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

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

Figure 2: λ_{J} syntax

3 Properties

Lemma 1. (ConcreteFunction). *if v is a value of type $\tau_1 \to \tau_2$, then $v = \lambda x : \tau_1.e$, 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 v or there is some e' such that $\vdash \langle \Sigma, \Delta, e \rangle \to \langle \Sigma', \Delta', e' \rangle$.*

Proof. According to the dynamic demantics 3 and static semantics 5 of λ_{J}, we will proof that any λ_{J} program holds the *progress* property by structral induction over the syntax of λ_{J}. According to Figure 2, there are nine kinds of expressions including the value expression v. Next we discuss each of them respectively.

- For the value expression v, according to the definition we can conclude that it holds the property immediately.

- For the expression e_1 (op) e_2, according to the hyposthesis, we know that sub-expression e_1 and e_2 are both closed and well-typed expressions. Thus if e_1 is not a value, according to E-OP1, there must be a expression e_1' such that $e_1 \to e_1'$. Therefore the expression e_1 (op) e_2 can be reduced to e_1' (op) e_2, and the same as if the e_1 is already a value but e_2 is not according to E-OP2. While if both e_1 and e_2 are values, we know that e_1 (op) e_2 is a value as well. Let $c = e_1\ (op)\ e_2$, according to E-OP, $\vdash \langle \Sigma, \Delta, e_1\ (op)\ e_2 \rangle \to \langle \Sigma', \Delta', c \rangle$. Hence, we can conclude that expression e_1 (op) e_2 holds the *progress* property.

- For the expression **if** e_1 **then** e_t **else** e_f, the conditional expression e_1 is a either a **concrete** expression or a **symbolic** expression. If expression e_1 is **concrete** and not a value, according to E-COND, there must be an expression e_1' such

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

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

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

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

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

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

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

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

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

$$\frac{\mathscr{G}\vdash\langle\Sigma,\Delta,e\rangle\to\langle\Sigma',\Delta',e'\rangle}{\mathscr{G}\vdash\langle\Sigma,\Delta,\textbf{defer}\ x{:}\tau\{e\}\ \textbf{default}\ \upsilon_d\rangle\to\langle\Sigma',\Delta',\textbf{defer}\ x{:}\tau\{e'\}\ \textbf{default}\ \upsilon_d\rangle}\ \text{E-DEFERCONSTRAINT}$$

$$\frac{\mathit{fresh}\ x'}{\mathscr{G}\vdash\langle\Sigma,\Delta,\textbf{defer}\ x{:}\tau\{e\}\ \textbf{default}\ \upsilon_d\rangle\to\langle\Sigma\cup\{\mathscr{G}\Rightarrow\upsilon_c[x\mapsto x']\},\Delta\cup\{\mathscr{G}\Rightarrow x'=\upsilon_d\},x'\rangle}\ \text{E-DEFER}$$

$$\frac{\mathscr{G}\vdash\langle\Sigma,\Delta,e\rangle\to\langle\Sigma',\Delta',e'\rangle}{\mathscr{G}\vdash\langle\Sigma,\Delta,\textbf{assert}\ e\rangle\to\langle\Sigma',\Delta',\textbf{assert}\ e'\rangle}\ \text{E-ASSERTCONSTRAINT}$$

$$\frac{}{\mathscr{G}\vdash\langle\Sigma,\Delta,\textbf{assert}\ \upsilon\rangle\to\langle\Sigma\cup\{\mathscr{G}\Rightarrow\upsilon\},\Delta,()\rangle}\ \text{E-ASSERT}$$

$$\frac{\mathscr{G}\vdash\langle\Sigma,\Delta,e\rangle\to\langle\Sigma',\Delta',e'\rangle}{\mathscr{G}\vdash\langle\Sigma,\Delta,\textbf{concretize}\ e\ \textbf{with}\ \upsilon_c\rangle\to\langle\Sigma',\Delta',\textbf{concretize}\ e'\ \textbf{with}\ \upsilon_c\rangle}\ \text{E-CONCRETIZEEXP}$$

