Journey Beyond Full Abstraction
Exploring Robust Property Preservation for Secure Compilation
(Online Appendix)

Anonymous Author(s)

November 1, 2018

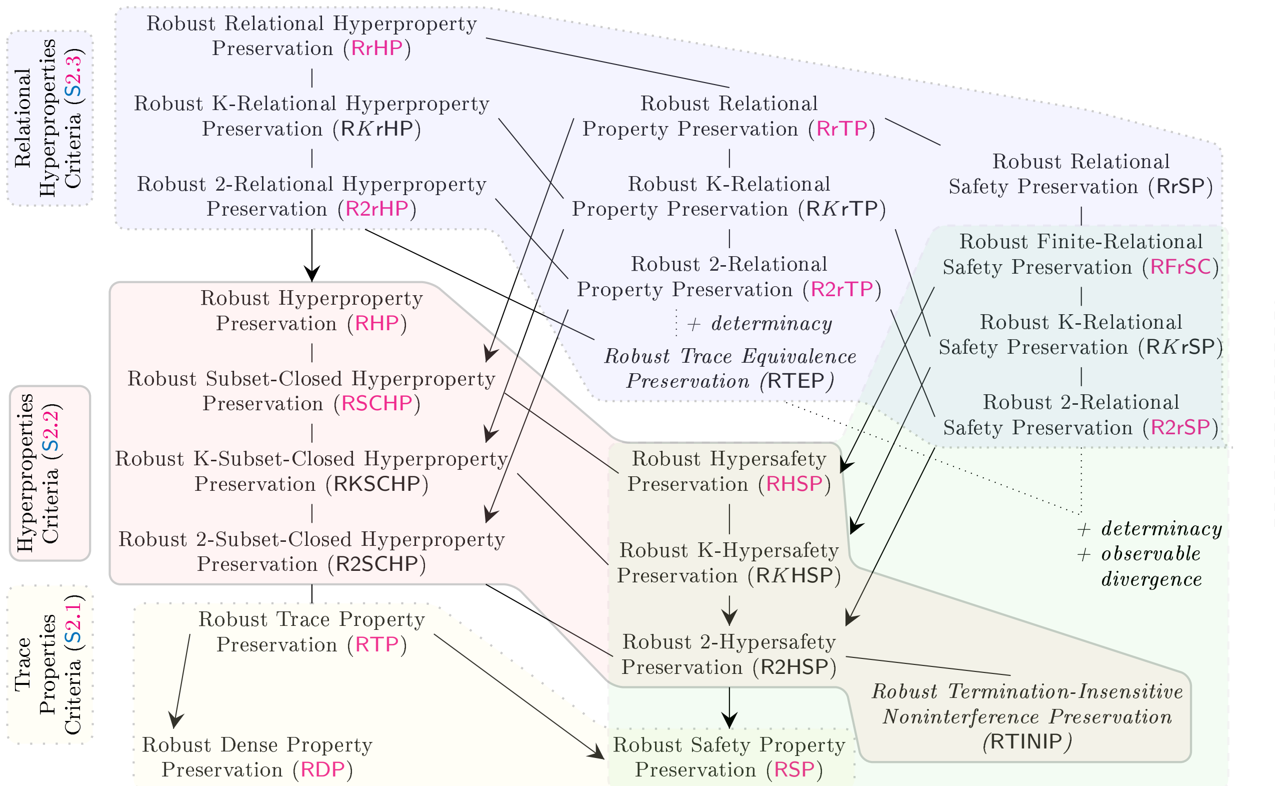
Contents

1 Notation and Background

We use blue, sans-serif font for *source* elements, **red, bold** font for *target* elements and *black*, *italic* for elements common to both languages (to avoid repeating similar definitions twice). Thus, P is a source-level program, $\mathbf{P}$ is a target-level program and P is generic notation for either a source-level or a target-level program.

Whole Programs	W
Partial Programs (Components)	P
Program Contexts	C
Events	e
Finite Trace Prefixes	$m \triangleq$
(terminated)	$e_1 \cdot \cdots \cdot e_n \bullet$
(not yet terminated)	$e_1 \cdot \cdots \cdot e_n \circ$
Traces	$t \triangleq$
(program termination)	$e_1 \cdot \cdots \cdot e_n \bullet$
(silent divergence)	$e_1 \cdot \cdots \cdot e_n \circlearrowleft$
(infinitely reactive)	$e_1 \cdot \cdots \cdot e_n \cdots$
Set of Traces	Trace
Behavioral Semantics	$W \rightsquigarrow t$
	$\mathtt{Behav}(W) = \{t \mid W \rightsquigarrow t\}$
Source Language	S
Target Language	$\mathbf{T}$
Compiler	$\cdot\downarrow$
(also denoted with)	$[\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{P} \to \mathbf{P}$
Property	$\pi \in 2^{\mathit{Trace}}$
Hyperproperty	$H \in 2^{2^{\mathit{Trace}}}$
Notation for Sets	$\widehat{x} \triangleq \{x_1, x_2, \cdots\}$
(also denoted with)	$\equiv 2^x$
(while sets of size k)	$\equiv 2^x_k$
Cardinality	$\lVert\cdot\rVert$

2 Secure Compilation Criteria



2.1 Trace Property-Based Criteria

2.1.1 Robust Trace Property Preservation

Definition 1 (RTP)**.**

$$\mathsf{RTP}: \quad \forall \pi \in 2^{Trace}.\ \forall \mathsf{P}.\ (\forall \mathsf{C_S}\ t.\ \mathsf{C_S}\left[\mathsf{P}\right] \rightsquigarrow t \Rightarrow t \in \pi) \Rightarrow (\forall \mathbf{C_T}\ t.\ \mathbf{C_T}\left[\mathsf{P}\!\downarrow\right] \rightsquigarrow t \Rightarrow t \in \pi)$$

Definition 2 (RTC)**.**

$$\mathsf{RTC}: \forall \mathsf{P}.\ \forall \mathbf{C_T}.\ \forall t.\ \mathbf{C_T}\left[\mathsf{P}\!\downarrow\right] \rightsquigarrow t \Rightarrow \exists \mathsf{C_S}.\ \mathsf{C_S}\left[\mathsf{P}\right] \rightsquigarrow t$$

Theorem 1 (RTP and RTC are equivalent)**.**

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RTP} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RTC}$$

Proof. See file Criteria.v, theorem RTC_RTP for a Coq proof. The proof is simple, but still illustrative for how such proofs work in general:

($\Rightarrow$) Let P be arbitrary. We need to show that $\forall \mathbf{C_T}.\ \forall t.\ \mathbf{C_T}\left[\mathsf{P}\!\downarrow\right] \rightsquigarrow t \Rightarrow \exists \mathsf{C_S}.\ \mathsf{C_S}\left[\mathsf{P}\right] \rightsquigarrow t$. We can directly conclude this by applying RTP to P and the property $\pi = \{t \mid \exists \mathsf{C_S}.\, \mathsf{C_S}\,[\mathsf{P}] \rightsquigarrow t\}$; for this application to be possible we need to show that $\forall \mathsf{C_S}\ t.\, \mathsf{C_S}\,[\mathsf{P}] \rightsquigarrow t \Rightarrow \exists \mathsf{C}'_\mathsf{S}.\, \mathsf{C}'_\mathsf{S}\,[\mathsf{P}] \rightsquigarrow t$, which is trivial if taking $\mathsf{C}'_\mathsf{S} = \mathsf{C_S}$.

($\Leftarrow$) Given a compilation chain that satisfies RTC and some P and π so that $\forall \mathsf{C_S}\ t.\, \mathsf{C_S}\,[\mathsf{P}] \rightsquigarrow t \Rightarrow t \in \pi$ (H) we have to show that $\forall \mathbf{C_T}\ t.\, \mathbf{C_T}\,[\mathsf{P}{\downarrow}] \rightsquigarrow t \Rightarrow t \in \pi$. Let $\mathbf{C_T}$ and t so that $\mathbf{C_T}\,[\mathsf{P}{\downarrow}] \rightsquigarrow t$, we still have to show that $t \in \pi$. We can apply RTC to obtain $\exists \mathsf{C_S}.\, \mathsf{C_S}\,[\mathsf{P}] \rightsquigarrow t$, which we can use to instantiate H to conclude that $t \in \pi$. □

2.1.2 Robust Safety Property Preservation

$$\mathit{Safety} \triangleq \left\{\pi \in 2^{\mathit{Trace}} \;\middle|\; \forall t{\notin}\pi.\ \exists m{\leq}t.\ \forall t'{\geq}m.\ t'{\notin}\pi\right\}$$

Definition 3 (RSP)**.**

$$\mathsf{RSP}: \quad \forall \pi \in \mathit{Safety}.\, \forall \mathsf{P}.\, (\forall \mathsf{C_S}\ t.\, \mathsf{C_S}\,[\mathsf{P}] \rightsquigarrow t \Rightarrow t \in \pi) \Rightarrow (\forall \mathbf{C_T}\ t.\, \mathbf{C_T}\,[\mathsf{P}{\downarrow}] \rightsquigarrow t \Rightarrow t \in \pi)$$

Definition 4 (RSC)**.**

$$\mathsf{RSC}: \quad \forall \mathsf{P}.\, \forall \mathbf{C_T}.\, \forall m.\, \mathbf{C_T}\,[\mathsf{P}{\downarrow}] \rightsquigarrow m \Rightarrow \exists \mathsf{C_S}.\, \mathsf{C_S}\,[\mathsf{P}] \rightsquigarrow m$$

Theorem 2 (RSP and RSC are equivalent)**.**

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RSP} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RSC}$$

Proof. See file Criteria.v, theorem RSC_RSP. □

2.1.3 Robust Dense Property Preservation

$$\mathit{Dense} \triangleq \left\{\pi \in 2^{\mathit{Trace}} \;\middle|\; \forall t\ \mathit{terminating}.\ t \in \pi\right\}$$

Definition 5 (RDP)**.**

$$\mathsf{RDP}: \quad \forall \pi \in \mathit{Dense}.\, \forall \mathsf{P}.\, (\forall \mathsf{C_S}\ t.\, \mathsf{C_S}\,[\mathsf{P}] \rightsquigarrow t \Rightarrow t \in \pi) \Rightarrow (\forall \mathbf{C_T}\ t.\, \mathbf{C_T}\,[\mathsf{P}{\downarrow}] \rightsquigarrow t \Rightarrow t \in \pi)$$

Definition 6 (RDC)**.**

$$\mathsf{RDC}: \quad \forall \mathsf{P}.\, \forall \mathbf{C_T}.\, \forall t\ \mathit{infinite}.\, \mathbf{C_T}\,[\mathsf{P}{\downarrow}] \rightsquigarrow t \Rightarrow \exists \mathsf{C_S}.\, \mathsf{C_S}\,[\mathsf{P}] \rightsquigarrow t$$

Theorem 3 (RDP and RDC are equivalent)**.**

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RDP} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RDC}$$

Proof. See file Criteria.v, theorem RDC_RDP. □

2.2 Hyperproperty-Based Criteria

2.2.1 Robust Hyperproperty Preservation

Definition 7 (RHP).

$$\mathsf{RHP}: \quad \forall H \in 2^{2^{\mathit{Trace}}}. \forall \mathsf{P}. (\forall \mathsf{C_S}. \mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P}]) \in H) \Rightarrow (\forall \mathbf{C_T}. \mathtt{Behav}\,(\mathbf{C_T}\,[\mathsf{P}{\downarrow}]) \in H)$$

Definition 8 (RHC).

$$\begin{aligned} \mathsf{RHC}: \quad & \forall \mathsf{P}. \forall \mathbf{C_T}. \exists \mathsf{C_S}. \mathtt{Behav}\,(\mathbf{C_T}\,[\mathsf{P}{\downarrow}]) = \mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P}]) \\ \iff & \forall \mathsf{P}. \forall \mathbf{C_T}. \exists \mathsf{C_S}. \forall t. \mathbf{C_T}\,[\mathsf{P}{\downarrow}] \rightsquigarrow t \iff \mathsf{C_S}\,[\mathsf{P}] \rightsquigarrow t \end{aligned}$$

Theorem 4 (RHP and RHC are equivalent).

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RHP} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RHC}$$

Proof. See file Criteria.v, theorem RHC_RHP. □

2.2.2 Robust Subset-Closed Hyperproperty Preservation

Definition 9 (RSCHP).

$$\mathsf{RSCHP}: \quad \forall H \in \mathit{SC}. \forall \mathsf{P}. (\forall \mathsf{C_S}. \mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P}]) \in H) \Rightarrow (\forall \mathbf{C_T}. \mathtt{Behav}\,(\mathbf{C_T}\,[\mathsf{P}{\downarrow}]) \in H)$$

Definition 10 (RSCHC).

$$\mathsf{RSCHC}: \quad \forall \mathsf{P}. \forall \mathbf{C_T}. \exists \mathsf{C_S}. \forall t. \mathbf{C_T}\,[\mathsf{P}{\downarrow}] \rightsquigarrow t \Rightarrow \mathsf{C_S}\,[\mathsf{P}] \rightsquigarrow t$$

Theorem 5 (RSCHP and RSCHC are equivalent).

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RSCHP} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RSCHC}$$

Proof. See file Criteria.v, theorem SSC_criterium. □

2.2.3 Robust Hypersafety Preservation

$$\begin{aligned} \mathit{Obs} &\triangleq 2^{\mathit{FinPref}}_{\mathit{Fin}} \\ \mathit{Hypersafety} &\triangleq \{H \mid \forall b \notin H. (\exists o \in \mathit{Obs}.\ o \leq b \wedge (\forall b' \geq o.\ b' \notin H))\} \end{aligned}$$

Definition 11 (RHSP).

$$\mathsf{RHSP}: \quad \forall H \in \mathit{Hypersafety}. \forall \mathsf{P}. (\forall \mathsf{C_S}. \mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P}]) \in H) \Rightarrow (\forall \mathbf{C_T}. \mathtt{Behav}\,(\mathbf{C_T}\,[\mathsf{P}{\downarrow}]) \in H)$$

Definition 12 (RHSC).

$$\mathsf{RHSC}: \quad \forall \mathsf{P}. \forall \mathbf{C_T}. \forall o \in \mathit{Obs}.\ o \leq \mathtt{Behav}\,(\mathbf{C_T}\,[\mathsf{P}{\downarrow}]) \Rightarrow \exists \mathsf{C_S}.\ o \leq \mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P}])$$

Theorem 6 (RHSP and RHSC are equivalent).

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RHSP} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RHSC}$$

Proof. See file Criteria.v, theorem RHSP_HSRC. □

2.2.4 Robust Hyperliveness Preservation

$$\mathit{Hyperliveness} \triangleq \{H \mid \forall o \in \mathit{Obs}.\ \exists b \geq o.\ b \in H\}$$

Definition 13 (RHLP)**.**

$$\mathsf{RHLP}: \quad \forall H \in \mathit{Hyperliveness}.\ \forall \mathsf{P}.\ (\forall \mathsf{C_S}.\ \mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P}]) \in H) \Rightarrow (\forall \mathbf{C_T}.\ \mathtt{Behav}\,(\mathbf{C_T}\,[\mathsf{P}{\downarrow}]) \in H)$$

Theorem 7 (RHP and RHLP are equivalent)**.**

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RHP} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RHLP}$$

Proof. See file Criteria.v, theorem RHLP_RHP. □

2.2.5 Robust K- and 2- Subset-Closed Hyperproperty Preservation

Definition 14 (KSC)**.**

$$KSC \triangleq \{H \mid \exists H' \in 2^{2^{\mathit{Trace}}_{\mathit{Fin}(K)}}, \forall b, b \notin H \iff \exists b' \in H', b' \subseteq b\}$$

Definition 15 (RKSCHP)**.**

$$\mathsf{RKSCHP}: \quad \forall H \in KSC.\ \forall \mathsf{P}.\ (\forall \mathsf{C_S}.\ \mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P}]) \in H) \Rightarrow (\forall \mathbf{C_T}.\ \mathtt{Behav}\,(\mathbf{C_T}\,[\mathsf{P}{\downarrow}]) \in H)$$

Definition 16 (RKSCHC)**.**

$$\begin{aligned}\mathsf{RKSCHC}: \quad & \forall \mathsf{P}, \mathbf{C_T}. \forall \widehat{t}. \|\widehat{t}\| = k. \\ & (\widehat{t} \subseteq \mathtt{Behav}\left(\mathbf{C_T}\left[[\![\mathsf{P}]\!]^{\mathsf{S}}_{\mathbf{T}}\right]\right)) \Rightarrow \exists \mathsf{C_S}. (\widehat{t} \subseteq \mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P}]))\end{aligned}$$

Theorem 8 (RKSCHP and RKSCHC are equivalent)**.**

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RKSCHP} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RKSCHC}$$

Proof. Analogous to that of Theorem 5. □

The definition of R2SCHC is analogous to Definition 15, but with $\|\widehat{t}\| = 2$. The definition of R2SCHP is analogous to Definition 16.

