単純型付ラムダ計算とその論理

五十嵐 淳
京都大学 大学院情報学研究科 通信情報システム専攻
igarashi@kuis.kyoto-u.ac.jp

December 11, 2012

ラムダ計算 (λ-calculus) は，Scheme などのいわゆる関数型プログラミング言語の中核の機能である

変数	(一引数) 関数	関数呼び出し
`x`, `y`, ...	`(lambda (x)` ⟨式⟩`)`	`(`⟨式⟩ ⟨式⟩`)`

(のみ) をモデル化した計算体系である．プログラム (式) はラムダ計算では項 (term) と呼ばれ，上の表に対応して

変数	ラムダ抽象	関数適用
x	$\lambda x.M$	$M_1\ M_2$

と表記する．M が項を表す記号である．

ラムダ計算における計算過程は (算術の時にみたような) 簡約で表現される．ラムダ計算における主要な計算ステップは β 簡約と呼ばれ，以下のようなパターン (規則) で表される．

$$(\lambda x.M[x])N \longrightarrow M[N]$$

(表記 $M[x]$ の意味は，算術の導出規則で見たものと同じである．) これは要するに，x を仮引数・パラメータとする関数を実引数 N に適用すると，結果として，関数本体 M 中のパラメータに N を代入したものになる，ということを表している．ラムダ計算は，チューリング機械と同等の計算能力を持ち，プログラムの理論研究では主要な道具のひとつとなっている．

型付ラムダ計算 (typed λ-calculus) は，ラムダ計算に OCaml や Coq に見られるような型の概念を導入したものである．(本当は OCaml や Coq が型付ラムダ計算に基いている，というべきだが．) 型付ラムダ計算は，ラムダ計算 (特に区別が必要な場合は「型無しラムダ計算」と呼ぶ) と同様，プログラムの理論研究での主要な道具であると同時に，論理体系と計算体系の橋渡しをする「カリー・ハワードの同型対応」と呼ばれる見方を与える大事な体系である．

型付ラムダ計算には，型としてどのようなものを考えるかによって様々なバリエーションがあるが，その中でも型として，`nat`, `bool` のような原始的な型 (primitive type, 基底型 base type ということも) と関数型を考える型付ラムダ計算を単純型付ラムダ計算 (simply typed λ-calculus) と呼ぶ．これは Coq でいうと，大体 Basics.v と List.v の範囲で書いたプログラムに相当する．

ここでは単純型付ラムダ計算の定義を通じて，算術の形式化ではややいい加減に扱った簡約の定義をより厳密なかたちで与えるとともに，算術を拡張して，自然数だけでなく真偽値，関数に関する議論ができる論理体系を作る．

1 単純型付ラムダ計算

1.1 型，ラムダ項，文脈，判断の構文

$$
\begin{array}{lcl}
\boldsymbol{S}, \boldsymbol{T} & ::= & \texttt{nat} \\
& | & \texttt{bool} \\
& | & \boldsymbol{S}\ \texttt{->}\ \boldsymbol{T} \\
\\
\boldsymbol{e} & ::= & \boldsymbol{x} \\
& | & \texttt{O} \\
& | & \texttt{S} \\
& | & \texttt{match}\ \boldsymbol{e_1}\ \texttt{with O =>}\ \boldsymbol{e_2}\ \texttt{| S}\ \boldsymbol{x}\ \texttt{=>}\ \boldsymbol{e_3}\ \texttt{end} \\
& | & \texttt{true} \\
& | & \texttt{false} \\
& | & \texttt{if}\ \boldsymbol{e_1}\ \texttt{then}\ \boldsymbol{e_2}\ \texttt{else}\ \boldsymbol{e_3} \\
& | & \texttt{fun}\ \boldsymbol{x} : \boldsymbol{T}\ \texttt{=>}\ \boldsymbol{e} \\
& | & \texttt{fix}\ \boldsymbol{x}\ (\boldsymbol{y} : \boldsymbol{S}) : \boldsymbol{T}\ \texttt{:=}\ \boldsymbol{e} \\
& | & \boldsymbol{e_1}\ \boldsymbol{e_2} \\
\\
\boldsymbol{\Gamma} & ::= & \bullet \\
& | & \boldsymbol{\Gamma},\ \boldsymbol{x} : \boldsymbol{T} \\
\\
\boldsymbol{\mathcal{J}} & ::= & \boldsymbol{\Gamma} \vdash \boldsymbol{e} : \boldsymbol{T} \\
& | & \boldsymbol{e_1} \longrightarrow \boldsymbol{e_2} \\
& | & \boldsymbol{e_1} \longrightarrow^* \boldsymbol{e_2}
\end{array}
$$