$$\frac{\text{MODLE}(\Delta,\Sigma\cup\{\mathscr{G}\cap\textbf{context}=\upsilon_c\})=\mathscr{M}\qquad c=\mathscr{M}[[\upsilon_\nu]]}{\mathscr{G}\vdash\langle\Sigma,\Delta,\textbf{concretize}\ \upsilon_\nu\ \textbf{with}\ \upsilon_c\rangle\to\langle\Sigma,\Delta,c\rangle}\ \text{E-CONCRETIZESAT}$$

$$\frac{\text{MODLE}(\Delta,\Sigma\cup\{\mathscr{G}\cap\textbf{context}=\upsilon_c\})=\text{UNSAT}}{\mathscr{G}\vdash\langle\Sigma,\Delta,\textbf{concretize}\ \upsilon_\nu\ \textbf{with}\ \upsilon_c\rangle\to\langle\Sigma,\Delta,\textbf{error}\rangle}\ \text{E-CONCRETIZEUNSAT}$$

Figure 3: Dynamic semantics for λ_{J}.

$$\begin{array}{rcll}
\delta & ::= & \textbf{concretize} \mid \textbf{sym} & \text{determinism tag} \\
\beta & ::= & \textbf{int}_c \mid \textbf{bool}_c \mid \textbf{unit} \mid \textbf{int} \mid \textbf{bool} & \text{base type} \\
\tau & ::= & \beta \mid \tau_1 \xrightarrow{nr} \tau_2 \mid \tau_1 \to \tau_2 & \text{type}
\end{array}$$

Figure 4: λ_{J} types

$$\boxed{\tau_1 <: \tau_2}$$

$$\frac{}{\tau{<:}\tau}\ \text{S-REFLEXIVE} \qquad \frac{}{\mathbf{int}_c{<:}\mathbf{int}}\ \text{S-INT} \qquad \frac{}{\mathbf{bool}_c{<:}\mathbf{bool}}\ \text{S-BOOL}$$

$$\frac{}{\tau_1 \xrightarrow{nr} \tau_2 {<:} \tau_1 \rightarrow \tau_2}\ \text{S-RECFUN} \qquad \frac{\tau_1'{<:}\tau_1 \quad \tau_2{<:}\tau_2'}{\tau_1\rightarrow\tau_2{<:}\tau_1'\rightarrow\tau_2'}\ \text{S-FUN}$$

$$\boxed{\mathbf{rep}\ \tau}$$

$$\frac{\mathbf{rep}\ \tau \quad \tau'{<:}\tau}{\mathbf{rep}\ \tau}\ \text{OK-SUBTYPE} \qquad \frac{}{\mathbf{rep}\ \beta}\ \text{OK-BASETYPE}$$

$$\frac{\mathbf{rep}\ \tau_2}{\mathbf{rep}\ \beta_1\rightarrow\tau_2}\ \text{OK-BASEFUNCTION} \qquad \frac{\mathbf{rep}\ (\tau_1\xrightarrow{nr}\tau') \quad \mathbf{rep}\ \tau_2}{\mathbf{rep}\ (\tau_1\xrightarrow{nr}\tau')\xrightarrow{nr}\tau_2}\ \text{OK-HOFUNCTION}$$

$$\frac{\mathbf{rep}\ \tau_1\rightarrow\tau_2}{\mathbf{rep}\ (\tau_1\rightarrow\tau_2)\rightarrow\beta}\ \text{OK-RECFUNCTIONBASE} \qquad \frac{\mathbf{rep}\ \tau_1\rightarrow\tau_2 \quad \mathbf{rep}\ \tau_1'\rightarrow\tau_2'}{\mathbf{rep}\ (\tau_1\rightarrow\tau_2)\rightarrow(\tau_1'\rightarrow\tau_2')}\ \text{OK-RECFUNCTION}$$

