YuchiKaml

Yuchiki

2018 Dec.

Contents

1 Introduction

YuchiKaml is a toy language. and YuchiKaml interpreter is an implementation of interpreter of YuchiKaml. Both are created in order to get accustomed to Sprache, a C#Parser Combinator Library. In this article, I introduce both the language and the interpreter.

2 YuchiKaml Language

YuchiKaml is a dynamic typed language with-ML like surface grammar.

2.1 Syntax

In this section, we define the syntax of YuchikiML.

2.1.1 Expression

Expressions of YuchiKaml are defined by the following BNF equations:

$$
\begin{aligned}
e ::= {} & () \mid x \mid n \mid \mathrm{true} \mid \mathrm{false} \mid s \mid (e) \\
& \mid e\ e \mid !\,e \\
& \mid e * e \mid e/e \\
& \mid e + e \mid e - e \\
& \mid e \leq e \mid e < e \mid e \geq e \mid e > e \\
& \mid e = e \mid e \neq e \\
& \mid e \mathbin{\&\&} e \\
& \mid e \| e \\
& \mid e \triangleright e \mid e \gg e \\
& \mid \mathrm{if}\, e\, \mathrm{then}\, e\, \mathrm{else}\, e \mid \mathrm{let}\, x\ \tilde{a} = e\, \mathrm{in}\, e \mid \mathrm{let\,rec}\, f\ a_1\ \tilde{a} = e\, \mathrm{in}\, e \mid \lambda x.\ e \\
& \mid e\, ;\, e
\end{aligned}
$$

The operators defined in earlier rows have stronger precedences than the operators defined in later rows. For example, $1+2*3$ is not parsed as $(1+2)*3$, but $1+(2*3)$.

Example 2.1 (GCD)**.** This is an example of a YuchiKaml source code of a program which calculates the greatest common divisor of 120 and 45.

Listing 1: Samples/gcd

```
let rec gcd m n =
      if m > n then gcd (m - n) n
l       else if m < n then gcd m (n - m)
l       else m
in gcd 120 45
```

2.1.2 Syntax Sugar

Some of the expressions shown above are syntactic sugars. We show the list and syntax sugars and how they are desugared.

$$
\begin{aligned}
e_1 \triangleright e_2 &::= e_2\ e_1 \\
e_1 \gg e_2 &::= \lambda x.\ g(fx) \\
\mathrm{let}\, f\ a_1 \cdots a_n = e_1 \,\mathrm{in}\, e_2 &::= \mathrm{let}\, f = \lambda a_1. \cdots \lambda a_n.\ e_1 \,\mathrm{in}\, e_2 \\
\mathrm{let\,rec}\, f\ a\ b_1 \cdots b_n = e_1 \,\mathrm{in}\, e_2 &::= \mathrm{let\,rec}\, f\ a = \lambda b_1. \cdots \lambda b_n.\ e_1 \,\mathrm{in}\, e_2 \\
e_1\, ; e_2 &::= \mathrm{let}\, \_ = e_1 \,\mathrm{in}\, e_2
\end{aligned}
$$

The semantics of expressions, introduced in a later section, is defined for desugared expressions.

2.1.3 real notation

In real source codes, the symbols above are notated as follows:

$\leq$	`<=`
$\geq$	`>=`
$\neq$	`! =`
&	`&&`
$\|\|$	`||`
$\triangleright$	`| >`
$\gg$	`>>`
$\lambda x.\ e$	$\backslash x - > e$

2.2 Semantics

Then we define the semantics of expressions. We assume that the behaviour of built-in functions are given. That is, built-in functions is a deterministic.

2.2.1 Value

Values of expressions are listed as below:

$$\begin{aligned}
v(\text{value}) &::= n \mid b \mid s \mid \text{cl} \mid f_b \\
\rho(\text{valuation}) &\in \text{Var} \not\rightarrow \text{Val} \\
f_b(\text{built-in function}) &\in \text{His} \times \text{Val} \not\rightarrow \text{Val} \\
\text{cl}(\text{closure}) &::= (x, e, \rho) \\
h(\text{history}) &::= \langle (f_1, v_1); \cdots ; (f_n, v_n) \rangle
\end{aligned}$$

Here Var is the set of the variables and Val is the set of the values.

Note 2.1. We assume that all the built-in functions are at least function. It means, all the built-in functions return the same value for the same input, if it does not diverge. This is not a correct assumption. for example, Random function and Read function may return variable values. the aim of this assumption is to define the possible implementations of processing systems. that is, intuitionally, to judge the processing system OK if it works as the same as the expected behaviour under this assumption.

2.2.2 Small-Step Evaluation

We have to define the behaviour of evaluation process of expressions clearly. To this end, we define the *evaluation* process of expression by a big-step semantics shown below.

An *evaluation relation* is a four-term relation of the form $\rho \vdash e_1 \longrightarrow e_2$.

