題目　Scheme,Guile 勉強ノート[1)]／Scheme-0 の小股意味論[2)]
作者　AlgoKajya
日付　2021.02.04
概要　このノートは，call/cc を含む単純な言語（勝手に Scheme-0 と呼んでいます）に対して簡約規則に基づく小股意味論（small-step sematics）を定義しようと試みています．おもな目的は call/cc の動作について理解を深めることです．なお，このノートの内容は，次の本の第 5 章までを勉強したことに基づいています．

- M. Felleisen, R. B. Findler, and M. Flatt: Semantics Engineering with PLT Redex, The MIT Press, 2009.

特に，評価文脈を使って call-by-value 戦略を簡約規則として定義する方法を学びました．

[1)] Lisp を学び直そうと思い立って，Scheme を勉強しています．処理系は Guile を使っています．Scheme や Guile について勉強したことをまとめて，勉強ノートを趣味として作っています．一応，読み物にしています．他の人達の参考になるかどうかは分かりませんが，参考になるのならば好き勝手に使って下さい．ただし，内容は無保証です．使うにしても自己責任でお願いします．

[2)]「小股意味論」なんて言葉は専門家の皆さんからお叱りを受けるかも知れません．でも，筆者のような素人の試みに対してはこんな名称が適当だろうと思います．

目 次

1　単純な言語

▌文法

1.1

単純な言語の文法を図 1 （3 頁）に示します．勝手ながら，この言語を **Scheme-0** と呼ぶことにします．

- M や M_i は一般的な式を表す構文カテゴリーで，V や V_i は「値として解釈される式」（以後，値式 ）を表す構文カテゴリーです．
- 「定数」は，真偽値，数値，文字，文字列などのことです．「標準手続き」は処理内容が事前に決められている手続きのことです．ただ，`#f` を除いて，「定数」や「標準手続き」の内容は以下の議論に本質的に関与しないので特定しません．具体例を考えるときには，Scheme の定数や標準手続きを自由に使います．
- `let`^{1} 式の上付き添字の 1 は束縛を 1 つしか指定しないことを示しています．束縛は 1 つだけなので，束縛部のカッコは二重にしていません．`let`^{1} 式は `lam` 式を使ったマクロとして簡単に定義できますが，具体例の中で頻繁に使用するので基本的な構文要素としています．
- `rec`^{1} 式は再帰的な手続きを表現するための式です．直感的には，`define` 形式を「式化」したものと見なすことができます．`rec`^{1} 式は手続きを表現しているのですが，手続きの形式（引数を受け取って本体を実行するという形式）をしていないので値式にはしません．

  上付き添字の 1 は手続きを 1 つしか作らないことを示しています．第 6.2 節にて，相互再帰的な複数の手続きを同時に定義可能な形式（`rec` 式）を示します．
- `ccl` 式は

  $$\texttt{(call/cc (lambda (}X\texttt{) }M\texttt{))}$$

  の省略形です．従って，X は継続を受け取るための変数です．
- `cnt` 式は継続そのものを表すための式で，X を引数とする手続きを表しています．継続は手続き（値の 1 つ）なので，`cnt` 式は値式になります．
- `lam` 式はラムダ式の省略記法です．

¶ 式全体，値式全体，変数記号全体，標準手続き（の名前の）全体を，それぞれ，M，V，X，O を太字の立体にして **M**， **V**， **X**， **O** で表します．さらに，定数全体を **C** で表します．

図 1: Scheme-0 の文法

$L,\ M,\ M_i,\ N,\ N_i,\ P,\ Q$	::=	V
		| $\texttt{(}M_0\ M_1\ \ldots\ M_n\texttt{)}$
		| $\texttt{(if }M_1\ M_2\ M_3\texttt{)}$
		| $\texttt{(let}^1\ \texttt{(}X\ M_1\texttt{)}\ M_2\texttt{)}$
		| $\texttt{(rec}^1\ X\ M\texttt{)}$
		| $\texttt{(ccl (}X\texttt{) }M\texttt{)}$
$U,\ V,\ V_i,\ W$	::=	`#f` | 定数 | O | X
		| $\texttt{(lam (}X_1\ \ldots\ X_n\texttt{) }M\texttt{)}$
		| $\texttt{(cnt (}X\texttt{) }M\texttt{)}$
$X,\ X_i,\ Y,\ Z$	::=	変数
O	::=	標準手続き

¶ 等式「$M = N$」は M と N が構文的に（記号的な式として）等しいことを示します．少し言い方を変えると，あらゆる文脈において M と N が交換可能であることを示します．例えば，

N = `(lam (x) (+ a x))`

といった等式は，左辺と右辺が構文的に等しいことを示し，あらゆる式の部分式として両辺が交換可能であることを示します．そのため，例えば，

`(lam (a)` N`)` と `(lam (a) (lam (x) (+ a x)))`

は構文的に同じ式を表します．

例 1.2

以下の式は自然数 n に対して $n!$ を計算する手続きを表しています．

```
1 (rec¹ fact (lam (n) (if (< n 2) 1 (* n (fact (- n 1))))))
```

`rec`^{1} を `define` に置き換えてみると，Scheme の手続き定義になります．`rec`^{1} 式は，直感的には，`define` 形式を「式化」したものです．`define` 形式の場合には，ラムダ式によって表された手続きが変数 `fact` に束縛され，変数 `fact` を通してその手続きが利用できるようになります．一方，`rec`^{1} 式は，手続きを変数に束縛することはなく，そのため名前を取り出して実引数に適用することはできません．つまり，`(fact 3)` といった式を書くことはできません．代わりに `rec`^{1} 式自身が手続きをマクロ的に表現することになり，実引数には `rec`^{1} 式そのものを適用します．例えば，次の式は 3! を計算します．

```
1 ((rec¹ fact (lam (n) (if (< n 2) 1 (* n (fact (- n 1))))))  3)
```

¶ 簡約規則に基づく操作的意味論は実行環境に相当する仕組みがないため，計算状況は式そのものによって表現しなければなりません．つまり，計算に必要なあらゆる情報は式そのものに内蔵していなければなりません．手続き名ではなく `rec`^{1} 式そのものを明示しなければならないのは，このためです．

¶ `rec`^{1} 式の用途は再帰的な手続きをマクロ的に表現することです[1)]．第 6.1 節にて `rec`^{1} 式をマクロ展開する方法を示します．その方法を使うと，`rec`^{1} 式に関する簡約規則（計算規則）を Scheme-0 から除外しても原理的には言語機能を減らすことにはなりません．

例 1.3

以下のラムダ式はリスト `lst` の反転を計算します．

```
1 (lam (lst)
2    ((rec¹ loop
3       (lam (olst rlst)
4          (if (null? olst)
5              rlst
6              (loop (cdr olst) (cons (car olst) rlst))))) lst '()))
```

`rec`^{1} 式は，それ自身が手続きを表現しているので，あらゆる式の中で自由に使うことができます．

例 1.4

`call/cc` を含む式を簡約（式変形）する様子を例示します．以下の等号のあとの式は簡約対象の部分式を赤字で示しています．矢印のうしろは簡約後の式を示しています．それから，where 節は

[1)]「再帰的」であることは必須ではありません．`rec`^{1} 式によって非再帰的な手続きを表現してもかまいません．

記述の複雑化を避けるための便法です．K は call/cc（ccl 式）によって生成された継続を表しています．

$$
\begin{array}{rl}
 & \texttt{(+ 50 (ccl (k) (let}^1\texttt{ (x (k 5)) (* 10 x))))} \\
= & \texttt{(+ 50 (ccl (k) (let}^1\texttt{ (x (k 5)) (* 10 x))))} \\
\xrightarrow[\text{CBV}]{} & \texttt{(+ 50 (let}^1\texttt{ (x (K 5)) (* 10 x)))} \quad \text{where } \texttt{K = (cnt (z) (+ 50 z))} \\
= & \texttt{(+ 50 (let}^1\texttt{ (x (K 5)) (* 10 x)))} \quad \text{where } \texttt{K = (cnt (z) (+ 50 z))} \\
\xrightarrow[\text{CBV}]{} & \texttt{(+ 50 5)} \\
= & \texttt{(+ 50 5)} \\
\xrightarrow[\text{CBV}]{} & 55
\end{array}
$$

1 番目の $\xrightarrow[\text{CBV}]{}$ は，継続 K を生成して ccl 式の仮引数 k に代入した結果を示しています．2 番目の $\xrightarrow[\text{CBV}]{}$ は，$\texttt{let}^1$ 式の束縛を評価するために継続 K を呼び出した結果を示しています．ここでは継続を囲んでいた文脈がすべて捨てられて，代わりに継続を生成したときの文脈に置き換わります．3 番目の $\xrightarrow[\text{CBV}]{}$ は加算を計算した結果を示しています．

Guile(2.2.4) を使って実行してみると同じ結果が得られます．

```
scheme@(guile-user)> (+ 50 (call/cc (lambda (k) (let ((x (k 5))) (* 10 x)))))
$ 1 = 55
scheme@(guile-user)> ,trace (+ 50 (call/cc  …… 同上 …… ))
trace: |  (_ #<procedure 55d839f88f10 at <unknown port>:2:22 (k)>)
trace: |  (_ #<continuation 55d83a1acfc0>)
trace: |  |  (_ 5)
trace: |  |  55
```

3 行目以降は trace コマンドを使って手続き呼び出しをトレースしてみた結果です．Guile Reference Manual を見ても詳しい説明が見当たらないのですが，おそらくアンダースコア（_）は呼び出し中の手続きを表していて，そのうしろは実引数を表していると思われます．縦棒は手続き呼び出しのネストを表しています．トレースの 1 行目は call/cc の呼び出し，2 行目はラムダ式の呼び出し，3 行目は継続の呼び出しを表していると思われます．Scheme-0 の ccl 式は

```
(call/cc (lmabda (k) ...  ))
```

という式を略記したものなので，先に示した簡約過程と Guile による計算過程は同じことをしていると見なすことができます．

▮ 用語

1.5

式を書き換えて計算を一歩進めることを **簡約**（**reduciton**）と言います．例えば，

- (+ 1 2) を 3 に書き換える
- (* (+ 1 2) (+ 2 3)) を (* 3 (+ 2 3)) に書き換える
- ((lam (x) (* x x)) 5) を (* 5 5) に書き換える

などが簡約です．さらに，変形前の式を変形後の式に**簡約する**（**reduce**）と言ったりします．

2　call-by-value 戦略

▮ CBV 戦略

2.1

call-by-value 戦略（**CBV 戦略**）は，次の条件を満たす評価戦略です．

- 手続き適用の式（M_0 M_1 ... M_n）を評価するとき，まず実引数 $M_1 \ldots M_n$ を評価して値式にしたあとで，それらの値式に手続き（M_0 の評価結果）を適用する．
- `let`[1] 式（`let`[1]（X M_1）M_2）を評価するとき，まず M_1 を評価して値式にして，その値式を X に束縛したあとで M_2 を評価する[2)]．
- `rec`[1] の本体，`ccl` 式の本体，`lam` 式の本体，`cnt` 式の本体は実引数に適用されるまで評価しない．同様に，`let`[1] 式の本体は束縛の評価が完了するまで評価しない．さらに，`if` 式の then 部と else 部は条件部の評価が完了するまで評価しない．

手続き適用の評価順序

2.2

CBV 戦略は，手続き適用の式（M_0 M_1 ... M_n）の手続き M_0 や実引数 M_1〜M_n の評価順序について特段の条件を設定していません．どんな順序で評価してもよいことになります．これは R[7]RS（4.1.3 節）の次の記述と一致します．

・ In contrast to other dialects of Lisp, the order of evaluation is unspecified, ...

・ The order of evaluation may be chosen differently for each procedure call.

手続きの呼び出しごとに評価順序が変化するような処理系があるとは思えませんが，Scheme の仕様書は手続き適用の評価順序に関して最大限の自由を認めていることになります．

ところが，次節で説明する 評価文脈 という概念は，手続き適用の評価順序を固定しないと定義できません．しかも，継続に関する簡約規則を定義するためには，この概念が必要と思われます．さらに， 継続 そのものの意味も評価順序に応じて変化します．例えば，

```
(someproc (* 1 2) (call/cc (lam (k) (k 3))) (* 4 5))
```

という式を評価するときに，`someproc` の引数を左から右に向かって評価したときと，右から左に向かって評価したときで `call/cc` によって生成される継続は違うものになります．つまり，継続を正確に定義するためには手続き適用の評価順序を固定しなければならないと思われます[3)]．

以上のような事情から，手続き適用の評価順序を固定します．具体的には，CBV 戦略に次の条件を追加します．

- 手続き適用を構成する手続きや実引数は左から右に向かって評価する．

評価文脈の概念にこの条件が組み込まれ，その概念を使って簡約規則（計算規則）が定義されます．

¶ これ以後の議論の中で，上の評価順序を強調するときには， **CBV 戦略+評価順序** と書くことにします．

3 評価文脈

形式上の評価文脈

3.1

形式上の 評価文脈 （**evaluation context**）E を図 2 （7 頁）の文法によって定義します[4)]．ここで，□ は ホール と呼ばれます．形式上の評価文脈はホールをちょうど 1 つ含むような式です．さらに，評価文脈 E のホールを式 M に置き換えて得られる式を $\boldsymbol{E[M]}$ で表します．

[2)] `let`[1] 式はラムダ式によって表現できるので，`let`[1] 式に関する条件はラムダ式に関する条件にまとめることができます．

[3)] もしかしたら評価順序とは無関係に継続を定義できるのかも知れません．そうだったとしても，いまの筆者には分かりません．

[4)] 評価文脈の定義の中にある □ は `[]` で表されるようなのですが，このノートでは □ を使用します．□ のほうが誤読が少ないと思っているだけで他意はありません．

$$
\begin{aligned}
E,\ E_i \quad ::= {} & \square \\
& \mid (E\ \ M\ \ \ldots\ \ M) \\
& \mid (V\ \ \ldots\ V\ \ E\ \ M\ \ \ldots\ \ M) \\
& \mid (\texttt{if}\ \ E\ \ M\ \ M) \\
& \mid (\texttt{let}^1\ \ (X\ \ E)\ \ M)
\end{aligned}
$$

図 2: 評価文脈の定義（文法）

¶ 手続き適用の形をした評価文脈には次のものがあります．

- $(O\ \ \ldots\ \ \ E\ \ \ldots\ \ )$
- $((\texttt{lam}\ \ (X_1\ \cdots\ X_n)\ \ M)\ \ \ldots\ \ \ E\ \ \ldots\ \ )$
- $((\texttt{cnt}\ \ (X)\ \ M)\ \ \ldots\ \ \ E\ \ \ldots\ \ )$