$$\boxed{\Gamma; \gamma \vdash e : \langle \tau, \delta \rangle}$$

$$\frac{x{\in}\Gamma}{\Gamma;\gamma{\vdash}x{:}\Gamma(x)}\ \text{T-VAR} \qquad \frac{}{\Gamma;\gamma{\vdash}n{:}\mathbf{int}_c}\ \text{T-INT} \qquad \frac{}{\Gamma;\gamma{\vdash}b{:}\mathbf{bool}_c}\ \text{T-BOOL} \qquad \frac{}{\Gamma;\gamma{\vdash}(){:}\mathbf{unit}}\ \text{T-UNIT}$$

$$\frac{\mathbf{rep}\ \tau}{\Gamma;\gamma{\vdash}\mathbf{context}\ \tau{:}\tau}\ \text{T-CONTEXT} \qquad \frac{\Gamma;\gamma{\vdash}e_1{:}\tau_1 \quad \Gamma;\gamma{\vdash}e_2{:}\tau_2 \quad \tau_1,\tau_2{<:}\tau \quad \mathbf{rep}\ \tau}{\Gamma;\gamma{\vdash}e_1\ (op)\ e_2{:}\tau}\ \text{T-OP}$$

$$\frac{\Gamma;\gamma{\vdash}e{:}\mathbf{bool}_c \quad \Gamma;\gamma{\vdash}e_t{:}\tau_1 \quad \Gamma;\gamma{\vdash}e_f{:}\tau_2 \quad \tau_1,\tau_2{<:}\tau \quad \mathbf{rep}\ \tau}{\Gamma;\gamma{\vdash}\mathbf{if}\ e\ \mathbf{then}\ e_t\ \mathbf{else}\ e_f{:}\tau}\ \text{T-CONDC}$$

$$\frac{\Gamma;\gamma{\vdash}e{:}\mathbf{bool} \quad \Gamma;\mathbf{sym}{\vdash}e_t{:}\beta_1 \quad \Gamma;\mathbf{sym}{\vdash}e_f{:}\beta_2 \quad \beta_1,\beta_2{<:}\beta_c \quad \mathbf{rep}\ \beta_c}{\Gamma;\gamma{\vdash}\mathbf{if}\ e\ \mathbf{then}\ e_t\ \mathbf{else}\ e_f{:}\beta_c}\ \text{T-CONDSYM}$$

$$\frac{\Gamma,x{:}\tau_d;\gamma{\vdash}e{:}\tau' \quad \mathbf{rep}\ \tau_d \quad \mathbf{rep}\ \tau'}{\Gamma;\gamma{\vdash}(\lambda x{:}\tau_d.e){:}\tau_d\rightarrow\tau'}\ \text{T-LAMBDA} \qquad \frac{\Gamma;\gamma{\vdash}e_1{:}\tau_1\xrightarrow{nr}\tau_2 \quad \Gamma,x{:}\tau_d;\gamma{\vdash}e_2{:}\tau_1' \quad \tau_1'{<:}\tau_1 \quad \mathbf{rep}\ \tau_1 \quad \mathbf{rep}\ \tau_2}{\Gamma;\gamma{\vdash}(e_1\ e_2){:}\tau_2}\ \text{T-APP}$$

$$\frac{\Gamma,f{:}\tau_1\rightarrow\tau_2;\gamma{\vdash}e_1{:}\tau_1\rightarrow\tau_2 \quad \Gamma,f{:}\tau_1\xrightarrow{nr}\tau_2;\gamma{\vdash}e_2{:}\tau_2 \quad \mathbf{rep}\ \tau \quad \mathbf{rep}\ \tau_2}{\Gamma;\gamma{\vdash}\mathbf{let\ rec}\ f{:}\tau_1\xrightarrow{nr}\tau_2{=}e_1\ \mathbf{in}\ e_2{:}\tau_2}\ \text{T-APPLETREC}$$

