Yil 言語 リファレンス

易 翠衡 (@yicuiheng)
王里国立大学大学院 魔術科学専攻 魔術言語理論研究室 修士課程

2021/9/18

1　Yil とはどのような言語か

TODO: 篩型とかプログラム合成の話をする

2　Yil のコア言語

2.1　Yil の構文の形式的定義

Yil のコア言語は 図 1 で定義される.

$$
\begin{aligned}
(\text{プログラム})\ P &::= \mathrm{func}(f,\overline{x},e) \mid P,\ \mathrm{func}(f,\overline{x},e)\\
(\text{式})\ e &::= x \mid n \mid \mathrm{op} \mid v_1\ \ v_2 \mid \mathrm{if}\ v\ \mathrm{then}\ e_1\ \mathrm{else}\ e_2\ \mid \mathrm{let}\ x\ =\ e_1\ \mathrm{in}\ e_2\\
(\text{値})\ v &::= x \mid n \mid f\ \overline{v} \qquad (\text{ただし}\ |\overline{v}| < \mathrm{arity}_P(f))
\end{aligned}
$$

図 1　Yil コア言語の構文

プログラム P は, 関数名が f, 引数が $\overline{x}$, 本体が e の再帰関数 $\mathrm{func}(f,\overline{x},e)$ の列である. x, y は式 e の上のメタ変数であり, f は関数変数の上のメタ変数である. op は組み込み関数の変数の集合である.これには整数上の和 $+$, 差 $-$, 積 $\times$, 比較 $<$ が含まれる. 組み込み関数の関数適用は慣習に合わせた表記をする場合がある.つまり $x = y$ は $=\ x\ y$ と同等である. $v_1\ v_2$ は関数適用であり, $\mathrm{if}\ v\ \mathrm{then}\ e_1\ \mathrm{else}\ e_2$ は条件分岐を表す. $\mathrm{if}\ v\ \mathrm{then}\ e_1\ \mathrm{else}\ e_2$ は v が 0 の時に e_1 を評価し, 1 の時に e_2 を評価する.読みやすさのために 0 を true, 1 を false と書くことがある.組み込み関数変数 not $\in$ op は true を false に, false を true に写す関数である. $\mathrm{arity}_P(\cdot)$ はプログラム P における関数の引数の数を表す. つまり $\mathrm{func}(f,\overline{x},e) \in P$ の時, $\mathrm{arity}_P(f) = |\overline{x}|$ であり $\mathrm{arity}_P(+) = 2$ である.

Yil 言語の実装と形式的定義とは異なる点がいくつかある.例えば実装では if 式の条件部分に式を書けるが, 形式的定義では型システムの都合上値に限定している.つまり実装では $\mathrm{if}\ x = 0\ \mathrm{then}\ \mathrm{true}\ \mathrm{else}\ \mathrm{false}$ と書けるが, 形式定義では $\mathrm{let}\ \mathrm{cond} = x = 0\ \mathrm{in}\ \mathrm{if}\ \mathrm{cond}\ \mathrm{then}\ \mathrm{true}\ \mathrm{else}\ \mathrm{false}$ などと書く必要がある.また, 実装では式が出現しうるあらゆる場所に関数を書けるが, 形式的定義ではトップレベルに限定している.この違いは lambda lifting [1] というプログラム上のメタ操作によって無視できるので本質的な差にはならない.

以降に使う補助関数を図 2 に定義する. $\mathrm{fv}(P)$, $\mathrm{fv}(e)$ はそれぞれプログラム P もしくは式 e の自由変数である.さらに $[e/x]\,P$, $[e_1/x]\,e_2$ はそれぞれプログラム P もしくは式 e_2 中の自由変数 x に式 e_1 を代入して得られたプログラムもしくは式を表す.

$$
\begin{aligned}
[e_1/x]\,(\mathrm{func}(f,\overline{x},e_2)) &\stackrel{\mathrm{def}}{=} \mathrm{func}(f,\overline{x},e) && (\text{ただし } x = f \text{ または } x \in \overline{x})\\
[e_1/x]\,(\mathrm{func}(f,\overline{x},e_2)) &\stackrel{\mathrm{def}}{=} \mathrm{func}(f,\overline{x},[e_1/x]\,(e_2)) && (\text{ただし } x \not\equiv f \text{ かつ } x \notin \overline{x})\\
[e_1/x]\,(P,\mathrm{func}(f,\overline{x},e_2)) &\stackrel{\mathrm{def}}{=} [e_1/x]\,(P),[e_1/x]\,(\mathrm{func}(f,\overline{x},e_2)) &&\\
[e_1/x_1]\,x_2 &\stackrel{\mathrm{def}}{=} x_2 && (\text{ただし } x_1 \not\equiv x_2)\\
[e/x]\,x &\stackrel{\mathrm{def}}{=} e &&\\
[e/x]\,n &\stackrel{\mathrm{def}}{=} n &&\\
[e/x]\,\mathrm{op} &\stackrel{\mathrm{def}}{=} \mathrm{op} &&\\
[e/x]\,(v_1\ \ v_2) &\stackrel{\mathrm{def}}{=} ([e/x]\,v_1)\,([e/x]\,v_2) &&\\
[e_1/x]\,(\mathrm{if}\ v\ \mathrm{then}\ e_2\ \mathrm{else}\ e_3\ ) &\stackrel{\mathrm{def}}{=} \mathrm{if}\ [e_1/x]\,v\ \mathrm{then}\ [e_1/x]\,e_2\ \mathrm{else}\ [e_1/x]\,e_3 &&\\
[e_1/x]\,(\mathrm{let}\ x = e_2\ \mathrm{in}\ e_3) &\stackrel{\mathrm{def}}{=} \mathrm{let}\ x = [e_1/x]\,e_2\ \mathrm{in}\ e_3 &&\\
[e_1/x_1]\,(\mathrm{let}\ x_2 = e_2\ \mathrm{in}\ e_3) &\stackrel{\mathrm{def}}{=} \mathrm{let}\ x_2 = [e_1/x_1]\,e_2\ \mathrm{in}\ [e_1/x_1]\,e_3 && (\text{ただし } x_1 \not\equiv x_2)
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{fv}(\mathrm{func}(f,\overline{x},e)) &\stackrel{\mathrm{def}}{=} \mathrm{fv}(e) \setminus (\{f\} \cup \{\overline{x}\})\\
\mathrm{fv}(P,\mathrm{func}(f,\overline{x},e)) &\stackrel{\mathrm{def}}{=} \mathrm{fv}(P) \cup \mathrm{fv}(\mathrm{func}(f,\overline{x},e))\\
\mathrm{fv}(x) &\stackrel{\mathrm{def}}{=} \{x\}\\
\mathrm{fv}(n) &\stackrel{\mathrm{def}}{=} \varnothing\\
\mathrm{fv}(\mathrm{op}) &\stackrel{\mathrm{def}}{=} \varnothing\\
\mathrm{fv}(v_1\ \ v_2) &\stackrel{\mathrm{def}}{=} \mathrm{fv}(v_1) \cup \mathrm{fv}(v_2)\\
\mathrm{fv}(\mathrm{if}\ v\ \mathrm{then}\ e_1\ \mathrm{else}\ e_2) &\stackrel{\mathrm{def}}{=} \mathrm{fv}(v) \cup \mathrm{fv}(e_1) \cup \mathrm{fv}(e_2)\\
\mathrm{fv}(\mathrm{let}\ x = e_1\ \mathrm{in}\ e_2) &\stackrel{\mathrm{def}}{=} \mathrm{fv}(e_1) \cup (\mathrm{fv}(e_2) \setminus \{x\})
\end{aligned}
$$

