Liquid Meta

Michael Borkowski
(summarizing joint work with Ranjit Jhala and Niki Vazou)

June 8, 2020

1 Our language λ_2

We work with a polymorphic, typed lambda calculus with call-by-value semantics which is augmented by refinement types, dependent function types, and existential types. Our language is based on the Sprite language λ in Jhala and Vazou's forthcoming manuscript [JV] and incorporates and extends aspects from the λ^H of Vazou et al [VSJ$^+$14]. The existential types were used in a metatheory by Knowles and Flanagan [KF09].

We start with the syntax of term-level expressions in our language:

Values	$v :=$	$\texttt{true}, \texttt{false}$	*boolean constants*
	$\mid$	$0, 1, 2, \ldots$	*integer constants*
	$\mid$	x	*variables*
	$\mid$	$\lambda x.e$	*abstractions*
	$\mid$	$\Lambda \alpha{:}k.e$	*type abstractions*
	$\mid$	$\wedge,\ \vee,\ \neg,\ \leftrightarrow,\ \leq,\ =$	*built−in primitives*

Expressions	$e :=$	v	*values*
	$\mid$	$e_1\ e_2$	*applications*
	$\mid$	$e\ [t]$	*type applications*
	$\mid$	$\texttt{let}\ x = e_1\ \texttt{in}\ e_2$	*let expressions*
	$\mid$	$e_1 : t$	*annotations*

Next, we give the syntax of the types and binding environments used in our language:

Basic types	$b :=$	Bool	*booleans*
	$\mid$	Int	*integers*

Types	$t :=$	$b\{r\}$	*refinement*
	$\mid$	α	*type variables*
	$\mid$	$x{:}t_x \rightarrow t$	*dependent function*

$$
\begin{array}{rrll}
 & | & \exists\, x{:}t_x.\, t & \textit{existential} \\
 & | & \forall\, \alpha{:}k.\, t & \textit{polymorphic} \\
 & & & \\
\mathsf{Kinds} & k ::= & B & \textit{base kind} \\
 & | & * & \textit{star kind} \\
 & & & \\
\mathsf{Environments} & \Gamma ::= & \varnothing & \textit{empty} \\
 & | & \Gamma, x{:}t & \textit{bind variable} \\
 & | & \Gamma, \alpha{:}k & \textit{bind type variable}
\end{array}
$$

Next, we give the syntax of the Boolean predicates and constraints involved in refinements and subtyping judgments. The ternary judgment $\vdash_B :$ is the typing judgment in the underlying System F calculus.

$$\mathsf{Refinements} \quad r ::= \{x : p\}$$

$$\mathsf{Predicates} \quad p ::= \quad \{e \mid \exists\,\Gamma.\,\Gamma \vdash_B e : \mathsf{Bool}\} \qquad \textit{expressions of type } \mathsf{Bool}$$

In the metatheory here we require that all variables bound in the environment be distinct. In the mechanization we use the locally-named representation: free and bound variables become distinct objects in the syntax. All free variables have unique names and these names never conflict with bound variables, which eliminates the possibility of capture in substitution and the need to perform alpha-renamings during substitution. This came at the cost of needing formal lemmas which permit us to change the name of a variable bound in the environment to maintain the uniqueness of the free variables only.

Our definition of predicates above departs from the Sprite languages of [JV] by allowing predicates to be arbitrary expressions from the main language (which are Boolean typed under the appropriate binding environment). In [JV] however, predicates are quantifier-free first-order formulae over a vocabulary of integers and a limited number of relations. We initially took this approach, but were unable to fully define the denotational semantics for this type of language. In particular, when we define closing substitutions we need to define the substitution of a type $\theta(t)$ as the type resulting from t after performing substitutions for all variables bound to values in $\theta = (x_1 \mapsto v_1, \ldots, x_n \mapsto v_n)$. Substituting arbitrary expressions into t requires substituting arbitrary expressions into predicates, and it isn't clear how to do this for functions like $(\lambda x.x)$ without taking predicates to be all Boolean-typed program expressions.

Returning to our λ_2, we next define the operational semantics of the language. We treat the reduction rules (small step semantics) of the various built-in primitives as external to our language, and we denote by $\delta(c, v)$ a function specifying them. The reductions are defined in a curried manner, so for instance we have that $c\ v_1\ v_2 \hookrightarrow^* \delta(\delta(c, v_1), v_2)$. Currying gives us unary relations like $m{\leq}$ which is a partially evaluated version of the $\leq$ relation.

$$\delta(\wedge, \mathtt{true}) := \lambda x.\, x \qquad\qquad \delta(\leftrightarrow, \mathtt{true}) := \lambda x.\, x$$

$$
\begin{aligned}
\delta(\wedge, \texttt{false}) &:= \lambda x.\, \texttt{false} & \delta(\leftrightarrow, \texttt{false}) &:= \lambda x.\, \neg x \\
\delta(\vee, \texttt{true}) &:= \lambda x.\, \texttt{true} & \delta(\leq, m) &:= m{\leq} \\
\delta(\vee, \texttt{false}) &:= \lambda x.\, x & \delta(m{\leq}, n) &:= \texttt{bval}(m \leq n) \\
\delta(\neg, \texttt{true}) &:= \texttt{false} & \delta(=, m) &:= m{=} \\
\delta(\neg, \texttt{false}) &:= \texttt{true} & \delta(m{=}, n) &:= \texttt{bval}(m = n)
\end{aligned}
$$

Now we give the reduction rules for the small-step semantics. In what follows, e and its variants refer to an arbitrary expression, v refers to a value, x to a variable, and c refers to a built-in primitive.