Theorem 9 (R2SCHP and R2SCHC are equivalent)**.**

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{R2SCHP} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{R2SCHC}$$

Proof. See file Criteria.v, theorem R2SCHP_R2SCHC. □

2.2.6 Robust K- and 2-Hypersafety Preservation

$$\begin{aligned} Obs_K &\triangleq 2^{FinPref}_{Fin(K)} \\ KHypersafety &\triangleq \{H \mid \forall b \notin H.\ (\exists o \in Obs_K.\ o \leq b \wedge (\forall b' \geq o.\ b' \notin H))\} \end{aligned}$$

Definition 17 (RKHSP)**.**

$$\mathsf{R}K\mathsf{HSP}: \quad \forall H \in KHypersafety.\ \forall \mathsf{P}.\ (\forall \mathsf{C_S}.\ \mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P}]) \in H) \Rightarrow (\forall \mathbf{C_T}.\ \mathtt{Behav}\,(\mathbf{C_T}\,[\mathsf{P}{\downarrow}]) \in H)$$

Definition 18 (RKHSC)**.**

$$\begin{aligned} \mathsf{R}K\mathsf{HSC}: \quad & \forall \mathsf{P}, \mathbf{C_T}. \forall \widehat{m}. \|\widehat{m}\| = k. \\ & (\widehat{m} \leq \mathtt{Behav}\left(\mathbf{C_T}\left[[\![\mathsf{P}]\!]^{\mathsf{S}}_{\mathbf{T}}\right]\right)) \Rightarrow (\exists \mathsf{C_S}. \widehat{m} \leq \mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P}])) \end{aligned}$$

Theorem 10 (RKHSP and RKHSC are equivalent)**.**

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{R}K\mathsf{HSP} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{R}K\mathsf{HSC}$$

Proof. Analogous to Theorem 6. □

The definition of R2HSC is analogous to Definition 15 but with $\|\widehat{m}\| = 2$. The definition of R2HSP is analogous to Definition 18.

Theorem 11 (R2HSP and R2HSC are equivalent)**.**

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{R2HSP} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{R2HSC}$$

Proof. See file Criteria.v, theorem R2HSP_R2HSC. □

2.3 Relational Hyperproperty-Based Criteria

2.3.1 Robust Relational Hyperproperty Preservation

Definition 19 (RrHP)**.**

$$\mathsf{RrHP}: \ \forall R \in 2^{(\mathsf{Progs} \to \mathbf{Behavs})}.\ (\forall \mathsf{C_S}.\ (\lambda \mathsf{P}.\ \mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P}])) \in R) \Rightarrow (\forall \mathbf{C_T}.\ (\lambda \mathsf{P}.\ \mathtt{Behav}\,(\mathbf{C_T}\,[\mathsf{P}{\downarrow}])) \in R)$$

Definition 20 (RrHC)**.**

$$\mathsf{RrHC}: \quad \forall \mathbf{C_T}.\ \exists \mathsf{C_S}.\ \forall \mathsf{P}.\ \mathtt{Behav}\,(\mathbf{C_T}\,[\mathsf{P}{\downarrow}]) = \mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P}])$$

Theorem 12 (RrHP and RrHC are equivalent)**.**

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RrHP} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RrHC}$$

Proof. See file Criteria.v, theorem RrHP_RrHC. □

2.3.2 Robust 2-Relational Hyperproperty Preservation

Definition 21 (R2rHP)**.**

$$\begin{aligned}\mathsf{R2rHP}: \quad & \forall R \in 2^{(\mathsf{Behavs}^2)}.\ \forall \mathsf{P}_1\ \mathsf{P}_2.\ (\forall \mathsf{C_S}.\ (\mathsf{Behav}\,(\mathsf{C_S}\,[\mathsf{P}_1]), \mathsf{Behav}\,(\mathsf{C_S}\,[\mathsf{P}_2])) \in R) \\ & \qquad \Rightarrow (\forall \mathbf{C_T}.\ (\mathsf{Behav}\,(\mathbf{C_T}\,[\mathsf{P}_1\!\downarrow]), \mathsf{Behav}\,(\mathbf{C_S}\,[\mathsf{P}_2\!\downarrow])) \in R)\end{aligned}$$

Definition 22 (R2rHC)**.**

$$\begin{aligned}\mathsf{R2rHC}: \quad \forall \mathsf{P}_1\ \mathsf{P}_2.\ \forall \mathbf{C_T}.\ \exists \mathsf{C_S}.\ & \mathsf{Behav}\,(\mathbf{C_T}\,[\mathsf{P}_1\!\downarrow]) = \mathsf{Behav}\,(\mathsf{C_S}\,[\mathsf{P}_1]) \wedge \\ & \mathsf{Behav}\,(\mathbf{C_T}\,[\mathsf{P}_2\!\downarrow]) = \mathsf{Behav}\,(\mathsf{C_S}\,[\mathsf{P}_2])\end{aligned}$$

Theorem 13 (R2rHP and R2rHC are equivalent)**.**

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{R2rHP} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{R2rHC}$$

Proof. See file Criteria.v, theorem R2rHP_R2rHC. □

2.3.3 Robust K-Relational Hyperproperty Preservation

To obtain the definitions of $\mathsf{R}K\mathsf{rHP}$ and $\mathsf{R}K\mathsf{rHC}$, take the definitions of R2rHP and R2rHC above and replace $\forall \mathsf{C}_1, \mathsf{C}_2$ with $\forall \mathsf{C}_1, \cdots, \mathsf{C}_\mathsf{k}$.

Theorem 14 ($\mathsf{R}K\mathsf{rHP}$ and $\mathsf{R}K\mathsf{rHC}$ are equivalent)**.**

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{R}K\mathsf{rHP} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{R}K\mathsf{rHC}$$

Proof. Analogous to Theorem 12. □

2.3.4 Robust Relational Trace Property Preservation

Definition 23 (RrTP)**.**

$$\begin{aligned}\mathsf{RrTP}: \quad & \forall R \in 2^{(\mathsf{Progs} \to \mathit{Trace})}.\ (\forall \mathsf{C_S}.\ \forall f.\ (\forall \mathsf{P}.\ \mathsf{C_S}\,[\mathsf{P}] \rightsquigarrow f(\mathsf{P})) \Rightarrow R(f)) \\ & \qquad \Rightarrow (\forall \mathbf{C_T}.\ \forall f.\ (\forall \mathsf{P}.\ \mathbf{C_T}\,[\mathsf{P}\!\downarrow] \rightsquigarrow f(\mathsf{P})) \Rightarrow R(f))\end{aligned}$$

Definition 24 (RrTC)**.**

$$\mathsf{RrTC}: \quad \forall f : \mathsf{Progs} \to \mathit{Trace}.\ \forall \mathbf{C_T}.\ (\forall \mathsf{P}.\ \mathbf{C_T}\,[\mathsf{P}\!\downarrow] \rightsquigarrow f(\mathsf{P})) \Rightarrow \exists \mathsf{C_S}.\ (\forall \mathsf{P}.\ \mathsf{C_S}\,[\mathsf{P}] \rightsquigarrow f(\mathsf{P}))$$

Theorem 15 (RrTP and RrTC are equivalent)**.**

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RrTP} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RrTC}$$

Proof. See file Criteria.v, theorem rRPP_rRC. □

2.3.5 Robust 2-Relational Trace Property Preservation

Definition 25 (R2rTP).

$$\begin{aligned}\mathsf{R2rTP}:\quad & \forall R \in 2^{(\mathit{Trace}^2)}.\ \forall \mathsf{P}_1\ \mathsf{P}_2.\ (\forall \mathsf{C_S}\ t_1\ t_2.\ (\mathsf{C_S}[\mathsf{P}_1] \rightsquigarrow t_1 \wedge \mathsf{C_S}[\mathsf{P}_2] \rightsquigarrow t_2) \Rightarrow (t_1, t_2) \in R)\\ & \Rightarrow (\forall \mathbf{C_T}\ t_1\ t_2.\ (\mathbf{C_T}[\mathsf{P}_1\downarrow] \rightsquigarrow t_1 \wedge \mathbf{C_T}[\mathsf{P}_2\downarrow] \rightsquigarrow t_2) \Rightarrow (t_1, t_2) \in R)\end{aligned}$$

Definition 26 (R2rTC).

$$\begin{aligned}\mathsf{R2rTC}:\ & \forall \mathsf{P}_1\ \mathsf{P}_2.\ \forall \mathbf{C_T}.\ \forall t_1\ t_2.\ (\mathbf{C_T}[\mathsf{P}_1\downarrow] \rightsquigarrow t_1 \wedge \mathbf{C_T}[\mathsf{P}_2\downarrow] \rightsquigarrow t_2)\\ & \Rightarrow \exists \mathsf{C_S}.\ (\mathsf{C_S}[\mathsf{P}_1] \rightsquigarrow t_1 \wedge \mathsf{C_S}[\mathsf{P}_2] \rightsquigarrow t_2)\end{aligned}$$

Theorem 16 (R2rTP and R2rTC are equivalent).

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{R2rTP} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{R2rTC}$$

Proof. See file Criteria.v, theorem r2RPP_r2RC. □

2.3.6 Robust K-Relational Trace Property Preservation

To obtain the definitions of RKrTP and $rkrtp$, take the definitions of R2rTP and R2rTC above, replace $\forall \mathsf{C}_1, \mathsf{C}_2$ with $\forall \mathsf{C}_1, \cdots, \mathsf{C}_k$., and replace the two implications/conjuncts with K instances.

Theorem 17 (RKrTP and RKrTC are equivalent).

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{R}K\mathsf{rTP} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{R}K\mathsf{rTC}$$

Proof. Analogous to Theorem 15. □

2.3.7 Robust Relational Safety Preservation

Definition 27 (RrSP).

$$\begin{aligned}\mathsf{RrTP}:\quad & \forall R \in 2^{(\mathsf{Progs} \to \mathit{FinPref})}.\ (\forall \mathsf{C_S}.\ \forall f.\ (\forall \mathsf{P}.\ \mathsf{C_S}[\mathsf{P}] \rightsquigarrow f(\mathsf{P})) \Rightarrow R(f))\\ & \Rightarrow (\forall \mathbf{C_T}.\ \forall f.\ (\forall \mathsf{P}.\ \mathbf{C_T}[\mathsf{P}\downarrow] \rightsquigarrow f(\mathsf{P})) \Rightarrow R(f))\end{aligned}$$

Definition 28 (RrSC).

$$\mathsf{RrSC}:\quad \forall f : \mathsf{Progs} \to \mathit{FinPref}.\ \forall \mathbf{C_T}.\ (\forall \mathsf{P}.\ \mathbf{C_T}[\mathsf{P}\downarrow] \rightsquigarrow f(\mathsf{P})) \Rightarrow \exists \mathsf{C_S}.\ (\forall \mathsf{P}.\ \mathsf{C_S}[\mathsf{P}] \rightsquigarrow f(\mathsf{P}))$$

Theorem 18 (RrSC and RrSP are equivalent).

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RrSC} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RrSC}$$

Proof. See file Criteria.v, theorem RrSP_RrSC. □

2.3.8 Robust Finite-Relational Safety Preservation

Definition 29 (RFrSC)**.**

$$\begin{aligned}\mathsf{RFrSC}:\ \forall K.\ \forall \mathsf{P}_1 \ldots \mathsf{P}_K.\ \forall \mathbf{C_T}.\ \forall m_1 \ldots m_K.\ (\mathbf{C_T}\left[\mathsf{P}_1\downarrow\right] \rightsquigarrow m_1 \wedge \ldots \wedge \mathbf{C_T}\left[\mathsf{P}_K\downarrow\right] \rightsquigarrow m_K)\\ \Rightarrow \exists \mathsf{C_S}.\ (\mathsf{C_S}\left[\mathsf{P}_1\right] \rightsquigarrow m_1 \wedge \ldots \wedge \mathsf{C_S}\left[\mathsf{P}_K\right] \rightsquigarrow m_K)\end{aligned}$$

Definition 30 (RFrSP)**.**

$$\begin{aligned}\mathsf{RFrSP}:\quad & \forall K, \mathsf{P}_1, \cdots, \mathsf{P}_k, R \in 2^{(FinPref^k)}.\\ & (\forall \mathsf{C_S}, m_1, \cdots, m_k, (\mathsf{C_S}\left[\mathsf{P}_1\right] \rightsquigarrow m_1 \wedge \cdots \wedge \mathsf{C_S}\left[\mathsf{P}_k\right] \rightsquigarrow m_k)\\ & \quad \Rightarrow (m_1, \cdots, m_k) \in R)\\ \Rightarrow\ & (\forall \mathbf{C_T}.(\mathbf{C_T}\left[[\![\mathsf{P}_1]\!]^{\mathsf{S}}_{\mathbf{T}}\right] \rightsquigarrow m_1 \wedge \cdots \wedge \mathbf{C_T}\left[[\![\mathsf{P}_k]\!]^{\mathsf{S}}_{\mathbf{T}}\right] \rightsquigarrow m_k)\\ & \quad \Rightarrow (m_1, \cdots, m_k) \in R)\end{aligned}$$

Theorem 19 (RFrSP and RFrSC are equivalent)**.**

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RFrSP} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{RFrSC}$$

Proof. Analogous to Theorem 20. □

2.3.9 Robust 2-Relational Safety Preservation

Definition 31 (R2rSP)**.**

$$\begin{aligned}\mathsf{R2rSP}: \forall R \in 2^{(FinPref^2)}.\ \forall \mathsf{P}_1\ \mathsf{P}_2.\ (\forall \mathsf{C_S}\ m_1\ m_2.\ (\mathsf{C_S}\left[\mathsf{P}_1\right] \rightsquigarrow m_1 \wedge \mathsf{C_S}\left[\mathsf{P}_2\right] \rightsquigarrow m_2) \Rightarrow (m_1, m_2) \in R)\\ \Rightarrow (\forall \mathbf{C_T}\ m_1\ m_2.\ (\mathbf{C_T}\left[\mathsf{P}_1\downarrow\right] \rightsquigarrow m_1 \wedge \mathbf{C_T}\left[\mathsf{P}_2\downarrow\right] \rightsquigarrow m_2) \Rightarrow (m_1, m_2) \in R)\end{aligned}$$

Definition 32 (R2rSC)**.**

$$\begin{aligned}\mathsf{R2rSC}:\ \forall \mathsf{P}_1\ \mathsf{P}_2.\ \forall \mathbf{C_T}.\ \forall m_1\ m_2.\ (\mathbf{C_T}\left[\mathsf{P}_1\downarrow\right] \rightsquigarrow m_1 \wedge \mathbf{C_T}\left[\mathsf{P}_2\downarrow\right] \rightsquigarrow m_2)\\ \Rightarrow \exists \mathsf{C_S}.\ (\mathsf{C_S}\left[\mathsf{P}_1\right] \rightsquigarrow m_1 \wedge \mathsf{C_S}\left[\mathsf{P}_2\right] \rightsquigarrow m_2)\end{aligned}$$

Theorem 20 (R2rSP and R2rSC are equivalent)**.**

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{R2rSP} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{R2rSC}$$

Proof. See file Criteria.v, R2rSP_R2rSC. □

2.3.10 Robust K-Relational Safety Preservation

For the definitions of RKrSP and RKrSC, take the definitions of R2rSP and R2rSC above, replace $\forall \mathsf{C}_1, \mathsf{C}_2$ with $\forall \mathsf{C}_1, \cdots, \mathsf{C}_k$, and replace the two implications/conjuncts with K instances.

Theorem 21 (RKrSP and RKrSC are equivalent)**.**

$$\forall [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}.\ [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{R}K\mathsf{rSP} \iff [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : \mathsf{R}K\mathsf{rSC}$$

Proof. Analogous to Theorem 20. □

3 Separation Results

3.1 RSP and RDP Do Not Imply RTP

In this section we show that the robust preservation of either all safety properties (Theorem 22) or of all dense properties (Theorem 23) is not enough to guarantee the robust preservation of all properties.

Consider an arbitrary language $\mathcal{L}$ described by a small-step semantics. Assume it is possible to write a non terminating program in $\mathcal{L}$, e.g., a program that produces some infinite trace. Assume moreover that such a program is independent from the context with which it is linked. For example consider a *while* language as our $\mathcal{L}$ and the following program P_Ω, where $n \in \mathbb{N}$:

```
WHILE true
  output(n);
END
```

Next we define a language transformer $\phi(\mathcal{L})$, which produces a new language that identical to $\mathcal{L}$, except that it bounds program executions by a certain number of steps (its *"fuel"*). In particular:

- If C is a context in $\mathcal{L}$, then for every $n \in \mathbb{N}$, (n, C) is a context in $\phi(\mathcal{L})$ with fuel n.
- Plugging in $\phi(\mathcal{L})$ is defined by $(n, C)[P]_{\phi(\mathcal{L})} \equiv (n, C[P]_{\mathcal{L}})$. Subscripts will be omitted when doing so introduces no ambiguities.
- The semantics of $\phi(\mathcal{L})$ extends the semantics of $\mathcal{L}$ as follows. Every time a step is taken the amount of fuel is decremented by one unit. If the amount fuel is 0, no steps are allowed.