図 2　補助関数

NOTE: 将来的にタプルや代数的データ構造およびパターンマッチも導入予定だが, 今のところは単純のため非関数の式は整数しかないような言語を考える.そのような高度なデータ構造は [2] や [3] にあるような拡張によって無理なく扱えるようになると考えている.

2.2　Yil の操作的意味

Yil の操作的意味を図 3 で定義する.

$$\frac{\mathrm{arity}_P(\mathrm{op}) = |\overline{v}|}{\mathrm{op}\ \overline{v} \ \rightarrow_P\ [\![\mathrm{op}]\!](\overline{v})}\ (\text{E-AppOp}) \qquad \frac{\mathrm{func}(f, \overline{x}, e) \in P \quad |\overline{x}| = |\overline{v}|}{f\ \overline{v}\ \rightarrow_P\ [\overline{v}/\overline{x}]\,e}\ (\text{E-FuncApp})$$

$$\text{if true then } e_1 \text{ else } e_2 \ \rightarrow_P\ e_1 \quad (\text{E-IfTrue}) \qquad \text{if false then } e_1 \text{ else } e_2 \ \rightarrow_P\ e_2 \quad (\text{E-IfFalse})$$

$$\frac{e_1 \ \rightarrow_P\ e_1'}{\text{let } x \ = \ e_1 \text{ in } e_2 \ \rightarrow_P\ \text{let } x \ = \ e_1' \text{ in } e_2}\ (\text{E-Let1}) \qquad \text{let } x \ = \ v \text{ in } e \ \rightarrow_P\ [v/x]\,e \quad (\text{E-Let2})$$

図 3 Yil の操作的意味

$e_1 \ \rightarrow_P\ e_2$ は プログラム P における式 e の 1 ステップの評価を表す.有限ステップで e_1 が e_2 に簡約されることを $e_1 \ \rightarrow_P^*\ e_2$ と書く. Yil の評価の例を 付録 A に示す.

2.3 Yil の単純型

Yil は篩型の型システムを持つ言語であるが, その前に単純型を図 4 で定義する.

int は整数を表す単純型である. $T_1 \rightarrow T_2$ は単純型 T_1 が付く値を受け取って単純型 T_2 が付く式を返す関数である.

$$\begin{aligned} (\text{単純型})\ T &::= \mathrm{int} \mid T \rightarrow T \\ (\text{単純型環境})\ \Delta &::= \emptyset \mid \Delta, x :\ T \end{aligned}$$

図 4 単純型の構文

単純型判断式 $\Delta \vdash_P e :\ T$ が 図 2.3 で定義される単純型付け規則によって導出可能なとき, 単純型環境 Δ のもとで式 e には単純型 T がつくという.

$T_1 \rightarrow (T_2 \rightarrow (T_3 \rightarrow \ldots (T_n \rightarrow T)))$ を $\overline{T} \rightarrow T$ と略記する. $\mathrm{func}(f, \overline{x}, e) \in P$ について $|\overline{x}| = |\overline{T}|$ かつ $\overline{x} :\ \overline{T} \vdash_P e :\ T$ を満たす単純型の列 $\overline{T}$ と単純型 T が存在するとき, 関数 $\mathrm{func}(f, \overline{x}, e)$ に単純型 $\overline{T} \rightarrow T$ がつくといい, $\mathrm{sty}(f) = \overline{T} \rightarrow T$ と書く.また, 組み込み関数の変数 $f \in \mathrm{op}$ の単純型も $\mathrm{sty}(f)$ で得られるとする.

以降は単純型がつく式および関数のみを考えることとする.

$$\frac{\Delta(x) = T}{\Delta \vdash_P x :\ T}\ (\text{ST-Var}) \qquad \frac{\mathrm{func}(f, \overline{x}, e) \in P}{\Delta \vdash_P f :\ \mathrm{sty}(f)}\ (\text{ST-Var}) \qquad \Delta \vdash_P n :\ \mathrm{int} \quad (\text{ST-Num}) \qquad \Delta \vdash_P \mathrm{op} :\ \mathrm{sty}(op) \quad (\text{ST-Op})$$

$$\frac{\Delta \vdash_P v_1 :\ T_1 \rightarrow T_2 \quad \Delta \vdash_P v_2 :\ T_1}{\Delta \vdash_P v_1\ v_2 :\ T_2}\ (\text{ST-App}) \qquad \frac{\Delta \vdash_P v :\ \mathrm{int} \quad \Delta \vdash_P e_1 :\ T \quad \Delta \vdash_P e_2 :\ T}{\Delta \vdash_P \text{if } v \text{ then } e_1 \text{ else } e_2 \ :\ T}\ (\text{ST-If})$$

$$\frac{\Delta \vdash_P e_1 :\ T_1 \quad \Delta, x :\ T_1 \vdash_P e_2 :\ T_2}{\Delta \vdash_P \text{let } x \ = \ e_1 \text{ in } e_2 \ :\ T_2}\ (\text{ST-Let})$$

図 5 単純型付け規則

Yil の単純型付けの例を 付録 B に示す.

