例外機構を備えた言語のための資源使用法解析

岩間 太
東北大学
大学院情報科学研究科
iwama@kb.ecei.tohoku.ac.jp

五十嵐 淳
京都大学
大学院情報学研究科
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

五十嵐と小林は，λ 計算を対象言語とし，ファイルやメモリなどの計算資源が正しく使われているかどうかを検証するための統一的な型システムを構築した．本論文では，計算資源にアクセスする実際のプログラムで頻繁に使われる例外処理機構を対象言語に付加し，計算資源利用法検証のための型システムを拡張する．例外処理機構を扱うには，五十嵐と小林による型システムで用いられている線形型の概念だけでは十分な解析精度は得られない．本論文では，線形型とエフェクトシステムを組み合わせることによってその問題を解決した．

1 Introduction

近年，プログラム検証の重要な問題の一つとして，ファイルやメモリ，ネットワークなどの外部の計算資源が正しく使用されているかどうかを検証する問題が多くの研究者によって取り組まれている [2, 3, 8, 10, 17, 18]．例えば，獲得されたロックが，必ず解放されるか，割り当てられたメモリ領域が必ず解放されるか，開かれたファイルが必ず閉じられるか，といった性質の検証手法が研究されている．

五十嵐と小林は，上記の問題（以下では**資源使用法解析**の問題と呼ぶ）に対し，型システムを用いた統一的な解析手法を与えている [10]．しかしながら，純粋な値呼び関数型言語に資源の使用プリミティブのみを追加した言語を対象としており，現実のプログラミング言語にそのまま適用することはできない．

本研究では，五十嵐と小林の資源使用法解析のための型システムを現実のプログラミング言語を扱えるように拡張するための第一歩として，対象言語に例外処理機構を加え，型システムを拡張する．資源のアクセスプリミティブの多くは例外発生（例えばファイルであれば，*End_of_File*，ファイル名に対応するファイルがないなど）を伴うため，例外処理機構を扱えることは計算資源を使用する現実のプログラムを解析する上で極めて重要である．

以下ではまず五十嵐と小林の型システムのアイデアを説明し，次に例外を扱うための本論文での拡張のアイデアを述べる．

五十嵐と小林の型システムのアイデアは，資源の型に使用順序に関する情報を付加して型システムを拡張することである．例えば，読み込み専用のファイルの型は $(\mathbf{File}, R^*; C)$，読み書き可能なファイルの型は $(\mathbf{File}, (R+W)^*; C)$ のように表すことができる．ここで，R, W, C は，ファイルからの読み込み操作，ファイルへの書き込み操作，ファイルのクローズ操作を表し，その順序を正則表現 (ただし，連接を ; で表している) で与えている．(実際には，使用順序に関する情報は正則表現ではなく，usage 式と呼ぶ，より表現力の高い言語を用いる)．

この拡張した型を反映して通常の型付け規則も変更する．例えば *let* 式に対する型付規則：

$$\frac{\Gamma \vdash M : \tau_1 \qquad \Gamma, x : \tau_1 \vdash N : \tau_2}{\Gamma \vdash \mathbf{let}\ x = M\ \mathbf{in}\ N : \tau_2}$$

は，

$$\frac{\Gamma \vdash M : \tau_1 \qquad \Delta, x : \tau_1 \vdash N : \tau_2}{\Gamma; \Delta \vdash \mathbf{let}\ x = M\ \mathbf{in}\ N : \tau_2}$$

に変更する．ここで，$\Gamma; \Delta$ は Γ および Δ 中の使用順序に関する部分を結合して得られるものを表す．例えば $\Gamma = x : (\mathbf{File}, R^*)$ かつ $\Delta = x : (\mathbf{File}, C)$ であれば，$\Gamma; \Delta = x : (\mathbf{File}, R^*; C)$ である．このように拡張した型システムに対する型推論を行うことにより，各資源に対する使用法を推論することができる．

本論文では，対象言語に例外の発生プリミティブ **raise**（簡単のため，例外を一種類に限定する）および例外処理プリミティブ **try** M_1 **with** M_2（M_1 を評価し，評価中に例外が発生したら M_2 を評価した結果を返す）を追加する．それらのプリミティブに関する解析が行えるよう

に，使用順序に関する情報を表す usage 式に 2 つの新しい構成子 E および $U_1;_E U_2$ を追加する．E は例外が発生することを，$U_1;_E U_2$ は，まず U_1 にしたがって使用されるが，例外が起った後は U_2 にしたがって使用されることを表す．例えば，式 $\mathbf{read}(x);\mathbf{raise}$ 中での x の使用法は $R;E$ であり，$\mathbf{try\ read}(x);\mathbf{raise\ with\ close}(x)$ 中の usage は $(R;E);_E C$ と表すことができる（後者は意味的には $R;C$ と等しい）．

型付け規則については，五十嵐と小林の型システムの単純な拡張として以下のような規則を導入することが考えられる．

$$\frac{Use(\tau_i) = E \quad for\ each\ i \in \{1, \cdots, n\}}{x_1 : \tau_1, \ldots, x_n : \tau_n \vdash \mathbf{raise} : \tau}$$

$$\frac{\Gamma \vdash M_1 : \tau \qquad \Delta \vdash M_2 : \tau \qquad dom(\Gamma) = dom(\Delta)}{\Gamma;_E \Delta \vdash \mathbf{try}\ M_1\ \mathbf{with}\ M_2 : \tau}$$

ここで，一番目の規則の仮定部は，τ_i が資源型ならばその usage は E である（すなわち資源は用いられずに例外だけが発生する）ことを表し，二番目の規則の結論部の $\Gamma;_E \Delta$ は，各変数の usage 部分を $;_E$ で結合して得られる型環境を表す．例えば，$M = \mathbf{try\ read}(x);\mathbf{raise\ with\ close}(x)$ に対する型判断は以下のように導出することができる．

$$\Pi = \frac{\begin{array}{c} x : (\mathbf{File}, R) \vdash \mathbf{read}(x) : \mathbf{bool} \\ x : (\mathbf{File}, E) \vdash \mathbf{raise} : \mathbf{bool} \end{array}}{x : (\mathbf{File}, R; E) \vdash \mathbf{read}(x); \mathbf{raise} : \mathbf{bool}}$$

$$\frac{\Pi \quad x : (\mathbf{File}, C) \vdash \mathbf{close}(x) : \mathbf{bool}}{x : (\mathbf{File}, (R; E);_E C) \vdash M : \mathbf{bool}}$$

しかしながら，上のように例外の発生の情報を usage 部分のみで管理すると，例外の発生のタイミングが正確に推論できず，結果として資源に対するアクセスが正確に解析できない．例えば，以下のような式を考えよう（f の定義を展開すれば，上の M が得られる）．

$$\mathbf{let}\ f = \lambda y.\mathbf{raise\ in\ try\ read}(x); f()\ \mathbf{with\ close}(x)$$

五十嵐と小林の型システムにおける λ 抽象に対する型付規則は以下のような形である．

$$\frac{\Gamma, x : \tau_1 \vdash M : \tau_2}{\diamond\Gamma \vdash \lambda x.M : \tau_1 \to \tau_2}$$

Γ で表された資源へのアクセスが起るのは $\lambda x.M$ の評価中ではないため，$\diamond\Gamma$ によって,「Γ によって表されるアクセスが $\lambda x.M$ の評価中あるいはそれ以後に起る」ことを表している．このため，五十嵐と小林の型システムを単純に拡張すると，上のプログラムに対する x の使用法は，$\diamond E;(R;_E C)$ となり，E で表される例外発生が $R;_E C$ で表される x の使用のどこでおこるかがわからず，結果として，x が読まれるかどうか，閉じられるかどうかが解析できない.（この問題は例外に限らず，リソースのアクセスに関する情報についても同じことがいえる [12] が，例外の場合は上記の例からもわかるとおり，発生のタイミングによって資源の使用法の推論の精度が特に大きく影響をうける.）

上記の問題を解決するため，本研究では例外の発生のタイミングの情報をエフェクト [15, 16] として推論する．新しい型判断は $\Gamma \parallel \varphi \vdash M : \tau$ の形であり「M の評価中に Γ にしたがった資源へのアクセスがおこり，φ にしたがった例外が起りうる」ことを表す．λ 抽象に対する規則は以下のようになる．

$$\frac{\Gamma, x : \tau_1 \,||\, \varphi \vdash M : \tau_2}{\diamond(\Gamma \setminus E) \,||\, \mathbf{0} \vdash \lambda x.M : (\tau_1 \setminus E) \xrightarrow{\varphi} \tau_2}$$

関数本体実行中の例外発生に関するエフェクト情報 φ は関数型の latent effect として表されるため，M を評価したときに発生する例外が関数 $\lambda x.M$ を評価したときに起こるという情報を正しく推論することができる．$\Gamma \setminus E, \tau_1 \setminus E$ は Γ, τ_1 から例外発生に関する情報を取り除いてえられる型環境，型を表すが，これは latent effect と型環境でひとつの **raise** を二重に数えないための処置である．

以後の構成を示す．まず 2 節では，我々が対象とする例外機構を導入した言語について述べる．3 節では，この言語に対して上記のアイデアのもと型システムを構築する．4 節では，型システムの健全性を形式的に述べ，その証明の概略を示す．5 節で型推論について簡単に述べる．6 節で我々のアプローチについて議論する．7 節で関連研究について触れ，8 節でまとめる．

2 対象言語 $\lambda_E^{\mathcal{R}}$

この節では，リソース生成，リソースアクセスプリミティブを導入した λ 計算 $\lambda^{\mathcal{R}}$ [10] に対し，例外機構を加えて拡張した体系 $\lambda_E^{\mathcal{R}}$ を導入する．$\lambda_E^{\mathcal{R}}$ は $\lambda^{\mathcal{R}}$ の項に $\mathbf{try}\ M_1\ \mathbf{with}\ M_2$, **raise** をつけて拡張したものである．

2.1 項

$\lambda_E^{\mathcal{R}}$ の項を定義する．その準備としてまず，**アクセスラベル**，**トレース**を定義する．

リソースに対し行われる通常の操作はアクセスラベルによって表されるものとし， このラベル (メタ変数として l_a

を用いる) の集合として可算集合 $\mathcal{L}_a$ を仮定する．例えば，初期化，読み込み，書き込み，クローズ操作に対してそれぞれ I, R, W, C といったアクセスラベルを仮定する．

また，リソースに対し行われる一連の操作列をアクセスラベルと記号 $\downarrow, \uparrow_E$ の列で表現する．この操作列の集合を

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

で表す．ここで，$\mathcal{L}_a^*$ は，$\mathcal{L}_a$ の要素の有限列の集合を表し，$\downarrow$，$\uparrow_E$ はそれぞれ一連の操作のあとに，実行が正常に終了すること，例外が発生して実行が終了すること，を表す特別な記号である．この $\mathcal{L}_a^{*,\downarrow,\uparrow_E}$ の要素を**トレース**とよぶ．

トレースはプログラムの実行過程におけるリソースに対するアクセスおよびプログラム終了，例外によるプログラムの異常終了の履歴を表す．トレース $l_{a1}l_{a2}\ldots l_{ak}$ は，プログラムの実行中，リソースに対し，各操作 l_{ai} $(i = 1, \ldots k)$ が $l_{a1}, l_{a2}, \ldots, l_{ak}$ の順序で行われたという履歴を表し，トレース $l_{a1}l_{a2}\ldots l_{ak}\downarrow$ (もしくは，$l_{a1}l_{a2}\ldots l_{ak}\uparrow_E$) は，リソースに対し，各操作 l_{ai} $(i = 1, \ldots k)$ が $l_{a1}, l_{a2}, \ldots, l_{ak}$ の順序で行われたあと正常に (もしくは，例外を発生して) 終了したと言う履歴を表す．リソースがあるトレース s に従って使用されるならば，そのリソースは s の prefix に従っても使用されなければならないことに注意．

$S \subset \mathcal{L}_a^{*,\downarrow,\uparrow_E}$ に対して，$S^\#$ で，S の要素の prefix の集合を表すこととする，つまり，$S^\# = \{s \in \mathcal{L}_a^{*,\downarrow,\uparrow_E} \mid \exists s'.ss' \in S\}$ と定義する．例えば，$(RW\downarrow)^\# = \{\epsilon, R, W, RW, RW\downarrow\}$ である．集合 S で，$S^\# \subseteq S$ を満たすものを**トレース集合**と呼び，Φ で表す．トレース集合 Φ はプログラマがリソースに対して与える仕様と見なすことができる．そのとき，仕様 Φ は，リソースが Φ の中のトレースのみに従って使用されることを指定する．

例 2.1 仕様 $\Phi = \{RWC\downarrow\}^\#$ に対して，プログラム $\mathbf{read}(x); \mathbf{write}(x); loop\_infinitely$ を考える．ここで，$\mathbf{read}(x); \mathbf{write}(x)$ はリソース x に対して，読み込み，書き込みを行う命令であり，$loop\_infinitely$ は無限にループを行う命令であるとする．このプログラムは，仕様 Φ を満たしている．このプログラムは実行が止まらず，常に実行中であり，リソース x は，$\{RW\}^\# \subset \Phi$ に従ってのみ使用される．しかし，プログラム $\mathbf{read}(x); \mathbf{write}(x)$ はこの仕様 Φ を満たさない．実際，このプログラムが終了したとき，リソース x に対し，クローズが行われておらず．そのようなトレース $(RW\downarrow)$ は仕様 Φ の中に存在しない．

