ソフトウェア基礎論配布資料 (4)
型なしλ計算

五十嵐 淳
京都大学 大学院情報学研究科知能情報学専攻
e-mail: igarashi@kuis.kyoto-u.ac.jp

平成 18 年 10 月 30 日

1 関数とλ記法，λ計算

プログラムにおいて，似たような式・計算手順が複数の箇所で必要になった時には，関数や手続きを定義して，式・手順の再利用を図ることが一般的である．数学でも

$$2^2\pi + 7^2\pi + 20^2\pi$$

と書く代わりに，

$$f(2) + f(7) + f(20) \quad \text{ただし } f(x) = x^2\pi$$

と書けば，式の見通しがよくなる．ここで x はパラメータと呼ばれ，使われる場所によって異なる部分を表す役割を担っている．

この f のように，関数の概念は，集合論で扱われる「入力と出力の対の集合」という扱いよりも,「入力から出力を計算する式」とする扱いの方がより直感的かもしれない．このような「計算可能な関数」を扱うための理論がλ**計算**(λ-*calculus*) であり，その中心となるのがλ記法と呼ばれる関数の記法である．

λ 記法は，$\lambda\langle$ パラメータ $\rangle.\langle$ 式 $\rangle$ という形で「パラメータを入力として式の計算結果を出力とする関数」を表現する．上の例の f は $\lambda x.x^2\pi$ と書くことができる．この λ 記法の特徴的な点は，

その関数自体に名前をつけずに関数を表現することができる

という点である．これにより，関数の概念そのものと関数に名前をつけるという行為を切離すことができる．

λ 記法による関数に，その引数を与えた時 (関数を適用する，という) 関数の値は「パラメータを引数で置き換えること」で得ることができる．すなわち，$f = \lambda x.x^2\pi$ とすると，

$$\begin{aligned} & f(2) + f(7) + f(20) \\ (\text{定義より}) \quad &= (\lambda x.x^2\pi)(2) + f(7) + f(20) \\ (x \text{ を } 2 \text{ で置き換え}) \quad &= 2^2\pi + f(7) + f(20) \\ &\vdots \\ &= 453\pi \end{aligned}$$

という計算が可能になる．λ 計算の体系は，λ 記法による関数，関数適用によるパラメータ置換の仕組みを形式的に実現したものであり，特に，パラメータ置換こそが計算ステップである，という立場をとる．

1.1 なぜ λ 計算なのか？

- 高級プログラミング言語にとって，関数・手続きの仕組みは必須のものであり，λ 計算は関数呼び出しの仕組みのモデルである．
- 様々なプログラム機構が λ 計算の枠組で「表現できる」．
- チューリング機械より単純，かつプログラミング言語に近い計算モデル．

上の例では，π という実数や乗算を仮定していたが，純粋な λ 計算は，通常のプログラミング言語に見られるような整数などの基本的なデータ (と，その上の演算) すら取り扱わない，関数と関数適用しかない言語になっている．純粋な体系上に，基本的なデータ型 (例えば自然数や真偽値) を付加することは容易であるので，後で扱うことにする．

2 λ 項の定義

λ 計算では，プログラムに対応する対象を λ **項**(λ *term*) と呼ぶ．λ 項のためのメタ変数として t を使用する．上で見たように λ 記法には関数パラメータである変数が現れるので，ここでは前章で扱った変数宣言・定義の仕組みを最初から取り入れるのが自然である．そこで，$\Gamma \vdash t \in \mathbf{Term}$ という判断を考える．

変数・定義　変数・定義の取り扱いは，図 1 に規則を示すように，算術式の時とほぼ同様である．

$$\frac{}{\bullet \in \mathbf{Env}} \quad (\textsc{Env-Empty}) \qquad \frac{\Gamma \vdash \#^i x \in \mathbf{Term}}{\Gamma, x \vdash \#^{i+1} x \in \mathbf{Term}} \quad (\textsc{Tm-Var2})$$

$$\frac{\Gamma \in \mathbf{Env}}{\Gamma, x \in \mathbf{Env}} \quad (\textsc{Env-Decl}) \qquad \frac{\Gamma \vdash \#^i x \in \mathbf{Term} \qquad \Gamma \vdash t \in \mathbf{Term}}{\Gamma, x = t \vdash \#^{i+1} x \in \mathbf{Term}} \quad (\textsc{Tm-Var3})$$

$$\frac{\Gamma \in \mathbf{Env} \qquad \Gamma \vdash t \in \mathbf{Term}}{\Gamma, x = t \in \mathbf{Env}} \quad (\textsc{Env-Defn}) \qquad \frac{\Gamma \vdash \#^i x \in \mathbf{Term} \qquad (x \neq y)}{\Gamma, y \vdash \#^i x \in \mathbf{Term}} \quad (\textsc{Tm-Var4})$$

$$\frac{\Gamma \in \mathbf{Env}}{\Gamma, x \vdash \#^0 x \in \mathbf{Term}} \quad (\textsc{Tm-Var0})$$

$$\frac{\Gamma \in \mathbf{Env} \qquad \Gamma \vdash t \in \mathbf{Term}}{\Gamma, x = t \vdash \#^0 x \in \mathbf{Term}} \quad (\textsc{Tm-Var1}) \qquad \frac{\Gamma \vdash \#^i x \in \mathbf{Term} \qquad \Gamma \vdash t \in \mathbf{Term} \qquad (x \neq y)}{\Gamma, y = t \vdash \#^i x \in \mathbf{Term}} \quad (\textsc{Tm-Var5})$$

