型輪講
15章　Subtyping (15.1〜15.4)

尾上 浩一

平成15年6月13日

1　はじめに

本章では $\lambda_{\rightarrow}$ に対する基本的な拡張である subtyping(subtype polymorphism ともいわれる) を導入した $\lambda_{<:}$ に関して取り扱う．

subtyping はオブジェクト指向言語の特徴と考えられている．(詳細は18章を参照)

以下の流れで本章を展開する．

- subtyping がなぜ必要かということに関して述べる．
- 一般的な規則，関数型，レコード型に関する規則を $\lambda_{\rightarrow}$ に導入した，$\lambda_{<:}$ の定義を述べる．
- $\lambda_{<:}$ の性質 (progress，preservation 等) に関して述べる．
- Top 型，Bot 型に関して述べる．

2　subtyping の目的

subtyping を導入していない $\lambda_{\rightarrow}$ では，プログラマにとっては正しい振る舞いをするように思われる，多くのプログラムが型エラーとなってしまう．以下にその例を示す．

$(\lambda r : \{x : \mathsf{Nat}\}.r.x)\ \{x = 0, y = 1\}$

上の項は型付け規則 T-APP [1] より，関数の仮引数の型 ({x:Nat}) と実引数の型 ({x:Nat, y:Nat}) が一致していないため型エラーとなる．しかし，上の関数では，レコード型の引数を取り，そのフィールド x が Nat である，ということが必要とされているだけで，他のフィールドに関してはどのような制

[1] $\dfrac{\Gamma \vdash t_1 : T_{11} \rightarrow T_{12} \quad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12} : T_{12}}$

$$\frac{\Gamma \vdash t : S \quad S <: T}{\Gamma \vdash t : T} \quad \text{(T-SUB)}$$

$$S <: S \quad \text{(S-REFL)}$$

$$\frac{S <: U \quad U <: T}{S <: T} \quad \text{(S-TRANS)}$$

図 1: 一般的な規則

約もつけられていない．さらに，関数内部でフィールド x 以外のフィールドを使用していない．そのため，型 x:Nat を持つ仮引数に，型 x:Nat, y:Nat を持つ実引数を渡すことは常に安全である．

subtyping は上記のような型付けを持つ項でも型エラーが生じないようにすることを目的とする．

3 $\lambda_{<:}$ の定義

subtyping の表記方法を以下のように定める．

S <: T : S が T の subtype である．(T が S の supertype である．)

principle of safe substitution，subset semantics に関して以下に述べる．

principle of safe substitution
S <: T ならば，型 S をもつ項は型 T をもつ項が期待されるコンテキスト内で安全に利用することができる

subset semantics
S <: T ならば，S の要素を持つ集合は T の要素を持つ部分集合である．

以降，subtyping のために必要となる，型付け規則に関して述べていく．

3.1 一般的な規則

まず，一般的な規則を定義する．(図**??**) T-SUB は type relation と subtype relation の間の調整を行う．先程の例でこの規則を利用し，$\vdash \{x = 0, y = 1\} : \{x : Nat\}$ とすることができる．

また，$\lambda_{<:}$ では反射的 (S-REFL) かつ推移的 (S-TRANS) とする．これらの規則はどのような subtyping が安全かを考慮すれば，直感的には明らかで

$$\{\mathsf{l}_i : \mathsf{T}_i \ ^{i\in 1..n+k}\} \ <: \ \{\mathsf{l}_i : \mathsf{T}_i \ ^{i\in 1..n}\} \qquad \text{(S-RCDWIDTH)}$$

$$\frac{\textit{for each } i \quad \mathsf{S}_i <: \mathsf{T}_i}{\{\mathsf{l}_i : \mathsf{S}_i \ ^{i\in 1..n}\} <: \{\mathsf{l}_i : \mathsf{T}_i \ ^{i\in 1..n}\}} \qquad \text{(S-RCDDEPTH)}$$

$$\frac{\{\mathsf{k}_j : \mathsf{T}_j \ ^{j\in 1..n}\} \textit{ is a permutation of } \mathsf{l}_i : \mathsf{T}_i \ ^{i\in 1..n}\}}{\{\mathsf{k}_j : \mathsf{T}_j \ ^{j\in 1..n}\} \ <: \ \mathsf{l}_i : \mathsf{T}_i \ ^{i\in 1..n}\}} \qquad \text{(S-RCDPERM)}$$

図 2: レコード型

$$\frac{\mathsf{T}_1 <: \mathsf{S}_1 \quad \mathsf{S}_2 <: \mathsf{T}_2}{\mathsf{S}_1 \to \mathsf{S}_2 \ <: \ \mathsf{T}_1 \to \mathsf{T}_2} \qquad \text{(S-ARROW)}$$

図 3: 関数型

ある.