Theorem 22. RSP $\not\Rightarrow$ RDP

Proof. Take $\phi(\mathcal{L})$ as source language, $\mathcal{L}$ as target, and the compiler to be the projection of contexts of $\phi(\mathcal{L})$ on their second component. We are going to show that all safety properties that are robustly satisfied in the source are also robustly satisfied in the target, but not all dense properties are preserved.

Let $S \in$ *Safety*. Assume that all safety properties are robustly preserved, i.e., that for every program P, every source context (n, C) and every trace t,

$$(n, C[P]) \rightsquigarrow t \Rightarrow t \in S$$

In addition, assume by contradiction that there exists some target context C' and trace t' such that

$$C'[P\downarrow] \rightsquigarrow t' \wedge t' \notin S$$

where $P\downarrow = P$. By definition of safety, there exists $m \leq t'$ such that every continuation t'' of m violates the property,

$$\forall t''.\ m \leq t'' \Rightarrow t'' \notin S$$

Consider the source context $(|m|, C')$ where $|m|$ is the length of m. Denote by t_m the trace that contains the events of m followed by the termination marker. Since $m \leq t_m$ we have that $t_m \notin S$. However, $(|m|, C') \rightsquigarrow t_m$, which implies that $t_m \in S$, a contradiction.

Next we show a dense property that is not robustly preserved by this compiler. Consider

$$L = \{t | \ t \ \textit{is finite} \vee t = \textit{output}(42)^\omega\}$$

Observe that L is a dense property as it includes all finite traces. Since source programs in the source can produce only finite traces, these will be in L. In the target, however, the program $P = P\downarrow$

```
WHILE true
  output(41);
END
```

is no longer forced to stop after a finite number of steps, and produces an infinite different from $\textit{output}(42)^\omega$. □

Theorem 23. $\mathsf{RDP} \not\Rightarrow \mathsf{RSP}$

Proof. Take $\mathcal{L}$ as source language, $\phi(\mathcal{L})$ as target, and the compiler to be the identity. We are going to show that all dense properties are robustly preserved but not all safety properties are robustly preserved.

Let L be a dense property. Every trace t produced by a program in the target is finite, so we may think of it as a finite trace prefix followed by a termination sign $\bullet$, and denote it by m_t. By definition of *Liveness*, m_t must have a continuation in L, but its only continuation is t, so that every trace produced by some target program is in L.

Consider now the following property

$$S = \{\textit{output}(42)^\omega\}$$

S is a safety property because for every trace $t \notin S$, t starts with a number of $\textit{output}(42)$ events, followed either by some other event $e \neq \textit{output}(42)$ or terminated by $\bullet$, i.e.,

$$\textit{output}(42)^n; e \leq t \vee \textit{output}(42)^n; \bullet \leq t$$

Here, every continuation of $\textit{output}(42)^n; e$ is different from $\textit{output}(42)^\omega$, and different from every finite trace.

Finally, consider the program $P = P\downarrow$

```
WHILE true
  output(42);
END
```

which, in the source, produces the infinite trace $\textit{output}(42)^\omega \in S$ regardless of the context. In the target, only traces of length k can be produced, which are not in S. □

Theorem 24. Neither RSP nor RDP separately imply RTP.

Proof. Consequence of Theorem 22 and Theorem 23 □

3.2 RSP Does Not Imply R2HSP

Theorem 25. There is a compiler that satisfies RSP but not R2HSP.

Proof. See Part II in `separation-results.txt`. □

3.3 RKHSP Does Not Imply R(K+1)HSP

Theorem 26. For any K, there is a compiler that satisfies RKHSP but not R(K+1)HSP.

Proof. See Part IV in `separation-results.txt`. □

3.4 Robust Non-Relational Property Preservation Does Not Imply Robust Relational Property Preservation

Theorem 27. There is a compiler that satisfies RHP but not R2rSP.

Proof. A proof sketch is provided in the paper. For the full proof see Part I in `separation-results.txt`. □

3.5 RTEP Does Not Imply Any Preservation Notion

Theorem 28. There is a compiler between two deterministic languages that satisfies RTEP, TP, SCC and CCC, but none of our robust preservation criteria.

Proof. See Part V in `separation-results.txt`. □

4 Relational Criteria and Robust Trace Equivalence Preservation

Definition 33 (Determinate Languages)**.** We say a language is determinate *iff*

$$\forall W.\ \forall t_1\ t_2. W \rightsquigarrow t_1 \land W \rightsquigarrow t_1 \Rightarrow t_1\ \mathcal{R}\ t_2$$

where

$$t_1\ \mathcal{R}\ t_2 \iff t_1 = t_2 \lor \exists m.\ \exists e_1 e_2 \in \mathit{Input}.\ e_1 \neq e_2\ \land\ m :: e_1 \leq t_1\ \land\ m :: e_2 \leq t_2$$

Intuitively, determinacy states that the programs contain no internal non-determinism: the only source of non-determinism is the inputs from the environment.

Definition 34 (Input Totality)**.** We say a language satisfies input totality *iff*

$$\forall W.\ \forall m.\ \forall e_1 e_2 \in \mathit{Input}.\ W \rightsquigarrow^* m\ ::\ e_1 \Rightarrow W \rightsquigarrow^* m\ ::\ e_2$$

Intuitively, input totality states that whenever a program receives an input from the environment, then it could have received any other input as well.

Theorem 29. $\mathsf{R2rHP} \Rightarrow \mathsf{RTEP}$.

Proof. See file Criteria.v, Theorem R2rHP_teq. □

Theorem 30. For deterministic source languages $\mathsf{R2rTP} \Rightarrow \mathsf{RTEP}$.

Proof. See file Criteria.v, Theorem r2RP_teq_preservation. □

Theorem 31. Assuming:

1. the source language is determinate
2. the target language satisfies input totality
3. for every target whole program W and for every t that is not produced by W, there exists $m \leq t$, that is produced by W and is maximal with this property.

then $\mathsf{R2rTP} \Rightarrow \mathsf{RTEP}$.

Proof. See file tep_teq.v. □

Theorem 32. Assuming:

1. the source language is determinate
2. the target language satisfies input totality
3. for every target whole program W and for every t infinite trace that does not silently diverge and is not produced by W, there exists a finite prefix m and an event e such that $m; e \leq t$, W produces m but not $m; e$.
4. target programs cannot produce silently diverging traces.

then $\mathsf{R2rSP} \Rightarrow \mathsf{RTEP}$.

Proof. See file r2RSC_teq.v, Theorem two_rRSC_teq. □

5 Unique Definition of Dense Properties in Our Trace Model

In this section we show that the class *Dense* of the dense properties is necessarily the class of the dense set in the topology on the set of all traces, whose closed sets are all and only the safety properties. This means that in our model, with the current definition of *Safety* a property is dense *iff* it contains all finite traces.

Theorem 33. Assuming

1. $\mathit{Safety} \cap \mathit{Dense} = \{\mathit{True}\}$
2. Decomposition theorem holds, i.e. that ever property can be written as intersection of a safety property and a dense one and such a decomposition is trivial for dense properties

Then the class *Dense* is the class of the dense sets in the topology defined by its closed sets being the class of all and only the safety properties.

Proof. See file TopologyTrace.v, Theorem X_dense_class □

We propose a final remark about dense sets. First recall that if a set is dense then every set including it is still dense. This means that if the topology allows for two disjoint dense sets $D_1 \cap D_2 = \varnothing$ we can write an arbitrary property π as intersection of two dense sets.

$$\pi = (D_1 \cup \pi) \cap (D_2 \cup \pi)$$

This happens for instance in the model by Clarkson *et al.*, where it is possible to write an arbitrary property as intersection of two liveness properties (that play the role of the dense sets).
In our model it is not possible to have disjoint dense sets as they must all include the set of all finite traces, so that a similar decomposition is not possible.

6 Composing Contexts Using Code Introspection or Internal Nondeterminism in the Source Language

In this section we analyze how some features of the source language can influence the diagram in Section 2.
First of all we introduce relational subset closed hyperproperties, the class of all relational hyperproperties that are downward closed on each of its arguments.

Definition 35 (2rSCH)**.** Given $R \in 2^{prop^2}$

$$R \in 2rSCH \iff \forall (b_1, b_2) \in R. \forall s_1 \subseteq b_1,\ s_2 \subseteq b_2. (s_1, s_2) \in R$$

Definition 36 (R2rSCP)**.**

$$\begin{aligned} \mathit{R2rSCP}: \quad & \forall \mathsf{P}_1 \mathsf{P}_2\ R \in \mathit{2rSCH}\ \forall \mathsf{C_s}. (\mathtt{Behav}(\mathsf{C_s}[\mathsf{P}_1]), \mathtt{Behav}(\mathsf{C_s}[\mathsf{P}_2])) \in R \\ & \forall \mathbf{C_t}. (\mathtt{Behav}(\mathbf{C_T}[\mathsf{P}_1\downarrow]), \mathtt{Behav}(\mathbf{C_T}[\mathsf{P}_1\downarrow])) \in R \end{aligned}$$

Definition 37 (R2rSCC)**.**

$$\begin{aligned} \mathit{R2rSCC}: \forall \mathsf{P}_1 \mathsf{P}_2 \mathbf{C_T}.\ \exists \mathsf{C_S}.\ & \mathtt{Behav}(\mathsf{C_s}[\mathsf{P}_1]) \subseteq (\mathtt{Behav}(\mathbf{C_T}[\mathsf{P}_1\downarrow]) \wedge \\ & \mathtt{Behav}(\mathsf{C_s}[\mathsf{P}_2]) \subseteq (\mathtt{Behav}(\mathbf{C_T}[\mathsf{P}_2\downarrow]) \end{aligned}$$

Lemma 1. R2rSCP $\iff$ R2rSCC

Proof. See file Criteria.v, Lemma two_scC. □

As usual it is possible to generalize the definition above to relation of finite or arbitrary arity.

Definition 38 (RrSCP)**.**

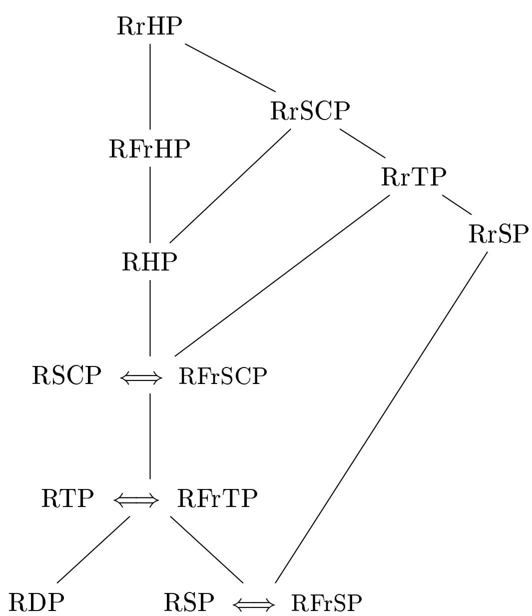
$$\begin{aligned} \mathit{RrSCP}: \ \forall R \in 2^{(\mathsf{Progs} \to SCH)}.\ & (\forall \mathsf{C_S}. (\lambda \mathsf{P}. \mathtt{Behav}(\mathsf{C_S}[\mathsf{P}])) \in R) \Rightarrow \\ & (\forall \mathbf{C_T}. (\lambda \mathsf{P}. \mathtt{Behav}(\mathbf{C_T}[\mathsf{P}\downarrow])) \in R) \end{aligned}$$

6.1 Composing Contexts using Nondeterministic Choice

Assume we have an operator $\oplus : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$ such that

$$\forall \mathsf{C}_1 \mathsf{C}_2 \mathsf{P}.\ beh((\mathsf{C}_1 \oplus \mathsf{C}_2)[\mathsf{P}]) \supseteq beh(\mathsf{C}_1\,[\mathsf{P}]) \cup beh(\mathsf{C}_2\,[\mathsf{P}])$$

In this case, the diagram in Section 2 reduces to



The file nd_ctxs.v contains proofs of $RSCP \Rightarrow R2rSCP$, $RTP \Rightarrow R2rTP$, $RSP \Rightarrow R2rSP$ that can be immediately generalized to finitary relations.

Theorem 34. $RSCP \Rightarrow RFrSCP$

Proof. Same argument used in nd_ctxs.v, RSCCP_R2SSCP. ☐

Theorem 35. $RTP \Rightarrow RFrTP$

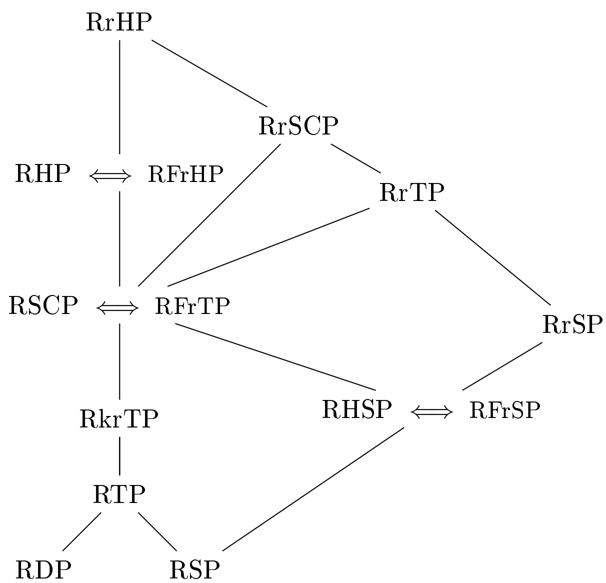
Proof. Same argument used in nd_ctxs.v, RTP_r2RTP. ☐

Theorem 36. $RSP \Rightarrow RFrSP$

Proof. Same argument used in nd_ctxs.v, RSP_r2RSP. ☐

6.2 Composing Contexts with Code Introspection

Assume source language programs can examine their own code, as is enabled by the use of *reflection* in languages like Java. Under these conditions, the diagram in Section 2 reduces to



The file reflection.v contains proofs of $R2HSP \Rightarrow R2rSP$, $RHP \Rightarrow R2rHP$, $RSCP \Rightarrow R2rSCP$ that can be immediately generalized to finitary relations.

Theorem 37. *$R2HSP \Rightarrow RFrSP$*

Proof. Same argument used in reflection.v, R2HSP_R2rSP. ☐

Theorem 38. *$RHP \Rightarrow RFrHP$*

Proof. Same argument used in reflection.v, RHP_R2rHP. ☐

Theorem 39. *$RSCP \Rightarrow RFrSCP$*

Proof. Same argument used in reflection.v, RSCP_R2rSCP. ☐

7 Safety-Like Small-Step Semantics

In this section we state the property that all small-step semantics have a certain "safety-like" behavior, in the sense that we can determine whether an infinite trace cannot be produced by a program after a finite number of steps.

Definition 39 ("Safety-like" semantics)**.** Given a language $\mathcal{L}$, it semantics is "safety-like" *iff*

$$\forall W. \forall t \text{ infinite}. W \not\rightsquigarrow t \Rightarrow \exists m. \exists e. W \rightsquigarrow m \wedge m :: e \leq t \wedge W \not\rightsquigarrow m :: e$$

Intuitively, any infinite trace that cannot not produced by a program can be explained as a finite prefix of that trace that *can* produced by the program, but after which the next event can no longer be produced by it.

As we will now show, any reasonable formalization of an arbitrary small-step semantics satisfies this property. First, we state our semantic model and its basic constituents.

Definition 40 (Small-step semantics)**.** A small-step semantics is defined in terms of the following abstract components:

- Program states are represented by *configurations*, c.
- An *initial relation* characterizes initial program states.
- A *step relation*, $c \xrightarrow{e} c'$ between pairs of states, producing an event. Its reflexive and transitive closure is denoted $\xrightarrow{e_1 \cdots e_n}^*$.
- A well-founded *order relation* on elements of a type of "measures."

Events can be either *visible* or *silent*. A configuration is *stuck* when there is no configuration it can step to; it can *loop silently* if there is an infinite sequence of silent steps starting from it.

A small-step semantics relates program configurations and the traces they produce; the relation is moreover parameterized by an element of the type of measures. In our trace model, there are four possible cases, starting from a configuration c:

- If c is stuck, the semantics produces the terminating trace $\bullet$.
- If c can loop silently, the semantics produces the silently diverging trace $\circlearrowleft$.
- If c can silently step $c \xrightarrow{\varnothing}^* c'$ to a c' while decreasing its ordering measure with respect to c, the semantics recurses on c'.
- If c can step with some visible events $c \xrightarrow{m}^* c'$, the semantics emits m and recurses on c'.

The addition of the well-founded order relation between measures is used to avoid the usual problem of infinite stuttering on silent events, which is properly captured by silent divergence. Between two visible events there must mediate a finite number of silent events. This requirement is enforced by having the ordered measure decrease when silent steps are taken (there are no restrictions on ordering between states connected by visible events). A similar device is used, for example, in the CompCert verified compiler.

The final result holds for a wide class of reasonable languages. The following determinacy condition is sufficient to prove the result.

Definition 41 (Weak determinacy)**.** Two program configurations are related if they produce the same traces under the semantics; we write $c_1 R c_2$ for this.

