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

住井 英二郎[†] 大根田 裕一[††] 米澤 明憲[††]

制御フローグラフに類似した一般的な命令型言語を、CPS (継続渡しスタイル) を中間表現としてコンパイルする方法を提案する。CPS とは、「残りの計算」(**継続**と呼ばれる) を表現する関数を生成することにより、すべての関数呼び出しや条件分岐を末尾位置においた関数型言語の一種であり、Scheme や ML といった関数型言語のコンパイラに広く用いられている。命令型言語を CPS に変換することで、スタックフレームや例外ハンドラが継続として明示化され、関数適用や例外処理における複雑な制御の流れが明確化される。結果として、インライン展開や末尾呼び出し最適化を含む多くの最適化が容易に実現できる。我々は命令型言語を CPS に変換する過程の単純な定式化を与え、その正しさを証明する。

CPS Conversion of Imperative Language with Exception Handling and its Formalization

Eijiro Sumii,[†] Yuichi Oneda[††] and Akinori Yonezawa[††]

We propose a method of compiling a generic imperative language (similar to control flow graphs) by using CPS (continuation passing style) as an intermediate representation. CPS is a form of functional programs that puts all function calls and all conditional branches into tail positions by generating functions to represent "the rest of the computation" (called *continuations*). It is widely used as an intermediate representation in compilers of functional languages such as Scheme and ML. By translating an imperative language into CPS, complicated control flow as in function application and exception handling is clarified because activation records and exception handlers are made explicit as continuations. As a result, many optimizations including inline expansion and tail-call optimization can be easily implemented. We give a simple formulation to the process of converting the imperative language into CPS and prove its correctness.

1. 序 論

コンパイラとは、高水準言語のプログラムをより低水準の言語に翻訳するプログラムである。一般にコンパイラの出力言語は入力言語からかけ離れていることが多く、また、どちらの言語もそのままでは効率的なコードを生成するための解析・最適化に適さないことが多いので、通常のコンパイラでは入力言語と出力言語以外に**中間表現**を使用する。そのような中間表現として、命令型言語では制御フローグラフや静的単一代入 (SSA) 形式[2),6),16)] が、関数型言語では継続渡しスタイル (CPS)[3),11),17)] や A 正規形[7)] が広く用いられてきた。しかし命令型言語と関数型言語のコンパイラおよびそれらの中間表現は、本質的な類似点が多いにも関わらず、外見やコミュニティの相違からほとんど独立に研究されており、しばしば一方の知見を他方に応用する機会を損失している。

この溝を埋める試みの一つとして、本稿では CPS を中間表現として命令型言語をコンパイルする方法を提案する。命令型言語を CPS に変換することにより、関数呼び出しや関数からの復帰、例外処理といった多様な制御の流れを明示的かつ統一的に表現できる。結果として、インライン展開、フロー解析、末尾呼び出し最適化といった解析や最適化を、関数の呼び出し・復帰だけでなく例外処理にも (ad hoc な方法によらず一様に) 適用することが可能になる。

命令型言語を CPS に変換する方法自体は古くから知られている[18)]。本稿の技術的貢献は、変換元言語に例外処理を追加し、拡張された変換の正しさを示した点にある。2.3.3 節で議論するように、例外処理のある

† ペンシルバニア大学コンピュータ情報科学科
Department of Computer and Information Science, University of Pennsylvania
†† 東京大学情報理工学系研究科コンピュータ科学専攻
Department of Computer Science, Graduate School of Information Science and Technology, University of Tokyo

命令型言語の CPS 変換には特有の問題があり、我々の拡張は自明でない。

以降の構成は次の通りである。2 節では変換の基本となるアイディアについて、いくつかの例題により説明する。3 節では変換の入力となる、制御フローグラフを模倣した単純な命令型言語の抽象構文と操作的意味論を定義する。4 節では、変換の出力となる我々の CPS の抽象構文と操作的意味論を定義する。5 節では、実際の変換を定義し、操作的意味論に基づく正しさを示す。6 節では関連研究について議論する。最後に 7 節では、より広範な視点から本稿の提案について考察し、今後の課題を議論する。

2. アイディア

2.1 関数呼び出しや例外処理のない場合

例として次のような命令型言語のプログラムを考える。これは 10 個の整数 a[0], ..., a[9] の積を求めるプログラムである。

```
     int a[], r, i;
L0:  r = 1;
     i = 0;
L1:  if i = 10 then return r;
L2:  r = r * a[i];
     i = i + 1;
     goto L1;
```

このプログラムを以下のように変換する。まず、一つの基本ブロックにつき一つの関数を定義する。(基本ブロックとは、入口が先頭だけ、出口が末尾だけしかないブロックのことである[1)]。) それらの関数は、基本ブロックの入口における変数の値を受け取り、出口における変数の値を、次の基本ブロックを表す関数に渡す。これにより、破壊的代入を関数的な束縛に変換することが可能となる。

```
f0(a, r, i) =
  let r = 1 in
  let i = 0 in
  f1(a, r, i)
f1(a, r, i) =
  if i = 10 then r else
  f2(a, r, i)
f2(a, r, i) =
  let r = r * a[i] in
  let i = i + 1 in
  f1(a, r, i)
```

関数 f0, f1, f2 が基本ブロック L0, L1, L2 に対応する。このように変換されたプログラムでは、すべての関数呼び出しや条件分岐が末尾位置にある。というのは、そもそも基本ブロックの定義より、すべてのジャンプや条件分岐は末尾位置にあるためである。

2.2 関数呼び出しのある場合

しかし、元々のプログラムに末尾でない関数呼び出しがあったら、前述の変換では除去できない。そこで、通常の関数型言語における CPS 変換と同様に、「以降の計算」を表現する関数である**継続**を導入する。たとえば、先の例において掛け算を別の関数に分けたプログラム

```
int main(int a[]) {
    int r, i;
    r = 1;
    i = 0;
L1: if i = 10 then return r;
L2: r = mul(r, a[i]);
    i = i + 1;
    goto L1;
}
int mul(int x, int y) {
  return x * y;
}
```

は、以下のように変換できる。

```
main(k, a) =
  let r = 1 in
  let i = 0 in
  f1(k, a, r, i)
f1(k, a, r, i) =
  if i = 10 then k(r) else
  f2(k, a, r, i)
f2(k, a, r, i) =
  let k'(r) =
    let i = i + 1 in
    f1(k, a, r, i) in
  mul(k', r, a[i])
mul(k, x, y) = k(x * y)
```

関数 f2 の内部で生成されている関数 k' が、関数 mul を呼び出した後の継続を表している。それぞれの関数 main, f1, f2, mul は継続 k を引数として受け取り、f1 における k(r) や mul における k(x * y) のように、return 文で値を返すときに継続を適用する。

なお、一般に継続を表現する関数 (この場合は k') には自由変数 (この場合は k, a, i) があるので、コードへのポインタと自由変数の値との組すなわち**関数クロージャ**として表現し、スタックに確保する実装が普通である。したがって、継続の生成 (この場合は let

k'(r) = ...）は生きている変数の値と戻り番地を待避することに相当し、継続の適用（この場合は k(x * y)）は生きている変数の値を復元して戻り番地へジャンプすることに相当する。CPS では継続の適用も含め、すべての関数呼び出しが末尾位置にあるので、ジャンプのように扱えることを思い出されたい。

2.3 例外処理のある場合

次に、例外処理のあるプログラムの例を考える。ただし、簡単のために例外の内容は省略し、例外の発生は raise、捕獲は try ... catch ...という構文を使用する。

2.3.1 関数内の例外処理

次のプログラムは、10 個の整数 a[0], ..., a[9] の積を求めるが、その途中で 0 に遭遇したら計算を中断し、直ちに 0 を返すという例である。

```
int main(int a[]) {
  int r, i;
  try {
      r = 1;
      i = 0;
  L1: if i = 10 then return r;
  L2: if a[i] = 0 then raise;
  L3: r = r * a[i];
      i = i + 1;
      goto L1;
  } catch {
  L4: return 0;
  }
}
```

このプログラムでは、ブロック L2 の raise 文で発生する例外が、catch 節のブロック L4 により処理されることが明白である。一般にプログラムの文面において、raise 文が同一の関数の try 節の内部にあれば、その例外はどのブロックによって処理されるか静的にわかる。そのような場合には、raise 文は goto 文と同様に変換することができる。たとえば上のプログラムは次のように変換される。

```
main(k, a) =
  let r = 1 in
  let i = 0 in
  f1(k, a, r, i)
f1(k, a, r, i) =
  if i = 10 then k(r) else
  f2(k, a, r, i)
f2(k, a, r, i) =
  if a[i] = 0 then f4(k, a, r, i) else
  f3(k, a, r, i)
f3(k, a, r, i) =
  let r = r * a[i] in
  let i = i + 1 in
  f1(k, a, r, i)
f4(k, a, r, i) = k(0)
```

ブロック L2 の raise 文は、ブロック L4 への goto 文と同等であるから、関数 f2 における関数 f4 の適用 f4(k, a, r, i) に変換されている。

2.3.2 関数間の例外処理

例外処理の特徴は、その**動的スコープ**にある。たとえば、次のプログラムを考える。

```
int main(int a[]) {
  int r, i;
  try {
      r = 1;
      i = 0;
  L1: if i = 10 then return r;
  L2: r = mul(r, a[i]);
      i = i + 1;
      goto L1;
  } catch {
  L3: return 0;
  }
}
int mul(int x, int y) {
    if y = 0 then raise;
L4: return x * y;
}
```

このプログラムは先のプログラムと同じ働きをするが、関数 mul で発生する例外が、mul を呼び出しているブロック L2 を囲む try 文の catch 節により捕獲される。

このように例外が関数の外部にエスケープする場合は、どの raise 文がどの catch 節を呼び出すか、一般には静的にわからない。したがって、実行時に「現時点で有効な外部の例外ハンドラ」h を保持・管理しておき、外部にエスケープする例外が発生したらその例外ハンドラを呼び出す、という処理が必要となる。例外を発生するときは、もし try 節の内部であれば対応する catch 節へ飛び、そうでなければ外部の例外ハンドラ h を呼び出す。関数を呼び出すときは、もし try 節の内部であれば対応する catch 節を呼び出す関数を新たな h として与え、そうでなければそれまでと同じ h を与える。