3.2 レコード型

次にレコード型に関して subtyping を導入していく. (図**??**)

(S-RCDWIDTH) は, " あるレコード型 R_s が別のレコード型 R_t 以上の数のフィールドを保持しているとき (ただし, R_s が R_t の全てのフィールドを持っていて, 各々の型が同じである), R_s は R_t の subtype である" ということを意味している.

(S-RCDDEPTH) は, " あるレコード型 R_s と別のレコード型 R_t の間の各々共通するフィールドが subtype の関係であるとき, R_s は R_t の subtype である" ということを意味している.

(S-RCDPERM) は" フィールドの順序の違いは型の安全性には影響しない" ということを意味している. たとえば, $\mathsf{S} = \{\mathsf{a} : \mathsf{Nat}, \mathsf{b} : \mathsf{Bool}\}$, $\mathsf{T} = \{\mathsf{b} : \mathsf{Bool}, \mathsf{a} : \mathsf{Nat}\}$ とすると, $\mathsf{S} <: \mathsf{T}$ かつ $\mathsf{T} <: \mathsf{S}$ である. しかし, $\mathsf{S} \neq \mathsf{T}$ であるため, $\lambda_{<:}$ では反対称の関係[2] は成り立たない.

したがって, $\lambda_{<:}$ は前順序[3] ではあるが, 半順序[4] ではない.

3.3 関数型

[2] $x <: y$ かつ $x :> y$ ならば $x = y$
[3] 反射的かつ推移的である
[4] 反射的かつ反対称的かつ推移的である

次に関数型に関して subtyping を導入していく．(図??)

(S-ARROW) では引数の subtype の関係 (前提の左側) が contravariant(反変的) であることに注意する．(他方，返り値 (前提の右側) に関しては covarinat(共変的) である．)

以下に例を挙げてこの定義の正しさを確かめてみる．

例：$\mathsf{R}_1 \stackrel{\text{def}}{=} \{\mathsf{x} : \mathsf{Nat}\}$ ， $\mathsf{R}_2 \stackrel{\text{def}}{=} \{\mathsf{x} : \mathsf{Nat}, \mathsf{y} : \mathsf{Nat}\}$

$\mathsf{f}_1 \stackrel{\text{def}}{=} \lambda \mathsf{r}_1 : \mathsf{R}_1.\ \{\mathsf{x} = \mathsf{r}_1.\mathsf{x},\ \mathsf{y} = \mathsf{r}_1.\mathsf{x}\} \quad (\mathsf{f}_1 : \mathsf{R}_1 \to \mathsf{R}_2)$

$\mathsf{f}_2 \stackrel{\text{def}}{=} \lambda \mathsf{r}_2 : \mathsf{R}_2.\ \{\mathsf{x} = \mathsf{r}_2.\mathsf{x},\ \mathsf{y} = \mathsf{r}_2.\mathsf{y}\} \quad (\mathsf{f}_2 : \mathsf{R}_2 \to \mathsf{R}_2)$

$\mathsf{f}_3 \stackrel{\text{def}}{=} \lambda \mathsf{r}_3 : \mathsf{R}_2.\ \{\mathsf{x} = \mathsf{r}_3.\mathsf{x}\} \quad (\mathsf{f}_2 : \mathsf{R}_2 \to \mathsf{R}_1)$

レコード型，関数型の subtyping の規則より，$\mathsf{f}_1 <: \mathsf{f}_2$，$\mathsf{f}_2 <: \mathsf{f}_3$ となる．各々の場合について考えてみる．