- 型としては，自然数の型 `nat`，真偽値の型 `bool`，と関数型 $\boldsymbol{S}$ `->` $\boldsymbol{T}$ を考える．この $\boldsymbol{S}$ を引数型，$\boldsymbol{T}$ を返値型ということがある．
- 項としては，変数，自然数のコンストラクタと `match` による場合分け，真偽値コンストラクタと `if` による場合分け，関数 (`fun`)，再帰関数 (`fix`)，関数適用を考える．
- 関数の構文は Coq に合わせているが，多くの文献では，$\lambda \boldsymbol{x} : \boldsymbol{T}.\boldsymbol{e}$ と書かれる．
- 再帰関数 `fix` $\boldsymbol{x}$ `(`$\boldsymbol{y}$ `:` $\boldsymbol{S}$`)` `:` $\boldsymbol{T}$ `:=` $\boldsymbol{e}$ は Coq の `Fixpoint` で定義される関数に相当する．$\boldsymbol{x}$ が関数 (を再帰的に参照するため) の名前，$\boldsymbol{y}$ がパラメータ，$\boldsymbol{S}$ がパラメータの型，$\boldsymbol{T}$ が，関数本体 $\boldsymbol{e}$ の型である．

  Coq では，適用した際に必ず停止するような関数しか書けないような制限が加わっているが，ここではその制限については扱わない．そのため，

  $$\texttt{fix}\ f\ (x : \texttt{nat}) : \texttt{nat}\ \texttt{:=}\ f\ x$$

  のような自明に止まらない関数も (型がつく) 項として認められる．

- 文脈は，変数に型を与えるもので，$x : T$ (変数 x の型は T) という形の**型宣言**(type declaration) の列である．型付ラムダ計算では文脈を**型環境**(type environment) ということも多い．
- 判断は，項 e に文脈 Γ のもとで型 T がつく，という意味の**型判断**(type judgment)，もしくは **型付け判断**(typing judgment) と，項 e が項 e' に (1 ステップもしくは複数ステップで) 簡約される，という意味の判断 ($\longrightarrow$ を二項関係と考えて**簡約関係**と呼ぶことが多い) の二種類である．これらを導出規則を使って定義する．型付け判断の導出規則は，**型付け規則**typing rule と呼ばれる．

1.2 導出規則

$\boxed{\Gamma \vdash e : T}$

$$\frac{(x : T \in \Gamma)}{\Gamma \vdash x : T} \quad \text{(T-Var)}$$

$$\frac{}{\Gamma \vdash \texttt{O} : \texttt{nat}} \quad \text{(T-Zero)}$$

$$\frac{}{\Gamma \vdash \texttt{S} : \texttt{nat -> nat}} \quad \text{(T-Succ)}$$

$$\frac{\Gamma \vdash e_1 : \texttt{nat} \qquad \Gamma \vdash e_2 : T \qquad \Gamma,\ x : \texttt{nat} \vdash e_3 : T}{\Gamma \vdash \texttt{match}\ e_1\ \texttt{with O =>}\ e_2\ \texttt{|\ S}\ x\ \texttt{=>}\ e_3\ \texttt{end} : T} \quad \text{(T-Match)}$$

$$\frac{}{\Gamma \vdash \texttt{true} : \texttt{bool}} \quad \text{(T-True)}$$

$$\frac{}{\Gamma \vdash \texttt{false} : \texttt{bool}} \quad \text{(T-False)}$$

$$\frac{\Gamma \vdash e_1 : \texttt{bool} \qquad \Gamma \vdash e_2 : T \qquad \Gamma \vdash e_3 : T}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : T} \quad \text{(T-If)}$$

$$\frac{\Gamma,\ x : S \vdash e : T}{\Gamma \vdash \texttt{fun}\ x : S\ \texttt{=>}\ e : S \texttt{->} T} \quad \text{(T-Fun)}$$

$$\frac{\Gamma,\ x : S \texttt{->} T,\ y : S \vdash e : T}{\Gamma \vdash \texttt{fix}\ x\ (y : S) : T\ \texttt{:=}\ e : S \texttt{->} T} \quad \text{(T-Fix)}$$

$$\frac{\Gamma \vdash e_1 : S \texttt{->} T \qquad \Gamma \vdash e_2 : S}{\Gamma \vdash e_1\ e_2 : T} \quad \text{(T-App)}$$