定義 2.2 (項)

$$\begin{aligned} M \quad ::= \quad & \mathbf{true} \mid \mathbf{false} \mid x \mid \mathbf{fun}(f, x, M) \mid M_1 M_2 \\ \mid \quad & \mathbf{if}\ M_1\ \mathbf{then}\ M_2\ \mathbf{else}\ M_3 \\ \mid \quad & \mathbf{let}\ x = M_1\ \mathbf{in}\ M_2 \mid M^{\{x\}} \\ \mid \quad & \mathbf{new}^\Phi() \mid \mathbf{acc}^{l_a}(M) \\ \mid \quad & \mathbf{try}\ M_1\ \mathbf{with}\ M_2 \mid \mathbf{raise} \end{aligned}$$

$\lambda_E^{\mathcal{R}}$ は通常の λ 計算にリソースを作るプリミティブ $\mathbf{new}^\Phi()$ とリソースにアクセスするプリミティブ $\mathbf{acc}^{l_a}(M)$ さらに例外処理のためのプリミティブ $\mathbf{try}\ M_1\ \mathbf{with}\ M_2$，例外発生プリミティブ $\mathbf{raise}$ を追加した体系である．

$\mathbf{try}\ M_1\ \mathbf{with}\ M_2$ は M_1 を評価しその結果を返す，ただし M_1 の評価中に例外が発生したら，M_2 を評価しその結果を返す．$\mathbf{raise}$ は例外を発生するプリミティブである．

$\mathbf{fun}(f, x, M)$ は項 M の中の f が $\mathbf{fun}(f, x, M)$ 自身を指す再帰的な関数を表す項である．M が自由な変数 f を含まないとき，$\lambda x.M$ と書く．$M^{\{x\}}$ の $^{\{x\}}$ は，項 M の評価結果が，x が参照しているリソースを含まないことを表す注釈である．例えば，$\mathbf{true}^{\{x\}}, (\mathbf{acc}^R(x))^{\{x\}}$ は正しい注釈であるが，$(\lambda x.\mathbf{acc}^R(y))^{\{y\}}$ などは間違った注釈である．これは，リソースが M からエスケープしないという情報から解析結果を良くすることができるため導入しているものである．$\mathbf{acc}^{l_a}(M)$ は M を評価した結果得られるリソースに l_a で表されるアクセスを行うプリミティブである．簡単のため $\mathbf{acc}^{l_a}(M)$ が返す値は真偽値の $\mathbf{true}$ もしくは，$\mathbf{false}$ とする．以降，$\mathbf{acc}^I(x)$，$\mathbf{acc}^R(x)$，$\mathbf{acc}^W(x)$，$\mathbf{acc}^C(x)$ を，$\mathbf{init}(x)$，$\mathbf{read}(x)$，$\mathbf{write}(x)$，$\mathbf{close}(x)$ などと書くことにする．$\mathbf{new}^\Phi()$ は，トレース集合 Φ に含まれるトレースのみに従って使用されなければならないリソースを生成するプリミティブである．Φ の具体的な記述方法については，特に限定せず一般的にトレース集合としておくが，本論文では基本的に正則表現を用いて表す．また，簡単のため，リソースの種類は 1 種類のみと仮定する．その他の項は通常通りの意味を持つ．$\mathbf{let}\ x = M_1\ \mathbf{in}\ M_2$ において，x が M_2 のなかに自由に出現しないとき $M_1; M_2$ と書く．

定義 2.3 (値，値の代入) 値とは，変数 x，関数 $\mathbf{fun}(f, x, M)$，$\mathbf{true}$，$\mathbf{false}$ のことである．また，$[v_1/x_1, \ldots, v_n/x_n]$ により，変数 $x_1, \ldots, x_n$ を同時に値 $v_1, \ldots, v_n$ で置き換える代入を表すものとする．

例 2.4 次の項 M_1 は，x を初期化し，成功したら (その結果が $\mathbf{true}$ なら) x への書き込みをおこない，x を閉じる．失敗したら (結果が $\mathbf{false}$ なら) 例外を発生する項を表す．

$$\begin{aligned} M_1 \triangleq\ & \mathbf{if}\ \mathbf{init}(x)\ \ \mathbf{then}\ \ \mathbf{write}(x); \mathbf{close}(x) \\ & \qquad\qquad\ \ \mathbf{else}\ \ \mathbf{raise} \end{aligned}$$

次の項 M_2 では，M_1 の中で発生した例外を処理し x が指すリソースをクローズしているので，結果としてこのリソースは必ずクローズされる．

$$M_2 \triangleq \mathbf{try}\ M_1\ \mathbf{with\ close}(x)$$

$M \triangleq \mathbf{let}\ x = \mathbf{new}^{(I(W)^*C\downarrow)^\#}()\ \mathbf{in}\ M_2$ は，初期化されたあと，必ずクローズされなければならない書き込み専用のリソースを生成し，M_2 の中で使用している．M_2 によるこのリソースの使用方法は $\{IWC\downarrow, IC\downarrow\}$ なので，リソースに与えられている仕様 $(I(W)^*C\downarrow)^\#$ に違反した操作は行われておらず，かつ，M_2 の中で仕様で指定されている操作が全て行われている．

例 2.5 次のような *OCaml* のプログラムを考える．このプログラムは x が参照するファイルへの入力チャネルから文字を読み込み，その文字を y が参照する別のファイルへの出力チャネルに書き込んでいる．ただし，読み込み時に例外 (*End_of_File* を想定) が発生する可能性がある．そして，その例外が発生したらファイルをクローズして計算を終える．

```
try
  while (true)
    do
      let c = read_char(x) in write_char(y,c)
    done
with
  End_of_File -> close(x);close(y)
```

ここで，関数 `read_char(x)`,`write_char(y,c)` は，入力チャネルから一文字読み込む関数，出力チャネルへ一文字書き込む関数として定義されているものとする．このような記述はファイルを扱うプログラムではよく見られるものである．このプログラムは次のような項として表せる．

$$M \triangleq \mathbf{try}\ \mathbf{fun}(f, z, M_1)true\ \mathbf{with\ close}(x); \mathbf{close}(y)$$

ただし，

$$M_1 \triangleq (\mathbf{if\ read}(x)\ \mathbf{then\ true\ else\ raise}); \mathbf{write}(y); f\ true$$

であり，ここで，例外を発生させる可能性のある関数 `read_char(x)` を項 $\mathbf{if\ read}(x)\ \mathbf{then\ true\ else\ raise}$ でモデル化している．

2.2 操作的意味

続いて，$\lambda_E^{\mathcal{R}}$ の操作的意味を定義する．プログラムの実行中に生成されるリソースのアクセス情報を考慮する必要がある．このため，各リソースに今後許されるアクセス順を関連付けたヒープの概念を導入し，操作的意味はヒープと項の組の書き換えで表現する．

定義 2.6 (ヒープ) ヒープを，変数に対しトレース集合を対応させる写像と定義し，メタ変数 H で表す．一般に $dom(H) = \{x_1, \ldots, x_n\}$ かつ $H(x_i) = \Phi_i$ を満たすヒープを $\{x_1 \mapsto \Phi_1, \ldots, x_n \mapsto \Phi_n\}$ と表す（n は 0 以上の自然数）．

$\{x_1 \mapsto \Phi_1, \ldots, x_n \mapsto \Phi_n\}$ は各変数 x_i にはリソースが割り当てられており，そのリソースのトレース集合が Φ であることを表現しているヒープである．また，$dom(H_1) \cap dom(H_2) = \emptyset$ のとき，$H = H_1 \uplus H_2$ を次のように定義する．

$$dom(H) = dom(H_1) \cup dom(H_2)$$
$$H(x) = H_i(x)\ \text{ if } x \in dom(H_i)\ (i = 1, 2)$$

簡約規則は，以下の評価文脈を使って定義する．

定義 2.7 (評価文脈)

$$\begin{array}{rcl} \mathcal{E} & ::= & [\,] \mid \mathbf{if}\ \mathcal{E}\ \mathbf{then}\ M_1\ \mathbf{else}\ M_2 \\ & \mid & \mathcal{E}\ M \mid v\ \mathcal{E} \mid \mathbf{let}\ x = \mathcal{E}\ \mathbf{in}\ M \\ & \mid & \mathcal{E}^{\{x\}} \mid \mathbf{acc}^{l_a}(\mathcal{E}) \mid \mathbf{try}\ \mathcal{E}\ \mathbf{with}\ M \end{array}$$

$$\begin{array}{rcl} \mathcal{E}^{\overline{try}} & ::= & [\,] \mid \mathbf{if}\ \mathcal{E}^{\overline{try}}\ \mathbf{then}\ M_1\ \mathbf{else}\ M_2 \\ & \mid & \mathcal{E}^{\overline{try}}\ M \mid v\ \mathcal{E}^{\overline{try}} \mid \mathbf{let}\ x = \mathcal{E}^{\overline{try}}\ \mathbf{in}\ M \\ & \mid & \mathcal{E}^{\overline{try}\{x\}} \mid \mathbf{acc}^{l_a}(\mathcal{E}^{\overline{try}}) \end{array}$$

$\mathcal{E}$ は値呼びの評価文脈を表し，$\mathcal{E}^{\overline{try}}$ は例外処理プリミティブを含まない評価文脈を表す．$\mathcal{E}$, $\mathcal{E}^{\overline{try}}$ の中の $[\,]$ を M で置き換えたものを $\mathcal{E}[M]$, $\mathcal{E}^{\overline{try}}[M]$ で表す．

定義 2.8 (操作的意味) P を組 (H', M') あるいは **Error** とする．$(H, M) \rightsquigarrow P$ は図 *1* の規則を満たす最小の関係である．ただし，$\Phi^{-l_a} = \{s \mid l_a s \in \Phi\}$ とする．

図 1 中，最初の規則はリソース生成を表す規則である．ここではリソースを新しい場所 z に確保し，それに対し許されるアクセス列を項中に指定されたトレース集合とする．次のふたつの規則はリソースへのアクセスに関する規則である．リソースにラベル l_a で表される種類のアクセスがなされたとき，以後そのリソースに許されるアクセス列は，そのリソースのトレース集合の中の各トレースから先頭の l_a が取り除いた Φ^{-l_a} で表される (先頭が l_a でないトレースは捨てられる)．ここで，$\Phi^{-l_a} = \emptyset$ の場合は l_a で始まるトレースがない，つまりアクセス l_a は許されていないという意味なので，その旨を示すため **Error** へと遷移する．また一般

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

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

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

$$(H, \mathcal{E}[\mathbf{fun}(f, x, M)\ v]) \rightsquigarrow (H, \mathcal{E}[[\mathbf{fun}(f, x, M)/f, v/x]M])$$

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

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

$$\frac{x \notin \mathbf{FV}(v)}{(H, \mathcal{E}[v^{\{x\}}]) \rightsquigarrow (H, \mathcal{E}[v])}$$

$$(H, \mathcal{E}[\mathbf{try}\ v\ \mathbf{with}\ M]) \rightsquigarrow (H, \mathcal{E}[v])$$

$$(H, \mathcal{E}[\mathbf{try}\ \mathcal{E}^{\overline{try}}[\mathbf{raise}]\ \mathbf{with}\ M]) \rightsquigarrow (H, \mathcal{E}[M])$$

図 1: 操作的意味

に，$\mathcal{E}^{\overline{try}}[\mathbf{raise}]$ と表される項は例外を発生している項であり，そこからの遷移はない． ただし，最後のルールが示しているように，項 $\mathcal{E}^{\overline{try}}[\mathbf{raise}]$ が $\mathbf{try}\ \mathcal{E}^{\overline{try}}[\mathbf{raise}]\ \mathbf{with}\ M_2$ の形で出現しているとき，例外処理である M_2 の評価へと移る．

3 型システム

この節では型付された項が必ずリソースを正しく使用する項になることを保証できる型システムを構築する．そのため我々が用いた中心となるアイデアは，第 1 節でも述べたように，(1) リソースの型にそのリソースがどのような順序と種類でアクセス（例外発生もふくむ）されるのかという情報 (usage 式) を付加し，型を拡張する，(2) さらに，例外発生情報をエフェクトとしても表現し，必要に応じて各リソースの usage 式に付加することにより，リソースへのアクセス履歴の中での例外の発生場所を特定可能にする，というものである．

3.1 Usages

ここでは，usage 式を定義し，その意味を厳密に定義する．

3.1.1 Usage 式

Usage 式は各リソースがどのような順序でどのような種類のアクセスを受けるのかという情報を表す式である．これまでに述べたように，例外機構を扱うために，例外発生，例外処理の情報を usage 式として付け加える必要がある．このため，論文 [10] の usage 式に，ラベル E, 構成子 $;_E$, $\backslash E$ を新たに加える．

さらに，内部的な動作を表すラベル τ, 関数の呼び出しを表すラベル 1 を導入し，$\mathcal{L} = \mathcal{L}_a \cup \{\tau, 1, E\}$ と定義する．$\mathcal{L}$ の要素を l で表し，ラベルとよぶ．ラベルは usage 式の一種である．

Usage 式 U（以下，簡単に usage と記す) は次で定義される．

定義 3.1 (Usages)

$$\begin{array}{rcl} U & ::= & \mathbf{0} \mid l \mid \alpha \mid U_1; U_2 \mid U_1 \otimes U_2 \mid U_1 \& U_2 \\ & \mid & U_1 \odot U_2 \mid \Diamond U \mid \blacklozenge U \mid \mu\alpha.U \\ & \mid & U_1;_E U_2 \mid U \backslash E \end{array}$$