3 Yil の篩型

3.1 型の構文

Yil の型システムを図 3.1 に示す.

$$
\begin{aligned}
(\text{篩型})\ \tau &::= \{x|\phi\} \mid (x:\ \tau_1) \to \tau_2 \\
(\text{型環境})\ \Gamma &::= \emptyset \mid \Gamma, x:\ \tau \\
(\text{論理式})\ \phi &::= t_1 \le t_2 \mid \top \mid \bot \mid \neg\phi \mid \phi_1 \land \phi_2 \mid \phi_1 \lor \phi_2 \\
(\text{項})\ t &::= n \mid x \mid t_1 + t_2 \mid n \cdot t
\end{aligned}
$$

図 6　型の構文

$\{x|\phi\}$ は 論理式 ϕ を満足させるような整数 x に付くような型である.例えば $\{x|x \ge 10\}$ は 10 以上の整数につく型の整数につく型である. $\{x|\top\}$, つまり任意の整数に付く型を単に int と書くことにする.また, $(x:\ \tau_1) \to \tau_2$ は 型 τ_1 がつく値 x を受け取って型 τ_2 がつく式を返す関数につく型である. $(x_1:\ \tau_1) \to ((x_2:\ \tau_2) \to \dots ((x_n:\ \tau_n) \to \tau))$ を $(\overline{x}:\ \overline{\tau}) \to \tau$ と略記する.

型環境 Γ は変数束縛 $x:\ \tau$ の列である.これを変数かた型への関数とみなして, 変数 x に束縛されている型を $\Gamma(x)$ と書くことがある. ν をこれまでに使われていない fresh な変数名として型環境 $\Gamma, \nu:\ \phi$ を Γ, ϕ と略記する. ここで 型 $\nu:\ \phi$ は変数を束縛しておらず 条件 ϕ を導入しているだけであるということに注意してほしい.型環境 $\Gamma, x_1:\ \tau_1, x_2:\ \tau_2, \dots, x_n:\ \tau_n, \Gamma'$ を $\Gamma, \overline{x}:\ \overline{\tau}, \Gamma'$ と略記する.

論理式 ϕ は量化子のない線形整数算術 (QF-LIA) である.この論理式は z3 [4] や CVC4 [5] などの SMT ソルバによって妥当性を判定できる. $\phi_1 \Rightarrow \phi_2$ などは一般的な慣習に沿って派生形式として定義される.つまり $\phi_1 \Rightarrow \phi_2, t_1 < t_2$, $t_1 = t_2$ をそれぞれ $\neg\phi_1 \lor \phi_2, t_1 + 1 \le t_2, t_1 \le t_2 \land t_2 \le t_1$ と定義する.

3.2 型付け規則

$$
\frac{\Gamma(x) = \tau}{\Gamma \vdash_P x:\ \tau}\ \text{Ty-Var}
\qquad
\frac{\mathrm{func}(f, \overline{x}, e) \in P \quad \Gamma, \overline{x}:\ \overline{\tau} \vdash_P e:\ \tau}{\Gamma \vdash_P f:\ (\overline{x}:\ \overline{\tau}) \to \tau}\ \text{Ty-Func}
\qquad
\Gamma \vdash_P n:\ \{\nu|\nu = n\}\quad \text{Ty-Num}
$$

$$
\frac{\Gamma \vdash_P v_1:\ (x:\ \tau_1) \to \tau_2 \quad \Gamma \vdash_P v_2:\ \tau_1}{\Gamma \vdash_P v_1\ v_2:\ [v_2/x]\tau_2}\ \text{Ty-App}
\qquad
\frac{\Gamma \vdash_P v:\ \{\nu|\phi\} \quad \Gamma, \phi \land \nu = 0 \vdash_P e_1:\ \tau \quad \Gamma, \phi \land \nu = 1 \vdash_P e_2:\ \tau}{\Gamma \vdash_P \text{if } v \text{ then } e_1 \text{ else } e_2\ :\ \tau}\ \text{Ty-If}
$$

$$
\frac{\Gamma \vdash_P e_1:\ \tau_1 \quad \Gamma, x:\ \tau_1 \vdash_P e_2:\ \tau_2 \quad x \notin \mathrm{fv}(\tau_2)}{\Gamma \vdash_P \text{let } x\ =\ e_1 \text{ in } e_2\ :\ \tau_2}\ \text{Ty-Let}
\qquad
\frac{\Gamma \vdash_P e:\ \tau' \quad \Gamma \vdash_P \tau' <:\ \tau}{\Gamma \vdash_P e:\ \tau}\ \text{Ty-Sub}
$$

$$
\frac{\models [\![\Gamma]\!] \land \phi_1 \Rightarrow \phi_2}{\Gamma \vdash_P \{\nu|\phi_1\} <:\ \{\nu|\phi_2\}}\ \text{SubTy-Int}
\qquad
\frac{\Gamma \vdash_P \tau_{21} <:\ \tau_{11} \quad \Gamma, \nu:\ \tau_{21} \vdash_P \tau_{12} <:\ \tau_{22}}{\Gamma \vdash_P (\nu:\ \tau_{11}) \to \tau_{12} <:\ (\nu:\ \tau_{21}) \to \tau_{22}}\ \text{SubTy-Func}
$$

図 7　Yil の型付け規則

型付けの例を 付録 C に示す.

3.3 型の健全性

定理 3.1: 進捗定理

$\Gamma \vdash_P e_1 : \tau$ のとき, 式 e_1 は値であるか, もしくは $e_1 \rightarrow_P e_2$ なる e_2 が存在する.