図 1: λ 項の定義 (抜粋)

関数適用　λ 計算では伝統的に関数 t_1 の引数 t_2 への適用を $t_1\ t_2$ と引数 t_2 に括弧をつけずに記述する．($\sin \pi$ などの記法と同じ!) ただし，算術式の時と同様，結合を示すためのメタな括弧 () が使われることはある．

$$\frac{\Gamma \vdash t_1 \in \mathbf{Term} \qquad \Gamma \vdash t_2 \in \mathbf{Term}}{\Gamma \vdash t_1\ t_2 \in \mathbf{Term}} \quad (\textsc{Tm-App})$$

この形の項を**関数適用**(*function application*) と呼ぶ．

λ 抽象　λ 記法による関数は λ 計算では **λ 抽象**(*λ-abstraction*) と呼ぶ．$\lambda x.t$ において，$\lambda x.$ は t を有効範囲とする変数 x の局所的な宣言であると考えられる．例えば $\bullet \vdash \lambda x.x \in \mathbf{Term}$ はもっともらしい判断であるが，$\bullet \vdash x(\lambda x.x) \in \mathbf{Term}$ は，最初の変数参照 x は $\lambda x.$ の宣言の有効範囲にもなく，環境も空であるので，導出されない判断である．一方，$x \vdash x(\lambda x.x) \in \mathbf{Term}$ や $x \vdash x(\lambda x.\#^1 x)$ はもっともらしい判断である．

λ 抽象のための規則は以下のように与えられる．

$$\frac{\Gamma, x \vdash t \in \mathbf{Term}}{\Gamma \vdash \lambda x.t \in \mathbf{Term}} \quad (\textsc{Tm-Abs})$$

つまり，Γ の下で $\lambda x.t$ が λ 項であるのは，Γ, x と局所変数宣言を加えた環境の下で t が λ 項であるときである．

2.1　メタ括弧の省略

λ 計算では，結合を示すための (メタな) 括弧を以下の要領で省略する．

- 関数適用同士は左結合する．つまり $t_1\ t_2\ t_3 \equiv (t_1\ t_2)\ t_3$ である．
- λ 抽象は可能な限り右に伸びる．つまり，$\lambda x.t_1\ t_2 \equiv \lambda x.(t_1\ t_2) \not\equiv (\lambda x.t_1)\ t_2$ である．

2.1.1 練習問題: 判断 $\bullet \vdash \lambda x.\lambda y.\lambda z.x\ y\ z \in \mathbf{Term}$ を導出せよ．

$\Gamma \vdash t \in \mathbf{Term}$ なる t 中の変数 (参照) で，Γ 内の宣言を参照しているものを自由変数 (参照) ということがある．また，$\lambda x.$ による局所的宣言そのものと，それを参照している変数参照を束縛変数ということがある．

3 λ計算の表現力

純粋な λ 計算は，一見すると余りにも貧弱にみえる．例えば，

- 関数呼び出しのモデルといいつつ，引数がふたつ以上ある関数すらない．
- 変数は全て関数パラメータで局所変数の定義すらできない．
- 再帰的関数が考えられていない．
- 基本的なデータ型すら入っていない．

などといった点に思いあたる．しかし，これらは実は全て λ 計算の中で表現できることである．以下では，こういった現実的な言語に備わっている様々な要素が，純粋な λ 計算の枠組みで表現できることを見る．

3.1 複数の引数をもつ関数と Curry 化

$$x \text{ と } y \text{ をパラメータとして } t \text{ を計算する関数}$$
$$= x \text{ をパラメータとして,「} y \text{ をパラメータとして } t \text{ を計算する関数」を返す関数}$$

3.2 Church Boolean

$$\begin{aligned} \texttt{true} &= \lambda\texttt{x}.\lambda\texttt{y}.\texttt{x} \\ \texttt{false} &= \lambda\texttt{x}.\lambda\texttt{y}.\texttt{y} \\ \texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 &\stackrel{\text{def}}{=} t_1\ t_2\ t_3 \end{aligned}$$

3.3 Pairing

$$
\begin{array}{rcl}
\langle t_1, t_2 \rangle & \stackrel{\text{def}}{=} & \lambda \mathtt{x.x}\ t_1\ t_2 \\
\mathtt{fst} & = & \lambda \mathtt{p.p}\ (\lambda \mathtt{f.}\lambda \mathtt{s.f}) \\
\mathtt{snd} & = & \lambda \mathtt{p.p}\ (\lambda \mathtt{f.}\lambda \mathtt{s.s})
\end{array}
$$