Under weak determinacy, if a pair of states is related and each element of this initial pair steps to another state producing the same sequence of events, the pair of final states is also related:

$$\forall c_1.\forall c_1'.\forall m.\forall c_2.\forall c_2'.c_1 R c_1' \Rightarrow c_1 \xrightarrow{m}{}^* c_2 \Rightarrow c_1' \xrightarrow{m}{}^* c_2' \Rightarrow c_2 R c_2'$$

Thus stated, the "safety-like" quality of small-step semantics follows easily.

Theorem 40. Assuming weak determinacy holds, all small-step semantics (that can be encoded by the scheme of Definition 40) are "safety-like."

Proof. See file SemanticsSafetyLike.v, theorem tgt_sem. □

8 Instances

This section presents a compiler and two proof techniques.

8.1 The Source Language L^{τ}

A list of elements $e_1, \cdots, e_n$ is indicated as $\overline{e}$, the empty list is $\varnothing$.

8.1.1 Syntax

$$
\begin{aligned}
\textit{Program}\ \mathsf{P} &::= \overline{\mathsf{I}}; \overline{\mathsf{F}} \\
\textit{Contexts}\ \mathbb{C} &::= \mathsf{e} \\
\textit{Interfaces}\ \mathsf{I} &::= \mathsf{f} : \tau \to \tau \\
\textit{Functions}\ \mathsf{F} &::= \mathsf{f}(\mathsf{x} : \tau) : \tau \mapsto \mathsf{return\ e} \\
\textit{Types}\ \tau &::= \mathsf{Bool} \mid \mathsf{Nat} \\
\textit{Operations}\ \oplus &::= + \mid - \\
\textit{Values}\ \mathsf{v} &::= \mathsf{true} \mid \mathsf{false} \mid \mathsf{n} \in \mathbb{N} \\
\textit{Expressions}\ \mathsf{e} &::= \mathsf{x} \mid \mathsf{v} \mid \mathsf{e} \oplus \mathsf{e} \mid \mathsf{let\ x} : \tau = \mathsf{e\ in\ e} \mid \mathsf{if\ e\ then\ e\ else\ e} \mid \mathsf{e} \geq \mathsf{e} \\
&\quad \mid \mathsf{call\ f\ e} \mid \mathsf{read} \mid \mathsf{write\ e} \mid \mathsf{fail} \\
\textit{Runtime Expr.}\ \mathsf{e} &::= \cdots \mid \mathsf{return\ e} \\
\textit{Eval. Ctxs.}\ \mathbb{E} &::= [\cdot] \mid \mathsf{e} \oplus \mathbb{E} \mid \mathbb{E} \oplus \mathsf{n} \mid \mathsf{let\ x} = \mathbb{E}\ \mathsf{in\ e} \mid \mathsf{if}\ \mathbb{E}\ \mathsf{then\ e\ else\ e} \mid \mathsf{e} \geq \mathbb{E} \mid \mathbb{E} \geq \mathsf{n} \\
&\quad \mid \mathsf{call\ f}\ \mathbb{E} \mid \mathsf{write}\ \mathbb{E} \mid \mathsf{return}\ \mathbb{E} \\
\textit{Substitutions}\ \rho &::= [\mathsf{v}/\mathsf{x}] \\
\textit{Prog. States}\ \Omega &::= \mathsf{P} \triangleright \mathsf{e} \mid \mathsf{fail} \\
\textit{Environments}\ \Gamma &::= \varnothing \mid \Gamma; (\mathsf{x} : \tau) \\
\textit{Labels}\ \lambda &::= \epsilon \mid \alpha \\
\textit{Actions}\ \alpha &::= \texttt{read}\ n \mid \texttt{write}\ n \mid \downarrow \mid \uparrow \mid \bot \\
\textit{Interactions}\ \gamma &::= \texttt{call}\ f\ v? \mid \texttt{ret}\ v! \\
\textit{Behaviors}\ \beta &::= \overline{\alpha} \\
\textit{Traces}\ \sigma &::= \varnothing \mid \sigma\alpha \mid \sigma\gamma
\end{aligned}
$$

8.1.2 Static Semantics

The static semantics follows these typing judgements.

$\vdash \mathsf{P}$	Program P is well-typed.
$\mathsf{P} \vdash \mathsf{F} : \tau \to \tau$	Function F has type $\tau \to \tau$ in program P.
$\Gamma \vdash \diamond$	Environment Γ is well-formed.
$\mathsf{P}; \Gamma \vdash \mathsf{e} : \tau$	Expression e has type τ in Γ and P.

$\boxed{\vdash \mathsf{P}}$

$$\frac{\mathsf{P} \equiv \overline{\mathsf{I}}; \overline{\mathsf{F}} \quad \mathsf{P} \vdash \overline{\mathsf{F}} : \overline{\tau \to \tau} \quad \mathtt{dom}\left(\overline{\mathsf{F}}\right) \subseteq \overline{\mathsf{I}}}{\vdash \mathsf{P}}\ (\mathsf{TL}^{\tau}\text{-component})$$

$\boxed{\mathsf{P} \vdash \mathsf{F} : \tau \to \tau}$

$$\frac{\mathsf{F} \equiv \mathsf{f}(\mathsf{x} : \tau) : \tau' \mapsto \mathsf{return\ e} \quad \mathsf{P}; \mathsf{x} : \tau \vdash \mathsf{e} : \tau'}{\mathsf{C} \vdash \mathsf{F} : \tau \to \tau'}\ (\mathsf{TL}^{\tau}\text{-function})$$

$\boxed{\mathsf{P}; \Gamma \vdash \mathsf{e} : \tau}$

$$\frac{\Gamma \vdash \diamond}{\mathsf{P}; \Gamma \vdash \mathsf{true} : \mathsf{Bool}}\ (\mathsf{TL}^{\tau}\text{-true}) \qquad \frac{\Gamma \vdash \diamond}{\mathsf{P}; \Gamma \vdash \mathsf{false} : \mathsf{Bool}}\ (\mathsf{TL}^{\tau}\text{-false}) \qquad \frac{\Gamma \vdash \diamond}{\mathsf{P}; \Gamma \vdash \mathsf{n} : \mathsf{Nat}}\ (\mathsf{TL}^{\tau}\text{-nat})$$

$$\frac{\mathsf{x} : \tau \in \Gamma}{\mathsf{P}; \Gamma \vdash \mathsf{x} : \tau}\ (\mathsf{TL}^{\tau}\text{-var}) \qquad \frac{\mathsf{P}; \Gamma \vdash \mathsf{e} : \mathsf{Nat} \quad \mathsf{P}; \Gamma \vdash \mathsf{e}' : \mathsf{Nat}}{\mathsf{P}; \Gamma \vdash \mathsf{e} \oplus \mathsf{e}' : \mathsf{Nat}}\ (\mathsf{TL}^{\tau}\text{-op}) \qquad \frac{\mathsf{P}; \Gamma \vdash \mathsf{e} : \mathsf{Nat} \quad \mathsf{P}; \Gamma \vdash \mathsf{e}' : \mathsf{Nat}}{\mathsf{P}; \Gamma \vdash \mathsf{e} \geq \mathsf{e}' : \mathsf{Bool}}\ (\mathsf{TL}^{\tau}\text{-geq})$$

$$\frac{\mathsf{P}; \Gamma \vdash \mathsf{e} : \tau \quad \mathsf{P}; \Gamma; \mathsf{x} : \tau \vdash \mathsf{e}' : \tau'}{\mathsf{P}; \Gamma \vdash \mathsf{let\ x} : \tau = \mathsf{e\ in\ e}' : \tau'}\ (\mathsf{TL}^{\tau}\text{-letin}) \qquad \frac{\mathsf{P}; \Gamma \vdash \mathsf{e} : \mathsf{Bool} \quad \mathsf{P}; \Gamma \vdash \mathsf{e_t} : \tau \quad \mathsf{P}; \Gamma \vdash \mathsf{e_f} : \tau}{\mathsf{P}; \Gamma \vdash \mathsf{if\ e\ then\ e_t\ else\ e_f} : \tau}\ (\mathsf{TL}^{\tau}\text{-if})$$

$$\frac{(\mathsf{f}(\mathsf{x} : \tau) : \tau' \mapsto \mathsf{return\ e} \in \overline{\mathsf{F}}) \quad \mathsf{P} \equiv \overline{\mathsf{I}}; \overline{\mathsf{F}} \quad \mathsf{P}; \Gamma \vdash \mathsf{e} : \tau}{\mathsf{P}; \Gamma \vdash \mathsf{call\ f\ e} : \tau'}\ (\mathsf{TL}^{\tau}\text{-function-call}) \qquad \frac{}{\mathsf{P}; \Gamma \vdash \mathsf{read} : \mathsf{Nat}}\ (\mathsf{TL}^{\tau}\text{-read})$$

$$\frac{\mathsf{P}; \Gamma \vdash \mathsf{e} : \mathsf{Nat}}{\mathsf{P}; \Gamma \vdash \mathsf{write\ e} : \mathsf{Nat}}\ (\mathsf{TL}^{\tau}\text{-write}) \qquad \frac{}{\mathsf{P}; \Gamma \vdash \mathsf{fail} : \tau}\ (\mathsf{TL}^{\tau}\text{-fail})$$

8.1.3 Dynamic Semantics

$\Omega \xrightarrow{\lambda} \Omega'$ Program state Ω steps to Ω' emitting action λ.

$\Omega \xRightarrow{\beta} \Omega'$ Program state Ω steps to Ω' with behavior β.

$\boxed{\mathsf{P} \triangleright \mathsf{e} \xrightarrow{\lambda} \mathsf{P} \triangleright \mathsf{e}'}$

$$\frac{n \oplus n' = n''}{\mathsf{P} \triangleright \mathsf{n} \oplus \mathsf{n}' \xrightarrow{\epsilon} \mathsf{P} \triangleright \mathsf{n}''}\ (\mathsf{EL}^{\tau}\text{-op}) \qquad \frac{n \geq n'}{\mathsf{P} \triangleright \mathsf{n} \geq \mathsf{n}' \xrightarrow{\epsilon} \mathsf{P} \triangleright \mathsf{true}}\ (\mathsf{EL}^{\tau}\text{-geq-true})$$

$$\frac{n < n'}{\mathsf{P} \triangleright \mathsf{n} \geq \mathsf{n}' \xrightarrow{\epsilon} \mathsf{P} \triangleright \mathsf{false}}\ (\mathsf{EL}^{\tau}\text{-geq-false}) \qquad \frac{}{\mathsf{P} \triangleright \mathsf{if\ true\ then\ e\ else\ e}' \xrightarrow{\epsilon} \mathsf{P} \triangleright \mathsf{e}}\ (\mathsf{EL}^{\tau}\text{-if-true})$$

$$\frac{}{\mathsf{P} \triangleright \mathsf{if\ false\ then\ e\ else\ e'} \xrightarrow{\epsilon} \mathsf{P} \triangleright \mathsf{e'}} \ (\mathsf{EL}^{\tau}\text{-if-false})$$

$$\frac{}{\mathsf{P} \triangleright \mathsf{let\ x = v\ in\ e} \xrightarrow{\epsilon} \mathsf{P} \triangleright \mathsf{e}[\mathsf{v}/\mathsf{x}]} \ (\mathsf{EL}^{\tau}\text{-let})$$

$$\frac{\mathsf{f}(\mathsf{x} : \tau_1) : \tau_2 \mapsto \mathsf{return\ e} \in \mathsf{P}}{\mathsf{P} \triangleright_{\overline{\mathsf{f}}} \mathsf{call\ f\ v} \xrightarrow{\epsilon} \mathsf{P} \triangleright_{\overline{\mathsf{f}},\mathsf{f}} \mathsf{return\ e}[\mathsf{v}/\mathsf{x}]} \ (\mathsf{EL}^{\tau}\text{-call-internal})$$

$$\frac{\mathsf{f}(\mathsf{x} : \tau_1) : \tau_2 \mapsto \mathsf{return\ e} \in \mathsf{P}}{\mathsf{P} \triangleright_{\epsilon} \mathsf{call\ f\ v} \xrightarrow{\mathtt{call\ f\ v?}} \mathsf{P} \triangleright_{\mathsf{f}} \mathsf{return\ e}[\mathsf{v}/\mathsf{x}]} \ (\mathsf{EL}^{\tau}\text{-call-in})$$

$$\frac{}{\mathsf{P} \triangleright_{\overline{\mathsf{f}},\mathsf{f},\mathsf{f}'} \mathsf{return\ v} \xrightarrow{\epsilon} \mathsf{P} \triangleright_{\overline{\mathsf{f}},\mathsf{f}} \mathsf{v}} \ (\mathsf{EL}^{\tau}\text{-ret-internal}) \qquad \frac{}{\mathsf{P} \triangleright_{\mathsf{f}} \mathsf{return\ v} \xrightarrow{\mathtt{ret\ v!}} \mathsf{P} \triangleright \mathsf{v}} \ (\mathsf{EL}^{\tau}\text{-ret-out})$$

$$\frac{}{\mathsf{P} \triangleright \mathsf{read} \xrightarrow{\mathtt{read\ n}} \mathsf{P} \triangleright \mathsf{n}} \ (\mathsf{EL}^{\tau}\text{-read}) \qquad \frac{}{\mathsf{P} \triangleright \mathsf{write\ n} \xrightarrow{\mathtt{write\ n}} \mathsf{P} \triangleright \mathsf{n}} \ (\mathsf{EL}^{\tau}\text{-write})$$

$$\frac{\mathsf{P} \triangleright \mathsf{e} \xrightarrow{\epsilon} \mathsf{P} \triangleright \mathsf{e'}}{\mathsf{P} \triangleright \mathbb{E}[\mathsf{e}] \xrightarrow{\epsilon} \mathsf{P} \triangleright \mathbb{E}[\mathsf{e'}]} \ (\mathsf{EL}^{\tau}\text{-ctx}) \qquad \frac{}{\mathsf{P} \triangleright \mathsf{fail} \xrightarrow{\perp} \mathsf{fail}} \ (\mathsf{EL}^{\tau}\text{-fail})$$

$\boxed{\mathsf{P} \triangleright \mathsf{e} \overset{\beta}{\Longrightarrow} \mathsf{P} \triangleright \mathsf{e'}}$

$$\frac{}{\Omega \Rightarrow \Omega} \ (\mathsf{EL}^{\tau}\text{-refl}) \qquad \frac{\Omega \not\Rightarrow \_}{\Omega \overset{\Downarrow}{\Longrightarrow} \Omega} \ (\mathsf{EL}^{\tau}\text{-terminate}) \qquad \frac{\forall \mathsf{n}.\ \Omega \xrightarrow{\epsilon}{}^{\mathsf{n}} \Omega'_{\mathsf{n}}}{\Omega \overset{\Uparrow}{\Longrightarrow} \Omega} \ (\mathsf{EL}^{\tau}\text{-diverge}) \qquad \frac{\Omega \xrightarrow{\epsilon} \Omega'}{\Omega \Rightarrow \Omega'} \ (\mathsf{EL}^{\tau}\text{-silent})$$

$$\frac{\Omega \xrightarrow{\alpha} \Omega'}{\Omega \overset{\alpha}{\Longrightarrow} \Omega'} \ (\mathsf{EL}^{\tau}\text{-single}) \qquad \frac{\Omega \xrightarrow{\gamma} \Omega'}{\Omega \Rightarrow \Omega'} \ (\mathsf{EL}^{\tau}\text{-silent-act}) \qquad \frac{\Omega \overset{\beta}{\Longrightarrow} \Omega'' \quad \Omega'' \overset{\beta'}{\Longrightarrow} \Omega'}{\Omega \overset{\beta\beta'}{\Longrightarrow} \Omega'} \ (\mathsf{EL}^{\tau}\text{-cons})$$

$\boxed{\mathsf{P} \triangleright \mathsf{e} \overset{\mathsf{t}}{\Longrightarrow\!\!\!\!\!\Rightarrow} \mathsf{P} \triangleright \mathsf{e'}}$

$$\frac{\Omega \xrightarrow{\epsilon} \Omega'}{\Omega \Longrightarrow\!\!\!\!\!\Rightarrow \Omega'} \ (\mathsf{EL}^{\tau}\text{-silent}) \qquad \frac{\Omega \overset{\alpha}{\Longrightarrow} \Omega'}{\Omega \overset{\alpha}{\Longrightarrow\!\!\!\!\!\Rightarrow} \Omega'} \ (\mathsf{EL}^{\tau}\text{-action}) \qquad \frac{\Omega \xrightarrow{\gamma} \Omega'}{\Omega \overset{\gamma}{\Longrightarrow\!\!\!\!\!\Rightarrow} \Omega'} \ (\mathsf{EL}^{\tau}\text{-single}) \qquad \frac{\Omega \overset{\sigma}{\Longrightarrow\!\!\!\!\!\Rightarrow} \Omega'' \quad \Omega'' \overset{\sigma'}{\Longrightarrow\!\!\!\!\!\Rightarrow} \Omega'}{\Omega \overset{\sigma\sigma'}{\Longrightarrow\!\!\!\!\!\Rightarrow} \Omega'} \ (\mathsf{EL}^{\tau}\text{-cons})$$

