Types and Programming Language Project Report*

Jiajun Jiang
jiajun.jiang@pku.edu.cn

Mengwei Xu
xumw@pku.edu.cn

Introduction

Here is the introduction...

1 Language Design

1.1 Jeeves

Here we discribe Jeeves syntax in Figure 1. Totally, it contains three types

$$
\begin{array}{rcl}
\mathit{Level} & ::= & \bot \mid \top \\
\mathit{Exp} & ::= & \upsilon \mid \mathit{Exp}_1\ (\mathit{op})\ \mathit{Exp}_2 \\
& \mid & \textbf{if } \mathit{Exp}_1 \textbf{ then } \mathit{Exp}_t \textbf{ else } \mathit{Exp}_f \\
& \mid & \mathit{Exp}_1\ \mathit{Exp}_2 \\
& \mid & \langle \mathit{Exp}_\bot \mid \mathit{Exp}_\top \rangle\,(\ell) \\
& \mid & \textbf{level } \ell \textbf{ in } \mathit{Exp} \\
& \mid & \textbf{policy } \ell :\ \mathit{Exp}_p \textbf{ then } \mathit{Level} \textbf{ in } \mathit{Exp} \\
\mathit{Stmt} & ::= & \textbf{let } x : \tau\ =\ \mathit{Exp} \\
& \mid & \textbf{print } \{\mathit{Exp}_c\}\ \mathit{Exp}
\end{array}
$$

Figure 1: Jeeves syntax

1.2 Lambda J

Here we discribe the λ_J language show in Figure 2.

2 Semantics

Here is the semantics...

*done at June 5th, 2016

$$
\begin{array}{rcl}
c & ::= & n \mid b \mid \lambda x : \tau.e \mid record\ \bar{x:v} \\
  & \mid & \textbf{error} \mid () \\
\sigma & ::= & x \mid \textbf{contex}\ \tau \\
  & \mid & c_1\ (op)\ \sigma_2 \mid \sigma_1\ (op)\ c_2 \\
  & \mid & \sigma_1\ (op)\ \sigma_2 \\
  & \mid & \textbf{if}\ \sigma\ \textbf{then}\ \upsilon_t\ \textbf{else}\ \upsilon_f \\
\upsilon & ::= & c \mid \sigma \\
e & ::= & \upsilon \mid e_1\ (op)\ e_2 \\
  & \mid & \textbf{if}\ e_1\ \textbf{then}\ e_t\ \textbf{else}\ e_f \mid e_1\ e_2 \\
  & \mid & \textbf{let}\ x : \tau = e_1\ \textbf{in}\ e_2 \\
  & \mid & \textbf{let rec}\ f : \tau = e_1\ \textbf{in}\ e_2 \\
  & \mid & \textbf{defer}\ x : \tau\{e\}\ \textbf{defaut}\ \upsilon_d \\
  & \mid & \textbf{assert}\ e \\
  & \mid & \textbf{concretize}\ e\ \textbf{with}\ \upsilon_c
\end{array}
$$

Figure 2: λ_{J} syntax

$$\boxed{\mathscr{G} \vdash \langle \Sigma, \Delta, e \rangle \to \langle \Sigma', \Delta', e' \rangle}$$

$$\frac{\mathscr{G}\vdash\langle\Sigma,\Delta,e_1\rangle\to\langle\Sigma',\Delta',e_1'\rangle}{\mathscr{G}\vdash\langle\Sigma,\Delta,e_1\ \ e_2\rangle\to\langle\Sigma',\Delta',e_1'\ \ e_2\rangle}\ \text{E-APP1} \qquad \frac{\mathscr{G}\vdash\langle\Sigma,\Delta,e_2\rangle\to\langle\Sigma',\Delta',e_2'\rangle}{\mathscr{G}\vdash\langle\Sigma,\Delta,\upsilon\ e_2\rangle\to\langle\Sigma',\Delta',\upsilon\ e_2\rangle}\ \text{E-APP2}$$

$$\frac{}{\mathscr{G}\vdash\langle\Sigma,\Delta,\lambda x.e\ \ \upsilon\rangle\to\langle\Sigma',\Delta',e[x\mapsto\upsilon]\rangle}\ \text{E-APPLAMBDA} \qquad \frac{c'=c_1\ (op)\ c_2}{\mathscr{G}\vdash\langle\Sigma,\Delta,c_1\ (op)\ c_2\rangle\to\langle\Sigma',\Delta',c'\rangle}\ \text{E-OP}$$

$$\frac{\mathscr{G}\vdash\langle\Sigma,\Delta,e_1\rangle\to\langle\Sigma',\Delta',e_1'\rangle}{\mathscr{G}\vdash\langle\Sigma,\Delta,e_1\ (op)\ e_2\rangle\to\langle\Sigma',\Delta',e_1'\ (op)\ e_2\rangle}\ \text{E-OP1} \qquad \frac{\mathscr{G}\vdash\langle\Sigma,\Delta,e_2\rangle\to\langle\Sigma',\Delta',e_2'\rangle}{\mathscr{G}\vdash\langle\Sigma,\Delta,\upsilon\ (op)\ e_2\rangle\to\langle\Sigma',\Delta',\upsilon\ (op)\ e_2'\rangle}\ \text{E-OP2}$$