3.4 Church 数

$$
\begin{array}{rcl}
\mathtt{0} & \stackrel{\text{def}}{=} & \lambda \mathtt{s.}\lambda \mathtt{z.z} \\
n & \stackrel{\text{def}}{=} & \lambda \mathtt{s.}\lambda \mathtt{z.}\underbrace{\mathtt{s}\ (\mathtt{s}\ (\cdots\ \mathtt{s}}_{n}(\mathtt{z})\cdots)) \\
\mathtt{succ} & = & \lambda \mathtt{n.}\lambda \mathtt{s.}\lambda \mathtt{z.s}\ (\mathtt{n\ s\ z}) \\
\mathtt{plus} & = & \lambda \mathtt{n.}\lambda \mathtt{m.}\lambda \mathtt{s.}\lambda \mathtt{z.n\ s}\ (\mathtt{m\ s\ z}) \\
\mathtt{times} & = & \lambda \mathtt{n.}\lambda \mathtt{m.}\lambda \mathtt{s.}\lambda \mathtt{z.n}\ (\mathtt{m\ s})\ \mathtt{z} \\
\mathtt{pred} & = & \lambda \mathtt{n.fst}\ (\mathtt{n}\ (\lambda \mathtt{p.}\langle \mathtt{snd\ p, succ}\ (\mathtt{snd\ p})\rangle)\ \langle \mathtt{0}, \mathtt{0}\rangle)
\end{array}
$$

3.5 再帰的関数

$$
\begin{array}{rcl}
\mathtt{omega} & = & (\lambda \mathtt{x.x\ x})\ (\lambda \mathtt{x.x\ x}) \\
\mathtt{fix} & = & \lambda \mathtt{f.}((\lambda \mathtt{x.f}\ (\mathtt{x\ x}))\ (\lambda \mathtt{x.f}\ (\mathtt{x\ x})))
\end{array}
$$

$f(x) = e$ (式 e には f が出現) なる再帰的関数は $\mathtt{fix}\ (\lambda \mathtt{f.}\lambda \mathtt{x.e})$ と表現される.

ここで紹介した計算の過程は，どうもプログラミムの実際の実行の様子とは少しギャップがあるようである．例えば，関数の中身は実際に呼び出されなくても計算しているし，計算はできるところどこからでも進めている．ここで挙げているのは，むしろ，プログラムの実行というよりは，簡約によるプログラムの変形をしていると思った方がよいだろう.

以下では，よりプログラムの実行に近い形の計算手順を評価関係として定義し，上に示した簡約が，評価関係から導かれる文脈同値関係を保存することを算術式と同様に示す.

$$\frac{\Delta \vdash \#^i x \Downarrow v \qquad \Delta \vdash t' \in \mathbf{Term}}{\Delta, y = t' \vdash (\#^i x \uparrow_y) \Downarrow (v \uparrow_y)} \quad \text{(E-Shift)}$$

$$\frac{\Delta \vdash t \Downarrow v}{\Delta, x = t \vdash x \Downarrow (v \uparrow_x)} \quad \text{(E-Def)} \qquad \frac{}{\Delta \vdash \lambda x.t \Downarrow \lambda x.t} \quad \text{(E-Abs)}$$

図 2: 評価関係 (共通規則)

4 評価関係

まず，何がプログラムの評価結果，すなわち，値であるかを定義する．λ 計算は，関数に関する計算体系であることからも想像がつくように値は関数抽象である．関数抽象をメタ変数 v で表す.

λ 項に対する shift 操作の定義は算術式のそれよりも，項の中に変数参照が含まれている可能性があるために多少複雑になる．復習すると， $t\uparrow_x$ は，Γ の下での式 t を Γ, x という環境で見直したものである.

例えば $(x\ y)\uparrow_x \equiv \#^1 x\ y$ であったり，$(\lambda x.x\ y)\uparrow_y \equiv \lambda x.x\ \#^1 y$ であるが，$(\lambda x.x\ y)\uparrow_x \equiv \lambda x.x\ y$ となるべきである．また，$(\lambda x.\#^1 x\ y)\uparrow_x \equiv \lambda x.\#^2 x\ y$ となるべきである.

より一般的に $t\uparrow_x^i$ という形で shift 関数を定義する.

$$\begin{array}{lcll}
\#^i x \uparrow_x^j & \equiv & \#^{i+1} x & \text{if } i \geq j \\
\#^i x \uparrow_y^j & \equiv & \#^i x & \text{if } i < j \text{ or } x \not\equiv y \\
(\lambda x.t_0)\uparrow_x^j & \equiv & \lambda x.(t_0 \uparrow_x^{j+1}) & \\
(\lambda x.t_0)\uparrow_y^j & \equiv & \lambda x.(t_0 \uparrow_y^j) & \text{if } x \not\equiv y \\
(t_1\ t_2)\uparrow_x^j & \equiv & (t_1 \uparrow_x^j)\ (t_2 \uparrow_x^j) &
\end{array}$$

以降，$t\uparrow_x$ は $t\uparrow_x^0$ の略記とする.

変数や値に関する評価規則は shift 関数を使うことを除けば，算術式の場合と同じ要領で定義することができる.