たとえば、上のプログラムは次のように変換するこ

とができる。

```
main(k, h, a) =
  let r = 1 in
  let i = 0 in
  f1(k, h, a, r, i)
f1(k, h, a, r, i) =
  if i = 10 then k(r) else
  f2(k, h, a, r, i)
f2(k, h, a, r, i) =
  let k'(r) =
    let i = i + 1 in
    f1(k, h, a, r, i)
  and h'() = f3(k, h, a, r, i) in
  mul(k', h', r, a[i])
f3(k, h, a, r, i) = k(0)
mul(k, h, x, y) =
  if y = 0 then h() else
  f4(k, h, x, y)
f4(k, h, x, y) = k(x * y)
```

それぞれの基本ブロック main, f1, f2, f3, mul, f4 は、通常の継続 k 以外に外部の例外ハンドラ h を引数にとる。ブロック L2 から関数 mul を呼び出すときは、内部の例外ハンドラ f3 を呼び出す関数 h' を生成して渡している。また、関数 mul の raise 文は、外部の例外ハンドラ h の適用に変換されている。

なお、上の例で継続 k' と例外ハンドラ h' は自由変数 k, h, a, i を共有するが、これを素朴に実装すると、k' と h' のクロージャに k, h, a, i の値が重複して待避されてしまう。このような冗長な格納は、クロージャの共有[3] により省略することができる。一般に関数を適用するときに生成される継続と例外ハンドラには共通の自由変数があるので、クロージャを共有する実装が有効である。本稿では、二個の関数を同時に定義する構文 let $k(r) = \ldots$ and $h() = \ldots$ in ... により、これを表現する。

2.3.3 関数内と関数間の例外処理を区別する理由

例外が同一の関数の内部で捕獲される場合と、外部にエスケープする場合との区別は、効率のためだけではなく、変数への破壊的代入を正しく扱うために一般に必要である。たとえば

```
int main() {
  int x;
  x = 0;
  try {
  L1: x = 1;
      raise;
  } catch {
  L2: return x;
  }
}
```

というプログラムを考える。もしこのように内部で捕獲される例外も外部にエスケープする例外と同様に（素朴に）変換してしまうと、

```
main(k, h) =
  let x = 0 in
  let h'() = f2(k, h, x) in
  f1(k, h', x)
f1(k, h, x) =
  let x = 1 in
  h()
f2(k, h, x) = k x
```

のように、f1 を呼び出す際に例外ハンドラ h' を生成することになるだろう。しかし、この変換結果は関数 main の継続 k に 1 ではなく 0 を返すので誤っている。h' が定義されたときの x の値は 0 だが、例外が発生する（すなわち h' が適用される）ときの x の値は 1 に変わっているからである。上のプログラムは、正しくは

```
main(k, h) =
  let x = 0 in
  f1(k, h, x)
f1(k, h, x) =
  let x = 1 in
  f2(k, h, x)
f2(k, h, x) = k x
```

と変換されるべきである。

以上のような注意は、変数への破壊的代入と例外処理の両方がある言語を CPS に変換する場合のみ必要となる、独特の問題である。ML にも参照セルへの破壊的代入 :=はあるが、変数はセルの内容ではなく参照自体に束縛されるので、変数の値が変わることはない。Scheme には変数への破壊的代入 set! があるが、CPS に変換する場合は set! の対象となる変数を解析し、やはりセルへの参照に束縛している[11]。命令型言語では多数の変数が破壊的代入の対象となるので、そのような実装は効率の観点から採用しがたい。

なお、(1) 例外が同一の関数の内部で捕獲されるかどうか、(2) されるとしたらどのブロックによって処理されるか、の二点は静的にわかるので（2.3.1 節参照）、上述のような区別をしても実行時の悪影響はない。むしろ、例外ハンドラを表現するクロージャの生成・適用が減少するので、一般に効率は向上すると期

待できる。

2.3.4 CPS への変換による例外処理の最適化

最後に、例外処理のある命令型言語を CPS に変換することで可能となる最適化の例を挙げておく。

もっとも明白な最適化は、例外ハンドラのインライン展開である。たとえば、2.3.1 節の関数 `f4` や 2.3.2 節の関数 `f3` をインライン展開すれば、それぞれに対応する `catch` 節をインライン展開した場合と同様の結果になる。このように CPS への変換により、例外処理について特別な配慮をすることなく、通常の関数と同一のインライン展開が自明に適用できる。

さらに、通常では面倒な `goto` 文のインライン展開や、例外処理の末尾呼び出し最適化等も自然に実現できる。たとえば、次のようなプログラムを考える。

```
int main(int x) {
  goto L1;
  try {
  L1: return sub(x);
  } catch {
  L2: raise;
  }
}
int sub(int x) {
    if x = 0 then raise;
L3: return x;
}
```

これを上述のアイディアにしたがって CPS に変換すると、以下のようになる。

```
main(k, h, x) = f1(k, h, x)
f1(k, h, x) =
  let k'(r) = k(r)
  and h'() = f2(k, h, x) in
  sub(k', h', x)
f2(k, h, x) = h()
sub(k, h, x) =
  if x = 0 then h() else k(x)
```

このプログラムにおいて、まず `f1`, `f2`, `sub` をインライン展開すると

```
main(k, h, x) =
  let k'(r) = k(r)
  and h'() = h() in
  if x = 0 then h'() else k'(x)
```

となる。さらに `k'` と `h'` をインライン展開すると

```
main(k, h, x) =
  if x = 0 then h() else k(x)
```

となる。これは変換元言語で考えると

```
int main(int x) {
  if x = 0 then raise;
  return x;
}
```

に当たるが、同様の最適化を変換元言語や SSA だけで実現することは容易でない。

3. 変換元言語

我々の変換の入力となる命令型言語の抽象構文を図 1 に示す。プログラム P は関数宣言 D の集合である。関数宣言 D は、宣言する関数の名前 f、その引数の名前 $x_1, \ldots, x_l$、ローカル変数の名前 $y_1, \ldots, y_m$、入口の基本ブロック B_0、それ以外の基本ブロック $B_1, \ldots, B_n$ から成る。ただし、可算無限個の関数名の集合 $\{f, g, h, \ldots\}$ ならびに可算無限個の変数名の集合 $\{x, y, z, \ldots\}$ が存在し、それらの交わりは空であるとする。また、本稿では簡便のために、$X_1, \ldots, X_n$ という形の列を一般に $\overline{X}$ と略記する。関数の入口以外の基本ブロック B_i には、ラベル L_i と例外ハンドラ H_i が付記される。H は、例外ハンドラがないことを表現する $\perp$ か、あるいは例外ハンドラとなる基本ブロックのラベル L である。2 節の言語に対応させると、$L(\perp) : B$ は `L: B` に、$L(L') : B$ は `L: try B catch goto L'` に相当する。基本ブロック B は、通常の制御フローグラフの基本ブロックに、例外を発生する `raise` 文を追加したものである。一般に例外が関数からエスケープするかどうか、しないならばどのブロックによって処理されるかはプログラムの文面から静的にわかるので、通常の例外処理のある命令型言語のプログラムは、上述の言語で表現することができる。ただし、本稿では例外の内容は省略する（これを追加する方法については 7 節で議論する）。

この言語の操作的意味論は、図 2 の規則による**実行状態** S の書き換えで定義される。実行状態 S とは、正常終了状態 $\langle P, \texttt{halt}\ i\rangle$ か、例外終了状態 $\langle P, \texttt{abort}\rangle$ か、8 つ組 $\langle P, f, H, B, \sigma, r, F, E\rangle$ である。P は実行しているプログラム、f は実行している関数の名前、H は内部の例外ハンドラ、B はこれから実行する基本ブロック、σ は**ストア**すなわち変数の名前から値への部分写像、r は関数の返値を代入する caller の変数、F は関数呼び出しの履歴を表現するデータ構造、E は外部の例外ハンドラを表現するデータ構造である。

図 2 において、最初の 3 つの規則は、変数の参照と代入の意味を定義している。次の 2 つの規則は、関数呼び出しの意味を定義している。内部の例外ハンドラが存在する場合は、例外ハンドラスタックに待避して

$$
\begin{array}{llll}
\text{プログラム} & P & ::= & \{D_1, \ldots, D_n\} \\
\text{関数宣言} & D & ::= & f(\overline{x})\{\texttt{var}\ \overline{y}; B_0; L_1(H_1) : B_1; \ldots; L_n(H_n) : B_n\} \\
\text{例外ハンドラ} & H & ::= & L \\
 & & | & \bot \\
\text{基本ブロック} & B & ::= & x := i; B \\
 & & | & x := y; B \\
 & & | & x := y - z; B \\
 & & | & x := f(\overline{y}); B \\
 & & | & \texttt{goto}\ L \\
 & & | & \texttt{return}\ x \\
 & & | & \texttt{if}\ x \leq y\ \texttt{then}\ L_1\ \texttt{else}\ L_2 \\
 & & | & \texttt{raise}
\end{array}
$$