8.1.4 Auxiliaries and Definitions

$\boxed{\text{Helpers}}$

$$\frac{\mathsf{P} \equiv \overline{\mathsf{I}}; \overline{\mathsf{F}} \quad \mathsf{fail} \notin \mathsf{P} \quad s\mathsf{read}, \mathsf{write}\ \_ \notin \mathbb{C} \quad \mathbb{C} \equiv \mathsf{e} \quad \forall \mathsf{call}\ \mathsf{f} \in \mathbb{C}, \mathsf{f} \in \overline{\mathsf{I}}}{\Omega_0(\mathbb{C}[\mathsf{P}]) = \mathsf{P} \triangleright \mathsf{e}} \ (\mathsf{L}^\tau\text{-Initial State})$$

Definition 42 (Program Behaviors)**.**

$$\mathtt{Behav}\,(\mathsf{P}) = \left\{\beta \;\middle|\; \exists \Omega'.\Omega_0(\mathsf{P}) \xRightarrow{\beta} \Omega'\right\}$$

Theorem 41 (Progress)**.**

Theorem 42 (Preservation)**.**

8.2 The Target Language $\mathbf{L^u}$

8.2.1 Syntax

$$\begin{aligned}
\textit{Program}\ \mathbf{P} &::= \overline{\mathbf{I}}; \overline{\mathbf{F}} \\
\textit{Contexts}\ \mathbb{C} &::= \mathbf{e} \\
\textit{Interfaces}\ \mathbf{I} &::= \mathbf{f} \\
\textit{Functions}\ \mathbf{F} &::= \mathbf{f(x)} \mapsto \mathbf{return\ e} \\
\textit{Types}\ \tau &::= \mathtt{Bool} \mid \mathbb{N} \\
\textit{Operations}\ \oplus &::= + \mid - \\
\textit{Values}\ \mathbf{v} &::= \mathbf{true} \mid \mathbf{false} \mid \mathbf{n} \in \mathbb{N} \\
\textit{Expressions}\ \mathbf{e} &::= \mathbf{x} \mid \mathbf{v} \mid \mathbf{e} \oplus \mathbf{e} \mid \mathbf{let\ x = e\ in\ e} \mid \mathbf{if\ e\ then\ e\ else\ e} \mid \mathbf{e} \geq \mathbf{e} \\
&\quad \mid \mathbf{call\ f\ e} \mid \mathbf{read} \mid \mathbf{write\ e} \mid \mathbf{fail} \mid \mathbf{e\ has}\ \tau \\
\textit{Runtime Expr.}\ \mathbf{e} &::= \cdots \mid \mathbf{return\ e} \\
\textit{Eval. Ctxs.}\ \mathbb{E} &::= [\cdot] \mid \mathbf{e} \oplus \mathbb{E} \mid \mathbb{E} \oplus \mathbf{n} \mid \mathbf{let\ x} = \mathbb{E}\ \mathbf{in\ e} \mid \mathbf{if}\ \mathbb{E}\ \mathbf{then\ e\ else\ e} \mid \mathbf{e} \geq \mathbb{E} \mid \mathbb{E} \geq \mathbf{n} \\
&\quad \mid \mathbf{call\ f}\ \mathbb{E} \mid \mathbf{write}\ \mathbb{E} \mid \mathbf{return}\ \mathbb{E} \mid \mathbb{E}\ \mathbf{has}\ \tau \\
\textit{Substitutions}\ \rho &::= [\mathbf{v}/\mathbf{x}] \\
\textit{Prog. States}\ \mathbf{\Omega} &::= \mathbf{P} \triangleright_{\overline{\mathbf{f}}} \mathbf{e} \mid \mathbf{fail} \\
\textit{Labels}\ \lambda &::= \epsilon \mid \alpha \mid \gamma \\
\textit{Actions}\ \alpha &::= \mathtt{read}\ n \mid \mathtt{write}\ n \mid \downarrow \mid \Uparrow \mid \bot \\
\textit{Interactions}\ \gamma &::= \mathtt{call}\ f\ v? \mid \mathtt{ret}\ v! \\
\textit{Behaviors}\ \beta &::= \overline{\alpha} \\
\textit{Traces}\ \sigma &::= \varnothing \mid \sigma\alpha \mid \sigma\gamma
\end{aligned}$$

Program states carry around the stack of called functions (the $\overline{\mathbf{f}}$ subscript) in order to correctly characterise calls and returns that go in Traces. We mostly omit this stack when it just clutters the presentation without itself changing and make it explicit only when it is needed.

8.2.2 Dynamic Semantics

$\Omega \xrightarrow{\lambda} \Omega'$	Program state Ω steps to Ω' emitting action λ.
$\Omega \xRightarrow{\beta} \Omega'$	Program state Ω steps to Ω' with behavior β.
$\Omega \overset{\sigma}{\twoheadrightarrow} \Omega'$	Program state Ω steps to Ω' with trace σ.

$$\boxed{\mathbf{P} \triangleright \mathbf{e} \xrightarrow{\lambda} \mathbf{P} \triangleright \mathbf{e}'}$$

$$\frac{n \oplus n' = n''}{\mathbf{P} \triangleright \mathbf{n} \oplus \mathbf{n}' \xrightarrow{\epsilon} \mathbf{P} \triangleright \mathbf{n}''}\ (\mathsf{EL^u\text{-}op}) \qquad \frac{n \geq n'}{\mathbf{P} \triangleright \mathbf{n} \geq \mathbf{n}' \xrightarrow{\epsilon} \mathbf{P} \triangleright \mathbf{true}}\ (\mathsf{EL^u\text{-}geq\text{-}true})$$

$$\frac{n < n'}{\mathbf{P} \triangleright \mathbf{n} \geq \mathbf{n}' \xrightarrow{\epsilon} \mathbf{P} \triangleright \mathbf{false}}\ (\mathsf{EL^u\text{-}geq\text{-}false})$$

$$\frac{}{\mathbf{P} \triangleright \mathbf{if\ true\ then\ e\ else\ e'} \xrightarrow{\epsilon} \mathbf{P} \triangleright \mathbf{e}}\ (\mathsf{EL^u\text{-}if\text{-}true})$$

$$\frac{}{\mathbf{P} \triangleright \mathbf{if\ false\ then\ e\ else\ e'} \xrightarrow{\epsilon} \mathbf{P} \triangleright \mathbf{e}'}\ (\mathsf{EL^u\text{-}if\text{-}false})$$

$$\frac{}{\mathbf{P} \triangleright \mathbf{let\ x = v\ in\ e} \xrightarrow{\epsilon} \mathbf{P} \triangleright \mathbf{e}[\mathbf{v}/\mathbf{x}]}\ (\mathsf{EL^u\text{-}let}) \qquad \frac{}{\mathbf{P} \triangleright \mathbf{read} \xrightarrow{\texttt{read n}} \mathbf{P} \triangleright \mathbf{n}}\ (\mathsf{EL^u\text{-}read})$$

$$\frac{}{\mathbf{P} \triangleright \mathbf{write\ n} \xrightarrow{\texttt{write n}} \mathbf{P} \triangleright \mathbf{n}}\ (\mathsf{EL^u\text{-}write}) \qquad \frac{\mathbf{P} \triangleright \mathbf{e} \xrightarrow{\epsilon} \mathbf{P} \triangleright \mathbf{e}'}{\mathbf{P} \triangleright \mathbb{E}[\mathbf{e}] \xrightarrow{\epsilon} \mathbf{P} \triangleright \mathbb{E}[\mathbf{e}']}\ (\mathsf{EL^u\text{-}ctx})$$

$$\frac{}{\mathbf{P} \triangleright \mathbf{fail} \xrightarrow{\epsilon} \mathbf{fail}}\ (\mathsf{EL^u\text{-}fail}) \qquad \frac{\mathbf{v} \equiv \mathbf{true} \vee \mathbf{v} \equiv \mathbf{false}}{\mathbf{P} \triangleright \mathbf{v}\ \mathbf{has}\ \texttt{Bool} \xrightarrow{\epsilon} \mathbf{P} \triangleright \mathbf{true}}\ (\mathsf{EL^u\text{-}check\text{-}bool\text{-}true})$$

$$\frac{}{\mathbf{P} \triangleright \mathbf{n}\ \mathbf{has}\ \texttt{Bool} \xrightarrow{\epsilon} \mathbf{P} \triangleright \mathbf{false}}\ (\mathsf{EL^u\text{-}check\text{-}bool\text{-}false}) \qquad \frac{}{\mathbf{P} \triangleright \mathbf{n}\ \mathbf{has}\ \mathbb{N} \xrightarrow{\epsilon} \mathbf{P} \triangleright \mathbf{true}}\ (\mathsf{EL^u\text{-}check\text{-}nat\text{-}true})$$

$$\frac{\mathbf{v} \equiv \mathbf{true} \vee \mathbf{v} \equiv \mathbf{false}}{\mathbf{P} \triangleright \mathbf{v}\ \mathbf{has}\ \mathbb{N} \xrightarrow{\epsilon} \mathbf{P} \triangleright \mathbf{false}}\ (\mathsf{EL^u\text{-}check\text{-}nat\text{-}false})$$

$$\frac{\mathbf{f}(\mathbf{x}) \mapsto \mathbf{return\ e} \in \mathbf{P}}{\mathbf{P} \triangleright_{\overline{\mathbf{f}}} \mathbf{call\ f\ v} \xrightarrow{\epsilon} \mathbf{P} \triangleright_{\overline{\mathbf{f}},\mathbf{f}} \mathbf{return\ e}[\mathbf{v}/\mathbf{x}]}\ (\mathsf{EL^u\text{-}call\text{-}internal})$$

$$\frac{\mathbf{f}(\mathbf{x}) \mapsto \mathbf{return\ e} \in \mathbf{P}}{\mathbf{P} \triangleright_{\epsilon} \mathbf{call\ f\ v} \xrightarrow{\texttt{call f v?}} \mathbf{P} \triangleright_{\mathbf{f}} \mathbf{return\ e}[\mathbf{v}/\mathbf{x}]}\ (\mathsf{EL^u\text{-}call\text{-}in})$$

$$\frac{}{\mathbf{P} \triangleright_{\overline{\mathbf{f}},\mathbf{f},\mathbf{f}'} \mathbf{return\ v} \xrightarrow{\epsilon} \mathbf{P} \triangleright_{\overline{\mathbf{f}},\mathbf{f}} \mathbf{v}}\ (\mathsf{EL^u\text{-}ret\text{-}internal}) \qquad \frac{}{\mathbf{P} \triangleright_{\mathbf{f}} \mathbf{return\ v} \xrightarrow{\texttt{ret v!}} \mathbf{P} \triangleright \mathbf{v}}\ (\mathsf{EL^u\text{-}ret\text{-}out})$$

$$\frac{\mathbf{v} \equiv \mathbf{true} \vee \mathbf{v} \equiv \mathbf{false} \vee \mathbf{v'} \equiv \mathbf{true} \vee \mathbf{v'} \equiv \mathbf{false}}{\mathbf{P} \triangleright \mathbf{v} \oplus \mathbf{v'} \xrightarrow{\perp} \mathbf{fail}} \ (\mathsf{EL^u\text{-}op\text{-}fail})$$

$$\frac{\mathbf{v} \equiv \mathbf{true} \vee \mathbf{v} \equiv \mathbf{false} \vee \mathbf{v'} \equiv \mathbf{true} \vee \mathbf{v'} \equiv \mathbf{false}}{\mathbf{P} \triangleright \mathbf{v} \geq \mathbf{v'} \xrightarrow{\perp} \mathbf{fail}} \ (\mathsf{EL^u\text{-}geq\text{-}fail})$$

$$\frac{}{\mathbf{P} \triangleright \mathbf{if\ n\ then\ e\ else\ e'} \xrightarrow{\perp} \mathbf{fail}} \ (\mathsf{EL^u\text{-}if\text{-}fail}) \qquad \frac{}{\mathbf{P} \triangleright \mathbf{fail} \xrightarrow{\perp} \mathbf{fail}} \ (\mathsf{EL^u\text{-}fail})$$

$\boxed{\mathbf{P} \triangleright \mathbf{e} \overset{\beta}{\Longrightarrow} \mathbf{P} \triangleright \mathbf{e'}}$

$$\frac{}{\mathbf{\Omega} \Rightarrow \mathbf{\Omega}} \ (\mathsf{EL^u\text{-}refl}) \qquad \frac{\mathbf{\Omega} \not\Rightarrow \_}{\mathbf{\Omega} \overset{\Downarrow}{\Longrightarrow} \mathbf{\Omega}} \ (\mathsf{EL^u\text{-}terminate}) \qquad \frac{\forall \mathbf{n}.\ \mathbf{\Omega} \xrightarrow{\epsilon}^{\mathbf{n}} \mathbf{\Omega'_n}}{\mathbf{\Omega} \overset{\Uparrow}{\Longrightarrow} \mathbf{\Omega}} \ (\mathsf{EL^u\text{-}diverge}) \qquad \frac{\mathbf{\Omega} \xrightarrow{\epsilon} \mathbf{\Omega'}}{\mathbf{\Omega} \Rightarrow \mathbf{\Omega'}} \ (\mathsf{EL^u\text{-}silent})$$

$$\frac{\mathbf{\Omega} \xrightarrow{\gamma} \mathbf{\Omega'}}{\mathbf{\Omega} \Rightarrow \mathbf{\Omega'}} \ (\mathsf{EL^u\text{-}silent\text{-}act}) \qquad \frac{\mathbf{\Omega} \xrightarrow{\alpha} \mathbf{\Omega'}}{\mathbf{\Omega} \overset{\alpha}{\Longrightarrow} \mathbf{\Omega'}} \ (\mathsf{EL^u\text{-}single}) \qquad \frac{\mathbf{\Omega} \overset{\beta}{\Longrightarrow} \mathbf{\Omega''} \quad \mathbf{\Omega''} \overset{\beta'}{\Longrightarrow} \mathbf{\Omega'}}{\mathbf{\Omega} \overset{\beta\beta'}{\Longrightarrow} \mathbf{\Omega'}} \ (\mathsf{EL^u\text{-}cons})$$

$\boxed{\mathbf{P} \triangleright \mathbf{e} \overset{\sigma}{\Longrightarrow}\!\!\!\!\!\!\Rightarrow \mathbf{P} \triangleright \mathbf{e'}}$

$$\frac{\mathbf{\Omega} \xrightarrow{\epsilon} \mathbf{\Omega'}}{\mathbf{\Omega} \Rightarrow\!\!\!\!\Rightarrow \mathbf{\Omega'}} \ (\mathsf{EL^u\text{-}silent}) \qquad \frac{\mathbf{\Omega} \overset{\alpha}{\Longrightarrow} \mathbf{\Omega'}}{\mathbf{\Omega} \overset{\alpha}{\Longrightarrow}\!\!\!\!\Rightarrow \mathbf{\Omega'}} \ (\mathsf{EL^u\text{-}action}) \qquad \frac{\mathbf{\Omega} \xrightarrow{\gamma} \mathbf{\Omega'}}{\mathbf{\Omega} \overset{\gamma}{\Longrightarrow}\!\!\!\!\Rightarrow \mathbf{\Omega'}} \ (\mathsf{EL^u\text{-}single}) \qquad \frac{\mathbf{\Omega} \overset{\sigma}{\Longrightarrow}\!\!\!\!\Rightarrow \mathbf{\Omega''} \quad \mathbf{\Omega''} \overset{\sigma'}{\Longrightarrow}\!\!\!\!\Rightarrow \mathbf{\Omega'}}{\mathbf{\Omega} \overset{\sigma\sigma'}{\Longrightarrow}\!\!\!\!\Rightarrow \mathbf{\Omega'}} \ (\mathsf{EL^u\text{-}cons})$$

8.2.3 Auxiliaries and Definitions

$\boxed{\text{Helpers}}$

$$\frac{\mathbf{P} \equiv \overline{\mathbf{I}}; \overline{\mathbf{F}} \qquad \mathbb{C} \equiv \mathbf{e} \qquad \mathbf{read}, \mathbf{write}\ \_ \notin \mathbb{C} \qquad \forall \mathbf{call\ f} \in \mathbb{C}, \mathbf{f} \in \overline{\mathbf{I}}}{\mathbf{\Omega_0}(\mathbb{C}[\mathbf{P}]) = \mathbf{P} \triangleright \mathbf{e}} \ (\mathsf{L^u\text{-}Initial\ State})$$

Definition 43 (Program Behaviors).

$$\mathsf{Behav}\left(\mathbf{P}\right) = \left\{\beta \;\middle|\; \exists \mathbf{\Omega'}.\mathbf{\Omega_0}(\mathbf{P}) \overset{\beta}{\Longrightarrow} \mathbf{\Omega'}\right\}$$

Definition 44 (Program Traces).

$$\mathsf{TR}(\mathbf{P}) = \left\{\sigma \;\middle|\; \exists \mathbf{\Omega'}.\mathbf{\Omega_0}(\mathbf{P}) \overset{\sigma}{\Longrightarrow}\!\!\!\!\Rightarrow \mathbf{\Omega'}\right\}$$