4.1 関数適用と定義参照の一括解決

関数適用項 $t_1\ t_2$ が与えられた時どのように計算をすべきかを考える．まず，t_1 の値がわからないと関数の呼びようがないので，t_1 を計算する．引数の取り扱いについては，二種類の選択肢が考えられる.

- 引数 t_2 の値 v_2 を計算してから，関数を呼び出す.
- t_2 には触れずに，いきなり関数を呼び出す.

前者が，多くのプログラミング言語で使われている**値呼び**(*call by value*) と呼ばれる関数・手続き呼び出しの実行方式である．後者は，**名前呼び**(*call by name*) と呼ばれ，Algol など

$$\frac{\Gamma \vdash t \in \mathbf{Term}}{\Gamma, x = t \vdash \#^0 x[\#^0 x] \Rightarrow t\uparrow_x} \quad \text{(S-Def1)}$$

$$\frac{\Gamma \vdash t \in \mathbf{Term} \qquad (x \not\equiv y \text{ or } i \neq j)}{\Gamma, x = t \vdash \#^j y[\#^i x] \Rightarrow \#^j y} \quad \text{(S-Def2)}$$

$$\frac{\Gamma \vdash \#^i x[\#^i x] \Rightarrow t'}{\Gamma, z \vdash (\#^i x\uparrow_z)[\#^i x\uparrow_z] \Rightarrow t'\uparrow_z} \quad \text{(S-Shift1)}$$

$$\frac{\Gamma \vdash t'' \in \mathbf{Term} \qquad \Gamma \vdash \#^i x[\#^i x] \Rightarrow t'}{\Gamma, z = t'' \vdash (\#^i x\uparrow_z)[\#^i x\uparrow_z] \Rightarrow t'\uparrow_z} \quad \text{(S-Shift2)}$$

$$\frac{\Gamma \vdash t_1[\#^i x] \Rightarrow t_1' \qquad \Gamma \vdash t_2[\#^i x] \Rightarrow t_2'}{\Gamma \vdash t_1\ t_2[\#^i x] \Rightarrow t_1'\ t_2'} \quad \text{(S-App)}$$

$$\frac{\Gamma, y \vdash t_0[\#^i x\uparrow_y] \Rightarrow t_0'}{\Gamma \vdash \lambda y.t_0[\#^i x] \Rightarrow \lambda y.t_0'} \quad \text{(S-Abs)}$$

図 3: 定義参照の一括解決

で採用されている．名前呼びは関数の引数の計算を (本当に必要になるまで) lazy にしていると考えられる．Haskell という関数型言語では call by name の変種が使われている．

ここでは，まず名前呼び方式を定義するが，どちらの呼び出し方法にしても，関数呼び出しとは，パラメータである x を実際の引数である t_2 と定義して t_0 を計算，つまり，$\Gamma, x = t_2$ という環境の下で t_0 を計算していくことに相当する．

このことを踏まえたのが以下の規則である．

$$\frac{\Delta \vdash t_1 \Downarrow \lambda x.t_0 \qquad \Delta, x = t_2 \vdash t_0 \Downarrow v_0}{\Delta \vdash t_1\ t_2 \Downarrow v_0} \quad \text{(BadE-AppN)}$$

しかし，この規則ではまだうまく定義できていない．この規則からは，例えば，

$$\bullet \vdash (\lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{x})\ (\lambda \mathtt{z}.\mathtt{z}) \Downarrow \lambda \mathtt{y}.\mathtt{x}$$

が導出されるが，$\bullet \vdash \lambda \mathtt{y}.\mathtt{x} \in \mathbf{Term}$ は成立しないし，計算結果としてもおかしなものである．何が間違っていたかというと，$\lambda \mathtt{y}.\mathtt{x}$ は環境 $\bullet, \mathtt{x} = \lambda \mathtt{z}.\mathtt{z}$ の下での式なのであるのに，関数本体実行中の一時的な定義である $\mathtt{x} = \lambda \mathtt{z}.\mathtt{z}$ をいきなり消してしまったことにある．

この問題を解消するために，一時的な定義は，計算終了時に全て項の中に展開する，という方法をとる．つまり，$\bullet, \mathtt{x} = \lambda \mathtt{z}.\mathtt{z}$ と $\lambda \mathtt{y}.\mathtt{x}$ から $\bullet$ の下での式 $\lambda \mathtt{y}.\lambda \mathtt{z}.\mathtt{z}$ を得て，

$$\bullet \vdash (\lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{x})\ (\lambda \mathtt{z}.\mathtt{z}) \Downarrow \lambda \mathtt{y}.\lambda \mathtt{z}.\mathtt{z}$$

という関係が成立するようにする．

この「まとめて定義を展開する」ことを規則で定義する．「Γ の下での項 t 中の全ての定義参照 $\#^i x$ を解決した項は t' である」という判断を $\Gamma \vdash t[\#^i x] \Rightarrow t'$ と書く．この判断の導出規則は図 3 で与えることができる．

このようにして，まとめて定義参照の解決を定義することができるが，得られた項は最早，定義 $x = t_1$ を参照していないので，$\Gamma, x = t_1$ という環境のもとでの項 t を Γ の下での表現に直すことができる．これは，逆 shift 関数とも呼べるもので，$t\downarrow_x^i$ と表記し，以下のよう

