情報処理学会論文誌　プログラミング　Vol. 3　No. 4　27–42 (Sep. 2010)

# ポインタのあるプログラミング言語のための資源使用法解析

上野慎平[†1]　小林直樹[†1]　海野広志[†1]

本論文では，C 言語のようなポインタのある言語に対して，計算資源の使用法を検証するための型システムを与える．この型システムに基づいて型検査を行うことにより，ファイルやメモリなどの計算資源が正しい順序でアクセスされることを自動検証することができる．我々の型システムは，小林と末永によって提案された，C プログラムの malloc と free の使用法を検証するための型システムの拡張であり，その特徴は，ポインタや計算資源の型に対して所有権と呼ばれるアクセスの権限と義務を表す情報を付加したことにある．所有権を有理数で表現することにより，型検査の問題を有理数上の線形計画法に帰着し，多項式時間で解くことができる．

# Resource Usage Verification for a Programming Language with Pointers

Shinpei Ueno,[†1] Naoki Kobayashi[†1] and Hiroshi Unno[†1]

We propose a type system for resource usage verification of programs written in a programming language with pointers. By checking that a program is well-typed in the type system, we can verify that the program accesses resources like files and memory cells in a valid manner. Our type system is an extension of Suenaga and Kobayashi's one for checking usage of malloc/free of C programs, and the main novelty is to augment resource types with ownerships, which express both capabilities and obligations to access resources. Thanks to the use of fractions as ownerships, the type checking problem is reduced to linear programming over rational numbers, which can be solved in polynomial time.

## 1. 導　　入

### 1.1 研究背景と目的

今日，交通システムや金融システムなど様々な社会基盤でコンピュータが用いられるようになっており，ソフトウェアの信頼性がますます重要になってきている．そこで，ソフトウェアの信頼性を向上させるため，プログラムを実行前に検証するための様々な手法が提案されている．本論文では，プログラム検証手法の 1 つとして，C 言語のようなポインタを持つ言語を対象とした，ファイルやメモリなどの計算資源の使用法（たとえば開いたファイルをいずれ必ず閉じ，その後は読み書きをしない，などの使用方法）を静的に検証するための手法を提案する．

計算資源使用法の静的検証技術についてはこれまでにも多くの研究がある．たとえば Igarashi and Kobayashi[1] は，関数型言語を対象とした計算資源使用法を検証するための型システムを提案し，Iwama ら[2] は，その手法を例外機構を持つ関数型言語に拡張している．これにより，計算資源使用法検証の問題は，型推論の問題に帰着される．しかしながら，それらの型システムはポインタの入った言語には適用できない．C 言語のようなポインタの入った言語に対する資源使用法の検証としては，Foster ら[3),4)] による CQual があるが，この解析手法は複雑であり，その正しさが証明されていない．本研究では，Suenaga and Kobayashi[5] によるメモリリークの検証のための型システムを拡張することにより，ポインタの入った言語に対する資源使用法検証のための簡潔な型システムを提案し，その正しさを証明する．

### 1.2 ポインタを扱う際の問題点

本節では，ポインタの入った言語に対する計算資源使用法検証を型を用いて行ううえでの問題点について述べ，次節で我々の解決策を述べる．

資源へのアクセス順序を型を用いて解析するための方法としては，資源の型をその状態によって区別することが考えられる．たとえば，読み込み専用のファイル（ポインタ）の場合，**図 1** に示すように，「読み込み可能で，いずれ閉じなければならない状態」$R$ と「すでに閉じた後で，読み書きできない状態」$C$ の 2 つの状態があり，`read`，`close` の操作によって状態遷移が起きる．そこで，前者の状態にあるファイルの型を `file(R)`，後者の状態にある

†1 東北大学
Tohoku University

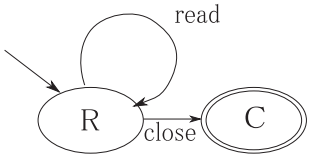


**図 1**　読み込み専用ファイルの仕様
Fig. 1　Specification of read only file.

| | |
|---|---|
| `x = open();` | `x : file(R)` |
| `read(x);` | `x : file(R)` |
| `close(x);` | `x : file(C)` |

**図 2**　単純な型付けの例
Fig. 2　Example of naive typing.

| | |
|---|---|
| `x = open();` | `x : file(R)` |
| `*y = x;` | `x : file(R), y : file(R) ref` |
| `close(*y);` | `x : file(R), y : file(C) ref` |
| `close(x);` | `x : file(C), y : file(C) ref` |

**図 3**　間違った型付けの例
Fig. 3　Example of wrong typing.

ファイルの型を `file(C)` とする．すると，**図 2** の C 言語風のプログラムについては，プログラムの各行について，その右側に示す型が変数 $x$ に割り当てられる．ここで `open` はファイルを読み込み専用で開くための命令であり，1 行目を実行後の $x$ の型は `file(R)` となる．ファイルが状態 $R$ にあるので `read` が可能で，実行後は `file(R)` のままであり，`close` 後に $x$ の型が `file(C)` となる．このように，各状態でファイルに型が割り当てられるので，図 2 のプログラムは正しく資源を使用していることが分かる．

しかしながら，上記のアイデアは，ポインタを用いるプログラムのように，エイリアスが頻繁に現れ，かつ動的に変化する場合には容易には適用できない．たとえば **図 3** のプログラムを考えよう．右側が上記のアイデアに従って各行の実行後の変数に型を割り当てたものである．ここで，$\tau$ ref は，「$\tau$ 型の値へのポインタの型」を表すものとする．2 行目を実行した後には，`y` の指す先は状態 $R$ のファイルなので，型 `file(R) ref` が `y` に割り当てられる．3 行目を実行した後にはそのファイルの状態が $C$ に変化するので，`y` の型は `file(C) ref` となる．4 行目では `x` が閉じられるので，その型は `file(C)` となり，プログラム全体に型を割り当てることができる．しかしながら，図 3 のプログラムは，開いたファイルに対して `*y` と `x` を介して 2 回 `close` を行っており，実際には間違ったプログラムである．このように，単純な型システムでは，ポインタの入ったプログラムが各資源の仕様を満たしているかどうかを正しく判定することができない．

上記の推論の誤りは，3 行目で `x` と `*y` には同じファイルが格納されていることを考慮しておらず，`x` の型を `file(R)` のままにしていることにある．上記のアイデアに基づいて正しく型を推論するためには，このようなエイリアス（別名）に関する情報を型に組み込むことが考えられるが，結果として「ポインタ p とポインタ q が同じメモリを指す**可能性がある**」ことを解析する may エイリアス解析と「ポインタ p とポインタ q が**必ず**同じメモリを指す」ことを解析する must エイリアス解析を包含するような型システムを構築しなければならず，複雑になってしまう．

### 1.3　本論文のアイデア

上記の問題を解決するため，本論文では，計算資源およびポインタの型に，**所有権**を付加する．所有権は 0 から 1 までの有理数であり，資源やポインタに対するアクセスの権限と義務を表す．たとえば変数 $x$ の型が $\texttt{file}_1\texttt{(R)}$ であれば，$x$ に対して `read`，`close` の操作を行う権限があり，また，いずれ `close` を行わなければならないという義務がある．一方，変数 $x$ の型が $\texttt{file}_0\texttt{(R)}$ であれば，$x$ を介してファイルにアクセスすることはできない[*1]．変数 $x$ の型が $\texttt{file}_r\texttt{(R)}$（$0 < r < 1$）であれば，$x$ に対しては，状態を変化させない `read` アクセスのみが許される．同様に，ポインタについては所有権が 1 のときは読み書きおよび解放を，1 未満の正の数のときには読み込みのみを行うことができ，0 のときは読み書きとも行うことができない．なお，有理数の権限または所有権については，近年様々な研究[5)–8)]で用いられているが，本論文で用いる所有権付き型の概念は末永と小林によるメモリリークの検証のための型システムに基づき，それを計算資源の型を加えて拡張したものである．

**図 4** に所有権を用いた型付けの例を示す．1 行目でファイルを新たに開いているので，$x$ には所有権つきのファイルポインタの型 $\texttt{file}_1\texttt{(R)}$ が割り当てられる．したがって，2 行目で $x$ に対して `read` アクセスを行うことができ，型は変化しない．3 行目ではファイルポイ

⋆1 なお，後述の型システムでは，型推論の便宜上，$\texttt{file}(\{\texttt{R} \mapsto r_1, \texttt{C} \mapsto r_2\})$ のように，資源ではなく資源の各状態に対して所有権を割り当てる．$\texttt{file}_1\texttt{(R)}$ は $\texttt{file}(\{\texttt{R} \mapsto 1, \texttt{C} \mapsto 0\})$ に相当する．

| | |
|---|---|
| `x = open();` | $\mathtt{x}:\mathtt{file}_1(\mathtt{R})$ |
| `read(x);` | $\mathtt{x}:\mathtt{file}_1(\mathtt{R})$ |
| `*y = x;` | $\mathtt{x}:\mathtt{file}_0(\mathtt{R}),\mathtt{y}:\mathtt{file}_1(\mathtt{R})\ \mathsf{ref}_1$ |
| `close(*y);` | $\mathtt{x}:\mathtt{file}_0(\mathtt{R}),\mathtt{y}:\mathtt{file}_1(\mathtt{C})\ \mathsf{ref}_1$ |

**図 4**　所有権のある型付けの例
Fig. 4　Example of typing with ownership.

ンタを $x$ からポインタ $y$ の指す先にコピーしている．これにより，変数 $x$ が持っていた所有権が $y$ に移り，$x$ の型は $\mathtt{file}_0(\mathtt{R})$ に変化して $y$ の型は $\mathtt{file}_1(\mathtt{R})\ \mathsf{ref}_1$ となる（なお，この際，所有権を移動させないこともできるし，所有権の一部，たとえば 0.3 のみを移動させることもできる．型検査の際には，全体として型の整合性がとれるように移動すべき所有権を自動推論する）．その結果，4 行目で $y$ を介して `close` アクセスを行うことができ，$y$ の型は $\mathtt{file}_1(\mathtt{C})\ \mathsf{ref}_1$ となる．

一方，図 3 のプログラムを型付けしようとすると，3 行目終了時点の型環境は，上と同様の推論で $\mathtt{x}:\mathtt{file}_0(\mathtt{R})$，$\mathtt{y}:\mathtt{file}_1(\mathtt{C})\ \mathsf{ref}_1$ となり，この時点で $x$ には所有権がないので 4 行目の `close(x)` は不正なアクセスと分かる．

このように，計算資源およびポインタに対するアクセスを所有権を用いて制御することにより，エイリアスが存在するか否かにかかわらず，アクセスの可否およびそれにともなう状態遷移を静的に推論することができる．たとえば変数 $x$ が持つファイルに対する所有権が 1 であれば，仮に同じ資源を保有するエイリアス $p$ が存在したとしても，$p$ が持つ所有権は 0 であり，$p$ を介してファイルにはアクセスすることができない．したがって，エイリアスの有無にかかわらず，所有権 1 を持っていれば，ファイルの状態を変更することができる．

上記のアイデアに基づき，本論文ではポインタと計算資源に関するプリミティブ，および一階の再帰手続きからなる言語に対して，計算資源およびポインタの使用法を検証するための型システムを構築する．この型システムにより，型付けされたプログラムは以下の性質を満たすことが保証される．

- 各計算資源はその仕様どおりにアクセスされる（たとえばファイルの場合は開いたらプログラム終了時までにいずれ必ず閉じられ，その後はアクセスされない）．
- 割り当てられたメモリはいずれ必ず解放され，その後はアクセスされない．

さらに，他の有理数権限に基づく型システム[5),9),10)] と同様，この型システムに対する型推論問題を，線形計画法の問題に帰着することができる．したがって，（所有権を含まない通常の型による注釈付きの）プログラムのサイズに対する多項式時間で型推論を行うことができる．

### 1.4　論文の構成

本論文の構成は以下のとおりである．2 章では，検証の対象となるプログラミング言語を定義する．3 章では，計算資源の使用法検証のための型システムを与え，その正しさを証明する．4 章で，本論文の手法の限界とそれに対する改善策について議論する．5 章で，関連研究について議論し，6 章で，結論および今後の課題について述べる．

## 2. 対 象 言 語

本章では，検証の対象言語として，ポインタおよび計算資源に関するプリミティブからなる単純なプログラミング言語を定義する．構造体やポインタ演算，型キャストは扱っていないが，Suenaga and Kobayashi[5)] と同様の手法により，それらを扱うように拡張できると思われる．

### 2.1　構文の定義

まず，対象言語で扱う計算資源を決定性有限状態オートマトン（から初期状態を除いたもの）を用いて抽象化する．

**定義 1.**（仕様オートマトン）仕様オートマトン $M$ は，4 つ組

$$M = (Q, A, \delta, F)$$

である．ここで，$Q$ は状態の有限集合であり，$A$ はアクセスラベルの有限集合である．$\delta : Q \times A \to Q$ は状態遷移関数であり，$F \subseteq Q$ は受理状態の集合である．

**例 1.** 1 章のファイルに相当する仕様オートマトンは，

$$(\{\mathtt{R}, \mathtt{C}\}, \{\mathtt{read}, \mathtt{close}\}, \delta, \{\mathtt{close}\})$$

によって与えられる．ただし，$\delta$ は以下によって定義される．

$$\delta(\mathtt{R}, \mathtt{read}) = \mathtt{R} \qquad \delta(\mathtt{R}, \mathtt{close}) = \mathtt{C}$$ □