8.3 $[\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}$: A Compiler from L^τ to $\mathbf{L^u}$

$$[\![\mathsf{I}_1,\cdots,\mathsf{I}_m;\mathsf{F}_1,\cdots,\mathsf{F}_n]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} = [\![\mathsf{I}_1]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}},\cdots,[\![\mathsf{I}_m]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}};[\![\mathsf{F}_1]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}},\cdots,[\![\mathsf{F}_n]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \quad ([\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\text{-Prog})$$

$$[\![\mathsf{f}:\tau\to\tau']\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} = \mathbf{f} \quad ([\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\text{-Intf})$$

$$[\![\mathsf{f}(\mathsf{x}:\tau):\tau'\mapsto\mathsf{return}\ \mathsf{e}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} = \mathbf{f}(\mathbf{x})\mapsto\mathbf{return\ if\ x\ has}\ [\![\tau]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\ \mathbf{then}\ [\![\mathsf{e}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\ \mathbf{else\ fail} \quad ([\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\text{-Fun})$$

$$[\![\mathsf{n}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} = \mathbf{n} \quad ([\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\text{-Nat})$$

$$[\![\mathsf{true}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} = \mathbf{true} \quad ([\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\text{-True})$$

$$[\![\mathsf{false}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} = \mathbf{false} \quad ([\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\text{-False})$$

$$[\![\mathsf{x}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} = \mathbf{x} \quad ([\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\text{-Var})$$

$$[\![\mathsf{e}\oplus\mathsf{e}']\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} = [\![\mathsf{e}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\oplus[\![\mathsf{e}']\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \quad ([\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\text{-Op})$$

$$[\![\mathsf{e}\geq\mathsf{e}']\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} = [\![\mathsf{e}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\geq[\![\mathsf{e}']\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \quad ([\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\text{-Geq})$$

$$[\![\mathsf{let}\ \mathsf{x}:\tau=\mathsf{e}\ \mathsf{in}\ \mathsf{e}']\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} = \mathbf{let\ x} = [\![\mathsf{e}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\ \mathbf{in}\ [\![\mathsf{e}']\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \quad ([\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\text{-Let})$$

$$[\![\mathsf{if}\ \mathsf{e}\ \mathsf{then}\ \mathsf{e}'\ \mathsf{else}\ \mathsf{e}'']\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} = \mathbf{if}\ [\![\mathsf{e}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\ \mathbf{then}\ [\![\mathsf{e}']\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\ \mathbf{else}\ [\![\mathsf{e}'']\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \quad ([\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\text{-If})$$

$$[\![\mathsf{call}\ \mathsf{f}\ \mathsf{e}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} = \mathbf{call\ f}\ [\![\mathsf{e}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \quad ([\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\text{-Call})$$

$$[\![\mathsf{read}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} = \mathbf{read} \quad ([\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\text{-Read})$$

$$[\![\mathsf{write}\ \mathsf{e}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} = \mathbf{write}\ [\![\mathsf{e}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \quad ([\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\text{-Write})$$

$$[\![\mathsf{Nat}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} = \mathbb{N} \quad ([\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\text{-Ty-Nat})$$

$$[\![\mathsf{Bool}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} = \mathbf{Bool} \quad ([\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\text{-Ty-Bool})$$

8.4 Proof That $[\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}$ Is RrSC

8.4.1 $\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}$: Backtranslation of Contexts from $\mathbf{L^u}$ to L^τ

Technically, the backtranslation needs one additional parameter to be passed around, the list of functions defined by the compiled component $\bar{\mathsf{I}}$, we elide it for simplicity when it is not necessary.

$$\langle\!\langle\mathbf{n}\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} = \mathsf{n}+2 \quad (\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\text{-Nat})$$

$$\langle\!\langle\mathbf{true}\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} = 1 \quad (\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\text{-True})$$

$$\langle\!\langle\mathbf{false}\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} = 0 \quad (\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\text{-False})$$

$$\langle\!\langle\mathbf{x}\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} = \mathsf{x} \quad (\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\text{-Var})$$

$$\langle\!\langle \mathsf{e} \oplus \mathsf{e}' \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} = \mathsf{let\ x1 : Nat{=}extract_{Nat}}(\langle\!\langle \mathsf{e} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}) \quad \text{(}\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\text{-Op)}$$
$$\mathsf{in\ let\ x2 : Nat{=}extract_{Nat}}(\langle\!\langle \mathsf{e}' \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau})$$
$$\mathsf{in\ inject_{Nat}(x1 \oplus x2)}$$

$$\langle\!\langle \mathsf{e} \geq \mathsf{e}' \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} = \mathsf{let\ x1 : Nat{=}extract_{Nat}}(\langle\!\langle \mathsf{e} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}) \quad \text{(}\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\text{-Geq)}$$
$$\mathsf{in\ let\ x2 : Nat{=}extract_{Nat}}(\langle\!\langle \mathsf{e}' \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau})$$
$$\mathsf{in\ inject_{Bool}(x1 \geq x2)}$$

$$\langle\!\langle \mathbf{let\ x = e\ in\ e'} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} = \mathsf{let\ x : Nat} = \langle\!\langle \mathsf{e} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\ \mathsf{in}\ \langle\!\langle \mathsf{e}' \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} \quad \text{(}\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\text{-Let)}$$

$$\langle\!\langle \mathbf{if\ e\ then\ e'\ else\ e''} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} = \mathsf{if\ extract_{Bool}}(\langle\!\langle \mathsf{e} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau})\ \mathsf{then}\ \langle\!\langle \mathsf{e}' \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\ \mathsf{else}\ \langle\!\langle \mathsf{e}'' \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} \quad \text{(}\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\text{-If)}$$

$$\langle\!\langle \mathbf{call\ f\ e} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} = \mathsf{inject}_{\tau'}(\mathsf{call\ f\ extract}_{\tau}(\langle\!\langle \mathsf{e} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau})) \quad \text{(}\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\text{-Call)}$$
$$\text{if } \mathsf{f} : \tau \to \tau' \in \bar{\mathsf{I}}$$

$$\langle\!\langle \mathbf{e\ has\ }\tau \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} = \begin{cases} \mathsf{let\ x : Nat} = \langle\!\langle \mathsf{e} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\ \mathsf{in\ if\ x \geq 2\ then\ 0\ else\ 1} & \text{if } \tau \equiv \mathtt{Bool} \\ \mathsf{let\ x : Nat} = \langle\!\langle \mathsf{e} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\ \mathsf{in\ if\ x \geq 2\ then\ 1\ else\ 0} & \text{if } \tau \equiv \mathbb{N} \end{cases} \quad \text{(}\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\text{-Check)}$$

Helper functions The universal type is Nat but the encoding is not straight from Nat but it is Nat shifted by 2. $\mathsf{inject}_\tau(\mathsf{e})$ takes an expression e of type τ and returns an expression whose type is the universal type. $\mathsf{extract}_\tau(\mathsf{e})$ takes an expression e of universal type and returns an expression whose type is τ.

$$\begin{aligned}
\mathsf{inject_{Nat}(e)} &= \mathsf{e + 2} \\
\mathsf{inject_{Bool}(e)} &= \mathsf{if\ e\ then\ 1\ else\ 0} \\
\mathsf{extract_{Nat}(e)} &= \mathsf{let\ x = e\ in\ if\ x \geq 2\ then\ x - 2\ else\ fail} \\
\mathsf{extract_{Bool}(e)} &= \mathsf{let\ x = e\ in\ if\ x \geq 2\ then\ fail\ else\ if\ x + 1 \geq 2\ then\ true\ else\ false}
\end{aligned}$$

8.4.2 Cross-Language Logical Relation

Language De-sugaring

$$\begin{aligned}
\mathsf{v} &::= \ \dots \mid \mathsf{call\ f} \\
\mathsf{e} &::= \ \dots \mid \mathsf{call\ f\ e} \\
\textit{Types}\ \tau &::= \sigma \mid \sigma \to \sigma \\
\textit{Base Types}\ \sigma &::= \mathsf{Nat} \mid \mathsf{Bool}
\end{aligned}$$

Replace Rule $\mathsf{T}\mathsf{L}^\tau$-function-call with these below.

$$\frac{\mathsf{f}(\mathsf{x} : \sigma) : \sigma' \mapsto \mathsf{return\ e} \in \mathtt{dom}\left(\overline{\mathsf{F}}\right)}{\mathsf{P}; \Gamma \vdash \mathsf{call\ f} : \sigma \to \sigma'}\ (\mathsf{TL}^\tau\text{-call}) \qquad \frac{\mathsf{P}; \Gamma \vdash \mathsf{call\ f} : \sigma' \to \sigma \quad \mathsf{P}; \Gamma \vdash \mathsf{e}' : \sigma'}{\mathsf{P}; \Gamma \vdash \mathsf{call\ f\ e}' : \sigma}\ (\mathsf{TL}^\tau\text{-app})$$

Apply the same changes above to $\mathbf{L^u}$ too.

Context well-formedness ensures that expressions are never turned into $\mathbf{call\ f}$ values.

$$\mathbf{\Gamma} ::= \varnothing \mid \mathbf{\Gamma}, \mathbf{x}$$

$$\frac{}{\mathbf{P};\mathbf{\Gamma} \vdash \mathbf{true}}\ (\mathsf{Ctx}\text{-}\mathbf{L^u}\text{-true}) \qquad \frac{}{\mathbf{P};\mathbf{\Gamma} \vdash \mathbf{false}}\ (\mathsf{Ctx}\text{-}\mathbf{L^u}\text{-false}) \qquad \frac{}{\mathbf{P};\mathbf{\Gamma} \vdash \mathbf{n}}\ (\mathsf{Ctx}\text{-}\mathbf{L^u}\text{-nat}) \qquad \frac{\mathbf{x} \in \mathsf{dom}(\mathbf{\Gamma})}{\mathbf{P};\mathbf{\Gamma} \vdash \mathbf{x}}\ (\mathsf{Ctx}\text{-}\mathbf{L^u}\text{-var})$$

$$\frac{\begin{array}{c}\mathbf{P};\mathbf{\Gamma} \vdash \mathbf{e'} \quad \mathbf{e'} \not\equiv \mathbf{call\ f}\\ \mathbf{f(x) \mapsto return\ e} \in \mathbf{P}\end{array}}{\mathbf{P};\mathbf{\Gamma} \vdash \mathbf{call\ f\ e'}}\ (\mathsf{Ctx}\text{-}\mathbf{L^u}\text{-app}) \qquad \frac{\begin{array}{c}\mathbf{P};\mathbf{\Gamma} \vdash \mathbf{e} \quad \mathbf{P};\mathbf{\Gamma} \vdash \mathbf{e'}\\ \mathbf{e},\mathbf{e'} \not\equiv \mathbf{call\ f}\end{array}}{\mathbf{P};\mathbf{\Gamma} \vdash \mathbf{e} \oplus \mathbf{e'}}\ (\mathsf{Ctx}\text{-}\mathbf{L^u}\text{-op})$$

$$\frac{\begin{array}{c}\mathbf{P};\mathbf{\Gamma} \vdash \mathbf{e} \quad \mathbf{P};\mathbf{\Gamma} \vdash \mathbf{e'}\\ \mathbf{e},\mathbf{e'} \not\equiv \mathbf{call\ f}\end{array}}{\mathbf{P};\mathbf{\Gamma} \vdash \mathbf{e} \geq \mathbf{e'}}\ (\mathsf{Ctx}\text{-}\mathbf{L^u}\text{-geq}) \qquad \frac{\begin{array}{c}\mathbf{P};\mathbf{\Gamma} \vdash \mathbf{e} \quad \mathbf{P};\mathbf{\Gamma},\mathbf{x} \vdash \mathbf{e'}\\ \mathbf{e},\mathbf{e'} \not\equiv \mathbf{call\ f}\end{array}}{\mathbf{P};\mathbf{\Gamma} \vdash \mathbf{let\ x = e\ in\ e'}}\ (\mathsf{Ctx}\text{-}\mathbf{L^u}\text{-letin})$$

$$\frac{\begin{array}{c}\mathbf{P};\mathbf{\Gamma} \vdash \mathbf{e} \quad \mathbf{P};\mathbf{\Gamma} \vdash \mathbf{e'} \quad \mathbf{P};\mathbf{\Gamma} \vdash \mathbf{e''}\\ \mathbf{e},\mathbf{e'},\mathbf{e''} \not\equiv \mathbf{call\ f}\end{array}}{\mathbf{P};\mathbf{\Gamma} \vdash \mathbf{if\ e\ then\ e'\ else\ e''}}\ (\mathsf{Ctx}\text{-}\mathbf{L^u}\text{-if}) \qquad \frac{\begin{array}{c}\mathbf{P};\mathbf{\Gamma} \vdash \mathbf{e}\\ \mathbf{e} \not\equiv \mathbf{call\ f}\end{array}}{\mathbf{P};\mathbf{\Gamma} \vdash \mathbf{e\ has}\ \tau}\ (\mathsf{Ctx}\text{-}\mathbf{L^u}\text{-check})$$

Replace Rule ($[\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}$-Call) with these below.

$$[\![\mathsf{call\ f}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} = \mathbf{call\ f} \qquad ([\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\text{-Call-v})$$

$$[\![\mathsf{e\ e'}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} = [\![\mathsf{e}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\ [\![\mathsf{e'}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \qquad ([\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}\text{-App})$$

Worlds

$$\begin{aligned}
\mathit{World}\ W &::= (n, (\mathsf{P}, \mathbf{P}))\\
\mathit{lev}((n, _)) &= n\\
\mathit{progs}((_, (\mathsf{P}, \mathbf{P}))) &= (\mathsf{P}, \mathbf{P})\\
\mathit{srcprog}((_, (\mathsf{P}, \mathbf{P}))) &= \mathsf{P}\\
\mathit{trgprog}((_, (\mathsf{P}, \mathbf{P}))) &= \mathbf{P}\\
\triangleright((0, _)) &= (0, _)\\
\triangleright((n+1, _)) &= (n, _)\\
W \sqsupseteq W' &= \mathit{lev}(W') \leq \mathit{lev}(W)\\
W \sqsupset_\triangleright W' &= \mathit{lev}(W') < \mathit{lev}(W)\\
O(W)_{\lesssim} &\stackrel{\mathsf{def}}{=} \left\{ (\mathsf{e}, \mathbf{e}) \;\middle|\; \begin{array}{l}\text{if } \mathit{lev}(W) = n \text{ and } \mathit{progs}(W) = (\mathsf{P}, \mathbf{P})\\ \text{and } \mathsf{P} \triangleright \mathsf{e} \xrightarrow{\beta}{}^{\mathsf{n}}\ \mathsf{P} \triangleright \mathsf{e'}\\ \text{then } \exists \mathbf{k}.\ \mathbf{P} \triangleright \mathbf{e} \xrightarrow{\beta}{}^{\mathbf{k}}\ \mathbf{P} \triangleright \mathbf{e'}\end{array} \right\}
\end{aligned}$$

$$O(W)_{\gtrsim} \stackrel{\mathsf{def}}{=} \left\{ (\mathsf{e}, \mathbf{e}) \;\middle|\; \begin{array}{l} \text{if } \mathit{lev}(W) = n \text{ and } \mathit{progs}(W) = (\mathsf{P}, \mathbf{P}) \\ \text{ and } \mathbf{P} \triangleright \mathbf{e} \xRightarrow{\beta}{}^{\mathbf{n}} \mathbf{P} \triangleright \mathbf{e}' \\ \text{ then } \exists \mathsf{k}.\ \mathsf{P} \triangleright \mathsf{e} \xRightarrow{\beta}{}^{\mathsf{k}} \mathsf{P} \triangleright \mathsf{e}' \end{array} \right\}$$

$$O(W)_{\approx} \stackrel{\mathsf{def}}{=} O(W)_{\lesssim} \cap O(W)_{\gtrsim}$$

$$\triangleright R \stackrel{\mathsf{def}}{=} \{(W, \mathsf{v}, \mathbf{v}) \mid \text{if } \mathit{lev}(W) > 0 \text{ then } (\triangleright(W), \mathsf{v}, \mathbf{v}) \in R\}$$

$$\nearrow(R) \stackrel{\mathsf{def}}{=} \{(W, \mathsf{v_1}, \mathbf{v_2}) \mid \forall W' \sqsupseteq W.(W', \mathsf{v_1}, \mathbf{v_2}) \in R\}$$

for R a world-values relation

The Universal Type and Pseudo Types We index the logical relation by a pseudo type, which captures all the standard types as well as the type of backtranslated stuff.

$$\hat{\tau} ::= \tau \mid \mathsf{EmulTy}$$

Function `toEmul` $(\cdot)$ takes a $\mathbf{\Gamma}$ and returns a $\mathsf{\Gamma}$ that has the same domain but where variables all have type Nat.