ここで，構成子 $\Diamond, \blacklozenge, \backslash E$ は他の構成子よりも優先的に結合するものとする．例えば，$\Diamond U_1; U_2$ は，usage $(\Diamond U_1); U_2$ のことを意図しており，$\Diamond(U_1; U_2)$ のことを意図しているのではない．構成子 $\otimes, \&, ;, ;_E$ はすべて左結合である．

Usage E $(\in \mathcal{L})$ は例外が発生してアクセスできなくなるリソースに対して与えられる．$U_1;_E U_2$ は，U_1 に従ってアクセスされるが，その途中に例外が発生したときには，その時点からは，U_2 に従ってアクセスされるようなリソースに対して与えられる．例えば，$((R\&E); C);_E C$ は，リードされクローズされるか，もしくは例外が発生したためすぐにクローズされるようなリソースに対して与えられる．$U \backslash E$ は，U の中の例外発生を無効化した usage に従って使用されるリソースに対して与えられる．機能的には U の中の例外ラベル E を無効にするものである．例えば $(l_1; E) \backslash E$ は l_1 と，$(l_1 \& E) \backslash E$ は $l_1 \& \mathbf{0}$ と等しい．また，$((l_1; l_2) \otimes E) \backslash E$ は，$l_1; l_2$，l_1，$\mathbf{0}$ の何れかに従ってのアクセスを表す usage ということになる．この $U \backslash E$ は関数を型付けするとき使用され (3.3 節参照)，引数のリソースの usage U の中の例外情報を消去し，関数のエフェクトとして表すために用いられる．

$E, U_1;_E U_2, U \backslash E$ 以外の usage については，論文 [10] で議論されているのでここでは簡単に説明する．

Usage $\mathbf{0}$ は，今後アクセスされないリソースに対して与えられ，l はラベル $l \in \mathcal{L} = \mathcal{L}_a \cup \{1, \tau, E\}$ で示されるアクセスが行われるリソースに対して与えられる．例えば，$R \in \mathcal{L}$ は読み込み操作が行われることを示す．α は usage 変数で

あり，再帰的な使用法を表す $\mu\alpha.U$ で束縛される．$U_1;U_2$ は，U_1 に従った使用の後に U_2 に従って使用されるリソースに与えられる．$U_1\&U_2$ は，U_1 もしくは U_2 に従った使用法，$U_1 \otimes U_2$ は，usage U_1 と U_2 をインターリーブさせて得られる usage に従った使用法を表す．$U_1 \odot U_2$ は，U_1 で表される関数の呼び出し回数だけ U_2 によって表されるアクセスが起ったとき全体としてどういうアクセスが起るかを表す usage である．例えば，$\lambda x.M$ を，自由変数 y が M の中で U_2 に従ってアクセスされような関数とする．このとき，この関数が 2 回連続で呼び出されるとすると（つまり関数 $\lambda x.M$ の usage が $1;1$）y は全体として，$(1;1) \odot U_2$ に従って使用されることとなる．$\Diamond U$ は今，もしくは今後の何れかの時点で U に従ってアクセスされるということを表す．例えば，$\Diamond l_1;l_2$ は，$l_1;l_2$，あるいは $l_2;l_1$ を表す．$\Diamond U_1;U_2$ は $\Diamond U_1 \otimes U_2$ と等しい usage とみなせる．$\blacklozenge U$ は，U に従ったアクセスが，今後のいつかではなく今起ることを表している．機能的には U の中の $\Diamond$ を無効にするものである．一般に，$\blacklozenge\Diamond U;U'$ は $U;U'$ と同じアクセス形式を表している．

例 3.2 例 *2.4* の M_2 の中で，x が参照するリソースに対して行われるアクセスは $I;((W;C)\&E);_E C$ で表される．

例 3.3 例 *2.5* の M で，x が参照するリソースに対して行われるアクセスは $\mu\alpha.(R;(\mathbf{0}\&E);\alpha);_E C$ で表される．

3.1.2 Usage の意味, 部分 usage 関係

ここでは，usage の意味をラベル遷移系を用いて形式的に定義する．直感的には，usage の意味は，その usage からの可能な遷移列に対応するラベル列の集合（つまり trace 集合）で与えられる．また，部分 usage 関係 $U_1 \leq U_2$ を定義する．この関係は，usage U_1 が U_2 よりも一般的なアクセス順序を表していることを表現するものである．

Usage の遷移規則　Usage の意味をラベル遷移系で与えるために，usage に対するラベル遷移規則 $U \xrightarrow{l} U'$ を定義する．この規則は，usage U のリソースは，まずラベル l が表す操作でもってアクセスされることができ，その場合，その後は usage U' にしたがってアクセスされるということを表す．

我々は，以下のように usage 間の構造的な合同関係を用いて usage の遷移規則の定式化を行う．この合同関係は π 計算におけるプロセスの合同関係に習ったものである．

定義 3.4 $U \equiv U'$ は，図 *2* の規則に関して閉じた最小の合同関係である．

$$\Diamond\mathbf{0} \equiv \mathbf{0} \quad \blacklozenge\mathbf{0} \equiv \mathbf{0}$$
$$\mathbf{0} \otimes U \equiv U \quad U \otimes \mathbf{0} \equiv U \quad U_1 \otimes U_2 \equiv U_2 \otimes U_1$$
$$U_1 \otimes (U_2 \otimes U_3) \equiv (U_1 \otimes U_2) \otimes U_3$$
$$U_1\&U_2 \equiv U_2\&U_1 \quad U_1\&(U_2\&U_3) \equiv (U_1\&U_2)\&U_3$$
$$\mathbf{0} \odot U \equiv \mathbf{0} \quad U \odot \mathbf{0} \equiv \mathbf{0}$$
$$\mathbf{0};U \equiv U \quad U;\mathbf{0} \equiv U \quad \mathbf{0};_E U \equiv \mathbf{0} \quad \mathbf{0}\backslash E \equiv \mathbf{0}$$
$$\Diamond U_1 \otimes \Diamond U_2 \equiv \Diamond(U_1 \otimes U_2) \quad \Diamond U_1;U_2 \equiv \Diamond U_1 \otimes U_2$$
$$U' \odot \Diamond U \equiv \Diamond(U' \odot U) \quad (\Diamond U)\backslash E \equiv \Diamond(U\backslash E)$$
$$\mu\alpha.U \equiv [\mu\alpha.U/\alpha]U$$

図 2: Usage の構造的合同関係

また，usage の前順序関係 $U \preceq U'$ を次のように定義する．

定義 3.5 $U \preceq U'$ は，以下の規則に関して閉じている最小の前合同 (*pre-congruence*) 関係である．

$$\frac{U \equiv U'}{U \preceq U'} \quad U_1\&U_2 \preceq U_1$$

例えば，$(\Diamond U_1\&U_2);U_3 \preceq \Diamond U_1 \otimes U_3$ が成り立ち，これは，U_3 が表す操作を最初に行うことができることを述べている．

Usage に対するラベル遷移規則は次のように定義される．

定義 3.6 (Usage の遷移規則) *Usage* 上の関係 $U \xrightarrow{l} U'$ は図 *3* の規則に関して閉じている最小の関係．

例 3.7 $R;(C\&E)$ は次のような遷移列を持つ：

$$R;(C\&E) \xrightarrow{R} E \xrightarrow{E} \mathbf{0}$$
$$R;(C\&E) \xrightarrow{R} C \xrightarrow{C} \mathbf{0}$$

例 3.8 $R;(C\&E);_E C$ は次のような遷移列を持つ：

$$R;(C\&E);_E C \xrightarrow{R} (C\&E);_E C \xrightarrow{C} \mathbf{0}$$
$$R;(C\&E);_E C \xrightarrow{R} (C\&E);_E C \xrightarrow{\tau} C \xrightarrow{C} \mathbf{0}$$

Usage が意味する trace 集合　ラベル列 $t \in (\mathcal{L}_a \cup \{\tau\})^*$ に対し，$\xrightarrow{t}$ を次のように定義する．

$$\xrightarrow{t} \stackrel{def}{=} \begin{cases} \preceq & \text{if } t = \epsilon \\ \xrightarrow{l_1} \ldots \xrightarrow{l_n} & \text{if } t = l_1 \cdots l_n \end{cases}$$

この遷移関係を使い，Usage U に対して，$[\![U]\!]$ は次のような trace の集合として定義される．

$$l \xrightarrow{l} \mathbf{0} \qquad \frac{U \xrightarrow{l} U'}{\Diamond U \xrightarrow{l} \Diamond U'} \qquad \frac{U \xrightarrow{l} U'}{\blacklozenge U \xrightarrow{l} \blacklozenge U'}$$

$$\frac{U_1 \xrightarrow{l} U_1'}{U_1 \otimes U_2 \xrightarrow{l} U_1' \otimes U_2} \qquad \frac{U_1 \xrightarrow{l} U_1'}{U_1; U_2 \xrightarrow{l} U_1'; U_2}$$

$$\frac{U_1 \xrightarrow{1} U_1'}{U_1 \odot U_2 \xrightarrow{\tau} U_2 \otimes (U_1' \odot U_2)}$$

$$\frac{U_1 \xrightarrow{l} U_1' \quad l \neq E}{U_1 ;_E U_2 \xrightarrow{l} U_1' ;_E U_2} \qquad \frac{U_1 \xrightarrow{E} U'}{U_1 ;_E U_2 \xrightarrow{\tau} U_2}$$

$$\frac{U \xrightarrow{l} U' \quad l \neq E}{U \backslash E \xrightarrow{l} U' \backslash E} \qquad \frac{U \xrightarrow{E} U'}{U \backslash E \xrightarrow{\tau} \mathbf{0}}$$

$$\frac{U_1 \preceq U_1' \quad U_1' \xrightarrow{l} U_2' \quad U_2' \preceq U_2}{U_1 \xrightarrow{l} U_2}$$

図 3: Usage の遷移規則

定義 3.9 (U が意味する trace 集合) トレース集合 $[\![U]\!]$ を次で定義する：

$$\begin{aligned}[\![U]\!] \quad = \quad & \{\hat{t} \mid \exists U'.(U \xrightarrow{t} U')\} \\ & \cup \{\hat{t}\downarrow \mid U \xrightarrow{t} \mathbf{0}\} \\ & \cup \{\hat{t}\uparrow_E \mid \exists U'.(U \xrightarrow{t} \xrightarrow{E} U')\}\end{aligned}$$

ただし $\hat{t} \in \mathcal{L}_a^*$ は，ラベル列 t から全ての τ を取り除いたアクセスラベル列である．

$[\![U]\!]$ の各要素であるトレースの prefix もまた $[\![U]\!]$ の要素であることに注意すると，$[\![U]\!]$ は 2 節で述べた意味でトレース集合である．

例 3.10 $[\![\mathbf{0}]\!] = \{\epsilon\}$, $[\![\mu\alpha.\alpha]\!] = \{\epsilon\}$,
$[\![\Diamond l_1; l_2]\!] = \{l_1 l_2\downarrow, l_2 l_1\downarrow\}^{\#}$, $[\![\blacklozenge\Diamond l_1; l_2]\!] = \{l_1 l_2\downarrow\}^{\#}$

例 3.11 $[\![(R\&E); C]\!] = \{RC\downarrow, \uparrow_E\}^{\#}$,
$[\![(R\&E); C);_E C]\!] = \{RC\downarrow, C\downarrow\}^{\#}$

部分 usage 関係 Usage U_1 が U_2 よりも一般的なアクセスの形式を表しているという部分 usage 関係 $U_1 \leq U_2$ を定義したい．これは論文 [10] にあるように，$[\![U_1]\!] \supseteq [\![U_2]\!]$ で定義することはできない．例えば，$[\![l]\!] \supseteq [\![\Diamond l]\!]$ であるが，l が $\Diamond l$ よりも一般的な使用を表しているとは言えない．そこで，usage U_1 が U_2 の挙動（簡約の過程で観察されるラベル列のこと) を模倣できるとき，$U_1 \leq U_2$ と定義する．この模倣の関係は π 計算 [6] に習って定義する．

Usage **文脈**を，usage の中の自由な変数を [ ] で置き換えたものと定義し，$\mathcal{C}[U]$ で $\mathcal{C}$ の中の [ ] を U で置き換えた usage とする．例えば，$\mathcal{C} = \mu\alpha.(l_1; (\alpha\&[\,]))$ ならば，$C[l_2] = \mu\alpha.(l_1; (\alpha\&l_2))$ である．

また，usage の間の遷移関係 $\Longrightarrow$, $\stackrel{l}{\Longrightarrow}$ を次のように定義する．

$$\Longrightarrow \stackrel{def}{=} \xrightarrow{\tau^*} \qquad \stackrel{l}{\Longrightarrow} \stackrel{def}{=} \Longrightarrow \xrightarrow{l} \Longrightarrow$$

定義 3.12 (部分 usage 関係 $U_1 \leq U_2$)
$U_1 \leq U_2$ は次の条件をすべて満たす最大の関係である：

(1) 任意の *usage* 文脈 $\mathcal{C}$ に対して，$\mathcal{C}[U_1] \leq \mathcal{C}[U_2]$.

(2) $U_2 \xrightarrow{l_a} U_2'$ ならば，ある U_1' に対して
$U_1 \stackrel{l_a}{\Longrightarrow} U_1'$ かつ $U_1' \leq U_2'$.

(3) $U_2 \preceq \mathbf{0}$ ならば $U_1 \Longrightarrow \mathbf{0}$.

(4) $U_2 \xrightarrow{E} U_2'$ ならば ある U_1' に対して $U_1 \stackrel{E}{\Longrightarrow} U_1'$.