- f_1と f_2の関係 (左側の前提 $\mathsf{T}_1 <: \mathsf{S}_1$)
  - $\mathsf{f}_1 <: \mathsf{f}_2$ である場合
    $\mathsf{f}_2\ \{\mathsf{x} = 1,\ \mathsf{y} = 0\}$ において f_2 を f_1 で置き換えた項 $\mathsf{f}_1\ \{\mathsf{x} = 1,\ \mathsf{y} = 0\}$ はうまく評価できる．
  - $\mathsf{f}_2 <: \mathsf{f}_1$ である場合
    $\mathsf{f}_1\ \{\mathsf{x} = 1\}$ において f_1を f_2 で置き換えた項 $\mathsf{f}_2\ \{\mathsf{x} = 1\}$ は f_2 ではフィールド y を参照しているが，引数としてフィールド y の情報が渡されないのでうまく評価できない．
- f_2と f_3の関係 (右側の前提 $\mathsf{S}_2 <: \mathsf{T}_2$)
  - $\mathsf{f}_2 <: \mathsf{f}_3$ である場合
    $(\mathsf{f}_3\ \{\mathsf{x} = 1,\ \mathsf{y} = 0\}).\ \mathsf{x}$ において $\mathsf{f}_3\ \{\mathsf{x} = 1,\ \mathsf{y} = 0\}$ を $\mathsf{f}_2\ \{\mathsf{x} = 1,\ \mathsf{y} = 0\}$ で置き換えた項 $(\mathsf{f}_2\ \{\mathsf{x} = 1,\ \mathsf{y} = 0\}).\ \mathsf{x}$ はうまく評価できる．
  - $\mathsf{f}_3 <: \mathsf{f}_2$ である場合
    $(\mathsf{f}_2\ \{\mathsf{x} = 1,\ \mathsf{y} = 0\}).\ \mathsf{y}$ において $\mathsf{f}_2\ \{\mathsf{x} = 1,\ \mathsf{y} = 0\}$ を $\mathsf{f}_3\ \{\mathsf{x} = 1,\ \mathsf{y} = 0\}$ で置き換えた項 $(\mathsf{f}_3\ \{\mathsf{x} = 1,\ \mathsf{y} = 0\}).\ \mathsf{y}$ は $\mathsf{f}_3\ \{\mathsf{x} = 1,\ \mathsf{y} = 0\}$ を評価した結果がフィールド y を含んでいないのでうまく評価できない．

4 $\lambda_{<:}$ の性質

$\lambda_{\to}$ で成り立つ性質が $\lambda_{<:}$ に関する性質でも成り立つこと (特に preservation と progress) をいくつかの補題導入し，以下で述べる．各々の証明に関しては付録に記述する．

まず，2 つの補題に関して述べる．

4.1 補題 (inversion of the subtype relation)

1. $\mathsf{S} <: \mathsf{T}_1 \to \mathsf{T}_2$ ならば, $\mathsf{S} \equiv \mathsf{S}_1 \to \mathsf{S}_2$ $(\mathsf{T}_1 < \mathsf{S}_1$ かつ $\mathsf{S}_2 < \mathsf{T}_2)$
2. $\mathsf{S} <: \{\mathsf{l}_i : \mathsf{T}_i \ ^{i \in 1..n}\}$ ならば, $\mathsf{S} \equiv \{\mathsf{k}_j : \mathsf{S}_j \ ^{i \in 1..m}\}$ $(\{\mathsf{k}_j : \mathsf{S}_j \ ^{i \in 1..m\}}\} \subseteq \{\mathsf{k}_j : \mathsf{S}_j \ ^{i \in 1..m}\}$ かつ $\mathsf{S}_j <: \mathsf{T}_i (\mathsf{l}_i = \mathsf{k}_j))$

4.2 補題 ＊

1. $\Gamma \vdash \lambda \mathsf{x} : \mathsf{S}_1.\ \mathsf{s}_2 \ : \ \mathsf{T}_1 \to \mathsf{T}_2$ ならば, $\mathsf{T}_1 <: \mathsf{S}_1$ かつ $\Gamma, \mathsf{x} : \mathsf{S}_1 \vdash \mathsf{s}_2 : \mathsf{T}_2$
2. $\Gamma \vdash \{\mathsf{k}_a = \mathsf{s}_a \ ^{a \in 1..m}\} : \{\mathsf{l}_i : \mathsf{T}_i \ ^{i \in 1..n}\}$ ならば, $\{\mathsf{l}_i \ ^{i \in 1..n}\} \subseteq \{\mathsf{k}_a \ ^{a \in 1..m}\}$ かつ $\Gamma \vdash \mathsf{s}_a : \mathsf{T}_i \ (k_a = l_i)$

