# 1 GhostLambda: Lambda-Calculus with Ghost Code

## 1.1 g$\lambda$-Calculus Syntax and Semantics

| | | | | | |
|---|---|---|---|---|---|
| $p$ | $::=$ | PROGRAMS | $c$ | $::=$ | CONSTANTS |
| | $\text{var } r^{\mathfrak{B}}_{\texttt{ref } \tau} = v; \quad p$ | *reference declaration* | | $0 \mid 1 \mid ... \mid n$ | *integer* |
| | $t$ | *body* | | $\texttt{true} \mid \texttt{false}$ | *boolean* |
| | | | | $()$ | *unit* |
| $t$ | $::=$ | TERMS | | $op$ | *built-in operators* |
| | $c$ | *constant* | | | |
| | $x^{\mathfrak{B}}_{\tau}$ | *variable* | $op$ | $:=$ | BUILT-IN OPERATORS |
| | $\lambda x^{\mathfrak{B}}_{\tau}.t$ | *abstraction* | | $+ \mid - \mid = \mid \texttt{not}$ | *built-in operators* |
| | $v\ v$ | *application* | | | |
| | $\texttt{ghost } t$ | *ghost term* | $\tau$ | $::=$ | TYPES |
| | $\texttt{let } x^{\mathfrak{B}}_{\tau} = t \texttt{ in } t$ | *local binding* | | $\texttt{int} \mid \texttt{bool} \mid \texttt{unit}$ | *built-in types* |
| | $\texttt{if } v \texttt{ then } t \texttt{ else } t$ | *if-branching* | | $\tau^{\mathfrak{B}} \to \tau$ | *function type* |
| | $\texttt{rec } f^{\mathfrak{B}}\ x^{\mathfrak{B}}_{\tau} : \tau.\ t$ | *recursive function* | | | |
| | $!r^{\mathfrak{B}}_{\texttt{ref } \tau}$ | *reference access* | $\texttt{ref } \tau$ | | REFERENCE'S TYPE |
| | $r^{\mathfrak{B}_{\mathfrak{B}}}_{\texttt{ref } \tau_\tau} := v$ | *reference assignment* | | | |
| | | | $\mathfrak{B}$ | $::=$ | GHOST INDICATOR |
| $v$ | $::=$ | VALUES | | $\bot(*\texttt{false}*)$ | *raw code* |
| | $c$ | *constant* | | $\top(*\texttt{true}*)$ | *ghost code* |
| | $x^{\mathfrak{B}}_{\tau}$ | *variable* | | | |
| | $\lambda x^{\mathfrak{B}}_{\tau}.t$ | *abstraction* | | | |
| | $\texttt{rec } f^{\mathfrak{B}}\ x^{\mathfrak{B}}_{\tau} : \tau.\ t$ | *recursive function* | | | |

Figure 1: ***ghost*-ml Syntax**

$\delta(+, n, m) \triangleq ||n + m||$ (where $n, m$ are integers)
$\delta(-, n, m) \triangleq ||n - m||$ (idem)
$\delta(\texttt{not}, b) \triangleq ||\neg b||$ (where b is a boolean)
$\delta(=, t, u) \triangleq ||t =_\tau u||$ (where $t$ and $u$ are both of the same type $\tau$)

Figure 2: ***ghost*-ml Semantics (Delta Rules)**

$$op\ v_1 \dots v_k \xrightarrow{\epsilon}_{|\mu} \delta(c, v_1 \dots v_k)_{|\mu} \quad \text{if } k = Arity(op) \text{ and } \delta(c, v) \text{ is defined} \quad \text{(E-Op)}$$

$$op\ v_1 \dots v_{k|\mu} \xrightarrow{\epsilon} \lambda x_\tau^{\mathfrak{B}}.op\ v_1 \dots v_k\ x_\tau^{\mathfrak{B}} \quad \text{if } 1 \leq k < arity(op) \quad \text{(E-Clo)}$$

$$(\lambda x_\tau^{\mathfrak{B}}.t)v_{|\mu} \xrightarrow{\epsilon} t[x_\tau^{\mathfrak{B}} \leftarrow\!\shortmid v]_{|\mu} \quad \text{(E-AppFun)}$$

$$(\texttt{rec}\ f^{\mathfrak{B}_1}_{\tau_2^{\mathfrak{B}} \to \tau_1}\ x^{\mathfrak{B}_2}_{\tau_2}.t)v_{|\mu} \xrightarrow{\epsilon} t[x^{\mathfrak{B}_2}_{\tau_2} \leftarrow\!\shortmid v, f^{1}_{\tau_2^{\mathfrak{B}} \to \tau_1} \leftarrow\!\shortmid \texttt{rec}\ f^{\mathfrak{B}_1}_{\tau_2^{\mathfrak{B}} \to \tau_1}\ x^{\mathfrak{B}_2}_{\tau_2}.t]_{|\mu} \quad \text{(E-AppRec)}$$

$$\texttt{let}\ x_\tau^{\mathfrak{B}} = v_1\ \texttt{in}\ t_{2|\mu} \xrightarrow{\epsilon} t_2[x_\tau^{\mathfrak{B}} \leftarrow\!\shortmid v_1]_{|\mu} \quad \text{(E-LetV)}$$

$$\texttt{if true then}\ t_1\ \texttt{else}\ t_{2|\mu} \xrightarrow{\epsilon} t_{1|\mu} \quad \text{(E-If-true)}$$

$$\texttt{if false then}\ t_1\ \texttt{else}\ t_{2|\mu} \xrightarrow{\epsilon} t_{2|\mu} \quad \text{(E-If-false)}$$

$$\text{ghost}\ t_{|\mu} \xrightarrow{\epsilon} t_{|\mu} \quad \text{(E-DeGhost)}$$

$$!r^{\mathfrak{B}}_{\texttt{ref}\ \tau|\mu} \xrightarrow{\epsilon} \mu(r^{\mathfrak{B}}_{\texttt{ref}\ \tau}) \quad \text{(E-Deref)}$$

$$r^{\mathfrak{B}}_{\texttt{ref}\ \tau} := v_{|\mu} \xrightarrow{\epsilon} ()_{|\mu[r^{\mathfrak{B}}_{\texttt{ref}\ \tau} \leftarrow\!\shortmid v]} \quad \text{(E-Assign)}$$