以下では，仕様オートマトンを 1 つ固定し，状態およびアクセスラベルのメタ変数としてそれぞれ $q$ および $a$ を用いる．

**定義 2.**（コマンド，関数定義，プログラム）**コマンド**および**関数定義**の構文を以下の BNF により定義する．

$s$(コマンド) ::= skip | $s_1; s_2$ | let $x = y$ in $s$ | ifnull$(x)$ then $s_1$ else $s_2$ | $F(x_1, \ldots, x_n)$
| $*x \leftarrow y$ | let $x = *y$ in $s$ | let $x =$ null in $s$
| let $x =$ malloc() in $s$ | free$(x)$
| let $x =$ new$^q$() in $s$ | acc$^a(x)$
| assert$(x = y)$ | assert$(x = *y)$

$d$(関数定義) ::= $F(x_1, \ldots, x_n) = s$

**プログラム**とは，関数定義の集合 $D$ とコマンドの組 $(D, s)$ である．ただし，$D = \{F_1(\widetilde{x_1}) = s_1, \ldots, F_m(\widetilde{x_m}) = s_m\}$（$\tilde{x}$ は列 $x_1, \ldots, x_n$ を表す）において，$F_1, \ldots, F_m$ は互いに異なるものとする．

コマンドの定義の 1 行目は通常の制御に関するコマンドである．skip は何もしないコマンドである．コマンド $s_1; s_2$ は，まず $s_1$ を実行し，次に $s_2$ を実行する．コマンド let $x = y$ in $s$ は，$x$ を $y$ の値に束縛し，$s$ を実行する．ifnull$(x)$ then $s_1$ else $s_2$ は，$x$ の値が null ならば $s_1$ を実行し，そうでなければ $s_2$ を実行する．コマンド $F(x_1, \ldots, x_n)$ は，関数呼び出しであり，引数 $x_1, \ldots, x_n$ はそれぞれ互いに異なる変数とする（$F(x, x)$ は，let $y = x$ in $F(x, y)$ と表現できることに注意）．関数に戻り値はなく，参照渡しによって結果を受け渡すものとする．

コマンドの定義の 2〜3 行目はポインタに関するコマンドである．コマンド $*x \leftarrow y$ は，ポインタ $x$ の指す先の値を $y$ の値で書き換える．コマンド let $x = *y$ in $s$ は，ポインタ $y$ の指す先の値に $x$ を束縛し，$s$ を実行する．let $x =$ null in $s$ は，null に $x$ を束縛し，$s$ を実行する．let $x =$ malloc() in $s$ は，メモリセルを確保し，そのメモリセルへのポインタを $x$ に束縛した後に $s$ を実行する．free$(x)$ は，ポインタ $x$ によって参照されるメモリセルを解放する．

コマンドの定義の 4 行目は，計算資源に関するコマンドである．コマンド let $x =$ new$^q$ in $s$ は，状態 $q$ の計算資源を生成し，$x$ をその資源に束縛して $s$ を実行する．コマンド acc$^a(x)$ は，計算資源 $x$ に対して $a$ で表されるアクセスを行う．

コマンドの定義の最後の行は，エイリアスに関する注釈であり，型システムに対するヒントとして用いられる．コマンド assert$(x = y)$ および assert$(x = *y)$ はそれぞれ，$x$ と $y$，$x$ と $*y$ が同じポインタまたは計算資源であることを表明する．これらのアサートコマンドがなくても後述する型システムの健全性には影響を与えないが，適切なエイリアスを挿入することにより，型システムの精度を上げることができる．アサートコマンドはプログラマが挿入してもよいし，must エイリアス解析[11] によって自動挿入することもできる．

なお，上記のコマンドにおいて，変数は関数型プログラミング言語同様，単一代入に基づくものであり，C 言語の変数とは異なることに注意されたい．C 言語の変数 $x$ に対しては，$x$ のアドレスを表す $\&x$ が本論文の言語の変数に相当する．たとえば，C のステートメント列 `int x = 1; x = x+1; ...` は本論文の言語では

let $\&x =$ malloc() in $(*\&x \leftarrow 1;$ let $y = *\&x$ in $(*\&x \leftarrow y + 1; \cdots;$ free$(\&x)))$

に相当する．

### 2.2　操作的意味

前節で与えた言語の操作的意味を定義する．

以下では，ヒープアドレスの集合からなる可算集合 $\mathcal{H}$ の存在を仮定する．

**定義 3.**（値）

$v$(値) ::= $h$ | Res $q$ | null

$h$ はヒープアドレスを表すメタ変数である．Res $q$ は状態 $q$ の資源を表す．以下では，$\mathcal{R} = \{\text{Res } q \mid q \in Q\}$ とする．

**定義 4.**（評価文脈）

$E ::= [\,] \mid E; s$

$E$ の中の $[\,]$ を $s$ で置き換えて得られるコマンドを $E[s]$ と書く．

実行状態を 3 つ組 $\langle H, R, s\rangle$ を用いて表す．$H$ はヒープの状態を表し，$\mathcal{H}$ の有限の部分集合から $\mathcal{H} \cup \mathcal{R} \cup \{\text{null}\}$ への写像，$R$ はレジスタの状態を表し，変数の有限の集合から $\mathcal{H} \cup \{\text{null}\}$ への写像である．3 つ組の集合を **St** とし，**St** $\cup \{\text{Error}, \text{AssertFail}, \text{NullEx}\}$ を **ESt** と表す．

**定義 5.**（遷移関係）$D$ をプログラム，$M$ を仕様オートマトンとする．遷移関係 $\longrightarrow_{D,M} \subseteq$ **St** $\times$ **ESt** は **図 5** および **図 6** の規則を満たす最小の関係である．

以下，図中で用いられている表記を説明する．reference$(H)$ は，ヒープアドレスの集合 $\{h \in \text{dom}(H) | H(h) \in \mathcal{H}\}$ を表す．$f\{x \mapsto v\}$ は，dom$(f') =$ dom$(f) \cup \{x\}$ かつ，$f'(x) = v$ かつ，すべての $y \in$ dom$(f) \setminus \{x\}$ に対して $f'(y) = f(y)$ を満たす写像 $f'$ を表す．$[x'/x]s$ は，$s$ の中の $x$ をすべて $x'$ で置き換えて得られるコマンドを表す．

図 5 は通常の実行のための遷移規則である．下から 4 番目の規則が計算資源の生成のための規則であり，新しいアドレス $h$ に状態 $q$ の資源が割り当てられる．下から 3 番目が計算資源へのアクセスのための規則で，仕様オートマトンの遷移関数に従って計算資源の状態が変化する．

図 6 は不正な命令が実行された場合の遷移規則である．エラー状態は null ポインタへの

$$\frac{}{\langle H, R, E[\mathsf{skip}; s]\rangle \longrightarrow_{D,M} \langle H, R, E[s]\rangle}$$

$$\frac{}{\langle H, R, E[\mathsf{let}\ x = y\ \mathsf{in}\ s]\rangle \longrightarrow_{D,M} \langle H, R\{x' \mapsto R(y)\}, E[[x'/x]s]\rangle}$$

$$\frac{}{\langle H, R\{x \mapsto \mathsf{null}\}, E[\mathsf{ifnull}(x)\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2]\rangle \longrightarrow_{D,M} \langle H, R\{x \mapsto \mathsf{null}\}, E[s_1]\rangle}$$

$$\frac{R(x) \neq \mathsf{null}}{\langle H, R, E[\mathsf{ifnull}(x)\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2]\rangle \longrightarrow_{D,M} \langle H, R, E[s_2]\rangle}$$

$$\frac{F(\tilde{y}) = s \in D}{\langle H, R, E[F(\tilde{x})]\rangle \longrightarrow_{D,M} \langle H, R, E[[\tilde{x}/\tilde{y}]s]\rangle}$$

$$\frac{R(x) \in \mathsf{reference}(H)}{\langle H, R, E[{*x} \leftarrow y]\rangle \longrightarrow_{D,M} \langle H\{R(x) \mapsto R(y)\}, R, E[\mathsf{skip}]\rangle}$$

$$\frac{R(y) \in \mathsf{reference}(H)}{\langle H, R, E[\mathsf{let}\ x = {*y}\ \mathsf{in}\ s]\rangle \longrightarrow_{D,M} \langle H, R\{x' \mapsto H(R(y))\}, E[[x'/x]s]\rangle}$$

$$\frac{}{\langle H, R, E[\mathsf{let}\ x = \mathsf{null}\ \mathsf{in}\ s]\rangle \longrightarrow_{D,M} \langle H, R\{x' \mapsto \mathsf{null}\}, E[[x'/x]s]\rangle}$$

$$\frac{h \notin \mathsf{dom}(H) \qquad v \in \mathcal{H} \cup \{\mathsf{null}\}}{\langle H, R, E[\mathsf{let}\ x = \mathsf{malloc}()\ \mathsf{in}\ s]\rangle \longrightarrow_{D,M} \langle H\{h \mapsto v\}, R\{x' \mapsto h\}, E[[x'/x]s]\rangle}$$

$$\frac{R(x) \in \mathsf{reference}(H) \cup \{\mathsf{null}\}}{\langle H, R, E[\mathsf{free}(x)]\rangle \longrightarrow_{D,M} \langle H\backslash\{R(x)\}, R, E[\mathsf{skip}]\rangle}$$

$$\frac{h \notin \mathsf{dom}(H)}{\langle H, R, E[\mathsf{let}\ x = \mathsf{new}^q()\ \mathsf{in}\ s]\rangle \longrightarrow_{D,M} \langle H\{h \mapsto \mathsf{Res}\ q\}, R\{x' \mapsto h\}, E[[x'/x]s]\rangle}$$

$$\frac{}{\langle H\{R(x) \mapsto \mathsf{Res}\ q\}, R, E[\mathsf{acc}^a(x)]\rangle \longrightarrow_{D,M} \langle H\{R(x) \mapsto \mathsf{Res}\ \delta(q,a)\}, R, E[\mathsf{skip}]\rangle}$$

$$\frac{R(x) = R(y)}{\langle H, R, E[\mathsf{assert}(x = y)]\rangle \longrightarrow_{D,M} \langle H, R, E[\mathsf{skip}]\rangle}$$

$$\frac{R(x) = H(R(y))}{\langle H, R, E[\mathsf{assert}(x = {*y})]\rangle \longrightarrow_{D,M} \langle H, R, E[\mathsf{skip}]\rangle}$$

**図 5**　遷移規則 1
Fig. 5　Transition rules 1.

$$\frac{R(x) = \mathsf{null}}{\langle H, R, E[{*x} \leftarrow y]\rangle \longrightarrow_{D,M} \mathsf{NullEx}}$$

$$\frac{R(y) = \mathsf{null}}{\langle H, R, E[\mathsf{let}\ x = {*y}\ \mathsf{in}\ s]\rangle \longrightarrow_{D,M} \mathsf{NullEx}}$$

$$\frac{R(x) = \mathsf{null}}{\langle H, R, E[\mathsf{acc}^a(x)]\rangle \longrightarrow_{D,M} \mathsf{NullEx}}$$

$$\frac{R(x) \notin \mathsf{reference}(H) \cup \{\mathsf{null}\}}{\langle H, R, E[{*x} \leftarrow y]\rangle \longrightarrow_{D,M} \mathsf{Error}}$$

$$\frac{R(y) \notin \mathsf{reference}(H) \cup \{\mathsf{null}\}}{\langle H, R, E[\mathsf{let}\ x = {*y}\ \mathsf{in}\ s]\rangle \longrightarrow_{D,M} \mathsf{Error}}$$

$$\frac{R(x) \notin \mathsf{reference}(H) \cup \{\mathsf{null}\}}{\langle H, R, E[\mathsf{free}(x)]\rangle \longrightarrow_{D,M} \mathsf{Error}}$$

$$\frac{\delta(q,a)\text{ が未定義}}{\langle H\{R(x) \mapsto \mathsf{Res}\ q\}, R, E[\mathsf{acc}^a(x)]\rangle \longrightarrow_{D,M} \mathsf{Error}}$$

$$\frac{}{\langle H\{R(x) \mapsto h\}, R, E[\mathsf{acc}^a(x)]\rangle \longrightarrow_{D,M} \mathsf{Error}}$$

$$\frac{R(x) \neq R(y)}{\langle H, R, E[\mathsf{assert}(x = y)]\rangle \longrightarrow_{D,M} \mathsf{AssertFail}}$$

$$\frac{R(y) \notin \mathsf{dom}(H) \vee R(x) \neq H(R(y))}{\langle H, R, E[\mathsf{assert}(x = {*y})]\rangle \longrightarrow_{D,M} \mathsf{AssertFail}}$$

**図 6**　遷移規則 2
Fig. 6　Transition rules 2.

アクセスを表す NullEx，計算資源への不正なアクセスなどを表す Error，assert コマンドの表明が成り立たなかったことを表す AssertFail の 3 種類に分けられている．このうち，後述の型システムで防止するのは Error によって表されるエラーである[⋆1]．

上から 3〜5 番目の規則は，計算資源または割り当てられていないアドレスに対してポイ

⋆1 後述の型システムは null ポインタが計算資源としてアクセスされる場合（図 6 の 3 番目の規則）も防止することができる．ただし証明が若干煩雑になるので，ここでは NullEx として扱う．

ンタの操作を行った場合のエラー状態への遷移を表す．6 番目の規則は計算資源の現在の状態で許されていない不正なアクセスを行った場合の遷移を表す．7〜8 番目の規則は誤ってポインタを計算資源としてアクセスした場合の遷移を表す．

## 3. 型システム

本章では，型付けされたプログラムは資源の使用法が仕様を満たすことを保証する型システムを与える．

### 3.1 型

**定義 6.**（型）値型，関数型の集合を以下の構文によって定義する．

$$\begin{array}{lll} \tau(\text{値型}) & ::= & \tau\ \mathsf{ref}_f \mid \mathsf{res}\ g \\ g & ::= & \{q_1 \mapsto f_1, \ldots, q_n \mapsto f_n\} \\ \sigma(\text{関数型}) & ::= & (\tau_1, \ldots, \tau_n) \to (\tau'_1, \ldots, \tau'_n) \\ f(\text{所有権}) & \in & [0,1] \end{array}$$