標準手続き，lam 式，および cnt 式は値式なので，これらはすべて $(V\ \ \ldots\ V\ \ E\ \ M\ \ \ldots\ M)$ といった形式の評価文脈です．

例

3.2

評価文脈は，評価対象となっている部分式を除いた残りの部分を表しています．ホール □ はその部分式を除いたときの「穴」を示すための構文要素です．例えば，

```
(* 1 (+ 2 3) (- 4 5))
```

という式を評価することを考えたとき，前節で述べた評価順序に従うと，(+ 2 3) という部分式を評価することになります．この部分式をホールに置き換えた

```
(* 1 □ (- 4 5))
```

という式が評価文脈です．評価文脈の真意は，「各時点で評価対象となっている部分式」の「その後の計算内容（つまり，継続）」を明示することです．

さらに，この評価文脈を E とおくとき，そのホール □ を，例えば $M =$ (/ 6 3) に置き換えれ得られる

```
(* 1 (/ 6 3) (- 4 5))
```

といった式を $E[M]$ で表します．

¶ 評価文脈の真意は次の 2 点です．

(1) 各時点の評価対象をホールによって明示すること．

(2) 評価対象のその後の計算内容（つまり，継続）を明示すること．

評価文脈の基本的な性質

3.3

評価文脈はホールをちょうど 1 つ持っています．そこで，これ以後の言葉遣いとして，ある位置にホールが存在できることを，その位置に ホールが入る と言うことにします．この表現を使うと，評価文脈は次のような性質を持ちます．いずれも評価文脈の文法から直ちに分かることです．

(1) 手続き適用の式 $(M_0\ \ M_1\ \ \ldots\ \ M_n)$ に対して，ホールが入っている M_k の左側（M_0〜M_{k-1}）はすべて値式（V）でなければなりません．逆に言うと，各 M_i に対して，M_i が値式でないとき M_i より右側にホールが入ることはありません．

(2) if 式の then 部と else 部にホールが入ることはありません．

(3) $\texttt{let}^1$ 式の本体にホールが入ることはありません．

(4) $\texttt{rec}^1$ 式，ccl 式，lam 式，cnt 式の内部にホールが入ることはありません．

¶ 評価文脈の真意は「各時点の評価対象」を特定し,「その後の計算内容（継続）」を明示することです．でも，評価文脈がその真意を発揮するためには，上の定義だけでは十分ではありません．例えば，上で例示した式に対して

(□ 1 (+ 2 3) (- 4 5)) や (* □ (+ 2 3) (- 4 5))

も評価文脈になります．でも，ホールに置き換えた部分式（*と 1）はすでに評価済みのため，これらは「各時点の評価対象」を特定することにはなっていません．本物の評価文脈（CBV-評価文脈）を定めるためには，次に定める CBV-簡約基と組み合わせることが必要です．これと組み合わせたことの真の意図をそのあとの命題が述べています．

CBV-簡約基

3.4

CBV-簡約基（**CBV-redex**）C を図 3 （8 頁）の文法によって定義します．CBV-簡約基は簡約を行う部分式を表しています．つまり，簡約規則に基づいて計算を進めたとき，各時点の CBV-簡約基を次々と簡約していきます．

¶ 手続き適用 $(V_0\ V_1\ \ldots\ V_n)$ の形をした CBV 簡約基には次のものがあります．

- $(O\ V_1\ \ldots\ V_n\ )$
- $(\texttt{(lam (x)}\ M\texttt{)}\ V_1\ \ldots\ V_n)$
- $(\texttt{(cnt (x)}\ M\texttt{)}\ V)$

命題（評価文脈と CBV-簡約基の一意性）

3.5

M を任意の式として，値式でないとします．このとき，

- $M = E[C]$ を満たす評価文脈 E と CBV-簡約基 C の組が一意的に定まります．

その一意的に定まる評価文脈 E を M の **CBV-評価文脈** と呼ぶことにします．

¶ この命題の証明は A.1 節に示します．

¶ これ以後，$M = E[C]$ や $M = E_i[C_i]$ と書いたとき，特に断っていなくても，E や E_i は CBV-評価文脈を表し，C や C_i は CBV-簡約基を表します．さらに，C や C_i の代わりに D や D_i を使うこともあります．

¶ あらゆる式 M は，一般に，CBV-簡約基を幾つも含んでいます．例えば，

(* 1 (+ 2 3) (* (+ 4 5) (/ 6 3)))

という式は，(+ 2 3)，(+ 4 5)，(/ 6 3) の 3 つの CBV-簡約基を含んでいます．これらのうち，CBV 戦略+評価順序に基づいて簡約すべきなのは (+ 2 3) です．この CBV-簡約基を特定するのが CBV-評価文脈です．実際，(+ 2 3) をホール □ に置き換えたものは形式上の評価文脈になりますが，(+ 4 5) や (/ 6 3) をホールに置き換えても形式上の評価文脈になりません．

上の命題は，評価文脈と CBV-簡約基を組み合わせることによって,「各時点で簡約すべき部分式」

図 3: CBV-簡約基の定義（文法）

$$
\begin{array}{rcl}
C,\ C_i, D, D_i & ::= & (V_0\ V_1\ \ldots\ V_n) \\
& | & (\texttt{if}\ V\ M_1\ M_2) \\
& | & (\texttt{let}^1\ (X\ V)\ M) \\
& | & (\texttt{rec}^1\ X\ M) \\
& | & (\texttt{ccl}\ (X)\ M)
\end{array}
$$

と「その後の計算内容（継続）」が一意的に特定できることを述べています．

例 3.6

CBV-評価文脈と CBV-簡約基を求めるには，とりあず色々なところにホールを入れてみて，それが形式上の評価文脈になるかどうか，ホールのところの部分式が CBV-簡約基になるかどうかを判定してみればよいと思います[5]．以下の E と C は，それぞれ，CBV-評価文脈と CBV-簡約基の候補を示しています．

(1) `(+ 1 (* 2 (+ 3 4) (+ 5 6)))`

ホールを色んなところに入れてみると，CBV-評価文脈と CBV-簡約基の候補として次のようなものがあります．

(a) $E = \square$, $C =$ `(+ 1 (* 2 (+ 3 4) (+ 5 6)))`

(b) $E =$ `(`$\square$ `1 (* 2 (+ 3 4) (+ 5 6)))`, $C =$ `+`

(c) $E =$ `(+ `$\square$ `(* 2 (+ 3 4) (+ 5 6)))`, $C =$ `1`

(d) $E =$ `(+ 1 `$\square$`)`, $C =$ `(* 2 (+ 3 4) (+ 5 6))`

(e) $E =$ `(+ 1 (`$\square$ `2 (+ 3 4) (+ 5 6)))`, $C =$ `*`

(f) $E =$ `(+ 1 (* `$\square$ `(+ 3 4) (+ 5 6)))`, $C =$ `2`

(g) $E =$ `(+ 1 (* 2 `$\square$ `(+ 5 6)))`, $C =$ `(+ 3 4)`

(h) $E =$ `(+ 1 (* 2 (+ 3 4) `$\square$`))`, $C =$ `(+ 5 6)`

本当はもっとたくさんありますが，とりあえずこれくらいにします．

(a)〜(g) の E は形式上の評価文脈です．でも，(h) の E は違います．なぜなら，手続き適用におけるホールの左側は値式しか許されないからです．(a)〜(f) の C は CBV-簡約基ではありません．(g) と (h) の C は CBV-簡約基です．以上から，この式の CBV-評価文脈と CBV-簡約基は (g) です．

(2) `((lam (x)` M_1`) (if (= 1 0)` M_2 M_3`))`

M_1〜M_3 は適当な式を表しています．評価文脈の定義から，`lam` 式の内部にホールが入ることはありません．さらに，`if` 式の条件部にホールが入ることはありますが，then 部や else 部にホールが入ることはありません．それから，CBV-簡約基の定義から `lam` 式そのものが CBV-簡約基になることはありません．さらに，条件の評価が完了していない `if` 式が CBV-簡約基になることもありません．以上のことから，この式の場合は次のものしかありません．

$E =$ `((lam (x)` M_1`) (if `$\square$ M_2 M_3`))`, $C =$ `(= 1 0)`

少し言い方を変えると，上のものを Scheme の式と見なしたとき，通常の Scheme の処理系は `if` 式の条件を評価しようとするでしょう．つまり，CBV-簡約基は，通常の処理系の各時点の評価対象を示していて，CBV-評価文脈はその評価が終わったあとの計算内容（つまり，継続）を示しています．

(3) `(+ 1 (let`^{1} `(x 10) (* x 5)))`

`let`^{1} 式の bind 部にホールが入ることはありますが，そこはすでに値式（`10`）になっています．それから，`let`^{1} 式の本体にホールが入ることはありません．従って，この式の CBV-評価文脈と CBV-簡約基は次のものになります．

---

[5] 場当たり的なことを述べていますが，CBV-評価文脈と CBV-簡約基を求めることは純粋に構文解析の問題です．ただ，その構文解析が単純なパターンマッチングでは済まないので，手計算で求めるときには多少の試行錯誤をしなければなりません．

$E =$ `(+ 1 □)`, $C =$ `(let`^{1} `(x 10) (* x 5))`

(4) `((rec`^{1} `someproc (lam (str)` M`)) "Hi")`
ここで，M は適当な式を表します．これは，`rec`^{1} 式が表す手続きを文字列`"Hi"`に適用するといった手続き適用の式です．`rec`^{1} 式は値式ではないので，その `rec`^{1} 式がホールになります．従って，この式の CBV-評価文脈と CBV-簡約基は次のものになります．

$E =$ `(□ "Hi")`, $C =$ `(rec`^{1} `someproc (lam (str)` M`))`

(5) `(let`^{1} `(x (ccl (k) k)) (x (lam (p) "hi")))`
`let`^{1} 式の bind 部の計算式がホールになります．従って，この式の CBV-評価文脈と CBV-簡約基は次のものになります．

$E =$ `(let`^{1} `(x □) (x (lam (p) "hi")))`, $C =$ `(ccl (k) k)`

(6) `(let`^{1} `(p (lam (x) (* 2 x))) (p 10)))`
`let`^{1} 式の bind 部の計算式はすでに値式（ラムダ式）になっています．さらに `let`^{1} 式の本体にホールが入ることはありません．そのため，`let`^{1} 式全体がホールになります．この式の CBV-評価文脈と CBV-簡約基は次のものになります．

$E = □$, $C =$ `(let`^{1} `(p (lam (x) (* 2 x))) (p 10)))`

(7) `(+ 1 (* 2 (+ 3 ((cnt (x) (+ 1 x)) 4))))`
この式の CBV-評価文脈と CBV-簡約基は次のものになります．

$E =$ `(+ 1 (* 2 (+ 3 □)))`, $C =$ `((cnt (x) (+ 1 x)) 4)`

少し補足します．まず，`cnt` 式は継続を表しています．つまり，1 引数の手続きです．上の CBV-簡約基はその手続きを 4 に適用する手続き適用になっています．`cnt` 式と 4 はともに値式なので，これは $(V_0\ V_1)$ といった CBV-簡約基の形式をしています．

4 簡約規則

Scheme-0 に限らず，簡約規則（式変形の規則）を定義するためには，代入 を定義する必要があり，代入を定義するためには自由変数 を定義する必要があります．

▮束縛変数と自由変数

4.1

式 M の中で，次の変数を 束縛変数 （**bound variable**）と言います．

- `let`^{1} 式のローカル変数
- `rec`^{1} 式の中で手続き名としてして使用している変数
- `ccl` 式，`lam` 式，`cnt` 式の中で仮引数として使用している変数

これら以外の変数のことを M における 自由変数 （**free variable**）と言います．

¶ 束縛変数と自由変数はそれぞれの式に相対的に決まります．例えば，

```
(lam (x) (+ x y))
```

という式において，`x` は束縛変数で `y` は自由変数です．でも，このラムダ式の本体である

```
(+ x y)
```

といった式だけに注目すると，`x` も `y` もともに自由変数になります．

¶ ある変数名が 1 つの式の中で束縛変数と自由変数の両方を表すことがあり得ます．例えば，

```
(+ x (cnt (x) x))
```

という式において，左から 1 番目の x は自由変数を表し，2 番目と 3 番目の x は束縛変数を表します．当然，自由変数である x と束縛変数である x は，名前は同じでも異なる変数を表しています．

自由変数（の記号）の集合

4.2

式 M の中の自由変数（の記号）からなる集合を **FV(M)** で表します．この集合は再帰的に次のように定義されます．

(1) 定数 c に対して $\mathrm{FV}(c) = \emptyset$.

(2) 標準手続き o に対して $\mathrm{FV}(o) = \emptyset$.

(3) $\mathrm{FV}(\texttt{(}M_0\ \ldots\ M_n\texttt{)}) = \mathrm{FV}(M_0) \cup \cdots \cup \mathrm{FV}(M_n)$

(4) $\mathrm{FV}(\texttt{(if }M_1\ M_2\ M_3\texttt{)}) = \mathrm{FV}(M_1) \cup \mathrm{FV}(M_2) \cup \mathrm{FV}(M_3)$

(5) $\mathrm{FV}(\texttt{(let}^1\ \texttt{(}X\ M_1\texttt{)}\ M_2\texttt{)}) = \mathrm{FV}(M_1) \cup (\mathrm{FV}(M_2) - \{X\})$

(6) $\mathrm{FV}(\texttt{(rec}^1\ X\ M\texttt{)}) = \mathrm{FV}(M) - \{X\}$

(7) $\mathrm{FV}(\texttt{(ccl (}X\texttt{)}\ M\texttt{)}) = \mathrm{FV}(M) - \{X\}$

(8) $\mathrm{FV}(\texttt{(lam (}X_1\ \ldots\ X_n\texttt{)}\ M\texttt{)}) = \mathrm{FV}(M) - \{X_1, \ldots, X_n\}$

(9) $\mathrm{FV}(\texttt{(cnt (}X\texttt{)}\ M\texttt{)}) = \mathrm{FV}(M) - \{X\}$

代入

4.3

式 M の中の自由変数 X を式 N によって記号的に置き換えることを 代入（**substitution**）と言います．このノートでは，

$$M\{X \leftarrow N\}$$

で表します．でも，このままの定義ではダメで，名前の衝突を避けなければなりません． 名前の衝突 というのは，

- 置き換えによって，N の中の自由変数が束縛変数に強制的に変化させられてしまう事態

のことです．例えば，$M =$ `(lam (x a) (+ x y a))` の中の自由変数 y を $N =$ `(* a 5)` で素朴に置き換えたとき，

```
(lam (x a) (+ x (* a 5) a))
```

となってしまって，N の中の自由変数 a が束縛変数に変化してしまいます．このような事態を避けなければなりません．つまり，自由変数は，どんな置き換えを行おうと自由変数のままにしなければいけません．