Figure 3: ***ghost*-ml Semantics (Head Reduction Rules)**

$$\frac{t_{|\mu} \xrightarrow{\epsilon}_{g\lambda} t'_{|\mu'}}{t_{|\mu} \rightarrow_{g\lambda} t'_{|\mu'}} \quad \text{(E-Head)}$$

$$\frac{t_{1|\mu} \rightarrow t'_{1|\mu'}}{\texttt{let } x_\tau^{\mathfrak{B}} = t_1 \texttt{ in } t_{2|\mu} \rightarrow \texttt{let } x_\tau^{\mathfrak{B}} = t'_1 \texttt{ in } t_{2|\mu'}} \quad \text{(E-Context)}$$

Figure 4: ***ghost*-ml Semantics Context Rules**

## 1.2 Typing Relation

$$\frac{\text{Typeof}(c) = \tau}{\vdash_{g\lambda} c : (\tau, \bot, \bot)}\ \text{(T-Const)} \qquad \frac{}{\vdash_{g\lambda} x^{\mathfrak{B}}_{\tau} : (\tau, \mathfrak{B}, \bot)}\ \text{(T-Var)}$$

$$\frac{\vdash_{g\lambda} v : (\mathfrak{B}, \tau, \bot)}{\vdash_{g\lambda} \text{var } r^{\mathfrak{B}}_{\texttt{ref } \tau} = v : (\text{ref } \tau, \mathfrak{B}, \bot)}\ \text{(T-Decl)}$$

$$\frac{\vdash_{g\lambda} t_1 : (\tau_1, \mathfrak{B}_1, \Sigma_1) \quad \overline{\mathfrak{B}_1 \wedge \Sigma_1}}{\vdash_{g\lambda} \lambda x^{\mathfrak{B}_2}_{\tau_2}.t_1 : (\tau_2^{\mathfrak{B}_2} \xrightarrow{\Sigma_1} \tau_1, \mathfrak{B}_1, \bot)}\ \text{(T-Abs)}$$

$$\frac{\vdash_{g\lambda} t_1 : (\tau_2^{\mathfrak{B}_2} \xrightarrow{\Sigma_0} \tau_1, \mathfrak{B}_1, \Sigma_1) \quad \vdash_{g\lambda} t_2 : (\tau_2, \mathfrak{B}'_2, \Sigma_2) \quad \overline{(\mathfrak{B}_1 \vee (\overline{\mathfrak{B}_2} \wedge \mathfrak{B}'_2)) \wedge (\Sigma_0 \vee \Sigma_1 \vee \Sigma_2)} \quad \overline{\mathfrak{B}'_2 \wedge \Sigma_2} \quad \mathfrak{B}_2 \Rightarrow \mathfrak{B}'_2}{\vdash_{g\lambda} t_1\ t_2 : (\tau_1, \mathfrak{B}_1 \vee (\overline{\mathfrak{B}_2} \wedge \mathfrak{B}'_2), \Sigma_0 \vee \Sigma_1 \vee \Sigma_2)}\ \text{(T-App)}$$

$$\frac{\vdash_{g\lambda} t_1 : (\text{bool}, \mathfrak{B}_1, \Sigma_1) \quad \vdash_{g\lambda} t_2 : (\tau_0, \mathfrak{B}_2, \Sigma_2) \quad \vdash_{g\lambda} t_3 : (\tau_0, \mathfrak{B}_3, \Sigma_3) \quad \overline{(\mathfrak{B}_1 \vee \mathfrak{B}_2 \vee \mathfrak{B}_3) \wedge (\Sigma_1 \vee \Sigma_2 \vee \Sigma_3)}}{\vdash_{g\lambda} \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : (\tau_0, \mathfrak{B}_1 \vee \mathfrak{B}_2 \vee \mathfrak{B}_3, \Sigma_1 \vee \Sigma_2 \vee \Sigma_3)}\ \text{(T-If)}$$

$$\frac{\vdash_{g\lambda} t_1 : (\tau_1, \mathfrak{B}_1, \Sigma_1) \quad \overline{\mathfrak{B}_1 \wedge \Sigma_1}}{\vdash_{g\lambda} \texttt{rec } f^{\mathfrak{B}_1}\ x^{\mathfrak{B}_2}_{\tau_2} : \tau_1.\ t : (\tau_2^{\mathfrak{B}_2} \xrightarrow{\Sigma_1} \tau_1, \mathfrak{B}_1, \bot)}\ \text{(T-Rec)}$$

$$\frac{\vdash_{g\lambda} t_1 : (\tau_1, \mathfrak{B}_1, \Sigma_1) \quad \vdash_{g\lambda} t_2 : (\tau_2, \mathfrak{B}'_2, \Sigma_2) \quad \overline{\mathfrak{B}'_2 \wedge \Sigma_2} \quad \overline{(\mathfrak{B}_1 \vee (\overline{\mathfrak{B}_2} \wedge \mathfrak{B}'_2)) \wedge (\Sigma_1 \vee \Sigma_2)} \quad \mathfrak{B}_2 \Rightarrow \mathfrak{B}'_2}{\vdash_{g\lambda} \text{let } x^{\mathfrak{B}_2}_{\tau_2} = t_2 \text{ in } t_1 : (\tau_1, \mathfrak{B}_1 \vee (\overline{\mathfrak{B}_2} \wedge \mathfrak{B}'_2), \Sigma_1 \vee \Sigma_2)}\ \text{(T-Let)}$$