4.3 補題 (substitution)

$\Gamma, \mathsf{x} : \mathsf{S} \vdash \mathsf{t} : \mathsf{T}$ かつ $\Gamma \vdash \mathsf{s} : \mathsf{S}$ ならば,$\Gamma \vdash [\mathsf{x} \mapsto \mathsf{s}]\ \mathsf{t} : \mathsf{T}$

4.4 preservation

$\Gamma \vdash \mathsf{t} : \mathsf{T}$ かつ $\mathsf{t} \longrightarrow \mathsf{t}'$ ならば, $\Gamma \vdash \mathsf{t}' : \mathsf{T}$

4.5 補題 (cannonical form)

1. v が $\mathsf{T}_1 \to \mathsf{T}_2$ に型付けされた閉じた *value* ならば, v は $\lambda \mathsf{x} : \mathsf{S}_1.\mathsf{t}_2$ の形をしている.
2. v が $\{\mathsf{l}_i : \mathsf{T}_i \ ^{i \in 1..n}\}$ に型付けされた閉じた *value* ならば, v が $\{\mathsf{k}_j = \mathsf{v}_j \ ^{a \in 1..m}\}$ (ただし, $\{\mathsf{l}_i \ ^{i \in 1..n}\} \subseteq \{\mathsf{k}_a \ ^{a \in 1..m}\}$) の形をしている.

New syntactic forms

T := ...	*types:*
Top	*maximum type*
Bot	*minimum type*

New subtyping rules $\boxed{\mathsf{S} <: \mathsf{T}}$

S <: Top	(S-TOP)
Bot <: T	(S-BOT)

図 4: Top 型と Bot 型

4.6 progress

t が閉じた well-typed な項ならば，t は *value* または t $\longrightarrow$ t′ を満たす t' である．

5 Top 型と Bot 型

Top 型と Bot 型を導入する．(図**??**)

Top 型はすべての型の supertype となる型である．Top 型を $\lambda_{<:}$ に導入しなくてもシステムの性質はほとんど変わらないが，以下の点で Top 型は有用である．

- オブジェクト指向言語における Object に対応する．
- subtyping と parametric polymorphism を組み合わせた，洗練されたシステムの構築に役立つ．

Bot はすべての型の subtype である．Bot 型は閉じた *value* を持たないので，*value* を返さない操作[5] の型に利用することができる．Bot 型を導入することは以下の点で有用である．

- *value* を返さないことをプログラマに通知するとき
- いかなる *value* の型が期待されているコンテキスト内でもその *value* を安全に利用することが可能であるということを型検査器に通知するとき

[5]例外，継続 (continuation) 操作

以下のような例を考えてみる．

例：$\lambda x : \mathsf{Nat}.\ \lambda y : \mathsf{Nat}.$

$$\mathsf{if}\ y = 0\ \mathsf{then}\ \ \mathsf{error}$$
$$\mathsf{else}\ \ x/y$$

上の例の場合，T-SUB を利用して，項 error を x/y と同じ型 (Nat) に型付けすることができるので，T-IF[6] より上の項は well-typed な項となる．

しかし，Bot 型を導入することで，型検査器を複雑にしてしまうという問題が生じる．導入前であれば，”$t_1\ t_2$ が well-typed な項であるとき，t_1 は関数型でなければならない” となる．しかし，Bot 型を導入することで”t_1 は関数型または Bot 型でなければならない” というように変更される．

[6] $\dfrac{\Gamma \vdash t_1 : \mathsf{Bool} \quad \Gamma \vdash t_2 : T \quad \Gamma \vdash t_3 : T}{\Gamma \vdash \mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3\ : T}$

