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

2021/9/7

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.

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

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, if v then e_1 else e_2 は条件分岐を表す. if v then e_1 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.

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 [1] や [2] 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 2 で定義する.

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

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

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

図 2　Yil の操作的意味

$e_1 \ \to_P \ e_2$ 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 $e_1 \ \to_P^* \ e_2$ 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 3 で定義する.

int は整数を表す単純型である. $T_1 \to T_2$ 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.

$$\text{(単純型) } T ::= \text{int} \mid T \to T$$
$$\text{(単純型環境) } \Delta ::= \varnothing \mid \Delta, x : T$$

図 3　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.

$T_1 \to (T_2 \to (T_3 \to \ldots (T_n \to T)))$ を $\overline{T} \to T$ と略記する. $\text{func}(f, \overline{x}, e) \in P$ 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 $\text{func}(f, \overline{x}, e)$ に単純型 $\overline{T} \to T$ がつくといい, $\text{sty}(f) = \overline{T} \to T$ と書く.また, 組み込み関数の変数 $f \in \text{op}$ の単純型も $\text{sty}(f)$ 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.

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

$$\frac{\Delta \vdash_P e_1 : T_1 \to T_2 \quad \Delta \vdash_P e_2 : T_1}{\Delta \vdash_P e_1\ e_2 : T_2}\ \text{(ST-App)} \qquad \frac{\Delta \vdash_P e_1 : \text{int} \quad \Delta \vdash_P e_2 : T \quad \Delta \vdash_P e_3 : T}{\Delta \vdash_P \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : T}\ \text{(ST-If)}$$

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

図 4　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.

$\{x \mid \phi\}$ 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 $\{x \mid x \geq 10\}$ は 10 以上の整数につく型の整数につく型である. $\{x \mid \top\}$, 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. $(x_1 : \tau_1) \to ((x_2 : \tau_2) \to \ldots ((x_n : \tau_n) \to \tau))$ を $(\overline{x} : \overline{\tau}) \to \tau$ と略記する.

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

図 5　型の構文

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 $\Gamma, x_1 : \tau_1, x_2 : \tau_2, \ldots, x_n : \tau_n, \Gamma'$ 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 [3] や CVC4 [4] 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 $\neg\phi_1 \vee \phi_2, t_1 + 1 \le t_2, t_1 \le t_2 \wedge t_2 \le t_1$ と定義する.

3.2　型付け規則

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

$$
\frac{\Gamma \vdash_P e_1 : (x : \tau_1) \to \tau_2 \quad \Gamma \vdash_P e_2 : \tau_1}{\Gamma \vdash_P e_1\ e_2 : \tau_2}\ \textsc{Ty-App} \qquad \frac{\Gamma \vdash_P e_1 : \{\nu \mid \phi\} \quad \Gamma, \phi \wedge \nu = 0 \vdash_P e_2 : \tau \quad \Gamma, \phi \wedge \nu = 1 \vdash_P e_2 : \tau}{\Gamma \vdash_P \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : \tau}\ \textsc{Ty-If}
$$

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

$$
\frac{\vDash [\![\Gamma]\!] \wedge \phi_1 \Rightarrow \phi_2}{\Gamma \vdash_P \{\nu \mid \phi_1\} <: \{\nu \mid \phi_2\}}\ \textsc{SubTy-Int} \qquad \frac{\Gamma \vdash_P \tau_{21} <: \tau_{11} \quad \Gamma, \nu : \tau_{21} \vdash_P \tau_{12} <: \tau_{22}}{\Gamma \vdash_P (\nu : \tau_{11}) \to \tau_{12} <: (\nu : \tau_{21}) \to \tau_{22}}\ \textsc{SubTy-Func}
$$

図 6　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 $e_1 \ \to_P \ e_2$ 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 $\Gamma_1, [x/v]\Gamma_2 \vdash_P [x/v]e : [x/v]\tau$

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

以下は 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です. TODO: 構文と評価規則の変更に合わせて修正する

付録 B　Yil の単純型付けの例

TODO: やる

付録 C　型付けの例

$$P \stackrel{\text{def}}{=} \begin{array}{l} \text{func}(\texttt{is\_even}, x, \text{if } x = 0 \text{ then true else not } (\texttt{is\_odd}\ (x-1))), \\ \text{func}(\texttt{is\_odd}, x, \text{if } x = 0 \text{ then false else not } (\texttt{is\_even}\ (x-1))) \end{array}$$

$$\dfrac{\dfrac{\textit{TODO : 書く}}{\Gamma \vdash_P \texttt{is\_even} : (x : \{\nu \mid \nu \geq 0\}) \to \left\{\nu' \mid \begin{array}{l}((\nu \% 2 = 0) \Rightarrow \nu' = 0) \wedge \\ ((\nu \% 2 \neq 0) \Rightarrow \nu' = 1)\end{array}\right\}}\ \textsc{Ty-Func} \qquad \Gamma \vdash_P 2 : \{\nu \mid \nu = 2\}\ \ \textsc{Ty-Num}}{\vdash_P \texttt{is\_even}\ 2 : \{\nu \mid \nu = 0\}}\ \textsc{Ty-App}$$

References

[1] 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.

[2] 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.

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

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