図 1 変換元言語の抽象構文
Fig. 1 Abstract Syntax of Source Language

$$
\begin{array}{llll}
\text{関数呼び出しスタック} & F & ::= & () \\
 & & | & (f, L, B, \sigma, r, F, E) \\
\text{例外ハンドラスタック} & E & ::= & () \\
 & & | & (f, L, \sigma, r, F, E)
\end{array}
$$

$$
\begin{array}{lcl}
\langle P, f, H, (x := i; B), \sigma, r, F, E\rangle & \to & \langle P, f, H, B, \sigma[x := i], r, F, E\rangle \\
\langle P, f, H, (x := y; B), \sigma, r, F, E\rangle & \to & \langle P, f, H, B, \sigma[x := \sigma(y)], r, F, E\rangle \\
\langle P, f, H, (x := y - z; B), \sigma, r, F, E\rangle & \to & \langle P, f, H, B, \sigma[x := \sigma(y) - \sigma(z)], r, F, E\rangle \\
\langle P, f, \bot, (x := g(\overline{y}); B), \sigma, r, F, E\rangle & \to & \langle P, g, \bot, B_0, [\overline{z} := \sigma(\overline{y}), \overline{x} := 0], \\
 & & \quad x, (f, \bot, B, \sigma, r, F, E), E\rangle \\
 & & \qquad \text{if } g(\overline{z})\{\texttt{var}\ \overline{x}; B_0; \ldots\} \in P \\
\langle P, f, L, (x := g(\overline{y}); B), \sigma, r, F, E\rangle & \to & \langle P, g, \bot, B_0, [\overline{z} := \sigma(\overline{y}), \overline{x} := 0], \\
 & & \quad x, (f, L, B, \sigma, r, F, E), (f, L, \sigma, r, F, E)\rangle \\
 & & \qquad \text{if } g(\overline{z})\{\texttt{var}\ \overline{x}; B_0; \ldots\} \in P \\
\langle P, f, H, \texttt{goto}\ L, \sigma, r, F, E\rangle & \to & \langle P, f, H', B, \sigma, r, F, E\rangle \\
 & & \qquad \text{if } f(\overline{x})\{\ldots; L(H') : B; \ldots\} \in P \\
\langle P, f, H, \texttt{return}\ x, \sigma, r, (g, H', B, \sigma', r', F, E), E'\rangle & \to & \langle P, g, H', B, \sigma'[r := \sigma(x)], r', F, E\rangle \\
\langle P, f, H, \texttt{return}\ x, \sigma, r, (), E\rangle & \to & \langle P, \texttt{halt}\ \sigma(x)\rangle \\
\langle P, f, H, \texttt{if}\ x \leq y\ \texttt{then}\ L_1\ \texttt{else}\ L_2, \sigma, r, F, E\rangle & \to & \langle P, f, H_1, B_1, \sigma, r, F, E\rangle \\
 & & \quad \text{if } \sigma(x) \leq \sigma(y) \text{ and } f(\overline{x})\{\ldots; L_1(H_1) : B_1; \ldots\} \in P \\
\langle P, f, H, \texttt{if}\ x \leq y\ \texttt{then}\ L_1\ \texttt{else}\ L_2, \sigma, r, F, E\rangle & \to & \langle P, f, H_2, B_2, \sigma, r, F, E\rangle \\
 & & \quad \text{if } \sigma(x) > \sigma(y) \text{ and } f(\overline{x})\{\ldots; L_2(H_2) : B_2; \ldots\} \in P \\
\langle P, f, L, \texttt{raise}, \sigma, r, F, E\rangle & \to & \langle P, f, H, B, \sigma, r, F, E\rangle \\
 & & \qquad \text{if } f(\overline{x})\{\ldots; L(H) : B; \ldots\} \in P \\
\langle P, f, \bot, \texttt{raise}, \sigma, r, F, (g, L, \sigma', r', F', E)\rangle & \to & \langle P, g, H, B, \sigma', r', F', E\rangle \\
 & & \qquad \text{if } g(\overline{x})\{\ldots; L(H) : B; \ldots\} \in P \\
\langle P, f, \bot, \texttt{raise}, \sigma, r, F, ()\rangle & \to & \langle P, \texttt{abort}\rangle
\end{array}
$$

図 2 変換元言語の操作的意味論
Fig. 2 Operational Semantics of Source Language

いる。次の規則は goto 文の意味を、その次の 2 つの規則は return 文の意味を、その次の 2 つの規則は if 文の意味を定義している。最後の 3 つの規則は raise 文の意味を定義している。内部の例外ハンドラが存在する場合はそこへ飛び、しない場合は外部の例外ハンドラを呼び出している。

議論を単純にするために、以降では**適格** (well-formed) なプログラムだけを考慮する。適格とは、直観としては関数名、変数名、ラベルの宣言に不足や重複がないことである。厳密には、図 3 の述語 $wf(P)$ を満足するプログラム P を適格と定義する。なお、図 3 ではプログラム P の適格性 $wf(P)$ を定義するために、関数宣言 D の適格性 $wf(\{\overline{f}\}, D)$、例外ハンドラ H の適格性 $wf(\{\overline{L}\}, H)$、基本ブロック B の適格性 $wf(\{\overline{f}\}, \{\overline{z}\}, \{\overline{L}\}, B)$ も同時に定義している。ただし、$\{\overline{f}\}$, $\{\overline{L}\}$, $\{\overline{z}\}$ はその時点で定義されている関数、ラベル、変数の集合であり、D, H, B の適格性を判定するために参照される。

さらに、図 4 のように適格性の定義を実行状態に拡張すると、以下の定理が成立する。なお、図 4 では実行状態 S の適格性 $wf(S)$ を定義するために、関数呼び出しスタック F の適格性 $wf(P, r, F)$ と例外ハンドラスタック E の適格性 $wf(P, E)$ も同時に定義している。

定理 1 S が適格であり、正常終了状態でも例外終了状態でもなければ、$S \to S'$ なる S' が一意に存在し、S' も適格である。

証明 $S = \langle P, f, H, B, \sigma, r, F, E\rangle$ において B の構文により場合わけする。S が適格であるという仮定と図 4 の定義を使用して、図 2 の規則が一意に適用できること、さらに S' が適格であることを確認する。□

4. 変換先言語

我々の変換の出力となる CPS 関数型言語の抽象構文は、図 5 の通りである。プログラム p は関数宣言 d の集合であり、関数宣言は関数名 f と引数名 $\overline{x}$ と式 e の組である。3 節と同様に、関数名と引数名の集合は独立とする。説明を明確にするために、式 e においては

- トップレベルで宣言され複数の引数をとる、自由変数のない通常の関数の適用 $f(\overline{x})$
- クロージャで表現され単一の引数をとる、自由変数があるかもしれない関数の適用 $x(y)$
- クロージャで表現され引数をとらない、自由変数があるかもしれない関数の適用 $x()$

のそれぞれを区別する。また、2 つの関数 x, z を同時に定義する構文 $\texttt{let}\ x = \lambda y.\, e_1\ \texttt{and}\ z = \lambda_.\, e_2\ \texttt{in}\ e_3$ を使用する。これは $\texttt{let}\ x = \lambda y.\, e_1\ \texttt{in let}\ z = \lambda_.\, e_2\ \texttt{in}\ e_3$ と（変数 x が式 e_2 に出現しなければ）意味としては等価であるが、実装としては 2 節で指摘したように継続と例外ハンドラでクロージャを共有することを想定している☆。

この言語の操作的意味論は、図 6 の規則による実行状態 $s = \langle p, e, \rho\rangle$ の書き換えで定義される。p は実行しているプログラム、e はこれから評価する式、ρ は現在の環境すなわち変数の名前から値 v への部分写像である。値 v は、整数 i か単一の引数 x をとるクロージャ $cls(x, e, \rho)$ または引数をとらないクロージャ $cls(_, e, \rho)$ である。規則自体は通常の関数型言語の操作的意味論と同様であり、特に変わった点はない。

5. 変換とその正しさ

3 節の言語から 4 節の言語への変換を、図 7 のように定義する。基本となるアイディアは 2 節の通りで、(1) 一つの基本ブロックにつき一つの関数を与え、継続、外部の例外ハンドラ、ならびにローカル変数の値を渡す、(2) 関数呼び出しや例外の発生については、内部の例外ハンドラが存在するかどうかにより場合わけする、の 2 点である。ただし、$f.L$ は変換元言語の関数名 f とラベル L から一意に生成される、変換先言語の関数名とする。

変換の正しさは、図 8 のように定義される、変換元言語の実行状態 S と変換先言語の実行状態 s との対応 $s = \mathcal{T}(S)$ により証明される。ただし、図 8 において $\mathcal{T}(P, k, r, F)$ は関数呼び出しスタック F の変換、$\mathcal{T}(P, h, E)$ は例外ハンドラスタック E の変換である。また、$halt$ と $abort$ は正常終了と例外終了を表現するための、予約された特殊な変数または関数（どちらでも以降の議論に影響はない）の名前とする。

定理 2 S が適格かつ $S \to S'$ ならば、$\mathcal{T}(S) \to^* \mathcal{T}(S')$。

証明 付録参照。□

系 3 $\texttt{main}(\overline{x})\{\texttt{var}\ \overline{y}; B; \ldots\} \in P$ かつ P は適格とする。$S = \langle P, \texttt{main}, \bot, B, [\overline{x} := \overline{i}; \overline{y} := 0], r, (), ()\rangle$ とすると、次のいずれか一つ（かつ一つのみ）が成り立つ。

☆ また、この構文により 5 節の証明が簡潔になるという効用もある。継続 k と例外ハンドラ h を生成するときに、もし単純に $\texttt{let}\ k = \lambda r.\, e_1\ \texttt{in let}\ h = \lambda_.\, e_2\ \texttt{in}\ e_3$ とすると、h のクロージャにおける e_2 の環境に k の束縛が混入してしまい、変換前言語と変換先言語の実行状態の対応が複雑になってしまうためである。

$$
\begin{array}{lcl}
wf(\{\overline{f}(\ldots)\{\ldots\}\}) & = & \wedge_{1\leq i\leq n} wf(\{\overline{f}\}, f_i(\ldots)\{\ldots\}) \wedge \overline{f} \text{ distinct} \\
\\
wf(\{\overline{f}\}, f(\overline{x})\{\texttt{var } \overline{y}; B_0; \overline{L}(\overline{H}) : \overline{B}\}) & = & \wedge_{0\leq i\leq n} wf(\{\overline{f}\}, \{\overline{x},\overline{y}\}, \{\overline{L}\}, B_i) \wedge f \in \{\overline{f}\} \wedge \\
 & & \wedge_{1\leq j\leq n} wf(\{\overline{L}\}, H_j) \wedge \overline{L} \text{ distinct} \wedge \overline{x},\overline{y} \text{ distinct} \\
\\
wf(\{\overline{L}\}, L) & = & L \in \{\overline{L}\} \\
wf(\{\overline{L}\}, \bot) & = & \mathit{true} \\
\\
wf(\{\overline{f}\}, \{\overline{z}\}, \{\overline{L}\}, (x := i; B)) & = & x \in \{\overline{z}\} \wedge wf(\{\overline{f}\}, \{\overline{z}\}, \{\overline{L}\}, B) \\
wf(\{\overline{f}\}, \{\overline{z}\}, \{\overline{L}\}, (x := y; B)) & = & \{x,y\} \subseteq \{\overline{z}\} \wedge wf(\{\overline{f}\}, \{\overline{z}\}, \{\overline{L}\}, B) \\
wf(\{\overline{f}\}, \{\overline{z}\}, \{\overline{L}\}, (x := y - z; B)) & = & \{x,y,z\} \subseteq \{\overline{z}\} \wedge wf(\{\overline{f}\}, \{\overline{z}\}, \{\overline{L}\}, B) \\
wf(\{\overline{f}\}, \{\overline{z}\}, \{\overline{L}\}, (x := f(\overline{y}); B)) & = & f \in \{\overline{f}\} \wedge \{x,\overline{y}\} \subseteq \{\overline{z}\} \wedge wf(\{\overline{f}\}, \{\overline{z}\}, \{\overline{L}\}, B) \\
wf(\{\overline{f}\}, \{\overline{z}\}, \{\overline{L}\}, \texttt{goto } L) & = & L \in \{\overline{L}\} \\
wf(\{\overline{f}\}, \{\overline{z}\}, \{\overline{L}\}, \texttt{return } x) & = & x \in \{\overline{z}\} \\
wf(\{\overline{f}\}, \{\overline{z}\}, \{\overline{L}\}, \texttt{if } x \leq y \texttt{ then } L'_1 \texttt{ else } L'_2) & = & \{x,y\} \subseteq \{\overline{z}\} \wedge \{L'_1, L'_2\} \subseteq \{\overline{L}\} \\
wf(\{\overline{f}\}, \{\overline{z}\}, \{\overline{L}\}, \texttt{raise}) & = & \mathit{true}
\end{array}
$$