名前の衝突は，

- 束縛変数の名前を新たなものに変更することによって避けることができます．

例えば，置き換えを行う際に，M の中の束縛変数 a の名前を z などに同時に変更すれば

```
(lam (x z) (+ x (* a 5) z))
```

となって N の中の a は置き換えたあとも自由変数でいることができます．なお，束縛変数（仮引数やローカル変数）の名前を変更しても式の意味（計算内容）は変化しないことに注意して下さい．束縛変数の名前を新たなものに変更すればよいとするのは，このことに基づいています．

名前の衝突は，lam 式だけでなく，仮引数や固有の名前を指定するあらゆる式で発生する可能性があります．Scheme-0 では，$\texttt{let}^1$ 式，$\texttt{rec}^1$ 式，ccl 式，cnt 式，そして lam 式で発生する可能性があります．

¶ 自由変数はどこかで（多くの場合，処理系で）特定の意味に束縛された名前です．その意味は計算が終了するまで維持しなければなりません．例えば，次のような式 N を考えてみましょう．

$N =$ `(cdr '((1 2) 3 4))`

N の cdr は「リストの cdr 部を取り出す」といった標準手続きに束縛された名前（自由変数）です．N を利用する計算は「N における cdr の意味」（つまり，標準手続きの意味）を維持しなければなりません．少し言い方を変えると，N を利用する計算は，N そのものを利用しても，N の評価結果である'(3 4) を利用しても，それらの結果は同じでなければなりません．

そこで，（ややムリスジの感もありますが）次のような式 M を考えてみましょう．

$M =$ `((lam (cdr) (cdr y)) car)`

この式の y に N の評価結果である'(3 4) を代入して M を評価すると，その結果は 3 になります．一方，何も考えずに y を N に置き換えた次の式を評価すると，その結果は 1 になります．

```
((lam (cdr) (cdr (cdr '((1 2) 3 4)))) car)
```

結果が変わってしまったのは，N の中の cdr が置き換えたあとのラムダ式の仮引数（束縛変数）に変化してしまって，元々の意味を失ってしまったからです．そこで，束縛変数の cdr を同時に変更して N を代入します．

```
((lam (p) (p (cdr '((1 2) 3 4)))) car)
```

こうすれば，「N における cdr」は元々の意味を失わずに済みます．

簡約規則に基づく意味論は，自由変数の意味（束縛）を維持する特別な仕組みはいっさいありません．そのため，自由変数の意味を維持するためには，自由変数のままにする必要があります．

代入の形式的な定義

4.4

束縛変数の名前を変更することによって名前の衝突を回避することをきちんと把握しておけば，代入の形式的な定義はほとんどの場合で必要ないように思います．でも，参考までに，代入の正確な定義を示します．代入 $M\{X \leftarrow N\}$ は再帰的に次のように定義されます．なお，新たな変数 Z は，代入に関わる式の中で使われていないものであれば，適当なものを自由に選択してかまいません．

(1) 定数 c に対して $c\{X \leftarrow N\} = c$

(2) 標準手続き o に対して $o\{X \leftarrow N\} = o$

(3) 変数 X, Y に対して，
$X\{X \leftarrow N\} = N$
$Y\{X \leftarrow N\} = Y \quad \text{if } Y \neq X$

(4) $(M_0\ \ldots\ M_n)\{X \leftarrow N\} = (M_0\{X \leftarrow N\}\ \ldots\ M_n\{X \leftarrow N\})$

(5) $(\texttt{if}\ M_1\ M_2\ M_3)\{X \leftarrow N\} = (\texttt{if}\ M_1\{X \leftarrow N\}\ M_2\{X \leftarrow N\}\ M_3\{X \leftarrow N\})$

(6) $(\texttt{let}^1\ (X\ M_1)\ M_2)\{X \leftarrow N\} = (\texttt{let}^1\ (X\ M_1\{X \leftarrow N\})\ M_2)$
$(\texttt{let}^1\ (Y\ M_1)\ M_2)\{X \leftarrow N\} = (\texttt{let}^1\ (Y\ M_1\{X \leftarrow N\})\ M_2\{X \leftarrow N\})$
if $Y \neq X$ かつ $Y \notin \mathrm{FV}(N)$.
$(\texttt{let}^1\ (Y\ M_1)\ M_2)\{X \leftarrow N\} = (\texttt{let}^1\ (Z\ M_1\{X \leftarrow N\})\ M_2\{Y \leftarrow Z\}\{X \leftarrow N\})$
if $Y \neq X$ かつ $Y \in \mathrm{FV}(N)$ where Z は新たな変数.

(7) $(\texttt{rec}^1\ X\ M)\{X \leftarrow N\} = (\texttt{rec}^1\ X\ M)$
$(\texttt{rec}^1\ Y\ M)\{X \leftarrow N\} = (\texttt{rec}^1\ Y\ M\{X \leftarrow N\})$

if $Y \neq X$ かつ $Y \notin \mathrm{FV}(N)$.

$(\texttt{rec}^1\ Y\ M)\{X \leftarrow N\} = (\texttt{rec}^1\ Z\ M\{Y \leftarrow Z\}\{X \leftarrow N\})$

if $Y \neq X$ かつ $Y \in \mathrm{FV}(N)$　where Z は新たな変数.

(8) $(\texttt{ccl}\ (X)\ M)\{X \leftarrow N\} = (\texttt{ccl}\ X\ M)$

$(\texttt{ccl}\ (Y)\ M)\{X \leftarrow N\} = (\texttt{ccl}\ (Y)\ M\{X \leftarrow N\})$

if $Y \neq X$ かつ $Y \notin \mathrm{FV}(N)$.

$(\texttt{ccl}\ (Y)\ M)\{X \leftarrow N\} = (\texttt{ccl}\ (Z)\ M\{Y \leftarrow Z\}\{X \leftarrow N\})$

if $Y \neq X$ かつ $Y \in \mathrm{FV}(N)$　where Z は新たな変数.

(9) $(\texttt{cnt}\ (X)\ M)\{X \leftarrow N\} = (\texttt{cnt}\ X\ M)$

$(\texttt{cnt}\ (Y)\ M)\{X \leftarrow N\} = (\texttt{cnt}\ (Y)\ M\{X \leftarrow N\})$

if $Y \neq X$ かつ $Y \notin \mathrm{FV}(N)$.

$(\texttt{cnt}\ (Y)\ M)\{X \leftarrow N\} = (\texttt{cnt}\ (Z)\ M\{Y \leftarrow Z\}\{X \leftarrow N\})$

if $Y \neq X$ かつ $Y \in \mathrm{FV}(N)$　where Z は新たな変数.

(10) $(\texttt{lam}\ (X_1\ \ldots\ X_n)\ M)\{X \leftarrow N\} = (\texttt{lam}\ (X_1\ \ldots\ X_n)\ M)$

if ある X_i に対して $X_i = X$.

$(\texttt{lam}\ (X_1\ \ldots\ X_n)\ M)\{X \leftarrow N\} = (\texttt{lam}\ (X_1\ \ldots\ X_n)\ M\{X \leftarrow N\})$

if すべての X_i に対して $X_i \neq X$ かつ $X_i \notin \mathrm{FV}(N)$.

$(\texttt{lam}\ (X_1\ \ldots\ X_n)\ M)\{X \leftarrow N\}$

$= (\texttt{lam}\ (Z_1\ \ldots\ Z_n)\ M\{X_1 \leftarrow Z_1\}\ldots\{X_n \leftarrow Z_n\}\{X \leftarrow N\})$

if すべて X_i に対して $X_i \neq X$　かつ　ある X_i に対して $X_i \in \mathrm{FV}(N)$

where

$$Z_i = \begin{cases} X_i & \text{if } X_i \notin \mathrm{FV}(N) \\ \text{新たな変数} & \text{if } X_i \in \mathrm{FV}(N) \end{cases}$$

▮同時代入

4.5

ラムダ式の手続き適用を簡約するときに，次のような同時代入（**simultaneous substitution**）が必要になります．ただし，以下の X_1〜X_n は互いに異なる変数とします．

$$M\{X_1 \leftarrow N_1, \ldots, X_n \leftarrow N_n\} \quad \text{または} \quad M\{X_j \leftarrow N_j \mid 1 \leq j \leq n\}$$

これは M の中の X_1〜X_n を，それぞれ，N_1〜N_n に同時に置き換えることを示しています．当然，このときにも名前の衝突は避けなければなりません．そのことに加えて，

- N_1〜N_n の中の X_1〜X_n をうっかり置き換えてしまうこと（以下「うっかり代入」）

があってはいけません．このため同時代入は，普通の代入の逐次的な列に安直に置き換えることができません．形式的には，普通の代入を用いて次のように定義できます．

$$M\{X_1 \leftarrow N_1, \ldots, X_n \leftarrow N_n\} = M\{X_1 \leftarrow Z_1\}\cdots\{X_N \leftarrow Z_n\}\{Z_1 \leftarrow N_1\}\cdots\{Z_n \leftarrow N_n\}$$

ただし，Z_1〜Z_n は M や N_i の中で使われていない新たな変数です．この右辺は，M の中の X_1〜X_n をまったく新たな変数 Z_1〜Z_n に置き換えた上で，新たな変数を N_1〜N_n に逐次的に置き換えるといったことをしています．こうすることによって，うっかり代入を避けています．

▮簡約規則

4.6

M 上の簡約規則 $M_1 \xrightarrow[\text{CBV}]{} M_2$ を次のように定義します．矢印 $\xrightarrow[\text{CBV}]{}$ の左側の式 M_1 を各規則の 左辺 と呼び，右側の式 M_2 を 右辺 と呼びます．矢印自身は左辺を右辺に記号的に書き換えることを示しています．簡約規則は，次に示すように，基本簡約と継続簡約の 2 つに分類されます．(β) や (δ) などはそれぞれの規則に付けた名前です．

(I) 基本簡約

以下の基本簡約はいずれも次の形式をしています．

$$E_1[C] \xrightarrow[\text{CBV}]{} E_1[D]$$

これは次のことを示しています．

- E_1 は左辺 M_1 の CBV-評価文脈で，角カッコ内の C は左辺 M_1 の CBV-簡約基です．
- 矢印 $\xrightarrow[\text{CBV}]{}$ は C を D に書き換えることを示しています．つまり，CBV-簡約基 C だけを書き換えて，評価文脈 E_1 は変更しないことを示しています．$E_1[D]$ は右辺 M_2 のことです．

書き換えたあとの $E_1[D]$ の E_1 や D は，一般に，右辺 M_2 の CBV-評価文脈や CBV-簡約基ではありません．それらは書き換えたあとに求め直す必要があります．

(β) $E_1[((\texttt{lam}\ (X_1\ \ldots\ X_n)\ N)\ V_1\ \ldots\ V_n)] \xrightarrow[\text{CBV}]{} E_1[N\{X_1 \leftarrow V_1, \ldots, X_n \leftarrow V_n\}]$

(δ) $E_1[(O\ V_1\ \ldots\ V_n)] \xrightarrow[\text{CBV}]{} E_1[\delta(O, V_1, \ldots, V_n)]$

⋆ ここで，$\delta(O, V_1, \ldots, V_n)$ は，実引数 $V_1, \ldots, V_n$ に標準手続き O を適用した値（値式）を示しています．具体的な内容は必要ないので，抽象的に示しています．具体例を示すときには，それぞれの標準手続きに応じて具体的な値を示します．

(if-f) $E_1[(\texttt{if}\ \texttt{\#f}\ N_1\ N_2)] \xrightarrow[\text{CBV}]{} E_1[N_2]$

(if-t) $E_1[(\texttt{if}\ V\ N_1\ N_2)] \xrightarrow[\text{CBV}]{} E_1[N_1]$ （ただし，$V \neq \texttt{\#f}$）

(let^1) $E_1[(\texttt{let}^1\ (X\ V)\ N)] \xrightarrow[\text{CBV}]{} E_1[N\{X \leftarrow V\}]$

(rec^1) $E_1[(\texttt{rec}^1\ X\ N)] \xrightarrow[\text{CBV}]{} E_1[N\{X \leftarrow (\texttt{rec}^1\ X\ N)\}]$

⋆ $\texttt{rec}^1$ 式は N を繰り返し実行するための構文です．式変形後の X に $\texttt{rec}^1$ 式自身を代入しているのは，次の繰り返しに備えるためです．計算が進んでいって，再びこの $\texttt{rec}^1$ 式に遭遇したら N を再び実行することになります．

(II) 継続簡約

継続に関する簡約規則は，CBV-評価文脈を巻き込みながら簡約するという点で，基本簡約とはかなり違っています．

(ccl) $E_1[(\texttt{ccl}\ (X)\ N)] \xrightarrow[\text{CBV}]{} E_1[N\{X \leftarrow (\texttt{cnt}\ (Z)\ E_1[Z])\}]$

⋆ ここで，E_1 は M_1 の CVB-評価文脈で，ccl 式は M_1 の CBV-簡約基です．Z は新しい変数です．この規則は，M_1 の CBV-評価文脈 E_1 を使って継続（cnt 式）を生成し，それを X に代入（束縛）して ccl 式の本体 N を簡約の俎上に載せます．ただし，N をすぐに簡約するかどうかは CBV-評価文脈 E_1 に依存します．基本簡約のときと同様に，一般に，E_1 や N は簡約後の M_2 の CBV-評価文脈や CBV-簡約基ではありません．それらは別途求める必要があります．

(cnt) $E_1[((\texttt{cnt}\ (X)\ N)\ V)] \xrightarrow[\text{CBV}]{} N\{X \leftarrow V\}$

⋆ E_1 は M_1 の CBV-評価文脈で，$((\texttt{cnt}\ (X)\ N)\ V)$ は M_1 の CBV-簡約基です．この規則は M_1 の CBV-評価文脈 E_1 をすべて捨てて，cnt 式が表す手続きを実引数の V に適用することを示しています．

(III) 反射推移閉包

さらに，$\xrightarrow[\text{CBV}]{}$ の反射推移閉包 $\xrightarrow[\text{CBV}]{}\!\!\!\!\rightarrow$ を次の規則によって定義します．

(ref) $M \xrightarrow[\text{CBV}]{}\!\!\!\!\rightarrow M$

(one) $M_1 \xrightarrow[\text{CBV}]{} M_2$ ならば $M_1 \xrightarrow[\text{CBV}]{}\!\!\!\!\rightarrow M_2$

(trans) $M_1 \xrightarrow[\text{CBV}]{}\!\!\!\!\rightarrow M_2$ かつ $M_2 \xrightarrow[\text{CBV}]{}\!\!\!\!\rightarrow M_3$ ならば，$M_1 \xrightarrow[\text{CBV}]{}\!\!\!\!\rightarrow M_3$

$M_1 \xrightarrow[\text{CBV}]{}\!\!\!\!\rightarrow M_2$ は 0 回以上の簡約によって M_1 を M_2 に変形できることを表しています．