$$
\frac{}{c\ v \hookrightarrow \delta(c, v)}\ \textsc{E-Prim} \qquad \frac{e \hookrightarrow e'}{e\ e_1 \hookrightarrow e'\ e_1}\ \textsc{E-App1}
$$

$$
\frac{e \hookrightarrow e'}{v\ e \hookrightarrow v\ e'}\ \textsc{E-App2} \qquad \frac{}{(\lambda x.\, e)\ v \hookrightarrow e[v/x]}\ \textsc{E-AppAbs}
$$

$$
\frac{e \hookrightarrow e'}{e\ [t] \hookrightarrow e'\ [t]}\ \textsc{E-AppT} \qquad \frac{}{(\Lambda \alpha{:}k.\, e)\ [t] \hookrightarrow e[t/\alpha]}\ \textsc{E-AppTAbs}
$$

$$
\frac{e_x \hookrightarrow e_x'}{\texttt{let}\, x{=}e_x\, \texttt{in}\, e \hookrightarrow \texttt{let}\, x{=}e_x'\, \texttt{in}\, e}\ \textsc{E-Let} \qquad \frac{}{\texttt{let}\, x{=}v\, \texttt{in}\, e \hookrightarrow e[v/x]}\ \textsc{E-LetV}
$$

$$
\frac{e \hookrightarrow e'}{e : t \hookrightarrow e' : t}\ \textsc{E-Ann} \qquad \frac{}{v : t \hookrightarrow v}\ \textsc{E-AnnV}
$$

We give the details of the type substitution operation used above in E-AppTAbs: *(note: decide on whether the type variable is a basic type or whether it cannot be refined)*

$$
\begin{aligned}
b\{x : p\}[t_\alpha/\alpha] &: = b\{x : p[t_\alpha/\alpha]\} \\
\alpha[t_\alpha/\alpha] &: = t_\alpha \\
(x{:}t_x \to t)[t_\alpha/\alpha] &: = x{:}(t_x[t_\alpha/\alpha]) \to t[t_\alpha/\alpha] \\
(\exists\, x{:}t_x.\, t)[t_\alpha/\alpha] &: = \exists\, x{:}(t_x[t_\alpha/\alpha]).\, t[t_\alpha/\alpha] \\
(\forall\, \alpha'{:}k.\, t)[t_\alpha/\alpha] &: = \forall\, \alpha'{:}k.\, t[t_\alpha/\alpha] & \alpha \neq \alpha'
\end{aligned}
$$

Next, we define the typing rules of our λ_2. The type judgments in the language λ_2 will be denoted $\vdash$ with a colon between term and type. For clarity, we distinguish between this and other judgments by using $\vdash$ with a subscript in most other settings. For instance, the judgement $\Gamma \vdash_w t : k$ says that type t is well-formed in environment Γ and has kind k:

$$\frac{y{:}b, \lfloor\Gamma\rfloor \vdash_B e[y/x] : \mathsf{Bool} \qquad y \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash_w b\{x : e\} : B}\ \textsc{WF-Refn} \qquad \frac{\Gamma \vdash_w t : B}{\Gamma \vdash_w t : *}\ \textsc{WF-Kind}$$

$$\frac{\alpha : k \in \Gamma}{\Gamma \vdash_w \alpha : k}\ \textsc{WF-Var} \qquad \frac{\Gamma \vdash_w t_x : k_x \qquad y{:}t_x, \Gamma \vdash_w t[y/x] : k \qquad y \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash_w x{:}t_x \to t : *}\ \textsc{WF-Func}$$

$$\frac{\Gamma \vdash_w t_x : k_x \quad y{:}t_x, \Gamma \vdash_w t[y/x] : k \quad y \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash_w \exists\, x{:}t_x.\, t : k}\ \textsc{WF-Exis}$$

$$\frac{\alpha'{:}k, \Gamma \vdash_w t[\alpha'/\alpha] : k_t \quad \alpha' \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash_w \forall\, \alpha{:}k.t : *}\ \textsc{WF-Poly}$$

The judgment $\vdash_w \Gamma$ says that the environment Γ is well formed, meaning that variables are only bound to well-formed types. We adopt the convention that our environments grow from right to left.

$$\frac{}{\vdash_w \varnothing}\ \textsc{WFE-Empty} \qquad \frac{\Gamma \vdash_w t_x : k_x \qquad \vdash_w \Gamma \quad x \notin \mathrm{dom}(\Gamma)}{\vdash_w x{:}t_x, \Gamma}\ \textsc{WFE-Bind}$$

$$\frac{\vdash_w \Gamma \quad \alpha \notin \mathrm{dom}(\Gamma)}{\vdash_w \alpha{:}k, \Gamma}\ \textsc{WFE-BindT}$$

Now we give the rules for the typing judgements. As with the reduction rules, we take the type of our built-in primitives to be external to our language. We denote by $ty(c)$ the function that specifies the most specific type possible for c. More details on $ty(c)$ are given in the next section. In order to express the exact type of variables, we introduce a "selfification" function that strengthens a refinement we the condition that a value is equal to itself; this is key to derive the fine grained type of $\lambda x.x$ being $x{:}\mathsf{Bool}\{z : \mathtt{true}\} \to \mathsf{Bool}\{z : z = x\}$. *The $=$ in the $z = x$ definition below is overloaded, but in our mechanization we would use either $z \leftrightarrow x$ or $z = x$ depending on the base type. But if we can refine type variables, then $=$ should be polymorphic.*

$$\begin{aligned}
\mathrm{self}(b\{z : p\}, x) &:= b\{z : p \wedge z = x\}\\
\mathrm{self}(\alpha, x) &:= \alpha\\
\mathrm{self}(z{:}t_z \to t, x) &:= z{:}t_z \to t\\
\mathrm{self}(\exists\, z{:}t_z.\, t, x) &:= \exists\, z{:}t_z.\, t\\
\mathrm{self}(\forall\, \alpha{:}k.\, t, x) &:= \forall\, \alpha{:}k.\, t
\end{aligned}$$

$$\frac{ty(c) = t}{\Gamma \vdash c : t}\ \textsc{T-Prim} \qquad \frac{x{:}t \in \Gamma}{\Gamma \vdash x : \mathrm{self}(t, x)}\ \textsc{T-Var} \qquad \frac{\Gamma \vdash e \,:\, x{:}t_x \to t \qquad \Gamma \vdash e' : t_x}{\Gamma \vdash e\ e' : \exists\, x{:}t_x.\, t}\ \textsc{T-App}$$

$$\frac{y{:}t_x, \Gamma \vdash e[y/x] : t[y/x] \quad \Gamma \vdash_w t_x : k_x \quad y \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash \lambda x.\, e \,:\, x{:}t_x \to t}\ \textsc{T-Abs}$$

$$\frac{\Gamma \vdash e \,:\, \forall\, \alpha{:}k.\, s \qquad \Gamma \vdash_w t : k}{\Gamma \vdash e[t] : s[t/\alpha]}\ \textsc{T-AppT}$$