$$\frac{\vdash_{g\lambda} t : (\tau, \mathfrak{B}, \bot)}{\vdash_{g\lambda} \texttt{ghost } t : (\tau, \top, \bot)}\ \text{(T-Ghost)} \qquad \frac{}{\vdash_{g\lambda} !r^{\mathfrak{B}}_{\texttt{ref } \tau} : (\tau, \mathfrak{B}, \bot)}\ \text{(T-Deref)}$$

$$\frac{\vdash_{g\lambda} t_2 : (\tau_2, \mathfrak{B}_2, \Sigma_2) \quad \overline{(\mathfrak{B}_1 \vee \mathfrak{B}_2) \wedge \Sigma_2} \quad \mathfrak{B}_2 \Rightarrow \mathfrak{B}_1}{\vdash_{g\lambda} r^{\mathfrak{B}_1}_{\texttt{ref } \tau_2} := t_2 : (\text{unit}, \mathfrak{B}_1, \overline{\mathfrak{B}_1} \vee \Sigma_2)}\ \text{(T-Assign)}$$

Figure 5: ***ghost*-ml Typing Rules with Effects**

### Invariant of Typing Relation

The following lemma states that in well typed terms the ghost code does not write in any not ghost global references.

*Lemma 1.1* [Invariant of Typing Relation].
*If* $\vdash_{g\lambda} t : (\tau, \mathfrak{B}, \Sigma)$ *is a well typed term, then for any sub-term* $t_1$ *of* $t$*, such that* $\vdash_{g\lambda} t_1 : (\tau_1, \mathfrak{B}_1, \Sigma_1)$*, the condition* $\overline{\mathfrak{B}_1 \wedge \Sigma_1}$ *holds.*

*Proof.* By induction on typing derivations and case analysis (the interesting cases are (T-App) and (T-Let) where the resulting invariant condition does not cover the case when ghost code is used as argument by a non-ghost function, so that argument invariant condition must be written in typing rules explicitly). □

### Ghost-Code Propagation

| $\mathbf{t_1}\ \mathbf{t_2} : \vdash_{g\lambda} t_1 : (\tau_2^{\mathfrak{B}_2} \xrightarrow{\Sigma_0} \tau_1, \mathfrak{B}_1, \Sigma_1) \quad \vdash_{g\lambda} t_2 : (\tau_2, \mathfrak{B}'_2, \Sigma_2) \quad \mathfrak{B}_2 \Rightarrow \mathfrak{B}'_2$ | | | | |
|---|---|---|---|---|
| $\mathfrak{B}_2$ | $\mathfrak{B}_1$ | $\mathfrak{B}'_2$ | result | |
| $\bot$ | $\bot$ | $\bot$ | $\bot$ | raw code application |
| $\top$ | $\bot$ | $\top$ | $\bot$ | ghost-code passing inside normal code |
| $\top$ | $\top$ | $\top$ | $\top$ | ghost code application |
| $\bot$ | $\top$ | $\bot$ | $\top$ | raw code passing inside ghost code |
| $\bot$ | $\top$ | $\top$ | $\top$ | formal parameter contamination |
| $\bot$ | $\bot$ | $\top$ | $\top$ | function parameter and body contamination |
| $\top$ | $\bot$ | $\bot$ | – | impossible |
| $\top$ | $\top$ | $\bot$ | – | impossible |

## 1.3 Ghost Code Erasure

*Definition* 1.1 (Type-Erasure). We define type-erasure function (parametrized by $\{\bot, top\}$) by induction on the structure of types :

$\mathcal{E}_\top(\tau) = \texttt{unit}$
$\mathcal{E}_\bot(\tau_2^{\mathfrak{B}_2} \to \tau_1) = \mathcal{E}_{\mathfrak{B}_2}(\tau_2) \to \mathcal{E}_\bot(\tau_1).$
$\mathcal{E}_\bot(\tau_1) = \tau_1$ otherwise.

Also $\mathcal{E}_\mathfrak{B}(\text{ref } \tau) = \text{ref } \mathcal{E}_\mathfrak{B}(\tau)$.

*Definition* 1.2 (Term-Erasure). Let $\texttt{t}$ be a term such that $\vdash_{g\lambda} t : (\tau, \mathfrak{B}, \Sigma)$ holds. We define term-erasure function $\mathcal{E}_\mathfrak{B}(t)$ by induction on the structure of $\texttt{t}$:
$\mathcal{E}_\top(t_1) = ()$ where $\vdash_{g\lambda} t_1 : (\tau_1, \top, \bot)$.
$\mathcal{E}_\bot(c) = c$
$\mathcal{E}_\bot(x_\tau^\mathfrak{B}) = x_{\mathcal{E}_\bot(\tau)}$
$\mathcal{E}_\bot(\lambda x_{\tau_2}^{\mathfrak{B}_2}.t) = \lambda x_{\mathcal{E}_{\mathfrak{B}_2}(\tau_2)}.\mathcal{E}_\bot(t_1)$ where $\vdash_{g\lambda} t_1 : (\tau_1, \bot, \Sigma)$.
$\mathcal{E}_\bot(t_1\ t_2) = \mathcal{E}_\bot(t_1)\ \mathcal{E}_{\mathfrak{B}_2}(t_2)$
where $\vdash_{g\lambda} t_1 : (\tau_2^{\mathfrak{B}_2} \xrightarrow{\Sigma_0} \tau_1, \bot, \Sigma_1)$ and $\vdash_{g\lambda} t_2 : (\tau_2, \mathfrak{B}_2', \Sigma_2)$.
$\mathcal{E}_\bot(\texttt{let } x_{\tau_2}^{\mathfrak{B}_2} = t_2 \texttt{ in } t_1) = \texttt{let } x_{\mathcal{E}_{\mathfrak{B}_2}(\tau_2)} = \mathcal{E}_{\mathfrak{B}_2}(t_2) \texttt{ in } \mathcal{E}_\bot(\ t_1)$
$\mathcal{E}_\bot(\texttt{rec } f^\bot\ x_{\tau_2}^{\mathfrak{B}_2} : \tau_1.t_1) = \texttt{rec } f\ x_{\mathcal{E}_{\mathfrak{B}_2}(\tau_2)} : \mathcal{E}_\bot(\tau_1).\mathcal{E}_\bot(t_1)$
$\mathcal{E}_\bot(\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3) = \texttt{if } \mathcal{E}_\bot(t_1) \texttt{ then } \mathcal{E}_\bot(t_2) \texttt{ else } \mathcal{E}_\bot(t_3)$
$\mathcal{E}_\bot(!r_{\text{ref } \tau}^\bot) = !r_{\text{ref } \mathcal{E}_\bot(\tau)}$
$\mathcal{E}_\bot(r_{\text{ref } \tau}^\bot := t_2) = \ r_{\text{ref } \mathcal{E}_\bot(\tau)} := \mathcal{E}_\bot(t_2)$