$\tau\ \mathsf{ref}_f$ は $\tau$ の型を持つ値を指すポインタの型である．所有権 $f$ が 1 のときは，読み書きおよび解放を行うことができ，$0 < f < 1$ のときは，読むことのみができるポインタを表す．$f = 0$ のときは，アクセスできないポインタを表す．$\mathsf{res}\ \{q_1 \mapsto f_1, \ldots, q_n \mapsto f_n\}$ は資源の型で，写像 $g = \{q_1 \mapsto f_1, \ldots, q_n \mapsto f_n\}$ は，各状態 $q_i$ に対する所有権が $f_i$ であることを表す．所有権 $f_i$ が正ならばその資源の状態が $q_i$ であると見なして，状態を変更しないアクセス（たとえばファイルの場合は `read` 操作）を行うことができる．さらに，$f_i = 1$ ならば状態の変更をともなうアクセス（ファイルの場合は `close` 操作）を行うことができる．1 章の例であげた型 $\mathtt{file}_1(\mathtt{R})$ および $\mathtt{file}_1(\mathtt{C})$ は，それぞれ $\mathsf{res}\ \{\mathtt{R} \mapsto 1, \mathtt{C} \mapsto 0\}$ および $\mathsf{res}\ \{\mathtt{R} \mapsto 0, \mathtt{C} \mapsto 1\}$ に相当する．なお，このように資源ではなく，資源の各状態に所有権を割り当てることにより，各資源の状態の推論も含めて線形計画法の問題に帰着することができる．以後，資源の型において所有権が 0 である要素は省略し，たとえば $\mathsf{res}\ \{\mathtt{R} \mapsto 1, \mathtt{C} \mapsto 0\}$ を $\mathsf{res}\ \{\mathtt{R} \mapsto 1\}$ と記す．

関数型 $(\tau_1, \ldots, \tau_n) \to (\tau'_1, \ldots, \tau'_n)$ は，$n$ 引数の関数の型であり，$\tau_1, \ldots, \tau_n$ が関数の呼び出し前の引数の型，$\tau'_1, \ldots, \tau'_n$ が，関数の呼び出し後の引数の型を表す．たとえば，$F(x) = \mathsf{acc}^{\mathtt{close}}(x)$ によって定義される関数 $F$ の型は，

$$(\mathsf{res}\ \{\mathtt{R} \mapsto 1\}) \to (\mathsf{res}\ \{\mathtt{C} \mapsto 1\})$$

である．

値型に関する関係を定義するために，値型の意味を以下のように定義する．$\{0\}^*$ は有限個の記号 0 の列の集合である．

**定義 7.** 値型 $\tau$ に対し，$\{0\}^*$ から $Q \cup \{\mathsf{ref}\} \to [0,1]$ への写像 $[\![\tau]\!]$ を以下にように $\tau$ の構造に関する帰納法で定義する．

$$[\![\mathsf{res}\ g]\!](\epsilon) = g$$
$$[\![\mathsf{res}\ g]\!](0w) = \emptyset$$
$$[\![\tau\ \mathsf{ref}_f]\!](\epsilon) = \{\mathsf{ref} \mapsto f\}$$
$$[\![\tau\ \mathsf{ref}_f]\!](0w) = [\![\tau]\!](w)$$

ただし，$\emptyset$ は，$\forall q \in Q \cup \{\mathsf{ref}\}.g(q) = 0$ となる写像を表す．

直感的には，$[\![\tau]\!](w)$ は，$\tau$ が表すポインタまたは資源からパス $w$ で到達できるポインタまたは資源に対する所有権を表す．たとえば，

$$[\![(\mathsf{res}\ \{\mathtt{R} \mapsto 1\})\ \mathsf{ref}_{0.5}]\!] = \{\epsilon \mapsto \{\mathsf{ref} \mapsto 0.5\}, 0 \mapsto \{\mathtt{R} \mapsto 1\}\}$$

が成り立つ．

上の定義を用い，値型に関する述語および関係をいくつか定義する[⋆1]．

**定義 8.**（well-formedness）すべての $w \in \{0\}^*$，$q \in Q \cup \{\mathsf{ref}\}$ に対して $[\![\tau]\!](w)(\mathsf{ref}) \geq 1/2 \times [\![\tau]\!](w0)(q)$ を満たすとき，$\tau$ は *well-formed* であるという[⋆2]．

特に，$[\![\tau]\!](w) = \emptyset$ ならば，任意の $w' \in \{0\}^*$ について $[\![\tau]\!](ww') = \emptyset$ であることに注意されたい．この条件の必要性については注 1 を参照されたい．以下では，well-formed である型のみを考える．

**定義 9.** すべての $w \in \{0\}^*$ に対して $[\![\tau]\!](w) = \emptyset$ であるとき，$\mathsf{empty}(\tau)$ と書く．また，$[\![\tau_1]\!] = [\![\tau_2]\!]$，$[\![\tau_1]\!] = [\![\tau_2]\!] + [\![\tau_3]\!]$，$[\![\tau_1]\!] + [\![\tau_2]\!] = [\![\tau_3]\!] + [\![\tau_4]\!]$ が成り立つとき，それぞれ $\tau_1 \approx \tau_2$，$\tau_1 \approx \tau_2 + \tau_3$，$\tau_1 + \tau_2 \approx \tau_3 + \tau_4$ と書く．ただし，$[\![\tau_1]\!] + [\![\tau_2]\!]$ は $([\![\tau_1]\!] + [\![\tau_2]\!])(w)(q) = [\![\tau_1]\!](w)(q) + [\![\tau_2]\!](w)(q)$ によって定義する．

以下の 2 つの条件が成り立つとき，$\tau_1$ は $\tau_2$ の部分型と呼び，$\tau_1 \leq \tau_2$ と書く．

(1)　$\forall w \in \{0\}^*,\ q \in F.[\![\tau_1]\!](w)(q) \geq [\![\tau_2]\!](w)(q)$

(2)　$\forall w \in \{0\}^*,\ q \in (Q \setminus F) \cup \{\mathsf{ref}\}.[\![\tau_1]\!](w)(q) = [\![\tau_2]\!](w)(q)$

上の部分型関係 $\tau_1 \leq \tau_2$ は $\tau_1$ 型の値を $\tau_2$ 型の値と見なしてもよいことを表す．たとえば，$\mathtt{C} \in F$ ならば，$\mathsf{res}\ \{\mathtt{C} \mapsto 1\} \leq \mathsf{res}\ \{\mathtt{C} \mapsto 0\}$ が成り立つ．これは，終了状態に対する所

⋆1 以下の関係を型の構造に関する帰納法により定義することも可能であるが，再帰型が入った場合の拡張のしやすさを考慮し，$[\![\tau]\!]$ を用いることにする．

⋆2 係数は必ずしも 1/2 である必要はなく，型システムが健全であるためには正の数であればよい．

有権は権限のみを表していて，（計算資源をいずれ終了状態にしなければならないという）義務はないため，一部の所有権を捨ててしまってもよいことを表す．一方，$\mathtt{R} \notin F$ とすると，$\mathsf{res}\ \{\mathtt{R} \mapsto 1\} \leq \mathsf{res}\ \{\mathtt{R} \mapsto 0\}$ は成り立たない．これは，$\mathsf{res}\ \{\mathtt{R} \mapsto 1\}$ が `read`，`close` の権限だけでなく，`close` アクセスをする義務も表しているからである．

### 3.2 型　判　定

型判定式は $\Theta; \Gamma \vdash s \Rightarrow \Gamma'$ である．関数型環境 $\Theta$ は変数から関数型への有限の写像で，型環境 $\Gamma$ と $\Gamma'$ は変数から値型への有限の写像である．$\Gamma$ は $s$ を評価する前の，各変数の（所有権を含めた）型を表し，$\Gamma'$ は $s$ を評価した後の各変数の型を表す．たとえば $\Theta; x : \mathsf{res}\ \{\mathtt{R} \mapsto 1\} \vdash \mathsf{acc}^{\mathtt{close}}(x) \Rightarrow x : \mathsf{res}\ \{\mathtt{C} \mapsto 1\}$ は，実行前の変数 $x$ の型が $\mathsf{res}\ \{\mathtt{R} \mapsto 1\}$ であれば（すなわち，$x$ を状態 R の計算資源としてアクセスすることが許されているならば），$\mathsf{acc}^{\mathtt{close}}(x)$ を行うことができ，実行後の $x$ の型は $\mathsf{res}\ \{\mathtt{C} \mapsto 1\}$ であることを表す．

**定義 10.** 型判定関係 $\Theta; \Gamma \vdash s \Rightarrow \Gamma'$ は，**図 7** および **図 8** の規則を満たす最小の関係である．

以下に，主要な型付け規則を説明する．

- 規則 T-Assign：$x$ の参照先を書き換えるため，実行前の $x$ の所有権は 1 でなければならない．また，代入により実行前のポインタ $x$ の指す先にはアクセスできなくなるので，$\tau'$ の所有権は空，すなわち $\mathsf{empty}(\tau')$ が成り立たなければならない．一方，$y$ の値は $x$ の指す先にコピーされるため，$\tau$ によって表されるコマンド実行前の $y$ の所有権は，$\tau_1$ と $\tau_2$ に分割され，実行後の $x$ と $y$ に割り当てられる．
- 規則 T-Let*：$y$ の指す先を参照するため，$y$ の所有権 $f$ は 0 より大きくなくてはならない．$y$ の指す先の値が $x$ にコピーされるため，その値の所有権を表す $\tau$ は $\tau_1$ と $\tau_2$ に分割され，実行後の $x$ と $y$ に割り当てられる．さらに，$s$ の実行後は変数 $x$ にはアクセスできないため，そのときの $x$ の値の所有権は空でなければならない．
- 規則 T-Free：実行前の $x$ の所有権は 1，実行後は 0 でなければならない．また，T-Assign と同様，$x$ の指す先にはアクセスできなくなるため，$\mathsf{empty}(\tau)$ が成り立たなければならない．
- 規則 T-New：$x$ は状態 $q$ の計算資源に束縛されるため，$s$ を実行前の $x$ の型は $\mathsf{res}\ \{q \mapsto 1\}$ である．また，$s$ の実行後は $x$ を参照できなくなるので，実行後の $x$ の型 $\tau$ は空でなければならない．
- 規則 T-Acc：仮定 $\Delta(g, g', a)$ は以下の条件がすべて成り立つことを表す．

（1）　$\sum_{q \in \{q' \mid \delta(q', a)\ \text{is defined}\}} g(q) > 0.$

$$\frac{}{\Theta; \Gamma \vdash \mathsf{skip} \Rightarrow \Gamma} \quad \text{(T-Skip)}$$

$$\frac{\Theta; \Gamma \vdash s_1 \Rightarrow \Gamma'' \qquad \Theta; \Gamma'' \vdash s_2 \Rightarrow \Gamma'}{\Theta; \Gamma \vdash s_1; s_2 \Rightarrow \Gamma'} \quad \text{(T-Seq)}$$

$$\frac{\Theta; \Gamma, x : \tau_1, y : \tau_2 \vdash s \Rightarrow \Gamma', x : \tau_1' \qquad \tau \approx \tau_1 + \tau_2 \qquad \mathsf{empty}(\tau_1')}{\Theta; \Gamma, y : \tau \vdash \mathsf{let}\ x = y\ \mathsf{in}\ s \Rightarrow \Gamma'} \quad \text{(T-Let)}$$

$$\frac{\Theta; \Gamma, x : \tau'\ \mathsf{ref}_{f'} \vdash s_1 \Rightarrow \Gamma' \qquad \Theta; \Gamma, x : \tau\ \mathsf{ref}_f \vdash s_2 \Rightarrow \Gamma'}{\Theta; \Gamma, x : \tau\ \mathsf{ref}_f \vdash \mathsf{ifnull}(x)\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2 \Rightarrow \Gamma'} \quad \text{(T-If)}$$

$$\frac{\Theta(f) = (\tilde{\tau}) \to (\tilde{\tau}')}{\Theta; \Gamma, \tilde{x} : \tilde{\tau} \vdash F(\tilde{x}) \Rightarrow \Gamma, \tilde{x} : \tilde{\tau}'} \quad \text{(T-Call)}$$

$$\frac{\tau \approx \tau_1 + \tau_2 \qquad \mathsf{empty}(\tau')}{\Theta; \Gamma, x : \tau'\ \mathsf{ref}_1, y : \tau \vdash *x \leftarrow y \Rightarrow \Gamma, x : \tau_1\ \mathsf{ref}_1, y : \tau_2} \quad \text{(T-Assign)}$$

$$\frac{\Theta; \Gamma, x : \tau_1, y : \tau_2\ \mathsf{ref}_f \vdash s \Rightarrow \Gamma', x : \tau_1' \qquad f > 0 \qquad \tau \approx \tau_1 + \tau_2 \qquad \mathsf{empty}(\tau_1')}{\Theta; \Gamma, y : \tau\ \mathsf{ref}_f \vdash \mathsf{let}\ x = *y\ \mathsf{in}\ s \Rightarrow \Gamma'} \quad \text{(T-Let*)}$$

$$\frac{\Theta; \Gamma, x : \tau\ \mathsf{ref}_f \vdash s \Rightarrow \Gamma', x : \tau'}{\Theta; \Gamma \vdash \mathsf{let}\ x = \mathsf{null}\ \mathsf{in}\ s \Rightarrow \Gamma'} \quad \text{(T-Null)}$$

$$\frac{\Theta; \Gamma, x : \tau\ \mathsf{ref}_1 \vdash s \Rightarrow \Gamma', x : \tau'\ \mathsf{ref}_0 \qquad \mathsf{empty}(\tau) \qquad \mathsf{empty}(\tau')}{\Theta; \Gamma \vdash \mathsf{let}\ x = \mathsf{malloc}()\ \mathsf{in}\ s \Rightarrow \Gamma} \quad \text{(T-Malloc)}$$

$$\frac{\mathsf{empty}(\tau)}{\Theta; \Gamma, x : \tau\ \mathsf{ref}_1 \vdash \mathsf{free}(x) \Rightarrow \Gamma, x : \tau\ \mathsf{ref}_0} \quad \text{(T-Free)}$$

$$\frac{\Theta; \Gamma, x : \mathsf{res}\ \{q \mapsto 1\} \vdash s \Rightarrow \Gamma', x : \tau \qquad \mathsf{empty}(\tau)}{\Theta; \Gamma \vdash \mathsf{let}\ x = \mathsf{new}^q()\ \mathsf{in}\ s \Rightarrow \Gamma'} \quad \text{(T-New)}$$

$$\frac{\Delta(g, g', a)}{\Theta; \Gamma, x : \mathsf{res}\ g \vdash \mathsf{acc}^a(x) \Rightarrow \Gamma, x : \mathsf{res}\ g'} \quad \text{(T-Acc)}$$

$$\frac{\Gamma \leq \Gamma_1 \qquad \Gamma_1' \leq \Gamma' \qquad \Theta; \Gamma_1 \vdash s \Rightarrow \Gamma_1'}{\Theta; \Gamma \vdash s \Rightarrow \Gamma'} \quad \text{(T-Sub)}$$