$$\frac{\alpha'{:}k, \Gamma \vdash e[\alpha'/\alpha] : t[\alpha'/\alpha] \quad \alpha'{:}k, \Gamma \vdash_w t : k' \quad \alpha' \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash \Lambda\alpha{:}k.e : \forall\, \alpha{:}k.\, t}\ \textsc{T-AbsT}$$

$$\frac{\Gamma \vdash e_x : t_x \quad y{:}t_x, \Gamma \vdash e_2[y/x] : t[y/x] \quad \Gamma \vdash_w t : k \quad y \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash \texttt{let}\, x{=}e_x\, \texttt{in}\, e : t}\ \textsc{T-Let}$$

$$\frac{\Gamma \vdash e : t \quad \Gamma \vdash_w t : k}{\Gamma \vdash e : t \,:\, t}\ \textsc{T-Ann} \qquad \frac{\Gamma \vdash e : s \qquad \Gamma \vdash s <: t \qquad \Gamma \vdash_w t : k}{\Gamma \vdash e : t}\ \textsc{T-Sub}$$

The last rule, T-Sub, uses the subtyping judgement $\Gamma \vdash s <: t$. The subtyping rules are as follows:

$$\frac{y{:}b\{x_1 : p_1\}, \Gamma \vdash_e p_2[y/x_2] \qquad y \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash b\{x_1 : p_1\} <: b\{x_2 : p_2\}}\ \textsc{S-Base}$$

$$\frac{\Gamma \vdash s_2 <: s_1 \qquad y{:}s_2, \Gamma \vdash t_1[y/x_1] <: t_2[y/x_2] \qquad y \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash x_1{:}s_1 \to t_1 <: x_2{:}s_2 \to t_2}\ \textsc{S-Func}$$

$$\frac{\Gamma \vdash v_x : t_x \qquad \Gamma \vdash t <: t'[v_x/x]}{\Gamma \vdash t <: \exists\, x{:}t_x.\, t'}\ \textsc{S-Witn} \qquad \frac{y{:}t_x, \Gamma \vdash t[y/x] <: t' \quad y \notin \mathit{free}(t')}{\Gamma \vdash \exists\, x{:}t_x.\, t <: t'}\ \textsc{S-Bind}$$

$$\frac{\alpha{:}k, \Gamma \vdash t_1[\alpha/\alpha_1] <: t_2[\alpha/\alpha_2] \qquad \alpha \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash \forall\, \alpha_1{:}k.\, t_1 <: \forall\, \alpha_2{:}k.\, t_2}\ \textsc{S-Poly}$$

The first rule above, S-Base, uses the entailment judgement $\Gamma \vdash_e p$ which (roughly) states that predicate p is valid (in the sense of a logical formula) when universally quantified over all variables bound in environment Γ. We give the inference rule for the entailment judgement:

$$\frac{\forall\, \theta.\, \theta \in [\![\Gamma]\!] \Rightarrow \theta(p) \hookrightarrow^* \texttt{true}}{\Gamma \vdash_e p}\ \textsc{Ent-Pred}$$

2 Preliminaries

For clarity, we distinguish between different typing judgments with a subscript. The type judgments in the underlying polymorphic lambda calculus (System F) will be denoted by $\vdash_B$ and a colon before the type. In order to speak about the base type underlying some type, we define a function that erases refinements in types:

$$\lfloor b\{x:p\}\rfloor := b, \quad \lfloor\alpha\rfloor := \alpha, \quad \lfloor x{:}t_x \to t\rfloor := \lfloor t_x\rfloor \to \lfloor t\rfloor, \quad \lfloor \exists\, x{:}t_x.\, t\rfloor := \lfloor t\rfloor, \quad \text{and} \quad \lfloor \forall\, \alpha{:}k.\, t\rfloor := \forall\, \alpha{:}k.\, \lfloor t\rfloor$$

We start our development of the meta-theory by giving a definition of *type denotations.* Roughly speaking, the denotation of a type t without type variables is the class of value terms v with the correct underlying base type such that this term satisfies the refinement predicates that appear within the structure of t. We formalize this notion with a recursive definition:

$$\begin{aligned}
[\![b]\!] &:= \{v \mid \varnothing \vdash_B v : b\}\\
[\![b\{x:p\}]\!] &:= \{v \mid (\varnothing \vdash_B v : b) \wedge (p[v/x] \hookrightarrow^* \texttt{true})\}\\
[\![x{:}t_x \to t]\!] &:= \{v \mid (\varnothing \vdash_B v : \lfloor t_x\rfloor \to \lfloor t\rfloor) \wedge (\forall\, v_x \in [\![t_x]\!].\, v\ v_x \hookrightarrow^* v' \,\text{such that}\, v' \in [\![t[v_x/x]]\!])\}\\
[\![\exists\, x{:}t_x.\, t]\!] &:= \{v \mid (\varnothing \vdash_B v : \lfloor t\rfloor) \wedge (\exists\, v_x \in [\![t_x]\!].\, v \in [\![t[v_x/x]]\!])\}\\
[\![\forall\, \alpha{:}k.\, t]\!] &:= \{v \mid (\varnothing \vdash_B v : \forall\, \alpha{:}k.\, \lfloor t\rfloor) \wedge (\forall\, t_\alpha.\, (\varnothing \vdash_w t_\alpha : k) \Rightarrow v\,[t_\alpha] \hookrightarrow^* v' \,\text{such that}\, v' \in [\![t[t_\alpha/\alpha]]\!])\}
\end{aligned}$$

The denotation of a type variable α is undefined.

