Types and Programming Languages
Chapter 8,9

Rei Tomori

July 14, 2025

概要

今回は以下の内容を扱う:

1. §11. 単純な拡張
2. §12. 正規化
3. §13. 参照

§11. 単純な拡張

基本型

1. プログラミング言語には基本型[1].i.e. 構造を持たない単純な値の集合とその上のプリミティブな演算がある.
   - 今後は $\mathtt{Nat}, \mathtt{Bool}$ に加え，$\mathtt{String}, \mathtt{Float}$ 型を用いる.
2. (その上の演算を捨象した) 一般的な基本型を扱いたいことがある．そのためには，言語が”非解釈の”基本型の集合 $\mathscr{A}$[2]を備えているとする.
   - これを表わすためには，型の構文規則を変更してメタ変数 A($\mathscr{A}$ の要素を表わす) を加える.
   - 以降，基本型の名前として $\mathsf{A}, \mathsf{B}, \mathsf{C}$ を用いる.
3. 非解釈な型を導入することで，基本型の要素上を特定することなく，その上で走る変数を束縛できる.
   - $\lambda \mathsf{x} : \mathsf{A}.\mathsf{x}$ は $\mathsf{x} : \mathsf{A}$ が何であれ，x を x 自身に写す恒等関数である.

[1]基底型

[2]原子型，つまり型システムにおいては内部システムを持たない型の略

§11.2. Unit 型

要素を 1 つしか持たない型である Unit 型を導入する．この型は次のように解釈される:

- 唯一の要素は項定数 unit(u，しばしば () で表わされる) で，任意の Unit 型の項は一意的に unit に評価される．

Unit 型は主に副作用を持つ言語で応用される[3]．たとえば，可変参照を変更する関数では，返り値ではなく副作用に興味があるため，Unit 型が返り値の型とされる．類似物として C 系言語の void 型がある．

[3]純粋函数型言語では，たとえば各 $n \in \mathbb{N}$ に対して

Unit 型

Unit 型は[4]，以下の構文，型付け規則および派生形式 (i.e. 糖衣構文)，すなわち STLC への埋め込み方で定義される．

Def.Unit 型の定義

新しい構文形式

(項) $\mathtt{t} ::= \ldots$

$\quad \mathtt{Unit}$

(項) $\mathtt{v} ::= \ldots$

$\quad \mathtt{unit}$

(型) $\mathtt{T} ::= \ldots$

$\quad \mathtt{Unit}$

新しい型付け規則

$$\Gamma \vdash \mathtt{unit} : \mathtt{Unit}$$

新しい派生形式

$$\mathtt{t}_1; \mathtt{t}_2 \stackrel{\text{def}}{=} (\lambda \mathtt{x} : \mathtt{Unit}.\mathtt{t}_2)\mathtt{t}_1$$

ただし$x \notin FV(\mathtt{t}_2)$

[4]Haskell でいうところの 0-tuple

§11.3. 派生形式: 逐次実行とワイルドカード

副作用のある言語における文の逐次実行を形式化する．逐次実行形式は項 $\mathtt{t}_1, \mathtt{t}_2$ に対し，$\mathtt{t}_1$ を正規形まで評価し，結果を捨てた後に $\mathtt{t}_2$ を評価する．

$(;)$ の意味を直接表わす方法と，$(;)$ を内部言語のある項の略記とする 2 通りの形式化が考えられる．

1. $\mathtt{t}_1; \mathtt{t}_2$ を新たな構文要素とする方法．評価規則

$$\frac{\mathtt{t}_1 \to \mathtt{t}_1'}{\mathtt{t}_1; \mathtt{t}_2 \to \mathtt{t}_1'; \mathtt{t}_2}\ (\text{E-SEQ})$$

$$\mathtt{v}_1; \mathtt{t}_2 \to \mathtt{t}_2\ (\text{E-SEQNEXT})$$

   および型付け規則

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Unit} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \mathtt{t}_1; \mathtt{t}_2 : \mathtt{T}_2}\ (\text{T-SEQ})$$