Value, Context, Expression and Environment relation

$$\mathcal{V}\llbracket \mathsf{Bool} \rrbracket_{\triangledown} \stackrel{\mathsf{def}}{=} \{(W, \mathsf{true}, \mathbf{true}), (W, \mathsf{false}, \mathbf{false})\}$$

$$\mathcal{V}\llbracket \mathsf{Nat} \rrbracket_{\triangledown} \stackrel{\mathsf{def}}{=} \{(W, \mathsf{n}, \mathbf{n})\}$$

$$\mathcal{V}\left\llbracket \hat{\tau} \to \hat{\tau}' \right\rrbracket_{\triangledown} \stackrel{\mathsf{def}}{=} \left\{ (W, \mathsf{call\ f}, \mathbf{call\ f}) \;\middle|\; \begin{array}{l} \mathsf{f}(\mathsf{x} : \tau) : \tau' \mapsto \mathsf{return\ e} \in \mathit{srcprog}(W) \text{ and} \\ \mathbf{f}(\mathbf{x}) \mapsto \mathbf{return\ e} \in \mathit{trgprog}(W) \\ \forall W', \mathsf{v}', \mathbf{v}'.\ \text{if } W' \sqsupset_{\triangleright} W \text{ and } (W', \mathsf{v}', \mathbf{v}') \in \mathcal{V}\llbracket \hat{\tau} \rrbracket_{\triangledown} \text{ then} \\ \ (W', \mathsf{return\ e}[\mathsf{v}/\mathsf{x}], \mathbf{return\ e}[\mathbf{v}/\mathbf{x}]) \in \mathcal{E}\left\llbracket \hat{\tau}' \right\rrbracket_{\triangledown} \end{array} \right\}$$

$$\mathcal{V}\llbracket \mathsf{EmulTy} \rrbracket_{\triangledown} \stackrel{\mathsf{def}}{=} \{(W, \mathsf{n}+2, \mathbf{n}), (W, 1, \mathbf{true}), (W, 0, \mathbf{false})\}$$

$$\mathcal{K}\llbracket \hat{\tau} \rrbracket_{\triangledown} \stackrel{\mathsf{def}}{=} \left\{ (W, \mathbb{E}, \mathbb{E}) \;\middle|\; \begin{array}{l} \forall W', \mathsf{v}, \mathbf{v}.\ \text{if } W' \sqsupseteq W \text{ and } (W', \mathsf{v}, \mathbf{v}) \in \mathcal{V}\llbracket \hat{\tau} \rrbracket_{\triangledown} \text{ then} \\ (\mathbb{E}[\mathsf{v}], \mathbb{E}[\mathbf{v}]) \in O(W')_{\triangledown} \end{array} \right\}$$

$$\mathcal{E}\llbracket \hat{\tau} \rrbracket_{\triangledown} \stackrel{\mathsf{def}}{=} \{(W, \mathsf{t}, \mathbf{t}) \mid \forall \mathbb{E}, \mathbb{E}.\ \text{if } (W, \mathbb{E}, \mathbb{E}) \in \mathcal{K}\llbracket \hat{\tau} \rrbracket_{\triangledown} \text{ then } (\mathbb{E}[\mathsf{t}], \mathbb{E}[\mathbf{t}]) \in O(W)_{\triangledown}\}$$

$$\mathcal{G}\llbracket \varnothing \rrbracket_{\triangledown} \stackrel{\mathsf{def}}{=} \{(W, \varnothing, \varnothing)\}$$

$$\mathcal{G}\left\llbracket \hat{\Gamma}, \mathsf{x} : \hat{\tau} \right\rrbracket_{\triangledown} \stackrel{\mathsf{def}}{=} \left\{ (W, \gamma[\mathsf{v}/\mathsf{x}], \gamma[\mathbf{v}/\mathbf{x}]) \;\middle|\; (W, \gamma, \gamma) \in \mathcal{G}\left\llbracket \hat{\Gamma} \right\rrbracket_{\triangledown} \text{ and } (W, \mathsf{v}, \mathbf{v}) \in \mathcal{V}\llbracket \hat{\tau} \rrbracket_{\triangledown} \right\}$$

Relation for Open and Closed Terms and Programs

Definition 45 (Logical relation up to n steps)**.**

$$\hat{\Gamma}; \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \;\triangledown_n\; \mathbf{e} : \hat{\tau} \stackrel{\mathsf{def}}{=} \hat{\Gamma}; \mathsf{P} \vdash \mathsf{e} : \hat{\tau}$$

$$
\begin{aligned}
\text{and}\ \ & \forall W. \\
& \text{if}\ \ \mathit{lev}(W) \geq n \text{ and } \mathit{progs}(W) = (\mathsf{P}, \mathbf{P}) \\
\text{then}\ \ & \forall \gamma, \gamma.\ (W, \gamma, \gamma) \in \mathcal{G}\left[\!\left[\hat{\Gamma}\right]\!\right]_{\triangledown}, \\
& (W, \mathsf{e}\gamma, \mathbf{e}\gamma) \in \mathcal{E}\left[\!\left[\hat{\tau}\right]\!\right]_{\triangledown}
\end{aligned}
$$

Definition 46 (Logical relation for expressions)**.**

$$
\hat{\Gamma}; \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \triangledown \mathbf{e} : \hat{\tau} \stackrel{\mathsf{def}}{=} \forall n \in \mathbb{N}.\ \hat{\Gamma}; \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \triangledown_n \mathbf{e} : \hat{\tau}
$$

Definition 47 (Logical relation for programs)**.**

$$
\begin{aligned}
\vdash \mathsf{P} \triangledown \mathbf{P} \stackrel{\mathsf{def}}{=}\ & \mathsf{f}(\mathsf{x} : \sigma') : \sigma \mapsto \mathsf{return}\ \mathsf{e} \in \mathsf{P} \text{ iff } \mathbf{f}(\mathbf{x}) \mapsto \mathbf{return}\ \mathbf{e} \in \mathbf{P} \\
& \mathsf{x} : \sigma'; \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \triangledown \mathbf{e} : \sigma
\end{aligned}
$$

Auxiliary Lemmas from Existing Work

Lemma 2 (No observation with 0 steps)**.**

$$
\begin{aligned}
\text{if}\ \ & \mathit{lev}(W) = 0 \\
\text{then}\ \ & \forall \mathsf{e}, \mathbf{e}. (\mathsf{e}, \mathbf{e}) \in O(W)_{\triangledown}
\end{aligned}
$$

Proof. Trivial adaptation of the same proof in [**?**, **?**]. ☐

Lemma 3 (No steps means relation)**.**

$$
\begin{aligned}
\text{if}\ \ & \mathit{lev}(W) = n \\
& \mathsf{P} \triangleright \mathsf{e} \xRightarrow{\beta}{}^{\mathsf{n}}\ \_ \\
& \mathbf{P} \triangleright \mathbf{e} \xRightarrow{\beta}{}^{\mathbf{n}}\ \_ \\
\text{then}\ \ & (\mathsf{e}, \mathbf{e}) \in O(W)_{\triangledown}
\end{aligned}
$$

Proof. Trivial adaptation of the same proof in [**?**, **?**]. ☐

Lemma 4 (Later preserves monotonicity)**.**

$$
\begin{aligned}
\text{if}\ \ & \forall R, R \subseteq \nearrow(R) \\
\text{then}\ \ & \triangleright R \subseteq \nearrow(\triangleright R)
\end{aligned}
$$

Proof. Trivial adaptation of the same proof in [**?**, **?**]. ☐

Lemma 5 (Monotonicity for environment relation)**.**

$$
\begin{aligned}
\text{if}\ \ & W' \sqsupseteq W \\
& (W, \gamma, \gamma) \in \mathcal{G}\left[\!\left[\Gamma\right]\!\right]_{\triangledown} \\
\text{then}\ \ & (W', \gamma, \gamma) \in \mathcal{G}\left[\!\left[\Gamma\right]\!\right]_{\triangledown}
\end{aligned}
$$

Proof. Trivial adaptation of the same proof in [?, ?]. □

Lemma 6 (Monotonicity for continuation relation)**.**

$$\begin{aligned} \text{if } & W' \sqsupseteq W \\ & (W, \mathbb{C}, \mathbb{C}) \in \mathcal{K}\llbracket \hat{\tau} \rrbracket_{\triangledown} \\ \text{then } & (W', \mathbb{C}, \mathbb{C}) \in \mathcal{K}\llbracket \hat{\tau} \rrbracket_{\triangledown} \end{aligned}$$

Proof. Trivial adaptation of the same proof in [?, ?]. □

Lemma 7 (Monotonicity for value relation)**.**

$$\mathcal{V}\llbracket \hat{\tau} \rrbracket_{\triangledown} \subseteq \nearrow(\mathcal{V}\llbracket \hat{\tau} \rrbracket_{\triangledown})$$

Proof. Trivial adaptation of the same proof in [?, ?]. □

Lemma 8 (Value relation implies term relation)**.**

$$\forall \hat{\tau}, \mathcal{V}\llbracket \hat{\tau} \rrbracket_{\triangledown} \subseteq \mathcal{E}\llbracket \hat{\tau} \rrbracket_{\triangledown}$$

Proof. Trivial adaptation of the same proof in [?, ?]. □

Lemma 9 (Adequacy for $\lesssim$)**.**

$$\begin{aligned} \text{if } & \varnothing; \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \lesssim_n \mathbf{e} : \tau \\ & \mathsf{P} \triangleright \mathsf{e} \xRightarrow{\beta}{}^{\mathsf{m}} \mathsf{P} \triangleright \mathsf{e}' \text{ with } n \geq m \\ \text{then } & \mathbf{P} \triangleright \mathbf{e} \xRightarrow{\beta} \mathbf{P} \triangleright \_. \end{aligned}$$

Proof. By Definition 46 (Logical relation for expressions) we have that $(W, \mathsf{e}, \mathbf{e}) \in \mathcal{E}\llbracket \tau \rrbracket_{\lesssim}$ for a W such that $lev(W) = n$.

By taking $(W, [\cdot], [\cdot]) \in \mathcal{K}\llbracket \tau \rrbracket_{\lesssim}$ we know that $(\mathsf{e}, \mathbf{e}) \in O(W)_{\lesssim}$.

By definition of $O(\cdot)_{\lesssim}$, with the HP of the source reduction, we conclude the thesis. □

Lemma 10 (Adequacy for $\gtrsim$)**.**

$$\begin{aligned} \text{if } & \varnothing; \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \gtrsim_n \mathbf{e} : \tau \\ & \mathbf{P} \triangleright \mathbf{e} \xRightarrow{\beta}{}^{\mathbf{m}} \mathbf{P} \triangleright \mathbf{e}'. \text{ with } n \geq m \\ \text{then } & \mathsf{P} \triangleright \mathsf{e} \xRightarrow{\beta} \mathsf{P} \triangleright \_ \end{aligned}$$

Proof. By Definition 46 (Logical relation for expressions) we have that $(W, \mathsf{e}, \mathbf{e}) \in \mathcal{E}\llbracket \tau \rrbracket_{\gtrsim}$ for a W such that $lev(W) = n$.

By taking $(W, [\cdot], [\cdot]) \in \mathcal{K}\llbracket \tau \rrbracket_{\gtrsim}$ we know that $(\mathsf{e}, \mathbf{e}) \in O(W)_{\gtrsim}$.

By definition of $O(\cdot)_{\gtrsim}$, with the HP of the target reduction, we conclude the thesis. □

Lemma 11 (Observation relation is closed under antireduction)**.**

$$\begin{aligned}
\text{if}\quad & \mathsf{P} \triangleright \mathsf{e} \xrightarrow{\beta}{}^{\mathsf{i}} \mathsf{P} \triangleright \mathsf{e}' \\
& \mathbf{P} \triangleright \mathbf{e} \xrightarrow{\beta}{}^{\mathbf{j}} \mathbf{P} \triangleright \mathbf{e}' \\
& (\mathsf{e}', \mathbf{e}') \in O(W')_{\triangledown} \text{ for } W' \sqsupseteq W \\
& \mathit{progs}(W) = \mathit{progs}(W') = (\mathsf{P}, \mathbf{P}) \\
& \mathit{lev}(W') \geq \mathit{lev}(W) - \mathtt{min}(i, j) \\
& (\text{ that is: } \mathit{lev}(W) \leq \mathit{lev}(W') + \mathtt{min}(i, j)) \\
\text{then}\quad & (\mathsf{e}, \mathbf{e}) \in O(W)_{\triangledown}
\end{aligned}$$

Proof. Trivial adaptation of the same proof in [?, ?]. □

Lemma 12 (Closedness under antireduction)**.**

$$\begin{aligned}
\text{if}\quad & \mathsf{P} \triangleright \mathbb{C}[\mathsf{e}] \xrightarrow{\beta}{}^{\mathsf{i}} \mathsf{P} \triangleright \mathbb{C}[\mathsf{e}'] \\
& \mathbf{P} \triangleright \mathbb{C}[\mathbf{e}] \xrightarrow{\beta}{}^{\mathbf{i}} \mathbf{P} \triangleright \mathbb{C}[\mathbf{e}'] \\
& (W', \mathsf{e}', \mathbf{e}') \in \mathcal{E}\,[\![\hat{\tau}]\!]_{\triangledown} \\
& W' \sqsupseteq W \\
& \mathit{lev}(W') \geq \mathit{lev}(W) - \mathtt{min}(i, j) \\
& (\text{ that is } \mathit{lev}(W) \leq \mathit{lev}(W') + \mathtt{min}(i, j)) \\
\text{then}\quad & (W, \mathsf{e}, \mathbf{e}) \in \mathcal{E}\,[\![\hat{\tau}]\!]_{\triangledown}
\end{aligned}$$

Proof. Trivial adaptation of the same proof in [?, ?]. □

Lemma 13 (Related terms plugged in related contexts are still related)**.**

$$\begin{aligned}
\text{if}\quad & (W, \mathsf{e}, \mathbf{e}) \in \mathcal{E}\left[\!\!\left[\hat{\tau}'\right]\!\!\right]_{\triangledown} \\
& \text{and if } W' \sqsupseteq W \\
& (W', \mathsf{v}, \mathbf{v}) \in \mathcal{V}\left[\!\!\left[\hat{\tau}'\right]\!\!\right]_{\triangledown} \\
& \text{then } (W', \mathbb{C}[\mathsf{v}], \mathbb{C}[\mathbf{v}]) \in \mathcal{E}\,[\![\hat{\tau}]\!]_{\triangledown} \\
\text{then}\quad & (W, \mathbb{C}[\mathsf{e}], \mathbb{C}[\mathbf{e}]) \in \mathcal{E}\,[\![\hat{\tau}]\!]_{\triangledown}
\end{aligned}$$

Proof. Trivial adaptation of the same proof in [?, ?]. □

Lemma 14 (Related functions applied to related arguments are related terms)**.**

$$\begin{aligned}
\text{if}\quad & (W, \mathsf{v}, \mathbf{v}) \in \mathcal{V}\left[\!\!\left[\hat{\tau}' \to \hat{\tau}\right]\!\!\right]_{\triangledown} \\
& (W, \mathsf{v}', \mathbf{v}') \in \mathcal{V}\left[\!\!\left[\hat{\tau}'\right]\!\!\right]_{\triangledown} \\
\text{then}\quad & (W, \mathsf{v}\ \mathsf{v}', \mathbf{v}\ \mathbf{v}') \in \mathcal{E}\,[\![\hat{\tau}]\!]_{\triangledown}
\end{aligned}$$

Proof. Trivial adaptation of the same proof in [?, ?]. □

Auxiliary Results

Lemma 15 (If Extract reduces, it preserves relatedness)**.**

$$\begin{aligned} \text{if} \quad & (W, \mathsf{v}, \mathbf{v}) \in \mathcal{V}\,[\![\mathsf{EmulTy}]\!]_{\triangledown} \\ & \mathsf{P} \triangleright \mathsf{extract}_{\sigma}(\mathsf{v}) \hookrightarrow^{*} \mathsf{P} \triangleright \mathsf{v}' \\ \text{then} \quad & (W, \mathsf{v}', \mathbf{v}) \in \mathcal{V}\,[\![\sigma]\!]_{\triangledown} \end{aligned}$$

Proof. Trivial case analysis:

$\sigma = \mathsf{Bool}$ means that $\mathsf{v}{=}0$ or 1, so by definition of $\mathcal{V}\,[\![\mathsf{EmulTy}]\!]_{\triangledown}$ $\mathbf{v}{=}\mathbf{false}$ or $\mathbf{true}$ (respectively).

Consider the 0 and $\mathbf{false}$ case, the other is analogous.

By definition the reduction of extract goes as follows.

$$\begin{aligned} & \mathsf{P} \triangleright \mathsf{extract}_{\mathsf{Bool}} 0 \\ \equiv & \mathsf{P} \triangleright \mathsf{let}\ \mathsf{x} = 0\ \mathsf{in}\ \mathsf{if}\ \mathsf{x} \geq 2\ \mathsf{then}\ \mathsf{fail}\ \mathsf{else}\ \mathsf{if}\ \mathsf{x}+1 \geq 2\ \mathsf{then}\ \mathsf{true}\ \mathsf{else}\ \mathsf{false} \\ \hookrightarrow \quad \hookrightarrow & \mathsf{P} \triangleright \mathsf{if}\ 1 \geq 2\ \mathsf{then}\ \mathsf{true}\ \mathsf{else}\ \mathsf{false} \\ \hookrightarrow & \mathsf{P} \triangleright \mathsf{false} \end{aligned}$$

We need to show that $(W, \mathsf{false}, \mathbf{false}) \in \mathcal{V}\,[\![\mathsf{Bool}]\!]_{\triangledown}$, which follows from its definition.