*Definition* 1.3 (Global-Variable-Erasure).
$\mathcal{E}_\top(\texttt{var } r_{\text{ref } \tau}^\top = v\ ) = \emptyset$
$\mathcal{E}_\bot(\texttt{var } r_{\text{ref } \tau}^\bot = v\ ) = \texttt{var } r_{\text{ref } \mathcal{E}_\bot(\tau)} : \mathcal{E}_\bot(v)$

*Definition* 1.4 (Memory-Erasure).
$\mathcal{E}_\bot(\mu) = \{\Big(r_{\text{ref } \mathcal{E}_\bot(\tau)},\ \mathcal{E}_\bot\big(\mu(r_{\text{ref } \tau}^\bot)\big)\Big)\}$

Figure 6: ***ghost*-ml Ghost-Code Erasure**

## 1.4 Properties of ghost erasure

Ghost code is a part of program specification. Suppose we have a source program $p$ that we want to specify using ghost code inside.

First of all, if our specification predicts statically some dynamic behaviour that $p$ does not even have, it is simply unsound.

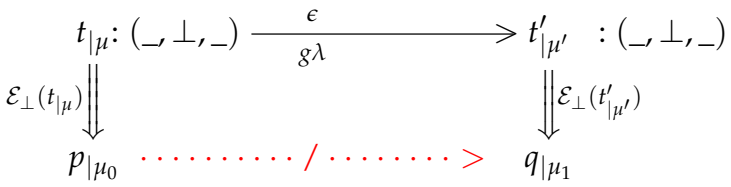


Figure 7: **The ghost-ML "*non ghost*" head reduction from a ghostML term $t$ (program $p$'s specification) to $t'$), to $t'$ does not correspond to any of reductions of $p$.**

On the other hand, if some dynamic behaviour of $p$ escapes from the specification, then such a specification does not reflect properly it's meaning, so it is useless for establishing the correctness of $p$:

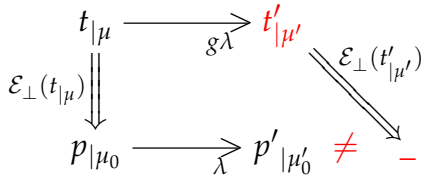


Figure 8: **None of ghost-ML term's $t$ ($p$'s specification) reductions can reflect the reduction step from source program $p$ to $p'$.**

In this section we check the absence of those two pathological situations, using a technique called *bi-simulation*, which consists of the following two theorems:

*Theorem 1.5* [Forward Simulation]. *If $t$ is a closed ghost-ML term, such that $\vdash_{g\lambda} t : (\tau, \bot, \Sigma)$ holds and $t_{|v} \to^\star_{g\lambda} \mu_{|\mu'}$ for some value $v$, then $\mathcal{E}_\bot(t)_{|\mathcal{E}_\bot(\mu)} \to^\star_\lambda \mathcal{E}_\bot(v)_{|\mathcal{E}_\bot(\mu')}$.*

*Theorem 1.6* [Backward Simulation]. *If $t$ is a closed ghost-ML term, such that $\vdash_{g\lambda} t : (\tau, \bot, \Sigma)$ holds and $\mathcal{E}_\bot(t)_{|\mathcal{E}_\bot(\mu)} = t_{0|\mu_0} \to_\lambda t_{1|\mu_1}$ for some ML value $t_1$ and*

*store $\mu_1$, then there exist a ghost-ML term $t'$ and a store $\mu'$ such that $t_1 = \mathcal{E}_\perp(t')$, $\mu 1 = \mathcal{E}_\perp(\mu')$ and $t_{|\mu} \to^\star_{g\lambda} t'_{|\mu'}$ .*

We begin by stating auxiliary lemmas which will help us deal with proving these theorems.

### 1.4.1 Forward simulation

*Lemma 1.2* [One Step Forward Simulation]. *If $t$ is a closed ghost-ML term, such that $\vdash_{g\lambda} t : (\tau, \perp, \Sigma)$ holds and $t_{|\mu} \to_{g\lambda} t'_{|\mu'}$ , then $\mathcal{E}_\perp(t)_{\mathcal{E}_\perp(\mu)} \to^{0|1} \mathcal{E}_\perp(t')_{\mathcal{E}_\perp(\mu')}$.*

*Proof.* By induction on the evaluation relation $t_{|\mu} \to_{g\lambda} t'_{|\mu'}$ .