に定義する．

$$
\begin{array}{lcll}
\#^i x\downarrow_x^j & \equiv & \#^{i-1}x & \text{if } i \geq j \text{ and } i > 0 \\
\#^i x\downarrow_y^j & \equiv & \#^i x & \text{if } i < j \text{ or } x \not\equiv y \\
\lambda x.t_0\downarrow_x^j & \equiv & \lambda x.(t_0\downarrow_x^{j+1}) & \\
\lambda x.t_0\downarrow_y^j & \equiv & \lambda x.(t_0\downarrow_y^j) & \text{if } x \not\equiv y \\
(t_1\ t_2)\downarrow_x^j & \equiv & (t_1\downarrow_x^j)\ (t_2\downarrow_x^j) &
\end{array}
$$

$t\downarrow_x^0$ のことを $t\downarrow_x$ と略記する．

これらを使って，関数適用の規則は改めて以下のように定義できる．

$$
\frac{\Delta\vdash t_1\Downarrow\lambda x.t_0 \qquad \Delta, x=t_2\vdash t_0\Downarrow v_0 \qquad \Delta, x=t_2\vdash v_0[\#^0 x]\Rightarrow v}{\Delta\vdash t_1\ t_2\Downarrow(v\downarrow_x)}\ \text{(E-AppN)}
$$

名前呼び評価関係 $\Delta\vdash t\Downarrow_n v$ は，E-Def, E-Shift, E-Abs, E-AppN によって，$\Delta\vdash t\Downarrow v$ が導出されることとして定義する．(添字の n は後で値呼び評価関係 $\Downarrow_v$ と区別するための記号である．)

文脈同値関係を定義する前に，まず，文脈の定義を行う．算術式と同様，文脈は穴をひとつだけ含むような項である．BNF で書くと

$$C ::= [\,] \mid \lambda x.C \mid C\ t \mid t\ C$$

である．厳密には，項と同じように，環境の下での文脈を，$\Gamma\vdash C\in\mathbf{Ctx}[\Gamma']$ という判断を考えて定義する．ここで Γ' は，穴の環境，つまり，後で穴に埋め込まれる項が参照できる変数の情報である．例えば，$\bullet\vdash\lambda x.x[\,]\in\mathbf{Ctx}[\bullet, x]$ という判断が導出されることを想定している．穴は $\lambda x.$ の中にあるかもしれないので，参照できる変数が Γ より多い可能性があるわけである．

さて，文脈同値の定義を行う．λ 計算の場合，評価結果の値自体はさほど重要ではなく，計算が止まるかどうかに主な興味があるので，文脈同値関係は「どんな文脈に置いて実行しても，両方計算が止まるか，両方とも止まらないかである」時に成立すると定義される．

4.1.1 定義: Γ の下での式 t_1 と t_2 が**名前呼び文脈同値**であること $\Gamma\vdash t_1\cong_n t_2$ は以下の条件が成立することと同値である．

> $\Gamma\leq\Delta$ かつ $\Delta\vdash C\in\mathbf{Ctx}[\Gamma]$ が成立する任意の Δ, C に対して，$(\exists v_1.\Delta\vdash C[t_1]\Downarrow_n v_1)\iff(\exists v_2.\Delta\vdash C[t_2]\Downarrow_n v_2))$ である．

5 簡約関係

簡約に関しては，算術式と同様に環境の下で考え $\Gamma\vdash t\longrightarrow_n t'$ と記述される．定義された変数に関する簡約規則は $t\uparrow_x$ の定義の変更以外は算術式の時と同じである．また，部分項を簡約するための規則は値呼び・名前呼びに共通なのでまとめて図 4 に掲げる．

$$\frac{\Gamma \vdash t \in \mathbf{Term}}{\Gamma, x = t \vdash x \longrightarrow t\uparrow_x} \text{(R-Def)} \qquad \frac{\Gamma, x \vdash t \longrightarrow t'}{\Gamma \vdash \lambda x.t \longrightarrow \lambda x.t'} \text{(R-Lam)}$$

$$\frac{\Gamma \vdash \#^i x \longrightarrow t}{\Gamma, y \vdash \#^i x\uparrow_y \longrightarrow t\uparrow_y} \text{(R-Shift1)} \qquad \frac{\Gamma \vdash t_1 \longrightarrow t_1'}{\Gamma \vdash t_1\ t_2 \longrightarrow t_1'\ t_2} \text{(R-App1)}$$

$$\frac{\Gamma \vdash \#^i x \longrightarrow t \qquad \Gamma \vdash t' \in \mathbf{Term}}{\Gamma, y = t' \vdash \#^i x\uparrow_y \longrightarrow t\uparrow_y} \text{(R-Shift2)} \qquad \frac{\Gamma \vdash t_2 \longrightarrow t_2'}{\Gamma \vdash t_1\ t_2 \longrightarrow t_1\ t_2'} \text{(R-App2)}$$

図 4: 簡約関係 (共通規則)

5.1 β 簡約

最初に述べたように，関数を具体的な引数に適用した値は,「パラメータを引数で置き換えること」で得ることができる．つまり，$(\lambda x.t_0)\ t_1$ の計算結果は，t_0 中の x を t_1 に「置き換え」たような項 t' の計算結果である．このパラメータ置換を簡約として表現したのが以下の簡約関係

$$\Gamma \vdash (\lambda x.t_0)\ t_1 \longrightarrow t'$$

である．この簡約過程を β **簡約**(β-*reduction*)，置き換え操作を**代入**(*substitution*) と呼び，この左辺の形の項を β**(簡約) 基**(β-*redex*) と呼ぶ.