$$\frac{\gamma{=}\mathbf{concrete} \quad \Gamma;\gamma{\vdash}e_1{:}\tau_1\rightarrow\tau_2 \quad \Gamma;\gamma{\vdash}e_2{:}\tau_1' \quad \tau_1'{<:}\tau_1 \quad \mathbf{rep}\ \tau_1 \quad \mathbf{rep}\ \tau_2}{\Gamma;\gamma{\vdash}(e_1\ e_2){:}\tau_2}\ \text{T-APPCURREC}$$

$$\frac{\Gamma,x{:}\beta;\gamma{\vdash}e_c{:}\mathbf{bool} \quad \Gamma;\gamma{\vdash}v{:}\beta}{\Gamma;\gamma{\vdash}(\mathbf{defer}x{:}\beta\{e_c\}\ \mathbf{default}\ v){:}\beta}\ \text{T-DEFER} \qquad \frac{\Gamma;\gamma{\vdash}e_c{:}\mathbf{bool}}{\Gamma;\gamma{\vdash}(\mathbf{assert}\ e_c){:}\mathbf{unit}}\ \text{T-ASSERT}$$

$$\frac{\Gamma;\gamma{\vdash}e_1{:}\beta \quad \Gamma;\gamma{\vdash}^c e_1{:}\beta' \quad \Gamma;\gamma{\vdash}v{:}\beta'}{\Gamma;\gamma{\vdash}(\mathbf{concretize}\ e_1\ \mathbf{with}\ v){:}\beta_c}\ \text{T-CONCRETIZE}$$

Figure 5: Static semantics for λ_J describing simple type-checking and enforce restriction on scope of nondeterminism and recursion. Recall that β refers to base (non-function) types.

that $\vdash \langle \Sigma, \Delta, \textbf{if } e_1 \textbf{ then } e_t \textbf{ else } e_f \rangle \rightarrow \langle \Sigma', \Delta', \textbf{if } e_1' \textbf{ then } e_t \textbf{ else } e_f \rangle$. Similarly, if expression e_1 is a **conceret** value, it must be **true** or **false**. According to E-CONDTRUE and E-CONDFALSE, **if** e_1 **then** e_t **else** e_f can be reduced as well. On the other hand, if the expression e_1 is **symbolic**, according to E-CONDSYMT and E-CONDSYMF, it holds the *progress* property.

- For the expression $e_1\ e_2$, if the sub-expression e_1 or e_2 is not a value, similar as the expression $e_1\ (op)\ e_2$, according to E-APP1 and E-APP2, it can be at least reduce one step and evetually both e_1 and e_2 are values. Then according to the previous **Concrete Function Lemma**, $e_1\ e_2$ must be the form as $\lambda x : e.v$, according to E-APPLAMBDA, it holds the property immediately.
- For the expression **defer** $x : \tau\{e\}$ **default** v_d, if sub-expression e is not a value, by the induction hypothesis, there must be an expression e' such that $\vdash \langle \Sigma, \Delta, e \rangle \rightarrow \langle \Sigma', \Delta', e' \rangle$, then by E-DEFERCONSTRAINT, the expression can be perform a reduction $\vdash \langle \Sigma, \Delta, \textbf{defer } x : \tau\{e\} \textbf{ default } v_d \rangle \rightarrow \langle \Sigma', \Delta', \textbf{defer } x : \tau\{e'\} \textbf{ default } v_d \rangle$
- For the expression **assert** e,
- For the expression **concretize** e **with** v_c,
- For the expression **let** $x : \tau = e_1$ **in** e_2 and **let rec** $f : \tau = e_1$ **in** e_2, expressions e_1 and e_2 are composed by one ore more above expressions. Thus, it holds the property accordingly.

□

Theorem 2. (Preservion). *If* $\Gamma \vdash e : \tau\delta$ *and* $e \rightarrow e'$, *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.