We also have the concept of the denotation of an environment Γ; we intuitively define this to be the set of all sequences of value bindings for the term variables and type bindings for the type variables in Γ such that the values respect the denotations of the types of the corresponding variables. A closing substitution is just a sequence of value bindings to variables:

$$\theta = (x_1 \mapsto v_1,\ \ldots,\ x_n \mapsto v_n, \alpha_1 \mapsto t_1,\ \ldots,\ \alpha_m \mapsto t_m) \quad \text{with all}\, x_i, \alpha_j \,\text{distinct}$$

We use the shorthand $\theta(x)$ to refer to v_i if $x = x_i$ and we use $\theta(\alpha)$ to refer to t_j if $\alpha = \alpha_j$. We define $\theta(t)$ to be the type derived from t by substituting for all variables in θ:

$$\theta(t) := t[v_1/x_1]\cdots[v_n/x_n][t_1/\alpha_1]\cdots[t_m/\alpha_m]$$

Then we can formally define the denotation of an environment:

$$\begin{aligned}
[\![\Gamma]\!] := \{\theta =&(x_1 \mapsto v_1, \ldots, x_n \mapsto v_n, \alpha_1 \mapsto t_1,\ \ldots,\ \alpha_m \mapsto t_m)\\
&\mid \forall\,(x:t) \in \Gamma.\, \theta(x) \in [\![\theta(t)]\!] \ \wedge\ \forall\,(\alpha:k) \in \Gamma.\, \varnothing \vdash_w \theta(\alpha) : k\}.
\end{aligned}$$

For each built-in primitive constant or function c we define $ty(c)$ to include the most specific possible refinement type for c.

$$\begin{aligned}
ty(\texttt{true}) &:= \mathsf{Bool}\{x : x = \texttt{true}\}\\
ty(\texttt{false}) &:= \mathsf{Bool}\{x : x = \texttt{false}\}\\
ty(3) &:= \mathsf{Int}\{x : x = 3\}\\
ty(n) &:= \mathsf{Int}\{x : x = n\}\\
ty(\wedge) &:= x{:}\mathsf{Bool} \to y{:}\mathsf{Bool} \to \mathsf{Bool}\{v : v = x \wedge y\}
\end{aligned}$$

$$\begin{aligned}
ty(\neg) &:= x{:}\mathsf{Bool} \to \mathsf{Bool}\{y : y = \neg x\}\\
ty(\leq) &:= x{:}\mathsf{Int} \to y{:}\mathsf{Int} \to \mathsf{Bool}\{v : v = (x \leq y)\}\\
ty(m\leq) &:= n{:}\mathsf{Int} \to \mathsf{Bool}\{v : v = (m \leq n)\}\\
ty(=) &:= x{:}\alpha \to y{:}\alpha \to \mathsf{Bool}\{v : v = (x = y)\}
\end{aligned}$$

and similarly for the others Note that we use $m\leq$ to represent an arbitrary member of the infinite family of primitives $0\leq$, $1\leq$, $2\leq, \ldots$. Then by the definitions above we get our primitive typing lemma:

Lemma 1. *(Primitive Typing) For every primitive c,*

1. *$\varnothing \vdash c : ty(c)$.*
2. *If $ty(c) = b\{x : p\}$, then $\varnothing \vdash_w ty(c) : B$, $c \in [\![ty(c)]\!]$ and for all c' such that $c' \neq c$, $c' \notin [\![ty(c)]\!]$.*
3. *If $ty(c) = x{:}t_x \to t$, then $\varnothing \vdash_w ty(c) : *$ and for each $v \in [\![t_x]\!]$, $\delta(c, v)$ is defined and we have both $\varnothing \vdash \delta(c, v) : t[v/x]$ and $\delta(c, v) \in [\![t[v/x]]\!]$. Thus $c \in [\![ty(c)]\!]$.*

3 Meta-theory

In this section, we seek to prove the operational soundness of our language λ_1. We begin by stating several standard properties and proving some basic facts used later on.

Lemma 2. *Values are closed under substitution of variables for values. If v is a value and $x \in \mathrm{free}(v)$ then for any value v_x, we have that $v[v_x/x]$ is also a value.*

Lemma 3. *The operational semantics of λ_2 are deterministic: For every expression e there exists at most one term e' such that $e \hookrightarrow e'$. (Moreover there exists at most one value term v such that $e \hookrightarrow^* v$.)*

Lemma 4. *(Weakenings of Judgments) For any environments Γ, Γ' and $x \notin dom(\Gamma', \Gamma)$:*

1. *If $\Gamma', \Gamma \vdash e : t$ then $\Gamma', x{:}t_x, \Gamma \vdash e : t$.*
2. *If $\Gamma', \Gamma \vdash s <: t$ then $\Gamma', x{:}t_x, \Gamma \vdash s <: t$.*
3. *If $\Gamma', \Gamma \vdash_e p$ then $\Gamma', x{:}t_x, \Gamma \vdash_e p$.*
4. *If $\Gamma', \Gamma \vdash_w t : k$ then $\Gamma', x{:}t_x, \Gamma \vdash_w t : k$.*

Proof. The proof is by mutual induction on the derivation trees of each type of judgment. □

Lemma 5. *(Reflexivity of <:) If $\Gamma \vdash_w t : k$ and t is not a type variable then $\Gamma \vdash t <: t$.*

TODO: Write up the other two cases from my notes

Proof. We proceed by induction of the structure of the derivation of $\Gamma \vdash_w t : k$. *We could dispense with the hypothesis that $t \not\equiv \alpha$ by either making α a basic type or adding another subtyping rule that states $(\alpha : k) \in \Gamma \Rightarrow \Gamma \vdash \alpha <: \alpha$*

Case WF-Refn: In the base case, we have $t \equiv bx : p$ and $k \equiv B$. By inversion, we have for some $y \notin \text{dom}(\Gamma)$, the judgment $y{:}b, \lfloor\Gamma\rfloor \vdash_B p[y/x] : \mathsf{Bool}$. Let $\theta \in [\![y{:}b\{x:p\}, \Gamma]\!]$. Then we have that $\theta(y) \in [\![\theta(b\{x:p\})]\!] = [\![b\{x:\theta(p)\}]\!]$, and so $\theta(p)[\theta(y)/x] \hookrightarrow^* \texttt{true}$. But $\theta(p)[\theta(y)/x] = \theta(p[y/x])$, and so by rule Ent-Pred, $y{:}b\{x:p\}, \Gamma \vdash_e p[y/x]$. By S-Base, we conclude $\Gamma \vdash b\{x:p\} <: b\{x:p\}$.

Case WF-Kind: We have $\Gamma \vdash_w t : *$ and by inversion we have $\Gamma \vdash_w t : B$. By induction, we get $\Gamma \vdash t <: t$ as desired.