($\alpha$)

($\beta$) Otherwise, $t$ is of the form $\texttt{let}\ x^{\mathfrak{B}_2}_{\tau_2} = t_2\ \texttt{in}\ t_1$ and the only case to consider is:

$$\frac{t_{2|\mu} \to t'_{2|\mu'}}{\texttt{let}\ x^{\mathfrak{B}_2}_{\tau} = t_2\ \texttt{in}\ t_{1|\mu} \to \texttt{let}\ x^{\mathfrak{B}_2}_{\tau} = t'_2\ \texttt{in}\ t_{1|\mu'}} \qquad \text{(E-Context)}$$

with the typing

$$\frac{\vdash_{g\lambda} t_1 : (\tau_1, \mathfrak{B}_1, \Sigma_1) \quad \vdash_{g\lambda} t_2 : (\tau_2, \mathfrak{B}'_2, \Sigma_2) \quad \mathfrak{B}_2 \Rightarrow \mathfrak{B}'_2}{\vdash_{g\lambda} \text{let}\ x^{\mathfrak{B}_2}_{\tau_2} = t_2 \text{ in } t_1 : (\tau_1, \mathfrak{B}_1 \vee (\overline{\mathfrak{B}_2} \wedge \mathfrak{B}'_2), \Sigma_1 \vee \Sigma_2)} \qquad \text{(T-Let)}$$

By hypothesis, $t$ itself is not ghost, so looking at the typing of `let`, we can deduce that either $\mathfrak{B}_2 = \mathfrak{B}'_2 = \top$ or $\mathfrak{B}_2 = \mathfrak{B}'_2 = \perp$.

($\beta_1$) If term $t_2$ is ghost, then

$$\mathcal{E}_\perp(\texttt{let}\ x^{\top}_{\tau_2} = t_2\ \texttt{in}\ t_1) = \texttt{let}\ x_{\texttt{unit}} = ()\ \texttt{in}\ \mathcal{E}_\perp(\ t_1) = \mathcal{E}_\perp(\texttt{let}\ x^{\top}_{\tau_2} = t'_2\ \texttt{in}\ t_1)$$

That is, $\mathcal{E}_\perp(t)_{|\mathcal{E}_\perp(\mu)} \to^0 \mathcal{E}_\perp(t')_{|\mathcal{E}_\perp(\mu')}$.

($\beta_2$) Otherwise, the binding of $x^{\mathfrak{B}_2}_{\tau_2}$ is non-ghost and we can apply the induction hypothesis on $t_{2|\mu} \to_{g\lambda} t'_{2|\mu'}$ :

$$\mathcal{E}_\perp(t_2)_{|\mathcal{E}_\perp(\mu)} \to^{0|1} \mathcal{E}_\perp(t'_2)_{|\mathcal{E}_\perp(\mu')}$$

and conclude in both cases ($\mathcal{E}_\perp(t_2)_{|\mathcal{E}_\perp(\mu)} = \mathcal{E}_\perp(t'_2)_{|\mathcal{E}_\perp(\mu')}$ or $\mathcal{E}_\perp(t_2)_{|\mathcal{E}_\perp(\mu)} \to \mathcal{E}_\perp(t'_2)_{|\mathcal{E}_\perp(\mu')}$) with application of E-Context rule in ML :

$$\frac{\mathcal{E}_\perp(t_2)_{|\mathcal{E}_\perp(\mu)} \to^{0|1} \mathcal{E}_\perp(t'_2)_{|\mathcal{E}_\perp(\mu')}}{\mathcal{E}_\perp(t)_{|\mathcal{E}_\perp(\mu)} \to^{0|1} \mathcal{E}_\perp(t')_{|\mathcal{E}_\perp(\mu')}} \qquad \text{(E-Context)}$$

□

*Theorem 1.7* [Forward Simulation]. *If $t$ is a closed ghost-ML term, such that $\vdash_{g\lambda} t : (\tau, \bot, \Sigma)$ holds and $t_{|\mu} \to^{\star}_{g\lambda} v_{|\mu'}$ for some value $v$, then $\mathcal{E}_{\bot}(t)_{|\mathcal{E}_{\bot}(\mu)} \to^{\star}_{\lambda} \mathcal{E}_{\bot}(v)_{|\mathcal{E}_{\bot}(\mu')}$ .*

*Proof.* By induction on the length of the evaluation of $t_{|\mu} \to^{\star}_{g\lambda} v_{|\mu'}$ .

($\alpha$) if $t_{|\mu} \xrightarrow{\epsilon}_{g\lambda} v_{|\mu'}$ then by *one-step* forward simulation lemma,

$$\mathcal{E}_{\bot}(t)_{\mathcal{E}_{\bot}(\mu)} \to^{0|1} \mathcal{E}_{\bot}(v)_{\mathcal{E}_{\bot}(\mu')}.$$

($\beta$) Assume now that $t_{|\mu} \to_{g\lambda} t''_{|\mu''}$ and $t''_{\mu''} \to^{n} v_{\mu'}$ for some arbitrary $n \in \mathbb{N}$. By the progress of Ghost-ML typing, $\vdash_{g\lambda} t'' : (\tau, \bot, \Sigma'')$ (for some $\Sigma$ with $\Sigma \Rightarrow \Sigma''$ and some store $\mu''$). By induction hypothesis on $t''$, and by *one step* forward lemma applied to $t$, we have that

$$\mathcal{E}_{\bot}(t)_{\mathcal{E}_{\bot}(\mu)} \to^{0|1} \mathcal{E}_{\bot}(t'')_{\mathcal{E}_{\bot}(\mu'')} \to^{n} \mathcal{E}_{\bot}(v)_{\mathcal{E}_{\bot}(\mu')}.$$

That is, for any $n \in \mathbb{N}$, $\mathcal{E}_{\bot}(t)_{|\mathcal{E}_{\bot}(\mu)} \to^{\star}_{g\lambda} \mathcal{E}_{\bot}(v)_{|\mathcal{E}_{\bot}(\mu')}$ .