図 3 適格なプログラム
Fig. 3 Well-Formed Programs

$$
\begin{array}{lcl}
wf(\langle P, f, H, B, \sigma, r, F, E\rangle) & = & wf(P) \wedge f(\overline{x})\{\texttt{var } \overline{y}; B_0; L_1(H_1) : B_1; \ldots; L_n(H_n) : B_n\} \in P \wedge \\
 & & wf(\{\overline{L}\}, H) \wedge wf(\mathit{dom}(P), \{\overline{x},\overline{y}\}, \{\overline{L}\}, B) \wedge \mathit{dom}(\sigma) = \{\overline{x},\overline{y}\} \wedge \\
 & & wf(P, r, F) \wedge wf(P, E) \\
wf(\langle P, \texttt{halt } i\rangle) & = & wf(P) \\
wf(\langle P, \texttt{abort}\rangle) & = & wf(P) \\
\\
wf(P, r, ()) & = & \mathit{true} \\
wf(P, r, (f, H, B, \sigma, r', F, E)) & = & f(\overline{x})\{\texttt{var } \overline{y}; B_0; L_1(H_1) : B_1; \ldots; L_n(H_n) : B_n\} \in P \wedge \\
 & & wf(\{\overline{L}\}, H) \wedge wf(\mathit{dom}(P), \{\overline{x},\overline{y}\}, \{\overline{L}\}, B) \wedge r \in \mathit{dom}(\sigma) = \{\overline{x},\overline{y}\} \wedge \\
 & & wf(P, r', F) \wedge wf(P, E) \\
\\
wf(P, ()) & = & \mathit{true} \\
wf(P, (f, L, \sigma, r, F, E)) & = & f(\overline{x})\{\texttt{var } \overline{y}; B_0; L_1(H_1) : B_1; \ldots; L_n(H_n) : B_n\} \in P \wedge \\
 & & L \in \{\overline{L}\} \wedge \mathit{dom}(\sigma) = \{\overline{x},\overline{y}\} \wedge wf(P, r, F) \wedge wf(P, E)
\end{array}
$$

図 4 適格な実行状態
Fig. 4 Well-Formed Execution States

- $S \rightarrow^* \langle P, \texttt{halt } i\rangle \not\rightarrow$ かつ $\mathcal{T}(S) \rightarrow^* \langle P, \mathit{halt}(r), [r := i]\rangle \not\rightarrow$
- $S \rightarrow^* \langle P, \texttt{abort}\rangle \not\rightarrow$ かつ $\mathcal{T}(S) \rightarrow^* \langle P, \mathit{abort}(), \emptyset\rangle \not\rightarrow$
- $S \rightarrow \ldots \rightarrow \ldots$ なる無限の書き換え列が存在し、かつ、$\mathcal{T}(S) \rightarrow \ldots \rightarrow \ldots$ なる無限の書き換え列が存在する

証明 定理 1 と定理 2 による。□

6. 関 連 研 究

継続 (continuation) の概念は、元々は goto 文をもつ命令型言語の表示的意味論を定義するために発見された[18]。したがって命令型言語をコンパイルするために CPS を使用することは、むしろ自然な発想ともいえる。しかし実際の歴史では、Guy Steele が MIT の修士研究で開発した Scheme コンパイラ RABBIT[17]、

$$
\begin{array}{llll}
\text{プログラム} & p & ::= & \{d_1, \ldots, d_n\} \\
\text{関数宣言} & d & ::= & f(\overline{x}) = e \\
\text{式} & e & ::= & \texttt{let}\ x = i\ \texttt{in}\ e \\
 & & | & \texttt{let}\ x = y\ \texttt{in}\ e \\
 & & | & \texttt{let}\ x = y - z\ \texttt{in}\ e \\
 & & | & f(\overline{x}) \\
 & & | & x(y) \\
 & & | & x() \\
 & & | & \texttt{if}\ x \leq y\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \\
 & & | & \texttt{let}\ x = \lambda y.\, e_1\ \texttt{in}\ e_2 \\
 & & | & \texttt{let}\ x = \lambda y.\, e_1\ \texttt{and}\ z = \lambda_.\, e_2\ \texttt{in}\ e_3
\end{array}
$$

図 5 変換先言語の抽象構文
Fig. 5 Abstract Syntax of Target Language

$$
\begin{array}{lcll}
\langle p, \texttt{let}\ x = i\ \texttt{in}\ e, \rho\rangle & \to & \langle p, e, \rho[x := i]\rangle & \\
\langle p, \texttt{let}\ x = y\ \texttt{in}\ e, \rho\rangle & \to & \langle p, e, \rho[x := \rho(y)]\rangle & \\
\langle p, \texttt{let}\ x = y - z\ \texttt{in}\ e, \rho\rangle & \to & \langle p, e, \rho[x := \rho(y) - \rho(z)]\rangle & \\
\langle p, f(\overline{x}), \rho\rangle & \to & \langle p, e, [\overline{y} := \rho(\overline{x})]\rangle & \text{if } (f(\overline{y}) = e) \in p \\
\langle p, x(y), \rho\rangle & \to & \langle p, e, \rho'[z := \rho(y)]\rangle & \text{if } \rho(x) = cls(z, e, \rho') \\
\langle p, x(), \rho\rangle & \to & \langle p, e, \rho'\rangle & \text{if } \rho(x) = cls(_, e, \rho') \\
\langle p, \texttt{if}\ x \leq y\ \texttt{then}\ e_1\ \texttt{else}\ e_2, \rho\rangle & \to & \langle p, e_1, \rho\rangle & \text{if } \rho(x) \leq \rho(y) \\
\langle p, \texttt{if}\ x \leq y\ \texttt{then}\ e_1\ \texttt{else}\ e_2, \rho\rangle & \to & \langle p, e_2, \rho\rangle & \text{if } \rho(x) > \rho(y) \\
\langle p, \texttt{let}\ x = \lambda y.\, e_1\ \texttt{in}\ e_2, \rho\rangle & \to & \langle p, e_2, \rho[x := cls(y, e_1, \rho)]\rangle & \\
\langle p, \texttt{let}\ x = \lambda y.\, e_1\ \texttt{and}\ z = \lambda_.\, e_2\ \texttt{in}\ e_2, \rho\rangle & \to & \langle p, e_3, \rho[x := cls(y, e_1, \rho), z := cls(_, e_2, \rho)]\rangle &
\end{array}
$$

図 6 変換先言語の操作的意味論
Fig. 6 Operational Semantics of Target Language

その後継である ORBIT[11]、さらに Appel らの Standard ML of New Jersey[3] と、CPS を中間表現とするコンパイラは関数型言語を対象に発展した。

CPS と SSA はどちらも変数が単一代入であり、`if` 文や関数呼び出しあるいは `goto` 文が関数または基本ブロックの末尾位置にあることから、以前より類似が指摘されている[3),5)]。Appel[4] は、直接支配の関係にもとづいて SSA を CPS に変換する方法を提案している。Kelsey[9] は、annotation のついた（継続以外の高階関数がない、一階の）CPS と SSA との相互の変換を定義している。これらと本稿の相違としては、次の点が挙げられる。(1) 命令型言語から SSA を経由せず、CPS へ直接の変換を定義している。(2) 例外処理を通常の CPS に変換している☆。(3) 変換の正しさが示されている。

関数型言語に由来する中間表現としては、CPS 以外に A 正規形[7] が広く知られている。文献[7] は、CPS で可能な簡約は変換元言語における特殊な簡約（A 正規化）によって実現できることを証明し、CPS 変換は不要であると議論している。しかし、変換元言語に一級継続や例外処理があると、そのような簡約は容易でない。（たとえば最近の文献[8] では、部分継続のある関数型言語の健全かつ完全な等式公理を定義しているが、一方向かつ合流性のある正規化の定義には成功していない。）CPS 変換をすれば通常の $\beta\eta$ 簡約のみ