**図 7**　型付け規則 1
Fig. 7　Typing rules 1.

$$\frac{\tau_1 + \tau_2 \approx \tau_1' + \tau_2'}{\Theta; \Gamma, x : \tau_1, y : \tau_2 \vdash \mathsf{assert}(x = y) \Rightarrow \Gamma, x : \tau_1', y : \tau_2'} \text{(T-Assert)}$$

$$\frac{\tau_1 + \tau_2 \approx \tau_1' + \tau_2'}{\Theta; \Gamma, x : \tau_1, y : \tau_2\ \mathsf{ref}_f \vdash \mathsf{assert}(x = *y) \Rightarrow \Gamma, x : \tau_1', y : \tau_2'\ \mathsf{ref}_f} \text{(T-Assert*)}$$

$$\frac{\Theta; \tilde{x} : \tilde{\tau} \vdash s \Rightarrow \tilde{x} : \tilde{\tau'} \qquad \Theta(f) = \tilde{\tau} \to \tilde{\tau'}\ (\text{for each } F(\tilde{x}) = s \in D) \quad \mathsf{dom}(\Theta) = \mathsf{dom}(D)}{\vdash D : \Theta} \text{(T-Def)}$$

$$\frac{\vdash D : \Theta \qquad \Theta; \emptyset \vdash s \Rightarrow \emptyset}{\vdash (D, s)} \text{(T-Prog)}$$

**図 8**　型付け規則 2
Fig. 8　Typing rules 2.

(2)　すべての $q' \in Q$ に対して，$g'(q') = \sum_{q \in \{q \mid \delta(q,a) = q'\}} g(q)$.
(3)　$\delta(q, a) = q' \wedge q \neq q'$ を満たす $q$，$q'$ が存在するならば，$\sum_{q \in Q} g(q) \geq 1$.

(1) は，計算資源にアクセスするには，所有権は 0 より大きい必要があることを意味している．(2) は，所有権が遷移先の状態へ移動することを意味している．(3) は，アクセス $a$ により状態が遷移する可能性があるならば，所有権は 1 必要であることを意味している．

- 規則 T-Sub：$\Gamma_1 \leq \Gamma_2$ は，$dom(\Gamma_1) = dom(\Gamma_2)$ かつ $\forall x \in dom(\Gamma_1).\Gamma_1(x) \leq \Gamma_2(x)$ が成り立つことを表す．この規則を用いて，たとえば $x{:}\mathsf{res}\ \{\mathtt{C} \mapsto 0\} \vdash s \Rightarrow x{:}\mathsf{res}\ \{\mathtt{C} \mapsto 1\}$ から $x : \mathsf{res}\ \{\mathtt{C} \mapsto 1\} \vdash s \Rightarrow x : \mathsf{res}\ \{\mathtt{C} \mapsto 0\}$ を導くことができる．

**例 2.** 型付けの例を示す．以下のコマンド $s_1$ を考える．

$s = *y \leftarrow x; \mathsf{let}\ z = *y\ \mathsf{in}\ s'$
$s' = \mathsf{acc}^{\mathtt{read}}(z); \mathsf{assert}(x = z); \mathsf{acc}^{\mathtt{close}}(x)$

コマンド $s_1$ の実行前の型環境が $x : \mathsf{res}\ \{\mathtt{R} \mapsto 1\}$，$y : (\mathsf{res}\ \emptyset)\ \mathsf{ref}_1$ のとき，$s$ の型付けは **図 9** の $\Pi_0$ のようになる．ただし，$\Gamma_1$ から $\Gamma_4$ を以下のように定める．

$\Gamma_1 = x : \mathsf{res}\ \{\mathtt{R} \mapsto 1\}, y : (\mathsf{res}\ \emptyset)\ \mathsf{ref}_1, z : \mathsf{res}\ \emptyset$
$\Gamma_2 = x : \mathsf{res}\ \emptyset, y : (\mathsf{res}\ \{\mathtt{R} \mapsto 1\})\ \mathsf{ref}_1, z : \mathsf{res}\ \emptyset$
$\Gamma_3 = x : \mathsf{res}\ \emptyset, y : (\mathsf{res}\ \emptyset)\ \mathsf{ref}_1, z : \mathsf{res}\ \{\mathtt{R} \mapsto 1\}$
$\Gamma_4 = x : \mathsf{res}\ \{\mathtt{C} \mapsto 1\}, y : (\mathsf{res}\ \emptyset)\ \mathsf{ref}_1, z : \mathsf{res}\ \emptyset$

$$\Pi_0 = \dfrac{\Pi_1 \quad \dfrac{\Pi_2 \quad 1 > 0 \quad (\mathsf{res}\ \{\mathtt{R} \mapsto 1\}) \approx \mathsf{res}\ \emptyset + \mathsf{res}\ \{\mathtt{R} \mapsto 1\} \quad \mathsf{empty}(\mathsf{res}\ \emptyset)}{\Theta; x : \mathsf{res}\ \emptyset, y : (\mathsf{res}\ \{\mathtt{R} \mapsto 1\})\ \mathsf{ref}_1 \vdash \mathsf{let}\ z = *y\ \mathsf{in}\ s' \Rightarrow x : \mathsf{res}\ \{\mathtt{C} \mapsto 1\}, y : (\mathsf{res}\ \emptyset)\ \mathsf{ref}_1}\ \text{(T-Let*)}}{\Theta; x : \mathsf{res}\ \{\mathtt{R} \mapsto 1\}, y : (\mathsf{res}\ \emptyset)\ \mathsf{ref}_1 \vdash s \Rightarrow x : \mathsf{res}\ \{\mathtt{C} \mapsto 1\}, y : (\mathsf{res}\ \emptyset)\ \mathsf{ref}_1}\ \text{(T-Seq)}$$

$$\Pi_1 = \frac{(\mathsf{res}\ \{\mathtt{R} \mapsto 1\}) \approx (\mathsf{res}\ \{\mathtt{R} \mapsto 1\}) + (\mathsf{res}\ \emptyset) \quad \mathsf{empty}(\mathsf{res}\ \emptyset)}{\Theta; x : \mathsf{res}\ \{\mathtt{R} \mapsto 1\}, y : (\mathsf{res}\ \emptyset)\ \mathsf{ref}_1 \vdash *y \leftarrow x \Rightarrow x : \mathsf{res}\ \emptyset, y : (\mathsf{res}\ \{\mathtt{R} \mapsto 1\})\ \mathsf{ref}_1}\ \text{(T-Assign)}$$

$$\Pi_2 = \dfrac{\Pi_3 \quad \dfrac{\Pi_4 \quad \Pi_5}{\Theta; \Gamma_3 \vdash \mathsf{assert}(x = z); \mathsf{acc}^{\mathtt{close}}(x) \Rightarrow \Gamma_4}\ \text{(T-Seq)}}{\Theta; \Gamma_3 \vdash s' \Rightarrow \Gamma_4}\ \text{(T-Seq)}$$

$$\Pi_3 = \frac{\Delta(\{\mathtt{R} \mapsto 1\}, \{\mathtt{R} \mapsto 1\}, \mathtt{read})}{\Theta; \Gamma_3 \vdash \mathsf{acc}^{\mathtt{read}}(z) \Rightarrow \Gamma_3}\ \text{(T-Acc)}$$

$$\Pi_4 = \frac{(\mathsf{res}\ \emptyset) + (\mathsf{res}\ \{\mathtt{R} \mapsto 1\}) \approx (\mathsf{res}\ \{\mathtt{R} \mapsto 1\}) + (\mathsf{res}\ \emptyset)}{\Theta; \Gamma_3 \vdash \mathsf{assert}(x = z) \Rightarrow \Gamma_1}\ \text{(T-Assert)}$$

$$\Pi_5 = \frac{\Delta(\{\mathtt{R} \mapsto 1\}, \{\mathtt{C} \mapsto 1\}, \mathtt{close})}{\Theta; \Gamma_1 \vdash \mathsf{acc}^{\mathtt{close}}(x) \Rightarrow \Gamma_4}\ \text{(T-Acc)}$$

**図 9**　型付けの例
Fig. 9　Example of typing.

$\Pi_5$ の $\Delta(\{\mathtt{R} \mapsto 1\}, \{\mathtt{C} \mapsto 1\}, \mathtt{close})$ について説明する．条件 (1) は，R の所有権が 1 であることから満たされる．条件 (2) は，$\delta(\mathtt{R}, \mathtt{close}) = \mathtt{C}$ であるので満たされる．条件 (3) は，close は状態遷移する可能性があるアクセスであるため，所有権が 1 必要であるが，R の所有権が 1 のため満たされる．なお，上の例から $\mathsf{assert}(x = z)$ を削除すると型付けできない．これは，$\mathsf{acc}^{\mathtt{read}}(z)$ の実行後の型環境は

$x : \mathsf{res}\ \{\mathtt{R} \mapsto r_x\}, y : (\mathsf{res}\ \{\mathtt{R} \mapsto r_y\})\ \mathsf{ref}_1, z : \mathsf{res}\ \{\mathtt{R} \mapsto r_z\}$

の形で，$r_x + r_y = r_z = 1$，$r_z > 0$ が成り立たなければならないためである（条件 $r_z > 0$ は，$z$ が読み込み可能という要請から導かれる）．したがって，$r_z < 1$ であり，$\mathsf{acc}^{\mathtt{close}}(x)$ の部分が型付けできない．　□

**例 3.** 有理数の所有権が必要になる例を示す．以下のコマンド $s_2$ を考える．

$s_2 = \mathsf{let}\ x = \mathsf{new}^{\mathtt{R}}\ \mathsf{in}\ \mathsf{let}\ y = x\ \mathsf{in}\ s_2'$
$s_2' = \mathsf{acc}^{\mathtt{read}}(x); \mathsf{acc}^{\mathtt{read}}(y); \mathsf{assert}(x = y); \mathsf{acc}^{\mathtt{close}}(x)$

$s_2'$ は以下のように型付けできる．

$\Theta; x : \mathsf{res}\ \{\mathtt{R} \mapsto 0.4\}, y : \mathsf{res}\ \{\mathtt{R} \mapsto 0.6\} \vdash s_2' \Rightarrow x : \mathsf{res}\ \{\mathtt{C} \mapsto 1\}, y : \mathsf{res}\ \{\mathtt{close} \mapsto 0\}$

$s_2'$ 中で $x$，$y$ とも read に用いられるため，$s_2'$ 実行前の型環境における $x$ と $y$ の read 操作に関する所有権はともに正でなければならない．したがって，所有権が 0，1 の整数のみであれば型付けできない．

なお，$\mathsf{acc}^{\mathtt{read}}(x)$ と $\mathsf{acc}^{\mathtt{read}}(y)$ の間に $\mathsf{assert}(x = y)$ を挿入すれば，所有権として 0，1 の整数のみを用いて型付け可能である．一般に，本論文で扱うような逐次言語においては，有理数の所有権を用いて型付けできるプログラムは，assert 文を適切に挿入することによって，整数の所有権のみを用いて型付け可能にできるものと思われる．したがって，本論文の型システムにおいて有理数の所有権を用いることによる主要な利点は，型判定問題が（多項式時間アルゴリズムが存在する）有理数上の線形不等式の充足可能性判定問題に帰着できることにある．□

**注 1.** 値型の well-formedness の条件は，$\tau\ \mathsf{ref}_0$ のように所有権が 0 であるポインタの中身の型が空（つまり $\mathsf{empty}(\tau)$）であることを保証するためのものである．この条件がないと，$\tau\ \mathsf{ref}_0$ 中の $\tau$ に含まれる所有権によって表される権限や義務を用いる前に，別名を通してポインタの中身を書き換えられてしまう恐れがある．たとえば，以下のような（誤った）型判定を考えよう．

$$x : (\mathsf{res}\ \{\mathtt{R} \mapsto 1\})\ \mathsf{ref}_1, z : \mathsf{res}\ \{\mathtt{C} \mapsto 1\} \vdash$$
$$\mathsf{let}\ y = x\ \mathsf{in}\ (*y \leftarrow z; \mathsf{assert}(x = y); \mathsf{let}\ u = *x\ \mathsf{in}\ \mathsf{acc}^{\mathtt{read}}(u); \mathsf{assert}(u = *x))$$
$$\Rightarrow x : (\mathsf{res}\ \{\mathtt{C} \mapsto 1\})\ \mathsf{ref}_1, z : \mathsf{res}\ \{\mathtt{C} \mapsto 1\}$$