今，β 基の環境を Γ とする．つまり， $\Gamma \vdash (\lambda x.t_0)\ t_1 \in \mathbf{Term}$ であるとする． このとき，$\Gamma, x \vdash t_0 \in \mathbf{Term}$ であることに注意すると， 関数呼び出しとは，パラメータである x を実際の引数である t_1 と定義して t_0 を計算，つまり，$\Gamma, x = t_1$ という 環境の下で t_0 を計算していくことに相当する． すると，代入操作は，t_0 中の全ての x への定義参照をまとめて解決することと考えられる．これをふまえて β 簡約は,

$$\frac{\Gamma, x = t_2 \vdash t_1[\#^0 x] \Rightarrow t'}{\Gamma \vdash (\lambda x.t_1)\ t_2 \longrightarrow t'\downarrow_x} \text{(R-Beta)}$$

と定義できる.

5.1.1 定義 [名前呼び簡約関係]: 判断 $\Gamma \vdash t \longrightarrow_n t'$ は図 4 の共通規則と R-Beta により定義される．□

5.1.2 練習問題: 以下の判断を導出せよ:

- $\mathtt{v}, \mathtt{w} \vdash (\lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{x})\ \mathtt{v}\ \mathtt{w} \longrightarrow_n (\lambda \mathtt{y}.\mathtt{v})\ \mathtt{w}$
- $\mathtt{v}, \mathtt{w} \vdash (\lambda \mathtt{y}.\mathtt{v})\ \mathtt{w} \longrightarrow_n \mathtt{v}$
- $\mathtt{x}, \mathtt{y}, \mathtt{f} = \lambda \mathtt{z}.\mathtt{x} \vdash (\lambda \mathtt{x}.\mathtt{f})\ \mathtt{y} \longrightarrow_n (\lambda \mathtt{x}.\lambda \mathtt{z}.\#^1 \mathtt{x})\ \mathtt{y}$

- $\mathtt{x}, \mathtt{y}, \mathtt{f} = \lambda\mathtt{y}.\mathtt{x} \vdash (\lambda\mathtt{x}.\lambda\mathtt{z}.\#^1\mathtt{x})\ \mathtt{y} \longrightarrow_n \lambda\mathtt{z}.\mathtt{x}$

λ 計算における簡約も，合流性を満たす．以下では，$\Gamma \vdash t_1 \longrightarrow_n^* t_n$ で (Γ を固定した) 簡約関係の反射推移閉包を，$\Gamma \vdash t_1 \leftrightarrow_n t_2$ で，同値閉包を示す．

5.1.3 定理 [合流性, confluence]: 任意の $\Gamma \in \mathbf{Env}$, $\Gamma \vdash t_1, t_2, t_3 \in \mathbf{Term}$ に対し，$\Gamma \vdash t_1 \longrightarrow_n^* t_2$，$\Gamma \vdash t_1 \longrightarrow_n^* t_3$ ならば，$\Gamma \vdash t_4 \in \mathbf{Term}$ なる t_4 が存在し，$\Gamma \vdash t_2 \longrightarrow_n^* t_4$ かつ $\Gamma \vdash t_3 \longrightarrow_n^* t_4$ である．

また，簡約関係を使って同値な項は文脈同値である．

5.1.4 定理: $\Gamma \vdash t_1 \leftrightarrow_n t_2$ ならば $\Gamma \vdash t_1 \cong_n t_2$ である．

しかし，この逆は，変数を導入した算術式の時とは違った根本的な理由で成立しない．実は，いくら簡約規則を補強しても，逆が成り立つようにはできない．

6 値呼び λ 計算

関数呼び出しの方法として値呼びを考えると上の結果が少し異なってくる．特に，定理 5.1.4 は成立しない．例えば，

$$\begin{aligned} t &\equiv \lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{y} \\ \Omega &\equiv (\lambda\mathtt{x}.\mathtt{x}\ \mathtt{x})\ (\lambda\mathtt{x}.\mathtt{x}\ \mathtt{x}) \end{aligned}$$

を考えると，

$$\bullet \vdash t\ \Omega \longrightarrow_n \lambda\mathtt{y}.\mathtt{y}$$

が成立するが，左辺は，値呼びの下では評価をしても値がない，つまり $\bullet \vdash t \Downarrow_v v$ なる v が存在しないのに対して，右辺はそれ自体値になっている．つまり，値呼びに関して，β 簡約は文脈同値に対して適切な簡約関係ではない，ということである．