$U_1 \leq U_2$ かつ $U_2 \leq U_1$ のとき，$U_1 \cong U_2$ と記す．

例 3.13 $U \leq \mu\alpha.\alpha$，$\Diamond U \leq U$ が任意の U について成り立つ．$U_1 \preceq U_2$ ならば，$U_1 \leq U_2$ が成り立つ．$U_1 \equiv U_2$ ならば，$U_1 \cong U_2$ が成り立つ．また，$E;_E C \leq C$, $(C\&E)\backslash E \leq C\&\mathbf{0}$ 等が成り立つ．

ここで，この部分 usage 関係に関して，
$U_1 \leq U_2$ ならば $[\![U_1]\!] \supseteq [\![U_2]\!]$ が成り立つ．

3.2 型

以下では，型とエフェクトを定義する．型は，リソースの型に usage を付加し，関数の型に usage とエフェクトを付加することで拡張したものである．

まず，例外発生に関する情報を表すエフェクトを定義する．

定義 3.14 (エフェクト) $\varphi \ ::= \ \mathbf{0} \mid E \mid E^? \mid \top$

エフェクトはある項についての評価の終了と例外発生に関する情報を表す．項の評価時に観測する事象を，評価の正常終了 ($\downarrow$) と例外発生 ($\uparrow_E$) とすると，各エフェクトの意味は，以下のような，項の評価時に観測しうる事象の集合として定義される．

$$\mathbf{0} = \{\downarrow\} \quad E = \{\uparrow_E\} \quad E^? = \{\downarrow, \uparrow_E\} \quad \top = \{\}$$

直感的には，E は評価が終了しないか例外が発生することを表し，$\mathbf{0}$ は評価が終了しないか正常終了すること，$\top$ は評価が終了しないことを表す．

3節のはじめに述べたように，例外発生の情報を各リソースに付け加えるため，必要に応じてエフェクトを usage として扱う．そのためエフェクトから usage への対応を与える関数として $(\cdot)^{use}$ を次のように定義する．

定義 3.15 エフェクト から *usage* への写像 $(\varphi)^{use}$ を以下で与える．

$$(\mathbf{0})^{use} = \mathbf{0} \quad (E)^{use} = E$$
$$(E^?)^{use} = \mathbf{0}\&E \quad (\top)^{use} = \mu\alpha.\alpha$$

定義 3.16 (型)

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

bool は真偽値の型である．$(\tau_1 \xrightarrow{\varphi} \tau_2, U)$ は引数として τ_1 型の値をとり，τ_2 型の値を返す関数の型であり，この関数が適用されたときエフェクト φ に従って例外が発生すること，および，この関数自身が U に従って使用される (呼び出される）ことを示している．$(\mathbf{R}, U)$ は，usage U に従って使用されるリソースに対して与えられる型である．

型，エフェクトに対して，$Use_\varphi(\tau)$ を次のように定める．

$$Use_\varphi(\mathbf{bool}) = (\varphi)^{use}$$
$$Use_\varphi((\tau_1 \xrightarrow{\varphi'} \tau_2, U)) = U$$
$$Use_\varphi((\mathbf{R}, U)) = U$$

以下，$Use(\tau)$ は $Use_{\mathbf{0}}(\tau)$ の略記であるとする．

例 3.17 $((File, ((R\&\mathbf{0}); C)) \xrightarrow{E^?} \mathbf{bool}, 1 \otimes 1)$ は引数として，リードされた後クローズされるか，もしくはすぐにクローズされるリソースを受取り，真偽値を返す関数で，かつ適用時に例外を発生する可能性がある関数の型である．$1 \otimes 1$ はこの関数の *usage* であり，この関数が *2* 回呼び出されるということを表している．

部分 usage 関係を使い部分型関係 $\tau_1 \leq \tau_2$ を定義する．通常の型システムと同様，型 τ_1 は型 τ_2 の部分型であるとは型 τ_1 の値が τ_2 型の値として使用することができる，ということを表し，これを $\tau_1 \leq \tau_2$ と表記する．

定義 3.18 (部分型関係) $\tau_1 \leq \tau_2$ は次の規則に関して閉じている最小の関係である：

$$\mathbf{bool} \leq \mathbf{bool}$$

$$\frac{U \leq U'}{(\tau_1 \xrightarrow{\varphi} \tau_2, U) \leq (\tau_1 \xrightarrow{\varphi} \tau_2, U')}$$

$$\frac{U \leq U'}{(\mathbf{R}, U) \leq (\mathbf{R}, U')}$$

3.3 型判断

型判断は

$$\Gamma \parallel \varphi \vdash M : \tau$$

の形である．ここで，Γ は変数から型への写像であり，型環境と呼ぶ．以下，メタ変数 Γ で型環境を表す．

型判断 $\Gamma \parallel \varphi \vdash M : \tau$ が述べていることは，項 M が評価されるとき，項 M の中に出現する自由変数が Γ で与えられる型に従って使用され，例外が φ に従って発生すること，および，項 M が値に評価されるならば τ 型の値に評価されるということである．

この節では，この関係を導く型判断規則を与え，関係 $\Gamma \parallel \varphi \vdash M : \tau$ を定義する．

3.3.1 型環境

上述したように，我々は $\Gamma \parallel \varphi \vdash M : \tau$ の関係式で，項 M の型 τ を判断する．ここで型環境 Γ は M の中の自由変数に対してその変数が指す値の型を指定しており，それは，その変数が M のなかでどのように使用されなければならないのかということを定めている．この型環境とエフェクトの組を $\langle \Gamma \parallel \varphi \rangle$ で表す．ただし，表記上，組であることが明らかなときは，$\langle , \rangle$ を省略して，今までの通り $\Gamma \parallel \varphi$ と書くことにする．

型判断規則を簡潔にするため以下の演算を使用する．

ただし以下で **op** は $;, \ \&, \ ;_E$ の何れかを表すものとする．

定義 3.19 (τ **op** τ，τ **op** U，U**op** τ)

bool	**op**	**bool**	$=$	**bool**
$(\tau_1 \xrightarrow{\varphi} \tau_2, U_1)$	**op**	$(\tau_1 \xrightarrow{\varphi} \tau_2, U_2)$	$=$	$(\tau_1 \xrightarrow{\varphi} \tau_2, U_1 \mathbf{op} U_2)$
$(\mathbf{R}, U_1)$	**op**	$(\mathbf{R}, U_2)$	$=$	$(\mathbf{R}, U_1 \mathbf{op} U_2)$
bool	**op**	U'	$=$	**bool**
$(\tau_1 \xrightarrow{\varphi} \tau_2, U)$	**op**	U'	$=$	$(\tau_1 \xrightarrow{\varphi} \tau_2, U \mathbf{op} U')$
$(\mathbf{R}, U)$	**op**	U'	$=$	$(\mathbf{R}, U \mathbf{op} U')$
U'	**op**	**bool**	$=$	**bool**
U'	**op**	$(\tau_1 \xrightarrow{\varphi} \tau_2, U)$	$=$	$(\tau_1 \xrightarrow{\varphi} \tau_2, U' \mathbf{op} U)$
U'	**op**	$(\mathbf{R}, U)$	$=$	$(\mathbf{R}, U' \mathbf{op} U)$

上に示されていないものについては定義されない．

定義 3.20 ($\tau \backslash E$，$\Diamond\tau$)

$(\tau_1 \xrightarrow{\varphi} \tau_2, U) \backslash E$	$=$	$(\tau_1 \xrightarrow{\varphi} \tau_2, U \backslash E)$
$(\mathbf{R}, U) \backslash E$	$=$	$(\mathbf{R}, U \backslash E)$
$\Diamond(\tau_1 \xrightarrow{\varphi} \tau_2, U)$	$=$	$(\tau_1 \xrightarrow{\varphi} \tau_2, \Diamond U)$
$\Diamond(\mathbf{R}, U)$	$=$	$(\mathbf{R}, \Diamond U)$

この $\tau\backslash E$, $\diamond\tau$ において，上で示されていないもの ($\diamond$**bool** など) については τ とする．

定義 3.21 ($U\odot\Gamma$, $\Gamma\backslash E$，$\diamond\Gamma$，$\blacklozenge_x\Gamma$)

$$
\begin{array}{lcl}
U\odot\Gamma(x) & = & U\odot\Gamma(x) \quad \mathit{if}\ x\in \mathit{dom}(\Gamma)\\
(\Gamma\backslash E)(x) & = & \Gamma(x)\backslash E \quad \mathit{if}\ x\in \mathit{dom}(\Gamma)\\
(\diamond\Gamma)(x) & = & \diamond\Gamma(x) \quad \mathit{if}\ x\in \mathit{dom}(\Gamma)\\
(\blacklozenge_x\Gamma)(y) & = & \begin{cases} (\mathbf{R},\blacklozenge U) & \mathit{if}\ y=x,\ \Gamma(x)=(\mathbf{R},U)\\ \Gamma(y) & \mathit{if}\ y\neq x,\ y\in\mathit{dom}(\Gamma)\end{cases}
\end{array}
$$

定義 3.22 (エフェクトの演算) エフェクトの二項演算 $\varphi_1\&\varphi_2$，$\varphi_1;\varphi_2$，$\varphi_1;_E\varphi_2$ を以下のように定義する．

$$
\varphi_1\&\varphi_2=\begin{cases}\mathbf{0} & \mathit{if}\ \varphi_1=\varphi_2=\mathbf{0}\\ E & \mathit{if}\ \varphi_1=\varphi_2=E\\ \varphi_1 & \mathit{if}\ \varphi_2=\top\\ \varphi_2 & \mathit{if}\ \varphi_1=\top\\ E^? & \text{上記以外}\end{cases}
$$

$$
\varphi_1;\varphi_2=\begin{cases}\varphi_2 & \mathit{if}\ \varphi_1=\mathbf{0}\\ E & \mathit{if}\ \varphi_1=E\\ \varphi_2\&E & \mathit{if}\ \varphi_1=E^?\\ \top & \mathit{if}\ \varphi_1=\top\end{cases}
$$

$$
\varphi_1;_E\varphi_2=\begin{cases}\mathbf{0} & \mathit{if}\ \varphi_1=\mathbf{0}\\ \varphi_2 & \mathit{if}\ \varphi_1=E\\ \mathbf{0}\&\varphi_2 & \mathit{if}\ \varphi_1=E^?\\ \top & \mathit{if}\ \varphi_1=\top\end{cases}
$$

$\varphi_1;\varphi_2$ は φ_1 に従って例外が起き，引き続いて φ_2 に従って例外が起きるとき，全体としてどのような例外が起きるのかということを表す．$\varphi_1\&\varphi_2$, $\varphi_1;_E\varphi_2$ についても同様である．

定義 3.23 (型環境とエフェクトの組の間の演算) 型環境とエフェクトの組 $\langle\Gamma\parallel\varphi\rangle$ の間の演算を次で定義する．

$$
\langle\Gamma_1\parallel\varphi_1\rangle\ \mathbf{op}\ \langle\Gamma_2\parallel\varphi_2\rangle\ =\ \langle\Gamma_{12}\parallel\varphi_1\mathbf{op}\varphi_2\rangle
$$

$$
\langle\Gamma\parallel\varphi\rangle\ \mathbf{op}\ \varphi'\ =\ \langle\Gamma\mathbf{op}(\varphi')^{use}\parallel\varphi\mathbf{op}\varphi'\rangle
$$

ここで，Γ_{12} は次のように定義される．

$$
\Gamma_{12}(x)=\begin{cases}\Gamma_1(x)\ \mathbf{op}\ \Gamma_2(x) & x\in\mathit{dom}(\Gamma_1)\cap\mathit{dom}(\Gamma_2)\\ \Gamma_1(x)\ \mathbf{op}\ (\varphi_2)^{use} & x\in\mathit{dom}(\Gamma_1)\setminus\mathit{dom}(\Gamma_2)\\ (\varphi_1)^{use}\ \mathbf{op}\ \Gamma_2(x) & x\in\mathit{dom}(\Gamma_2)\setminus\mathit{dom}(\Gamma_1)\end{cases}
$$

例えば $\langle\Gamma_1\parallel\varphi_1\rangle;\langle\Gamma_2\parallel\varphi_2\rangle$ は，各リソースが型環境 Γ_1 に従ってアクセスされかつ φ_1 で表される例外が起ったのち，型環境 Γ_2 に従ってアクセスが起こり，例外が φ_2 に従って起きたとき，各リソースが全体としてどのようにアクセスされるのか，そして，全体としてどのように例外が発生するのかということを表す組である．

例 3.24 $\Gamma_1=x:(\mathbf{R},C\&\mathbf{0}),y:(\mathbf{R},\mathbf{0})$, $\varphi_1=E^?$, $\Gamma_2=x:(\mathbf{R},C),z:(\mathbf{R},\mathbf{0})$, $\varphi_2=\mathbf{0}$ とする．このとき，

$$
\begin{array}{l}\langle\Gamma_1\parallel\varphi_1\rangle;_E\langle\Gamma_2\parallel\varphi_2\rangle=\\ x:(\mathbf{R},(C\&\mathbf{0});_E C),y:(\mathbf{R},\mathbf{0};_E\mathbf{0}),z:(\mathbf{R},(\mathbf{0}\&E);_E\mathbf{0})\parallel\mathbf{0}\end{array}
$$

である．

3.3.2 型判断規則

以上の準備をもとに型判断関係 $\Gamma\parallel\varphi\vdash M:\tau$ を型判断規則を用いて定義する．

定義 3.25 (型判断) 型判断関係 $\Gamma\parallel\varphi\vdash M:\tau$ は図 *4* の規則に関して閉じている最小の関係である．