証明 TODO: 証明を書く

(証明終)

補題 3.2: 代入補題

$\Gamma_1 \vdash_P v : \tau'$ かつ $\Gamma_1, x : \tau', \Gamma_2 \vdash_P e : \tau$ ならば $\Gamma_1, [v/x]\Gamma_2 \vdash_P [v/x]e : [v/x]\tau$

証明 TODO: 証明を書く

(証明終)

定理 3.3: 保存定理

TODO: 書く

証明 TODO: 証明を書く

(証明終)

付録 A Yil の式の評価例

以下に Yil で与えられた引数が偶数か奇数かを判定するプログラムの評価の例を示す.ただし $\mathrm{func}(\mathtt{is\_even}, x, \ldots) \in P$ にように文脈から明らかなものは紙面の都合上 ... で省略することがある.

$$
P \stackrel{\mathrm{def}}{=} \left\{
\begin{array}{l}
\mathrm{func}(\mathtt{is\_even}, x, \\
\quad \mathrm{let\ cond} \ = \ x = 0 \ \mathrm{in} \\
\quad \mathrm{if\ cond\ then\ true} \\
\quad \mathrm{else} \\
\quad\quad \mathrm{let}\ x' \ = \ x - 1 \ \mathrm{in} \\
\quad\quad \mathrm{let}\ y \ = \ \mathtt{is\_odd}\ x' \ \mathrm{in} \\
\quad\quad \mathrm{not}\ y \ ), \\
\mathrm{func}(\mathtt{is\_odd}, x, \\
\quad \mathrm{let\ cond} \ = \ x = 0 \ \mathrm{in} \\
\quad \mathrm{if\ cond\ then\ false} \\
\quad \mathrm{else} \\
\quad\quad \mathrm{let}\ x' \ = \ x - 1 \ \mathrm{in} \\
\quad\quad \mathrm{let}\ y \ = \ \mathtt{is\_even}\ x' \ \mathrm{in} \\
\quad\quad \mathrm{not}\ y \ )
\end{array}
\right. ,
$$

```
    is_even 2
→P  let cond = 2 = 0 in
    if cond then true
    else
      let x' = 2 − 1 in
      let y = is_odd x' in
      not y
→P  let cond = false in
    if cond then true
    else
      let x' = 2 − 1 in
      let y = is_odd x' in
      not y
→P  if false then true
    else
      let x' = 2 − 1 in
      let y = is_odd x' in
      not y
→P  let x' = 2 − 1 in
    let y = is_odd x' in
    not y
→P  let x' = 1 in
    let y = is_odd x' in
    not y
→P  let y = is_odd 1 in
    not y
→P  let y =
      let cond = 1 = 0 in
      if cond then false
      else
        let x' = 1 − 1 in
        let y = is_even x' in
        not y in
    not y
→P  let y =
      let cond = false in
      if cond then false
      else
        let x' = 1 − 1 in
        let y = is_even x' in
        not y in
```

```
    not y
→_P let y =
        if false then false
        else
            let x' = 1 − 1 in
            let y = is_even x' in
            not y in
    not y
→_P let y =
        let x' = 1 − 1 in
        let y = is_even x' in
        not y in
    not y
→_P let y =
        let x' = 0 in
        let y = is_even x' in
        not y in
    not y
→_P let y =
        let y = is_even 0 in
        not y in
    not y
→_P let y =
        let y =
            let cond = 0 = 0 in
            if cond then true
            else
                let x' = 0 − 1 in
                let y = is_odd x' in
                not y in
        not y in
    not y
→_P let y =
        let y =
            let cond = true in
            if cond then true
            else
                let x' = 0 − 1 in
                let y = is_odd x' in
                not y in
        not y in
    not y
```

$$
\begin{aligned}
\rightarrow_P\ & \text{let } y \ = \\
& \quad \text{let } y = \\
& \quad\quad \text{if true then true} \\
& \quad\quad \text{else} \\
& \quad\quad\quad \text{let } x' \ = \ 0 - 1 \text{ in} \\
& \quad\quad\quad \text{let } y \ = \ \mathtt{is\_odd}\ x' \text{ in} \\
& \quad\quad\quad \text{not } y \text{ in} \\
& \quad \text{not } y \text{ in} \\
& \text{not } y \\
\rightarrow_P\ & \text{let } y \ = \\
& \quad \text{let } y \ = \ \text{true in} \\
& \quad \text{not } y \text{ in} \\
& \text{not } y \\
\rightarrow_P\ & \text{let } y \ = \ \text{not}\ \ \text{true in not } y \\
\rightarrow_P\ & \text{let } y \ = \ \text{false in not } y \\
\rightarrow_P\ & \text{not}\ \ \text{false} \\
\rightarrow_P\ & \text{true}
\end{aligned}
$$

つまり　$\mathtt{is\_even}\ 2\ \rightarrow_P^{*}\ \text{true}$

$$
\dfrac{\text{func}(\mathtt{is\_even}, x, \dots) \in P}{\mathtt{is\_even}\ 2\ \rightarrow_P\ \text{let cond} = 2 = 0 \text{ in if cond then true else let } x' = 2 - 1 \text{ in let } y = \mathtt{is\_odd}\ x' \text{ in not } y}\ (\text{E-FuncApp})
$$

$$
\dfrac{\dfrac{[\![=]\!](2,0) = \text{false}}{2 = 0\ \rightarrow_P\ \text{false}}\ (\text{E-AppOp})}{\text{let cond} = 2 = 0 \text{ in if cond then true else let } x' = 2 - 1 \text{ in let } y = \mathtt{is\_odd}\ x' \text{ in not } y\ \rightarrow_P\ \text{let cond} = \text{false in} \dots}\ (\text{E-Let1})
$$

$$
\text{let cond} = \text{false in if cond then true else let } x' = 2 - 1 \text{ in let } y = \mathtt{is\_odd}\ x' \text{ in not } y\ \rightarrow_P\ \text{if false then true else} \dots \quad (\text{E-Let2})
$$

$$
\text{if false then true else let } x' = 2 - 1 \text{ in let } y = \mathtt{is\_odd}\ x' \text{ in not } y\ \rightarrow_P\ \text{let } x' = 2 - 1 \text{ in let } y = \mathtt{is\_odd}\ x' \text{ in not } y \quad (\text{E-IfFalse})
$$