$$\frac{\mathscr{G}\vdash\langle\Sigma,\Delta,e_c\rangle\to\langle\Sigma',\Delta',e_c'\rangle}{\mathscr{G}\vdash\langle\Sigma,\Delta,\textbf{if}\ e_c\ \textbf{then}\ e_t\ \textbf{else}\ e_f\rangle\to\langle\Sigma',\Delta',\textbf{if}\ e_c'\ \textbf{then}\ e_t\ \textbf{else}\ e_f\rangle}\ \text{E-COND}$$

$$\frac{\mathscr{G}\vdash\langle\Sigma,\Delta,e_t\rangle\to\langle\Sigma',\Delta',e_t'\rangle}{\mathscr{G}\vdash\langle\Sigma,\Delta,\textbf{if true then}\ e_t\ \textbf{else}\ e_f\rangle\to\langle\Sigma',\Delta',e_t'\rangle}\ \text{E-CONDTRUE}$$

$$\frac{\mathscr{G}\vdash\langle\Sigma,\Delta,e_f\rangle\to\langle\Sigma',\Delta',e_f'\rangle}{\mathscr{G}\vdash\langle\Sigma,\Delta,\textbf{if false then}\ e_t\ \textbf{else}\ e_f\rangle\to\langle\Sigma',\Delta',e_f'\rangle}\ \text{E-CONDFALSE}$$

$$\frac{\mathscr{G}\vdash\langle\Sigma,\Delta,e_t\rangle\to\langle\Sigma',\Delta',e_t'\rangle}{\mathscr{G}\vdash\langle\Sigma,\Delta,\textbf{if}\ \sigma\ \textbf{then}\ e_t\ \textbf{else}\ e_f\rangle\to\langle\Sigma',\Delta',\textbf{if}\ \sigma\ \textbf{then}\ e_t'\ \textbf{else}\ e_f\rangle}\ \text{E-CONDSYMT}$$

$$\frac{\mathscr{G}\vdash\langle\Sigma,\Delta,e_f\rangle\to\langle\Sigma',\Delta',e_f'\rangle}{\mathscr{G}\vdash\langle\Sigma,\Delta,\textbf{if}\ \sigma\ \textbf{then}\ \upsilon_t\ \textbf{else}\ e_f\rangle\to\langle\Sigma',\Delta',\textbf{if}\ \sigma\ \textbf{then}\ \upsilon_t\ \textbf{else}\ e_f'\rangle}\ \text{E-CONDSYMF}$$

Figure 3: Dynamic semantics for λ_{J}.

3 Properties

Lemma 1. (ConcreteFunction). *if υ is a value of type $\tau_1 \to \tau_2$, then $\upsilon = \lambda x : \tau_1.e$, where e has type τ_2.*

Proof. According to the λ_J syntax, we can get Lemma 1 immediately. □

Theorem 1. (Progress). *Suppose e is a closed, well-typed expression. Then e is either a value υ or there is some e' such that $\vdash \langle \phi, \phi, e \rangle \to \langle \Sigma', \Delta', e' \rangle$.*

Proof. we can ... □

Theorem 2. (Preservision). *If $\Gamma \vdash e : \tau\delta$ and $e \to e'$, then $\Gamma \vdash e' : \tau\delta$.*

Proof. we can ... □

4 Evaluation

Here is the example area.

```
data Exp = E_BOOL Bool | E_NAT Int
                | E_STR String | E_CONST String
                | E_VAR Var    | E_CONTEXT
                | E_LAMBDA Var Exp | E_THUNK Exp
                | E_OP Op Exp Exp   | E_UOP UOp Exp
                | E_IF Exp Exp Exp | E_APP Exp Exp
                | E_DEFER Var Exp | E_ASSERT Exp Exp
                | E_LET Var Exp Exp
                | E_RECORD [(FieldName,Exp)]
                | E_FIELD Exp FieldName
                deriving (Ord,Eq)

data Op = OP_PLUS | OP_MINUS
                | OP_LESS | OP_GREATER
                | OP_EQ | OP_AND | OP_OR | OP_IMPLY
                deriving (Ord,Eq)

data UOp = OP_NOT   deriving (Ord,Eq)

data FieldName = FIELD_NAME String  deriving(Ord,Eq)
data Var = VAR String deriving (Ord,Eq)
```

```
---------
let name =
    level a in
    policy a: !(context = alice) then bottom in < "Anonymous" | "Alice" >(a)

let msg = "Author is " + name

print {alice} msg
print {bob} msg
------
```

5 Conclusion

Here is the conclusion section...

References

[1] J. Yang, K. Yessenov, and A. Solar-Lezama. A language for automatically enforcing privacy policies. In Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '12). ACM, New York, NY, USA, 85-96.

[2] B. Pierce. Types and Programming Languages, *MIT Press*, 2002.