Case WF-Func:

Case WF-Exis:

Case WF-Poly: We have $t \equiv \Lambda\alpha{:}k.t'$ and $\Gamma \vdash_w \Lambda\alpha{:}k.t' : *$. By inversion we have for some $\alpha' \notin \text{dom}(\Gamma)$, $\alpha'{:}k, \Gamma \vdash_w t'[\alpha'/\alpha] : k_t$. By induction, we have $\alpha'{:}k, \Gamma \vdash t'[\alpha'/\alpha] <: t'[\alpha'/\alpha]$. By rule S-Poly we conclude that $\Gamma \vdash \forall\,\alpha{:}k.\,t' <: \forall\,\alpha{:}k.\,t'$. □

Our proof of the soundness theorems begin with several helping lemmas.

Lemma 6. *(Type Denotations) Our typing and subtyping relations are sound with respect to the denotational semantics of our types:*
1. If $\Gamma \vdash t_1 <: t_2$ then $\forall\theta.\theta \in [\![\Gamma]\!] \Rightarrow [\![\theta(t_1)]\!] \subseteq [\![\theta(t_2)]\!]$.
2. If $\Gamma \vdash e : t$, then $\forall\theta.\theta \in [\![\Gamma]\!] \Rightarrow \theta(e) \hookrightarrow^* v'$ such that $v' \in [\![\theta(t)]\!]$.

The proof is by mutual induction on the derivation trees of the respective subtyping and typing judgements. The need for mutual induction contrasts with Lemma 4 of [VSJ+14] and comes from the appearance of the typing judgement $\Gamma \vdash v_x : t_x$ in the antecedent of rule S-Witn.

Proof. (1) Suppose $\Gamma \vdash t_1 <: t_2$. We proceed by induction on the derivation tree of the subtyping relation.

Case Sub-Base: We have that $\Gamma \vdash b\{x_1 : p_1\} <: b\{x_2 : p_2\}$ where $t_1 \equiv b\{x_1 : p_1\}$ and $t_2 \equiv b\{x_2 : p_2\}$. By inversion, for some $y \notin \text{dom}(\Gamma)$ we have

$$y{:}b\{x_1 : p_1\}, \Gamma \vdash_e p_2[y/x_2].$$

By inversion of Ent-Pred we have

$$\forall\,\theta'.\,\theta' \in [\![y{:}b\{x_1 : p_1\}, \Gamma]\!] \Rightarrow \theta'(p_2[y/x_2]) \hookrightarrow^* \texttt{true}. \tag{1}$$

We need to show $\forall\theta.\ \theta \in [\![\Gamma]\!] \Rightarrow [\![\theta(b\{x_1 : p_1\})]\!] \subseteq [\![\theta(b\{x_2 : p_2\})]\!]$. Equivalently,

$$\forall\theta.\,\theta \in [\![\Gamma]\!] \Rightarrow \{v \mid \varnothing \vdash_B v : b \ \wedge\ (\theta(p_1[v/x_1]) \hookrightarrow^* \texttt{true})\} \tag{2}$$

$$\subseteq \{v \mid \varnothing \vdash_B v : b \ \wedge\ (\theta(p_2[v/x_2]) \hookrightarrow^* \texttt{true})\} \tag{3}$$

Let $\theta \in [\![\Gamma]\!]$ be a closing substitution and let v a term in $[\![\theta(t_1)]\!]$. Then $\theta(t_1) = b\{x_1 : \theta(p_1)\}$ and $\theta(p_1[v/x_1]) \hookrightarrow^* \texttt{true}$. Let $\theta' = (y \mapsto v, \theta) \in [\![y{:}b\{x_1 : p_1\}, \Gamma]\!]$. By (1) we have $\theta'(p_2[y/x_2]) \hookrightarrow^* \texttt{true}$ and $\theta'(p_2[y/x_2]) = \theta(p_2[y/x_2][v/y]) = \theta(p_2[v/x_2])$, which proves $v \in [\![\theta(t_2)]\!]$.

Case Sub-Func: We have that $\Gamma \vdash x_1{:}s_1 \to t_1' <: x_2{:}s_2 \to t_2'$ where $t_1 \equiv x_1{:}s_1 \to t_1'$ and $t_2 \equiv x_2{:}s_2 \to t_2'$. By inversion of this rule, for some $y \notin \text{dom}(\Gamma)$,

$$\Gamma \vdash s_2 <: s_1 \quad \text{and} \quad y{:}s_2, \Gamma \vdash t_1'[y/x_1] <: t_2'[y/x_2]$$

By the inductive hypothesis,

$$\forall\theta.\, \theta \in [\![\Gamma]\!] \Rightarrow [\![\theta(s_2)]\!] \subseteq [\![\theta(s_1)]\!]$$

and

$$\forall\theta'.\, \theta' \in [\![y{:}s_2, \Gamma]\!] \Rightarrow [\![\theta'(t'_1[y/x_1])]\!] \subseteq [\![\theta'(t'_2[y/x_2])]\!] \tag{4}$$

We need to show $\forall\theta.\ \theta \in [\![\Gamma]\!] \Rightarrow [\![\theta(x_1{:}s_1 \to t'_1)]\!] \subseteq [\![\theta(x_2{:}s_2 \to t'_2)]\!]$. Equivalently,

$$\forall\theta.\, \theta \in [\![\Gamma]\!] \Rightarrow \{v \mid \varnothing \vdash_B v : \lfloor s_1 \rfloor \to \lfloor t'_1 \rfloor \ \wedge\ (\forall\, v' \in [\![\theta(s_1)]\!].\, v\, v' \hookrightarrow^* v^* \in [\![\theta(t'_1)[v'/x_1]]\!])\} \tag{5}$$

$$\subseteq \{v \mid \varnothing \vdash_B v : \lfloor s_2 \rfloor \to \lfloor t'_2 \rfloor \ \wedge\ (\forall\, v' \in [\![\theta(s_2)]\!].\, v\, v' \hookrightarrow^* v^* \in [\![\theta(t'_2)[v'/x_2]]\!])\} \tag{6}$$

Fix $\theta \in [\![\Gamma]\!]$ and let v be a term in set (5) and let $v' \in [\![\theta(s_2)]\!]$. Then by induction, $v' \in [\![\theta(s_1)]\!]$. So there exists a value v^* such that $(v\, v') \hookrightarrow^* v^*$ and $v^* \in [\![(\theta(t'_1)[v'/x_1]]\!]$. Let $\theta' = (y \mapsto v', \theta)$. From (4) we also have that $[\![\theta'(t'_1[y/x_1])]\!] \subseteq [\![\theta'(t'_2[y/x_2])]\!]$. But $\theta'(t'_1[y/x_1]) = \theta(t'_1[y/x_1])[v'/y] = \theta(t'_1)[v'/x_1]$ and $\theta'(t'_2[y/x_2]) = \theta(t'_2)[v'/x_2]$. Therefore $v^* \in [\![\theta(t'_1)[v'/x_2]]\!] \subseteq [\![\theta(t'_2)[v'/x_2]]\!]$ and so v is in set (6) as desired.