☆ もちろん、Standard ML of New Jersey も例外処理をサポートしているが、Appel の CPS[3] には例外処理のための特殊なプリミティブ `gethdlr`, `sethdlr` があり、通常の関数や継続と同一の解析・最適化が一様に適用できるとはいえない。実際に、文献[3] では `gethdlr`, `sethdlr` について特殊な最適化を提案しているが（75 頁）、本稿の方法では通常の最適化により実現できる。また、Appel の変換元言語は変数への破壊的代入のない関数型言語 (ML) だが、我々の変換元言語は命令型言語であり、特有の配慮が必要となる（2.3.3 節参照）。

$$
\begin{array}{lcl}
\mathcal{T}(\{D_1,\ldots,D_n\}) & = & \mathcal{T}(D_1)\cup\ldots\cup\mathcal{T}(D_n) \\
\\
\mathcal{T}(f(\overline{x})\{\texttt{var}\ \overline{y}; B_0; L_1(H_1):B_1;\ldots L_n(H_n):B_n\}) & = & \{f(k,h,\overline{x})\ = \\
 & & \quad \texttt{let}\ y_1=0\ \texttt{in} \\
 & & \quad \ldots \\
 & & \quad \texttt{let}\ y_m=0\ \texttt{in} \\
 & & \quad \mathcal{T}(f,k,h,(\overline{x},\overline{y}),\bot,B_0), \\
 & & f.L_1(k,h,\overline{x},\overline{y})=\mathcal{T}(f,k,h,(\overline{x},\overline{y}),H_1,B_1), \\
 & & \ldots \\
 & & f.L_n(k,h,\overline{x},\overline{y})=\mathcal{T}(f,k,h,(\overline{x},\overline{y}),H_n,B_n)\} \\
 & & \qquad\qquad k,\ h \text{ fresh} \\
\\
\mathcal{T}(f,k,h,V,H,(x:=i;B)) & = & \texttt{let}\ x=i\ \texttt{in}\ \mathcal{T}(f,k,h,V,H,B) \\
\mathcal{T}(f,k,h,V,H,(x:=y;B)) & = & \texttt{let}\ x=y\ \texttt{in}\ \mathcal{T}(f,k,h,V,H,B) \\
\mathcal{T}(f,k,h,V,H,(x:=y-z;B)) & = & \texttt{let}\ x=y-z\ \texttt{in}\ \mathcal{T}(f,k,h,V,H,B) \\
\mathcal{T}(f,k,h,V,\bot,(x:=g(\overline{y});B)) & = & \texttt{let}\ k'=\lambda x.\,\mathcal{T}(f,k,h,V,\bot,B)\ \texttt{in} \\
 & & g(k',h,\overline{y}) \qquad\qquad k' \text{ fresh} \\
\mathcal{T}(f,k,h,(\overline{z}),L,(x:=g(\overline{y});B)) & = & \texttt{let}\ k'=\lambda x.\,\mathcal{T}(f,k,h,V,\bot,B) \\
 & & \texttt{and}\ h'=\lambda_.\,f.L(k,h,\overline{z})\ \texttt{in} \\
 & & g(k',h',\overline{y}) \qquad\qquad k',\ h' \text{ fresh} \\
\mathcal{T}(f,k,h,(\overline{z}),H,\texttt{goto}\ L) & = & f.L(k,h,\overline{z}) \\
\mathcal{T}(f,k,h,V,H,\texttt{return}\ x) & = & k(x) \\
\mathcal{T}(f,k,h,(\overline{z}),H,\texttt{if}\ x\le y\ \texttt{then}\ L_1\ \texttt{else}\ L_2) & = & \texttt{if}\ x\le y\ \texttt{then}\ f.L_1(k,h,\overline{z})\ \texttt{else}\ f.L_2(k,h,\overline{z}) \\
\mathcal{T}(f,k,h,V,\bot,\texttt{raise}) & = & h() \\
\mathcal{T}(f,k,h,(\overline{z}),L,\texttt{raise}) & = & f.L(k,h,\overline{z})
\end{array}
$$

図 7 プログラムの変換
Fig. 7 Translation of Programs

$$
\begin{array}{lcl}
\mathcal{T}(\langle P,f,H,B,\sigma,r,F,E\rangle) & = & \langle\mathcal{T}(P),\mathcal{T}(f,k,h,(\overline{x},\overline{y}),H,B),\sigma\uplus\mathcal{T}(P,k,r,F)\uplus\mathcal{T}(P,h,E)\rangle \\
 & & \qquad \text{if } f(\overline{x})\{\texttt{var}\ \overline{y};\ldots\}\in P \text{ and } k,\ h \text{ fresh} \\
\mathcal{T}(\langle P,\texttt{halt}\ i\rangle) & = & \langle P,halt(r),[r:=i]\rangle \qquad\qquad r \text{ fresh} \\
\mathcal{T}(\langle P,\texttt{abort}\rangle) & = & \langle P,abort(),\emptyset\rangle \\
\\
\mathcal{T}(P,k,r,()) & = & [k:=cls(r,halt(r),\emptyset)] \\
\mathcal{T}(P,k,r,(f,H,B,\sigma,r',F,E)) & = & [k:=cls(r,\mathcal{T}(f,k',h,(\overline{x},\overline{y}),H,B),\sigma\uplus\mathcal{T}(P,k',r',F)\uplus\mathcal{T}(P,h,E))] \\
 & & \qquad \text{if } f(\overline{x})\{\texttt{var}\ \overline{y};\ldots\}\in P \text{ and } k',\ h \text{ fresh} \\
\\
\mathcal{T}(P,h,()) & = & [h:=cls(_,abort(),\emptyset)] \\
\mathcal{T}(P,h,(f,L,\sigma,r,F,E)) & = & [h:=cls(_,f.L(k,h',\overline{x},\overline{y}),\sigma\uplus\mathcal{T}(P,k,r,F)\uplus\mathcal{T}(P,h',E))] \\
 & & \qquad \text{if } f(\overline{x})\{\texttt{var}\ \overline{y};\ldots\}\in P \text{ and } k,\ h' \text{ fresh}
\end{array}
$$

図 8 実行状態の変換
Fig. 8 Translation of Execution States

で、様々な制御構造の簡約を自然に実現できる。

一般に例外処理は様々な実装が可能である。本稿の方法はその一つに過ぎないが、例外処理も通常の関数や継続により表現されるので、同一の解析・最適化が適用できるというメリットがある。例外処理を実装する様々な方法については、中間言語 C--についての文献[15] が参考になるだろう。この文献では CPS による例外処理は詳細な比較・検討の対象から除外されているが、結果として生成されるコードの観点からは、本稿の方法は stack cutting に相当する。これは C 言語でいう setjmp と longjmp による実装や、Objective Caml[13] における実装と同様である。setjmp と longjmp が継続の取得と適用に相当することを考慮すれば、これは自然な対応である。また、本稿の方法における例外ハンドラ `h`（2.3.2 節参照）は、Objective Caml の実装における `trapsp`[12] に相当する。これらの実装は、`try` 文にややオーバーヘッドがかかるものの、`raise` 文のオーバーヘッドが小さく、例外の発生がある程度以上に頻繁な場合は有利である[14),15)]。ML や Java のような現代のプログラミング言語において、そのようなプログラムは少なくない[14)]。

7. 考察と今後の課題

本稿では例外処理の制御フローのみを議論し、例外の内容を省略してきたが、これは以下のように追加することが可能である。まず、変換元言語の `raise` 文を `raise` $C(\overline{x})$、例外ハンドラとなるブロックを $L(H) : B$ から $L(H) : y \Rightarrow B$ のように拡張する。ただし、C は可算無限個の例外コンストラクタの集合の要素である。そして、`raise` 文の変換においては、() ではなく $C(\overline{x})$ に内部または外部の例外ハンドラを適用する。例外を処理するブロックでは、例外の内容 $C(\overline{x})$ を変数 y に束縛し、パターンマッチングによる場合わけなどの必要な処理をする。

5 節の変換では、継続や例外ハンドラも含め、スコープにあるすべての変数の値をすべての基本ブロックに渡すようにしている。これは無駄な場合がある。たとえば、2.3.1 節の関数 `f4` の変数 `a`, `r`, `i` や 2.3.2 節の関数 `f3` の変数 `h`, `a`, `r`, `i` は不要である。このような変数は型システムを応用した静的解析[10] により、プログラムのサイズ n についてほぼ線形時間 $O(n\alpha(n,n))$ で、ある意味で最適に除去することができる。（ただし、α は逆アッカーマン関数であり、4 以下とみなしてよい[19]。）直観としては、不要な変数は特殊な型システムにおいて型 `unit` を付与することができ、union-find アルゴリズム[19] による単一化にもとづく型推論で検出できるためである。これは Appel の SSA から CPS への変換[4] における、不要な変数を除去するための解析を包摂する。すなわち、Appel の変換で除去できる変数は、我々の変換においては小林の解析で除去できる。

一般に命令型言語の実装では、関数や例外ハンドラのフレームはスタックに確保するのが普通である。CPS の実装においても、（変換元言語に `callcc` がなければ）すべての継続や例外ハンドラを、ごみ収集が必要なヒープではなくスタックに確保することが可能である[3)]。したがって、本稿の方法がメモリ管理の観点から通常のコンパイラより不利になることはない。

6 節で述べたように、本稿の例外処理の方法は通常の実装でいう stack cutting に相当する。文献[14),15)] では stack cutting は callee save と共存できないとされているが、CPS では callee save を模倣することが可能なので[3)]、本稿の提案により stack cutting と callee save の共存が可能になることも期待される。たとえば、関数 `f1` が別の関数 `f2` を呼び出し、`f2` がさらに別の関数 `f3` を呼び出し、`f3` で発生した例外を `f1` が捕獲するというプログラムを考える。`f1` はある生きている変数の値を callee save に選び、`f2` の仮引数 `r` に当たる実引数として渡すとする。さらに、`f2` は `r` の値を callee save に選び、`f3` の仮引数 `r'` に当たる実引数として渡すとする。すると、`f3` は外部の例外ハンドラ `h` を `r'` に適用するだけで、結果として `f2` を経由せず `f1` の `catch` 節を正しく呼び出すことができる。これは、`f2` に渡される callee save の値を、`f3` の呼び出しでも callee save に選んだためである。そうでない場合は stack unwinding[15] と同様に、`f3` で発生した例外を `f2` がまず捕獲し、外部の例外ハンドラを `r` にふたたび適用する（ように変換する）必要がある。この場合わけは `f2` の変換においてのみ必要であり、`f1` や `f3` の変換には影響しないことに注意されたい。

一般に SSA では様々な効率のよい解析が可能であるが[2),6),16)]、CPS では 1 個の基本ブロックが 1 個の関数として表現されるので、同様の解析は関数間解析を必要とし SSA より困難であるとされる[9)]。しかし、CPS への変換において適切な annotation をつければ、構文が表現する情報は等価であるから、原理としては同一の処理が可能である。たとえば、継続の適用を通常の関数適用と区別し、関数呼び出しではなく制御フローグラフの辺とみなせば、通常の関数内解析が可能である。実際に文献[9)] では SSA から CPS への変換だけでなく、CPS から SSA への逆変換も定義

