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:

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

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: gcd

```
let rec gcd m n =
    if m > n then gcd (m - n) n
    else if m < n then gcd m (n - m)
    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 Actual Notation

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

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

2.1.4 Comment

YuchiKaml has two kinds of comments.

Line Comment Characters from *//* to the end of line are ignored.

Bracket Comment Characters from *(** to **)* are ignored.

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 are assumed deterministic.

2.2.1 Value

Values of expressions are listed as below:

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

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 return the same value for the same input, if it does not diverge. This is enough to see weather different two ways of execution of a program is equivalent or not ,given a situation.

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 $\rho \vdash h; e \longrightarrow h'; v$.

The evaluation rules of YuchiKaml are shown below:

$$\frac{}{\rho \vdash h; x \longrightarrow h; \rho(x)} \quad \text{(E-Var)}$$

$$\frac{\rho \vdash h; e_1 \longrightarrow h'; (x, e_1', \rho') \qquad \rho \vdash h'; e_2 \longrightarrow h''; v_2}{\rho \vdash h; e_1\ e_2 \longrightarrow h'''; v_1'} \quad \text{(E-App)}$$

$$\frac{\rho \vdash h; e \longrightarrow h'; b}{\rho \vdash h; !e \longrightarrow h';\ [\![!]\!]\, b} \quad \text{(E-Not)}$$

$$\frac{\rho \vdash h; e_1 \longrightarrow h'; n_1 \qquad \rho \vdash h'; e_2 \longrightarrow h''; n_2}{\rho \vdash h; e_1 * e_2 \longrightarrow h''; n_1\, [\![*]\!]\, n_2} \quad \text{(E-Mul)}$$

$$\frac{\rho \vdash h; e_1 \longrightarrow h'; n_1 \qquad \rho \vdash h'; e_2 \longrightarrow h''; n_2}{\rho \vdash h; e_1 / e_2 \longrightarrow h''; n_1\, [\![/]\!]\, n_2} \quad \text{(E-Div)}$$

$$\frac{\rho \vdash h; e_1 \longrightarrow h'; n_1 \qquad \rho \vdash h'; e_2 \longrightarrow h''; n_2}{\rho \vdash h; e_1 + e_2 \longrightarrow h''; n_1\, [\![+]\!]\, n_2} \quad \text{(E-Plus)}$$

$$\frac{\rho \vdash h; e_1 \longrightarrow h'; n_1 \qquad \rho \vdash h'; e_2 \longrightarrow h''; n_2}{\rho \vdash h; e_1 - e_2 \longrightarrow h''; n_1\, [\![-]\!]\, n_2} \quad \text{(E-Minus)}$$

$$\frac{\rho \vdash h; e_1 \longrightarrow h'; n_1 \qquad \rho \vdash h'; e_2 \longrightarrow h''; n_2}{\rho \vdash h; e_1 \leq e_2 \longrightarrow h''; n_1\, [\![\leq]\!]\, n_2} \quad \text{(E-Leq)}$$

$$\frac{\rho \vdash h; e_1 \longrightarrow h'; n_1 \qquad \rho \vdash h'; e_2 \longrightarrow h''; n_2}{\rho \vdash h; e_1 < e_2 \longrightarrow h''; n_1\, [\![<]\!]\, n_2} \quad \text{(E-Lt)}$$