$$
\dfrac{\dfrac{[\![-]\!](2,1) = 1}{2 - 1\ \rightarrow_P\ 1}\ (\text{E-AppOp})}{\text{let } x' = 2 - 1 \text{ in let } y = \mathtt{is\_odd}\ x' \text{ in not } y\ \rightarrow_P\ \text{let } x' = 1 \text{ in let } y = \mathtt{is\_odd}\ x' \text{ in not } y}\ (\text{E-Let1})
$$

$$
\text{let } x' = 1 \text{ in let } y = \mathtt{is\_odd}\ x' \text{ in not } y\ \rightarrow_P\ \text{let } y = \mathtt{is\_odd}\ 1 \text{ in not } y \quad (\text{E-Let2})
$$

$$
\dfrac{\dfrac{\text{func}(\mathtt{is\_odd}, x, \dots) \in P}{\mathtt{is\_odd}\ 1\ \rightarrow_P\ \dots}\ (\text{E-FuncApp})}{\begin{array}{l}\text{let } y = \mathtt{is\_odd}\ 1 \text{ in} \\ \text{not } y\end{array}\ \rightarrow_P\ \begin{array}{l}\text{let } y = \\ \quad \text{let cond} = 1 = 0 \text{ in} \\ \quad \text{if cond then false else let } x' = 1 - 1 \text{ in let } y = \mathtt{is\_even}\ x' \text{ in not } y \text{ in} \\ \text{not } y\end{array}}\ (\text{E-Let1})
$$

$$\dfrac{\dfrac{\dfrac{[\![=]\!](1,0) = \text{false}}{1 = 0 \ \to_P \ \text{false}}\ (\text{E-AppOp})}{\begin{array}{l}\text{let cond} = 1 = 0 \text{ in}\\ \text{if cond then false}\\ \quad \text{else let } x' = 1-1 \text{ in let } y = \texttt{is\_even}\ x' \text{ in not } y\end{array} \to_P \begin{array}{l}\text{let cond} = \text{false in}\\ \text{if cond then false}\\ \quad \text{else let } x' = 1-1 \text{ in let } y = \texttt{is\_even}\ x' \text{ in not } y\end{array}}\ (\text{E-Let1})}{\begin{array}{l}\text{let } y =\\ \quad \text{let cond} = 1 = 0 \text{ in}\\ \quad \text{if cond then false}\\ \quad \text{else let } x' = 1-1 \text{ in let } y = \texttt{is\_even}\ x' \text{ in not } y \text{ in}\\ \text{not } y\end{array} \to_P \begin{array}{l}\text{let } y =\\ \quad \text{let cond} = \text{false in}\\ \quad \text{if cond then false}\\ \quad \text{else let } x' = 1-1 \text{ in let } y = \texttt{is\_even}\ x' \text{ in not } y \text{ in}\\ \text{not } y\end{array}}\ (\text{E-Let1})$$

$$\dfrac{\begin{array}{l}\text{let cond} = \text{false in}\\ \text{if cond then false}\\ \text{else let } x' = 1-1 \text{ in let } y = \texttt{is\_even}\ x' \text{ in not } y\end{array} \to_P \begin{array}{l}\text{if false then false}\\ \text{else let } x' = 1-1 \text{ in let } y = \texttt{is\_even}\ x' \text{ in not } y\end{array}\quad (\text{E-Let2})}{\begin{array}{l}\text{let } y =\\ \quad \text{let cond} = \text{false in}\\ \quad \text{if cond then false}\\ \quad \text{else let } x' = 1-1 \text{ in let } y = \texttt{is\_even}\ x' \text{ in not } y \text{ in}\\ \text{not } y\end{array} \to_P \begin{array}{l}\text{let } y =\\ \quad \text{if false then false}\\ \quad \text{else let } x' = 1-1 \text{ in let } y = \texttt{is\_even}\ x' \text{ in not } y \text{ in}\\ \text{not } y\end{array}}\ (\text{E-Let1})$$

$$\dfrac{\begin{array}{l}\text{if false then false}\\ \text{else let } x' = 1-1 \text{ in let } y = \texttt{is\_even}\ x' \text{ in not } y\end{array} \to_P \text{let } x' = 1-1 \text{ in let } y = \texttt{is\_even}\ x' \text{ in not } y \quad (\text{E-IfFalse})}{\begin{array}{l}\text{let } y =\\ \quad \text{if false then false}\\ \quad \text{else let } x' = 1-1 \text{ in let } y = \texttt{is\_even}\ x' \text{ in not } y \text{ in}\\ \text{not } y\end{array} \to_P \begin{array}{l}\text{let } y =\\ \quad \text{let } x' = 1-1 \text{ in let } y = \texttt{is\_even}\ x' \text{ in not } y \text{ in}\\ \text{not } y\end{array}}\ (\text{E-Let1})$$

$$\dfrac{\dfrac{\dfrac{[\![-]\!](1,1) = 0}{1 - 1 \ \to_P \ 0}\ (\text{E-AppOp})}{\text{let } x' = 1-1 \text{ in let } y = \texttt{is\_even}\ x' \text{ in not } y \ \to_P \ \text{let } x' = 0 \text{ in let } y = \texttt{is\_even}\ x' \text{ in not } y}\ (\text{E-Let1})}{\begin{array}{l}\text{let } y =\\ \quad \text{let } x' = 1-1 \text{ in let } y = \texttt{is\_even}\ x' \text{ in not } y \text{ in}\\ \text{not } y\end{array} \to_P \begin{array}{l}\text{let } y =\\ \quad \text{let } x' = 0 \text{ in let } y = \texttt{is\_even}\ x' \text{ in not } y \text{ in}\\ \text{not } y\end{array}}\ (\text{E-Let1})$$