し、SSA と（annotation をつけた一階の）CPS が等価であることを証明している。一方で、たとえばインライン展開のように通常の関数と継続などを区別する必要がない場合は annotation を無視すればよいだけなので、CPS のほうが SSA よりも一様な処理が可能である。

もちろん、実際のコンパイラを開発して十分な規模の実験をしなければ、一般に CPS が命令型言語の中間表現としてどれぐらい有用か断定はできない。本稿の考察から、理屈としては従来より簡潔で（例外処理を stack cutting で実装する場合と比較して）同等以上の性能のコンパイラを構築できると期待されるが、その実証は今後の課題である。

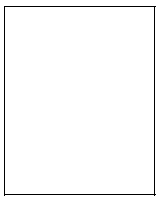
謝辞 本稿の研究と草稿について貴重なコメントをくださった田浦健次朗氏と大山恵弘氏に感謝します。

参 考 文 献

1) Aho, A. V., Sethi, R. and Ullman, J. D.: *Compilers: Principles, Techniques and Tools*, Addison-Wesley (1985).
2) Alpern, B., Wegman, M. N. and Zadeck, F. K.: Detecting equality of variables in programs, *Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 1–11 (1988).
3) Appel, A. W.: *Compiling with Continuations*, Cambridge University Press (1992).
4) Appel, A. W.: *Modern Compiler Implementation in ML*, Cambridge University Press (1998).
5) Appel, A. W.: SSA is Functional Programming, *ACM SIGPLAN Notices*, Vol. 33, No. 4, pp. 17–20 (1998).
6) Cytron, R., Ferrante, J., Rosen, B. K., Wegman, M. N. and Zadeck, F. K.: Efficiently computing static single assignment form and the control dependence graph, *ACM Transactions on Programming Languages and Systems*, Vol. 13, No. 4, pp. 451–490 (1991). Extended abstract appeared in *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1989, pages 25–35.
7) Flanagan, C., Sabry, A., Duba, B. F. and Felleisen, M.: The Essence of Compiling with Continuations, *Proceedings of the ACM SIGPLAN '93 Conference on Programming Language Design and Implementation*, pp. 237–247 (1993). In *ACM SIGPLAN Notices*, 28(6), June 1993.
8) Kameyama, Y. and Hasegawa, M.: A sound and complete axiomatization of delimited continuations, *Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming*, pp. 177–188 (2003).
9) Kelsey, R. A.: A correspondence between continuation passing style and static single assignment form, *Papers from the 1995 ACM SIGPLAN Workshop on Intermediate Representations*, pp. 13–22 (1995).
10) Kobayashi, N.: Type-Based Useless-Variable Elimination, *Higher-Order and Symbolic Computation*, Vol. 14, No. 2–3, pp. 221–260 (2001). Extended abstract appeared in *Proceedings of the 2000 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation*.
11) Kranz, D., Kelsey, R., Rees, J., Hudak, P., Philbin, J. and Adams, N.: ORBIT: an optimizing compiler for Scheme, *Symposium on Compiler Construction*, pp. 219–233 (1986).
12) Le Botlan, D. and Schmitt, A.: The OCaml System – Implementation, `http://pauillac.inria.fr/~lebotlan/docaml_html/english/` (2001).
13) Leroy, X.: Objective Caml, `http://caml.inria.fr/`.
14) Ogasawara, T., Komatsu, H. and Nakatani, T.: A Study of Exception Handling and Its Dynamic Optimization in Java, *Proceedings of the 16th ACM SIGPLAN conference on Object Oriented Programming, Systems, Languages, and Applications*, pp. 83–95 (2001).
15) Ramsey, N. and Peyton Jones, S.: A single intermediate language that supports multiple implementations of exceptions, *Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation*, pp. 285–298 (2000).
16) Rosen, B. K., Wegman, M. N. and Zadeck, F. K.: Global value numbers and redundant computations, *Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 12–27 (1988).
17) Steele, G. L.: RABBIT: A compiler for SCHEME, Technical Report TR474, Artificial Intelligence Laboratory, Massachusetts Institute of Technology (1978).
18) Strachey, C. and Wadsworth, C. P.: Continuations: A mathematical semantics for Handling Full Jumps, Technical Report PRG-11, Programming Research Group Technical Monograph, Oxford University Computing Laboratory (1974). Reprinted in *Higher-Order and Symbolic Computation*, Vol. 13, No. 1–2, Pages 135–152, 2000.
19) Tarjan, R. E.: Data structures and network al-

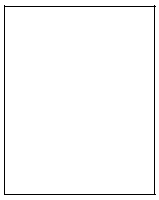
gorithms, *CBMS-NSF Regional Conference Series in Applied Mathematics*, p. 131 (1983).

(平成 16 年 2 月 23 日受付)
(平成 16 年 5 月 10 日採録)



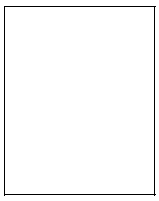
住井英二郎

1975 年生。1998 年 3 月東京大学理学部情報科学科卒業。2000 年 3 月同大学院理学系研究科情報科学専攻修士課程修了。2000 年 4 月より 2001 年 3 月まで同博士課程学生、日本学術振興会特別研究員、およびペンシルバニア大学コンピュータ情報科学科 Visiting Scholar。2001 年 4 月より 2003 年 3 月まで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助手。2003 年 4 月よりペンシルバニア大学コンピュータ情報科学科 Research Associate。情報セキュリティ、プロセス計算、部分評価等の領域における、プログラミング言語および型システムの理論と応用について研究。ACM 会員。



大根田裕一

昭和 55 年 7 月生。平成 15 年 3 月東京大学理学部情報科学科卒業。同年 4 月同大学院情報理工学系研究科コンピュータ科学専攻入学。プログラミング言語、特にアスペクト指向言語に関する研究に従事。



米澤 明憲（正会員）

1947 年 6 月に生まれ、MIT 計算機科学科博士課程修了 (Ph.D. in Computer Science), MIT 計算機科学研究所および人工知能研究所において並列・分散計算モデルの研究に従事し、「並列オブジェクト」概念のパイオニアの一人として知られている。帰国後、東工大を経て、1988 年に東京大学理学部情報科学科教授に着任した。米国計算機学会終身フェロー (ACM Fellow) であり、ACM TOPLAS 副編集長、日本ソフトウエア科学会理事長、ドイツ国立情報科学技術研究所 (GMD) 科学顧問、政府情報科学技術委員会委員などを歴任。

付録　定理 2 の証明

図 2 のどの規則を使用して $S \to S'$ を導出したかにより場合わけする。

$S = \langle P, f, H, (x := i; B), \sigma, r, F, E\rangle \to \langle P, f, H, B, \sigma[x := i], r, F, E\rangle = S'$ **の場合**　S は適格であるから、$f(\overline{x})\{\texttt{var}\ \overline{y}; \ldots\} \in P$ が一意に存在して、

$$
\begin{aligned}
&\mathcal{T}(S)\\
&= \langle \mathcal{T}(P), \mathcal{T}(f, k, h, (\overline{x}, \overline{y}), H, (x := i; B)), \sigma \uplus \mathcal{T}(P, k, r, F) \uplus \mathcal{T}(P, h, E)\rangle\\
&= \langle \mathcal{T}(P), \texttt{let}\ x = i\ \texttt{in}\ \mathcal{T}(f, k, h, (\overline{x}, \overline{y}), H, B), \sigma \uplus \mathcal{T}(P, k, r, F) \uplus \mathcal{T}(P, h, E)\rangle\\
&\to \langle \mathcal{T}(P), \mathcal{T}(f, k, h, (\overline{x}, \overline{y}), H, B), \sigma[x := i] \uplus \mathcal{T}(P, k, r, F) \uplus \mathcal{T}(P, h, E)\rangle\\
&= \mathcal{T}(S')
\end{aligned}
$$

$S = \langle P, f, H, (x := y; B), \sigma, r, F, E\rangle \to \langle P, f, H, B, \sigma[x := \sigma(y)], r, F, E\rangle = S'$ **の場合**　S は適格であるから、$f(\overline{x})\{\texttt{var}\ \overline{y}; \ldots\} \in P$ が一意に存在して、

$$
\begin{aligned}
&\mathcal{T}(S)\\
&= \langle \mathcal{T}(P), \mathcal{T}(f, k, h, (\overline{x}, \overline{y}), H, (x := y; B)), \sigma \uplus \mathcal{T}(P, k, r, F) \uplus \mathcal{T}(P, h, E)\rangle\\
&= \langle \mathcal{T}(P), \texttt{let}\ x = y\ \texttt{in}\ \mathcal{T}(f, k, h, (\overline{x}, \overline{y}), H, B), \sigma \uplus \mathcal{T}(P, k, r, F) \uplus \mathcal{T}(P, h, E)\rangle\\
&\to \langle \mathcal{T}(P), \mathcal{T}(f, k, h, (\overline{x}, \overline{y}), H, B), \sigma[x := \sigma(y)] \uplus \mathcal{T}(P, k, r, F) \uplus \mathcal{T}(P, h, E)\rangle\\
&= \mathcal{T}(S')
\end{aligned}
$$

$S = \langle P, f, H, (x := y - z; B), \sigma, r, F, E\rangle \to \langle P, f, H, B, \sigma[x := \sigma(y) - \sigma(z)], r, F, E\rangle = S'$ **の場合**　S は適格であるから、$f(\overline{x})\{\texttt{var}\ \overline{y}; \ldots\} \in P$ が一意に存在して、

$$
\begin{aligned}
&\mathcal{T}(S)\\
&= \langle \mathcal{T}(P), \mathcal{T}(f, k, h, (\overline{x}, \overline{y}), H, (x := y - z; B)), \sigma \uplus \mathcal{T}(P, k, r, F) \uplus \mathcal{T}(P, h, E)\rangle\\
&= \langle \mathcal{T}(P), \texttt{let}\ x = y - z\ \texttt{in}\ \mathcal{T}(f, k, h, (\overline{x}, \overline{y}), H, B), \sigma \uplus \mathcal{T}(P, k, r, F) \uplus \mathcal{T}(P, h, E)\rangle\\
&\to \langle \mathcal{T}(P), \mathcal{T}(f, k, h, (\overline{x}, \overline{y}), H, B), \sigma[x := \sigma(y) - \sigma(z)] \uplus \mathcal{T}(P, k, r, F) \uplus \mathcal{T}(P, h, E)\rangle\\
&= \mathcal{T}(S')
\end{aligned}
$$

