages Design and Implementation (TLDI'03)*, pages 50–61, New Orleans, Louisiana, January 2003.

[7] Yitzhak Mandelbaum, David Walker, and Robert Harper. An effective theory of type refinements. In *Proceedings of the Eighth ACM SIGPLAN International Conference on Functional programming (ICFP'03)*, pages 213–225, Uppsala, Sweden, August 2003. ACM Press.

[8] Flemming Nielson and Hanne Riis Nielson. Infinitary control flow analysis: a collecting semantics for closure analysis. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'97)*, pages 332–345, Paris, France, January 1997.

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

[10] Olin Shivers. Control flow analysis in Scheme. In *Proceedings of the ACM SIGPLAN 1988 Conference on Programming Language Design and Implementation (PLDI'88)*, pages 164–174, Atlanta, Georgia, June 1988.

[11] Mads Tofte and Jean-Pierre Talpin. Region-based memory management. *Information and Computation*, 111(2):245–296, 1994.

A 定理 3.18 の証明の概略.

補題 A.1 (Inversion)**.** (i) $\Gamma \vdash x : \tau$ ならば $\Gamma \leq x : \tau$ を満たす.

(ii) $\Gamma \vdash \mathbf{true} : \tau$ もしくは $\Gamma \vdash \mathbf{false} : \tau$ ならば $\Gamma \leq \emptyset$, $\tau = \mathbf{bool}$ を満たす.

(iii) $\Gamma \vdash \mathbf{fun}(f, x, M) : \tau$ ならば, ある $\alpha, U_1, U_2, \tau_1, \tau_2, \Gamma_1$ が存在して $\Gamma_1, f : (\tau_1 \to \tau_2, U_1), x : \tau_1 \vdash M : \tau_2$, $\Gamma \leq U_2[\mu\alpha.\ (\Gamma \otimes U_1[\alpha])]$, $(\tau_1 \to \tau_2, U_2) \leq \tau$ を満たす.

(iv) $\Gamma \vdash M_1 \ @^l\ M_2 : \tau$ ならば, ある $\Gamma_1, \Gamma_2, \tau_1, \tau_2, \tau_2'$ が存在して $\Gamma_1 \vdash M_1 : (\tau_1 \to \tau_2, l)$, $\Gamma_2 \vdash M_2 : l[\tau_1]$, $\tau_2 \leq \tau_2'$, ${\tau_2'}^{\Downarrow l}$, $\Gamma \leq \Gamma_1 \otimes \Gamma_2$, $\tau_2' \leq \tau$ を満たす.

(v) $\Gamma \vdash \mathbf{new}^{\Phi}() : \tau$ ならば, ある U が存在して $\Gamma \leq \emptyset$, $[\![U]\!] \subseteq \Phi$, $(\mathbf{R}, U) \leq \tau$ を満たす.

(vi) $\Gamma \vdash \mathbf{acc}^l(M) : \tau$ ならば, ある Γ_1 が存在して $\Gamma_1 \vdash M : (\mathbf{R}, l)$, $\Gamma \leq \Gamma_1$, $\tau = \mathbf{bool}$ を満たす.

(vii) $\Gamma \vdash \mathbf{if}\ M_1\ \mathbf{then}\ M_2\ \mathbf{else}\ M_3 : \tau$ ならば, ある $\Gamma_1, \Gamma_2, \Gamma_3, \tau_1$ が存在して $\Gamma_1 \vdash M_1 : \mathbf{bool}$, $\Gamma_2 \vdash M_2 : \tau_1$, $\Gamma_3 \vdash M_3 : \tau_1$, $\Gamma \leq \Gamma_1 \otimes (\Gamma_2 \,\&\, \Gamma_3)$, $\tau_1 \leq \tau$ を満たす.

(viii) $\Gamma \vdash \langle M \rangle_l : \tau$ ならば, ある Γ_1, τ_1 が存在して $\Gamma_1 \vdash M : \tau_1$, ${\tau_1}^{\Downarrow(\uparrow . l)}$, $\Gamma \leq l[\Gamma_1]$, $l[\tau_1] \leq \tau$ を満たす.

証明. 型付け規則より自明である. □

補題 A.2. 使用法表現に関して, 次が成り立つ.

(1) $\mathbf{0} \otimes U \leq U$

(2) $U_1 \leq U_2$ ならば $[\![U_2]\!] \subseteq [\![U_1]\!]$.

(3) $[\![U]\!] \supseteq [\![l[U]]\!]$

(4) $l[\uparrow [U]] \leq U$

補題 A.3. $\Gamma \vdash v : \tau \iff l[\Gamma] \vdash v : l[\tau]$.

証明. 型付けの導出に関する帰納法. □

補題 A.4 (代入補題)**.** $\Gamma_1, x : \tau_1 \vdash M : \tau_2$ かつ $\Gamma_2 \vdash v : \tau_1$ ならば $\Gamma_1 \otimes \Gamma_2 \vdash [v/x]M : \tau_2$.