   を付け加えることで $(;)$ の振る舞いを特徴付ける．

2. 内部言語の項の略記とする方法．$\mathtt{t}_1; \mathtt{t}_2 \stackrel{\mathtt{def}}{=} (\lambda \mathtt{x} : \mathtt{Unit}.\mathtt{t}_2)\mathtt{t}_1$，ただし $x\mathtt{x} \notin FV(\mathtt{t}_2)$ とする．

派生形式

はじめの形式化で定めた評価および型付け規則は，Unit のみを型として持つ STLC の評価関係および型付け規則より従う．このことを確認しよう．

Thm.11.3.1[逐次実行は派生形式である]
Unit 型，逐次実行およびそれらの評価・型付け規則を持つ STLC を λ^E と書く．[5]また，Unit 型のみを持つ STLC を λ^I と書き，$e \in \lambda^I \to \lambda^E$ を，λ^I の各項を対応する λ^E の項に写す詳細化関数[6]とする．つまり，e は $t_1; t_2$ の各出現を，$(\lambda x : \mathtt{Unit}.t_2)t_1 (x \notin FV(t_2))$ に置き換える．すると，λ^E の各項 t に対して，

1. $t \to_E t' \Rightarrow e(t) \to_I e(t')$．逆に $e(t) \to_I u \Rightarrow \exists t'.t'$ は λ^E の項 $\wedge u = e(t') \wedge t \to_E t'$.
2. $\Gamma \vdash^E t : T \Leftrightarrow \Gamma \vdash^I e(t) : T$

[5]外部言語の略.
[6]elaboration function

Thm.11.3.1. の証明 (一部)

Proof.
$\mathtt{t}$ の構造帰納法で示す．(2) の主張は白板で示すことにする．

1. (1) の証明．
   1. ($\Rightarrow$) 新たな構文要素，つまり $\mathtt{t} = \mathtt{t}_1; \mathtt{t}_2$ または $\mathtt{t} = \mathtt{v}_1; \mathtt{t}_2$ の場合を考え，$\mathtt{t} \to_{\mathsf{E}} \mathtt{t}'$ なる $\mathtt{t}'$ の存在を仮定．
      - $\mathtt{t} = \mathtt{v}_1; \mathtt{t}_2$．仮定より $\mathtt{t}' = \mathtt{t}_2$．いま $\mathtt{x} \notin \mathit{FV}(\mathtt{t}_2)$ を任意に取ると，定義より $e(\mathtt{t}) = (\lambda \mathtt{x} : \mathtt{Unit}.e(\mathtt{t}_2))\ \mathtt{v}_1$．(E-APPABS) から $\mathtt{t} \to_I [\mathtt{x} \mapsto \mathtt{v}_1]\, e(\mathtt{t}_2) = \mathtt{t}'$ がいえ，$\mathtt{x} \notin \mathit{FV}(e(\mathtt{t}_2))$ からこれは $\mathtt{v}_1$ に等しい．
      - $\mathtt{t} = \mathtt{t}_1; \mathtt{t}_2$．仮定より $\mathtt{t}' = \mathtt{t}_1'; \mathtt{t}_2 \wedge \mathtt{t}_1 \to_{\mathsf{E}} \mathtt{t}_1'$ なる $\mathtt{t}', \mathtt{t}_1'$ が存在．IH より $e(\mathtt{t}_1) \to_I e(\mathtt{t}_1')$．(E-APP1) から $e(\mathtt{t}) = e(\mathtt{t}_1)e(\mathtt{t}_2) \to_I e(\mathtt{t}_1')e(\mathtt{t}_2 = e(\mathtt{t}')$.
   2. ($\Leftarrow$)．$e(\mathtt{t}) \to_I \mathtt{u}$ を仮定する．(1) と同様に場合を分ける．
      - $\mathtt{t} = \mathtt{v}_1; \mathtt{t}_2$ のとき．$e(\mathtt{t}) = (\lambda \mathtt{x} : \mathtt{Unit}.e(\mathtt{t}_2))\mathtt{v}_1$($\mathtt{x}$ は fresh な変数)，$\mathtt{u} = e(\mathtt{t}_2)$. このとき．$\mathtt{t}' = \mathtt{t}_2$ とすれば従う．
      - $\mathtt{t} = \mathtt{t}_1; \mathtt{t}_2$ のとき．$\mathtt{t} = (\lambda \mathtt{x} : \mathtt{Unit}.e(\mathtt{t}_2))\ e(\mathtt{t}_1)$ であり，仮定より $e(\mathtt{t}_1) \to \mathtt{u}_1'$ なる $\mathtt{u}_1'$ を取ると $\mathtt{u} = (\lambda \mathtt{x} : \mathtt{Unit}.e(\mathtt{t}_1))\ \mathtt{u}_1'$．帰納法の仮定より，$\mathtt{u}_1' = e(\mathtt{t}_1') \wedge \mathtt{t}_1 \to_E \mathtt{t}_1'$ なる λ^E の項 $\mathtt{t}_1'$ が取れる．ゆえに $\mathtt{u} = (\lambda \mathtt{x} : \mathtt{Unit}.e(\mathtt{t}_2))e(\mathtt{t}_1') = e(\mathtt{t}_1'; \mathtt{t}_2)$ となり従う．