Case Sub-Witn: We have that $\Gamma \vdash t_1 <: \exists\, x{:}t_x.\, t'_2$ where $t_2 \equiv \exists\, x{:}t_x.\, t'_2$. By inversion, there exists some value term v_x such that

$$\Gamma \vdash v_x : t_x \quad \text{and} \quad \Gamma \vdash t_1 <: t'_2[v_x/x].$$

By the inductive hypothesis, we have

$$\forall\, \theta.\, \theta \in [\![\Gamma]\!] \Rightarrow [\![\theta(t_1)]\!] \subseteq [\![\theta(t'_2[v_x/x])]\!] \tag{7}$$

and by mutual induction we also have that there exists a value v' such that

$$\forall\, \theta.\, \theta \in [\![\Gamma]\!] \Rightarrow \theta(v_x) \hookrightarrow^* v' \in [\![\theta(t_x)]\!];$$

values are closed under substitution and cannot be reduced further, so we must have that $v' = \theta(v_x)$. We need to show that $\forall\, \theta$, if $\theta \in [\![\Gamma]\!]$, then $[\![\theta(t_1)]\!] \subseteq [\![\theta(\exists\, x{:}t_x.\, t'_2)]\!]$. Fix some $\theta \in [\![\Gamma]\!]$. Then

$$[\![\theta(\exists\, x{:}t_x.\, t'_2)]\!] = \{v \mid \varnothing \vdash_B v : \lfloor \theta(t'_2) \rfloor \ \wedge\ (\exists\, v' \in [\![\theta(t_x)]\!].\, v \in [\![\theta(t'_2)[v'/x]]\!])\} \tag{8}$$

because $\theta(\exists\, x{:}t_x.\, t'_2) = \exists\, x{:}\theta(t_x).\, \theta(t'_2)$. Let $v \in [\![\theta(t_1)]\!]$ and let $v' = \theta(v_x) \in [\![\theta(t_x)]\!]$ be as above. Then by (7), $v \in [\![\theta(t'_2[v_x/x])]\!] = [\![\theta(t'_2)[\theta(v_x)/x]]\!] = [\![\theta(t'_2)[v'/x]]\!]$. By 7 and by definition of the denotation of a type, $\varnothing \vdash_B v : \lfloor \theta(t'_2[v_x/x]) \rfloor = \lfloor \theta(t'_2) \rfloor$. Therefore v is in the right hand side of (8).

Case Sub-Bind: We have that $\Gamma \vdash \exists\, x{:}t_x.\, t'_1 <: t_2$ where $t_1 \equiv \exists\, x{:}t_x.\, t'_1$. By inversion we have for some $y \notin \mathrm{dom}(\Gamma)$

$$y{:}t_x, \Gamma \vdash t'_1[y/x] <: t_2 \quad \text{and} \quad y \notin \mathit{free}(t_2).$$

By the inductive hypothesis, we have

$$\forall\, \theta'.\, \theta' \in [\![y{:}t_x, \Gamma]\!] \Rightarrow [\![\theta'(t'_1[y/x])]\!] \subseteq [\![\theta'(t_2)]\!]. \tag{9}$$

We need to show that for every $\theta \in [\![\Gamma]\!]$ that it holds that $[\![\theta(\exists\, x{:}t_x.\, t'_1)]\!] \subseteq [\![\theta(t_2)]\!]$. Fix some $\theta \in [\![\Gamma]\!]$ and let $v \in [\![\theta(\exists\, x{:}t_x.\, t'_1)]\!]$. By definition, $\theta(\exists\, x{:}t_x.\, t'_1) = \exists\, x{:}\theta(t_x).\, \theta(t'_1)$ so

$$[\![\theta(\exists\, x{:}t_x.\, t'_1)]\!] = \{v \mid \varnothing \vdash_B v : \lfloor \theta(t'_1) \rfloor \ \wedge\ (\exists\, v' \in [\![\theta(t_x)]\!].\, v \in [\![\theta(t'_1)[v'/x]]\!])\}. \tag{10}$$

Take v' as in (10) and let $\theta' = (y \mapsto v', \theta)$. We note that $\theta' \in [\![y{:}t_x, \Gamma]\!]$ because $\theta'(y) = v' \in [\![\theta(t_x)]\!] = [\![\theta'(t_x)]\!]$ where the last equality follows from the fact that x cannot appear free in t_x. Then $v \in [\![\theta(t'_1)[v'/x]]\!] = [\![\theta(t'_1[y/x])[v'/y]]\!] = [\![\theta'(t'_1[y/x])]\!]$, so from (9) we can conclude $v \in [\![\theta'(t_2)]\!] = [\![\theta(t_2)]\!]$ because y does not appear free in t_2 so $\theta'(t_2) = \theta(t_2)$.

Case Sub-Poly: We have that $\Gamma \vdash \forall\, \alpha_1{:}k.\, t'_1 \;<:\; \forall\, \alpha_2{:}k.\, t'_2$, where $t_1 \equiv \forall\, \alpha_1{:}k.\, t'_1$ and $t_2 \equiv \forall\, \alpha_2{:}k.\, t'_2$. By inversion, for some $\alpha \notin \mathrm{dom}(\Gamma)$, $\alpha{:}k, \Gamma \vdash t_1[\alpha/\alpha_1] <: t_2[\alpha/\alpha_2]$. By the inductive hypothesis, we have that for all $\theta' \in [\![\alpha{:}k, \Gamma]\!]$ we have $[\![\theta'(t_1[\alpha/\alpha_1])]\!] \subseteq [\![\theta'(t_2[\alpha/\alpha_2])]\!]$. We need to show

$$\forall \theta.\ \theta \in [\![\Gamma]\!] \Rightarrow [\![\theta(\forall\, \alpha_1{:}k.\, t_1)]\!] \subseteq [\![\theta(\forall\, \alpha_2{:}k.\, t_2)]\!].$$

