Types and Programming Language Project Report

Jiajun Jiang Mengwei Xu

2016-06-5

Introduction

Here is the introduction...

1 Language Design

1.1 Jeeves

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

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

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...

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.*

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

Figure 2: λ_J syntax

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. (Preservion). *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.