実行前の型環境は，ポインタ $x$ の指す先には（read アクセスが可能な）状態 R の資源が，変数 $z$ には状態 C の資源が格納されていることを表す．コマンド中では $x$ の別名 $y$ を介して $x$ の指す先に $z$ が書き込まれ，その資源に対して $\mathsf{acc}^{\mathtt{read}}(u)$ で read アクセスを行おうとしているので，意味的には間違った型判定である．しかしながら，well-formedness の条件がないと，$\mathsf{let}\ y = x\ \mathsf{in} \cdots$ において $x$ の型を $x : (\mathsf{res}\ \{\mathtt{R} \mapsto 1\})\ \mathsf{ref}_0, y : (\mathsf{res}\ \{\mathtt{R} \mapsto 0\})\ \mathsf{ref}_1$ に分割することができる．これにより，$\mathsf{assert}(x = y)$ 実行の前後の型環境としてそれぞれ

$$x : (\mathsf{res}\ \{\mathtt{R} \mapsto 1\})\ \mathsf{ref}_0, y : (\mathsf{res}\ \{\mathtt{C} \mapsto 0\})\ \mathsf{ref}_1, z : \mathsf{res}\ \{\mathtt{C} \mapsto 1\}$$

および

$$x : (\mathsf{res}\ \{\mathtt{R} \mapsto 1\})\ \mathsf{ref}_1, y : (\mathsf{res}\ \{\mathtt{C} \mapsto 0\})\ \mathsf{ref}_0, z : \mathsf{res}\ \{\mathtt{C} \mapsto 1\}$$

を割り当てることにより，上記の型判定が導出できてしまう．□

### 3.3 型システムの健全性

上記の型システムは以下のような健全性を満たす．

**定理 1.** $\vdash (D, s)$ ならば，次の 2 つの性質が成り立つ．

(1) $\langle \emptyset, \emptyset, s\rangle \not\longrightarrow^*_{D,M} \mathsf{Error}$

(2) $\langle \emptyset, \emptyset, s\rangle \longrightarrow^*_{D,M} \langle H, R, \mathsf{skip}\rangle$ ならば，すべての $h \in \mathsf{dom}(H)$ に対して，ある $q \in Q$ が存在して，$H(h) = \mathsf{Res}\ q \wedge q \in F$ である．

1 つ目の性質は，計算資源やメモリに対する不正なアクセスがないことを意味する．2 つ目の性質は，プログラム終了時に，計算資源は仕様で行わなければならないとされたアクセスをすべて行っており，メモリはすべて解放されていることを意味する．

健全性（定理 1）を証明するために，簡約によって保存される所有権に関する不変条件 $\mathsf{Con}(\Gamma, H, R)$ を用意する．

まず，$\{0\}^*$ から $\mathcal{H} \cup \{\mathsf{null}\}$ への写像 $[\![H, v]\!]$ を以下のように定義する．

$$[\![H, \mathsf{null}]\!](\epsilon) = \mathsf{null}$$
$$[\![H, h]\!](\epsilon) = h$$
$$[\![H, \mathsf{Res}\ q]\!](\epsilon) = \mathsf{null}$$
$$[\![H, v]\!](w0) = \begin{cases} H(h) & \text{if } [\![H, v]\!](w) = h \in \mathsf{reference}(H) \\ \mathsf{null} & \text{otherwise} \end{cases}$$

これを用いて，$\mathsf{ownall}(H, v, \tau)$ を次のように定義する．

$$\mathsf{own}(H, v, \tau)(w) = \begin{cases} \{[\![H, v]\!](w) \mapsto [\![\tau]\!](w)\} & \text{if } [\![H, v]\!](w) \neq \mathsf{null} \\ \emptyset & \text{otherwise} \end{cases}$$
$$\mathsf{ownall}(H, v, \tau) = \textstyle\sum_{w \in \{0\}^*} \mathsf{own}(H, v, \tau)(w)$$

ここで，関数の和 $f_0 + f_1$ は，$\forall x \in \mathsf{dom}(f_0) \cap \mathsf{dom}(f_1).(f_0 + f_1)(x) = f_0(x) + f_1(x)$ かつ $\forall x \in \mathsf{dom}(f_i) \setminus \mathsf{dom}(f_{1-i}).(f_0 + f_1)(x) = f_i(x)$ と定義される．最後の行の $\Sigma$ は，この関数の和に関する総和演算を表す．

上で定義した $\mathsf{ownall}(H, v, \tau)$ は，直観的には，型 $\tau$ を持つ値 $v$ が所持する，各ヒープアドレスおよび計算資源に対する所有権を表す．たとえば，$H = \{h_1 \mapsto h_2, h_2 \mapsto \mathsf{Res}\ \mathtt{R}\}$ であれば，

$$\mathsf{ownall}(H, h_1, \mathsf{res}\ \{\mathtt{R} \mapsto 1\}\ \mathsf{ref}_{0.5}) = \{h_1 \mapsto \{\mathsf{ref} \mapsto 0.5\}, h_2 \mapsto \{\mathtt{R} \mapsto 1\}\}$$

である．これは，$\mathsf{res}\ \{\mathtt{R} \mapsto 1\}\ \mathsf{ref}_{0.5}$ 型の値 $h_1$ は，$h_1$ そのものに対する所有権 0.5 だけでなく，そこから参照される計算資源 $h_2$ の状態 R に対する所有権 1 を有することを意味する．

$\Gamma$，$H$，$R$ が以下の 2 つの条件を満たすとき，$\mathsf{Con}(\Gamma, H, R)$ と書く．直観的には，型環境 $\Gamma$ によって表される各値の持つ所有権の総和が，現在のヒープの状態と一致する（すなわち，まだ解放されていないヒープアドレス，終了状態にない計算機資源それぞれに対し，所有権の総和は 1 である）ことを表す．

(1)　$\mathsf{dom}(\Gamma) = \mathsf{dom}(R)$

(2)　$O = \sum_{x \in \mathtt{dom}(\Gamma)} .\mathsf{ownall}(H, R(x), \Gamma(x))$. とすると，以下の 2 つの条件が成り立つ．

(i) すべての $h \in \mathsf{dom}(H)$ について，次のいずれかが成り立つ．

- $\exists h' \in \mathcal{H} \cup \{\mathsf{null}\}.H(h) = h' \wedge O(h) = \{\mathsf{ref} \mapsto 1\}$
- $\exists q \in Q.\exists f \in [0,1].H(h) = \mathsf{Res}\ q \wedge O(h) = \{q \mapsto f\} \wedge (f = 1 \vee q \in F)$

(ii) すべての $h \in (\mathsf{dom}(O) \setminus \mathsf{dom}(H))$ について，$O(h) = \emptyset$.

次の補題は，プログラムが型付けされていて，かつ条件 $\mathsf{Con}(\Gamma, H, R)$ が成立すれば，すぐには Error に到達しないことを表す．

**補題 1.** $\Theta; \Gamma \vdash s \Rightarrow \Gamma''$ かつ $\vdash D : \Theta$ かつ $\mathsf{Con}(\Gamma, H, R)$ が成り立つならば，$\langle H, R, s\rangle \not\longrightarrow_{D,M} \mathsf{Error}$

**証明**　$\Theta; \Gamma \vdash s \Rightarrow \Gamma''$，$\vdash D : \Theta$ および $\mathsf{Con}(\Gamma, H, R)$ を仮定する．

$\langle H, R, s\rangle \longrightarrow_{D,M} \mathsf{Error}$ となりうるのは $s$ が $E[*x \leftarrow y], E[\mathsf{free}(x)], E[\mathsf{let}\ y = *x\ \mathsf{in}\ s'], E[\mathsf{acc}^a(x)]$ のいずれかで，$R(x) \neq \mathsf{null}$ の場合のみである．

s が $E[*x \leftarrow y]$，$E[\mathsf{free}(x)]$，$E[\mathsf{let}\ y = *x\ \mathsf{in}\ s']$ のいずれかの場合，T-Assign，T-Free，T-Let*，T-Sub より，$\exists \tau,\ k.\Gamma(x) \leq \tau\ \mathsf{ref}_k \wedge k > 0$ が成り立つ．$\mathsf{Con}(\Gamma, H, R)$ の条件 (2) より，あるヒープアドレス $h'$ が存在して $H(R(x)) = h'$ が成り立つ．よって，$\langle H, R, s\rangle \not\longrightarrow_{D,M} \mathsf{Error}$.

$s$ が $E[\mathsf{acc}^a(x)]$ のとき，T-Acc，T-Sub より，$\Gamma(x) \leq \mathsf{res}\ g$ かつ $\Delta(g, g', a)$ が成り立つ．$\Delta(g, g', a)$ の定義の条件 (1) より，ある $q$，$q'$ が存在して $g(q) > 0$ かつ $\delta(q, a) = q'$ が成り立つ．したがって $\Gamma(x) \leq \mathsf{res}\ g$ より $[\![\Gamma(x)]\!](\epsilon)(q) > 0$. ownall の定義より，$\mathsf{ownall}(H, R(x), \Gamma(x))(R(x))(q) > 0$ が成り立つ．よって，$\mathsf{Con}(\Gamma, H, R)$ の条件 (2) より，$H(R(x)) = \mathsf{Res}\ q$ が成り立つので $\langle H, R, s\rangle \not\longrightarrow_{D,M} \mathsf{Error}$. □

すべての $x \in \mathsf{dom}(\Gamma)$ について，$R(x) = \mathsf{null}$ または $\forall w \in \{0\}^*.\forall q \in (Q \setminus F) \cup \{\mathsf{ref}\}.[\![\Gamma(x)]\!](w)(q) = 0$ ならば $\mathsf{empty}_R(\Gamma)$ と書く．次の補題は，$\mathsf{empty}_R(\Gamma)$ を満たす型環境 $\Gamma$ によって表される状態では，すべてのメモリが解放され，計算機資源は終了状態にあることを意味する．

**補題 2.** $\mathsf{empty}_R(\Gamma)$ かつ $\mathsf{Con}(\Gamma, H, R)$ ならば，$\forall h \in \mathsf{dom}(H).\exists q \in Q.H(h) = \mathsf{Res}\ q \wedge q \in F$ が成り立つ．

**証明**　$\mathsf{empty}_R(\Gamma)$，$\mathsf{Con}(\Gamma, H, R)$ かつ $h \in dom(H)$ が成り立つと仮定する．

$\mathsf{empty}_R(\Gamma)$ と ownall の定義より，$\mathsf{Con}(\Gamma, H, R)$ の条件 (2) の $O$ は，$\forall q \in (Q \setminus F) \cup \{\mathsf{ref}\}.O(h)(q) = 0$ を満たす．したがって，$\mathsf{Con}(\Gamma, H, R)$ の条件 (2) より，$H(h) = \mathsf{Res}\ q$ かつ $q \in F$ が成り立つ．□

次の補題は簡約の過程で型付けおよび不変条件 $\mathsf{Con}(\Gamma, H, R)$ が成り立つことを表す．

**補題 3.**（type preservation）$\Theta; \Gamma \vdash s \Rightarrow \Gamma''$ と $\vdash D : \Theta$，$\mathsf{Con}(\Gamma, H, R)$ を仮定する．もし，$\langle H, R, s\rangle \longrightarrow_{D,M} \langle H', R', s'\rangle$ および $(\mathsf{dom}(R') \setminus \mathsf{dom}(R)) \cap \mathsf{dom}(\Gamma) = \emptyset$ が成り立つならば，$\Gamma'$ と $\Gamma_0$ が存在して，$\Theta; \Gamma' \vdash s' \Rightarrow \Gamma'', \Gamma_0$，$\mathsf{Con}(\Gamma', H', R')$ および $\mathsf{empty}_{R'}(\Gamma_0)$ が成り立つ．

**証明**　付録を参照．□

以上の補題を用いて定理 1 を証明する．

**定理 1 の証明**

$\vdash (D, s)$ を仮定する．T-Prog より，$\Theta$ が存在して，$\vdash D : \Theta$ および $\Theta; \emptyset \vdash s \Rightarrow \emptyset$ が成り立つ．また，$\mathsf{Con}(\emptyset, \emptyset, \emptyset)$ が成り立つ．

1 つ目の性質について証明する．$\langle \emptyset, \emptyset, s\rangle \longrightarrow^*_{D,M} \langle H', R', s'\rangle$ ならば $\langle H', R', s'\rangle \not\longrightarrow_{D,M} \mathsf{Error}$ が成り立つことをいえばよい．補題 3 より，$\Theta; \Gamma_1 \vdash s' \Rightarrow \Gamma''$，$\Gamma_0$，$\mathsf{Con}(\Gamma_1, H', R')$ かつ $\mathsf{empty}_R(\Gamma_0)$ となるような $\Gamma_1$，$\Gamma_0$ が存在する．よって 補題 1 より，$\langle H', R', s'\rangle \not\longrightarrow_{D,M} \mathsf{Error}$.