証明. 型判断 $\Gamma_1, x : \tau_1 \vdash M : \tau_2$ の導出に関する帰納法による. □

補題 A.5. (a) $\Gamma \vdash \mathcal{E}[\mathbf{fun}(f, x, M_f)\ @^l\ v] : \tau$ ならば $\Gamma \vdash \mathcal{E}[\langle [\mathbf{fun}(f, x, M_f)/f, v/x]M_f \rangle_l] : \tau$.

(b) $\Gamma \vdash \mathcal{E}[\mathbf{if}\ b\ \mathbf{then}\ M_1\ \mathbf{else}\ M_2] : \tau$ ならば $\Gamma \vdash \mathcal{E}[M_i] : \tau$ が成り立つ. ただし, (b, i) は $(\mathbf{true}, 1)$ か $(\mathbf{false}, 2)$ のいずれか.

(c) $\Gamma \vdash \mathcal{E}[\mathbf{new}^{\Phi}()]$ ならば, ある $l_1, \cdots, l_n$ が存在して $\Gamma, z : (\mathbf{R}, l^n[U]) \vdash \mathcal{E}[z]$. ただし, $l^n[U] = l_1[\cdots l_n[U] \cdots]$, z は新しい変数.

(d) $\Gamma \vdash \mathcal{E}[\mathbf{acc}^l(x)] : \tau$ ならばある $\Gamma_1, l_1, \cdots, l_n$ が存在して

$$\begin{array}{l} \Gamma \leq \Gamma_1 \otimes x : (\mathbf{R}, l_1.\cdots.l_n.l) \\ \Gamma_1(x)^{\Downarrow l_1 \ldots l_n . l} \\ \Gamma \setminus \{x\}, x : \Gamma_1(x) \vdash \mathcal{E}[b] : \tau \end{array}$$

(e) $\Gamma \vdash \mathcal{E}[\langle v \rangle_l]$ ならば $\Gamma \vdash \mathcal{E}[v]$.

証明. $\mathcal{E}$ の構造に関する帰納法による. 代表的な場合に対して証明を与える.

(a) の証明.

$\mathcal{E} = [\,]$ のときのみ示す.

$M = \mathbf{fun}(f, x, M_f)\ @^l\ v$, $M' = \langle [\mathbf{fun}(f, x, M_f)/f, v/x]M_f \rangle_l$ とする. 補題 A.1 (iv) より, ある $\Gamma_1, \Gamma_2, \tau_{f_1}, \tau_{f_2}, \tau'_{f_2}$, に対して,

$$\begin{aligned} &\Gamma_1 \vdash \mathbf{fun}(f, x, M_f) : (\tau_{f_1} \to \tau_{f_2}, l) \\ &\Gamma_2 \vdash v : l[\tau_{f_1}] \\ &\Gamma \leq \Gamma_1 \otimes \Gamma_2 \\ &\tau_{f_2} \leq \tau'_{f_2} \\ &{\tau'_{f_2}}^{\Downarrow l} \\ &\tau'_{f_2} \leq \tau \end{aligned} \tag{1}$$

が成り立つ. 更に, 補題 A.1 (iii) より, ある $\Gamma_3, \tau_1, \tau_2, U_1, U_2$ に対して,

$$\begin{aligned} \Gamma_3, f : (\tau_1 \to \tau_2, U_1), x : \tau_1 \vdash M_f : \uparrow[\tau_2] \\ \Gamma_1 \leq U_2[\mu\alpha.\ (\Gamma_3 \otimes (U_1[\alpha]))] \\ (\tau_1 \to \tau_2, U_2) \leq (\tau_{f_1} \to \tau_{f_2}, l) \end{aligned} \tag{2}$$

が成り立つ. これより, $\tau_1 = \tau_{f_1}, \tau_2 = \tau_{f_2}, U_2 \leq l$ である. (2) より,

$$(l[U_1])[\mu\alpha.\ (\Gamma_3 \otimes (U_1[\alpha]))] \vdash \mathbf{fun}(f, x, M_f) : (\tau_1 \to \tau_2, l[U_1]) \tag{3}$$

${\tau'_{f_2}}^{\Downarrow l}$ より $(\uparrow[\tau'_{f_2}])^{\Downarrow(\uparrow.l)}$ がいえるので (2) と $\uparrow[\tau_2] \leq \uparrow[\tau'_{f_2}]$ より,

$$l[\Gamma_3], f : (\tau_1 \to \tau_2, l[U_1]), x : l[\tau_1] \vdash \langle M_f \rangle_l : l[\uparrow[\tau'_{f_2}]] \tag{4}$$

(3), (4), (1) より, 代入補題 A.4 を用いて,

$$l[\Gamma_3] \otimes ((l[U_1])[\mu\alpha.\ (\Gamma_3 \otimes (U_1[\alpha]))]) \otimes \Gamma_2 \vdash \langle [\mathbf{fun}(f, x, M_f)/f, v/x]M_f \rangle_l : l[\uparrow[\tau'_{f_2}]] \tag{5}$$

$\Gamma \leq l[\Gamma_3] \otimes ((l[U_1])[\mu\alpha.\ (\Gamma_3 \otimes (U_1[\alpha]))]) \otimes \Gamma_2$ と補題 A.2 (4) より $\Gamma \vdash \langle [\mathbf{fun}(f, x, M_f)/f, v/x]M_f \rangle_l : \tau$ がいえる.