$$\dfrac{\text{let } x' = 0 \text{ in let } y = \texttt{is\_even}\ x' \text{ in not } y \ \to_P \ \text{let } y = \texttt{is\_even}\ 0 \text{ in not } y \quad (\text{E-Let2})}{\begin{array}{l}\text{let } y =\\ \quad \text{let } x' = 0 \text{ in let } y = \texttt{is\_even}\ x' \text{ in not } y \text{ in}\\ \text{not } y\end{array} \to_P \begin{array}{l}\text{let } y =\\ \quad \text{let } y = \texttt{is\_even}\ 0 \text{ in not } y \text{ in}\\ \text{not } y\end{array}}\ (\text{E-Let1})$$

$$
\dfrac{\dfrac{\dfrac{\mathrm{func}(\texttt{is\_even}, x, \dots) \in P}{\texttt{is\_even}\ 0 \quad \to_P \ \text{let cond} = 0 = 0 \text{ in if cond then true else} \dots} \ (\text{E-FuncApp})}{\text{let } y = \texttt{is\_even}\ 0 \text{ in not } y \quad \to_P \quad \begin{array}{l} \text{let } y = \\ \quad \text{let cond} = 0 = 0 \text{ in if cond then true else} \dots \text{ in} \\ \text{not } y \end{array}} \ (\text{E-Let1})}{\begin{array}{l} \text{let } y = \\ \quad \text{let } y = \texttt{is\_even}\ 0 \text{ in not } y \text{ in} \\ \text{not } y \end{array} \quad \to_P \quad \begin{array}{l} \text{let } y = \\ \quad \text{let } y = \\ \quad\quad \text{let cond} = 0 = 0 \text{ in if cond then true else} \dots \text{ in} \\ \quad \text{not } y \text{ in} \\ \text{not } y \end{array}} \ (\text{E-Let1})
$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{[\![=]\!](0,0) = \text{true}}{0 = 0 \ \to_P \ \text{true}} \ (\text{E-AppOp})}{\text{let cond} = 0 = 0 \text{ in if cond then true else} \dots \quad \to_P \quad \text{let cond} = \text{true in if cond then true else} \dots} \ (\text{E-Let1})}{\begin{array}{l} \text{let } y = \\ \quad \text{let cond} = 0 = 0 \text{ in if cond then true else} \dots \text{ in} \\ \text{not } y \end{array} \quad \to_P \quad \begin{array}{l} \text{let } y = \\ \quad \text{let cond} = \text{true in if cond then true else} \dots \text{ in} \\ \text{not } y \end{array}} \ (\text{E-Let1})}{\begin{array}{l} \text{let } y = \\ \quad \text{let } y = \\ \quad\quad \text{let cond} = 0 = 0 \text{ in if cond then true else} \dots \text{ in} \\ \quad \text{not } y \text{ in} \\ \text{not } y \end{array} \quad \to_P \quad \begin{array}{l} \text{let } y = \\ \quad \text{let } y = \\ \quad\quad \text{let cond} = \text{true in if cond then true else} \dots \text{ in} \\ \quad \text{not } y \text{ in} \\ \text{not } y \end{array}} \ (\text{E-Let1})
$$

$$
\dfrac{\dfrac{\dfrac{\text{let cond} = \text{true in if cond then true else} \dots \quad \to_P \quad \text{if true then true else} \dots \quad (\text{E-Let2})}{\begin{array}{l} \text{let } y = \\ \quad \text{let cond} = \text{true in if cond then true else} \dots \text{ in} \\ \text{not } y \end{array} \quad \to_P \quad \begin{array}{l} \text{let } y = \\ \quad \text{if true then true else} \dots \text{ in} \\ \text{not } y \end{array}} \ (\text{E-Let1})}{\begin{array}{l} \text{let } y = \\ \quad \text{let cond} = \text{true in if cond then true else} \dots \text{ in} \\ \text{not } y \end{array} \quad \to_P \quad \begin{array}{l} \text{let } y = \\ \quad \text{if true then true else} \dots \text{ in} \\ \text{not } y \end{array}} \ (\text{E-Let1})}{\begin{array}{l} \text{let } y = \\ \quad \text{let } y = \\ \quad\quad \text{let cond} = \text{true in if cond then true else} \dots \text{ in} \\ \quad \text{not } y \text{ in} \\ \text{not } y \end{array} \quad \to_P \quad \begin{array}{l} \text{let } y = \\ \quad \text{let } y = \\ \quad\quad \text{if true then true else} \dots \text{ in} \\ \quad \text{not } y \text{ in} \\ \text{not } y \end{array}} \ (\text{E-Let1})
$$

$$
\dfrac{\dfrac{\text{if true then true else} \dots \quad \to_P \quad \text{true} \quad (\text{E-IfTrue})}{\text{let } y = \text{if true then true else} \dots \text{ in not } y \quad \to_P \quad \text{let } y = \text{true in not } y} \ (\text{E-Let1})}{\begin{array}{l} \text{let } y = \\ \quad \text{let } y = \\ \quad\quad \text{if true then true else} \dots \text{ in} \\ \quad \text{not } y \text{ in} \\ \text{not } y \end{array} \quad \to_P \quad \begin{array}{l} \text{let } y = \\ \quad \text{let } y = \text{true in not } y \text{ in} \\ \text{not } y \end{array}} \ (\text{E-Let1})
$$

$$\dfrac{\text{let } y = \text{true in not } y \quad \to_P \quad \text{not true} \quad \text{(E-Let2)}}{\begin{array}{l}\text{let } y = \\ \quad \text{let } y = \text{true in not } y \text{ in} \\ \text{not } y\end{array} \quad \to_P \quad \text{let } y = \text{not true in not } y} \text{(E-Let1)}$$

$$\dfrac{\dfrac{[\![\text{not}]\!](\text{true}) = \text{false}}{\text{not true} \ \to_P \ \text{false}} \text{(E-}\textsc{AppOp}\text{)}}{\text{let } y = \text{not true in not } y \ \to_P \ \text{let } y = \text{false in not } y} \text{(E-Let1)}$$

$$\text{let } y = \text{false in not } y \ \to_P \ \text{not false} \quad \text{(E-Let2)}$$

$$\dfrac{[\![\text{not}]\!](\text{false}) = \text{true}}{\text{not false} \ \to_P \ \text{true}} \text{(E-}\textsc{AppOp}\text{)}$$