Let $\theta \in [\![\Gamma]\!]$ arbitrary and let $v \in [\![\theta(\forall\, \alpha_1{:}k.\, t_1)]\!] = [\![\forall\, \alpha_1{:}k.\, \theta(t_1)]\!]$. Then $\varnothing \vdash_B v : \forall\, \alpha_1{:}k.\, \lfloor\theta(t_1)\rfloor$. By Lemma **??**, we also have $\varnothing \vdash_B v : \forall\, \alpha_2{:}k.\, \lfloor\theta(t_2)\rfloor$ because $\lfloor\theta(t_1)\rfloor = \lfloor\theta(t_1[\alpha/\alpha_1]\rfloor = \lfloor\theta(t_2[\alpha/\alpha_2])\rfloor = \lfloor\theta(t_2)\rfloor$. Now let t_α be a type such that $\varnothing \vdash_w t_\alpha : k$. Then we have that there exists a value v' such that $v\,[t_\alpha] \hookrightarrow^* v'$ and $v' \in [\![\theta(t_1)[t_\alpha/\alpha_1]]\!] = [\![\theta(t_1[\alpha/\alpha_1])[t_\alpha/\alpha]]\!]$. Let $\theta' = (\alpha \mapsto t_\alpha, \theta)$. Then $v' \in [\![\theta'(t_1[\alpha/\alpha_1])]\!]$ and by induction we have

$$v' \in [\![\theta'(t_2[\alpha/\alpha_2])]\!] = [\![\theta(t_2[\alpha/\alpha_2])[t_\alpha/\alpha]]\!] = [\![\theta(t_2)[t_\alpha/\alpha_2]]\!].$$

This proves that $v \in [\![\forall\, \alpha_2{:}k.\, \theta(t_2)]\!] = [\![\theta(\forall\, \alpha_2{:}k.\, t_2)]\!]$ as desired.

(2) Suppose $\Gamma \vdash e : t$. We proceed by induction on the derivation tree of the typing relation.

Case T-Prim: We have $\Gamma \vdash e : t$ where $e \equiv c$, a built-in primitive function or constant. By inversion, $ty(c) = t$. Let $\theta \in [\![\Gamma]\!]$. In one case $t \equiv b\{x : p\}$; then by Lemma 1 on constants, $\theta(c) = c \in [\![ty(c)]\!] = [\![\theta(ty(c))]\!]$. In the other case, $ty(c) \equiv x{:}t_x \to t'$; by Lemma 1, $c\ v_x \hookrightarrow \delta(c, v_x) \in [\![t'[v_x/x]]\!]$ for any $v_x \in [\![t_x]\!]$. There are no free variables in c or t so $\theta(c)$ is a value and $\theta(c) = c \in [\![ty(c)]\!] = [\![\theta(ty(c))]\!]$.

Case T-Var: We have $\Gamma \vdash e : t$ where $e \equiv x$ and $t \equiv \mathrm{self}(t', x)$. By inversion, $(x{:}t') \in \Gamma$. Then for any $\theta \in [\![\Gamma]\!]$, we have by definition $\theta(x) \in [\![\theta(t')]\!]$. If $\mathrm{self}(t', x) = t'$ then we are done; otherwise $t' \equiv b\{z : p\}$ and $\mathrm{self}(t', x) = b\{z : p \wedge z = x\}$. Then $\theta(\mathrm{self}(t', x)) = b\{z : \theta(p) \wedge z = \theta(x)\}$. We have $\varnothing \vdash_B \theta(x) : \lfloor\mathrm{self}(t', x)\rfloor$ because $\lfloor\mathrm{self}(t', x)\rfloor = \lfloor t'\rfloor$. Finally, we have $\theta(p \wedge z = x)[\theta(x)/z] = \theta(p)[\theta(x)/z] \wedge \theta(x) = \theta(x) \hookrightarrow^* \texttt{true}$ because $\theta(p)[\theta(x)/z] \hookrightarrow^* \texttt{true}$ from $\theta(x) \in [\![b\{z : \theta(p)\}]\!]$. Therefore $\theta(x) \in [\![\theta(\mathrm{self}(t', x))]\!]$, as desired.

Case T-AbsT. We have $\Gamma \vdash e : t$ where $e \equiv \Lambda\alpha{:}k.e'$ and $t \equiv \forall\, \alpha{:}k.\, t'$. By inversion we have $\alpha{:}k, \Gamma \vdash e' : t'$, and by the inductive hypothesis,

$$\forall\, \theta'.\, \theta' \in [\![\alpha{:}k, \Gamma]\!] \Rightarrow \theta'(e') \in [\![\theta'(t')]\!]. \tag{11}$$

Let $\theta \in [\![\Gamma]\!]$ arbitrary and let t_α be a type such that $\varnothing \vdash_w t_\alpha : k$. Then we have $\theta' := (\alpha \mapsto t_\alpha, \theta)$. Then from (11), we have

$$\theta(e')[t_\alpha/\alpha] = \theta'(e') \in [\![\theta'(t')]\!] = [\![\theta(t')[t_\alpha/\alpha]]\!].$$

By the closure of values under substitution, $\theta(e')[t_\alpha/\alpha]$ is itself a value and by rule E-AppTAbs we have that $\theta(\Lambda\alpha{:}k.e')\,[t_\alpha] \hookrightarrow^* \theta(e')[t_\alpha/\alpha]$, which proves that $\theta(e) \in [\![\theta(\forall\, \alpha{:}k.\, t')]\!]$.