(c) の証明.

$\mathcal{E} = [\,]$ と $\mathcal{E} = \langle \mathcal{E} \rangle_l$ のときのみ示す.

$\mathcal{E} = [\,]$ のとき.

補題 A.1 (v) より

$$\begin{aligned} &\Gamma_0 \leq \emptyset,\ [\![U]\!] \subseteq \Phi,\ (\mathbf{R}, U) \leq \tau \\ &\Gamma \leq \emptyset \end{aligned}$$

である. これと $z : (\mathbf{R}, U) \vdash z : (\mathbf{R}, U)$ より $\Gamma, z : (\mathbf{R}, U) \vdash z : \tau$

$\mathcal{E} = \langle \mathcal{E} \rangle_l$ のとき. 補題 A.1 (v) より

$$\Gamma_0 \leq \emptyset, [\![U]\!] \subseteq \Phi, (\mathbf{R}, U) \leq \tau$$

である. $\Gamma \vdash \langle \mathcal{E}[\mathbf{new}^\Phi()] \rangle_l : \tau$ と補題 A.1 (viii) より

$$\begin{aligned} &\Gamma_1 \vdash \mathcal{E}[\mathbf{new}^\Phi()] : \tau_1 \\ &\Gamma \leq l[\Gamma_1] \\ &l[\tau_1] \leq \tau, {\tau_1}^{\Downarrow(\uparrow.l)} \end{aligned}$$

が成り立つ. 帰納法の仮定より, ある $l_1, \cdots, l_m$ が存在して, $\Gamma_1, z : (\mathbf{R}, l^m[U]) \vdash \mathcal{E}[z] : \tau_1$ がいえる. これと ${\tau_1}^{\Downarrow(\uparrow.l)}$ より, $l[\Gamma_1], z : (\mathbf{R}, l[l^m[U]]) \vdash \langle \mathcal{E}[z] \rangle_l : l[\tau_1]$ がいえ, $\Gamma, z : (\mathbf{R}, l[l^m[U]]) \vdash \langle \mathcal{E}[z] \rangle_l : \tau$ が導かれる. □

定理 **3.18 の証明**. $(H, M) \rightsquigarrow (H', M')$ の規則に対する場合分けを行う.

(R-New) の場合. 補題 (A.5) (c) より, ある $l_1, \cdots, l_n$, 新しい変数 z について $\Gamma, z : (\mathbf{R}, l_1[l_2 \cdots l_n[U] \cdots]) \vdash \mathcal{E}[z] : \tau$ がいえる. 補題 A.2 (3) より $[\![l_1[l_2 \cdots l_n[U] \cdots]]\!] \subseteq \Phi$. これより $\mathcal{H}(\Gamma, z : (\mathbf{R}, l_1[l_2 \cdots l_n[U] \cdots]), H \uplus \{z \mapsto \Phi\})$ である.

(R-Acc) の場合. 補題 A.5 (d) よりある $\Gamma_1, u = l_1.\cdots.l_n.l$ が存在して

$$\Gamma \leq \Gamma_1 \otimes x : (\mathbf{R}, u) \tag{6}$$

$$\Gamma_1(x)^{\Downarrow u} \tag{7}$$

$$\Gamma \setminus \{x\}, x : \Gamma_1(x) \vdash \mathcal{E}[b] : \tau$$

を満たす. b は **true** か **false** のいずれかである. $\Gamma' = \Gamma \setminus \{x\}, x : \Gamma_1(x), M' = \mathcal{E}[b]$ として $\mathcal{H}(\Gamma', H')$ を導く. (6), (7) より $\Gamma(x) \xrightarrow{u} \Gamma_1(x) \otimes \mathbf{0}$ である. これより $H(x) \supseteq [\![\Gamma(x)]\!] \supseteq \{ls \mid s \in [\![\Gamma_1(x)]\!]\}$ がいえるので $H'(x) \supseteq [\![\Gamma_1(x)]\!]$ がいえる. 従って $\mathcal{H}(\Gamma', H')$ である.

(R-App) の場合. 補題 A.5 (a) による.

(R-Return) の場合. $\Gamma_0 \vdash \langle v \rangle_l : \tau_0$ とする. 補題 A.1 (viii) より, $\Gamma_1 \vdash v : \tau_1$, $\Gamma_0 \leq l[\Gamma_1]$, $l[\tau_1] \leq \tau_0$, ${\tau_1}^{\Downarrow(\uparrow.l)}$ が成り立つ. 補題 A.3 より $l[\Gamma_1] \vdash v : l[\tau_1]$ がいえ, これより $\Gamma_0 \vdash v : \tau_0$. 更に, 補題 A.5 (e) より, $\Gamma \vdash \mathcal{E}[v] : \tau$.

□