6.1 評価関係

値呼びの下では，$x = t$ という定義も，x を t の値として定義すると考えるのが自然である．(この意味では，定義は単なるマクロとは違ってくる．) そのため，評価関係については，関数適用項の評価を変更するだけでなく,「環境中の定義を評価する」という計算過程が加わる．値呼び評価関係 $\Delta \vdash t \Downarrow_v v$ は，E-Def, E-Shift, E-Abs に以下の規則を加えて定義される．導出中に表われる判断 $\Delta \vdash t \Downarrow v$ の Δ は常に，定義の右辺が λ 抽象項であるような定義環境になることに注意．

$$\frac{}{\bullet \Downarrow \bullet} \qquad \text{(E-EmpEnv)}$$

$$\frac{\Delta \Downarrow \Delta' \qquad \Delta' \vdash t \Downarrow v}{\Delta, x = t \Downarrow \Delta', x = v} \qquad \text{(E-DefEnv)}$$

$$\frac{\Delta \vdash t_1 \Downarrow \lambda x.t_0 \qquad \Delta \vdash t_2 \Downarrow v_2 \qquad \Delta, x = v_2 \vdash t_0 \Downarrow v_0 \qquad \Delta, x = v_2 \vdash v_0[\#^0 x] \Rightarrow v}{\Delta \vdash t_1\ t_2 \Downarrow (v \downarrow_x)} \qquad \text{(E-AppV)}$$

$$\frac{\Delta \Downarrow \Delta' \qquad \Delta' \vdash t \Downarrow v}{\Delta \vdash t \Downarrow_v v} \qquad \text{(E-CBV)}$$

値呼び同値関係 $\Gamma \vdash t_1 \cong_v t_2$ は，名前呼びの時と全く同様な方法で定義することができる．

6.2 簡約関係

上の反例で見たように，β 簡約そのものは $\cong_v$ を保存しない．とはいえ，実は，引数が値を表す項の時のみ β 簡約を許すように制限すれば，$\cong_v$ を保存する簡約関係が得られる．この制限された β 簡約を β_v 簡約と呼ぶ.「値を表す項」とは，λ 抽象項と定義なしで宣言された変数である．定義なしで宣言された変数は，λ で宣言された局所的なものにしろ，環境中で宣言された大域的なものにしろ，評価時には値を指すことになるので「値を表す」と考えられるが，一方，定義された変数は，定義内容の項が値を持つことがわからない限り値とは見なせないので，一般的には「値を表す項」とは見なせない．ということで，値呼び簡約関係 $\Gamma \vdash t \longrightarrow_v t'$ は，図 4 の共通規則に加えて (R-Beta の代わりに) 以下の R-BetaV から定義される．R-BetaV で使われる補助判断 $\Gamma \vdash t \in \mathbf{VTerm}$ は「t が Γ の下で値を表す項である」ことを示している．

$$\frac{\Gamma \in \mathbf{Env}}{\Gamma, x \vdash \#^0 x \in \mathbf{VTerm}} \qquad \text{(V-Var)}$$

$$\frac{\Gamma \vdash t \in \mathbf{VTerm}}{\Gamma, x \vdash t\uparrow_x \in \mathbf{VTerm}} \qquad \text{(V-Shift)}$$

$$\frac{\Gamma \vdash \lambda x.t \in \mathbf{Term}}{\Gamma \vdash \lambda x.t \in \mathbf{VTerm}} \qquad \text{(V-Abs)}$$

$$\frac{\Gamma \vdash t_2 \in \mathbf{VTerm} \qquad \Gamma, x = t_2 \vdash t_1[\#^0 x] \Rightarrow t'}{\Gamma \vdash (\lambda x.t_1)\ t_2 \longrightarrow t' \downarrow_x} \qquad \text{(R-BetaV)}$$

6.3 諸性質

6.3.1 定理 [合流性, confluence]: 任意の $\Gamma \in \mathbf{Env}$, $\Gamma \vdash t_1, t_2, t_3 \in \mathbf{Term}$ に対し，$\Gamma \vdash t_1 \longrightarrow_v^* t_2$，$\Gamma \vdash t_1 \longrightarrow_v^* t_3$ ならば，$\Gamma \vdash t_4 \in \mathbf{Term}$ なる t_4 が存在し，$\Gamma \vdash t_2 \longrightarrow_v^* t_4$ かつ $\Gamma \vdash t_3 \longrightarrow_v^* t_4$ である．

また，簡約関係を使って同値な項は文脈同値である．

6.3.2 定理: $\Gamma \vdash t_1 \leftrightarrow_v t_2$ ならば $\Gamma \vdash t_1 \cong_v t_2$ である．

7 整数 + 真偽値をもつ値呼び λ 計算: ゲーム Untyped

7.1 動機

- Church 数表現は call-by-value reduction では，自然数との対応関係が明らかでなくなる．(`plus 1 1` $\longrightarrow_v^*$ `2` にならない．)
- すこぶる読みにくい
- `plus true 1` みたいな項でも，簡約がすすむ．(その結果，意味のわからない λ 項が得られる．)