$S = \langle P, f, \bot, (x := g(\overline{y}); B), \sigma, r, F, E\rangle \to \langle P, g, \bot, B_0, [\overline{z} := \sigma(\overline{y}), \overline{x} := 0], x, (f, \bot, B, \sigma, r, F, E), E\rangle = S'$ **かつ** $g(\overline{z})\{\texttt{var}\ \overline{x}; B_0; \ldots\} \in P$ **の場合**　S は適格であるから、$f(\overline{x}')\{\texttt{var}\ \overline{y}'; \ldots\} \in P$ が一意に存在して、

$$
\begin{aligned}
& \mathcal{T}(S) \\
= \ & \langle \mathcal{T}(P), \mathcal{T}(f,k,h,(\overline{x}',\overline{y}'),\bot,(x := g(\overline{y});B)), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)\rangle \\
= \ & \langle \mathcal{T}(P), \texttt{let } k' = \lambda x.\, \mathcal{T}(f,k,h,(\overline{x}',\overline{y}'),\bot,B) \texttt{ in } g(k',h,\overline{y}), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)\rangle \\
\to \ & \langle \mathcal{T}(P), g(k',h,\overline{y}), \\
& \quad \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E) \uplus \\
& \quad [k' := cls(x, \mathcal{T}(f,k,h,(\overline{x}',\overline{y}'),\bot,B), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E))]\rangle \\
\to \ & \langle \mathcal{T}(P), \texttt{let } x_1 = 0 \texttt{ in } \ldots \texttt{ let } x_m = 0 \texttt{ in } \mathcal{T}(g,k'',h',(\overline{z},\overline{x}),\bot,B_0), \\
& \quad [k'' := cls(x, \mathcal{T}(f,k,h,(\overline{x}',\overline{y}'),\bot,B), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)), \\
& \quad\ h' := \mathcal{T}(P,h,E)(h), \overline{z} := \sigma(\overline{y})]\rangle \\
\to^* \ & \langle \mathcal{T}(P), \mathcal{T}(g,k'',h',(\overline{z},\overline{x}),\bot,B_0), \\
& \quad [k'' := cls(x, \mathcal{T}(f,k,h,(\overline{x}',\overline{y}'),\bot,B), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)), \\
& \quad\ h' := \mathcal{T}(P,h,E)(h), \overline{z} := \sigma(\overline{y}), \overline{x} := 0]\rangle \\
= \ & \langle \mathcal{T}(P), \mathcal{T}(g,k'',h',(\overline{z},\overline{x}),\bot,B_0), \\
& \quad [k'' := cls(x, \mathcal{T}(f,k,h,(\overline{x}',\overline{y}'),\bot,B), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E))] \uplus \\
& \quad \mathcal{T}(P,h',E) \uplus [\overline{z} := \sigma(\overline{y}), \overline{x} := 0]\rangle \\
= \ & \langle \mathcal{T}(P), \mathcal{T}(g,k'',h',(\overline{z},\overline{x}),\bot,B_0), \\
& \quad [\overline{z} := \sigma(\overline{y}), \overline{x} := 0] \uplus \\
& \quad [k'' := cls(x, \mathcal{T}(f,k,h,(\overline{x}',\overline{y}'),\bot,B), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E))] \uplus \\
& \quad \mathcal{T}(P,h',E)\rangle \\
= \ & \langle \mathcal{T}(P), \mathcal{T}(g,k'',h',(\overline{z},\overline{x}),\bot,B_0), \\
& \quad [\overline{z} := \sigma(\overline{y}), \overline{x} := 0] \uplus \mathcal{T}(P,k'',x,(f,\bot,B,\sigma,r,F,E)) \uplus \mathcal{T}(P,h',E)\rangle \\
= \ & \mathcal{T}(S')
\end{aligned}
$$

$S = \langle P, f, L, (x := g(\overline{y}); B), \sigma, r, F, E\rangle \to \langle P, g, \bot, B_0, [\overline{z} := \sigma(\overline{y}), \overline{x} := 0], x, (f, L, B, \sigma, r, F, E), (f, L, \sigma, r, F, E)\rangle = S'$ かつ $g(\overline{z})\{\texttt{var } \overline{x}; B_0; \ldots\} \in P$ の場合　S は適格であるから、$f(\overline{x}')\{\texttt{var } \overline{y}'; \ldots\} \in P$ が一意に存在して、

$$
\begin{aligned}
& \mathcal{T}(S) \\
= \ & \langle \mathcal{T}(P), \mathcal{T}(f,k,h,(\overline{x}',\overline{y}'),L,(x := g(\overline{y});B)), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)\rangle \\
= \ & \langle \mathcal{T}(P), \texttt{let } k' = \lambda x.\, \mathcal{T}(f,k,h,(\overline{x}',\overline{y}'),\bot,B) \texttt{ and } h' = \lambda_.\, f.L(k,h,\overline{x}',\overline{y}') \texttt{ in } g(k',h',\overline{y}), \\
& \quad \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)\rangle \\
\to \ & \langle \mathcal{T}(P), g(k',h',\overline{y}), \\
& \quad \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E) \uplus \\
& \quad [k' := cls(x, \mathcal{T}(f,k,h,(\overline{x}',\overline{y}'),\bot,B), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E))] \uplus \\
& \quad [h' := cls(_, f.L(k,h,\overline{x}',\overline{y}'), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E))]\rangle \\
\to \ & \langle \mathcal{T}(P), \texttt{let } x_1 = 0 \texttt{ in } \ldots \texttt{ let } x_m = 0 \texttt{ in } \mathcal{T}(g,k'',h'',(\overline{z},\overline{x}),\bot,B_0), \\
& \quad [k'' := cls(x, \mathcal{T}(f,k,h,(\overline{x}',\overline{y}'),\bot,B), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)), \\
& \quad\ h'' := cls(_, f.L(k,h,\overline{x}',\overline{y}'), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)), \overline{z} := \sigma(\overline{y})]\rangle \\
\to^* \ & \langle \mathcal{T}(P), \mathcal{T}(g,k'',h'',(\overline{z},\overline{x}),\bot,B_0), \\
& \quad [k'' := cls(x, \mathcal{T}(f,k,h,(\overline{x}',\overline{y}'),\bot,B), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)), \\
& \quad\ h'' := cls(_, f.L(k,h,\overline{x}',\overline{y}'), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)), \overline{z} := \sigma(\overline{y}), \overline{x} := 0]\rangle \\
= \ & \langle \mathcal{T}(P), \mathcal{T}(g,k'',h'',(\overline{z},\overline{x}),\bot,B_0), \\
& \quad [\overline{z} := \sigma(\overline{y}), \overline{x} := 0] \uplus \\
& \quad [k'' := cls(x, \mathcal{T}(f,k,h,(\overline{x}',\overline{y}'),\bot,B), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E))] \uplus \\
& \quad [h'' := cls(_, f.L(k,h,\overline{x}',\overline{y}'), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)\rangle \\
= \ & \langle \mathcal{T}(P), \mathcal{T}(g,k'',h'',(\overline{z},\overline{x}),\bot,B_0), \\
& \quad [\overline{z} := \sigma(\overline{y}), \overline{x} := 0] \uplus \mathcal{T}(P,k'',x,(f,\bot,B,\sigma,r,F,E)) \uplus \mathcal{T}(P,h'',(f,L,\sigma,r,F,E))\rangle \\
= \ & \mathcal{T}(S')
\end{aligned}
$$

$S = \langle P, f, H, \texttt{goto } L, \sigma, r, F, E\rangle \to \langle P, f, H', B, \sigma, r, F, E\rangle = S'$ かつ $f(\overline{x})\{\ldots; L(H') : B; \ldots\} \in P$ の場合

S は適格なので、$f(\overline{x})\{\texttt{var}\ \overline{y}; \ldots; L(H') : B; \ldots\} \in P$ が一意に存在して $dom(\sigma) = \{\overline{x}, \overline{y}\}$ であるから、

$$
\begin{aligned}
&\mathcal{T}(S)\\
&= \langle \mathcal{T}(P), \mathcal{T}(f,k,h,(\overline{x},\overline{y}), H, \texttt{goto}\ L), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)\rangle\\
&= \langle \mathcal{T}(P), f.L(k,h,\overline{x},\overline{y}), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)\rangle\\
&\to \langle \mathcal{T}(P), \mathcal{T}(f,k',h',(\overline{x},\overline{y}), H', B), [k' := \mathcal{T}(P,k,r,F)(k), h' := \mathcal{T}(P,h,E)(h), \overline{x} := \sigma(\overline{x}), \overline{y} := \sigma(\overline{y})]\rangle\\
&= \langle \mathcal{T}(P), \mathcal{T}(f,k',h',(\overline{x},\overline{y}), H', B), \sigma \uplus \mathcal{T}(P,k',r,F) \uplus \mathcal{T}(P,h',E)\rangle\\
&= \mathcal{T}(S')
\end{aligned}
$$

$S = \langle P, f, H, \texttt{return}\ x, \sigma, r, (g, H', B, \sigma', r', F, E), E'\rangle \to \langle P, g, H', B, \sigma'[r := \sigma(x)], r', F, E\rangle = S'$ の場合 S は適格であるから、$f(\overline{x})\{\texttt{var}\ \overline{y}; \ldots\} \in P$ ならびに $g(\overline{x}')\{\texttt{var}\ \overline{y}'; \ldots\} \in P$ が一意に存在して、

$$
\begin{aligned}
&\mathcal{T}(S)\\
&= \langle \mathcal{T}(P), \mathcal{T}(f,k,h,(\overline{x},\overline{y}), H, \texttt{return}\ x), \sigma \uplus \mathcal{T}(P,k,r,(g,H',B,\sigma',r',F,E)) \uplus \mathcal{T}(P,h,E')\rangle\\
&= \langle \mathcal{T}(P), k(x), \sigma \uplus \mathcal{T}(P,k,r,(g,H',B,\sigma',r',F,E)) \uplus \mathcal{T}(P,h,E')\rangle\\
&= \langle \mathcal{T}(P), k(x), \sigma \uplus [k := cls(r, \mathcal{T}(g,k',h',(\overline{x}',\overline{y}'), H', B), \sigma' \uplus \mathcal{T}(P,k',r',F) \uplus \mathcal{T}(P,h',E'))] \uplus \mathcal{T}(P,h,E')\rangle\\
&\to \langle \mathcal{T}(P), \mathcal{T}(g,k',h',(\overline{x}',\overline{y}'), H', B), \sigma' \uplus \mathcal{T}(P,k',r',F) \uplus \mathcal{T}(P,h',E') \uplus [r := \sigma(x)]\rangle\\
&= \mathcal{T}(S')
\end{aligned}
$$