□

### 1.4.2 Evaluation Preservation

*Lemma 1.3* [Substitution under erasure].
*If $\vdash_{g\lambda} t_1 : (\tau_1, \bot, \Sigma_1)$ and $\vdash_{g\lambda} v_2 : (\tau_2, \mathfrak{B}_2, \Sigma_2)$ hold, then $\mathcal{E}_{\bot}(t_1[x^{\mathfrak{B}_2}_{\tau_2} \leftarrowtail v_2]) = \mathcal{E}_{\bot}(t_1)[x_{\mathcal{E}_{\mathfrak{B}_2}(\tau_2)} \leftarrowtail \mathcal{E}_{\mathfrak{B}_2}(v_2)]$*

*Proof.* By induction on the structure of $\mathtt{t}_1$. □

*Lemma 1.4* [One-step evaluation under erasure]. *For any closed $g\lambda$-term $\mathtt{t}$ such that $\vdash_{g\lambda} t : (\tau, \bot, \Sigma_1)$ holds, if $t_{|\mu} \to_{g\lambda} t'_{|\mu'}$ for some term $\mathtt{t}'$, then either $\mathcal{E}_{\bot}(t) \to_{\lambda} \mathcal{E}_{\bot}(t')$ or $\mathcal{E}_{\bot}(t) = \mathcal{E}_{\bot}(t')$.*

*Proof.* By induction on the evaluation relation of $\mathtt{t} \to_{g\lambda} \mathtt{t}'$.

*Case* E-AppAbs: $\mathtt{t} = (\lambda x^{\mathfrak{B}_2}_{\tau_2}.\mathtt{t}_1)\mathtt{v}_1$ with $(\lambda x^{\mathfrak{B}_2}_{\tau_2}.\mathtt{t}_1)\mathtt{v}_1 \xrightarrow{\epsilon}_{g\lambda} \mathtt{t}_1[x^{\mathfrak{B}_2}_{\tau_2} \leftarrowtail \mathtt{v}_1]$
$\vdash_{g\lambda} (\lambda x^{\mathfrak{B}_2}_{\tau_2}.\mathtt{t}_1)\mathtt{v}_1 : (\tau_1, \mathfrak{B}_1)$, $\vdash_{g\lambda} \mathtt{v}_1 : (\tau_2, \mathfrak{B}'_2)$,
$\mathfrak{B}_1 = \bot$, $\vDash \mathfrak{B}_2 \Leftrightarrow \mathfrak{B}'_2$

| | |
|---|---|
| $\mathcal{E}_{\bot}[(\lambda x^{\mathfrak{B}_2}_{\tau_2}.\mathtt{t}_1)\mathtt{v}_1]$ | |
| $= \lambda x_{\mathcal{E}_{\mathfrak{B}_2}(\tau_2)}.\mathcal{E}_{\bot}(\mathtt{t}_1))\mathcal{E}_{\mathfrak{B}'_2}(\mathtt{v}_1)$ | (as $\mathfrak{B}_1 = \bot$) |
| $\xrightarrow{\epsilon}_{\lambda} \mathcal{E}_{\bot}(\mathtt{t}_1)[x_{\mathcal{E}_{\mathfrak{B}_2}(\tau_2)} \leftarrowtail \mathcal{E}_{\mathfrak{B}'_2}(\mathtt{v}_1)]$ | (head red.) |
| $= \mathcal{E}_{\bot}(\mathtt{t}_1[x^{\mathfrak{B}_2}_{\tau_2} \leftarrowtail \mathtt{v}_1])$ | (by Substitution under erasure lemma) |

*Case* E-DeGhost:
Trivially verified, as for any instance of $\vdash_{g\lambda} (\mathit{ghost}\ \mathtt{t}_1) : (\tau_1, \mathfrak{B}_1)$, $\mathfrak{B}_1 = \top$.

*Case* E-AppLeft: $\mathtt{t} = \mathtt{t}_1\mathtt{t}_2$, $\mathtt{t}' = \mathtt{t}_1'\mathtt{t}_2$, with $\mathtt{t}_1 \to_{g\lambda} \mathtt{t}_1'$
$\vdash_{g\lambda} \mathtt{t}_1 : (\tau_2^{\mathfrak{B}_2} \to \tau_1, \mathfrak{B}_1)$, $\vdash_{g\lambda} \mathtt{t}_2 : (\tau_2, \mathfrak{B}_2')$,
$\mathfrak{B}_1 = \bot$, $\models \mathfrak{B}_2 \Leftrightarrow \mathfrak{B}_2'$

As $\mathfrak{B}_1 = \bot$, we can apply induction hypothesis on $\mathtt{t}_1$ which gives $\mathcal{E}_\bot(\mathtt{t}_1) \to_\lambda \mathcal{E}_\bot(\mathtt{t}_1')$. Then, applying E-AppRight rule, we obtain:

$$\mathcal{E}_\bot(\mathtt{t}) = \mathcal{E}_\bot(\mathtt{t}_1)\mathcal{E}_\bot(\mathtt{t}_2) \to_\lambda \mathcal{E}_\bot(\mathtt{t}_1')\mathcal{E}_\bot(\mathtt{t}_2) = \mathcal{E}_\bot(\mathtt{t}').$$