＜付録＞

$(\lambda \mathsf{r} : \{\mathsf{x} : \mathsf{Nat}\}.\ \mathsf{r.x})\ \{\mathsf{x} = 0, \mathsf{y} = 1\}$ の導出木

$$\dfrac{\vdash \lambda \mathsf{r}{:}\{\mathsf{x}{:}\mathsf{Nat}\}.\ \mathsf{r.x}{:}\{\mathsf{x}{:}\mathsf{Nat}\}{\to}\mathsf{Nat} \qquad \dfrac{\dfrac{\dfrac{}{\vdash 0{:}\mathsf{Nat}} \quad \dfrac{}{\vdash 1{:}\mathsf{Nat}}}{\vdash \{\mathsf{x}{=}0,\mathsf{y}{=}0\}{:}\{\mathsf{x}{:}\mathsf{Nat},\mathsf{y}{:}\mathsf{Nat}\}}(\mathsf{T}{-}\mathsf{RCD}) \quad \dfrac{}{\{\mathsf{x}{:}\mathsf{Nat},\mathsf{y}{:}\mathsf{Nat}\} <: \{\mathsf{x}{:}\mathsf{Nat}\}}(\mathsf{S}{-}\mathsf{RCDWIDTH})}{\vdash \{\mathsf{x}{=}0,\mathsf{y}{=}0\}{:}\{\mathsf{x}{:}\mathsf{Nat}\}}(\mathsf{T}{-}\mathsf{SUB})}{\vdash (\lambda \mathsf{r}{:}\{\mathsf{x}{:}\mathsf{Nat}\}.\ \mathsf{r.x})\ \{\mathsf{x}{=}0,\mathsf{y}{=}0\}\ :\ \mathsf{Nat}}(\mathsf{T}-\mathsf{APP})$$

無限の subtyping の関係

無限の subtyping の関係を作成することができる．

(降順)

レコード型を利用する．

例:$S_0 = \{\}, S_1 = \{\mathsf{a} : \mathsf{Nat}\}, S_2 = \{\mathsf{a} : \mathsf{Nat}, \mathsf{b} : \mathsf{Nat}\}, S_3 = \{\mathsf{a} : \mathsf{Nat}, \mathsf{b} : \mathsf{Nat}, \mathsf{c} : \mathsf{Nat}\}$, ... とする．

$S_0 :> S_1 :> S_2 :> S_3 :>$ ....

(昇順)

関数型を利用する．このことから全ての関数型の supertype となる関数型はないことがわかる．

例:$T_0 = S_0 \to \mathsf{Nat}$, $T_1 = S_1 \to \mathsf{Nat}$, $T_2 = S_2 \to \mathsf{Nat}$, $T_3 = S_3 \to \mathsf{Nat}$, ... とする．

$T_0 <: T_1 <: T_2 <: T_3 <:$ ....

T-SUB による変換

T-SUB を利用して，S-TRANS を利用せずに導出を行うことが可能である．以下に例を示す．

- T-TRANS を利用した場合

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \mathsf{t} : \mathsf{S}} \qquad \dfrac{\dfrac{\vdots}{\mathsf{S} <: \mathsf{U}} \quad \dfrac{\vdots}{\mathsf{U} <: \mathsf{T}}\ (S-TRANS)}{\mathsf{S} <: \mathsf{T}}}{\Gamma \vdash \mathsf{t} : \mathsf{T}}\ (T-SUB)$$

- T-TRANS を利用しない場合

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash \mathsf{t} : \mathsf{S}} \quad \dfrac{\vdots}{\mathsf{S} <: \mathsf{U}}}{\Gamma \vdash \mathsf{t} : \mathsf{U}}\ (T-SUB) \qquad \dfrac{\vdots}{\mathsf{U} <: \mathsf{T}}}{\Gamma \vdash \mathsf{t} : \mathsf{T}}\ (T-SUB)$$

証明

補題 inversion of the subtypes relation

(1)(関数型) に関してのみ証明を行う．

subtyping の導出に関する帰納法を用いる．

$S <: T_1 \to T_2$ の導出に関して最後に用いる規則としてはS-REFL, S-TRANS, S-ARROW が考えられる．以下，各々に関して証明を行う．

- **S – REFL の場合**
  反射性より，$T_1 <: T_1 (= S_1)$，$T_2 (= S_2) <: T_2$ ． よって成り立つ．

- **S – TRANS の場合**
  前提より，$S <: U$ かつ $U <: T_1 \to T_2$. 帰納法の仮定より $U \equiv U_1 \to U_2$ ($T_1 <: U_1$ かつ $U_2 <: T_2$) である．U が関数型であるとわかるので，さらに帰納法の仮定より $S \equiv S_1 \to S_2$ ($U_1 <: S_1$ かつ $S_2 <: U_2$) である．S-TRANS を 2 回利用し，$S \equiv S_1 \to S_2$ ($T_1 <: S_1$ かつ $S_2 <: T_2$) .