2 つ目の性質について証明する．$\langle \emptyset, \emptyset, s\rangle \longrightarrow^*_{D,M} \langle H, R, \mathsf{skip}\rangle$ と仮定すると， 補題 3 より，$\Theta; \Gamma_1 \vdash \mathsf{skip} \Rightarrow \Gamma_0$，$\mathsf{Con}(\Gamma_1, H, R)$ かつ $\mathsf{empty}_R(\Gamma_0)$ となる $\Gamma_1$，$\Gamma_0$ が存在する．T-Sub，T-Skip から，$\Gamma_1 \leq \Gamma_0$ が成り立つ．したがって，$\mathsf{empty}_R(\Gamma_0)$ と $\Gamma_1 \leq \Gamma_0$ より，$\mathsf{empty}_R(\Gamma_1)$ が成り立つ．よって，$\mathsf{Con}(\Gamma_1, H, R)$ と 補題 2 より，$\forall h \in \mathsf{dom}(H).\exists q \in Q.H(h) = \mathsf{Res}\ q \wedge q \in F$ が成り立つ．□

### 3.4 型 推 論

本節では，上述の型システムに基づき型検査を行うためのアルゴリズムの概略を述べる．定理 1 により，型検査が成功すれば，プログラムが正しく計算機資源を使用していることが分かる．

入力として，各変数に対して所有権を含まない通常の型の注釈がついたプログラムおよび仕様オートマトンが与えられているものとし，型検査（所有権を推論する必要があるので型推論とも呼ぶ）は次の手順で行う．

(1)　各プログラムポイントの各変数に所有権つきの型を割り当てる．この際，所有権は未知なので $[0,1]$ 上の値をとる変数（以下，所有権変数と呼ぶ）を割り当てる．

(2)　型付け規則と型の well-formedness の条件から，部分型の制約および所有権変数に関する線形不等式からなる制約集合を作る．

(3)　部分型の制約を所有権変数に関する線形不等式に帰着し，所有権変数に関する線形不等式のみからなる制約集合を作る．

(4)　線形不等式を解き，解が存在すれば，型付け可能，存在しなければ型付け不能と出力する．

なお，(2) の制約の生成にあたって，あらかじめ図 7 および図 8 の型判定規則中の規則 T-Sub を他の規則と融合しておく．たとえば，規則 T-Let および T-Acc は T-Sub と融合して以下のような規則が得られる[⋆1]．

$$\frac{\Theta;\Gamma, x:\tau_1, y:\tau_2 \vdash s \Rightarrow \Gamma', x:\tau_1' \qquad \tau \leq\approx \tau_1+\tau_2 \qquad \tau_1' \leq \tau_1'' \qquad \mathsf{empty}(\tau_1'')}{\Theta;\Gamma, y:\tau \vdash \mathsf{let}\ x=y\ \mathsf{in}\ s \Rightarrow \Gamma'} \quad \text{(T-Let')}$$

$$\frac{\tau \leq \mathsf{res}\ g \qquad \Delta(g,g',a) \qquad \mathsf{res}\ g' \leq \tau' \qquad \Gamma \leq \Gamma'}{\Theta;\Gamma, x:\tau \vdash \mathsf{acc}^a(x) \Rightarrow \Gamma', x:\mathsf{res}\ \tau'} \quad \text{(T-Acc')}$$

(2) で生成される線形不等式集合のサイズは，たかだかプログラムのサイズの 2 乗のオーダである（各規則から生成される制約のサイズがプログラムのサイズに比例することに注意されたい）．(4) で線形不等式を解く時間は，線形不等式のサイズに対して多項式時間である．したがって，上記のアルゴリズムの計算量は入力プログラムのサイズに対して多項式時間である．

なお，上では入力プログラムには通常の型の注釈がついているものと仮定したが，本論文の型システムは所有権を無視すれば単純型システムなので，通常の型を自動推論してから上のアルゴリズムを適用することができる．推論された型の（木表現での）サイズが（型注釈なしの）プログラムのサイズに対する多項式であれば[⋆2]，型推論アルゴリズム全体の計算量は，やはり入力プログラムのサイズの多項式時間である．

**例 4. 図 10** は検証対象のプログラムと，上記の型推論の手順 (1) まで行った状態の型を表す．手順 (2) では，たとえば，$\mathsf{acc}^{\mathtt{close}}(z)$ からは，

| | |
|---|---|
| $\mathsf{let}\ x = \mathsf{new}^{\mathtt{R}}()\ \mathsf{in}$ | $x : \mathsf{res}\ \{\mathtt{R} \mapsto f_{1,x,\mathtt{R}}, \mathtt{C} \mapsto f_{1,x,\mathtt{C}}\}$ |
| $\mathsf{let}\ y = \mathsf{malloc}()\ \mathsf{in}$ | $x : \mathsf{res}\ \{\mathtt{R} \mapsto f_{2,x,\mathtt{R}}, \mathtt{C} \mapsto f_{2,x,\mathtt{C}}\},$ |
| | $y : (\mathsf{res}\ \{\mathtt{R} \mapsto f_{2,y,\mathtt{R}}, \mathtt{C} \mapsto f_{2,y,\mathtt{C}}\})\ \mathsf{ref}_{f_{2,y,\mathsf{ref}}}$ |
| $*y \leftarrow x;$ | $x : \mathsf{res}\ \{\mathtt{R} \mapsto f_{3,x,\mathtt{R}}, \mathtt{C} \mapsto f_{3,x,\mathtt{C}}\},$ |
| | $y : (\mathsf{res}\ \{\mathtt{R} \mapsto f_{3,y,\mathtt{R}}, \mathtt{C} \mapsto f_{3,y,\mathtt{C}}\})\ \mathsf{ref}_{f_{3,y,\mathsf{ref}}}$ |
| $\mathsf{let}\ z = *y\ \mathsf{in}$ | $x : \mathsf{res}\ \{\mathtt{R} \mapsto f_{4,x,\mathtt{R}}, \mathtt{C} \mapsto f_{4,x,\mathtt{C}}\},$ |
| | $y : (\mathsf{res}\ \{\mathtt{R} \mapsto f_{4,y,\mathtt{R}}, \mathtt{C} \mapsto f_{4,y,\mathtt{C}}\})\ \mathsf{ref}_{f_{4,y,\mathsf{ref}}},$ |
| | $z : \mathsf{res}\ \{\mathtt{R} \mapsto f_{4,z,\mathtt{R}}, \mathtt{C} \mapsto f_{4,z,\mathtt{C}}\}$ |
| $\mathsf{acc}^{\mathtt{close}}(z)$ | $x : \mathsf{res}\ \{\mathtt{R} \mapsto f_{5,x,\mathtt{R}}, \mathtt{C} \mapsto f_{5,x,\mathtt{C}}\},$ |
| | $y : (\mathsf{res}\ \{\mathtt{R} \mapsto f_{5,y,\mathtt{R}}, \mathtt{C} \mapsto f_{5,y,\mathtt{C}}\})\ \mathsf{ref}_{f_{5,y,\mathsf{ref}}},$ |
| | $z : \mathsf{res}\ \{\mathtt{R} \mapsto f_{5,z,\mathtt{R}}, \mathtt{C} \mapsto f_{5,z,\mathtt{C}}\}$ |
| $\mathsf{free}(y)$ | $x : \mathsf{res}\ \{\mathtt{R} \mapsto f_{6,x,\mathtt{R}}, \mathtt{C} \mapsto f_{6,x,\mathtt{C}}\},$ |
| | $y : (\mathsf{res}\ \{\mathtt{R} \mapsto f_{6,y,\mathtt{R}}, \mathtt{C} \mapsto f_{6,y,\mathtt{C}}\})\ \mathsf{ref}_{f_{6,y,\mathsf{ref}}},$ |
| | $z : \mathsf{res}\ \{\mathtt{R} \mapsto f_{6,z,\mathtt{R}}, \mathtt{C} \mapsto f_{6,z,\mathtt{C}}\}$ |

**図 10**　型推論の例
Fig. 10　Example of type inference.

$$f_{4,z,\mathtt{R}} > 0 \qquad f_{5,z,\mathtt{R}} = 0 \qquad f_{5,z,\mathtt{C}} \leq f_{4,z,\mathtt{R}} \qquad f_{4,z,\mathtt{R}} \geq 1$$
$$\mathsf{res}\ \{\mathtt{R} \mapsto f_{4,x,\mathtt{R}}, \mathtt{C} \mapsto f_{4,x,\mathtt{C}}\} \leq \mathsf{res}\ \{\mathtt{R} \mapsto f_{5,x,\mathtt{R}}, \mathtt{C} \mapsto f_{5,x,\mathtt{C}}\}$$
$$(\mathsf{res}\ \{\mathtt{R} \mapsto f_{4,y,\mathtt{R}}, \mathtt{C} \mapsto f_{4,y,\mathtt{C}}\})\ \mathsf{ref}_{f_{4,y,\mathsf{ref}}} \leq (\mathsf{res}\ \{\mathtt{R} \mapsto f_{5,y,\mathtt{R}}, \mathtt{C} \mapsto f_{5,y,\mathtt{C}}\})\ \mathsf{ref}_{f_{5,y,\mathsf{ref}}}$$

などの線形不等式および部分型の制約が得られる．また，型の well-formedness の条件から

$$f_{i,y,\mathsf{ref}} \geq 1/2 \times f_{i,y,\mathtt{R}} \qquad f_{i,y,\mathsf{ref}} \geq 1/2 \times f_{i,y,\mathtt{C}}$$

が各 $i(\in \{2,\ldots,6\})$ について得られる．

手順 (3) では部分型の制約を，定義に従って線形不等式に分解する．たとえば上記の $(\mathsf{res}\ \{\mathtt{R} \mapsto f_{4,y,\mathtt{R}}, \mathtt{C} \mapsto f_{4,y,\mathtt{C}}\})\ \mathsf{ref}_{f_{4,y,\mathsf{ref}}} \leq (\mathsf{res}\ \{\mathtt{R} \mapsto f_{5,y,\mathtt{R}}, \mathtt{C} \mapsto f_{5,y,\mathtt{C}}\})\ \mathsf{ref}_{f_{5,y,\mathsf{ref}}}$ からは

$$f_{4,y,\mathsf{ref}} = f_{5,y,\mathsf{ref}} \qquad f_{4,y,\mathtt{C}} \geq f_{5,y,\mathtt{C}} \qquad f_{4,y,\mathtt{R}} = f_{5,y,\mathtt{R}}$$

が得られる．

手順 (4) では，（他の行から同様にして生成される制約を合わせて）導出されたすべての線形不等式を解き，$f_{1,x,\mathtt{R}}$，$f_{2,x,\mathtt{R}}$，$f_{2,y,\mathtt{ref}}$，$f_{3,y,\mathtt{R}}$，$f_{3,y,\mathsf{ref}}$，$f_{4,y,\mathsf{ref}}$，$f_{4,z,\mathtt{R}}$，$f_{5,y,\mathsf{ref}}$，$f_{5,z,\mathtt{C}}$，$f_{6,z,\mathtt{C}}$ が 1 で，残りが 0 という解が求まり，プログラムが型付けできることが分かる．□

⋆1 実際には T-Sub との融合の仕方を工夫することにより，生成される制約がより少なくなる（たとえば T-Acc' で $\Gamma = \Gamma'$ でよい）ようにできるが，型に基づくプログラム解析に関する標準的な議論なのでここでは割愛する．

⋆2 実際，本論文では組型がないためにこの仮定はつねに成り立つが，組型がある場合には推論される型の木表現でのサイズは入力プログラムの指数サイズになりうる．

## 4. 議　　論

本章では，本研究の手法の限界とその改善策について議論する．

他のプログラム解析と同様，本研究の解析手法は健全ではあるが完全ではない．したがって，本来安全であるプログラムも危険と判断してしまう可能性がある．本論文で提案したような型に基づく手法の典型的な問題は，以下のような，資源の状態が値に依存するようなプログラムをうまく扱えないことである．

$(\mathsf{ifnull}(x)\ \mathsf{then}\ \mathsf{acc}^{\mathtt{close}}(y)\mathsf{else}\ \mathsf{skip}); (\mathsf{ifnull}(x)\ \mathsf{then}\ \mathsf{skip}\ \mathsf{else}\ \mathsf{acc}^{\mathtt{close}}(y))$

このプログラムは，$x$ が null の場合は前半の式で，そうでない場合は後半の式で $y$ を閉じており，（$y$ が開いた状態で実行すれば）意味的には安全なプログラムである．しかしながら，本論文の型システムでは中間の状態に対して型環境を与えることができず，型付けすることができない（中間状態では，$x$ が null であれば，$y$ の型は $\mathsf{res}\ \{\mathtt{C} \mapsto 1\}$，null でなければ $y$ の型は $\mathsf{res}\ \{\mathtt{R} \mapsto 1\}$ であるべきである）．このような問題は，型に基づくプログラム解析の一般的な問題であり，解決方法の 1 つとしては，依存型を用いることが考えられる．

本論文の型システムのもう 1 つの欠点として，関数の型に多相性がないことがあげられる．たとえば $F(x) = \mathsf{skip}$ という関数定義と次の式を考える．

$$\mathsf{let}\ x = \mathsf{new}^{\mathtt{R}}\ \mathsf{in}\ F(x); \mathsf{acc}^{\mathtt{close}}(x); F(x)$$

1 回目，および 2 回目の $F$ の呼び出し直前の型環境はそれぞれ $x : \mathsf{res}\ \{\mathtt{R} \mapsto 1\}$ および $x : \mathsf{res}\ \{\mathtt{C} \mapsto 1\}$ なので，$F$ の型は $(\mathsf{res}\ \{\mathtt{R}\} \mapsto 1) \to (\mathsf{res}\ \{\mathtt{R}\} \mapsto 1)$ および $(\mathsf{res}\ \{\mathtt{C}\} \mapsto 1) \to (\mathsf{res}\ \{\mathtt{C}\} \mapsto 1)$ でなければならない．しかしながら，本論文の型システムでは関数には単一の型しか許していないので型付けに失敗する．上の例の場合には，以下のように関数呼び出しの型付け規則を拡張することによっても対処できる．

$$\frac{\Theta(f) = (\tilde{\tau}) \to (\tilde{\tau}')}{\Theta; \Gamma, \tilde{x} : (\tilde{\tau} + \tilde{\tau}'') \vdash F(\tilde{x}) \Rightarrow \Gamma, \tilde{x} : (\tilde{\tau}' + \tilde{\tau}'')} \qquad (\text{T-Call'})$$

より一般的な解決策としては，関数の型に所有権に関する多相性を導入することが考えられる[12),13)]．

本論文の手法のさらなる欠点としては，assert 文をプログラム中に適切に挿入しなければならないことがあげられる．ただし，型付け規則から明らかなように，assert 文を挿入すればするほど解析の精度は上がる（assert 文の挿入により，型付けできなかったプログラムが型付けできるようになることはあっても，逆はない）ことから，本研究の手法を適用する前に must エイリアス解析を行い，求まった情報をすべて assert 文の形で挿入することによって，検証全体を自動化することができる．ポインタの使用法のみに着目した先行研究[5)] の実験では必要な assert 文のほとんどは単純なものであり，簡単な must エイリアス解析で十分な効果が得られるものと期待できる．