$\Longrightarrow$ 自然数・真偽値などの基本的なデータ型を言語のプリミティブとして導入する．

7.2 定義

ここでは項の定義のみを示すのみとする．簡約などの定義は算術式や純粋な λ 計算のそれから，素直に導くことができる．項を導出する判断は純粋な λ 計算のそれを流用して $\Gamma \vdash t \in \mathbf{Term}$ という形式である．

追加された項のほとんどは説明不要であろうと思われる．`fix` は，再帰関数を定義するための機構で，`fix` $\lambda x.t$ という項で，t 中の x が再帰的に自分 (`fix` $\lambda x.t$ 全体) 自身を指すことができる．具体的には，以下のような簡約規則となる．

$$\frac{\Gamma, x = \texttt{fix}\ \lambda x.t \vdash t[\#^0 x] \Rightarrow t'}{\Gamma \vdash \texttt{fix}\ \lambda x.t \longrightarrow t' \downarrow_x} \quad \text{(R-Fix)}$$

$$\frac{}{\bullet \in \mathbf{Env}} \quad (\textsc{Env-Empty})$$

$$\frac{\Gamma \in \mathbf{Env}}{\Gamma, x \in \mathbf{Env}} \quad (\textsc{Env-Decl})$$

$$\frac{\Gamma \in \mathbf{Env} \qquad \Gamma \vdash t \in \mathbf{Term}}{\Gamma, x = t \in \mathbf{Env}} \quad (\textsc{Env-Defn})$$

$$\frac{\Gamma \in \mathbf{Env}}{\Gamma, x \vdash \#^0 x \in \mathbf{Term}} \quad (\textsc{Tm-Var0})$$

$$\frac{\Gamma \in \mathbf{Env} \qquad \Gamma \vdash t \in \mathbf{Term}}{\Gamma, x = t \vdash \#^0 x \in \mathbf{Term}} \quad (\textsc{Tm-Var1})$$

$$\frac{\Gamma \vdash \#^i x \in \mathbf{Term}}{\Gamma, x \vdash \#^{i+1} x \in \mathbf{Term}} \quad (\textsc{Tm-Var2})$$

$$\frac{\Gamma \vdash \#^i x \in \mathbf{Term} \qquad \Gamma \vdash t \in \mathbf{Term}}{\Gamma, x = t \vdash \#^{i+1} x \in \mathbf{Term}} \quad (\textsc{Tm-Var3})$$

$$\frac{\Gamma \vdash \#^i x \in \mathbf{Term} \qquad (x \not\equiv y)}{\Gamma, y \vdash \#^i x \in \mathbf{Term}} \quad (\textsc{Tm-Var4})$$

$$\frac{\Gamma \vdash \#^i x \in \mathbf{Term} \qquad \Gamma \vdash t \in \mathbf{Term} \qquad (x \not\equiv y)}{\Gamma, y = t \vdash \#^i x \in \mathbf{Term}} \quad (\textsc{Tm-Var5})$$

$$\frac{\Gamma \vdash t_1 \in \mathbf{Term} \qquad \Gamma \vdash t_2 \in \mathbf{Term}}{\Gamma \vdash t_1\ t_2 \in \mathbf{Term}} \quad (\textsc{Tm-App})$$

$$\frac{\Gamma, x \vdash t \in \mathbf{Term}}{\Gamma \vdash \lambda x.t \in \mathbf{Term}} \quad (\textsc{Tm-Abs})$$

$$\frac{\Gamma \in \mathbf{Env} \qquad (n \text{ is a natural number})}{\Gamma \vdash n \in \mathbf{Term}} \quad (\textsc{Tm-Num})$$

$$\frac{\Gamma \vdash t_1 \in \mathbf{Term} \qquad \Gamma \vdash t_2 \in \mathbf{Term}}{\Gamma \vdash t_1\ \texttt{-}\ t_2 \in \mathbf{Term}} \quad (\textsc{Tm-Minus})$$

$$\frac{\Gamma \in \mathbf{Env}}{\Gamma \vdash \texttt{true} \in \mathbf{Term}} \quad (\textsc{Tm-True})$$

$$\frac{\Gamma \in \mathbf{Env}}{\Gamma \vdash \texttt{false} \in \mathbf{Term}} \quad (\textsc{Tm-False})$$

$$\frac{\Gamma \vdash t_1 \in \mathbf{Term} \qquad \Gamma \vdash t_2 \in \mathbf{Term}}{\Gamma \vdash t_1\ \texttt{>}\ t_2 \in \mathbf{Term}} \quad (\textsc{Tm-Gt})$$

$$\frac{\Gamma \vdash t_1 \in \mathbf{Term} \qquad \Gamma \vdash t_2 \in \mathbf{Term} \qquad \Gamma \vdash t_3 \in \mathbf{Term}}{\Gamma \vdash \texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \in \mathbf{Term}} \quad (\textsc{Tm-If})$$

$$\frac{\Gamma \vdash t \in \mathbf{Term}}{\Gamma \vdash \texttt{fix}\ t \in \mathbf{Term}} \quad (\textsc{Tm-Fix})$$

図 5: 拡張 λ 項の定義