規則 (T-Raise), (T-Try) 以外の規則はエフェクト部分を除いて論文 [10] と同様である．ここでは，主に拡張部分について，いくつかの規則を説明する．

規則 (T-Raise): **raise** は例外発生命令であるので，例外が発生するという条件のもとでしか型付けできない．よってエフェクトは E でなければならない．

規則 (T-Fun): 簡単のため f が M に自由に現れていない場合，つまり，$\lambda x.M$ について考える．$\lambda x.M$ の型付けは次のようになる．

$$
\frac{\Gamma,x:\tau_1\parallel\varphi\vdash M:\tau_2\quad\tau_1'\leq(\tau_1\backslash E)}{U_1\odot(\diamond(\Gamma\backslash E))\parallel\mathbf{0}\vdash\lambda x.M:(\tau_1'\xrightarrow{\varphi}\tau_2,U_1)}
$$

前提の M を評価したとき，例外が φ に従って起るが，関数 $\lambda x.M$ を評価したときにはその例外は発生しない．よって，関数が評価される時点では，型環境の中の自由変数が指すリソース，x が指すリソースとも，例外の影響はうけない．規則の結論部のエフェクト $\mathbf{0}$, 型環境 $(\Gamma\backslash E)$, 規則の条件部の $\tau_1'\leq(\tau_1\backslash E)$ はこのことを述べている．この例外の情報は latent エフェクトとして関数の型の中で示され，関数の呼び出し時に型環境中の各リソースの usage に付加される (規則 T-App)．その際，この規則の $\backslash E$ 部分がなければ，例外発生 E を重複して数えてしまうことになることに注意．usage U_1 は，この関数 $\lambda x.M$ が外から何度呼ばれるかと言う情報を表している (例えば，$U_1=1\otimes1$ なら 2 回)．関数が呼び出される度に，各リソースは $(\diamond(\Gamma\backslash E)$ に従いアクセスされる．よって，U_1 回呼び出されるのであれば，各リソースは今後，$U_1\odot(\diamond(\Gamma\backslash E))$ に従って使用されることになる ($U_1=1\otimes1$ ならば，$U_1\odot(\diamond(\Gamma\backslash E))=\diamond(\Gamma\backslash E)\otimes\diamond(\Gamma\backslash E)$)．規則 (T-Fun) において，一般に $\mathbf{fun}(f,x,M)$ は再帰関数であるので，この呼び出し回数の部分が $\lambda x.M$ と異なる．規則 (T-Fun) 中，U_2 は関数 $\mathbf{fun}(f,x,M)$ が外側から何度呼ばれる

かという情報を表しており，U_1 は関数 $f(=\mathbf{fun}(f,x,M))$ が呼び出されたとき，M の中で，自分自身を再帰的に何度呼び出すかと言う情報を表している．このとき，この関数は合計 $U_2 \odot \mu\alpha.(1 \otimes (U_1 \odot \alpha))$ 回呼び出されると言う情報をこの規則の結論部分は表している．

規則 (T-App): M_1M_2 の評価は，まず M_1 が評価され，次に M_2 が評価されて，その後，関数適用がおこる．よって M_1M_2 の中のリソースは，まず，Γ_1 に従ってアクセスされ，M_1 を評価したときのエフェクト φ_1 に従った例外が生じる，その後，Γ_2, φ_2 に従って，アクセス，例外が起きた後，関数の適用時に，関数の latent エフェクト φ_3 に従って例外が生じることとなる．(T-App) 規則の結論の型環境部分はこの事を述べている．

規則 (T-Now): 我々の型システムは，適当なエスケープ解析と組み合わせることで解析の精度を上げている．簡単のため，我々はある種のエスケープ解析が正しく行われ，その結果が，項 $M^{\{x\}}$ の形でプログラム中に正しく注釈されていることを仮定する．簡単なエスケープ解析の一つは，項 M の (通常の) 型を調べることで行うことができる [4, 9]. M の評価結果に x が現れていないのであれば，M の評価時以降に x が使用されることはない．よって，型環境中の，$x : (\mathbf{R}, \Diamond U)$ の $\Diamond$ を $x : (\mathbf{R}, \blacklozenge\Diamond U)$ とすることで取り除くことができる．

規則 (T-Weak)：この規則の中で使われている $\Gamma_1 \leq_\varphi \Gamma_2$ は，$dom(\Gamma_2) \subseteq dom(\Gamma_1)$ のとき，次のように定義される．

$$\Gamma_1 \leq_\varphi \Gamma_2 \quad \Leftrightarrow \quad \begin{cases} \Gamma_1(x) \leq \Gamma_2(x) & \text{for all } x \in dom(\Gamma_2) \\ Use_\varphi(\Gamma_1(x)) \leq (\varphi)^{use} & \text{for all } x \in dom(\Gamma_1) \setminus dom(\Gamma_2) \end{cases}$$

ただし，$\varphi \leq \varphi'$ は次を満たす最小の関係である．

$$\varphi \leq \varphi \quad E^? \leq \mathbf{0} \quad E^? \leq E \quad \varphi \leq \top$$

$\Gamma' \parallel \varphi' \vdash M : \tau'$ は，例外が φ' に従って発生するという条件のもとでの型付けである．そのため，リソースを参照する変数を環境 Γ' に付け加えるとき，その変数の型は φ' を反映していないとならない．このことを述べているのが，$\Gamma \leq_{\varphi'} \Gamma'$ の条件である．

例 3.26 $\mathbf{let}\ x = \mathbf{new}^{(C\downarrow)^\#}()\ \mathbf{in}\ \mathbf{true}$ は型付けできない．実際，規則 (T-Let),(T-Const) より $x : (\mathbf{R}, C) \parallel \mathbf{0} \vdash \mathbf{true} : \mathbf{bool}$ が成り立つ必要があるが，(T-Const),(T-Weak) より，このような型付けは導けない．また，同様に (T-Weak) 規則により $x : (\mathbf{R}, \mathbf{0}) \parallel E \not\vdash \mathbf{raise} : \tau$ であることがわかる．

例 3.27 例 *2.4* の項 M_1, M_2, M について，次が成り立つ．

$$x : (\mathbf{R}, I; ((W; C)\&E)) \parallel E^? \vdash M_1 : \mathbf{bool}$$
$$x : (\mathbf{R}, I; ((W; C)\&E));_E C \parallel \mathbf{0} \vdash M_2 : \mathbf{bool}$$
$$\emptyset \parallel \mathbf{0} \vdash M : \mathbf{bool}$$

例 3.28 例 *2.5* の M の型付けは次のようになる．この型付けにより，M の中で x, y が保持するリソースは何れも M の評価終了時には，必ずクローズされていることが導ける．まず，M に対して適当なエスケープ解析を行い，その結果を注釈としてつけたものを改めて M，M_1 と置く．ここでは，これらの項に対して型付けを行っていく．

$\mathbf{read}(x)^{\{x\}}$ 部分は，(T-Var)，(T-Acc)，(T-Now)，(T-Weak) 規則を順に使い，次のように型付けられる．

$$\dfrac{\dfrac{\dfrac{x : (\mathbf{R}, \Diamond R) \parallel \mathbf{0} \vdash x : (\mathbf{R}, R)}{x : (\mathbf{R}, \Diamond R) \parallel \mathbf{0} \vdash \mathbf{read}(x) : \mathbf{bool}}}{x : (\mathbf{R}, \blacklozenge\Diamond R) \parallel \mathbf{0} \vdash \mathbf{read}(x)^{\{x\}} \vdash \mathbf{bool}}}{x : (\mathbf{R}, \blacklozenge\Diamond R) \parallel E^? \vdash \mathbf{read}(x)^{\{x\}} \vdash \mathbf{bool}}$$

ここで，さらに (T-Weak) 規則を適用すると，

$x : (\mathbf{R}, R) \parallel E^? \vdash \mathbf{read}(x)^{\{x\}} \vdash \mathbf{bool}$ が得られる．$\mathbf{write}(y)^{\{y\}}, \mathbf{close}(x)^{\{x\}}, \mathbf{close}(y)^{\{y\}}$ もまったく同様に型付けできる．$\mathbf{true}, \mathbf{raise}$ は，(T-Const),(T-Raise),(T-Weak) 規則を使い，次のように型付けできる．

$$\dfrac{\emptyset \parallel \mathbf{0} \vdash \mathbf{true} : \mathbf{bool}}{\emptyset \parallel E^? \vdash \mathbf{true} : \mathbf{bool}} \quad \dfrac{\emptyset \parallel E \vdash \mathbf{raise} : \mathbf{bool}}{\emptyset \parallel E^? \vdash \mathbf{raise} : \mathbf{bool}}$$

これらを使い，M の部分式を順に型付けていく．まず，M_1 の部分式である $M_2 \triangleq \mathbf{if}\ \mathbf{read}(x)^{\{x\}}\ \mathbf{then}\ \mathbf{true}\ \mathbf{else}\ \mathbf{raise}$ は (T-If) 規則より，次のように型づけられる．

$$\Pi_1 = \dfrac{\begin{array}{c} x : (\mathbf{R}, R) \parallel E^? \vdash \mathbf{read}(x)^{\{x\}} : \mathbf{bool} \\ \emptyset \parallel E^? \vdash \mathbf{true} : \mathbf{bool} \quad \emptyset \parallel E^? \vdash \mathbf{raise} : \mathbf{bool} \end{array}}{x : (\mathbf{R}, R; (\mathbf{0}\&E)) \parallel E^? \vdash M_2 : \mathbf{bool}}$$

規則 (T-Let) により，$M_1 \triangleq M_2; \mathbf{write}(y)^{\{y\}}; f\ \mathbf{true}$ が次のように型付けられる．M_1 の中で f は一回呼び出されているので，f の *usage* は 1 である．

$$\Pi_2 = \dfrac{\Pi_1 \quad y : (\mathbf{R}, W) \parallel \mathbf{0} \vdash \mathbf{write}(y)^{\{y\}} : \mathbf{bool}}{\begin{array}{l} f : (\mathbf{bool} \xrightarrow{E} \mathbf{bool}, 1), \\ x : (\mathbf{R}, R; (\mathbf{0}\&E); E), \\ y : (\mathbf{R}, (\mathbf{0}\&E); W; E) \end{array} \quad \parallel E \vdash M_1 : \mathbf{bool}}$$

以降，$U_x = R; (\mathbf{0}\&E); E$, $U_y = (\mathbf{0}\&E); W; E$ と略記する．$U_x \setminus E, U_y \setminus E$ は R, W と意味的に等しい．$\mathbf{fun}(f, z, M_1)$ の型付けは (T-Fun) より次のようになる．

$$\Pi_3 = \dfrac{\Pi_2}{\begin{array}{l} (1 \odot \mu\alpha.(1 \otimes (1 \odot \alpha))) \odot \Gamma' \quad \parallel \mathbf{0} \\ \quad \vdash \mathbf{fun}(f, z, M_1) : (\mathbf{bool} \xrightarrow{E} \mathbf{bool}, 1) \end{array}}$$

ただし，$\Gamma' = x : (\mathbf{R}, \Diamond(U_x \backslash E)), y : (\mathbf{R}, \Diamond(U_y \backslash E))$ である．つぎに，$\mathbf{fun}(f, z, M_1)\mathbf{true}$ の型付けを行う．Π_3 結論部の型環境を $x : (\mathbf{R}, U'_x), y : (\mathbf{R}, U'_y)$ と略記する．ただし，$U'_x = (\mu\alpha.1 \otimes \alpha) \odot \Diamond(U_x \backslash E), U'_y = (\mu\alpha.1 \otimes \alpha) \odot \Diamond(U_y \backslash E)$ である ($(1 \odot \alpha = \alpha)$ に注意)．型付けは (T-App) より，次のようになる．

$$\Pi_4 = \frac{\Pi_3}{\begin{array}{l} x : (\mathbf{R}, U'_x; E), \\ y : (\mathbf{R}, U'_y; E) \end{array} \parallel E \vdash \mathbf{fun}(f, z, M_1)\mathbf{true} : \mathbf{bool}}$$

$M \triangleq \mathbf{try}\ \mathbf{fun}(f, z, M_1)\mathbf{true}\ \mathbf{with}\ \mathbf{close}(x)^{\{x\}}; \mathbf{close}(y)^{\{y\}}$ は (T-Try) を使い次のように型付けられる．

$$\frac{\Pi_4 \quad \begin{array}{l} x : (\mathbf{R}, C) \parallel \mathbf{0} \vdash \mathbf{close}(x)^{\{x\}} : \mathbf{bool} \\ y : (\mathbf{R}, C) \parallel \mathbf{0} \vdash \mathbf{close}(y)^{\{y\}} : \mathbf{bool} \end{array}}{\begin{array}{l} x : (\mathbf{R}, (U'_x; E);_E C), \\ y : (\mathbf{R}, (U'_y; E);_E C) \end{array} \parallel \mathbf{0} \vdash M : \mathbf{bool}}$$

この型付けにより，M を評価する際，リソース x に対しては $U'_x; E;_E C$ で表されるアクセスが起ることが示される．

$$\begin{array}{rcl} U'_x & = & (\mu\alpha.1 \otimes \alpha) \odot \Diamond(U_x \backslash E) \\ & = & \Diamond(U_x \backslash E) \otimes \Diamond(U_x \backslash E) \otimes \cdots \\ & \cong & \Diamond R \otimes \Diamond R \otimes \cdots \end{array}$$