## 5. 関 連 研 究

本論文の型システムは，Suenaga and Kobayashi[5)] のメモリリークの静的検証のための型システムを，一般の計算資源を扱えるように拡張したものである．そのために，計算資源の各状態に対して所有権を割り当て，メモリリークの検証同様に型検査を線形計画法に帰着できるような型システムを構築した．

既存の計算資源使用法の静的検証手法としては，Igarashi, Kobayashi and Iwama[1),2)] による関数型言語のための型システム，Foster, Terauchi, Aiken, Kodumal[3),4)] による C 言語のための検証手法，DeLine らによる低レベル言語のための型システム[14)]，Strom らによる型状態[15)] とその拡張[16)]，などがある．Igarashi, Kobayashi and Iwama[1),2)] による手法はポインタを扱っておらず，以下の理由により，十分な解析精度を維持しつつポインタを扱えるように拡張することは困難であると思われる．Igarashi and Kobayashi による型システムでは，ファイルなどの計算資源の型に，計算資源の状態ではなく，「今後どのような順序でアクセスすべきか」を表す使用法（アクセス列の集合）を付加している．たとえば，読み込み専用ファイルには $\mathtt{File}(\mathtt{read}^*\mathtt{close})$ が，閉じた後のファイルには $\mathtt{File}(1)$ が，割り当てられる．変数 $x$ が型 $\mathtt{File}(\mathtt{read}^*\mathtt{close})$ を持つ際に，$x$ の値を $y$ にコピーした場合，$x$ の使用法 $\mathtt{read}^*\mathtt{close}$ が 2 つの独立な使用法に分割され，型環境は $x : \mathtt{File}(\mathtt{read}^*\mathtt{close})$, $y : \mathtt{File(1)}$ または $x : \mathtt{File(1)}$, $y : \mathtt{File}(\mathtt{read}^*\mathtt{close})$ となる．したがって，この型システムでは，1 章の図 3 のような問題は生じない（2 行目で $x$ の使用法 $\mathtt{read}^*\mathtt{close}$ を 2 つの $\mathtt{read}^*\mathtt{close}$ に分割することはできない）．しかしながら，異なる変数を介した同一の計算資源に対するアクセス順序を表せないため，`let x=y in read(x); close(y)` と `let x=y in close(y); close(x)`，（仮にポインタ型を導入して拡張したとして）`let x=*y in read(x); close(*y)` と `let x=*y in close(*y); read(x)` を区別することはできず，すべて型エラーとしてしまう．2 番目の例の前者は通常の C プログラムなどで頻繁に現れる `read(*y); close(*y)` に相当するため，このような解析精度の粗さは致命的である．

Foster, Terauchi, Aiken, Kodumal[3),4)] による手法では，計算資源の型に，計算資源の（may エイリアス集合の）抽象化であるリージョンを割り当て，そのリージョンごとに計算

資源の状態を管理している．たとえば読み込み可能なファイルには，File($\rho$) という型を割り当て，同時に $\{\rho \mapsto \mathtt{R}\}$ のように各リージョンに状態を割り当てる．この手法では，リージョンごとに状態を管理しているため，型 File($\rho$) の計算資源に対して状態の変更をともなうアクセスを行うためには，$\rho$ によって表される計算資源がただ 1 つであることが静的に保証されなければならない．さらにそれだけでは（複数の計算資源が同一のリージョンに抽象化されることによって）検証が保守的になりすぎるため，restrict と呼ばれる特殊なコンストラクトを導入し，その自動推論を行っている．以上のように，Foster らの解析は別名解析，（リージョンの）線形性の解析，restrict の推論など複数の解析を組み合わせた複雑なものとなっており，その正しさは証明されていない．それに対し，本論文の解析手法は単一の型システムで表現され，その正しさが証明されている．また，Foster らの解析ではエイリアス解析が型システムの正しさを保証するために本質的に必要なものであるのに対し，我々の型システムでは（assert 文で提供される）エイリアス情報がなくても型システムの健全性には影響がないという点が異なる．両者の解析の効率，精度の比較は今後の課題である．

DeLine らによる型システム[14] では，存在型などを含む複雑な型をユーザが記述しなければならない．Strom らによる型状態[15] はエイリアスには対応していない．型状態をエイリアスを扱えるように拡張した手法として Ramalingam らによるデータフロー解析[16] もあるが，上記の Foster らの手法と同様，エイリアス解析を包含した複雑な解析となっており，その正しさは証明されていない．

有理数の所有権または権限の概念は，すでに様々なプログラム解析のために用いられている[6)–8),10)]．有理数権限に基づく型システムの型判定問題を線形計画問題に帰着するというアイデアも寺内ら[8] によって提案されている．

## 6. 結　　論

本論文では，ポインタのある言語を対象に資源使用法検証を行うための型システムを提案し，その正しさを証明した．今後の課題としては，提案した型システムに基づく検証器の実装と評価，また assert の自動挿入手法の考案があげられる．

**謝辞**　本論文の査読者から有益なコメントをいただき深く感謝いたします．本論文は科研費 20240001 の支援を受けて実施しました．

## 参 考 文 献

1) Igarashi, A. and Kobayashi, N.: Resource usage analysis, *ACM Trans. Program. Lang. Syst.*, Vol.27, No.2, pp.264–313 (2005).
2) Iwama, F., Igarashi, A. and Kobayashi, N.: Resource usage analysis for a functional language with exceptions, *PEPM*, pp.38–47 (2006).
3) Foster, J.S., Terauchi, T. and Aiken, A.: Flow-Sensitive Type Qualifiers, *PLDI*, pp.1–12 (2002).
4) Aiken, A., Foster, J.S., Kodumal, J. and Terauchi, T.: Checking and inferring local non-aliasing, *PLDI*, pp.129–140, ACM (2003).
5) Suenaga, K. and Kobayashi, N.: Fractional Ownerships for Safe Memory Deallocation, *APLAS*, Hu, Z. (Ed.), Lecture Notes in Computer Science, Vol.5904, pp.128–143, Springer (2009).
6) Ueda, K.: Resource-Passing Concurrent Programming, *TACS*, Kobayashi, N. and Pierce, B.C. (Eds.), Lecture Notes in Computer Science, Vol.2215, pp.95–126, Springer (2001).
7) Boyland, J.: Checking Interference with Fractional Permissions, *SAS*, Cousot, R. (Ed.), Lecture Notes in Computer Science, Vol.2694, pp.55–72, Springer (2003).
8) Terauchi, T.: Checking race freedom via linear programming, *PLDI*, Gupta, R. and Amarasinghe, S.P. (Eds.), pp.1–10, ACM (2008).
9) Terauchi, T. and Aiken, A.: A Capability Calculus for Concurrency and Determinism, *CONCUR*, Baier, C. and Hermanns, H. (Eds.), Lecture Notes in Computer Science, Vol.4137, pp.218–232, Springer (2006).
10) Kikuchi, D. and Kobayashi, N.: Type-Based Automated Verification of Authenticity in Cryptographic Protocols, *ESOP*, Castagna, G. (Ed.), Lecture Notes in Computer Science, Vol.5502, pp.222–236, Springer (2009).
11) Godefroid, P., Nori, A.V., Rajamani, S.K. and Tetali, S.: Compositional may-must program analysis: Unleashing the power of alternation, *POPL*, Hermenegildo, M.V. and Palsberg, J. (Eds.), pp.43–56, ACM (2010).
12) Heine, D.L. and Lam, M.S.: A practical flow-sensitive and context-sensitive C and C++ memory leak detector, *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp.168–181 (2003).
13) Yasuoka, H. and Terauchi, T.: Polymorphic Fractional Capabilities, *SAS*, Palsberg, J. and Su, Z. (Eds.), Lecture Notes in Computer Science, Vol.5673, pp.36–51, Springer (2009).
14) DeLine, R. and Fähndrich, M.: Enforcing High-Level Protocols in Low-Level Software, *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp.59–69 (2001).
15) Strom, R.E. and Yemini, S.: Typestate: A Programming Language Concept for Enhancing Software Reliability, *Trans. Software Engineering*, Vol.12, No.1, pp.157–171 (1986).

16) Fink, S.J., Yahav, E., Dor, N., Ramalingam, G. and Geay, E.: Effective typestate verification in the presence of aliasing, *ACM Trans. Softw. Eng. Methodol.*, Vol.17, No.2 (2008).

## 付　　録

### A.1　補題 3 の証明

補題 3 を示すために，補題 4 と補題 5 を用意する．

すべての $h \in \mathsf{dom}(F_0) \cap \mathsf{dom}(F_1)$ について $F_0(h) = F_1(h)$ であり，かつすべての $h \in \mathsf{dom}(F_i) \setminus \mathsf{dom}(F_{1-i})$ について，$F_i(h) = \emptyset$ が成り立つとき，$F_0 \approx F_1$ と書く（つまり，値域が $\emptyset$ であるような部分についてのみ定義域が異なる関数を同一視する）．

**補題 4.** $dom(H') \cup \{h\} = dom(H)$ かつ $\forall h' \in dom(H) \setminus \{h\}.H'(h') = H(h)$ が成り立つものとする．このとき，$\mathsf{ownall}(H, h, \tau)(h) = \emptyset$ ならば，$\mathsf{ownall}(H, v, \tau) \approx \mathsf{ownall}(H', v, \tau)$ が成り立つ．

**証明**　$F_0' = \mathsf{own}(H, v, \tau)$ と $F_1' = \mathsf{own}(H', v, \tau)$ とすると，$\mathsf{ownall}(H, v, \tau) = \sum_{w \in \{0\}^*} F_0'(w)$，$\mathsf{ownall}(H', v, \tau) = \sum_{w \in \{0\}^*} F_1'(w)$．よって，すべての $w \in \{0^*\}$ について，$F_0'(w) \approx F_1'(w)$ が成り立つことを示せばよい．

$w$ の前方部分列で，$[\![H, v]\!](w_1) = h$ を満たす $w_1$ が存在する場合，その中で最も短いものを $w_1'$ とする．$\mathsf{ownall}(H, h, \tau)(h) = \emptyset$ より，$[\![\tau]\!](w_1') = \emptyset$ が得られる．$\tau$ の well-formedness より，$[\![\tau]\!](w) = \emptyset$ が成り立つ．よって，$\mathsf{own}(H, v, \tau)$ の定義より，$F_0'(w) \approx F_1'(w)$ が成り立つ．

$w$ の前方部分列で，$[\![H, v]\!](w_1) = h$ を満たす $w_1$ が存在しない場合，$[\![H, v]\!](w) = [\![H', v]\!](w)$ なので明らかに $F_0'(w) \approx F_1'(w)$．□

**補題 5.**（weakening）$\Theta; \Gamma \vdash s \Rightarrow \Gamma'$ かつ $x \notin \mathsf{dom}(\Gamma) \cup \mathsf{dom}(\Gamma')$ ならば，$\Theta; \Gamma, x : \tau \vdash s \Rightarrow \Gamma', x : \tau$ が成り立つ．

**証明**　$\Theta; \Gamma \vdash s \Rightarrow \Gamma''$ の導出に関する帰納法より明らか．□

以上の補題を用いて 補題 3 を証明する．

**補題 3 の証明** $\Theta; \Gamma \vdash s \Rightarrow \Gamma''$ の導出に関する帰納法で証明する．最後に用いる型判定規則によって場合分けをする．以下では主要な場合のみ記す．

- T-Seq の場合：仮定より，以下の条件が成り立つ．

$$s = s_1; s_2 \qquad s' = s_1'; s_2 \qquad \langle H, R, s_1 \rangle \longrightarrow_{D,M} \langle H', R', s_1' \rangle$$
$$\Theta; \Gamma \vdash s_1 \Rightarrow \Gamma_1 \qquad \Theta; \Gamma_1 \vdash s_2 \Rightarrow \Gamma''$$

帰納法の仮定より，$\Theta; \Gamma' \vdash s_1' \Rightarrow \Gamma_1, \Gamma_0$ かつ $\mathsf{empty}_{R'}(\Gamma_0)$ および $\mathsf{Con}(\Gamma', H', R')$ を満たす $\Gamma'$ と $\Gamma_0$ が存在する．$\Theta; \Gamma_1 \vdash s_2 \Rightarrow \Gamma''$ に 補題 5 を適用し，$\Theta; \Gamma_1, \Gamma_0 \vdash s_2 \Rightarrow \Gamma'', \Gamma_0$ が導ける．よって，型付け規則の T-Seq より，$\Theta; \Gamma' \vdash s' \Rightarrow \Gamma'', \Gamma_0$ が導かれる．

- T-Assign の場合：仮定より，以下の条件が成り立つ．

$$s = *x \leftarrow y \qquad s' = \mathsf{skip}$$
$$\Gamma = \Gamma_1, x : \tau'\ \mathsf{ref}_1, y : \tau \qquad \Gamma'' = \Gamma_1, x : \tau_1\ \mathsf{ref}_1, y : \tau_2$$
$$\mathsf{empty}(\tau') \qquad \tau \approx \tau_1 + \tau_2$$
$$H' = H\{R(x) \mapsto R(y)\} \qquad R' = R$$