The evaluation rules of YuchiKaml are shown below:

TODO: modify totally. count Histories

$$\frac{\rho(x) = v}{\rho \vdash x \longrightarrow v} \qquad (\text{E-VAR})$$

$$\frac{\rho \vdash e_1 \longrightarrow e_1'}{\rho \vdash e_1\ e_2 \longrightarrow e_1'\ e_2} \qquad (\text{E-APPLEFT})$$

$$\frac{\rho \vdash e_2 \longrightarrow e_2'}{\rho \vdash v_1\ e_2 \longrightarrow v_1'\ e_2'} \qquad (\text{E-APPRIGHT})$$

$$\frac{\rho' \cup \{x \mapsto v\} \vdash e_1 \longrightarrow e_1'}{\rho \vdash (x, e_1, \rho')\ v_2 \longrightarrow (x, e_1', \rho')v_2} \qquad (\text{E-APPCLS})$$

$$\frac{}{\rho \vdash (x, v_1, \rho')\ v_2 \longrightarrow v_1} \qquad (\text{E-APPCLS2})$$

$$\frac{f_1 @ v_2 = v_3}{\rho \vdash f_1\ v_2 \longrightarrow v_3} \qquad (\text{E-APPBUILTIN})$$

$$\frac{\rho \vdash e_1 \longrightarrow e_1'}{\rho \vdash e_1 \,\mathrm{op}\, e_2 \longrightarrow e_1' \,\mathrm{op}\, e_2} \quad (\textsc{E-BinOp-Left})$$

$$\frac{\rho \vdash e_2 \longrightarrow e_2'}{\rho \vdash v_1 \,\mathrm{op}\, e_2 \longrightarrow v_1 \,\mathrm{op}\, e_2'} \quad (\textsc{E-BinOp-Right})$$

$$\frac{n_1 \llbracket * \rrbracket n_2 = n_3}{\rho \vdash n_1 * n_2 \longrightarrow n_3} \quad (\textsc{E-Mul})$$

$$\frac{n_1 \llbracket / \rrbracket n_2 = n_3}{\rho \vdash n_1 / n_2 \longrightarrow n_3} \quad (\textsc{E-Div})$$

$$\frac{n_1 \llbracket + \rrbracket n_2 = n_3}{\rho \vdash n_1 + n_2 \longrightarrow n_3} \quad (\textsc{E-Plus})$$

$$\frac{n_1 \llbracket - \rrbracket n_2 = n_3}{\rho \vdash n_1 - n_2 \longrightarrow n_3} \quad (\textsc{E-Minus})$$

$$\frac{n_1 \llbracket \leq \rrbracket n_2 = b_3}{\rho \vdash n_1 \leq n_2 \longrightarrow b_3} \quad (\textsc{E-Leq})$$

$$\frac{n_1 \llbracket < \rrbracket n_2 = b_3}{\rho \vdash n_1 < n_2 \longrightarrow b_3} \quad (\textsc{E-Lt})$$

$$\frac{n_1 \llbracket \geq \rrbracket n_2 = b_3}{\rho \vdash n_1 \geq n_2 \longrightarrow b_3} \quad (\textsc{E-Geq})$$

$$\frac{n_1 \llbracket > \rrbracket n_2 = b_3}{\rho \vdash n_1 > n_2 \longrightarrow b_3} \quad (\textsc{E-Gt})$$

$$\frac{b_1 \llbracket \&\& \rrbracket b_2 = b_3}{\rho \vdash b_1 \,\&\&\, b_2 \longrightarrow b_3} \quad (\textsc{E-And})$$

$$\frac{b_1 \llbracket || \rrbracket b_2 = b_3}{\rho \vdash b_1 \,||\, b_2 \longrightarrow b_3} \quad (\textsc{E-Or})$$

$$\frac{(v_1 = v_2) = b_3}{\rho \vdash v_1 = v_2 \longrightarrow b_3} \quad (\textsc{E-Eq})$$

$$\frac{(v_1 \neq v_2) = b_3}{\rho \vdash v_1 \neq v_2 \longrightarrow b_3} \quad (\textsc{E-Neq})$$

$$\frac{\rho \vdash e_1 \longrightarrow e_1'}{\rho \vdash \mathrm{if}\, e_1 \,\mathrm{then}\, e_2 \,\mathrm{else}\, e_3 \longrightarrow \mathrm{if}\, e_1' \,\mathrm{then}\, e_2 \,\mathrm{else}\, e_3} \tag{E-IFCond}$$

$$\frac{}{\rho \vdash \mathrm{if\,true\,then}\, e_2 \,\mathrm{else}\, e_3 \longrightarrow e_2} \tag{E-IfTrue}$$

$$\frac{}{\rho \vdash \mathrm{if\,true\,then}\, e_2 \,\mathrm{else}\, e_3 \longrightarrow e_3} \tag{E-IfFalse}$$

$$\frac{\rho \vdash e_1 \longrightarrow e_1'}{\rho \vdash \mathrm{let}\, x = e_1 \in e_2 \longrightarrow \mathrm{let}\, x = e_1' \,\mathrm{in}\, e_2} \tag{E-LetBody}$$

$$\frac{\rho \cup \{x \mapsto v_1\} \vdash e_2 \longrightarrow e_2'}{\rho \vdash \mathrm{let}\, x = v_1 \,\mathrm{in}\, e_2 \longrightarrow \mathrm{let}\, x = v_1 \,\mathrm{in}\, e_2'} \tag{E-LetRem}$$

$$\frac{}{\rho \vdash \mathrm{let}\, x = v_1 \,\mathrm{in}\, v_2 \longrightarrow v_2} \tag{E-LetValue}$$

$$\tag{E-LetRec}$$

TODO: define it.

$$\frac{}{\rho \vdash \lambda x.\ e \longrightarrow (x, e, \rho)} \tag{E-Abs}$$

Using this small-step evaluation relation, we define the next big-step evaluation relation.

Note 2.2. For any expression e_1, if ρ, e_2 and e_3 satisfy $\rho \vdash e_1 \longrightarrow e_2$ and $\rho \vdash e_1 \longrightarrow e_3$, then $e_2 = e_3$.

2.2.3 Big-Step Evaluation

3 YuchiKaml Interpreter

3.1 Usage

3.2 Preprocess