例

4.7

次の $\texttt{rec}^1$ 式は $n!$ を計算します．F は $\texttt{rec}^1$ 式に付けた便宜的な名前です．$\texttt{rec}^1$ 式が長いので，代わりに F を使います．

F = (rec^1 fact (lam (n) (if (< n 2) 1 (* n (fact (- n 1))))))

簡約規則の (rec^1) より，F に対して，

F $\xrightarrow[\text{CBV}]{}$ (lam (n) (if (< n 2) 1 (* n (F (- n 1)))))

が成り立ちます．以下ではこれを簡約規則として使用します．

(F 3) を計算してみます．少し丁寧に，簡約規則を適用する際の CBV-評価文脈と CBV-簡約基を示した上で簡約を行うことにします．等号のうしろに書いた式が CBV-評価文脈（黒字部分）と CBV-簡約基（赤字部分）を示しています．

```
        (F 3)
=       (F 3)
-CBV->  ((lam (n) (if (< n 2) 1 (* n (F (- n 1))))) 3)
=       ((lam (n) (if (< n 2) 1 (* n (F (- n 1))))) 3)
-CBV->  (if (< 3 2) (* 3 (F (- 3 1))))
=       (if (< 3 2) (* 3 (F (- 3 1))))
-CBV->  (if #f (* 3 (F (- 3 1))))
=       (if #f (* 3 (F (- 3 1))))
-CBV->  (* 3 (F (- 3 1)))
=       (* 3 (F (- 3 1)))
-CBV->  (* 3 ((lam (n) (if (< n 2) 1 (* n (F (- n 1))))) (- 3 1)))
=       (* 3 ((lam (n) (if (< n 2) 1 (* n (F (- n 1))))) (- 3 1)))
-CBV->  (* 3 ((lam (n) (if (< n 2) 1 (* n (F (- n 1))))) 2))
=       (* 3 ((lam (n) (if (< n 2) 1 (* n (F (- n 1))))) 2))
-CBV->  (* 3 (if (< 2 2) 1 (* 2 (F (- 2 1)))))
=       (* 3 (if (< 2 2) 1 (* 2 (F (- 2 1)))))
-CBV->  (* 3 (if #f 1 (* 2 (F (- 2 1)))))
```

```
=          (* 3 (if #f 1 (* 2 (F (- 2 1)))))
⟶CBV       (* 3 (* 2 (F (- 2 1))))
=          (* 3 (* 2 (F (- 2 1))))
⟶CBV       (* 3 (* 2 ((lam (n) (if (< n 2) 1 (* n (F (- n 1))))) (- 2 1))))
=          (* 3 (* 2 ((lam (n) (if (< n 2) 1 (* n (F (- n 1))))) (- 2 1))))
⟶CBV       (* 3 (* 2 ((lam (n) (if (< n 2) 1 (* n (F (- n 1))))) 1)))
=          (* 3 (* 2 ((lam (n) (if (< n 2) 1 (* n (F (- n 1))))) 1)))
⟶CBV       (* 3 (* 2 (if (< 1 2) 1 (* 1 (F (- 1 1))))))
=          (* 3 (* 2 (if (< 1 2) 1 (* 1 (F (- 1 1))))))
⟶CBV       (* 3 (* 2 (if #t 1 (* 1 (F (- 1 1))))))
=          (* 3 (* 2 (if #t 1 (* 1 (F (- 1 1))))))
⟶CBV       (* 3 (* 2 1))
=          (* 3 (* 2 1))
⟶CBV       (* 3 2)
=          (* 3 2)
⟶CBV       6
```

例 4.8

まずは継続を捨ててしまう例を示します．where 節は記述の複雑化を避けるための便法です．

```
           (+ 1 (ccl (k) 2))
=          (+ 1 (ccl (k) 2))
⟶CBV       (+ 1 2[k ← K])        where K = (cnt (z) (+ 1 z))
=          (+ 1 2)
⟶CBV       3
```

ちなみに，これの Guile(2.2.4) による実行例も示します．3 行目以降はトレースした結果です．

```
scheme@(guile-user)> (+ 1 (call/cc (lambda (k) 2)))⏎
$1 = 3
scheme@(guile-user)> ,trace (+ 1 (call/cc (lambda (k) 2)))⏎
trace: |  (_ #<procedure 5599cba8f6c0 at <unknown port>:10:21 (k)>)
trace: |  (_ #<continuation 5599cb75cb40>)
trace: |  2
trace: 3
```

例 4.9

次に継続を使用する例を示します．

```
           (+ 1 (ccl (k) (* 2 (+ 3 (k 4)))))
=          (+ 1 (ccl (k) (* 2 (+ 3 (k 4)))))
⟶CBV       (+ 1 (* 2 (+ 3 (K 4))))       where K = (cnt (z) (+ 1 z))
=          (+ 1 (* 2 (+ 3 (K 4))))       where K = (cnt (z) (+ 1 z))
```

$\xrightarrow[\text{CBV}]{}$ (+ 1 4)

$=$ (+ 1 4)

$\xrightarrow[\text{CBV}]{}$ 5

これの実行例も示します．3 行目以降はトレースした結果です．

```
scheme@(guile-user)> (+ 1 (call/cc (lambda (k) (* 2 (+ 3 (k 4))))))
$1 = 5
scheme@(guile-user)> ,trace (+ 1 (call/cc … 同上 …))
trace: |  (_ #<procedure 5599cba8a4a0 at <unknown port>:2:21 (k)>)
trace: |  (_ #<continuation 5599cbcc6f80>)
trace: |  |  (_ 4)
trace: |  |  5
```

例

4.10

Dybvig の本 [Dyv09, 3.3 節] に次のような具体例が示されています．

```
(let ((x (call/cc (lambda (k) k))))
  (x (lambda (ignore) "hi")))
```

これを Scheme-0 に翻訳すると次のようになります．

```
(let¹ (x (ccl (k) k)) (x (lam (p) "hi")))}
```

ただし，ignore を p に変えています．これを計算してみます．

(let¹ (x (ccl (k) k)) (x (lam (p) "hi")))

$=$ (let¹ (x (ccl (k) k)) (x (lam (p) "hi")))

$\xrightarrow[\text{CBV}]{}$ (let¹ (x K) (x (lam (p) "hi")))
where K = (cnt (z) (let¹ (x z) (x (lam (p) "hi"))))

$=$ (let¹ (x K) (x (lam (p) "hi")))
where K = (cnt (z) (let¹ (x z) (x (lam (p) "hi"))))

$\xrightarrow[\text{CBV}]{}$ (K (lam (p) "hi"))
where K = (cnt (z) (let¹ (x z) (x (lam (p) "hi"))))

$=$ (K (lam (p) "hi"))
where K = (cnt (z) (let¹ (x z) (x (lam (p) "hi"))))

$\xrightarrow[\text{CBV}]{}$ (let¹ (x (lam (p) "hi")) (x (lam (p) "hi")))

$=$ (let¹ (x (lam (p) "hi")) (x (lam (p) "hi")))

$\xrightarrow[\text{CBV}]{}$ ((lam (p) "hi") (lam (p) "hi"))

$=$ ((lam (p) "hi") (lam (p) "hi"))

$\xrightarrow[\text{CBV}]{}$ "hi"

これの実行例も示します．3 行目以降はトレースした結果です．

```
scheme@(guile-user)> (let ((x (call/cc (lambda (k) k)))) (x (lambda (p) "hi")))
$1 = "hi"
scheme@(guile-user)> ,trace (let … 同上 …)
```

```
trace: |  (_ #<procedure 5599cba90b08 at <unknown port>:12:25 (k)>)
trace: |  (_ #<continuation 5599cbdd5700>)
trace: |  #<continuation 5599cbdd5700>
trace: (_ #<procedure 5599cba90b18 at <unknown port>:12:46 (p)>)
trace: (_ #<procedure 5599cba90b18 at <unknown port>:12:46 (p)>)
trace: "hi"
```

例 4.11

上で示した具体例のあと，Dybvig の本 [Dyv09, 3.3 節] に次のような記述があります．

- The following variation of the example above is probably the most confusing Scheme program of its size; it might be easy to guess what it returns, but it takes some thought to figure out why.

```
(((call/cc (lambda (k) k)) (lambda (x) x)) "HEY!")
```

筆者は何を計算するのかさっぱり分かりません．そこで，Scheme-0 に翻訳して計算してみます．

```
      (((ccl (k) k) (lam (x) x)) "HEY!")
=     (((ccl (k) k) (lam (x) x)) "HEY!")
→CBV  ((K (lam (x) x)) "HEY!")    where K = (cnt (z) ((z (lam (x) x)) "HEY!"))
=     ((K (lam (x) x)) "HEY!")    where K = (cnt (z) ((z (lam (x) x)) "HEY!"))
→CBV  (((lam (x) x) (lam (x) x)) "HEY!")
=     (((lam (x) x) (lam (x) x)) "HEY!")
→CBV  ((lam (x) x) "HEY!")
=     ((lam (x) x) "HEY!")
→CBV  "HEY!"
```

これの実行例も示します．3 行目以降はトレースした結果です．

```
scheme@(guile-user)> (((call/cc (lambda (k) k)) (lambda (x) x)) "HEY!")⏎
$1 = "HEY!"
scheme@(guile-user)> ,trace (((call/cc (lambda (k) k)) (lambda (x) x)) "HEY!")⏎
trace: |  (_ #<procedure 5599cba91fa8 at <unknown port>:14:18 (k)>)
trace: |  (_ #<continuation 5599cb964240>)
trace: |  #<continuation 5599cb964240>
trace: |  (_ #<procedure 5599cba91fb8 at <unknown port>:14:34 (x)>)
trace: |  |  (_ #<procedure 5599cba91fb8 at <unknown port>:14:34 (x)>)
trace: |  |  #<procedure 5599cba91fb8 at <unknown port>:14:34 (x)>
trace: |  (_ "HEY!")
trace: |  "HEY!"
```

5 Yin-Yang Puzzle

yin-yang puzzle（陰陽パズル）と呼ばれている Scheme のプログラムがあります[6)]．

[6)] Wikipedia: call-with-current-continuation

```
(let* ((yin ((lambda (cc) (display #\@) cc) (call/cc (lambda (c) c))))
       (yang ((lambda (cc) (display #\*) cc) (call/cc (lambda (c) c)))))
  (yin yang))
```

あるサイト[7]によれば，このプログラムは

```
@*@**@***@****@*****@****** ...
```

と表示するのだそうです．そこで，問題は，なぜそうなるのかを説明すること，つまり，プログラムの動作を説明することです．小股意味論の応用例として，このプログラムの振る舞いを解析してみます．

let*式はネストしたlet[1]式で記述できます．さらに，displayは外部環境（標準出力）のみを変化させ，その副作用はプログラムの計算過程そのものには影響しません．従って，式の簡約と標準出力のモニタリングを同時に行うことにすれば，Scheme-0でもなんとかこの式の振る舞いを解析できます．とりあえずScheme-0に翻訳します．

```
(let[1] (yin ((lam (cc) (display #\@) cc) (ccl (c) c)))
  (let[1] (yang ((lam (cc) (display #\*) cc) (ccl (c) c)))
    (yin yang)))
```

Scheme-0ではlet[1]式の束縛はカッコが2重になっていないことに注意して下さい．それから，Scheme-0は逐次実行の構文がないので，これは厳密に言えばScheme-0のプログラムではありません．この点についてはすぐあとで補足します．

このまま簡約していくには式が長すぎます．そこで，次のような省略記号を用意します．

AT = (lam (cc) (display #\@) cc)

STAR = (lam (cc) (display #*) cc)

CCL = (ccl (c) c)

これらを使うと上の式は次のようになります．

```
1 (let[1] (yin (AT CCL)) (let[1] (yang (STAR CCL)) (yin yang)))
```

ATとSTARのラムダ式は逐次実行を含んでいるので，厳密に言えばScheme-0では簡約できません．でも，display手続きを無視すれば，あらゆる式Mに対して

$$\texttt{(AT}\ M\texttt{)} \xrightarrow[\text{CBV}]{} M \quad \text{および} \quad \texttt{(STAR}\ M\texttt{)} \xrightarrow[\text{CBV}]{} M$$

が成り立つし，display手続きを無視してもプログラムの計算過程に影響しません．そこで以下では，これらを簡約規則として捉え，これらの簡約の途中でそれぞれの文字（#\@と#*）が表示されると考えることにします．さらに，文字が表示されることを強調するために，

$$\texttt{(AT}\ M\texttt{)} \xrightarrow[\text{CBV}]{\texttt{\#\textbackslash@}} M \quad \text{および} \quad \texttt{(STAR}\ M\texttt{)} \xrightarrow[\text{CBV}]{\texttt{\#\textbackslash*}} M$$

と書くことにします．それから，CCLについては次が成り立ちます．

$$E[\texttt{CCL}] \xrightarrow[\text{CBV}]{} E[\texttt{(cnt (}Z\texttt{) }E[Z]\texttt{)}]$$

ここで，Eは各時点のCBV-評価文脈を表し，CCLはその時点のCBV-簡約基になります．Zは新たな変数を表します．これも簡約規則として活用します．

[7] Understanding the yin-yang puzzle.