$\Gamma' = \Gamma''$，$\Gamma_0 = \emptyset$ とおくと，$\Theta; \Gamma' \vdash s' \Rightarrow \Gamma''$，$\Gamma_0$ および $\mathsf{empty}(\Gamma_0)$ が成り立つ．$\mathsf{Con}(\Gamma', H', R')$ は以下のように導かれる．まず条件 ( 1 ) は，$\mathsf{dom}(R') = \mathsf{dom}(R) = \mathsf{dom}(\Gamma) = \mathsf{dom}(\Gamma'') = \mathsf{dom}(\Gamma')$ より成立する．条件 ( 2 ) を示すために，$O_0$ と $O_1$ を次のように定義する．

$$O_0 = O_0' + \mathsf{ownall}(H, R(x), \Gamma(x)) + \mathsf{ownall}(H, R(y), \Gamma(y))$$
$$O_1 = O_1' + \mathsf{ownall}(H', R'(x), \Gamma'(x)) + \mathsf{ownall}(H', R'(y), \Gamma'(y))$$
$$O_0' = \sum_{z \in \mathsf{dom}(\Gamma) \setminus \{x,y\}} \mathsf{ownall}(H, R(z), \Gamma(z))$$
$$O_1' = \sum_{z \in \mathsf{dom}(\Gamma') \setminus \{x,y\}} \mathsf{ownall}(H', R'(z), \Gamma'(z))$$

$O_0 \approx O_1$ を示せば，仮定 $\mathsf{Con}(\Gamma, H, R)$ の条件 ( 2 ) から $\mathsf{Con}(\Gamma', H', R')$ の条件 ( 2 ) が導かれる．

まず，$\mathsf{empty}(\tau')$ より，$\mathsf{ownall}(H, R(x), \Gamma(x)) = \{R(x) \mapsto \{\mathsf{ref} \mapsto 1\}\}$ が成り立つ．これと $\mathsf{Con}(\Gamma, H, R)$ より，$O_0(R(x)) = \{\mathsf{ref} \mapsto 1\}$ が導ける．よって，$O_0'(R(x)) = \emptyset$．したがって，補題 4 より，$O_0' \approx O_1'$ が成り立つ．

さらに，$\tau \approx \tau_1 + \tau_2$ および $H'(R'(x)) = R'(y) = R(y)$ より，

$$\begin{aligned}
&\mathsf{ownall}(H', R'(x), \Gamma'(x)) + \mathsf{ownall}(H', R'(y), \Gamma'(y)) \\
&\approx \{R(x) \mapsto \{\mathsf{ref} \mapsto 1\}\} + \mathsf{ownall}(H', H'(R'(x)), \tau_1) + \mathsf{ownall}(H', R'(y), \tau_2) \\
&\approx \{R(x) \mapsto \{\mathsf{ref} \mapsto 1\}\} + \mathsf{ownall}(H, R(y), \tau) \\
&\approx \mathsf{ownall}(H, R(x), \Gamma(x)) + \mathsf{ownall}(H, R(y), \Gamma(y))
\end{aligned}$$

以上より，$O_0 \approx O_1$ が成り立つ．

- T-Free の場合：仮定より，次の条件が成り立つ．

$$\begin{array}{ll}
s = \mathsf{free}(x) & s' = \mathsf{skip} \\
\Gamma = \Gamma_1, x : \tau\ \mathsf{ref}_1 & \Gamma'' = \Gamma_1, x : \tau\ \mathsf{ref}_0 \\
\mathsf{empty}_R(\tau) & H' = H \setminus \{R(x)\} \\
R' = R & R(x) \in \mathsf{dom}(H) \cup \{\mathsf{null}\}
\end{array}$$

$\Gamma' = \Gamma''$，$\Gamma_0 = \emptyset$ とおくと，$\Theta; \Gamma' \vdash s' \Rightarrow \Gamma''$，$\Gamma_0$ および $\mathsf{empty}(\Gamma_0)$ が成り立つ．次に $\mathsf{Con}(\Gamma', H', R')$ を示す．条件 ( 1 ) は $\mathsf{dom}(\Gamma') = \mathsf{dom}(\Gamma) = \mathsf{dom}(R) = \mathsf{dom}(R')$ により成立．条件 ( 2 ) を証明するために $O_0$ と $O_1$ を次のように定義する．

$$\begin{aligned}
O_0 &= O_0' + \mathsf{ownall}(H, R(x), \tau\ \mathsf{ref}_1) \\
O_1 &= O_1' + \mathsf{ownall}(H', R'(x), \tau\ \mathsf{ref}_0) \\
O_0' &= \sum_{z \in \mathsf{dom}(\Gamma) \setminus \{x\}} \mathsf{ownall}(H, R(z), \Gamma(z)) \\
O_1' &= \sum_{z \in \mathsf{dom}(\Gamma') \setminus \{x\}} \mathsf{ownall}(H', R'(z), \Gamma'(z))
\end{aligned}$$

以下，$R(x) = \mathsf{null}$ であるか否かで場合分けをする．$R(x) = \mathsf{null}$ のとき，$O_0 \approx O_1$ かつ $H' = H$ なので，$\mathsf{Con}(\Gamma, H, R)$ より $\mathsf{Con}(\Gamma', H', R')$ が導ける．

$R(x) = h \neq \mathsf{null}$ のとき，$\mathsf{Con}(\Gamma, H, R)$ より $O_0(h) = \{\mathsf{ref} \mapsto 1\}$ および $O_0'(h) = \emptyset$ が成り立つ．したがって，補題 4 より，$O_0' \approx O_1'$ が成り立つ．よって $\mathsf{Con}(\Gamma, H, R)$ の条件 ( 2 ) より，以下のように $\mathsf{Con}(\Gamma', H', R')$ の条件 ( 2 ) が導かれる．各 $h' \in dom(H')$ について，$H(h') = H'(h')$ かつ $O_0(h') = O_1(h')$ なので条件 ( 2 )(i) が成り立つ．また，$h' \in dom(O_1) \setminus dom(H')$ ならば，$h' \in dom(O_1) \setminus dom(H)$ または $h' = h$ であり，前者の場合は $O_1(h') = O_0(h') = \emptyset$，後者の場合も $O_1(h') = O_1(h) = O_1'(h) = O_0'(h) = \emptyset$ が成り立つ．したがって，条件 ( 2 )(ii) も成り立つ．

- T-New の場合：仮定より，以下の条件が成り立つ．

$$\begin{array}{lll}
s = \mathsf{let}\ x = \mathsf{new}^q()\ \mathsf{in}\ s_1 & s' = [x'/x]s_1 & \\
H' = H\{h \mapsto \mathsf{Res}\ q\} & R' = R\{x' \mapsto h\} & h \notin \mathsf{dom}(H) \\
\Theta; \Gamma, x : \mathsf{res}\ \{q \mapsto 1\} \vdash s_1 \Rightarrow \Gamma'', x : \mathsf{res}\ g' & \mathsf{empty}(\mathsf{res}\ g') &
\end{array}$$

$\Gamma' = \Gamma, x' : \mathsf{res}\ \{q \mapsto 1\}$，$\Gamma_0 = x' : \mathsf{res}\ g'$ とすると，$\Theta; \Gamma' \vdash s' \Rightarrow \Gamma'', \Gamma_0$ および $\mathsf{empty}(\Gamma_0)$ が成り立つ．次に $\mathsf{Con}(\Gamma', H', R')$ を証明する．条件 ( 1 ) は $\mathsf{dom}(\Gamma') = \mathsf{dom}(\Gamma) \cup \{x'\} = \mathsf{dom}(R') \cup \{x'\} = \mathsf{dom}(R)$ よりただちに導かれる．
条件 ( 2 ) は以下の関係と $\mathsf{Con}(H, R, \Gamma)$ から導かれる．

$$\begin{aligned}
&\sum_{z \in \mathsf{dom}(\Gamma')} \mathsf{ownall}(H', R'(z), \Gamma'(z)) \\
&= \left( \sum_{z \in \mathsf{dom}(\Gamma)} \mathsf{ownall}(H, R(z), \Gamma(z)) \right) + \mathsf{ownall}(H\{h \mapsto \mathsf{Res}\ q\}, h, \mathsf{res}\ \{q \mapsto 1\}) \\
&= \left( \sum_{z \in \mathsf{dom}(\Gamma)} \mathsf{ownall}(H, R(z), \Gamma(z)) \right) + \{h \mapsto \{q \mapsto 1\}\}
\end{aligned}$$

- T-Acc の場合：仮定より，以下の条件が成り立つ．

$$\begin{array}{lll}
s = \mathsf{acc}^a(x) & s' = \mathsf{skip} & \Delta(g, g', a) \\
H = H_1\{R(x) \mapsto \mathsf{Res}\ q\} & H' = H_1\{R(x) \mapsto \mathsf{Res}\ \delta(q, a)\} & R' = R \\
\Gamma = \Gamma_1, x : \mathsf{res}\ g & \Gamma'' = \Gamma_1, x : \mathsf{res}\ g' &
\end{array}$$

$\Gamma' = \Gamma''$，$\Gamma_0 = \emptyset$ とおくと，$\Theta; \Gamma' \vdash s' \Rightarrow \Gamma''$，$\Gamma_0$ および $\mathsf{empty}(\Gamma_0)$ が成り立つ．次に $\mathsf{Con}(\Gamma', H', R')$ を示す．

条件 ( 1 ) は $\mathsf{dom}(R') = \mathsf{dom}(R) = \mathsf{dom}(\Gamma) = \mathsf{dom}(\Gamma'') = \mathsf{dom}(\Gamma')$ より成立．$\mathsf{Con}(\Gamma, H, R)$ の条件 ( 2 )(i) より，$g = \{q \mapsto f\}$．以下，$\delta(q, a) = q$ であるか否かにより場合分けを行う．

$\delta(q, a) = q$ のとき．$\Delta(g, g', a)$ より，$g' = \{q \mapsto f\}$．よって，$\mathsf{Con}(\Gamma', H', R')$ は $\mathsf{Con}(\Gamma, H, R)$ よりただちに導かれる．

$\delta(q, a) = q' \neq q$ のとき．$\Delta(g, g', a)$ より，$f = 1$ かつ $g' = \{q' \mapsto 1\}$．ここで，

$$O_0 = O_0' + \mathsf{ownall}(H, R(x), \mathsf{res}\ g)(= O_0' + \{R(x) \mapsto g\})$$
$$O_1 = O_1' + \mathsf{ownall}(H', R'(x), \mathsf{res}\ g')(= O_1' + \{R(x) \mapsto g'\})$$
$$O_0' = \sum_{z \in \mathtt{dom}(\Gamma) \setminus \{x\}} \mathsf{ownall}(H, R(z), \Gamma(z))$$
$$O_1' = \sum_{z \in \mathtt{dom}(\Gamma') \setminus \{x\}} \mathsf{ownall}(H', R'(z), \Gamma'(z))$$

とすると，$\mathsf{Con}(\Gamma, H, R)$ の条件 (2) および $g = \{q \mapsto f\}$ より，$O_0'(R(x)) = \emptyset$. また，ownall の定義，$R$，$R'$，$H$，$H'$，$\Gamma$，$\Gamma'$ の関係より $O_1' = O_0'$. したがって，$O_1 = O_0\{R(x) \mapsto g'\}$ であり，$\mathsf{Con}(\Gamma', H', R')$ の条件 (2) は $\mathsf{Con}(\Gamma, H, R)$ の条件 (2) よりただちに導かれる．

- T-Assert*の場合：仮定より，以下の条件が成り立つ．

$$s = \mathsf{assert}(x = *y) \qquad s' = \mathsf{skip}$$
$$H' = H \qquad R' = R \qquad R(x) = H(R(y))$$
$$\Gamma = \Gamma_1, x : \tau_1, y : \tau_2\ \mathsf{ref}_f \qquad \Gamma'' = \Gamma_1, x : \tau_1', y : \tau_2'\ \mathsf{ref}_f \qquad \tau_1 + \tau_2 \approx \tau_1' + \tau_2'$$

$\Gamma' = \Gamma''$，$\Gamma_0 = \emptyset$ とすると，$\Theta; \Gamma' \vdash s' \Rightarrow \Gamma''$，$\Gamma_0$ および $\mathsf{empty}(\Gamma_0)$ が成り立つ．次に，$\mathsf{Con}(\Gamma', H', R')$ を示す．条件 (1) は，$\mathsf{dom}(\Gamma') = \mathsf{dom}(\Gamma_1) \cup \{x, y\} = \mathsf{dom}(\Gamma) = \mathsf{dom}(R) = \mathsf{dom}(R')$ により成立．条件 (2) を示すには，$\mathsf{ownall}(H, R(y), \tau_2\ \mathsf{ref}_f) + \mathsf{ownall}(H, R(x), \tau_1) = \mathsf{ownall}(H', R'(y), \tau_2'\ \mathsf{ref}_f) + \mathsf{ownall}(H', R'(x), \tau_1')$ を示せばよいが，これは $H(R(y)) = R(x)$ と $\tau_1 + \tau_2 \approx \tau_1' + \tau_2'$ よりただちに導かれる．　□

(平成 22 年 2 月 15 日受付)
(平成 22 年 4 月 20 日採録)

**上野　慎平**

1985 年生．2008 年東北大学工学部電気情報・物理工学科卒業．2010 年同大学院情報科学研究科修士課程修了．

**小林　直樹**（正会員）

1968 年生．1991 年東京大学理学部情報科学科卒業．1993 年同大学院理学系研究科情報科学専攻修士課程修了，同年博士課程進学．東京大学大学院理学系研究科情報科学専攻助手，講師，東京工業大学大学院情報理工学研究科助教授を経て 2004 年より東北大学大学院情報科学研究科教授，現在に至る．博士（理学）．型理論，プログラム解析，並行計算等に興味を持つ．ACM，日本ソフトウェア科学会各会員．2001 年 IFIP TC2 Manfred Paul Award，2003 年日本 IBM 科学賞，2009 年日本学術振興会賞等を受賞

**海野　広志**

1982 年生．2004 年法政大学情報科学部卒業．2006 年東京大学大学院情報理工学系研究科コンピュータ科学専攻修士課程修了．2009 年同大学院博士後期課程修了．同年より東北大学大学院情報科学研究科にて博士研究員として勤務．現在に至る．博士（情報理工学）．