Case T-App: We have $\Gamma \vdash e : t$ where $e \equiv e'\ e_x$ and $t \equiv \exists\, x{:}t_x.\, t'$. By inversion, $\Gamma \vdash e' : x{:}t_x \to t'$ and $\Gamma \vdash e_x : t_x$. By the inductive hypothesis we have both

$$\forall \theta.\, \theta \in [\![\Gamma]\!] \Rightarrow \theta(e') \in [\![\theta(x{:}t_x \to t')]\!] \tag{12}$$

and

$$\forall \theta.\, \theta \in [\![\Gamma]\!] \Rightarrow \theta(e_x) \in [\![\theta(t_x)]\!]. \tag{13}$$

First suppose,

Next suppose that $\theta(e_x)$ does not evaluate to a value. Then $\theta(e')\ \theta(e_x)$ cannot evaluate to a value ever (because the only rule we could ever apply is E-App1 and no expression of the form $e_1'\ e_2'$ is ever a value)

From (14), we have that for all $\theta \in [\![\Gamma]\!]$, $\theta(e')\ \theta(e_x) \in [\![\theta(t')[\theta(e_x)/x]]\!]$. Thus

$$\theta(e) = \theta(e')\ \theta(e_x) \in [\![\exists\, x{:}\theta(t_x).\, \theta(t')]\!] = [\![\theta(\exists\, x{:}t_x.\, t')]\!]. \tag{14}$$

Case T-Abs: We have $\Gamma \vdash e : t$ where $e \equiv \lambda x.e'$ and $t \equiv x{:}t_x \to t'$. By inversion, $\Gamma, x{:}t_x \vdash e' : t'$ and by the inductive hypothesis,

$$\forall \theta'.\, \theta' \in [\![\Gamma, x{:}t_x]\!] \Rightarrow \theta'(e') \in [\![\theta'(t')]\!]. \tag{15}$$

Let $\theta \in [\![\Gamma]\!]$ and let $v_x \in [\![\theta(t_x)]\!]$ a value. Then let

$$\theta' := (\theta, x \mapsto v_x) \in [\![\Gamma, x{:}t_x]\!].$$

Then from (12),

$$\theta(e')[v_x/x] = \theta'(e') \in [\![\theta'(t')]\!] = [\![\theta(t')[v_x/x]]\!]. \tag{16}$$

We need to show that for every $\theta \in [\![\Gamma]\!]$, it holds that

$$\begin{aligned}\theta(e) &\in [\![\theta(x{:}t_x \to t')]\!] = [\![x{:}\theta(t_x) \to \theta(t')]\!] \\ &= \{\hat{e} \mid (\varnothing \vdash_B \hat{e} : \lfloor t_x \rfloor \to \lfloor t' \rfloor) \wedge (\forall\, \hat{v}_x \in [\![\theta(t_x)]\!].\ \hat{e}\ \hat{v}_x \in [\![\theta(t')[\hat{v}_x/x]]\!])\}\end{aligned}$$

We have $\varnothing \vdash_B \theta(e) : \lfloor t_x \rfloor \to \lfloor t' \rfloor$ because substitutions do not affect bare types, only the refinement predicates. We have $\theta(e)\ v_x = (\lambda x.\theta(e'))\ v_x$. Then $\theta(e)\ v_x \hookrightarrow \theta(e')[v_x/x] \in [\![\theta(t')[v_x/x]]\!]$. By Lemma **??**, we conclude that $\theta(e)\ v_x \in [\![\theta(t')[v_x/x]]\!]$ also.

Case T-Let: We have $\Gamma \vdash e : t$ where $e \equiv \texttt{let}\, x{=}e_x\, \texttt{in}\, e'$. By inversion, we have $\Gamma \vdash e_x : t_x$, $(\Gamma, x{:}t_x) \vdash e' : t$, and $\Gamma \vdash_w t$ for some t_x. Then by the inductive hypothesis we have

$$\forall \theta.\, \theta \in [\![\Gamma]\!] \Rightarrow \theta(e_x) \in [\![\theta(t_x)]\!]$$

and

$$\forall \theta'.\, \theta' \in [\![\Gamma, x{:}t_x]\!] \Rightarrow \theta'(e') \in [\![\theta'(t)]\!]. \tag{17}$$

Let $\theta \in [\![\Gamma]\!]$. There are two cases for the semantics of e_x. In the case that there exists some value v_x such that $\theta(e_x) \hookrightarrow^* v_x$, let $\theta' = (\theta, x \mapsto v_x) \in [\![\Gamma, x{:}t_x]\!]$ because we chose $\theta'(x) = v_x \in [\![\theta(t_x)]\!]$. From the operational semantics $\theta(\texttt{let}\, x{=}e_x\, \texttt{in}\, e') = \texttt{let}\, x{=}\theta(e_x)\, \texttt{in}\, \theta(e') \hookrightarrow^* \texttt{let}\, x{=}v_x\, \texttt{in}\, \theta(e') \hookrightarrow \theta(e')[v_x/x]$. Then from (17),

$$\theta(e')[v_x/x] = \theta'(e') \in [\![\theta'(t)]\!] = [\![\theta(t)[v_x/x]]\!] = [\![\theta(t)]\!],$$

where the last equality follows from the fact that the judgement $\Gamma \vdash_w t$ implies $\varnothing \vdash_w \theta(t)$ by part 3 of this lemma, which in turn implies that x cannot be free in $\theta(t)$. The above implies $\theta(e) \hookrightarrow^* \theta(e')[v_x/x]$, so by Lemma **??**, $\theta(e) \in [\![\theta(t)]\!]$.

In the second case, $\theta(e_x)$ does not reduce to any value. In that case, the only rule we can ever apply to $\theta(e) = \texttt{let}\, x{=}\theta(e_x)\, \texttt{in}\, \theta(e')$ is E-Let so $\theta(e)$ never reduces to a value, and by Lemma **??** $\theta(e) \in [\![\theta(t)]\!]$.

Case T-Ann: We have $\Gamma \vdash e : t$ where $e \equiv (e' : t)$. By inversion, $\Gamma \vdash e' : t$ and by the inductive hypothesis, $\theta(e') \in [\![\theta(t)]\!]$. By the operational semantics of type annotations, $\theta(e) = (\theta(e') : \theta(t)) \hookrightarrow \theta(e') \in [\![\theta(t)]\!]$, so we conclude that $\theta(e) \in [\![\theta(t)]\!]$ by Lemma **??**.

Case T-Sub: We have $\Gamma \vdash e : t$ and by inversion, we have $\Gamma \vdash e : s$ and $\Gamma \vdash s <: t$ for some type s. By the inductive hypothesis, $\forall \theta. \theta \in [\![\Gamma]\!] \Rightarrow \theta(e) \in [\![\theta(s)]\!]$ and by mutual induction, part 1 of the Lemma gives us that $\forall \theta. \theta \in [\![\Gamma]\!] \Rightarrow [\![\theta(s)]\!] \subseteq [\![\theta(t)]\!]$. Then we conclude that $\forall \theta. \theta \in [\![\Gamma]\!] \Rightarrow \theta(e) \in [\![\theta(t)]\!]$. □