付録 B　Yil の単純型付けの例

TODO: やる

付録 C　型付けの例

$$P \quad \left\{\begin{array}{l} \text{func}(\texttt{is\_even}, x, \\ \quad \text{let cond} \ = \ x = 0 \text{ in} \\ \quad \text{if cond then true} \\ \quad \text{else} \\ \quad\quad \text{let } x' \ = \ x - 1 \text{ in} \\ \quad\quad \text{let } y \ = \ \texttt{is\_odd } x' \text{ in} \\ \quad\quad \text{not } y \ ), \\ \text{func}(\texttt{is\_odd}, x, \\ \quad \text{let cond} \ = \ x = 0 \text{ in} \\ \quad \text{if cond then false} \\ \quad \text{else} \\ \quad\quad \text{let } x' \ = \ x - 1 \text{ in} \\ \quad\quad \text{let } y \ = \ \texttt{is\_even } x' \text{ in} \\ \quad\quad \text{not } y \ ) \end{array}\right. ,$$

$$\tau_{\texttt{is\_even}} \stackrel{\text{def}}{=} (x : \{\nu_x | \nu_x \geq 0\}) \to \left\{ \nu \;\middle|\; \begin{array}{l} ((\nu_x \% 2 = 0) \Rightarrow \nu = 0) \wedge \\ ((\nu_x \% 2 \neq 0) \Rightarrow \nu = 1) \end{array} \right\}$$

$$\tau_{\texttt{is\_odd}} \stackrel{\text{def}}{=} (x : \{\nu_x | \nu_x \geq 0\}) \to \left\{ \nu \;\middle|\; \begin{array}{l} ((\nu_x \% 2 = 0) \Rightarrow \nu = 1) \wedge \\ ((\nu_x \% 2 \neq 0) \Rightarrow \nu = 0) \end{array} \right\}$$

$$\phi_{\texttt{cond}} \stackrel{\text{def}}{=} \begin{array}{l} (\nu_x = 0) \Rightarrow \nu_{\texttt{cond}} = 0 \wedge \\ (\nu_x \neq 0) \Rightarrow \nu_{\texttt{cond}} = 1 \end{array}$$

$$\tau_{\texttt{cond}} \stackrel{\text{def}}{=} \{\nu_{\texttt{cond}} \, | \phi_{\texttt{cond}}\}$$

$\tau_{\mathrm{if}} \stackrel{\mathrm{def}}{=} \{\nu_{\mathtt{if}} \mid$ *TODO：書く*$\}$

$\Gamma' \stackrel{\mathrm{def}}{=} x : \{\nu_x \mid \nu_x \geq 0\}, \mathrm{cond} : \tau_{\mathrm{cond}}, \phi_{\mathrm{cond}} \land \nu_{\mathrm{cond}} \neq 0, x' : \{\nu'_x \mid \nu'_x = \nu_x - 1\}$

$$
\dfrac{
\vdash_P \mathtt{is\_even} : \tau_{\mathtt{is\_even}}
\qquad
\dfrac{
\vdash_P 2 : \{\nu_x \mid \nu_x = 2\}
\quad
\dfrac{\vDash \nu_x = 2 \Rightarrow \nu_x \geq 0}{\vdash_P \{\nu_x \mid \nu_x = 2\} <: \{\nu_x \mid \nu_x \geq 0\}}\,\textsc{SubTy-Int}
}{\vdash_P 2 : \{\nu_x \mid \nu_x \geq 0\}}\,\textsc{Ty-Sub}
}{
\vdash_P \mathtt{is\_even}\ 2 : \left\{\nu \;\middle|\; \begin{array}{l} ((2\%2 = 0) \Rightarrow \nu = 0) \land \\ ((2\%2 \neq 0) \Rightarrow \nu = 1) \end{array}\right\}
}\,\textsc{Ty-App}
\qquad
\dfrac{
\vDash \left(\begin{array}{l} ((2\%2 = 0) \Rightarrow \nu = 0) \land \\ ((2\%2 \neq 0) \Rightarrow \nu = 1) \end{array}\right) \Rightarrow \nu = 0
}{
\vdash_P \left\{\nu \;\middle|\; \begin{array}{l} ((2\%2 = 0) \Rightarrow \nu = 0) \land \\ ((2\%2 \neq 0) \Rightarrow \nu = 1) \end{array}\right\} <: \{\nu \mid \nu = 0\}
}\,\textsc{SubTy-Int}
$$
$$
\overline{\qquad\qquad\qquad \vdash_P \mathtt{is\_even}\ 2 : \{\nu \mid \nu = 0\} \qquad\qquad\qquad}\,\textsc{Ty-Sub}
$$

$$
\dfrac{
\dfrac{
x : \{\nu_x \mid \nu_x \geq 0\} \vdash_P x = 0 : \tau_{\mathrm{cond}} \qquad \mathrm{A}
}{
x : \{\nu_x \mid \nu_x \geq 0\} \vdash_P
\begin{array}{l}
\mathrm{let\ cond\ } = x = 0 \mathrm{\ in} \\
\mathrm{if\ cond\ then\ true} \\
\mathrm{else} \\
\quad \mathrm{let\ } x' = x - 1 \mathrm{\ in} \\
\quad \mathrm{let\ } y = \mathtt{is\_odd}\ x' \mathrm{\ in} \\
\quad \mathrm{not\ } y
\end{array}
: \left\{\nu \;\middle|\; \begin{array}{l} ((\nu_x\%2 = 0) \Rightarrow \nu = 0) \land \\ ((\nu_x\%2 \neq 0) \Rightarrow \nu = 1) \end{array}\right\}
}\,\textsc{Ty-Let}
}{
\vdash_P \mathtt{is\_even} : \tau_{\mathtt{is\_even}}
}\,\textsc{Ty-Func}
$$