- 規則 T-Match, T-If では，分岐後の項の型が等しいことを要求している．

- 規則 T-Fun によると，関数に型 S `->` T がつくのは，パラメータの型を S として (文脈に追加して)，本体式に T 型がつく時であることがわかる．再帰関数についても，x の型が全体の型と等しくなることに注意すれば，ほぼ同様である．

$\boxed{e \longrightarrow e'}$

簡約を定義する導出規則は，計算のステップを直接表す前提のない規則 (R-XXX) と，部分項に関する簡約を許すための規則 (RC-XXX) に大別できる．

$$\frac{}{\texttt{match O with O => } e_2 \texttt{ | S } x \texttt{ => } e_3 \texttt{ end} \longrightarrow e_2} \quad \text{(R-MatchZ)}$$

$$\frac{}{\texttt{match S } e_1 \texttt{ with O => } e_2 \texttt{ | S } x \texttt{ => } e_3[x] \texttt{ end} \longrightarrow e_3[e_1]} \quad \text{(R-MatchS)}$$

$$\frac{}{\texttt{if true then } e_1 \texttt{ else } e_2 \longrightarrow e_1} \quad \text{(R-IfT)}$$

$$\frac{}{\texttt{if false then } e_1 \texttt{ else } e_2 \longrightarrow e_2} \quad \text{(R-IfF)}$$

$$\frac{}{\texttt{(fun } x : T \texttt{ => } e_0[x]\texttt{) } e_1 \longrightarrow e_0[e_1]} \quad \text{(R-Beta)}$$

$$\frac{}{\texttt{(fix } x \texttt{ (} y : S\texttt{) : } T \texttt{ := } e_0[x,y]\texttt{) } e_1 \longrightarrow e_0[\texttt{fix } x \texttt{ (} y : S\texttt{) : } T \texttt{ := } e_0[x,y], e_1]} \quad \text{(R-Fix)}$$

- 場合分けに関する計算は，場合分けの対象となるデータのコンストラクタの種類に応じて規則がある．特に R-MatchS でパターンマッチがどう表現されているかに注意.
- 規則 R-Beta が，上述した β 簡約の規則を Coq の構文で書いたものである．また，R-Fix は，再帰関数の β 簡約であるが，引数だけでなく，呼出された関数全体が x に代入されていることに注意.

$$\frac{e_1 \longrightarrow e_1'}{\begin{array}{l}\texttt{match } e_1 \texttt{ with O => } e_2 \texttt{ | S } x \texttt{ => } e_3 \texttt{ end} \\ \longrightarrow \texttt{match } e_1' \texttt{ with O => } e_2 \texttt{ | S } x \texttt{ => } e_3 \texttt{ end}\end{array}} \quad \text{(RC-Match1)}$$

$$\frac{e_2 \longrightarrow e_2'}{\begin{array}{l}\texttt{match } e_1 \texttt{ with O => } e_2 \texttt{ | S } x \texttt{ => } e_3 \texttt{ end} \\ \longrightarrow \texttt{match } e_1 \texttt{ with O => } e_2' \texttt{ | S } x \texttt{ => } e_3 \texttt{ end}\end{array}} \quad \text{(RC-Match2)}$$

$$\frac{e_3 \longrightarrow e_3'}{\begin{array}{l}\texttt{match } e_1 \texttt{ with O => } e_2 \texttt{ | S } x \texttt{ => } e_3 \texttt{ end} \\ \longrightarrow \texttt{match } e_1 \texttt{ with O => } e_2 \texttt{ | S } x \texttt{ => } e_3' \texttt{ end}\end{array}} \quad \text{(RC-Match3)}$$

$$\frac{e_1 \longrightarrow e_1'}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \longrightarrow \texttt{if } e_1' \texttt{ then } e_2 \texttt{ else } e_3} \quad \text{(RC-If1)}$$

$$\frac{e_2 \longrightarrow e_2'}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \longrightarrow \texttt{if } e_1 \texttt{ then } e_2' \texttt{ else } e_3} \quad \text{(RC-If2)}$$

$$\frac{e_3 \longrightarrow e_3'}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \longrightarrow \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3'} \quad \text{(RC-If3)}$$

$$\frac{e_0 \longrightarrow e_0'}{\texttt{fun } x : T \texttt{ => } e_0 \longrightarrow \texttt{fun } x : T \texttt{ => } e_0'} \quad \text{(RC-Fun)}$$

$$\frac{e_0 \longrightarrow e_0'}{\texttt{fix } x \texttt{ (} y : S \texttt{) : } T \texttt{ := } e_0 \longrightarrow \texttt{fix } x \texttt{ (} y : S \texttt{) : } T \texttt{ := } e_0'} \quad \text{(RC-Fix)}$$

$$\frac{e_1 \longrightarrow e_1'}{e_1\ e_2 \longrightarrow e_1'\ e_2} \quad \text{(RC-App1)}$$

$$\frac{e_2 \longrightarrow e_2'}{e_1\ e_2 \longrightarrow e_1\ e_2'} \quad \text{(RC-App2)}$$