試しに，少し計算してみることにします．以下の四角は標準出力のモニタリングを示します．それから，これまでの具体例とは違って，簡約後の式と次の簡約への CBV-評価文脈（黒字）・CBV-簡約基（赤字）の表示を一緒にします．これまでの具体例に比べて少し分かりにくいと思いますが，簡約前後の式のどこが変化しているかを見るようにして下さい．例えば，以下の最初の簡約では CCL が K_1 に書き換わっています．

$(\mathtt{let}^1\ (\mathtt{yin}\ (\mathtt{AT}\ \mathtt{CCL}))\ (\mathtt{let}^1\ (\mathtt{yang}\ (\mathtt{STAR}\ \mathtt{CCL}))\ (\mathtt{yin}\ \mathtt{yang})))$

$\xrightarrow[\text{CBV}]{}$ $(\mathtt{let}^1\ (\mathtt{yin}\ (\mathtt{AT}\ \mathtt{K}_1))\ (\mathtt{let}^1\ (\mathtt{yang}\ (\mathtt{STAR}\ \mathtt{CCL}))\ (\mathtt{yin}\ \mathtt{yang})))$

where

$\mathtt{K}_1 = (\mathtt{cnt}\ (\mathtt{z})\ (\mathtt{let}^1\ (\mathtt{yin}\ (\mathtt{AT}\ \mathtt{z}))\ (\mathtt{let}^1\ (\mathtt{yang}\ (\mathtt{STAR}\ \mathtt{CCL}))\ (\mathtt{yin}\ \mathtt{yang}))))$

$\xrightarrow[\text{CBV}]{\#\backslash @}$ $(\mathtt{let}^1\ (\mathtt{yin}\ \mathtt{K}_1)\ (\mathtt{let}^1\ (\mathtt{yang}\ (\mathtt{STAR}\ \mathtt{CCL}))\ (\mathtt{yin}\ \mathtt{yang})))$

where

$\mathtt{K}_1 = (\mathtt{cnt}\ (\mathtt{z})\ (\mathtt{let}^1\ (\mathtt{yin}\ (\mathtt{AT}\ \mathtt{z}))\ (\mathtt{let}^1\ (\mathtt{yang}\ (\mathtt{STAR}\ \mathtt{CCL}))\ (\mathtt{yin}\ \mathtt{yang}))))$

@

$\xrightarrow[\text{CBV}]{}$ $(\mathtt{let}^1\ (\mathtt{yang}\ (\mathtt{STAR}\ \mathtt{CCL}))\ (\mathtt{K}_1\ \mathtt{yang}))$

where

$\mathtt{K}_1 = (\mathtt{cnt}\ (\mathtt{z})\ (\mathtt{let}^1\ (\mathtt{yin}\ (\mathtt{AT}\ \mathtt{z}))\ (\mathtt{let}^1\ (\mathtt{yang}\ (\mathtt{STAR}\ \mathtt{CCL}))\ (\mathtt{yin}\ \mathtt{yang}))))$

@

$\xrightarrow[\text{CBV}]{}$ $(\mathtt{let}^1\ (\mathtt{yang}\ (\mathtt{STAR}\ \mathtt{K}_2))\ (\mathtt{K}_1\ \mathtt{yang}))$

where

$\mathtt{K}_1 = (\mathtt{cnt}\ (\mathtt{z})\ (\mathtt{let}^1\ (\mathtt{yin}\ (\mathtt{AT}\ \mathtt{z}))\ (\mathtt{let}^1\ (\mathtt{yang}\ (\mathtt{STAR}\ \mathtt{CCL}))\ (\mathtt{yin}\ \mathtt{yang}))))$

$\mathtt{K}_2 = (\mathtt{cnt}\ (\mathtt{z})\ (\mathtt{let}^1\ (\mathtt{yang}\ (\mathtt{STAR}\ \mathtt{z}))\ (\mathtt{K}_1\ \mathtt{yang})))$

@

$\xrightarrow[\text{CBV}]{\#\backslash *}$ $(\mathtt{let}^1\ (\mathtt{yang}\ \mathtt{K}_2)\ (\mathtt{K}_1\ \mathtt{yang}))$

where

$\mathtt{K}_1 = (\mathtt{cnt}\ (\mathtt{z})\ (\mathtt{let}^1\ (\mathtt{yin}\ (\mathtt{AT}\ \mathtt{z}))\ (\mathtt{let}^1\ (\mathtt{yang}\ (\mathtt{STAR}\ \mathtt{CCL}))\ (\mathtt{yin}\ \mathtt{yang}))))$

$\mathtt{K}_2 = (\mathtt{cnt}\ (\mathtt{z})\ (\mathtt{let}^1\ (\mathtt{yang}\ (\mathtt{STAR}\ \mathtt{z}))\ (\mathtt{K}_1\ \mathtt{yang})))$

@*

$\xrightarrow[\text{CBV}]{}$ $(\mathtt{K}_1\ \mathtt{K}_2)$

where

$\mathtt{K}_1 = (\mathtt{cnt}\ (\mathtt{z})\ (\mathtt{let}^1\ (\mathtt{yin}\ (\mathtt{AT}\ \mathtt{z}))\ (\mathtt{let}^1\ (\mathtt{yang}\ (\mathtt{STAR}\ \mathtt{CCL}))\ (\mathtt{yin}\ \mathtt{yang}))))$

$\mathtt{K}_2 = (\mathtt{cnt}\ (\mathtt{z})\ (\mathtt{let}^1\ (\mathtt{yang}\ (\mathtt{STAR}\ \mathtt{z}))\ (\mathtt{K}_1\ \mathtt{yang})))$

@*

$\xrightarrow[\text{CBV}]{}$ $(\text{let}^1\ (\text{yin}\ (\text{AT}\ K_2))\ (\text{let}^1\ (\text{yang}\ (\text{STAR CCL}))\ (\text{yin yang})))$

where

$K_1 = (\text{cnt}\ (z)\ (\text{let}^1\ (\text{yin}\ (\text{AT}\ z))\ (\text{let}^1\ (\text{yang}\ (\text{STAR CCL}))\ (\text{yin yang}))))$

$K_2 = (\text{cnt}\ (z)\ (\text{let}^1\ (\text{yang}\ (\text{STAR}\ z))\ (K_1\ \text{yang})))$

`@*`

$\xrightarrow[\text{CBV}]{\#\backslash @}$ $(\text{let}^1\ (\text{yin}\ K_2)\ (\text{let}^1\ (\text{yang}\ (\text{STAR CCL}))\ (\text{yin yang})))$

where

$K_1 = (\text{cnt}\ (z)\ (\text{let}^1\ (\text{yin}\ (\text{AT}\ z))\ (\text{let}^1\ (\text{yang}\ (\text{STAR CCL}))\ (\text{yin yang}))))$

$K_2 = (\text{cnt}\ (z)\ (\text{let}^1\ (\text{yang}\ (\text{STAR}\ z))\ (K_1\ \text{yang})))$

`@*@`

$\xrightarrow[\text{CBV}]{}$ $(\text{let}^1\ (\text{yang}\ (\text{STAR CCL}))\ (K_2\ \text{yang}))$

where

$K_1 = (\text{cnt}\ (z)\ (\text{let}^1\ (\text{yin}\ (\text{AT}\ z))\ (\text{let}^1\ (\text{yang}\ (\text{STAR CCL}))\ (\text{yin yang}))))$

$K_2 = (\text{cnt}\ (z)\ (\text{let}^1\ (\text{yang}\ (\text{STAR}\ z))\ (K_1\ \text{yang})))$

`@*@`

$\xrightarrow[\text{CBV}]{}$ $(\text{let}^1\ (\text{yang}\ (\text{STAR}\ K_3))\ (K_2\ \text{yang}))$

where

$K_1 = (\text{cnt}\ (z)\ (\text{let}^1\ (\text{yin}\ (\text{AT}\ z))\ (\text{let}^1\ (\text{yang}\ (\text{STAR CCL}))\ (\text{yin yang}))))$

$K_2 = (\text{cnt}\ (z)\ (\text{let}^1\ (\text{yang}\ (\text{STAR}\ z))\ (K_1\ \text{yang})))$

$K_3 = (\text{cnt}\ (z)\ (\text{let}^1\ (\text{yang}\ (\text{STAR}\ z))\ (K_2\ \text{yang})))$

`@*@`

$\xrightarrow[\text{CBV}]{\#\backslash *}$ $(\text{let}^1\ (\text{yang}\ K_3)\ (K_2\ \text{yang}))$

where

$K_1 = (\text{cnt}\ (z)\ (\text{let}^1\ (\text{yin}\ (\text{AT}\ z))\ (\text{let}^1\ (\text{yang}\ (\text{STAR CCL}))\ (\text{yin yang}))))$

$K_2 = (\text{cnt}\ (z)\ (\text{let}^1\ (\text{yang}\ (\text{STAR}\ z))\ (K_1\ \text{yang})))$

$K_3 = (\text{cnt}\ (z)\ (\text{let}^1\ (\text{yang}\ (\text{STAR}\ z))\ (K_2\ \text{yang})))$

`@*@*`

$\xrightarrow[\text{CBV}]{}$ $(K_2\ K_3)$

where

$K_1 = (\text{cnt}\ (z)\ (\text{let}^1\ (\text{yin}\ (\text{AT}\ z))\ (\text{let}^1\ (\text{yang}\ (\text{STAR CCL}))\ (\text{yin yang}))))$

$K_2 = (\text{cnt}\ (z)\ (\text{let}^1\ (\text{yang}\ (\text{STAR}\ z))\ (K_1\ \text{yang})))$

$K_3 = (\text{cnt}\ (z)\ (\text{let}^1\ (\text{yang}\ (\text{STAR}\ z))\ (K_2\ \text{yang})))$

`@*@*`

$\xrightarrow[\text{CBV}]{}$ $(\text{let}^1\ (\text{yang}\ (\text{STAR}\ K_3))\ (K_1\ \text{yang}))$

where

$K_1 = (\text{cnt}\ (\text{z})\ (\text{let}^1\ (\text{yin}\ (\text{AT}\ \text{z}))\ (\text{let}^1\ (\text{yang}\ (\text{STAR CCL}))\ (\text{yin yang}))))$

$K_2 = (\text{cnt}\ (\text{z})\ (\text{let}^1\ (\text{yang}\ (\text{STAR z}))\ (K_1\ \text{yang})))$

$K_3 = (\text{cnt}\ (\text{z})\ (\text{let}^1\ (\text{yang}\ (\text{STAR z}))\ (K_2\ \text{yang})))$

@*@*

$\xrightarrow[\text{CBV}]{\#\backslash *}$ $(\text{let}^1\ (\text{yang}\ K_3)\ (K_1\ \text{yang}))$

where

$K_1 = (\text{cnt}\ (\text{z})\ (\text{let}^1\ (\text{yin}\ (\text{AT}\ \text{z}))\ (\text{let}^1\ (\text{yang}\ (\text{STAR CCL}))\ (\text{yin yang}))))$

$K_2 = (\text{cnt}\ (\text{z})\ (\text{let}^1\ (\text{yang}\ (\text{STAR z}))\ (K_1\ \text{yang})))$

$K_3 = (\text{cnt}\ (\text{z})\ (\text{let}^1\ (\text{yang}\ (\text{STAR z}))\ (K_2\ \text{yang})))$

@*@**

$\xrightarrow[\text{CBV}]{}$ $(K_1\ K_3)$

where

$K_1 = (\text{cnt}\ (\text{z})\ (\text{let}^1\ (\text{yin}\ (\text{AT}\ \text{z}))\ (\text{let}^1\ (\text{yang}\ (\text{STAR CCL}))\ (\text{yin yang}))))$

$K_2 = (\text{cnt}\ (\text{z})\ (\text{let}^1\ (\text{yang}\ (\text{STAR z}))\ (K_1\ \text{yang})))$

$K_3 = (\text{cnt}\ (\text{z})\ (\text{let}^1\ (\text{yang}\ (\text{STAR z}))\ (K_2\ \text{yang})))$

@*@**

$\xrightarrow[\text{CBV}]{}$ $(\text{let}^1\ (\text{yin}\ (\text{AT}\ K_3))\ (\text{let}^1\ (\text{yang}\ (\text{STAR CCL}))\ (\text{yin yang})))$

where

$K_1 = (\text{cnt}\ (\text{z})\ (\text{let}^1\ (\text{yin}\ (\text{AT}\ \text{z}))\ (\text{let}^1\ (\text{yang}\ (\text{STAR CCL}))\ (\text{yin yang}))))$

$K_2 = (\text{cnt}\ (\text{z})\ (\text{let}^1\ (\text{yang}\ (\text{STAR z}))\ (K_1\ \text{yang})))$

$K_3 = (\text{cnt}\ (\text{z})\ (\text{let}^1\ (\text{yang}\ (\text{STAR z}))\ (K_2\ \text{yang})))$

@*@**

$\xrightarrow[\text{CBV}]{\#\backslash @}$ $(\text{let}^1\ (\text{yin}\ K_3)\ (\text{let}^1\ (\text{yang}\ (\text{STAR CCL}))\ (\text{yin yang})))$

where

$K_1 = (\text{cnt}\ (\text{z})\ (\text{let}^1\ (\text{yin}\ (\text{AT}\ \text{z}))\ (\text{let}^1\ (\text{yang}\ (\text{STAR CCL}))\ (\text{yin yang}))))$

$K_2 = (\text{cnt}\ (\text{z})\ (\text{let}^1\ (\text{yang}\ (\text{STAR z}))\ (K_1\ \text{yang})))$

$K_3 = (\text{cnt}\ (\text{z})\ (\text{let}^1\ (\text{yang}\ (\text{STAR z}))\ (K_2\ \text{yang})))$

@*@**@

$\xrightarrow[\text{CBV}]{}$ $(\text{let}^1\ (\text{yang}\ (\text{STAR CCL}))\ (K_3\ \text{yang}))$

where

$K_1 = (\text{cnt}\ (\text{z})\ (\text{let}^1\ (\text{yin}\ (\text{AT}\ \text{z}))\ (\text{let}^1\ (\text{yang}\ (\text{STAR CCL}))\ (\text{yin yang}))))$

$K_2 = (\text{cnt}\ (\text{z})\ (\text{let}^1\ (\text{yang}\ (\text{STAR z}))\ (K_1\ \text{yang})))$

$K_3 = (\text{cnt}\ (\text{z})\ (\text{let}^1\ (\text{yang}\ (\text{STAR z}))\ (K_2\ \text{yang})))$

@*@**@

$\xrightarrow[\text{CBV}]{}$ (let^1 (yang (STAR K$_4$)) (K$_3$ yang))

where

K$_1$ = (cnt (z) (let^1 (yin (AT z)) (let^1 (yang (STAR CCL)) (yin yang))))

K$_2$ = (cnt (z) (let^1 (yang (STAR z)) (K$_1$ yang)))

K$_3$ = (cnt (z) (let^1 (yang (STAR z)) (K$_2$ yang)))

K$_4$ = (cnt (z) (let^1 (yang (STAR z)) (K$_3$ yang)))

```
@*@**@
```

$\xrightarrow[\text{CBV}]{}$ …… 疲れました．でも，計算のパターンは見えてきました．

▌命題（Yin-Yang Puzzle の解析結果）

5.1

AT，STAR，CCL，Yin-Yang を次のようにおきます．Yin-Yang は元々のプログラムです．

AT = (lam (cc) (display #\@) cc)

STAR = (lam (cc) (display #*) cc)

CCL = (ccl (c) c)

Yin-Yang = (let^1 (yin (AT CCL)) (let^1 (yang (STAR CCL)) (yin yang)))

ただし，Scheme-0 の計算過程（簡約過程）を考えるときには，display 手続きの呼び出しを無視します．これを無視しても計算過程に影響を与えることはありません．

さらに，上の計算例の中で示したように

K$_1$ = (cnt (z) (let^1 (yin (AT z)) (let^1 (yang (STAR CCL)) (yin yang))))

と定め，$i \geq 2$ に対して，

K$_i$ = (cnt (z) (let^1 (yang (STAR z)) (K$_{i-1}$ yang)))

と定めます．このとき，次が成り立ちます．

(1) Yin-Yang

$\xrightarrow[\text{CBV}]{}$ (let^1 (yin (AT K$_1$)) (let^1 (yang (STAR CCL)) (yin yang)))