- **S – ARROW の場合**
  S が期待する形 ($S_1 \to S_2$) であり，前提の部分が満たすべき条件 ($T_1 <: S_1$ かつ $S_2 <: T_2$) であるため，これが証明すべきことに対応する．

補題 ＊

(1)(関数型) に関してのみ証明を行う．typing の導出に関する帰納法を用いる．

- **T – ABS の場合**
  $S_1 = T_1$ とすれば，$T_1 <: T_1$かつ$\Gamma, x : T_1 \vdash s_2 : T_2$. よって成立.

- **T – SUB の場合**
  T-SUB より，$\lambda S_1.s_2 : S$. 補題 inversion より，$S = S_1' \to S_2'$ ($T_1 <: S_1'$ かつ $S_2' <: T_2$) 帰納法の仮定より，$S_1' <: S_1$ かつ $\Gamma, x : S_1 \vdash s_2 : S_2'$.T-TRANS より，$S_1 <: T_1$.T-SUB より，$\Gamma, x : S_1 \vdash s_2 : T_2$.

補題 substitution

substitution の証明は 9 章 (9.3.8) で行ったが，ここでは，この章で導入された規則に関して証明を行う．typing の導出に関する帰納法を用いる．

- **T – SUB の場合**
  前提より，$\Gamma, x : S \vdash t : S$ かつ $\Gamma \vdash s : S$. 帰納法の仮定より，$\Gamma \vdash [x \mapsto s]\, t : S$. $S <: T$ と組み合わせ,T-SUB を用いて，$\Gamma \vdash [x \mapsto s]\, t : T$.

- **T – RCD の場合**
  前提より，$\Gamma, x : S \vdash t_i : T_i$ ($i \in 1..n$) . 帰納法の仮定より，$\Gamma \vdash [x \mapsto s]\, t_i : T_i$ ($i \in 1..n$) . 代入後の各 i に対して型は変わらないので，レコード全体の型は保存される．

- **T – PROJ** の場合
  前提より, $\Gamma, x : S \vdash t_i : \{ l_i : T_i{}^{i \in 1..n} \}$. 帰納法の仮定より, $\Gamma \vdash [x \mapsto s]\, t_i : \{ l_i : T_i{}^{i \in 1..n} \}$. 代入後の各 i に対して型は変わらないので, レコード全体の型は保存される.

preservation

preservation に関しても substitution と同様, 9 章 (9.3.9) で証明を行ったが, ここでは 9 章の証明と異なる規則, 本章で導入された規則に関して証明を行う. typing の導出に関する帰納法を用いる.

- **T – APP の場合**
  9 章の証明と異なる部分 ($t \longrightarrow t'$ の評価に用いる E-APPABS の場合) について証明を行う.
  T-APP より, $t = t_1 t_2$, $\Gamma \vdash t_1 : T_{11} \to T_{12}$, $\Gamma \vdash t_2 : T_{11}$. E-APPABS より, $t_1 = \lambda x : S_{11}.\, t_{12}$, $t_2 = v_2$, $t' = [x \mapsto v_2]\, t_{12}$. 補題 $*$の (1) より, $S_{11} <: T_{11}$, $\Gamma, x : S_{11} \vdash t_{12} : T_{12}$. T-SUB より, $\Gamma \vdash t_2 : S_{11}$. 補題 substitution より, $\Gamma \vdash t' : T_{12}$.

- **T – SUB の場合**
  前提より, $\Gamma \vdash t : S$ かつ $\Gamma \vdash S <: T$. 帰納法の仮定より, $\Gamma \vdash t' : S$. $S <: T$ と組み合わせ, $\Gamma \vdash t' : T$.