各規則は，各構文について，部分項に簡約関係が成立するなら，全体同士も簡約関係にある，ということで，これを繰り返し使うことで，大きな項の一部分が単純化される，という過程が厳密に定義できる．

以下の規則は，マルチステップの簡約 $\longrightarrow^*$ を導出規則の形で定義したものである．

$$\frac{}{e \longrightarrow^* e} \quad \text{(MR-Zero)}$$

$$\frac{e \longrightarrow e'}{e \longrightarrow^* e'} \quad \text{(MR-One)}$$

$$\frac{e \longrightarrow e' \qquad e' \longrightarrow e''}{e \longrightarrow^* e''} \quad \text{(MR-Trans)}$$

1.3 単純型付ラムダ計算に関する重要な性質

型を使うことの恩恵のひとつは「意味のないプログラム」を排除することができることにある．「意味のないプログラム」とは，データ・関数に対して想定されていない使い方をするような項で

- `0 true` のような，関数ではない値の適用
- `if (fun x : nat => ...) then ... else ...` や真偽値以外での場合わけ

が簡約の過程で発生するような項ということができる．

単純型付ラムダ計算については，以下の「型保存定理」と「前進性」という定理が成立する．

定理 1 (型保存定理) $\Gamma \vdash e : T$ かつ，$e \longrightarrow e'$ ならば，$\Gamma \vdash e' : T$ である．

項 e が値である，とは，e が S(...(O)...)，true，false，fun $x : T$ => e，もしくは，fix x ($y : S$) : T := e いずれかの形をしていることをいう．

定理 2 (前進性) $\vdash e : T$ ならば，e は何らかの値であるか，$e \longrightarrow e'$ なる e' が存在する．

これらのふたつの定理を合わせると，空の文脈の下で型を持つ項について，もし簡約が停止するならば，その結果得られた項は値であることがいえる．

2 単純型付ラムダ項に関する論理

ここに示す論理体系は算術を，自然数だけでなく，単純型付ラムダ項についての推論を行えるように拡張したものである．以下では単純型付ラムダ計算との差分を示す形で定義しているため，算術の定義と重複する部分もかなりある．主な違いは，文脈や判断についての

$\Gamma, H : P$ という形の文脈については以下の二条件が成立しているものとする．

1. $H \notin dom(\Gamma)$
2. P の自由変数は Γ で宣言されている，つまり，$FV(P) \subseteq dom(\Gamma)$

といった条件も，導出規則を使って厳密に表現されているところである．このため,「文脈 Γ のもとで P は命題である」という意味の判断

$$\Gamma \vdash P : \mathtt{Prop}$$

を考える．この判断は P が「形の整った」命題であることのみを主張していて，P が証明できる (つまり $\Gamma \vdash P$ が導出できる) ことは意味しないので注意が必要である．

例えば，$\vdash$ O = S O : Prop は導出できるが，$\vdash$ O = S O は導出できない．

2.1 式，命題，文脈，判断の構文

$$
\begin{array}{lcl}
T & ::= & \cdots \\
\\
e & ::= & \cdots \\
\\
P, Q & ::= & e_1 = e_2 \\
& | & P \to Q \\
& | & \forall x : T, P \\
\\
\Gamma & ::= & \cdots \\
& | & \Gamma, H : P \\
\\
\mathcal{J} & ::= & \cdots \\
& | & \Gamma \vdash P : \mathtt{Prop} \\
& | & \Gamma \vdash P
\end{array}
$$

2.2 導出規則 (追加・変更分)

判断 $\Gamma \vdash P$ の導出規則に関して算術から追加・変更された部分は網かけで示している.

