1 GhostLambda: Lambda-Calculus with Ghost Code

Annotating a program with logical formulae predicting statically it dynamic behaviour is a keystone of the deductive software verification approach.

Typically, these logical formulae are assertions about program assigned variables, loop invariants, recursive function variants, etc.

However, it is often useful to annotate programs with some data that appear in program that appear in assertions but do not affect the program dynamic behaviour in any way.

When a correct-by-construction executable code is extracted from the specified program, this , called *ghost code* disappear together with specification.

In this section we describe a language where one can annotate programs with such *ghost code*. We start by formalizing *ghostλ-calculus*, a tiny language of simply typed λ-calculus enriched with ghost variables and ghost expressions. We then define ghost code *erasure*, which transforms a well-typed ghostLambda term to a term of standard λ-calculus. Finally we state and proof a few basic preservation properties of such translation.

1.1 Syntax and Semantics

The syntax and small-step operational semantics of *ghost-λ* is summarized in Figure 1.

Syntax

$\mathtt{t}$	$::=$		TERMS:
		$()$	*unit*
		$\mathtt{x}_\tau^{\mathfrak{B}}$	*variable*
		$\lambda \mathtt{x}_\tau^{\mathfrak{B}}.\mathtt{t}$	*abstraction*
		$\mathtt{t}\,\mathtt{t}$	*application*
		$\mathtt{ghost\ t}$	*ghost term*
$\mathtt{v}$	$::=$		VALUES:
		$()$	*unit*
		$\lambda \mathtt{x}_\tau^{\mathfrak{B}}.\mathtt{t}$	*abstraction*
$\mathfrak{B}$	$::=$		GHOST STATUS :
		$\top \ (*true*)$	*ghost code*
		$\bot \ (*false*)$	*physical code*
τ	$::=$		TYPES:
		$\mathtt{unit}$	*unit type*
		$\tau^{\mathfrak{B}} \to \tau$	*function type*

Evaluation

$$(\lambda x_\tau^b.\mathtt{t})v \xrightarrow{\epsilon} \mathtt{t}[x_\tau^b \leftarrow v] \qquad \text{(E-AppFun)}$$

$$\mathtt{ghost\ t} \xrightarrow{\epsilon} \mathtt{t} \qquad \text{(E-DeGhost)}$$

$$\frac{\mathtt{t} \to \mathtt{t}'}{\mathtt{E\ t} \to \mathtt{E\ t}'} \qquad \text{(E-Context)}$$

where $\mathtt{E} ::= \square \mid \mathtt{E\ t} \mid \mathtt{v\ E}$

Figure 1: ***ghost-λ* syntax and semantics**

1.2 Typing

The typing relation of *ghost-λ* is summarized in Figure 2.

$$\frac{}{\vdash_{g\lambda} () : (unit, \bot)} \quad \text{(T-Unit)}$$

$$\frac{}{\vdash_{g\lambda} x_{\tau}^{\mathfrak{B}} : (\tau, \mathfrak{B})} \quad \text{(T-Var)}$$

$$\frac{\vdash_{g\lambda} \mathtt{t} : (\tau, \mathfrak{B})}{\vdash_{g\lambda} \mathtt{ghost\ t} : (\tau, \top)} \quad \text{(T-Ghost)}$$

$$\frac{\vdash_{g\lambda} \mathtt{t} : (\tau_2, \mathfrak{B}_2)}{\vdash_{g\lambda} \lambda x_{\tau_1}^{\mathfrak{B}_1}.\mathtt{t} : (\tau_1^{\mathfrak{B}_1} \to \tau_2, \mathfrak{B}_2)} \quad \text{(T-Abs)}$$

$$\frac{\vdash_{g\lambda} \mathtt{t}_1 : (\tau_2^{\mathfrak{B}_2} \to \tau_1, \mathfrak{B}_1) \qquad \vdash_{g\lambda} \mathtt{t}_2 : (\tau_2, \mathfrak{B}'_2) \qquad \mathfrak{B}_2 \Rightarrow \mathfrak{B}'_2}{\vdash_{g\lambda} \mathtt{t}_1\ \mathtt{t}_2 : (\tau_1, \mathfrak{B}_1 \vee (\neg\mathfrak{B}_2 \wedge \mathfrak{B}'_2))} \quad \text{(T-App)}$$

Figure 2: ***ghost-λ*** typing relation

1.3 Ghost Code Erasure

let $\mathtt{t}$ be a term such that $\vdash_{g\lambda} \mathtt{t} : (\tau, \mathfrak{B})$ holds. Then define $\mathcal{E}$ that translate gλ-terms and types to λ-calculus as follows.
The erasure of types is defined by :

$\mathcal{E}(\tau_1^{\top} \to \tau_2) = \mathtt{unit} \to \tau_2$
$\mathcal{E}(\tau) = \tau$ otherwise

For the erasure of terms, if $\mathfrak{B}$ is equal to $\top$ then $\mathcal{E}(\mathtt{t}) = ()$. Otherwise :

$\mathcal{E}(()) = ()$
$\mathcal{E}(x_{\tau}^{\bot}) = x_{\mathcal{E}(\tau)}$
$\mathcal{E}(\lambda x_{\tau}^{\top}.t) = \lambda x_{\mathtt{unit}}.\mathcal{E}(t)$
$\mathcal{E}(\lambda x_{\tau}^{\bot}.t) = \lambda x_{\mathcal{E}(\tau)}.\mathcal{E}(t)$
$\mathcal{E}(\mathtt{t}_1\ \mathtt{t}_2) = \mathcal{E}(\mathtt{t}_1)\ \mathcal{E}(\mathtt{t}_2)$

1.4 Properties of Ghost Code Erasure

1.4.1 Typing Erasure

Lemma 1.1 [Translation of Typing Relation]. *If* $\vdash_{g\lambda} t : (\tau, \bot)$ *then* $\vdash_{\lambda} \mathcal{E}(t) : \mathcal{E}(\tau)$.

Proof. By induction on a derivation of the statement $\vdash_{g\lambda} \mathcal{E}(\mathtt{t}) : \mathcal{E}(\tau)$. For a given derivation, we proceed by case analysis on the final typing rule used in the proof.

Case T-Unit: $\vdash_{g\lambda} () : (\mathtt{unit}, \bot)$
As $\mathcal{E}(()) = ()$ and $\mathcal{E}(\mathtt{unit}) = \mathtt{unit}$ we have immediately $\vdash_{\lambda} () : \mathtt{unit}$.

Case T-Var: $\vdash_{g\lambda} x_{\tau}^{\bot} : (\tau, \bot)$
As $\mathcal{E}(x_{\tau}^{\bot}) = x_{\mathcal{E}(\tau)}$, we have immediately $\vdash_{\lambda} x_{\mathcal{E}(\tau)} : \mathcal{E}(\tau)$.

Case T-ABS: $\vdash_{g\lambda} \lambda x^{\mathfrak{B}_1}_{\tau_1}.\mathtt{t} : (\tau_1^{\mathfrak{B}_1} \to \tau_2, \bot)$ with $\vdash_{g\lambda} \mathtt{t} : (\tau_2, \bot)$
Case T-APP: □

1.4.2 Evaluation Preservation