ただし，この計算過程の中では何も表示されません．

(2) $i \geq 2$ に対して

(let^1 (yin (AT K$_{i-1}$)) (let^1 (yang (STAR CCL)) (yin yang)))

$\xrightarrow[\text{CBV}]{}$ (let^1 (yang (STAR K$_i$)) (K$_{i-1}$ yang))

が成り立ちます．さらに，この計算過程の中で #\@ が表示されます．

(3) $1 < j < i$ に対して

(let^1 (yang (STAR K$_i$)) (K$_j$ yang))

$\xrightarrow[\text{CBV}]{}$ (let^1 (yang (STAR K$_i$)) (K$_{j-1}$ yang))

が成り立ちます．さらに，この計算過程の中で #* が表示されます．

(4) $i \geq 2$ に対して

(let^1 (yang (STAR K$_i$)) (K$_1$ yang))

$\xrightarrow[\text{CBV}]{}$ (let^1 (yin (AT K$_i$)) (let^1 (yang (STAR CCL)) (yin yang)))

が成り立ちます．さらに，この計算過程の中で #* が表示されます．

証明　証明は先の例で示した簡約を行うだけです．

(1) $\texttt{Yin-Yang} = (\texttt{let}^1\ (\texttt{yin}\ (\texttt{AT CCL}))\ (\texttt{let}^1\ (\texttt{yang}\ (\texttt{STAR CCL}))\ (\texttt{yin yang})))$

$\xrightarrow[\text{CBV}]{}$ $(\texttt{let}^1\ (\texttt{yin}\ (\texttt{AT K}_1))\ (\texttt{let}^1\ (\texttt{yang}\ (\texttt{STAR CCL}))\ (\texttt{yin yang})))$

where

$\texttt{K}_1 = (\texttt{cnt}\ (\texttt{z})\ (\texttt{let}^1\ (\texttt{yin}\ (\texttt{AT z}))\ (\texttt{let}^1\ (\texttt{yang}\ (\texttt{STAR CCL}))\ (\texttt{yin yang}))))$

(2) $(\texttt{let}^1\ (\texttt{yin}\ (\texttt{AT K}_{i-1}))\ (\texttt{let}^1\ (\texttt{yang}\ (\texttt{STAR CCL}))\ (\texttt{yin yang})))$

$\xrightarrow[\text{CBV}]{\texttt{\#\@}}$ $(\texttt{let}^1\ (\texttt{yin K}_{i-1})\ (\texttt{let}^1\ (\texttt{yang}\ (\texttt{STAR CCL}))\ (\texttt{yin yang})))$

$\xrightarrow[\text{CBV}]{}$ $(\texttt{let}^1\ (\texttt{yang}\ (\texttt{STAR CCL}))\ (\texttt{K}_{i-1}\ \texttt{yang}))$

$\xrightarrow[\text{CBV}]{}$ $(\texttt{let}^1\ (\texttt{yang}\ (\texttt{STAR K}_i))\ (\texttt{K}_{i-1}\ \texttt{yang}))$

where

$\texttt{K}_i = (\texttt{cnt}\ (\texttt{z})\ (\texttt{let}^1\ (\texttt{yang}\ (\texttt{STAR z}))\ (\texttt{K}_{i-1}\ \texttt{yang})))$

(3) $(\texttt{let}^1\ (\texttt{yang}\ (\texttt{STAR K}_i))\ (\texttt{K}_j\ \texttt{yang}))$

$\xrightarrow[\text{CBV}]{\texttt{\#*}}$ $(\texttt{let}^1\ (\texttt{yang K}_i)\ (\texttt{K}_j\ \texttt{yang}))$

$\xrightarrow[\text{CBV}]{}$ $(\texttt{K}_j\ \texttt{K}_i)$

where

$\texttt{K}_j = (\texttt{cnt}\ (\texttt{z})\ (\texttt{let}^1\ (\texttt{yang}\ (\texttt{STAR z}))\ (\texttt{K}_{j-1}\ \texttt{yang})))$

$\xrightarrow[\text{CBV}]{}$ $(\texttt{let}^1\ (\texttt{yang}\ (\texttt{STAR K}_i))\ (\texttt{K}_{j-1}\ \texttt{yang}))$

(4) $(\texttt{let}^1\ (\texttt{yang}\ (\texttt{STAR K}_i))\ (\texttt{K}_1\ \texttt{yang}))$

$\xrightarrow[\text{CBV}]{\texttt{\#*}}$ $(\texttt{let}^1\ (\texttt{yang K}_i)\ (\texttt{K}_1\ \texttt{yang}))$

$\xrightarrow[\text{CBV}]{}$ $(\texttt{K}_1\ \texttt{K}_i)$

where

$\texttt{K}_1 = (\texttt{cnt}\ (\texttt{z})\ (\texttt{let}^1\ (\texttt{yin}\ (\texttt{AT z}))\ (\texttt{let}^1\ (\texttt{yang}\ (\texttt{STAR CCL}))\ (\texttt{yin yang}))))$

$\xrightarrow[\text{CBV}]{}$ $(\texttt{let}^1\ (\texttt{yin}\ (\texttt{AT K}_i))\ (\texttt{let}^1\ (\texttt{yang}\ (\texttt{STAR CCL}))\ (\texttt{yin yang})))$

¶ Yin-Yang は，(1) の計算を行ったあと，i の値を増やしながら (2),(3),(4) の計算を繰り返しています．つまり，

$(\texttt{let}^1\ (\texttt{yin}\ (\texttt{AT K}_{i-1}))\ (\texttt{let}^1\ (\texttt{yang}\ (\texttt{STAR CCL}))\ (\texttt{yin yang})))$

$\xrightarrow[\text{CBV}]{\texttt{\#\@}}{}^{*}$ $(\texttt{let}^1\ (\texttt{yang}\ (\texttt{STAR K}_i))\ (\texttt{K}_{i-1}\ \texttt{yang}))$

$\xrightarrow[\text{CBV}]{\texttt{\#*}}{}^{*}$ $(\texttt{let}^1\ (\texttt{yang}\ (\texttt{STAR K}_i))\ (\texttt{K}_{i-2}\ \texttt{yang}))$

……

$\xrightarrow[\text{CBV}]{\texttt{\#*}}{}^{*}$ $(\texttt{let}^1\ (\texttt{yang}\ (\texttt{STAR K}_i))\ (\texttt{K}_1\ \texttt{yang}))$

$\xrightarrow[\text{CBV}]{\texttt{\#*}}{}^{*}$ $(\texttt{let}^1\ (\texttt{yin}\ (\texttt{AT K}_i))\ (\texttt{let}^1\ (\texttt{yang}\ (\texttt{STAR CCL}))\ (\texttt{yin yang})))$

という計算過程を i を増やしながら繰り返しています．ただし，$i \geq 2$ です．そこで，(1) の計算を「第 1 ステージ」と呼び，上の計算過程を「第 i ステージ」($i \geq 2$) と呼ぶことにすると，上の解析結果から次の系が得られます．

■ 系（Yin-Yang Puzzle の動作） 5.2

yin-yang puzzle は次のように動作します．

- まず第 1 ステージを実行します．このステージは何も表示しません．
- 次に $i = 2, 3, \ldots$ に対してこの順に第 i ステージを実行します．
  第 i ステージでは，1 つの`#\@`と $i-1$ 個の`#\*`を表示します．

6　再帰的な手続きについて

6.1　rec[1] 式のマクロ展開

次の命題は，rec[1] 式をマクロとして定義する方法を示しています．

命題（rec[1] 式のマクロ展開） 6.1

rec[1] 式と簡約規則の (rec[1]) を Scheme-0 から除外します．そこで改めて，変数 $X \in \mathbf{X}$ と式 $M \in \mathbf{M}$ に対して rec[1] マクロ を次のように定めます．

$$\texttt{(rec}^1\ \ \boldsymbol{X}\ \ \boldsymbol{M}\texttt{)} \ = \ \texttt{(}N\ \ N\texttt{)} \ \text{ where } N = \texttt{(lam (}X\texttt{)}\ M\{X \leftarrow \texttt{(}X\ \ X\texttt{)}\})$$

この rec[1] マクロを CBV-簡約基に登録します．このとき，次の (rec[1]) が成り立ちます．

$$(\text{rec}^1)\ \ E[\texttt{(rec}^1\ X\ M\texttt{)}] \xrightarrow[\text{CBV}]{} E[M\{X \leftarrow \texttt{(rec}^1\ X\ M\texttt{)}\}]$$

ここで，E は評価対象式の CBV-評価文脈を表し，rec[1] マクロは CBV-簡約基になります．

証明

$$\begin{aligned}
& E[\texttt{(rec}^1\ \ X\ \ M\texttt{)}] \\
= \ & E[\texttt{(}N\ \ N\texttt{)}] \ \text{ where } \ N = \texttt{(lam (}X\texttt{)}\ M\{X \leftarrow \texttt{(}X\ \ X\texttt{)}\}) \\
= \ & E[\texttt{((lam (}X\texttt{)}\ M\{X \leftarrow \texttt{(}X\ \ X\texttt{)}\})\ N\texttt{)}] \ \text{ where } \ N = \texttt{(lam (}X\texttt{)}\ M\{X \leftarrow \texttt{(}X\ \ X\texttt{)}\}) \\
\xrightarrow[\text{CBV}]{(\beta)} \ & E[M\{X \leftarrow \texttt{(}N\ \ N\texttt{)}\}] \ \text{ where } \ N = \texttt{(lam (}X\texttt{)}\ M\{X \leftarrow \texttt{(}X\ \ X\texttt{)}\}) \\
= \ & E[M\{X \leftarrow \texttt{(rec}^1\ \ X\ \ M\texttt{)}\}]
\end{aligned}$$

ここで，$\xrightarrow[\text{CBV}]{(\beta)}$ は簡約規則の (β) を使用していることを示しています．□

¶ 自己適用の式 `(`N N`)` は，手続き側の N が内蔵している計算式 M を評価の俎上に載せるとともに，将来の再帰的な計算に備えて実引数側の N を計算式 M の中に埋め込んでいます．

例 6.2

自然数 n に対して $n!$ を計算する次の rec[1] 式（例 1.2 参照）をマクロ展開してみます．

```
1 (rec¹ fact (lam (n) (if (< n 2) 1 (* n (fact (- n 1))))))
```

この例の場合，上の命題における X と M は

$X =$ `fact`
$M =$ `(lam (n) (if (< n 2) 1 (* n (fact (- n 1)))))`

なので，マクロ展開は次のようになります．

$$\begin{aligned}
& \texttt{(rec}^1\ \ \texttt{fact}\ \ M\texttt{)} = \texttt{(}N\ \ N\texttt{)} \\
& \text{where } \ N = \texttt{(lam (}X\texttt{)}\ M\{X \leftarrow \texttt{(}X\ \ X\texttt{)}\}) \\
& \qquad = \texttt{(lam (fact)}\ M\{\texttt{fact} \leftarrow \texttt{(fact fact)}\}\texttt{)} \\
& \qquad = \texttt{(lam (fact) (lam (n) (if (< n 2) 1 (* n ((fact fact) (- n 1))))))}
\end{aligned}$$

マクロ展開後の式 $(N\ N)$ と自然数 k に対して，次が成り立ちます．

$$
\begin{aligned}
& ((N\ N)\ k) \\
= \quad & (((\texttt{lam}\ (\texttt{fact})\ (\texttt{lam}\ (\texttt{n})\ \ \cdots\cdots(\text{省略})\cdots\cdots\ \ ))\ N)\ k) \\
\xrightarrow[\text{CBV}]{(\beta)} \quad & (((\texttt{lam}\ (\texttt{n})\ (\texttt{if}\ (<\ \texttt{n}\ 2)\ 1\ (*\ \texttt{n}\ ((N\ N)\ (-\ \texttt{n}\ 1)))))\ k) \\
\xrightarrow[\text{CBV}]{(\beta)} \quad & (\texttt{if}\ (<\ k\ 2)\ 1\ (*\ k\ ((N\ N)\ (-\ k\ 1)))) \\
\xrightarrow[\text{CBV}]{(\delta)} \quad & \begin{cases} (\texttt{if}\ \texttt{\#t}\ 1\ (*\ k\ ((N\ N)\ (-\ k\ 1)))) & \text{if } k < 2 \\ (\texttt{if}\ \texttt{\#f}\ 1\ (*\ k\ ((N\ N)\ (-\ k\ 1)))) & \text{if } k \geq 2 \end{cases} \\
\xrightarrow[\text{CBV}]{\text{(if-f),(if-t)}} \quad & \begin{cases} 1 & \text{if } k < 2 \\ (*\ k\ ((N\ N)\ (-\ k\ 1))) & \text{if } k \geq 2 \end{cases}
\end{aligned}
$$

これと k に関する数学的帰納法より，$(N\ N)$ が階乗関数を計算することは明らかでしょう．

試しに，マクロ展開後の式を Guile で実行してみます．

```
1 ;;; rec1.scm
2 (define N
3   (lambda (fact)
4     (lambda (n) (if (< n 2) 1 (* n ((fact fact) (- n 1)))))))
```

```
$ guile⏎
GNU Guile 2.2.4
   …… 起動メッセージ ……
scheme@(guile-user)> (load "rec1.scm")⏎
scheme@(guile-user)> ((N N) 2)⏎
$1 = 2
scheme@(guile-user)> ((N N) 3)⏎
$2 = 6
scheme@(guile-user)> ((N N) 4)⏎
$3 = 24
scheme@(guile-user)> ,trace ((N N) 4)⏎
trace: |  (N #<procedure N (fact)>)
trace: |  #<procedure 55d0b3d828c0 at … 省略 … >
trace: (_ 4)
trace: |  (N #<procedure N (fact)>)
trace: |  #<procedure 55d0b3892ca0 at … 省略 … >
trace: |  (_ 3)
trace: |  |  (N #<procedure N (fact)>)
trace: |  |  #<procedure 55d0b38d3fa0 at … 省略 … >
trace: |  |  (_ 2)
trace: |  |  |  (N #<procedure N (fact)>)
trace: |  |  |  #<procedure 55d0b39afa40 at … 省略 … >
trace: |  |  |  (_ 1)
trace: |  |  |  1
trace: |  |  2
trace: |  6
trace: 24
```

¶ 上の命題は，自己適用をうまく利用すれば，再帰的な計算を素朴な計算過程として実現できるこ

とを示しています[8]．つまり，rec^1 式や (rec^1) と同等の機能を素朴な計算過程として実現するためのプログラミングテクニックを示しています．そのため，rec^1 式や (rec^1) を除外しても言語機能は低下しません．