*Case* E-AppRight: $\mathtt{t} = \mathtt{v}_1\mathtt{t}_2$, $\mathtt{t}' = \mathtt{v}_1\mathtt{t}_2'$, with $\mathtt{t}_2 \to_{g\lambda} \mathtt{t}_2'$
$\vdash_{g\lambda} \mathtt{v}_1 : (\tau_2^{\mathfrak{B}_2} \to \tau_1, \mathfrak{B}_1)$, $\vdash_{g\lambda} \mathtt{t}_2 : (\tau_2, \mathfrak{B}_2')$,
$\mathfrak{B}_1 = \bot$, $\models \mathfrak{B}_2 \Leftrightarrow \mathfrak{B}_2'$
If $\mathfrak{B}_2 = \mathfrak{B}_2' = \top$, then

$$\mathcal{E}_\bot(\mathtt{t}) = \mathcal{E}_\bot(\mathtt{v}_1)\mathcal{E}_\top(\mathtt{t}_2) = (\mathcal{E}_\bot(\mathtt{v}_1))() = \mathcal{E}_\bot(\mathtt{v}_1)\mathcal{E}_\top(\mathtt{t}_2') = \mathcal{E}_\bot(\mathtt{t}').$$

Otherwise, $\mathfrak{B}_2 = \mathfrak{B}_2' = \bot$. By induction hypothesis, $\mathcal{E}_\bot(\mathtt{t}_2) \to_\lambda \mathcal{E}_\bot(\mathtt{t}_2')$. Then, applying E-AppRight rule, we obtain:

$$\mathcal{E}_\bot(\mathtt{t}) = \mathcal{E}_\bot(\mathtt{v}_1)\mathcal{E}_\bot(\mathtt{t}_2) \to_\lambda \mathcal{E}_\bot(\mathtt{v}_1)\mathcal{E}_\bot(\mathtt{t}_2') = \mathcal{E}_\bot(\mathtt{t}').$$

□

Now we can prove the main theorem.

*Theorem 1.8* [Value preservation under erasure]. *For any closed $g\lambda$-term $t$ such that $\vdash_{g\lambda} t : (\tau, \bot)$ holds, if $t \to_{g\lambda}^* v$ for some value $v$, then $\mathcal{E}(t) \to_\lambda^* \mathcal{E}(v)$.*

*Proof.* By induction on the length of the evaluation of $\mathtt{t} \to_{g\lambda}^* \mathtt{v}$.

We already have proved the base case : indeed, if $\mathtt{t} \to_{g\lambda} \mathtt{v}$ then by the one-step evaluation lemma, $\mathcal{E}_\bot(\mathtt{t}) \to_\lambda^{0|1} \mathcal{E}_\bot(\mathtt{v})$.

Now, assume that $\mathtt{t} \to_{g\lambda}^1 \mathtt{t}' \to_{g\lambda}^n \mathtt{v}$ for some arbitrary $n \in \mathbb{N}$. By the progress of typing, $\vdash_{g\lambda} \mathtt{t}' : (\tau, \bot)$, so we can apply induction hypothesis on $\mathtt{t}'$ which gives $\mathcal{E}(\mathtt{t}') \to_\lambda^* \mathcal{E}(\mathtt{v})$. By the one-step evaluation lemma again, we have $\mathcal{E}_\bot(\mathtt{t}) \to_\lambda^{0|1} \mathcal{E}_\bot(\mathtt{t}')$. That is, $\mathcal{E}_\bot(\mathtt{t}) \to_\lambda^* \mathcal{E}_\bot(\mathtt{v})$. □

### 1.4.3 Typing Erasure

*Lemma 1.5* [Typing relation under erasure].
*If* $\vdash_{g\lambda}\ t : (\tau, \bot)$ *then* $\vdash_{\lambda}\ \mathcal{E}_{\bot}(t)\ :\ \mathcal{E}_{\bot}(\tau)$.

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

*Case* T-Unit: $\vdash_{g\lambda} \mathtt{()} : (\mathtt{unit}, \bot)$

Immediately by definition of $\mathcal{E}_{\bot}$.

*Case* T-Var: $\vdash_{g\lambda} x^{\bot}_{\tau} : (\tau, \bot)$

$\mathcal{E}_{\bot}(x^{\bot}_{\tau}) = x_{\mathcal{E}_{\bot}(\tau)}$ gives immediately $\vdash_{\lambda} x_{\mathcal{E}_{\bot}(\tau)} : \mathcal{E}(\tau)$.

*Case* T-Abs: $\vdash_{g\lambda} \lambda x^{\mathfrak{B}_2}_{\tau_2}.\mathtt{t}_1 : (\tau_2^2 \to \tau_1, \bot)$ with $\vdash_{g\lambda} \mathtt{t}_1 : (\tau_1, \bot)$

By induction hypothesis $\vdash_{\lambda} \mathcal{E}_{\bot}(\mathtt{t}_1) : \mathcal{E}_{\bot}(\tau_1)$. There are two cases to consider, depending on whether the parameter of the abstraction is ghost or not. If $\mathfrak{B}_2 = \top$ then $\mathcal{E}_{\bot}(\lambda x^{\top}_{\tau_2}.\mathtt{t}_1) = \lambda x_{\mathtt{unit}}.\mathcal{E}(\mathtt{t}_1)$ and therefore

$$\frac{\vdash_{\lambda} \mathcal{E}_{\bot}(\mathtt{t}_1) : \mathcal{E}_{\bot}(\tau_2)}{\vdash_{\lambda} \lambda x_{\mathtt{unit}}.\mathcal{E}_{\bot}(\mathtt{t}_1) : \mathtt{unit} \to \mathcal{E}_{\bot}(\tau_1)} \qquad \text{(T-Abs)}$$

Otherwise $\mathfrak{B}_2 = \bot$ and again by the rule T-Abs we obtain :

$$\frac{\vdash_{\lambda} \mathcal{E}_{\bot}(\mathtt{t}_1) : \mathcal{E}_{\bot}(\tau_1)}{\vdash_{\lambda} \lambda x_{\mathcal{E}_{\bot}(\tau_2)}.\mathcal{E}_{\bot}(\mathtt{t}_1) : \mathcal{E}_{\bot}(\tau_2) \to \mathcal{E}_{\bot}(\tau_1)} \qquad \text{(T-Abs)}$$

*Case* T-App: $\vdash_{g\lambda} \mathtt{t}_1\ \mathtt{t}_2 : (\tau_1, \bot)$ with sub-derivations:

$\vdash_{g\lambda} \mathtt{t}_1 : (\tau_2^{\mathfrak{B}_2} \to \tau_1, \mathfrak{B}_1)$

$\vdash_{g\lambda} \mathtt{t}_2 : (\tau_2, \mathfrak{B}'_2)$,

As $\vdash_{g\lambda} \mathtt{t}_1\ \mathtt{t}_2 : (\tau_1, \bot)$, the inversion lemma gives as By inversion that $\vDash \mathfrak{B}_2 \Leftrightarrow \mathfrak{B}'_2$. That is, we have two cases to consider.