□

派生形式

1. 派生形式として導入する利点には，表層構文を拡張しつつ型安全性を保証しなければならない内部言語を単純に保てることにある．
2. 他の派生形式として，抽象の本体で使わない引数を束縛するようなワイルドカード[7]の慣習がある．ワイルドカード束縛子は (_) で表わされる．
   - $\lambda \_ : \mathsf{S.t} \stackrel{\mathsf{def}}{=} \lambda x : \mathsf{S.t}$，ただし x は t に現れない．

[7] hole などということもある

§11.4. 型指定

1. 所与の項に特定の型を明示的に指定する機能を型指定 といい，型 T を指定した項 t は t as T と書かれる．用途としては次のようなものがある:
   - ドキュメンテーション．特に型シグネチャが複雑になる場合などは有用．
   - 複雑な型の表示の制御．複雑な型式の略記[8]を導入し，型検査器に必要に応じて略記を折り畳み/展開させる．
   - 抽象化．項 t が複数の異なる型を持ちうる体系 (たとえば部分型付けのある) において，t がより少数の型を持つかのよう型検査器に指示し，型の一部を隠蔽させる．

[8]型シノニムなどと呼ばれる

型指定

1. 型指定の型付けおよび評価規則は以下の通り:

Def. 型指定

$$t ::= \ldots$$
$$\quad t \text{ as } T$$

評価規則

$$v_1 \text{ as } T \to v_1 \text{ (E-ASCRIBE)}$$

$$\frac{t_1 \to t_1'}{t_1 \text{ as } T \to t_1' \text{ as } T} \text{ (E-ASCRIBE1)}$$

型付け規則

$$\frac{\Gamma \vdash t_1 : T}{\Gamma \vdash t_1 \text{ as } T} \text{ (T-ASCRIBE)}$$

型指定

演習 11.4,1.(推奨)

1. 型指定を派生形式として形式化せよ．また，p.10 の型付け規則および評価規則が派生形式と対応することを示せ．
2. E-ASCRIBE, E-ASCRIBE1 の代りに，以下の先行的な規則が与えられたとする:

$$\mathsf{t}_1 \text{ as } \mathsf{T} \to \mathsf{t}_1 \quad \text{(E-ASCRIBEEAGER)}$$

この場合に型指定を派生形式として扱うことは出来るか．

Proof.

1. 型指定を持つ STLC を外部言語 λ^E，非解釈な型を持つ STLC を内部言語 λ^I とし，詳細化関数 e を各 t as T に対し $e(\mathsf{t} \text{ as } \mathsf{T}) = (\lambda \mathsf{x} : \mathsf{T}.\mathsf{x})\ \mathsf{t}$ と定める．評価規則と型付け規則を示そう．

□