¶ 再帰的な計算を実現する方法として，不動点コンビネータを利用する方法が一般によく知られています．しかし，その方法は上の命題のように単純ではありません．

一般的に知られている不動点コンビネータを CBV 戦略のために利用する場合，不動点コンビネータに遅延（delay）を組み込み，rec^1 式のマクロ展開後の式に強制（force）を組み込む必要があります．そのために，簡約規則の (rec^1) に示した左辺と右辺に対して，ある種の等価性は保証できるものの[9]，その等価性を単方向的で素朴な計算過程に置き換えることはできません．そのため，不動点コンビネータに執着して，自己適用を利用したプログラミングテクニックを知らなければ，(rec^1) またはそれに相当する簡約規則を除外できるとは判断できそうもありません．つまり，等価性を根拠に，rec^1 式固有の簡約規則を用意しなければならないと思うことでしょう．

実のところ，筆者は最初に不動点コンビネータを利用する方法を勉強しました．ここで述べていることはその勉強に基づいています．でも，途中で自己適用を利用する方法を勉強できたので，そちらに切り替えたのです．個人的な印象では，CBV 戦略にとって，不動点コンビネータは世間で流布されているほどには有用ではなく，自己適用を利用する方法のほうが有用であると感じています[10]．

6.2 rec 式（相互再帰）

Scheme-0 は相互再帰を実現する機能がありません．でも，rec^1 式と (rec^1) の拡張版を用意することによって実現できます．

rec 式 6.3

Scheme-0 に次のような rec 式 を追加します．これは一般の式であって値式ではありません．

$$M \ ::= \ (\texttt{rec}\ \ Y\ \ (X_1\ \ M_1)\ \cdots\ (X_n\ \ M_n))$$

Y を rec 式の ラベル と呼び，$(X_1\ \ M_1)\ \cdots\ (X_n\ \ M_n)$ を rec 式の本体 と呼ぶことにします．

rec 式の本体は相互再帰している手続き名と本体の組を表し，ラベルはその中の 1 つの手続き名を表しています．ただし，構文的に Y は X_1〜X_n 以外の変数名でもかまわないし，X_1〜X_n の中に同じ名前が重複して現れていてもかまいません．もちろん，次に示す簡約規則では，そういった rec 式は簡約できずにエラーと解釈されます．

自由変数と代入 6.4

新たな構文要素に対しては，必ず，自由変数の集合と代入を定義しなければなりません．そうしておかないと代入を伴う簡約規則を適用できないからです．

rec 式の自由変数（の記号）の集合は次のように定義されます．

- $FV((\texttt{rec}\ \ Y\ \ (X_1\ \ M_1)\ \cdots\ (X_n\ \ M_n))) = (FV(M_1) \cup \cdots \cup FV(M_n)) - \{Y, X_1, \ldots, X_n\}$

[8] ここで言う素朴な計算過程は，簡約規則の (rec^1) や継続簡約を利用しない計算過程のことです．より一般的な言い方をすると，手続き適用と条件分岐だけを使って実現された計算過程のことです・

[9] CBV 戦略の場合，不動点コンビネータによる方法は，式の間のある種の等価性の理論に依存しています．その等価性の理論は Felleisen たちの本 [FFF09] で $=_{\mathrm{v}}$ として議論しています．

[10] 筆者は，等価性の理論に依存せずに CBV 戦略に対して有用な不動点コンビネータが存在するかどうか，つまり単方向的で素朴な計算過程に展開できるような不動点コンビネータが存在するかどうは分かっていません．さらに，ラムダ計算や CBV 戦略の深層を理解しているわけではないので，深層においては不動点コンビネータに大きな意義があるのだろうと思います．

rec 式に対する代入は次のように定義されます．rec 式内の Y や X_1〜X_n は束縛変数で，手続き名を表しています．そこで，代入項（下記の N）の中の自由変数とこれらの束縛変数の衝突が起こらないように束縛変数の名前を変えながら各 M_i に代入を適用します．以下，Γ は rec 式の本体 $(X_1\ M_1)\ \cdots\ (X_n\ M_n)$ を略記しています．

- $(\texttt{rec}\ Y\ \Gamma)\{X \leftarrow N\} = (\texttt{rec}\ Y\ \Gamma)$
  if $X \in \{Y, X_1, \ldots, X_n\}$

  $(\texttt{rec}\ Y\ \Gamma)\{X \leftarrow N\} = (\texttt{rec}\ Y\ (X_1\ M_1\{X \leftarrow N\})\ \cdots\ (X_n\ M_n\{X \leftarrow N\}))$
  if $X \notin \{Y, X_1, \ldots, X_n\}$ かつ $\{Y, X_1, \ldots, X_n\} \cap \mathrm{FV}(N) = \emptyset$.

  $(\texttt{rec}\ Y\ \Gamma)\{X \leftarrow N\} = (\texttt{rec}\ Y\ (X_1\ M_1\sigma)\ \cdots\ (X_n\ M_n\sigma))$
  if $X \notin \{Y, X_1, \ldots, X_n\}$ かつ $\{Y, X_1, \ldots, X_n\} \cap \mathrm{FV}(N) \neq \emptyset$.
  where

$$Z = \begin{cases} Y & \text{if } Y \notin \mathrm{FV}(N) \\ \text{新たな変数} & \text{if } Y \in \mathrm{FV}(N) \end{cases}$$

$$Z_i = \begin{cases} X_i & \text{if } X_i \notin \mathrm{FV}(N) \\ \text{新たな変数} & \text{if } X_i \in \mathrm{FV}(N) \end{cases}$$

$$\sigma = \{Y \leftarrow Z\}\{X_1 \leftarrow Z_1\} \ldots \{X_n \leftarrow Z_n\}\{X \leftarrow N\}$$

ちなみに，rec 式の代入は lam 式の場合（同時代入）の考え方と同じです．次節のマクロ展開を見ると，この点がもっとよく分かると思います．

簡約規則 (rec)

6.5

rec 式を CBV-簡約基に登録した上で，次の簡約規則を基本簡約の 1 つとして追加します．以下，E は簡約対象の式の CBV-評価文脈を表していて，そのカッコ内の rec 式は CBV-簡約基になります．それから，記述を簡略化するために，rec 式の本体 $(X_1\ M_1)\ \cdots\ (X_n\ M_n)$ を Γ で略記しています．

(rec) $E[(\texttt{rec}\ Z\ \Gamma)] \xrightarrow[\text{CBV}]{} E[M_k\{X_j \leftarrow (\texttt{rec}\ X_j\ \Gamma) \mid 1 \leq j \leq n\}]$
（ただし，$Z = X_k$ かつ X_1〜X_n は互いに異なる）

⋆ rec 式は $Z = X_k$ に対応する式 M_k を評価対象として取り出すと同時に，将来の計算に備えて M_k の中のそれぞれの X_j に rec 式を代入します．

例

6.6

次の rec 式は相互再帰を使って，引数として与えられた自然数が奇数か否かを判定します．ラベルを even に変えれば偶数か否かを判定できます．

```
1 (rec odd
2    (even (lam (n) (if (= n 0) #t (odd (- n 1)))))
3    (odd  (lam (n) (if (= n 0) #f (even (- n 1))))))
```

以下に計算例を示します．Γ は rec 式の本体を略記しています．それから，最初のステップだけ少し丁寧に簡約します．

```
((rec odd Γ) 3)
```

$\xrightarrow[\text{CBV}]{}$ `((lam (n) (if (= n 0) #f (even (- n 1))))`σ `3)`
where σ = { even ← (rec even Γ), odd ← (rec odd Γ) }

```
=      ((lam (n) (if (= n 0) #f ((rec even Γ) (- n 1)))) 3)
CBV→   (if (= 3 0) #f ((rec even Γ) (- 3 1)))
CBV→   (if #f #f ((rec even Γ) (- 3 1)))
CBV→   ((rec even Γ) (- 3 1))
CBV→   ((lam (n) (if (= n 0) #t ((rec odd Γ) (- n 1)))) (- 3 1))
CBV→   ((lam (n) (if (= n 0) #t ((rec odd Γ) (- n 1)))) 2)
CBV→   (if (= 2 0) #t ((rec odd Γ) (- 2 1)))
CBV→   (if #f #t ((rec odd Γ) (- 2 1)))
CBV→   ((rec odd Γ) (- 2 1))
CBV→   ((lam (n) (if (= n 0) #f ((rec even Γ) (- n 1)))) (- 2 1))
CBV→   ((lam (n) (if (= n 0) #f ((rec even Γ) (- n 1)))) 1)
CBV→   (if (= 1 0) #f ((rec even Γ) (- 1 1)))
CBV→   (if #f #f ((rec even Γ) (- 1 1)))
CBV→   ((rec even Γ) (- 1 1))
CBV→   ((lam (n) (if (= n 0) #t ((rec odd Γ) (- n 1)))) (- 1 1))
CBV→   ((lam (n) (if (= n 0) #t ((rec odd Γ) (- n 1)))) 0)
CBV→   (if (= 0 0) #t ((rec odd Γ) (- 0 1)))
CBV→   (if #t #t ((rec odd Γ) (- 0 1)))
CBV→   #t
```

6.3 rec 式のマクロ展開

rec[1] 式と同様に，rec 式もマクロとして定義できます．

命題（rec 式のマクロ展開） 6.7

rec 式と簡約規則の (rec) を Scheme-0 から除外します．そこで改めて，rec マクロ を次のように定めます．以下の Γ は rec 式の本体 $(X_1\ M_1)\ \cdots\ (X_n\ M_n)$ を略記しています．

$$\begin{aligned}(\texttt{rec}\ X_k\ \Gamma) \;=\; & (N_k\ N_1\ \cdots\ N_n)\\ & \text{where}\ \ N_i = (\texttt{lam}\ (X_1\ \cdots\ X_n)\ M_i\sigma)\\ & \qquad \sigma = \{\, X_j \leftarrow (X_j\ X_1\ \ldots\ X_n) \mid 1 \le j \le n \,\}\end{aligned}$$

ただし，X_k が X_1〜X_n のいずれかであり，X_1〜X_n が互いに異なるときにのみマクロ展開が可能であるとします．この rec マクロを CBV-簡約基に登録します．このとき，次の (rec) が成り立ちます．

$$\text{(rec)}\ \ E[(\texttt{rec}\ X_k\ \Gamma)] \xrightarrow[\text{CBV}]{} E[M_k\{X_j \leftarrow (\texttt{rec}\ X_j\ \Gamma) \mid 1 \le j \le n\}]$$

ここで，E は評価対象式の CBV-評価文脈を表し，`rec` マクロは CBV-簡約基です．

証明

$$
\begin{aligned}
& E[(\texttt{rec}\ X_k\ \Gamma)] \\
= \quad & E[(N_k\ N_1\ \cdots\ N_n)] \\
& \quad \text{where}\ \ N_i = (\texttt{lam}\ (X_1\ \cdots\ X_n)\ M_i\sigma) \\
& \qquad\qquad \sigma = \{\, X_j \leftarrow (X_j\ X_1\ \ldots\ X_n) \mid 1 \le j \le n \,\} \\
= \quad & E[((\texttt{lam}\ (X_1\ \ldots\ X_n)\ M_k\sigma)\ N_1\ \ldots\ N_n)] \\
& \quad \text{where}\ \ N_i = (\texttt{lam}\ (X_1\ \cdots\ X_n)\ M_i\sigma) \\
& \qquad\qquad \sigma = \{\, X_j \leftarrow (X_j\ X_1\ \ldots\ X_n) \mid 1 \le j \le n \,\} \\
\xrightarrow[\text{CBV}]{(\beta)} \quad & E[M_k\{X_j \leftarrow (N_j\ N_1\ \cdots\ N_n) \mid 1 \le j \le n\}] \\
& \quad \text{where}\ \ N_i = (\texttt{lam}\ (X_1\ \cdots\ X_n)\ M_i\sigma) \\
& \qquad\qquad \sigma = \{\, X_j \leftarrow (X_j\ X_1\ \ldots\ X_n) \mid 1 \le j \le n \,\} \\
= \quad & E[M_k\{X_j \leftarrow (\texttt{rec}\ X_j\ \Gamma) \mid 1 \le j \le n\}]
\end{aligned}
$$

ここで，$\xrightarrow[\text{CBV}]{(\beta)}$ は簡約規則の (β) を使用していることを示しています．□

¶ マクロ展開後の式 $(N_k\ N_1\ \ldots\ N_n)$ は，N_k が内蔵している計算式 M_k を評価の俎上に載せるとともに，将来の再帰的な計算に備えて N_1〜N_n を計算式 M_k に埋め込んでいます．

▮例 6.8

次の `rec` 式をマクロ展開してみます．以下，上の命題で使用した記法をそのまま使います．

```
1 (rec even
2    (even (lam (n) (if (= n 0) #t (odd (- n 1)))))
3    (odd  (lam (n) (if (= n 0) #f (even (- n 1))))))
```

この式の X_j と M_j は次のようになります．

$X_1 =$ `even`
$M_1 =$ `(lam (n) (if (= n 0) #t (odd (- n 1))))`
$X_2 =$ `odd`
$M_2 =$ `(lam (n) (if (= n 0) #f (even (- n 1))))`

そのため，σ と N_i は次のようになります．

$$
\begin{aligned}
\sigma &= \{\, X_1 \leftarrow (X_1\ X_1\ X_2),\ X_2 \leftarrow (X_2\ X_1\ X_2) \,\} \\
&= \{\, \texttt{even} \leftarrow (\texttt{even even odd}),\ \texttt{odd} \leftarrow (\texttt{odd even odd}) \,\} \\
N_1 &= (\texttt{lam}\ (X_1\ X_2)\ M_1\sigma) \\
&= (\texttt{lam}\ (\texttt{even odd})\ (\texttt{lam}\ (\texttt{n})\ (\texttt{if}\ (\texttt{= n 0})\ \texttt{\#t}\ (\texttt{odd}\ (\texttt{- n 1}))))\sigma) \\
&= (\texttt{lam}\ (\texttt{even odd})\ (\texttt{lam}\ (\texttt{n})\ (\texttt{if}\ (\texttt{= n 0})\ \texttt{\#t}\ ((\texttt{odd even odd})\ (\texttt{- n 1}))))) \\
N_2 &= (\texttt{lam}\ (X_1\ X_2)\ M_2\sigma) \\
&= (\texttt{lam}\ (\texttt{even odd})\ (\texttt{lam}\ (\texttt{n})\ (\texttt{if}\ (\texttt{= n 0})\ \texttt{\#f}\ (\texttt{even}\ (\texttt{- n 1}))))\sigma) \\
&= (\texttt{lam}\ (\texttt{even odd})\ (\texttt{lam}\ (\texttt{n})\ (\texttt{if}\ (\texttt{= n 0})\ \texttt{\#f}\ ((\texttt{even even odd})\ (\texttt{- n 1})))))
\end{aligned}
$$