If $\mathfrak{B}_2 = \mathfrak{B}'_2 = \bot$ then by induction hypotheses
$\vdash_{\lambda} \mathcal{E}_{\bot}(\mathtt{t}_1) : \mathcal{E}_{\bot}(\tau_2) \to \mathcal{E}_{\bot}(\tau_1)$ and $\vdash_{\lambda} \mathcal{E}_{\bot}(\mathtt{t}_2) : \mathcal{E}_{\bot}(\tau_2)$. By T-App rule,

$$\frac{\vdash_{\lambda} \mathcal{E}_{\bot}(\mathtt{t}_1) : \mathcal{E}_{\bot}(\tau_2) \to \mathcal{E}_{\bot}(\tau_1) \qquad \vdash_{\lambda} \mathcal{E}_{\bot}(\mathtt{t}_2) : \mathcal{E}_{\bot}(\tau_2)}{\vdash_{\lambda} \mathcal{E}_{\bot}(\mathtt{t}_1\ \mathtt{t}_2) : \mathcal{E}(\tau_1)} \qquad \text{(T-App)}$$

If $\mathfrak{B}_2 = \mathfrak{B}'_2 = \top$, then by definition of $\mathcal{E}$ we have $\mathcal{E}(\mathtt{t}_2) = \mathtt{()}$ and $\mathcal{E}_{\mathfrak{B}'_2}(\tau_2) = \mathtt{unit}$. By induction hypothesis on $\mathtt{t}_1$, $\vdash_{\lambda} \mathcal{E}_{\bot}(\mathtt{t}_1) : \mathtt{unit} \to \mathcal{E}_{\bot}(\tau_1)$. Applying T-App rule gives us

$$\dfrac{\dfrac{}{\vdash_\lambda \mathcal{E}_\perp(\mathtt{t}_1\ \mathtt{t}_2) : \mathcal{E}(\tau_1)} \quad \dfrac{}{\vdash_\lambda \mathtt{()} : \mathtt{unit}}\ \text{(T-Unit)}}{\vdash_\lambda \mathcal{E}_\perp(\mathtt{t}_1\ \mathtt{t}_2) : \mathcal{E}_\perp(\tau_1)}\ \text{(T-App)}$$

The case of (T-Ghost) as well as any other valid derivation where a typed term is marked as ghost do not satisfy lemma's requirement, so these cases are trivially verified. □

# 2 logic

In previous section we described how to define formally ghost terms inside ML-like programs.

However, the ghost code becomes useful only within a full-featured specification language.

In this section we define a high-order logic with simple types such as units, integers, boolean. To make our examples more interesting we also provide built-in integer lists and trees.

First off, we describe our logic's syntax, semantics and typing. Then we extend the ghost-ML language with specifications such as assertions and functional pre- and post-conditions.

| | | | | | |
|---|---|---|---|---|---|
| $\tau$ | ::= | TYPES | $c$ | ::= | CONSTANTS |
| | unit \| int \| bool | *built-in simple types* | | $0 \mid 1 \mid ... \mid n$ | *integer* |
| | list int \| tree int | *built-in recursive types* | | `true` \| `false` | *boolean* |
| | $\tau \to \tau$ | *function type* | | $()$ | *unit* |
| | `prop` | *proposition* | | $\mu$`l. Nil \| Cons n l` | *integer list* |
| | | | | $\mu$`t. Empty \| Node t n t` | *integer binary tree* |
| $t$ | ::= | TERMS | | `True` \| `False` | *proposition value* |
| | $c$ | *constant* | | $+ \mid - \mid = \mid ...$ | *built-in operators* |
| | $x_\tau$ | *variable* | | | |
| | $\lambda x_\tau.t$ | *abstraction* | $f$ | ::= | FORMULAS |
| | $t\ t$ | *application* | | $True \mid False$ | *logical truth values* |
| | let $x_\tau = t$ in $t$ | *local binding* | | $t \wedge t$ | *conjunction* |
| | rec $g\ x_\tau : \tau = t$ | *recursive function* | | $t \vee t$ | *disjunction* |
| | $f$ | *formula* | | $\neg t$ | *negation* |
| | | | | $\exists x_\tau^{\mathfrak{B}}\, x\tau.f$ | *existential quantification* |
| | | | | $\forall x_\tau^{\mathfrak{B}}\, x\tau.f$ | *universal quantification* |
| | | | | $f = f$ | *equality* |

Figure 9: Logic Syntax

# 3 Inlining

**Motivation:** instantiate higher-order iterators with previously defined or anonymous functions, in order to obtain a first-order function, whose proof obligation is of the same complexity as p.o. of equivalent loop statement.

**Goal:** reduce every application of a higher-order function to a term that is not a bound variable.

**Input:** a higher-order language in *A-normal* form with functions whose codomain is of some base type, and whose formal parameters is partially ordered (the higher order parameters come before lesser order parameters)

**Output:** a language where the only high-order applications are those where argument of application is a bound variable. That is, a language where higher-order applications can occur only under **inside** high-order functions.

**Input Language Syntax:**

---

$$t \quad ::= \quad v \mid vv \mid let$$

Figure 10: Source language in A-normal form

---