A :

$$
\dfrac{
\dfrac{\Gamma(\mathrm{cond}) = \tau_{\mathrm{cond}}}{\begin{array}{l} x : \{\nu_x \mid \nu_x \geq 0\}, \\ \mathrm{cond} : \tau_{\mathrm{cond}} \end{array} \vdash_P \mathrm{cond} : \tau_{\mathrm{cond}}}\,\textsc{Ty-Var}
\quad
\dfrac{\mathrm{B}}{\begin{array}{l} x : \{\nu_x \mid \nu_x \geq 0\}, \\ \mathrm{cond} : \tau_{\mathrm{cond}}, \\ \phi_{\mathrm{cond}} \land \nu_{\mathrm{cond}} = 0 \end{array} \vdash_P \mathrm{true} : \tau_{\mathrm{if}}}
\quad
\dfrac{\mathrm{C}}{\begin{array}{l} x : \{\nu_x \mid \nu_x \geq 0\}, \\ \mathrm{cond} : \tau_{\mathrm{cond}}, \\ \phi_{\mathrm{cond}} \land \nu_{\mathrm{cond}} \neq 0 \end{array} \vdash_P \begin{array}{l} \mathrm{let\ } x' = x - 1 \mathrm{\ in} \\ \mathrm{let\ } y = \mathtt{is\_odd}\ x' \mathrm{\ in} \\ \ldots \end{array} : \tau_{\mathrm{if}}}
}{
x : \{\nu_x \mid \nu_x \geq 0\}, \mathrm{cond} : \tau_{\mathrm{cond}} \vdash_P \mathrm{if\ cond\ then\ true\ else} \ldots : \tau_{\mathrm{if}}
}\,\textsc{Ty-If}
$$

B :

$$
\dfrac{\textit{TODO：書く}}{x : \{\nu_x \mid \nu_x \geq 0\}, \mathrm{cond} : \tau_{\mathrm{cond}}, \phi_{\mathrm{cond}} \land \nu_{\mathrm{cond}} = 0 \vdash_P \mathrm{true} : \tau_{\mathrm{if}}}
$$

C :

$$
\dfrac{
\dfrac{\textit{TODO：書く}}{\begin{array}{l} x : \{\nu_x \mid \nu_x \geq 0\}, \\ \mathrm{cond} : \tau_{\mathrm{cond}}, \\ \phi_{\mathrm{cond}} \land \nu_{\mathrm{cond}} \neq 0 \end{array} \vdash_P x - 1 : \{\nu'_x \mid \nu'_x = \nu_x - 1\}}\,\textsc{Ty-App}
\quad
\dfrac{\mathrm{D} \quad \mathrm{E}}{\Gamma' \vdash_P \begin{array}{l} \mathrm{let\ } y = \mathtt{is\_odd}\ x' \mathrm{\ in} \\ \mathrm{not\ } y \end{array} : \textit{TODO：書く}}\,\textsc{Ty-Let}
}{
x : \{\nu_x \mid \nu_x \geq 0\}, \mathrm{cond} : \tau_{\mathrm{cond}}, \phi_{\mathrm{cond}} \land \nu_{\mathrm{cond}} \neq 0 \vdash_P \mathrm{let\ } x' = x - 1 \mathrm{\ in\ let\ } y = \mathtt{is\_odd}\ x' \mathrm{\ in} \ldots : \tau_{\mathrm{if}}
}\,\textsc{Ty-Let}
$$

D :

$$
\dfrac{
\Gamma' \vdash_P \mathtt{is\_odd} : \tau_{\mathtt{is\_odd}}
\quad
\dfrac{
\dfrac{\Gamma'(x') = \{\nu'_x \mid \nu'_x = \nu_x - 1\}}{\Gamma' \vdash_P x' : \{\nu'_x \mid \nu'_x = \nu_x - 1\}}\,\textsc{Ty-Var}
\quad
\dfrac{\vDash ([\![\Gamma']\!] \land \nu'_x = \nu_x - 1) \Rightarrow \nu'_x \geq 0}{\Gamma' \vdash_P \{\nu'_x \mid \nu'_x = \nu_x - 1\} <: \{\nu'_x \mid \nu'_x \geq 0\}}\,\textsc{SubTy-Int}
}{
\Gamma' \vdash_P x' : \{\nu'_x \mid \nu'_x \geq 0\}
}\,\textsc{Ty-Sub}
}{
\Gamma' \vdash_P \mathtt{is\_odd}\ x' : \left\{\nu_y \;\middle|\; \begin{array}{l} ((\nu'_x\%2 = 0) \Rightarrow \nu_y = 1) \land \\ ((\nu'_x\%2 \neq 0) \Rightarrow \nu_y = 0) \end{array}\right\}
}\,\textsc{Ty-App}
$$

E :

$$\dfrac{}{\Gamma, y : \left\{ \nu_y \;\middle|\; \begin{array}{l} ((\nu_x' \% 2 = 0) \Rightarrow \nu_y = 1) \wedge \\ ((\nu_x' \% 2 \neq 0) \Rightarrow \nu_y = 0) \end{array} \right\} \vdash_P \text{not } y : \textit{TODO}：書く} \textsc{Ty-App}$$

References

[1] Thomas Johnsson. "Lambda Lifting: Treansforming Programs to Recursive Equations". In: *Functional Programming Languages and Computer Architecture, FPCA 1985, Nancy, France, September 16-19, 1985, Proceedings*. Ed. by Jean-Pierre Jouannaud. Vol. 201. Lecture Notes in Computer Science. Springer, 1985, pp. 190–203. DOI: 10.1007/3-540-15975-4_37. URL: https://doi.org/10.1007/3-540-15975-4%5C_37.

[2] C.-H. Luke Ong and Steven J. Ramsay. "Verifying higher-order functional programs with pattern-matching algebraic data types". In: *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*. Ed. by Thomas Ball and Mooly Sagiv. ACM, 2011, pp. 587–598. DOI: 10.1145/1926385.1926453. URL: https://doi.org/10.1145/1926385.1926453.

[3] Hiroshi Unno and Naoki Kobayashi. "Dependent type inference with interpolants". In: *Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, September 7-9, 2009, Coimbra, Portugal*. Ed. by António Porto and Francisco Javier López-Fraguas. ACM, 2009, pp. 277–288. DOI: 10.1145/1599410.1599445. URL: https://doi.org/10.1145/1599410.1599445.

[4] *z3*. https://github.com/Z3Prover/z3.

[5] *CVC4*. https://cvc4.github.io.