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*
		$\mathtt{x}^{\mathcal{B}}_{\tau}$	*variable*
		$\lambda \mathtt{x}^{\mathcal{B}}_{\tau}.\mathtt{t}$	*abstraction*
		$\mathtt{t}\,\mathtt{t}$	*application*
		$\mathtt{ghost\ t}$	*ghost term*
$\mathtt{v}$	$::=$		VALUES :
		$()$	*unit*
		$\lambda \mathtt{x}^{\mathcal{B}}_{\tau}.\mathtt{t}$	*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

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

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

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

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.

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

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

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

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

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

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

1.3 Ghost Code Erasure

1.4 Properties of Ghost Code Erasure

1.4.1 Typing Erasure

1.4.2 Evaluation Preservation