以上から上の `rec` マクロ `(rec even Γ)` は次のように展開されます．なお，Γ は上で示した `rec` 式の本体を表します．

```
(rec even Γ)
```

$$
\begin{aligned}
&= (N_1\ N_1\ N_2) \\
&= (((\texttt{lam}\ (\texttt{even odd})\ (\texttt{lam}\ (\texttt{n})\ (\texttt{if}\ (\texttt{= n 0})\ \texttt{\#t}\ ((\texttt{odd even odd})\ (\texttt{- n 1})))))\ N_1\ N_2) \\
&= (\texttt{lam}\ (\texttt{even odd})\ (\texttt{lam}\ (\texttt{n})\ (\texttt{if}\ (\texttt{= n 0})\ \texttt{\#t}\ ((N_2\ N_1\ N_2)\ (\texttt{- n 1})))))
\end{aligned}
$$

同様に，rec マクロ (rec odd Γ) は次のように展開されます．

$$
\begin{aligned}
&(\texttt{rec odd}\ \Gamma) \\
&= (N_2\ N_1\ N_2) \\
&= (((\texttt{lam}\ (\texttt{even odd})\ (\texttt{lam}\ (\texttt{n})\ (\texttt{if}\ (\texttt{= n 0})\ \texttt{\#f}\ ((\texttt{even even odd})\ (\texttt{- n 1})))))\ N_1\ N_2) \\
&= (\texttt{lam}\ (\texttt{even odd})\ (\texttt{lam}\ (\texttt{n})\ (\texttt{if}\ (\texttt{= n 0})\ \texttt{\#f}\ ((N_1\ N_1\ N_2)\ (\texttt{- n 1})))))
\end{aligned}
$$

これらのマクロと任意の自然数 k に対して次が成り立ちます．

$$
\begin{aligned}
&(1) && (\texttt{(rec even}\ \Gamma\texttt{)}\ k) \\
&= && ((N_1\ N_1\ N_2)\ k) \\
&\xrightarrow[\text{CBV}]{(\beta)} && ((\texttt{lam}\ (\texttt{n})\ (\texttt{if}\ (\texttt{= n 0})\ \texttt{\#t}\ ((N_2\ N_1\ N_2)\ (\texttt{- n 1}))))\ k) \\
&\xrightarrow[\text{CBV}]{(\beta)} && (\texttt{if}\ (\texttt{= k 0})\ \texttt{\#t}\ ((N_2\ N_1\ N_2)\ (\texttt{- k 1}))) \\
&\xrightarrow[\text{CBV}]{(\delta)} && \begin{cases} (\texttt{if \#t \#t}\ ((N_2\ N_1\ N_2)\ (\texttt{- k 1}))) & \text{if } k = 0 \\ (\texttt{if \#f \#t}\ ((N_2\ N_1\ N_2)\ (\texttt{- k 1}))) & \text{if } k > 0 \end{cases} \\
&\xrightarrow[\text{CBV}]{\text{(if-f),(if-t)}} && \begin{cases} \texttt{\#t} & \text{if } k = 0 \\ ((N_2\ N_1\ N_2)\ (\texttt{- }k\ \texttt{1})) & \text{if } k > 0 \end{cases} \\
&= && \begin{cases} \texttt{\#t} & \text{if } k = 0 \\ (\texttt{(rec odd}\ \Gamma\texttt{)}\ (\texttt{- }k\ \texttt{1})) & \text{if } k > 0 \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
&(2) && (\texttt{(rec odd}\ \Gamma\texttt{)}\ k) \\
&= && ((N_2\ N_1\ N_2)\ k) \\
&\xrightarrow[\text{CBV}]{(\beta)} && ((\texttt{lam}\ (\texttt{n})\ (\texttt{if}\ (\texttt{= n 0})\ \texttt{\#f}\ ((N_1\ N_1\ N_2)\ (\texttt{- n 1}))))\ k) \\
&\xrightarrow[\text{CBV}]{(\beta)} && (\texttt{if}\ (\texttt{= k 0})\ \texttt{\#f}\ ((N_1\ N_1\ N_2)\ (\texttt{- k 1}))) \\
&\xrightarrow[\text{CBV}]{(\delta)} && \begin{cases} (\texttt{if \#t \#f}\ ((N_1\ N_1\ N_2)\ (\texttt{- k 1}))) & \text{if } k = 0 \\ (\texttt{if \#f \#f}\ ((N_1\ N_1\ N_2)\ (\texttt{- k 1}))) & \text{if } k > 0 \end{cases} \\
&\xrightarrow[\text{CBV}]{\text{(if-f),(if-t)}} && \begin{cases} \texttt{\#f} & \text{if } k = 0 \\ ((N_1\ N_1\ N_2)\ (\texttt{- }k\ \texttt{1})) & \text{if } k > 0 \end{cases} \\
&= && \begin{cases} \texttt{\#f} & \text{if } k = 0 \\ (\texttt{(rec even}\ \Gamma\texttt{)}\ (\texttt{- }k\ \texttt{1})) & \text{if } k > 0 \end{cases}
\end{aligned}
$$

これらから，$(N_1\ N_1\ N_2)$ と $(N_2\ N_1\ N_2)$ が相互再帰的に機能することは明らかでしょう．正確に言うと，これらと k に関する数学的帰納法より，それぞれの rec マクロが自然数の偶奇性を正しく判定することは明らかでしょう．

試しに，マクロ展開後の式を Guile で実行してみます．

```
1 ;;; rec.scm
2 (define N1
3   (lambda (even odd)
4     (lambda (n) (if (= n 0) #t ((odd even odd) (- n 1))))))
5 (define N2
6   (lambda (even odd)
7     (lambda (n) (if (= n 0) #f ((even even odd) (- n 1))))))
```

```
$ guile⏎
GNU Guile 2.2.4
   …… 起動メッセージ ……
scheme@(guile-user)> (load "rec.scm")⏎
scheme@(guile-user)> ((N1 N1 N2) 1)⏎
$1 = #f
scheme@(guile-user)> ((N1 N1 N2) 2)⏎
$2 = #t
scheme@(guile-user)> ((N1 N1 N2) 3)⏎
$3 = #f
scheme@(guile-user)> ((N1 N1 N2) 4)⏎
$4 = #t
scheme@(guile-user)> ,trace ((N1 N1 N2) 5)⏎
trace: |  (N1 #<procedure N1 (even odd)> #<procedure N2 (even odd)>)
trace: |  #<procedure 55dffe6c5c00 at … 省略 … >
trace: (_ 5)
trace: |  (N2 #<procedure N1 (even odd)> #<procedure N2 (even odd)>)
trace: |  #<procedure 55dffe7a55c0 at … 省略 … >
trace: (_ 4)
trace: |  (N1 #<procedure N1 (even odd)> #<procedure N2 (even odd)>)
trace: |  #<procedure 55dffe7ce6e0 at … 省略 … >
trace: (_ 3)
trace: |  (N2 #<procedure N1 (even odd)> #<procedure N2 (even odd)>)
trace: |  #<procedure 55dffea05820 at … 省略 … >
trace: (_ 2)
trace: |  (N1 #<procedure N1 (even odd)> #<procedure N2 (even odd)>)
trace: |  #<procedure 55dffeae3980 at … 省略 … >
trace: (_ 1)
trace: |  (N2 #<procedure N1 (even odd)> #<procedure N2 (even odd)>)
trace: |  #<procedure 55dffebf3b20 at … 省略 … >
trace: (_ 0)
trace: #f
```

なお，`(N1 N1 N2)` と `(N2 N1 N2)` の呼び出しは末尾呼び出しなので，繰り返し処理に変換されています．

参考文献

[FFF09] M. Felleisen, R. B. Findler, and M. Flatt: Semantics Engineering with PLT Redex, The MIT Press, 2009.

⋆ このノートは，この本の第 5 章までを勉強したことに基づいています．

[FF06] M. Felleisen and M. Flatt: Programming Languages and Lambda Caluculi, July 12, 2006.
⋆ これは [FFF09] の Draft だと思います.

[Dyv09] R. K. Dybvig: The Scheme Programming Language, 4th Edition, 2009.
`https://www.scheme.com/tspl4/`

独り言

今回の勉強を通して，継続は実行時評価文脈（計算状況）の手続き化だということがよく分かりました.「実行時」というところがミソです．筆者は，しばらくの間,「静的」な評価文脈の手続き化だと勘違いしていました．つまり，`call/cc` 式を含む「ソースコード上の」式全体を，`call/cc` 式のところを仮引数に置き換えて手続き化したものだと思い込んでいたのです．色々な説明を読むと辻褄が合わないので，実行時の計算状況を手続き化しているらしいことは推測できるようになりました．でも，詳しい動作についてまったくイメージが持てませんでした．そこで，今回のような勉強をしてみようと思ったのです．さらに，今回の勉強の副作用として，再帰的な手続きや相互再帰的な手続きを，素朴な計算過程を実行するマクロとして定義する方法も勉強できました．

A 付録

A.1 命題 3.5 （CBV-評価文脈の一意性）の証明

命題 3.5 を証明します．その命題は次の主張でした．

> M を任意の式として，値式でないとします．このとき，
>
> - $M = E[C]$ を満たす評価文脈 E と CBV-簡約基 C の組が一意的に定まります．

証明 M の構造に関する帰納法によって証明します．

まず，M そのものがが CBV-簡約基のときには，$E = \Box$ となり，上の主張が成り立ちます ... などと言ってすませてしまいたいところですが，本当はもう少し真面目に議論しなければなりません．

いま，$M = (V_0\ V_1\ \ldots\ V_n)$ だったとします．このとき，

$(\star)$ $E = \Box$, $C = M$

が形式上の評価文脈と CBV-簡約基の組になることは明らかです．さらに，各 V_i は値式であり，CBV-簡約基にはなり得ないので，ホールになることはありません．さらに V_i の内部にホールが入る可能性を考えたとき，V_i は構造を持った値式（つまり，ラムダ式か `cnt` 式）でなければなりませんが，形式上の評価文脈の定義より，それらの内部にホールが入ることはありません．以上をまとめると，M が CBV-簡約基 $(V_0\ V_1\ \ldots\ V_n)$ だったとき，上の $(\star)$ がただ 1 つの評価文脈と CBV 簡約基の組になります．

次，$M = (\texttt{if}\ V\ M_2\ M_3)$ だったとします．このときにも上の $(\star)$ が形式上の評価文脈と CBV-簡約基の組になります．一方，`if` 式の条件はすでに値式なのでホールになることはなく，形式上の評価文脈の定義より，`if` 式の内部（then 部と else 部）にホールが入ることはありません．従って，この場合も $(\star)$ がただ 1 つの評価文脈と CBV 簡約基の組になります．

$M = (\texttt{let}^1\ (X\ V)\ N)$ だった場合も `if` 式だった場合と同様です．さらに，$M = (\texttt{rec}^1\ X\ N)$ や $M = (\texttt{ccl}\ (X)\ N)$ だった場合にはホールが M の内部に入ることはありません．

以上より，M が CBV-簡約基だったとき，$E = \Box$ となり，上の主張が成り立ちます．

そこで，M が CBV-簡約基ではなかったとします．つまり，M は手続き適用か，`if` 式か，`let`[1] 式のいずれかとします．ここで，`rec`[1] 式や `ccl` 式は CBV-簡約基以外にはなれないこと，`lam` 式や `cnt` 式は値式であることに注意して下さい．従って，これらは以後の議論から除外されます．さらに，帰納法の仮定として，M より構造的に簡単なあらゆる式に関して上の主張が成り立っていると仮定します．

まず，$M = (M_0\ M_1\ \ldots\ M_n)$ だった場合を考えます．さらに，M_0 が値式でなかった場合を考えます．このとき，形式上の評価文脈の定義より，M_0 を超えて M_1〜M_n とそれらの内部にホールが入ることはできません．従って，このときには M_0 の内部にホールが入ります．帰納法の仮定より，M_0 に対して主張の条件を満たす評価文脈（つまり，CBV-評価文脈）E_0 が一意的に定まります．このとき，形式上の評価文脈の定義より，

$E = (E_0\ M_1\ \ldots\ M_n)$

は M の評価文脈になります．これは一意性以外の主張の条件（つまり，ホールは CBV-簡約基と置き換えていること）を満たします．なぜなら E_0 が M_0 の CBV-評価文脈であることから，E_0 のホール（それは E のホールでもあります）は CBV-簡約基を置き換えたものだからです．あとは一意性を示せばよいので，いま仮に

$E' = (E'_0\ M_1\ \ldots\ M_n)$

を（一意性以外の）主張の条件を満たす M の任意の評価文脈とします．ホールは M_0 の内部にしか入らないので，いまの評価文脈はこの形式になります．さらに，E' が主張の条件（つまり，ホールは CBV-簡約基と置き換えていること）を満たすとすると，E'_0 のホールも明らかにその条件を満たします（なぜなら，E' と E'_0 のホールは同じだからです）．つまり，E'_0 は M_0 の CBV-評価文脈になります．従って，M_0 の CBV-評価文脈の一意性より $E'_0 = E_0$ となり，結局，$E' = E$ となります．

次に，M_0 が値式だった場合を考えます．さらに，M_0〜M_{k-1} はすべて値式で，M_k が値式でなかったとします．このとき，帰納法の仮定より，M_k の CBV-評価文脈 E_k が一意的に定まります．さらに，

$E = (M_0\ \ldots\ M_{k-1}\ E_k\ M_{k+1}\ \ldots\ M_n)$

とおくと，これは M の評価文脈になり，M_0〜M_{k-1} 自身およびその内部にホールが入ることはなく，M_k の右側にもホールが入ることはないので，M の評価文脈はすべてこの形式になります．さらに，E_k の一意性より E の一意性を導くことができます（この議論は前段落とまったく同じです）．

今度は，$M = (\texttt{if}\ M_1\ M_2\ M_3)$ だった場合を考えます．M は CBV-簡約基ではないので M_1 は値式ではありません．従って，帰納法の仮定より，M_1 の CBV-評価文脈 E_1 が一意的に定まります．そこで，

$E = (\texttt{if}\ E_1\ M_2\ M_3)$

とおくと，前々段落とまったく同じ論法によって，この E が M の CBV-評価文脈であることを示せます．

最後に，$M = (\texttt{let}^1\ (X\ M_1)\ M_2)$ だった場合を考えます．M は CBV-簡約基ではないので M_1 は値式ではありません．従って，帰納法の仮定より，M_1 の CBV-評価文脈 E_1 が一意的に定まります．そこで，

$E = (\texttt{let}^1\ (X\ E_1)\ M_2)$

とおくと，前々々段落とまったく同じ論法によって，この E が M の CBV-評価文脈であることを示せます．

だいぶ端折りましたが，これくらいにしておきます．

更新履歴

- 2020.10.30　勉強始め.
- 2020.02.04　Github に保存.