- **T – RCD の場合**
  評価規則としては E-RCD が考えられる. この規則と T-RCD の前提より, $t_j \longrightarrow t'_j$ $(j \in 1..n)$, $\Gamma \vdash t_j : T_j$. 帰納法の仮定により, $t'_j : T_j$. E-RCD では t_j 以外のフィールドの型は変わらないため, レコード全体の型は保存される.

- **T – PROJ の場合**
  T-PROJ より, $t = t_1.l_j$, $\Gamma \vdash t_1 : \{ l_i : T_i{}^{i \in 1..n} \}$, $T = T_j$. 評価規則としては E-PROJRCD, E-PROJ が考えられる.

  – E-PROJRCD の場合
    $t_1 = \{ k_a = v_a{}^{a \in 1..n} \}$, $l_j = k_b$, $t' = v_b$. 補題 $*$ (2) より, $\{ l_i{}^{i \in 1..n} \} \subseteq \{ v_a{}^{a \in 1..n} \}$ かつ $\Gamma \vdash v_a : T_i$ $(k_a = l_i)$. $l_j = k_b$ より $t' : T_j$.
  – E-PROJ の場合
    帰納法の仮定より, レコード全体の型は保存され, E-PROJ より, 評価後のフィールドの型も保存されている.

補題 cannonical forms

(1)(関数型) に関してのみ証明を行う.

typing の導出に関する帰納法を用いる. $\vdash\ v : T_1 \to T_2$ の導出に関して最後に用いる規則としては T-ABS, T-SUB が考えられる.

- **T – ABS の場合**
  T-ABS が証明すべきことそのものである.
- **T – SUB の場合**
  T-SUB と補題 inversion より, S は $S'_1 \to S'_2$ の形になる . 帰納法の仮定より, 証明すべきことがいえる.

progress

typing の導出に関する帰納法を用いる. 変数に関しては t が閉じた項であるためこのような場合は起らない. λ 抽象に関しては *value* であるため明らかに成り立つ. 以下ではその他の場合に関して証明を行う.

- **T – APP の場合**
  $t = t_1 t_2$ とし, 帰納法の仮定より t_1, t_2が *value* かどうかで場合わけを行う . t_1 が *value* でなければ, E-APP1 を利用して t に対して 評価を行う. t_1 が *value* であり, かつ t_2 が *value* でなければ, E-APP2 を利用して t に対して 評価を行う. t_1, t_2 がともに *value* であれば, 補題 cannonical forms (1) より, t_1 が $\lambda x : S_1.t_2$ の形を持つ. したがって E-APPABS を利用して t に対して 評価を行う.
- **T – SUB の場合**
  帰納法の仮定より, t は *value*, または 1 ステップ簡約可能であるので, これは証明すべきことである.
- **T – RCD の場合**
  帰納法の仮定より, 各 t_i $(i \in 1..n)$ は *value*, または 1 ステップ簡約可能である. すべての t_i が *value* であれば, t も *value* となる. 少なくとも t_i のどれか 1 つが *value* でなければ, E-RCD を利用して 1 ステップ簡約可能である.
- **T – PROJ の場合**
  帰納法の仮定より, t_1 は *value*, または 1 ステップ簡約可能である. t_1 が *value* であれば, 補題 cannonical form (2) より, t_1 は $\{ k_a = v_a{}^{a \in 1..m}\}$ $(\{ l_i{}^{i \in 1..n}\} \subseteq \{ k_a{}^{a \in 1..m}\})$ の形である. $l_j \subseteq \{k_a{}^{a \in 1..m}\}$ であるので, E-PROJRCD を利用して 1 ステップ簡約が可能である. t_1 が 1 ステップ簡約可能であれば, E-PROJ を利用して, 1 ステップ簡約可能である.

Bot 型は閉じた *value* を持たない

Bot 型は閉じた *value* を持つ ($\Gamma \vdash \mathsf{v} : \mathsf{Bot}$) と仮定する．このとき，T-SUB と S-Bot から関数型 $\Gamma \vdash \mathsf{v} : \mathsf{Top} \to \mathsf{Top}$ を導出することが可能となる．他方，同様にレコード型 $\Gamma \vdash \mathsf{v} : \{\}$ を導出することも可能となる．しかし，v が構文上，関数型とレコード型の両方になり得ないので，仮定が間違っていたことになる．したがって，Bot 型は閉じた *value* を持たない．