

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 describe Jeeves syntax in Figure 1. Totally, it contains three types

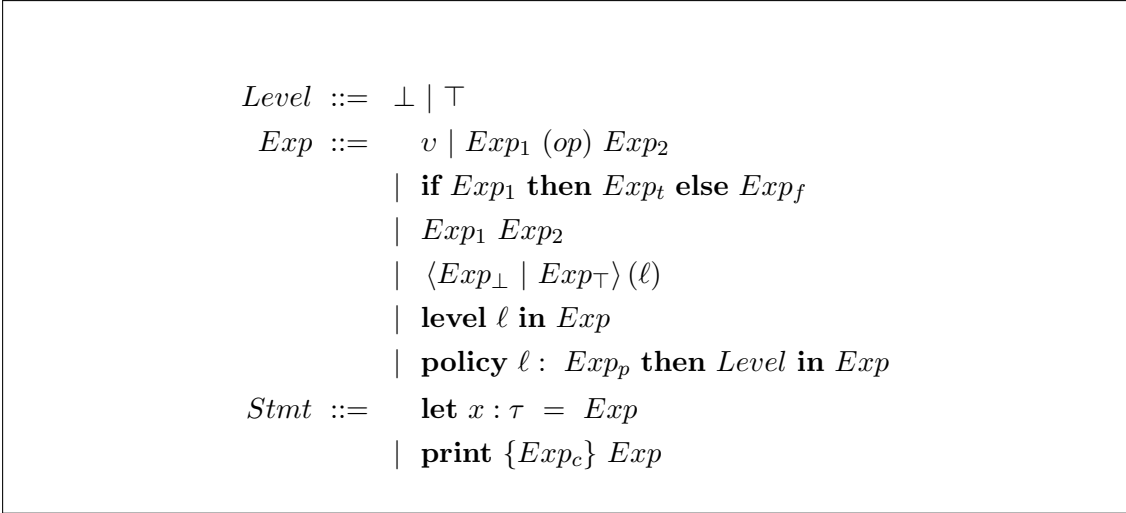

$$\begin{aligned} Level &::= \perp \mid \top \\ Exp &::= v \mid Exp_1 (op) Exp_2 \\ &\quad \mid \text{if } Exp_1 \text{ then } Exp_t \text{ else } Exp_f \\ &\quad \mid Exp_1 Exp_2 \\ &\quad \mid \langle Exp_\perp \mid Exp_\top \rangle (\ell) \\ &\quad \mid \text{level } \ell \text{ in } Exp \\ &\quad \mid \text{policy } \ell : Exp_p \text{ then } Level \text{ in } Exp \\ Stmt &::= \text{let } x : \tau = Exp \\ &\quad \mid \text{print } \{Exp_c\} Exp \end{aligned}$$

Figure 1: Jeeves syntax

1.2 Lambda J

Here we describe the λ_J language shown in Figure 2.

2 Semantics

Here is the semantics...

*done at June 5th, 2016

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

Figure 2: λ_J syntax

3 Properties

Lemma 1. (ConcreteFunction). *if v is a value of type $\tau_1 \rightarrow \tau_2$, 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. \square

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 $\vdash \langle \phi, \phi, e \rangle \rightarrow \langle \Sigma', \Delta', e' \rangle$.*

Proof. we can ... \square

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

Proof. we can ... \square

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.