であるので，この *usage* について，

$$[\![U'_x; E;_E C]\!] = [\![(\Diamond R \otimes \Diamond R \otimes \cdots ; E);_E C]\!] = (R^* C \downarrow)^{\#}$$

であり，同様に変数 y が保持するリソースの *usage* について $[\![U'_y; E;_E C]\!] = (W^* C \downarrow)^{\#}$ である．

4 型システムの健全性

3 節で定義した型システムは，型付けされた項はリソースを正しく ($\mathbf{new}^{\Phi}()$ の Φ で指定した通りに) 使用する，という意味で健全である．本節では，まず，我々の型システムの健全性を形式的に述べ (4.1 節)，その証明の概略を示す (4.2 節)．

4.1 型判断関係の健全性

定理 4.1 (型判断関係の健全性)
$\emptyset \parallel \varphi \vdash M : \tau$ かつ $Use_{\mathbf{0}}(\tau) \leq \mathbf{0}$, ならば

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

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

(3) $(\{\}, M) \rightsquigarrow^* (H, \mathcal{E}^{\overline{try}}[\mathbf{raise}])$ ならば
$\forall x \in dom(H).\uparrow_E \in H(x)$

が成立する．

$Use_{\mathbf{0}}(\tau) \leq \mathbf{0}$ という条件は，M の評価結果がリソースであっても，そのリソースは M の評価以後アクセスされない可能性があるということを示している．条件 (2)(3) は，プログラムの実行終了時には，実行中に生成されたリソースが trace 集合として指定された仕様通りに使われたことを述べている．なお，項 M が型付けされるなら常に (例え $Use_{\mathbf{0}}(\tau) \not\leq \mathbf{0}$ であっても)，(1) が成り立つ．

4.2 健全性定理の証明の概要

証明の概略は論文 [10] と同様，次の手順でなされる．まず，各リソースが項の中の どの部分 でどのようにアクセスされるのかといった情報を明示的に表すための，動的な式と呼ばれる拡張した項と操作的意味論を導入する．そして，この拡張した意味論と元の意味論との等価性，および，拡張した意味論に対する健全性を示していく．

動的な式とは，構文的には，2 節で定義した通常の項に $\mathbf{let}_{\mathbf{R}}\ x : U\ \mathbf{in}\ D$ という形式の項を導入したものである．この項は，変数 x が参照するリソースは，式 D の中で usage U に従って使用される，ということを表し，通常の項の簡約で使われるヒープの情報だけでなく，リソースの局所的な使用方法をも表現している．動的な式を形式的に以下のように定義する．

定義 4.2 (動的な式)

$$\begin{array}{rcl} D & ::= & \mathbf{let}_{\mathbf{R}}\ x : U\ \mathbf{in}\ D \\ & | & \mathbf{true} \mid \mathbf{false} \mid x \mid \mathbf{fun}(f, x, M) \mid D_1 D_2 \\ & | & \mathbf{if}\ D_1\ \mathbf{then}\ D_2\ \mathbf{else}\ D_3 \mid \mathbf{let}\ x = D_1\ \mathbf{in}\ D_2 \\ & | & D^{\{x\}} \mid \mathbf{new}^{\Phi}() \mid \mathbf{acc}^{l_a}(D) \\ & | & \mathbf{try}\ D_1\ \mathbf{with}\ D_2 \mid \mathbf{raise} \end{array}$$

動的な式の型付け規則は，図 4 に示した，通常の項のための規則の M をすべて D に変えた規則 (規則名は T-Name を TD-Name とする) に，以下の $\mathbf{let}_{\mathbf{R}}$ のための規則を追加して与えられる．

$$\frac{\Gamma, x : (\mathbf{R}, U) \parallel \varphi \vdash D : \tau}{\Gamma \parallel \varphi \vdash \mathbf{let}_{\mathbf{R}}\ x : U\ \mathbf{in}\ D : \tau}\ (\text{TD-LetR})$$

通常の項のための型判断と区別するために，以降では，動的な式のための型判断を $\Gamma \parallel \varphi \vdash_{\mathrm{dyn}} D : \tau$ と書く．

動的な式の簡約規則は次の評価文脈を使い与えられる．

$$\frac{c = \textbf{true} \text{ or } \textbf{false}}{\emptyset \parallel \textbf{0} \vdash c : \textbf{bool}} \quad \text{(T-Const)}$$

$$x : \Diamond\tau \parallel \textbf{0} \vdash x : \tau \quad \text{(T-Var)}$$

$$\frac{[\![U]\!] \subseteq \Phi}{\emptyset \parallel \textbf{0} \vdash \textbf{new}^{\Phi}() : (\textbf{R}, U)} \quad \text{(T-New)}$$

$$\frac{\Gamma \parallel \varphi \vdash M : (\textbf{R}, l_a)}{\Gamma \parallel \varphi \vdash \textbf{acc}^{l_a}(M) : \textbf{bool}} \quad \text{(T-Acc)}$$

$$\frac{\Gamma_1 \parallel \varphi_1 \vdash M_1 : \textbf{bool} \quad \Gamma_2 \parallel \varphi_2 \vdash M_2 : \tau \quad \Gamma_2 \parallel \varphi_2 \vdash M_3 : \tau}{\langle \Gamma_1 \parallel \varphi_1 \rangle; \langle \Gamma_2 \parallel \varphi_2 \rangle \vdash \textbf{if } M_1 \textbf{ then } M_2 \textbf{ else } M_3 : \tau} \quad \text{(T-If)}$$

$$\frac{\Gamma_1 \parallel \varphi_1 \vdash M_1 : \tau_1 \quad \Gamma_2, x : \tau_1 \parallel \varphi_2 \vdash M_2 : \tau_2}{\langle \Gamma_1 \parallel \varphi_1 \rangle; \langle \Gamma_2 \parallel \varphi_2 \rangle \vdash \textbf{let } x = M_1 \textbf{ in } M_2 : \tau_2} \quad \text{(T-Let)}$$

$$\frac{\Gamma_1 \parallel \varphi_1 \vdash M_1 : \tau_1 \xrightarrow{\varphi_3} \tau_2 \quad \Gamma_2 \parallel \varphi_2 \vdash M_2 : \tau_1}{\langle \Gamma_1 \parallel \varphi_1 \rangle; \langle \Gamma_2 \parallel \varphi_2 \rangle; \varphi_3 \vdash M_1 M_2 : \tau_2} \quad \text{(T-App)}$$

$$\frac{\Gamma, f : (\tau_1' \xrightarrow{\varphi} \tau_2, U_1), x : \tau_1 \parallel \varphi \vdash M : \tau_2 \quad \tau_1' \leq \tau_1 \setminus E}{(U_2 \odot \mu\alpha.(1 \otimes (U_1 \odot \alpha))) \odot \Diamond(\Gamma \setminus E) \parallel \textbf{0} \vdash \textbf{fun}(f, x, M) : (\tau_1' \xrightarrow{\varphi} \tau_2, U_2)} \quad \text{(T-Fun)}$$

$$\frac{\Gamma \parallel \varphi \vdash M : \tau}{\blacklozenge_x \Gamma \parallel \varphi \vdash M^{\{x\}} : \tau} \quad \text{(T-Now)}$$

$$\emptyset \parallel E \vdash \textbf{raise} : \tau \quad \text{(T-Raise)}$$

$$\frac{\Gamma_1 \parallel \varphi_1 \vdash M_1 : \tau \quad \Gamma_2 \parallel \varphi_2 \vdash M_2 : \tau}{\langle \Gamma_1 \parallel \varphi_1 \rangle ;_E \langle \Gamma_2 \parallel \varphi_2 \rangle \vdash \textbf{try } M_1 \textbf{ with } M_2 : \tau} \quad \text{(T-Try)}$$

$$\frac{\varphi \leq \varphi' \quad \Gamma \leq_{\varphi} \Gamma' \quad \Gamma' \parallel \varphi' \vdash M : \tau \quad \tau' \leq \tau}{\Gamma \parallel \varphi \vdash M : \tau} \quad \text{(T-Weak)}$$

図 4: 型判断規則

定義 4.3 (動的な式の評価文脈)

$$\begin{aligned}\mathcal{E}_{\mathcal{D}} \quad ::= \quad & [\,] \mid \mathbf{let_R}\ x : U\ \mathbf{in}\ \mathcal{E}_{\mathcal{D}} \mid \mathbf{if}\ \mathcal{E}_{\mathcal{D}}\ \mathbf{then}\ D_1\ \mathbf{else}\ D_2 \\ \mid \quad & \mathcal{E}_{\mathcal{D}} D \mid (\mathbf{let_R}\ \tilde{x} : \tilde{U}\ \mathbf{in}\ v)\mathcal{E}_{\mathcal{D}} \\ \mid \quad & \mathbf{acc}^{l_a}(\mathcal{E}_{\mathcal{D}}) \mid \mathbf{let}\ x = \mathcal{E}_{\mathcal{D}}\ \mathbf{in}\ D \mid \mathcal{E}_{\mathcal{D}}{}^{\{x\}} \\ \mid \quad & \mathbf{try}\ \mathcal{E}_{\mathcal{D}}\ \mathbf{with}\ D\end{aligned}$$

$$\begin{aligned}\mathcal{E}_D^{\overline{try}} \quad ::= \quad & [\,] \mid \mathbf{let_R}\ x : U\ \mathbf{in}\ \mathcal{E}_D^{\overline{try}} \\ \mid \quad & \mathbf{if}\ \mathcal{E}_D^{\overline{try}}\ \mathbf{then}\ D_1\ \mathbf{else}\ D_2 \\ \mid \quad & \mathcal{E}_D^{\overline{try}} D \mid (\mathbf{let_R}\ \tilde{x} : \tilde{U}\ \mathbf{in}\ v)\mathcal{E}_D^{\overline{try}} \\ \mid \quad & \mathbf{acc}^{l}(\mathcal{E}_D^{\overline{try}}) \mid \mathbf{let}\ x = \mathcal{E}_D^{\overline{try}}\ \mathbf{in}\ D \mid \mathcal{E}_D^{\overline{try}\{x\}}\end{aligned}$$

例外プリミティブ以外の規則は論文 [10] と同様であるので紙面の都合上主要な規則のみ示す．Q を動的な式，あるいは **Error** とするとき，動的な式の操作的意味 $D \xrightarrow{\xi} Q$ は，例えば図 5 などの規則で定義される．この図の中で $\tilde{x}, \tilde{U}$ は，変数，usage 式のベクターを表す．また，$\xrightarrow{\epsilon}$ は通常の項の簡約に相当する簡約を表し (例えば最後の規則は通常の例外処理の簡約に相当する)，$\xrightarrow{x}$ はリソース x の usage を x を含む各部分式に分割して割り当てるという操作を表している，このステップは通常の項の簡約では明示されない．対応関係を考慮するため，$D \Longrightarrow Q$ を $D \xrightarrow{x_1} \cdots \xrightarrow{x_n} \xrightarrow{\epsilon} Q$ で定義する．$D \Longrightarrow Q$ が通常の項の簡約関係の 1 ステップに対応する．

ここで、次の意味で 2 つの意味論が等価であることが示せる．(1) 動的な式の評価列に対して，対応する通常の項の評価列が存在する．(2) 動的な式が **Error** に評価されない評価列を持つなら，対応する通常の項の評価列も **Error** に評価されない．(3) 動的な式が $\mathcal{E}_D^{\overline{try}}[\mathbf{raise}]$ に評価されない評価列を持つなら，対応する通常の項の評価列も $\mathcal{E}_D^{\overline{try}}[\mathbf{raise}]$ に評価されない．

定理 4.1 の証明の方針　動的な式について型付け規則より

補題 4.4

(1) $\Gamma \parallel \varphi \vdash_{\mathrm{dyn}} v : \tau$ ならば $\varphi \leq \mathbf{0}$

(2) $\Gamma \parallel \varphi \vdash_{\mathrm{dyn}} \mathcal{E}_D^{\overline{try}}[\mathbf{raise}] : \tau$ ならば $\varphi \leq E$

が成り立つことを使い，定理 4.1 の各条件は次のようにして示すことができる．

まず，D の操作的意味論について，安全性 ($\Gamma \parallel \varphi \vdash_{\mathrm{dyn}} D : \tau$ なら，D は **Error** に簡約されないということ) と，subject reduction が成り立つことが示せる．これらと，2 つの意味論の等価性より条件 (1) が示される．

条件 (2) については次のように示せる．$(\{\}, M) \rightsquigarrow^* (H, v)$ かつ $\emptyset \parallel \varphi \vdash v : \tau$ ならば，2 つの意味論の等価性より，動的な式の簡約列 $M \Longrightarrow^* \mathbf{let_R}\ \tilde{x} : \tilde{U}\ \mathbf{in}\ v$ が存在し，$\emptyset \parallel \varphi \vdash \mathbf{let_R}\ \tilde{x} : \tilde{U}\ \mathbf{in}\ v : \tau$ が成り立つ．ここで，上の補題 4.4 と，TD-LetR 規則の inversion により $\tilde{x} : (\tilde{\mathbf{R}}, \tilde{U}) \parallel \varphi \vdash v : \tau,\ \varphi \leq \mathbf{0}$ が成り立つ必要がある．$Use_{\mathbf{0}}(\tau) \leq \mathbf{0}$ ならば，(値の) 型付け規則と T-Weak 規則より，$\tilde{U}$ の中の usage U_i はすべて $U_i \leq \mathbf{0}$ であることが導け，その結果，定義 3.9 より，$\downarrow \in [\![U_i]\!]$ が導ける．((3) についても同様である．)