$S = \langle P, f, H, \texttt{return}\ x, \sigma, r, (), E\rangle \to \langle P, \texttt{halt}\ \sigma(x)\rangle = S'$ の場合 S は適格であるから、$f(\overline{x})\{\texttt{var}\ \overline{y}; \ldots\} \in P$ が一意に存在して、

$$
\begin{aligned}
&\mathcal{T}(S)\\
&= \langle \mathcal{T}(P), \mathcal{T}(f,k,h,(\overline{x},\overline{y}), H, \texttt{return}\ x), \sigma \uplus \mathcal{T}(P,k,r,()) \uplus \mathcal{T}(P,h,E)\rangle\\
&= \langle \mathcal{T}(P), k(x), \sigma \uplus \mathcal{T}(P,k,r,()) \uplus \mathcal{T}(P,h,E)\rangle\\
&= \langle \mathcal{T}(P), k(x), \sigma \uplus [k := cls(r, halt(r), \emptyset)] \uplus \mathcal{T}(P,h,E)\rangle\\
&\to \langle \mathcal{T}(P), halt(r), [r := \sigma(x)]\rangle\\
&= \mathcal{T}(S')
\end{aligned}
$$

$S = \langle P, f, H, \texttt{if}\ x \le y\ \texttt{then}\ L_1\ \texttt{else}\ L_2, \sigma, r, F, E\rangle \to \langle P, f, H_1, B_1, \sigma, r, F, E\rangle = S'$ ただし $\sigma(x) \le \sigma(y)$ かつ $f(\overline{x})\{\ldots; L_1(H_1) : B_1; \ldots\} \in P$ の場合 S は適格なので、$f(\overline{x})\{\texttt{var}\ \overline{y}; \ldots; L_1(H_1) : B_1; \ldots\} \in P$ が一意に存在して、$dom(\sigma) = \{\overline{x}, \overline{y}\}$ であるから、

$$
\begin{aligned}
&\mathcal{T}(S)\\
&= \langle \mathcal{T}(P), \mathcal{T}(f,k,h,(\overline{x},\overline{y}), H, \texttt{if}\ x \le y\ \texttt{then}\ L_1\ \texttt{else}\ L_2), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)\rangle\\
&= \langle \mathcal{T}(P), \texttt{if}\ x \le y\ \texttt{then}\ f.L_1(k,h,\overline{x},\overline{y})\ \texttt{else}\ f.L_2(k,h,\overline{x},\overline{y}), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)\rangle\\
&\to \langle \mathcal{T}(P), f.L_1(k,h,\overline{x},\overline{y}), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)\rangle\\
&\to \langle \mathcal{T}(P), \mathcal{T}(f,k',h',(\overline{x},\overline{y}), H_1, B_1), [k' := \mathcal{T}(P,k,r,F)(k), h' := \mathcal{T}(P,h,E), \overline{x} := \sigma(\overline{x}), \overline{y} := \sigma(\overline{y})]\rangle\\
&= \langle \mathcal{T}(P), \mathcal{T}(f,k',h',(\overline{x},\overline{y}), H_1, B_1), \sigma \uplus \mathcal{T}(P,k',r,F) \uplus \mathcal{T}(P,h',E)\rangle\\
&= \mathcal{T}(S')
\end{aligned}
$$

$S = \langle P, f, H, \texttt{if}\ x \le y\ \texttt{then}\ L_1\ \texttt{else}\ L_2, \sigma, r, F, E\rangle \to \langle P, f, H_2, B_2, \sigma, r, F, E\rangle = S'$ ただし $\sigma(x) > \sigma(y)$ かつ $f(\overline{x})\{\ldots; L_2(H_2) : B_2; \ldots\} \in P$ の場合 S は適格なので、$f(\overline{x})\{\texttt{var}\ \overline{y}; \ldots; L_2(H_2) : B_2; \ldots\} \in P$ が一意に存在して、$dom(\sigma) = \{\overline{x}, \overline{y}\}$ であるから、

$$
\begin{aligned}
&\mathcal{T}(S)\\
&= \langle \mathcal{T}(P), \mathcal{T}(f,k,h,(\overline{x},\overline{y}), H, \texttt{if}\ x \le y\ \texttt{then}\ L_1\ \texttt{else}\ L_2), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)\rangle\\
&= \langle \mathcal{T}(P), \texttt{if}\ x \le y\ \texttt{then}\ f.L_1(k,h,\overline{x},\overline{y})\ \texttt{else}\ f.L_2(k,h,\overline{x},\overline{y}), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)\rangle\\
&\to \langle \mathcal{T}(P), f.L_2(k,h,\overline{x},\overline{y}), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)\rangle\\
&\to \langle \mathcal{T}(P), \mathcal{T}(f,k',h',(\overline{x},\overline{y}), H_2, B_2), [k' := \mathcal{T}(P,k,r,F)(k), h' := \mathcal{T}(P,h,E), \overline{x} := \sigma(\overline{x}), \overline{y} := \sigma(\overline{y})]\rangle\\
&= \langle \mathcal{T}(P), \mathcal{T}(f,k',h',(\overline{x},\overline{y}), H_2, B_2), \sigma \uplus \mathcal{T}(P,k',r,F) \uplus \mathcal{T}(P,h',E)\rangle\\
&= \mathcal{T}(S')
\end{aligned}
$$

$S = \langle P, f, L, \texttt{raise}, \sigma, r, F, E\rangle \to \langle P, f, H, B, \sigma, r, F, E\rangle = S'$ かつ $f(\overline{x})\{\ldots; L(H) : B; \ldots\} \in P$ の場合 S は適格であるから、$f(\overline{x})\{\texttt{var}\ \overline{y}; \ldots; L(H) : B; \ldots\} \in P$ が一意に存在して、

$$
\begin{aligned}
&\mathcal{T}(S)\\
&= \langle \mathcal{T}(P), \mathcal{T}(f,k,h,(\overline{x},\overline{y}),L,\texttt{raise}), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)\rangle\\
&= \langle \mathcal{T}(P), f.L(k,h,\overline{x},\overline{y}), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,E)\rangle\\
&\to \langle \mathcal{T}(P), \mathcal{T}(f,k',h',(\overline{x},\overline{y}),H,B), [k' := \mathcal{T}(P,k,r,F)(k), h' := \mathcal{T}(P,h,E)(h), \overline{x} := \sigma(\overline{x}), \overline{y} := \sigma(\overline{y})]\rangle\\
&= \langle \mathcal{T}(P), \mathcal{T}(f,k',h',(\overline{x},\overline{y}),H,B), \sigma \uplus \mathcal{T}(P,k',r,F) \uplus \mathcal{T}(P,h',E)\rangle\\
&= \mathcal{T}(S')
\end{aligned}
$$

$S = \langle P, f, \bot, \texttt{raise}, \sigma, r, F, (g, L, \sigma', r', F', E)\rangle \to \langle P, g, H, B, \sigma', r', F', E\rangle = S'$ **かつ** $g(\overline{x})\{\ldots; L(H) : B; \ldots\} \in P$ **の場合** S は適格であるから、$f(\overline{x})\{\texttt{var}\ \overline{y}; \ldots\} \in P$ ならびに $g(\overline{x}')\{\texttt{var}\ \overline{y}'; \ldots; L(H) : B; \ldots\} \in P$ が一意に存在して、

$$
\begin{aligned}
&\mathcal{T}(S)\\
&= \langle \mathcal{T}(P), \mathcal{T}(f,k,h,(\overline{x},\overline{y}),\bot,\texttt{raise}), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,(g,L,\sigma',r',F',E))\rangle\\
&= \langle \mathcal{T}(P), h(), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,(g,L,\sigma',r',F',E))\rangle\\
&= \langle \mathcal{T}(P), h(), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus [h := cls(_, g.L(k,h',\overline{x}',\overline{y}'), \sigma' \uplus \mathcal{T}(P,k,r',F') \uplus \mathcal{T}(P,h',E))]\rangle\\
&\to \langle \mathcal{T}(P), g.L(k,h',\overline{x}',\overline{y}'), \sigma' \uplus \mathcal{T}(P,k,r',F') \uplus \mathcal{T}(P,h',E)\rangle\\
&\to \langle \mathcal{T}(P), \mathcal{T}(g,k',h'',(\overline{x}',\overline{y}'),H,B), [k' := \mathcal{T}(P,k,r',F')(k), h'' := \mathcal{T}(P,h',E)(h'), \overline{x}' := \sigma(\overline{x}'), \overline{y}' := \sigma(\overline{y}')]\rangle\\
&= \langle \mathcal{T}(P), \mathcal{T}(g,k',h'',(\overline{x}',\overline{y}'),H,B), \sigma \uplus \mathcal{T}(P,k',r',F') \uplus \mathcal{T}(P,h'',E)\rangle\\
&= \mathcal{T}(S')
\end{aligned}
$$

$S = \langle P, f, \bot, \texttt{raise}, \sigma, r, F, ()\rangle \to \langle P, \texttt{abort}\rangle = S'$ **の場合** S は適格であるから、$f(\overline{x})\{\texttt{var}\ \overline{y}; \ldots\} \in P$ が一意に存在して、

$$
\begin{aligned}
&\mathcal{T}(S)\\
&= \langle \mathcal{T}(P), \mathcal{T}(f,k,h,(\overline{x},\overline{y}),\bot,\texttt{raise}), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,())\rangle\\
&= \langle \mathcal{T}(P), h(), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus \mathcal{T}(P,h,()\rangle\\
&= \langle \mathcal{T}(P), h(), \sigma \uplus \mathcal{T}(P,k,r,F) \uplus [h := cls(_, abort(), \emptyset)]\rangle\\
&\to \langle \mathcal{T}(P), abort(), \emptyset\rangle\\
&= \mathcal{T}(S')
\end{aligned}
$$

□