$\boxed{\vdash \Gamma}$ $\boxed{\Gamma \vdash P : \mathtt{Prop}}$

$$\frac{}{\vdash \bullet} \tag{E-Empty}$$

$$\frac{\vdash \Gamma \qquad (x \notin dom(\Gamma))}{\vdash \Gamma,\ x : T} \tag{E-Var}$$

$$\frac{\vdash \Gamma \qquad (H \notin dom(\Gamma)) \qquad \Gamma \vdash P : \mathtt{Prop}}{\vdash \Gamma,\ H : P} \tag{E-Asmp}$$

$$\frac{\Gamma \vdash e_1 : T \qquad \Gamma \vdash e_1 : T}{\Gamma \vdash e_1 = e_2 : \mathtt{Prop}} \tag{P-Eq}$$

$$\frac{\Gamma \vdash P : \mathtt{Prop} \qquad \Gamma \vdash Q : \mathtt{Prop}}{\Gamma \vdash P \mathtt{ -> } Q : \mathtt{Prop}} \tag{P-Imp}$$

$$\frac{\Gamma,\ x : T \vdash P : \mathtt{Prop}}{\Gamma \vdash \forall x : T, P : \mathtt{Prop}} \tag{P-Forall}$$

- 規則 E-XXX は，算術での文脈に関する条件に相当する.
- 規則 P-Eq から，等号の両辺には型が同じ項が来る必要があることがわかる.

$\boxed{\Gamma \vdash P}$

$$\frac{\vdash \Gamma \qquad (H : P \in \Gamma)}{\Gamma \vdash P} \tag{Assumption}$$

$$\frac{\Gamma \vdash \forall x : T, P[x] \qquad \Gamma \vdash e : T}{\Gamma \vdash P[e]} \tag{$\forall$E}$$

$$\frac{\Gamma \vdash e : T \qquad e \longrightarrow^* e'}{\Gamma \vdash e = e'} \tag{=I}$$

$$\frac{\Gamma \vdash \mathtt{S}\ e_1 = \mathtt{S}\ e_2}{\Gamma \vdash e_1 = e_2} \tag{InjS}$$

$$\frac{\Gamma \vdash \mathtt{O} = \mathtt{S}\ e \qquad \Gamma \vdash P : \mathtt{Prop}}{\Gamma \vdash P} \tag{ContraNat}$$

$$\frac{\Gamma \vdash P[\mathtt{O}] \qquad \Gamma,\ y : \mathtt{nat},\ H : P[y] \vdash P[\,\mathtt{S}\ y\,]}{\Gamma \vdash \forall x : \mathtt{nat}, P[x]} \tag{IndNat}$$

$$\frac{\Gamma \vdash \mathtt{true} = \mathtt{false} \qquad \Gamma \vdash P : \mathtt{Prop}}{\Gamma \vdash P} \tag{ContraBool}$$

$$\frac{\Gamma \vdash P[\mathtt{true}] \qquad \Gamma \vdash P[\mathtt{false}]}{\Gamma \vdash \forall x : \mathtt{bool}, P[x]} \tag{IndBool}$$

変更はほとんど自明なものばかりだが，いくつか補足をしておく．

- 規則 ∀E: 全称量化された命題を具体化する際には変数の型に沿った項で具体化する必要がある．
- 規則 =I: ここでは簡約も導出規則で定義したので，() がとれて付帯条件ではなくなっている．
- 規則 ContraNat, ContraBool: 矛盾から導かれる命題は文脈のもとで形の整ったものである必要がある．
- 規則 IndBool: 「帰納法」の規則だが，真偽値の場合は単なる場合分けの原理である．

2.3 偶数性

$$\begin{aligned} P, Q \ ::= \ & \cdots \\ \mid \ & \mathtt{even}\ e \end{aligned}$$

$$\frac{\Gamma \vdash e : \mathtt{nat}}{\Gamma \vdash \mathtt{even}\ e : \mathtt{Prop}} \tag{P-Even}$$

$$\frac{}{\Gamma \vdash \mathtt{even}\ \mathtt{O}} \tag{Ev-I1}$$

$$\frac{\Gamma \vdash \mathtt{even}\ e}{\Gamma \vdash \mathtt{even}\ (\mathtt{S}\ (\mathtt{S}\ e))} \tag{Ev-I2}$$

$$\frac{\Gamma \vdash P[\mathtt{O}] \qquad \Gamma, x : \mathtt{nat}, H_1 : \mathtt{even}\ x, H_2 : P[x] \vdash P[\mathtt{S}\ (\mathtt{S}\ x)]}{\Gamma \vdash \forall x : \mathtt{nat}, \mathtt{even}\ x \to P[x]} \tag{IndEv}$$