$$\frac{\rho \vdash h; e_1 \longrightarrow h'; n_1 \qquad \rho \vdash h'; e_2 \longrightarrow h''; n_2}{\rho \vdash h; e_1 \geq e_2 \longrightarrow h''; n_1\, [\![\geq]\!]\, n_2} \quad \text{(E-Geq)}$$

$$\frac{\rho \vdash h; e_1 \longrightarrow h'; n_1 \qquad \rho \vdash h'; e_2 \longrightarrow h''; n_2}{\rho \vdash h; e_1 > e_2 \longrightarrow h''; n_1\, [\![>]\!]\, n_2} \quad \text{(E-Gt)}$$

$$\frac{\rho \vdash h; e_1 \longrightarrow h'; v_1 \qquad \rho \vdash h'; e_2 \longrightarrow h''; v_2}{\rho \vdash h; e_1 = e_2 \longrightarrow h''; v_1\, [\![=]\!]\, v_2} \quad \text{(E-Eq)}$$

$$\frac{\rho \vdash h; e_1 \longrightarrow h'; v_1 \qquad \rho \vdash h'; e_2 \longrightarrow h''; v_2}{\rho \vdash h; e_1 \neq e_2 \longrightarrow h''; v_1 \, [\![\neq]\!] \, v_2} \quad \text{(E-NEQ)}$$

$$\frac{\rho \vdash h; e_1 \longrightarrow h'; b_1 \qquad \rho \vdash h'; e_2 \longrightarrow h''; b_2}{\rho \vdash h; e_1 \,\&\&\, e_2 \longrightarrow h''; v_1 \, [\![\&\&]\!] \, v_2} \quad \text{(E-AND)}$$

$$\frac{\rho \vdash h; e_1 \longrightarrow h'; b_1 \qquad \rho \vdash h'; e_2 \longrightarrow h''; b_2}{\rho \vdash h; e_1 \,||\, e_2 \longrightarrow h''; v_1 \, [\![||]\!] \, v_2} \quad \text{(E-OR)}$$

$$\frac{\rho \vdash h; e_1 \longrightarrow h'; \mathrm{true} \qquad \rho \vdash h'; e_2 \longrightarrow h''; v_2}{\rho \vdash h; \mathrm{if}\, e_1 \,\mathrm{then}\, e_2 \,\mathrm{else}\, e_3 \longrightarrow h''; v_2} \quad \text{(E-IF-TRUE)}$$

$$\frac{\rho \vdash h; e_1 \longrightarrow h'; \mathrm{false} \qquad \rho \vdash h'; e_3 \longrightarrow h''; v_3}{\rho \vdash h; \mathrm{if}\, e_1 \,\mathrm{then}\, e_2 \,\mathrm{else}\, e_3 \longrightarrow h''; v_3} \quad \text{(E-IF-FALSE)}$$

$$\frac{\rho \vdash h; e_1 \longrightarrow h'; v_1 \qquad \rho \cup \{x \mapsto v_1\} \vdash h'; e_2 \longrightarrow h''; v_2}{\rho \vdash h; \mathrm{let}\, x = e_1 \,\mathrm{in}\, e_2 \longrightarrow h''; v_2} \quad \text{(E-LET)}$$

$$\frac{\mu X.\rho \cup \{f \mapsto (x, e_1, X)\} \vdash h; e_2 \longrightarrow h'; v_2}{\rho \vdash h; \mathrm{let\,rec}\, f \ x = e_1 \,\mathrm{in}\, e_2 \longrightarrow h'; v_2} \quad \text{(E-LETREC)}$$

$$\frac{\rho \vdash h; e_1 \longrightarrow h'; f_b \qquad \rho \vdash h'; e_2 \longrightarrow h''; v_2}{\rho \vdash h; e_1 \ e_2 \longrightarrow \langle h'', (f_b, v_2) \rangle \, ; (f_b h'' v_2)} \quad \text{(E-APPBUILTIN)}$$

$$\frac{}{\rho \vdash h; \lambda x.\ e \longrightarrow h; (x, e, \rho)} \quad \text{(E-ABS)}$$

We write $[\![e]\!]$ for (h, v) such that $\emptyset \vdash \langle\rangle \, ; e \longrightarrow h; v$.

A *valid execution* of a YuchiKaml program expression e has the built-in function call history h and the return value v such that $(h, v) = [\![e]\!]$.

3 YuchiKaml Interpreter

3.1 Usage

3.2 Preprocess