5　型推論

型推論は通常通り，次のステップで行なうことができる．

(1) 型制約、エフェクト制約を型付け規則に従って取り出す．(2) 型制約，エフェクト制約を解く，つまり，(2-1) まず型制約，エフェクト制約から usage 制約を取り出す．(2-2)usage 式制約を解き，仕様 Φ を満たしているかどうかを検査する．

(1) で取り出されるエフェクト制約は $\psi_k \leq \xi_k \quad (k = 1, \ldots, m)$ という形をしている．ここで，ψ はエフェクト変数を表す．ξ は，エフェクト制約式であり次で定義される．

定義 5.1 (エフェクト制約式)

$$\xi \quad ::= \quad \psi \mid \mathbf{0} \mid E \mid E^{?} \mid \xi;\xi \mid \xi \& \xi \mid \xi;_E \xi$$

(2-1) では，まずエフェクトを決定する．そのため，各 ξ_k から，単調関数 $F_k(\psi_1, \ldots, \psi_m)$ を定義する．F_k は，エフェクトの組 $(\varphi_1, \ldots, \varphi_m)$ を受取り，$\xi_k[\varphi_1/\psi_1, \ldots, \varphi_m/\psi_m]$ を定義 3.22 で与えられるエフェクトの演算に従い計算した結果のエフェクトを返す関数である．これらの関数を使い以下のような不等式の集合を考える．

$$\begin{aligned}&\{\psi_1 \leq F_1(\psi_1, \ldots, \psi_m), \\ &\cdots, \\ &\psi_m \leq F_m(\psi_1, \ldots, \psi_m)\}\end{aligned}$$

この不等式集合を使い，エフェクトの組 $(\varphi_1^{(j)}, \cdots, \varphi_m^{(j)})$ $(j = 1, 2, \cdots)$ を次のように計算する．

$$\begin{aligned}\varphi_k^{(0)} &= \top \\ \varphi_k^{(j+1)} &= F_k(\varphi_1^{(j)}, \cdots, \varphi_m^{(j)})\end{aligned}$$

各 F_k の単調性より，$(\varphi_1^{(l)}, \cdots, \varphi_m^{(l)}) = (\varphi_1^{(l+1)}, \cdots, \varphi_m^{(l+1)})$ を満たす l が存在する．このとき，各エフェクトは $\psi_k = \varphi_k^{(l)}$ で決定される．

ここで決定したエフェクトを，型制約から取り出した usage 制約のエフェクト変数部分へ代入することで usage のみの制約が得られる．

$$\frac{[\![U]\!] \subseteq \Phi \qquad z \text{ fresh}}{\mathcal{E}_{\mathcal{D}}[\mathbf{new}^{\Phi}()] \xrightarrow{\epsilon} \mathcal{E}_{\mathcal{D}}[\mathbf{let_R}\ z : U\ \mathbf{in}\ z]}$$

$$\mathcal{E}_{\mathcal{D}}[\mathbf{let_R}\ x : U\ \mathbf{in}\ \mathbf{acc}^{l}(D)] \xrightarrow{x} \mathcal{E}_{\mathcal{D}}[\mathbf{acc}^{l}(\mathbf{let_R}\ x : U\ \mathbf{in}\ D)]$$

$$\frac{U_i \xrightarrow{l} U_i' \qquad U_k' = U_k \text{ for } k \neq i \qquad b = \mathbf{true} \text{ or } \mathbf{false}}{\mathcal{E}_{\mathcal{D}}[\mathbf{acc}^{l}(\mathbf{let_R}\ \tilde{x} : \tilde{U}\ \mathbf{in}\ x_i)] \xrightarrow{\epsilon} \mathcal{E}_{\mathcal{D}}[\mathbf{let_R}\ \tilde{x} : \tilde{U'}\ \mathbf{in}\ b]}$$

$$\frac{\nexists U.U_i \xrightarrow{l} U}{\mathcal{E}_{\mathcal{D}}[\mathbf{acc}^{l}(\mathbf{let_R}\ \tilde{x} : \tilde{U}\ \mathbf{in}\ x_i)] \xrightarrow{\epsilon} \mathbf{Error}}$$

$$\frac{U \leq U_1;_E U_2}{\mathcal{E}_{\mathcal{D}}[\mathbf{let_R}\ x : U\ \mathbf{in\ try}\ D_1\ \mathbf{with}\ D_2] \xrightarrow{x} \mathcal{E}_{\mathcal{D}}[\mathbf{try\ let_R}\ x : U_1\ \mathbf{in}\ D_1\ \mathbf{with\ let_R}\ x : U_2\ \mathbf{in}\ D_2]}$$

$$\mathcal{E}_{\mathcal{D}}[\mathbf{try\ let_R}\ \tilde{x} : \tilde{U}\ \mathbf{in}\ v\ \mathbf{with}\ D] \xrightarrow{\epsilon} \mathcal{E}_{\mathcal{D}}[\mathbf{let_R}\ \tilde{x} : \tilde{U}\ \mathbf{in}\ v]$$

$$\mathcal{E}_{\mathcal{D}}[\mathbf{try\ let_R}\ \tilde{x} : \tilde{U_1}\ \mathbf{in\ let_R}\ \tilde{y} : \tilde{U_2}\ \mathbf{in}\ \mathcal{E}_{D}^{\overline{try}}[\mathbf{raise}]\ \mathbf{with\ let_R}\ \tilde{x} : \tilde{U_3}\ \mathbf{in}\ M] \xrightarrow{\epsilon} \mathcal{E}_{\mathcal{D}}[\mathbf{let_R}\ \tilde{x} : \tilde{U_1};_E \tilde{U_3}\ \mathbf{in\ let_R}\ \tilde{y} : \tilde{U_2};_E \tilde{\mathbf{0}}\ \mathbf{in}\ M]$$

図 5: 動的な式の操作的意味 (一部)

その結果，(2-1) で生成される usage 制約は次の 2 つの形で表される．(F1) $\alpha_i \leq U_i \ \ (i = 1, \ldots, n)$ (F2) $\Phi_1 \supseteq \alpha_{i_1}, \ldots, \Phi_m \supseteq \alpha_{i_{n'}} \ \ (n' \leq n)$．(2-2) では，(F1) の形の制約を解く．つまり，usage 制約の集合 $\mathbf{UC} = \{\alpha_i \leq U_i \ (i = 1, \ldots, n)\}, \mathbf{US} = \emptyset$ から始めて，

$$(\{\alpha \leq U\} \cup \mathbf{UC}, \mathbf{US}) \rightarrow (\mathbf{UC}, \mathbf{US}[\mu\alpha.U/\alpha] \cup \{\alpha \leq U\})$$

を n 回適応することで，各 usage α_i を決定する．(F2) の制約を検査するアルゴリズムは，論文 [10] で述べられているように一般には決定不能であるが，Φ の表現を制限することで，完全ではなくても実用上有用なアルゴリズムを得られる可能性がある．この部分は今後の課題である．

例 5.2 例 2.5 の項 M に対する型推論は次のようになされる．ただし，型付け例 3.28 と同様に，M に対してエスケープ解析を行い，適当な注釈をつけた項を改めて M とおく．

通常の型の部分は通常の型推論と同様であるので，説明を単純にするために，usage 部分，エフェクト部分のみの推論を考える．

まず，(1) 型付け規則より usage 制約、エフェクト制約を取り出す．簡単に表記するため，部分式 N の中でリソース x, y での usage を α_N, β_N で表す．ただし，α_x は，$\mathbf{read}(x)$ の部分式である x の usage を表すものとする．また，部分式 N を型付けるエフェクトを ψ_N で表す．

最初に，$\mathbf{read}(x)$ について，以下の右端に記した各規則より次の usage 制約、エフェクト制約が得られる．

$$\begin{array}{lcll}
\alpha_x & \leq & \Diamond R & \text{(T-Var)} \\
\psi_x & \leq & \mathbf{0} & \text{(T-Var)} \\
\alpha_{\mathbf{read}(x)} & \leq & \alpha_x & \text{(T-Acc)} \\
\psi_{\mathbf{read}(x)} & \leq & \psi_x & \text{(T-Acc)} \\
\alpha_{\mathbf{read}(x)\{x\}} & \leq & \blacklozenge\alpha_{\mathbf{read}(x)} & \text{(T-Now)} \\
\psi_{\mathbf{read}(x)\{x\}} & \leq & \psi_{\mathbf{read}(x)} & \text{(T-Now)}
\end{array}$$

これを解くと，$\alpha_{\mathbf{read}(x)\{x\}} \leq \blacklozenge\Diamond R$，$\psi_{\mathbf{read}(x)\{x\}} \leq \mathbf{0}$ が得られる．同様に，$\beta_{\mathbf{write}(y)} \leq \blacklozenge\Diamond W$, $\alpha_{\mathbf{close}(x)} \leq \blacklozenge\Diamond C$, $\psi_{\mathbf{write}(y)\{y\}} \leq \mathbf{0}$, $\psi_{\mathbf{close}(y)\{y\}} \leq \mathbf{0}$ などが usage 制約，エフェクト制約として得られる．

同様にして，M の部分式について，usage 制約，エフェクト制約を取り出していく．

部分式 $M_2 \triangleq \mathbf{if\ read}(x)^{\{x\}}\ \mathbf{then\ true\ else\ raise}$ からは以下の usage 制約、エフェクト制約が取り出される．

$$\begin{array}{lcll}
\alpha_{M_2} & \leq & \alpha_{\mathbf{read}(x)\{x\}}; ((\psi_{\mathbf{true}})^{use} \& (\psi_{\mathbf{raise}})^{use}) & \text{(T-If)} \\
\psi_{M_2} & \leq & \psi_{\mathbf{read}(x)\{x\}}; (\psi_{\mathbf{true}} \& \psi_{\mathbf{raise}}) & \text{(T-If)}
\end{array}$$

$$\begin{array}{lcll}
\psi_{\mathbf{true}} & \leq & \mathbf{0} & \text{(T-Const)} \\
\psi_{\mathbf{raise}} & \leq & E & \text{(T-Raise)}
\end{array}$$

部分式 $M_1 \triangleq M_2; \mathbf{write}(y)^{\{y\}}; f\ \mathbf{true}$ からは次の制約が得られる．ただし，γ_f は，関数 f の usage を表し，ψ_{f^*} は，

関数 f の latent エフェクトを表す.

$$\begin{array}{lcll} \psi_f & \leq & \mathbf{0} & \text{(T-Fun)} \\ \gamma_f & \leq & 1 & \text{(T-App)} \\ \psi_{(f\mathbf{true})} & \leq & \psi_f; \psi_{\mathbf{true}}; \psi_{f^*} & \text{(T-App)} \end{array}$$

$$\begin{array}{lcl} \alpha_{M_1} & \leq & \alpha_{M_2}; (\psi_{\mathbf{write}(y)^{\{y\}}})^{use}; (\psi_{(f\mathbf{true})})^{use} \\ \psi_{M_1} & \leq & \psi_{M_2}; \psi_{\mathbf{write}(y)^{\{y\}}}; \psi_{(f\mathbf{true})} \end{array} \quad \text{(T-Let)}$$

部分式 $\mathbf{fun}(f, z, M_1)$ からは次の制約が得られる．ただし，$\gamma_{\mathbf{fun}}$ は，関数 $\mathbf{fun}(f, z, M_1)$ の usage とし，$\psi^*_{\mathbf{fun}}$ は関数 $\mathbf{fun}(f, x, M_1)$ の latent エフェクトとする.

$$\begin{array}{lcl} \alpha_{\mathbf{fun}(f,z,M_1)} & \leq & \gamma_{\mathbf{fun}} \odot \mu\alpha.(1 \otimes (\gamma_f \odot \alpha)) \otimes \Diamond(\alpha_{M_1} \setminus E) \\ \beta_{\mathbf{fun}(f,z,M_1)} & \leq & \gamma_{\mathbf{fun}} \odot \mu\alpha.(1 \otimes (\gamma_f \odot \alpha)) \otimes \Diamond(\beta_{M_1} \setminus E) \end{array} \quad \text{(T-Fun)}$$

$$\begin{array}{lcll} \psi_{\mathbf{fun}(f,z,M_1)} & \leq & \mathbf{0} & \text{(T-Fun)} \\ \psi_{f^*} & \leq & \psi_{M_1} & \text{(T-Fun)} \\ \psi_{\mathbf{fun}^*} & \leq & \psi_{f^*} & \text{(T-Fun)} \end{array}$$

部分式 $M_3 \triangleq \mathbf{fun}(f, z, M_1)\mathbf{true}$ からは次の制約が得られる.

$$\begin{array}{lcll} \alpha_{M_3} & \leq & \alpha_{\mathbf{fun}(f,z,M_1)}; (\psi_{\mathbf{true}})^{use}; (\psi_{\mathbf{fun}^*})^{use} & \text{(T-App)} \\ \beta_{M_3} & \leq & \beta_{\mathbf{fun}(f,z,M_1)}; (\psi_{\mathbf{true}})^{use}; (\psi_{\mathbf{fun}^*})^{use} & \text{(T-App)} \\ \gamma_{\mathbf{fun}} & \leq & 1 & \text{(T-App)} \\ \psi_{M_3} & \leq & \psi_{\mathbf{fun}(f,z,M_1)}; \psi_{\mathbf{true}}; \psi_{\mathbf{fun}^*} & \text{(T-App)} \end{array}$$

$M \triangleq \mathbf{try}\ \mathbf{fun}(f, z, M_1)\mathbf{true}\ \mathbf{with}\ \mathbf{close}(x)^{\{x\}}; \mathbf{close}(y)^{\{y\}}$ 部分からは，次の制約が生成される.

$$\begin{array}{lcl} \alpha_M & \leq & \alpha_{M_3};_E (\alpha_{\mathbf{close}(x)^{\{x\}}}; (\psi_{\mathbf{close}(y)^{\{y\}}})^{use}) \\ \beta_M & \leq & \beta_{M_3};_E ((\psi_{\mathbf{close}(x)^{\{x\}}})^{use}; \beta_{\mathbf{close}(y)^{\{y\}}}) \\ \psi_M & \leq & \psi_{M_3};_E (\psi_{\mathbf{close}(x)^{\{x\}}}; \psi_{\mathbf{close}(y)^{\{y\}}}) \end{array} \quad \text{(T-Try,T-Let)}$$

次に，(2) これらの制約を解く．まず，(2-1) 取り出したエフェクト制約を解き各部分項のエフェクトを決定する.

例えば，エフェクト制約

$$\begin{array}{lcl} \psi_{\mathbf{read}(x)^{\{x\}}} & \leq & \mathbf{0} \\ \psi_{\mathbf{write}(y)^{\{y\}}} & \leq & \mathbf{0} \\ \psi_f & \leq & \mathbf{0} \\ \psi_{\mathbf{true}} & \leq & \mathbf{0} \\ \psi_{\mathbf{raise}} & \leq & E \\ \psi_{M_2} & \leq & \psi_{\mathbf{read}(x)^{\{x\}}}; (\psi_{\mathbf{true}} \& \psi_{\mathbf{raise}}) \\ \psi_{(f\mathbf{true})} & \leq & \psi_f; \psi_{\mathbf{true}}; \psi_{f^*} \\ \psi_{M_1} & \leq & \psi_{M_2}; \psi_{\mathbf{write}(y)^{\{y\}}}; \psi_{(f\mathbf{true})} \\ \psi_{f^*} & \leq & \psi_{M_1} \\ \psi_{\mathbf{fun}^*} & \leq & \psi_{f^*} \end{array}$$

を解くことで,

$$\begin{array}{lcl} \psi_{M_2} & = & E^? \\ \psi_{(f\mathbf{true})} & = & E \\ \psi_{M_1} & = & E \\ \psi_{f^*} & = & E \\ \psi_{\mathbf{fun}^*} & = & E \end{array}$$

と各エフェクトを決定できる .

この決定したエフェクトを usage 制約のエフェクト変数部分に代入することで，usage の間の関係のみの制約が得られる．例えば，リソース x に関する usage 制約は次のように得られる.

$$\begin{array}{lcl} \gamma_f & \leq & 1 \\ \gamma_{\mathbf{fun}} & \leq & 1 \\ \alpha_{\mathbf{read}(x)^{\{x\}}} & \leq & \blacklozenge\Diamond R \\ \alpha_{\mathbf{close}(x)^{\{x\}}} & \leq & \blacklozenge\Diamond C \\ \alpha_x & \leq & \Diamond R \\ \alpha_{M_2} & \leq & \alpha_{\mathbf{read}(x)^{\{x\}}}; (\mathbf{0} \& E) \\ \alpha_{M_1} & \leq & \alpha_{M_2}; \mathbf{0}; (\mathbf{0} \& E) \\ \alpha_{\mathbf{fun}(f,z,M_1)} & \leq & \gamma_{\mathbf{fun}} \odot \mu\alpha.(1 \otimes (\gamma_f \odot \alpha)) \otimes \Diamond(\alpha_{M_1} \setminus E) \\ \alpha_{M_3} & \leq & \alpha_{\mathbf{fun}(f,z,M_1)}; \mathbf{0}; (\mathbf{0} \& E) \\ \alpha_M & \leq & \alpha_{M_3};_E (\alpha_{\mathbf{close}(x)^{\{x\}}}; \mathbf{0}) \end{array}$$

(2-2) この usage 制約を解くと，項 M におけるリソース x の usage が (よって型が) 推論できる．実際この制約を解くと,

$$\begin{array}{lcl} \alpha_M & \leq & (\mu\alpha.1 \otimes \alpha) \odot \Diamond((\blacklozenge\Diamond R; (\mathbf{0} \& E); (\mathbf{0} \& E)) \setminus E) \\ & & ; E;_E \blacklozenge\Diamond C \end{array}$$

が得られる．ここで，$\blacklozenge\Diamond R$，$(R; (\mathbf{0} \& E); (\mathbf{0} \& E)) \setminus E$ などを意味的に等しい R と略して書き整理すると以下が得られる.

$$\begin{array}{lcl} \alpha_M & \leq & ((\mu\alpha.1 \otimes \alpha) \odot \Diamond R); E;_E C \\ & = & ((\Diamond R \otimes \Diamond R \otimes \cdots); E);_E C \end{array}$$

これは，型付け例 3.28 で示した x の usage と同じである．このように我々の型システムは，例 2.5 のようなプログラムに対して十分な精度の型を機械的に推論することが可能である.

6 議論

本論文では，リソースに対するアクセスの順序の制御については従来の五十嵐と小林の型システム [10] にならって線形型の拡張である usage を用い，例外の発生についてはエフェクトを用いている．以下では，例外を扱うための他の手法について議論する.

五十嵐と小林の型システム [10] に基づいて例外処理機構を扱う最も単純な方法として，例外処理のためのプリミティブを関数にエンコードした上で五十嵐と小林の解析 [10] を適用することが考えられる．例外処理プリミティブをエンコードするためには，例外ハンドラを関数として表現し，関数のパラメタとして渡せばよい [1]．例えば，次のプログラムを考えよう．

$$\textbf{let } f = \lambda y.\textbf{raise in } (\textbf{try } f() \textbf{ with } \textbf{write}(x)); \textbf{close}(x)$$

これは，例外を発生する関数 f を定義し，$\textbf{write}(x)$ という例外ハンドラが定義されている中で関数 f を呼び出した後，$\textbf{close}(x)$ を実行するプログラムである．このプログラムは以下のような関数として表現できる．

$$\begin{array}{l}\textbf{let } f = \lambda y.\lambda h.\lambda c.h() \textbf{ in}\\ \qquad\qquad \textbf{let } c = \lambda z.\textbf{close}(x) \textbf{ in}\\ \qquad\qquad \textbf{let } h = \lambda z.(\textbf{write}(x); c()) \textbf{ in } f()(h)(c)\end{array}$$

例外ハンドラ $\textbf{write}(x)$ が関数 $\lambda z.(\textbf{write}(x); c)$ として表され，通常の継続 c とともに関数 f の引数として渡されていることに注意されたい．

しかしながら，上記のように例外処理プリミティブをエンコードして五十嵐と小林の解析 [10] を適用した場合，十分な解析精度を得ることはできない．例えば上のプログラムの場合，五十嵐と小林の解析を適用すると，x の usage は $\diamond C; \diamond W$ となり，W と C の操作のどちらが先に起こるかが判別できない．一方，我々の解析手法を用いると，x の usage は $(E;_E W); C$ と推論でき，W, C の順でアクセスが起こることがわかる．

例外処理プリミティブをもつ言語を扱うための別の方法として，usage を用いずに，リソースへのアクセス情報も含めてエフェクトを用いて解析するという方法も考えられる．例えば，$\lambda x.(\textbf{write}(x); \textbf{close}(x); \textbf{raise})$ の型は，$(\mathbf{R}, \rho) \xrightarrow{\rho^W;\rho^C;E} \textbf{bool}$ と表現できる．ここで ρ はリージョンと呼ばれるリソースの抽象化であり，エフェクト $\rho^W; \rho^C; E$ は，関数が呼ばれたときに ρ に属するリソースに W, C で表されるアクセスが起こったのち，例外が発生することを表す．このようにリソースへのアクセスもエフェクトとして表現すると，リソースへのアクセスと例外発生の順序関係を解析することは容易である．しかし，[12] で議論されているように，エフェクトに基づいた解析では異なるリソースが同じリージョンに抽象化される場合に，解析の精度が悪くなってしまう．

7 関連研究

リソースへのアクセス順序を解析する手法は，近年になって盛んに研究されているが [2, 3, 8, 10, 17, 18]，例外処理プリミティブを扱っているものは少ない．例外処理プリミティブを扱っているものとして，Bigriardi と Laneve[3]，岩間と小林 [11] による Java バイトコードのロックプリミティブの使用法の解析がある．しかしながら，それらの解析はメソッド単位の局所的な解析であり，かつ例外が発生する場所，ハンドラが静的に定まっているため，if 文の解析と本質的に変わらないため，エフェクトを用いる必要はない．

例外処理プリミティブを扱ってはいないが，扱うように拡張するのが比較的容易であると思われるものとして，エフェクトを用いた資源利用法解析 [5, 7, 14] がある．しかしながら，6 節で述べたようにエフェクトのみに基づく解析では十分な解析精度が得られない．また，エフェクトベースのシステムに線形論理を取り入れた研究 [13] もある．彼らの型システムではリソースへのアクセス関数の型を，資源状態の事前条件，事後条件で与えておき，その型情報を使い検証を行っている．その際，事前事後条件を一階の線形的な論理で記述することで，局所的な検証を可能 (分割検証可能）にしている．しかし，表現は異なっているがエフェクトに基づく解析と同様 6 節で述べた欠点がある．詳しい表現能力の比較については今後の課題である．

小林 [12] は，例外発生情報だけでなくリソースへのアクセス情報一般についてもエフェクトに含め，線形型に基づく解析とエフェクトによる解析を融合した型システムを提案している．しかし型システムが複雑であり，型推論の問題が未解決である．また，例外処理プリミティブの扱いについては簡単に議論しているのみで，定式化および正しさの証明は行っていない．

8 まとめ，今後の課題

例外処理機構の入った言語のための資源使用法解析のための型システムを提案した．従来の五十嵐と小林の型システムに新しい usage 構成子 E および $U_1;_E U_2$ を導入するとともに，例外発生のタイミングの解析のためにエフェクトを導入して拡張した．これにより，資源へのアクセス履歴のなかでの例外の発生場所が特定できるようになり，例外機構のもとでも精度のより解析が可能になる．

本解析手法に基づき実際のプログラミング言語のための資源使用法解析器を実装する上での今後の課題として，従来からの未解決部分である，推論された各リソースの usage U が仕様 Φ を満たしているか（$[\![U]\!] \subseteq \Phi$ が成り立つか）を

判定する（完全ではなくても）実用的なアルゴリズムの考案，ポインタなど他のプリミティブを扱うための拡張が挙げられる．

参考文献

[1] 住井 英二郎, 大根田 裕一, 米澤 明憲. 例外処理機構を備えた命令型言語の CPS 変換とその定式化. **情報処理学会論文誌：プログラミング**, 45(SIG12):67–82, 2004.

[2] Alexander Aiken, Manuel Fähndrich, and Raph Levien. Improving region-based analysis of higher-order languages. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 174–185, 1995.

[3] Gaetano Bigliardi and Cosimo Laneve. A type system for JVM threads. In *Proceedings of 3rd ACM SIGPLAN Workshop on Types in Compilation (TIC2000)*, Montreal, Canada, 2000.

[4] Bruno Blanchet. Escape analysis: Correctness, proof, implementation and experimental results. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages*, pages 25–37, 1998.

[5] Robert DeLine and Manuel Fähndrich. Adoption and focus: Practical linear types for imperative programming. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2002.

[6] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan Van Leeuwen, editor, *Handbook of Theoretical Computer Science Volume B*, chapter 6, pages 243–320. The MIT press/Elsevier, 1990.

[7] Jeffrey S. Foster, Tachio Terauchi, and Alex Aiken. Flow-sensitive type qualifiers. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2002.

[8] Stephen N. Freund and John C. Mitchell. The type system for object initialization in the Java bytecode language. *ACM Transactions on Programming Languages and Systems*, 21(6):1196–1250, 1999.

[9] John Hannan. A type-based analysis for stack allocation in functional languages. In *Proceedings of SAS'95*, volume 983 of *Lecture Notes in Computer Science*, pages 172–188, 1995.

[10] Atsushi Igarashi and Naoki Kobayashi. Resource usage analysis. *ACM Transactions on Programming Languages and Systems*, 27(2). To appear. *A preliminary summary appeared in Proceedings of POPL 2002, page 331-342.*

[11] Futoshi Iwama and Naoki Kobayashi. A new type system for JVM lock primitives. In *Proceedings of ASIA-PEPM'02*, pages 156–168. ACM Press, 2002.

[12] 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, 2003.

[13] Yizhak Mandelbaum, David Walker, and Robert Harper. An effective theory of type refinements. In *Proceedings of International Conference on Funcitonal Programming*, 2003.

[14] Christian Skalka and Scott Smith. History effects and verification. In *Asian Programming Languages Symposium, November 2004 , Lecture Notes in Computer Science 3302*. Springer, 2004.

[15] Jean-Pierre Talpin and Pierre Jouvelot. Polymorphic type, region and effect inference. *Journal of Functional Programming*, 2(3):245–271, 1992.

[16] Jean-Pierre Talpin and Pierre Jouvelot. The type and effect discipline. In *Proceedings of IEEE Symposium on Logic in Computer Science*, pages 162–173, 1992.

[17] Mads Tofte and Jean-Pierre Talpin. Region-based memory management. *Information and Computation*, 132(2):109–176, 1997.

[18] David Walker, Karl Crary, and J. Gregory Morrisett. Typed memory management via static capabilities. *ACM Transactions on Programming Languages and Systems*, 22(4):701–771, 2000.