$\sigma = \mathsf{Nat}$ means that $\mathsf{v}{=}\mathsf{n}+2$ and $\mathbf{v}{=}\mathbf{n}$

By definition the reduction of extract goes as follows. (we write n+2 as a value, not as an expression to simplify this)

$$\begin{aligned} & \mathsf{P} \triangleright \mathsf{extract}_{\mathsf{Nat}} \mathsf{n}+2 \\ \equiv & \mathsf{P} \triangleright \mathsf{let}\ \mathsf{x} = \mathsf{n}+2\ \mathsf{in}\ \mathsf{if}\ \mathsf{x} \geq 2\ \mathsf{then}\ \mathsf{x}-2\ \mathsf{else}\ \mathsf{fail} \\ \hookrightarrow & \mathsf{P} \triangleright \mathsf{if}\ \mathsf{n}+2 \geq 2\ \mathsf{then}\ \mathsf{x}-2\ \mathsf{else}\ \mathsf{fail} \\ \hookrightarrow & \mathsf{P} \triangleright \mathsf{n} \end{aligned}$$

We need to show that $(W, \mathsf{n}, \mathbf{n}) \in \mathcal{V}\,[\![\mathsf{Nat}]\!]_{\triangledown}$, which follows from its definition.

□

Lemma 16 (Inject reduces and preserves relatedness)**.**

$$\begin{aligned} \text{if} \quad & (W, \mathsf{v}, \mathbf{v}) \in \mathcal{V}\,[\![\sigma]\!]_{\triangledown} \\ & \mathsf{P} \triangleright \mathsf{inject}_{\sigma} \mathsf{v} \hookrightarrow^{*} \mathsf{P} \triangleright \mathsf{v}' \\ \text{then} \quad & (W, \mathsf{v}', \mathbf{v}) \in \mathcal{V}\,[\![\mathsf{EmulTy}]\!]_{\triangledown} \end{aligned}$$

Proof. Trivial case analysis on σ.

$\sigma = \mathsf{Bool}$ By definition of $\mathcal{V}\llbracket\mathsf{Bool}\rrbracket_{\triangledown}$ we have $\mathsf{v}{=}\mathsf{true}$ and $\mathbf{v}{=}\mathbf{true}$ or $\mathsf{false}/\mathbf{false}$. We consider the first case only, the second is analogous.

By definition of inject we have:

$$\begin{aligned} &\mathsf{P} \triangleright \mathsf{if\ true\ then\ 1\ else\ 0} \\ \hookrightarrow\ &\mathsf{P} \triangleright \mathsf{1} \end{aligned}$$

So we need to prove that $(W, \mathsf{1}, \mathbf{true}) \in \mathcal{V}\llbracket\mathsf{EmulTy}\rrbracket_{\triangledown}$ which follows from its definition.

$\sigma = \mathsf{Nat}$ By definition of $\mathcal{V}\llbracket\mathsf{Nat}\rrbracket_{\triangledown}$ we have $\mathsf{v}{=}\mathsf{n}$ and $\mathbf{v}{=}\mathbf{n}$.

By definition of inject, we have:

$$\begin{aligned} &\mathsf{P} \triangleright \mathsf{n} + \mathsf{2} \\ \hookrightarrow\ &\mathsf{P} \triangleright \mathsf{n} + \mathsf{2} \end{aligned}$$

(we keep the value as a sum for simplicity)

So we need to prove that $(W, \mathsf{n} + \mathsf{2}, \mathbf{n}) \in \mathcal{V}\llbracket\mathsf{EmulTy}\rrbracket_{\triangledown}$ which follows from its definition.

□

Compatibility Lemmas for τ Types

Lemma 17 (Compatibility lemma for calls)**.**

$$\begin{aligned} \text{if}\ \ &\Gamma, \mathsf{x} : \sigma'; \mathsf{P}; \mathbf{P} \vdash \mathsf{e}\ \triangledown_n\ \mathbf{e} : \sigma \\ &\mathsf{f}(\mathsf{x} : \sigma') : \sigma \mapsto \mathsf{return\ e} \in \mathsf{P} \\ &\mathbf{f}(\mathbf{x}) \mapsto \mathbf{return\ if\ x\ has}\ \sigma'\ \mathbf{then\ e\ else\ fail} \in \mathbf{P} \\ \text{then}\ \ &\Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{call\ f}\ \triangledown_n\ \mathbf{call\ f} : \sigma' \to \sigma \end{aligned}$$

Proof. We need to prove that

$$\Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{call\ f}\ \triangledown_n\ \mathbf{call\ f} : \sigma' \to \sigma$$

Take W such that $lev(W) \leq n$ and HG $(W, \gamma, \gamma) \in \mathcal{G}\llbracket\mathtt{toEmul}\,(\mathbf{\Gamma})\rrbracket_{\triangledown}$, the thesis is:

- $(W, \mathsf{call\ f}, \mathbf{call\ f}) \in \mathcal{E}\llbracket\sigma' \to \sigma\rrbracket_{\triangledown}$

By Lemma 8 (Value relation implies term relation) the thesis is:

- $(W, \mathsf{call\ f}, \mathbf{call\ f}) \in \mathcal{V}\llbracket\sigma' \to \sigma\rrbracket_{\triangledown}$

By definition of the $\mathcal{V}\llbracket\cdot\rrbracket_{\triangledown}$ we take HV $(W', \mathsf{v}, \mathbf{v}) \in \mathcal{V}\llbracket\sigma'\rrbracket_{\triangledown}$ such that $W' \sqsupseteq_{\triangleright} W$ and the thesis is:

- $(W', \mathsf{return\ e}[\mathsf{v}/\mathsf{x}]\gamma, \mathbf{return\ if\ x\ has}\ \sigma'\ \mathbf{then\ e\ else\ fail}[\mathbf{v}/\mathbf{x}]\gamma) \in \mathcal{E}\llbracket\sigma\rrbracket_{\triangledown}$

The reductions proceed as:

$$
\begin{aligned}
&\mathbf{P} \triangleright \mathbf{return\ if\ x\ has\ } \sigma' \mathbf{\ then\ e\ else\ fail}[\mathbf{v}/\mathbf{x}]\gamma \\
\equiv &\mathbf{P} \triangleright \mathbf{return\ if\ v\ has\ } \sigma' \mathbf{\ then\ } (\mathbf{e}[\mathbf{v}/\mathbf{x}]\gamma) \mathbf{\ else\ fail} \\
\hookrightarrow\ &\mathbf{P} \triangleright \mathbf{return\ if\ true\ then\ } (\mathbf{e}[\mathbf{v}/\mathbf{x}]\gamma) \mathbf{\ else\ fail} \\
\hookrightarrow\ &\mathbf{P} \triangleright \mathbf{return\ } (\mathbf{e}[\mathbf{v}/\mathbf{x}]\gamma)
\end{aligned}
$$

By Lemma 12 the thesis becomes:

- $(W', \mathsf{return\ e}[\mathsf{v}/\mathsf{x}]\gamma, \mathbf{return\ e}[\mathbf{v}/\mathbf{x}]\gamma) \in \mathcal{E}\,[\![\sigma]\!]_{\triangledown}$

This follows from the definition of logical relation if

- $(W', [\mathsf{v}/\mathsf{x}]\gamma, [\mathbf{v}/\mathbf{x}]\gamma) \in \mathcal{G}\,[\![\Gamma, \mathsf{x} : \sigma']\!]_{\triangledown}$

This follows from HG with Lemma 5 and by HV and Lemma 7 and by the definition of $\mathcal{G}\,[\![\cdot]\!]_{\triangledown}$. □

Lemma 18 (Compatibility lemma for application)**.**

$$
\begin{aligned}
\text{if}\ \ &\Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \triangledown_n \mathbf{e} : \sigma' \to \sigma \\
&\Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{e}' \triangledown_n \mathbf{e}' : \sigma' \\
\text{then}\ \ &\Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{e\ e}' \triangledown_n \mathbf{e\ e}' : \sigma
\end{aligned}
$$

Proof. This is standard using Lemma 8, Lemma 7, Lemma 13 and Lemma 12. □

Lemma 19 (Compatibility lemma for op)**.**

$$
\begin{aligned}
\text{if}\ \ &\Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \triangledown_n \mathbf{e} : \mathsf{Nat} \\
&\Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{e}' \triangledown_n \mathbf{e}' : \mathsf{Nat} \\
\text{then}\ \ &\Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \oplus \mathsf{e}' \triangledown_n \mathbf{e} \oplus \mathbf{e}' : \mathsf{Nat}
\end{aligned}
$$

Proof. This is standard and analogous to the proof of Lemma 18. □

Lemma 20 (Compatibility lemma for geq)**.**

$$
\begin{aligned}
\text{if}\ \ &\Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \triangledown_n \mathbf{e} : \mathsf{Nat} \\
&\Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{e}' \triangledown_n \mathbf{e}' : \mathsf{Nat} \\
\text{then}\ \ &\Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \geq \mathsf{e}' \triangledown_n \mathbf{e} \geq \mathbf{e}' : \mathsf{Bool}
\end{aligned}
$$

Proof. This is standard and analogous to the proof of Lemma 18. □

Lemma 21 (Compatibility lemma for letin)**.**

$$
\begin{aligned}
\text{if}\ \ &\Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \triangledown_n \mathbf{e} : \sigma \\
&\Gamma, \mathsf{x} : \sigma; \mathsf{P}; \mathbf{P} \vdash \mathsf{e}' \triangledown_n \mathbf{e}' : \sigma' \\
\text{then}\ \ &\Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{let\ x = e\ in\ e}' \triangledown_n \mathbf{let\ x = e\ in\ e}' : \sigma'
\end{aligned}
$$

Proof. This is standard and analogous to the proof of Lemma 18. □

Lemma 22 (Compatibility lemma for if)**.**

$$\begin{aligned} \text{if} \quad & \Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \; \nabla_n \; \mathbf{e} : \mathsf{Bool} \\ & \Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{e}' \; \nabla_n \; \mathbf{e}' : \sigma \\ & \Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{e}'' \; \nabla_n \; \mathbf{e}'' : \sigma \\ \text{then} \quad & \Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{if\ e\ then\ e'\ else\ e''} \; \nabla_n \; \mathbf{if\ e\ then\ e'\ else\ e''} : \sigma \end{aligned}$$

Proof. This is standard and analogous to the proof of Lemma 18. □

Lemma 23 (Compatibility lemma for read)**.**

$$\begin{aligned} \text{if} \quad & \\ \text{then} \quad & \Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{read} \; \nabla_n \; \mathbf{read} : \mathsf{Nat} \end{aligned}$$

Proof. By definition of the $O(W)_{\triangledown}$. □

Lemma 24 (Compatibility lemma for write)**.**

$$\begin{aligned} \text{if} \quad & \Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \; \nabla_n \; \mathbf{e} : \mathsf{Nat} \\ \text{then} \quad & \Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{write\ e} \; \nabla_n \; \mathbf{write\ e} : \mathsf{Nat} \end{aligned}$$

Proof. We need to prove that

$$\Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{write\ e} \; \nabla_n \; \mathbf{write\ e} : \mathsf{Nat}$$

Take W such that $lev(W) \leq n$ and $(W, \gamma, \gamma) \in \mathcal{G}\,[\![\mathtt{toEmul}\,(\mathbf{\Gamma})]\!]_{\triangledown}$, the thesis is: (we omit substitutions as they don't play an active role)

- $(W, \mathsf{write\ e}, \mathbf{write\ e}) \in \mathcal{E}\,[\![\mathsf{Nat}]\!]_{\triangledown}$

By Lemma 13 (Related terms plugged in related contexts are still related) with HE, we have that for HW $W' \sqsupseteq W$, and HV $(W', \mathsf{n}, \mathbf{n}) \in \mathcal{V}\,[\![\mathsf{Nat}]\!]_{\triangledown}$, the thesis becomes:

- $(W', \mathsf{write\ n}, \mathbf{write\ n}) \in \mathcal{E}\,[\![\mathsf{Nat}]\!]_{\triangledown}$

The reductions proceed as:

$$\mathsf{P} \triangleright \mathsf{write\ n} \xRightarrow{\mathtt{write\ n}} \mathsf{P} \triangleright \mathsf{n}$$

and

$$\mathbf{P} \triangleright \mathbf{write\ n} \xRightarrow{\mathtt{write\ n}} \mathbf{P} \triangleright \mathbf{n}$$

By Lemma 12 (Closedness under antireduction) the thesis is:

- $(W', \mathsf{n}, \mathbf{n}) \in \mathcal{E}\,[\![\mathsf{Nat}]\!]_{\triangledown}$

So the theorem holds by Lemma 8 (Value relation implies term relation) with HV. □

Semantic Preservation Results

Theorem 43 ($[\![\cdot]\!]^{\mathsf{L}^{\tau}}_{\mathbf{L^u}}$ is semantics preserving for expressions)**.**

$$\begin{aligned} \text{if} \quad & \mathsf{P}; \Gamma \vdash \mathsf{e} : \tau \\ & \vdash \mathsf{P} \bigtriangledown_n \mathbf{P} \\ \text{then} \quad & \forall n.\ \Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \bigtriangledown_n [\![\mathsf{e}]\!]^{\mathsf{L}^{\tau}}_{\mathbf{L^u}} : \tau \end{aligned}$$

Proof. The proof proceeds by induction on the type derivation.

true, false, nat By definition of $\mathcal{V}[\![\cdot]\!]_{\bigtriangledown}$.

var By definition of $\mathcal{G}[\![\cdot]\!]_{\bigtriangledown}$.

call By Lemma 17 (Compatibility lemma for calls).

app By IH with Lemma 18 (Compatibility lemma for application).

op By IH with Lemma 19 (Compatibility lemma for op).

geq By IH with Lemma 20 (Compatibility lemma for geq).

letin By IH with Lemma 21 (Compatibility lemma for letin).

if By IH with Lemma 22 (Compatibility lemma for if).

read By Lemma 23 (Compatibility lemma for read).

write By IH with Lemma 24 (Compatibility lemma for write).

□

Theorem 44 ($[\![\cdot]\!]^{\mathsf{L}^{\tau}}_{\mathbf{L^u}}$ is semantics preserving for programs)**.**

$$\begin{aligned} \text{if} \quad & \vdash \mathsf{P} \\ \text{then} \quad & \vdash \mathsf{P} \bigtriangledown [\![\mathsf{P}]\!]^{\mathsf{L}^{\tau}}_{\mathbf{L^u}} \end{aligned}$$

Proof. By induction on the size of P and then Rule ($[\![\cdot]\!]^{\mathsf{L}^{\tau}}_{\mathbf{L^u}}$-Prog) and with Theorem 43 ($[\![\cdot]\!]^{\mathsf{L}^{\tau}}_{\mathbf{L^u}}$ is semantics preserving for expressions) on each function body. □

Compatibility Lemmas for Pseudo Types

Lemma 25 (Compatibility lemma for backtranslation of op)**.**

$$\begin{aligned} \text{if} \quad & (HE)\ \texttt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \bigtriangledown_n \mathbf{e} : \mathsf{EmulTy} \\ & (HEP)\ \texttt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{e}' \bigtriangledown_n \mathbf{e}' : \mathsf{EmulTy} \\ \text{then} \quad & \texttt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \begin{aligned}[t] & \mathsf{let\ x1 : Nat{=}extract_{Nat}(e)} \\ & \mathsf{in\ let\ x2 : Nat{=}extract_{Nat}(e')} \\ & \mathsf{in\ inject_{Nat}(x1 \oplus x2)} \end{aligned} \quad \bigtriangledown_n \mathbf{e} \oplus \mathbf{e}' : \mathsf{EmulTy} \end{aligned}$$

Proof. We need to prove that

$$\texttt{toEmul}\,(\boldsymbol{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \begin{array}{l}\mathsf{let}\ \mathsf{x1} : \mathsf{Nat}{=}\mathsf{extract}_{\mathsf{Nat}}(\mathsf{e})\\ \mathsf{in}\ \mathsf{let}\ \mathsf{x2} : \mathsf{Nat}{=}\mathsf{extract}_{\mathsf{Nat}}(\mathsf{e}')\\ \mathsf{in}\ \mathsf{inject}_{\mathsf{Nat}}(\mathsf{x1} \oplus \mathsf{x2})\end{array} \quad \nabla\ \mathbf{e} \oplus \mathbf{e}' : \mathsf{EmulTy}$$

Take W such that $lev(W) \leq n$ and $(W, \gamma, \boldsymbol{\gamma}) \in \mathcal{G}\left[\!\left[\texttt{toEmul}\,(\boldsymbol{\Gamma})\right]\!\right]_{\nabla}$, the thesis is:

- $(W, \begin{array}{l}\mathsf{let}\ \mathsf{x1} : \mathsf{Nat}{=}\mathsf{extract}_{\mathsf{Nat}}(\mathsf{e})\\ \mathsf{in}\ \mathsf{let}\ \mathsf{x2} : \mathsf{Nat}{=}\mathsf{extract}_{\mathsf{Nat}}(\mathsf{e}')\\ \mathsf{in}\ \mathsf{inject}_{\mathsf{Nat}}(\mathsf{x1} \oplus \mathsf{x2})\end{array}, \mathbf{e} \oplus \mathbf{e}') \in \mathcal{E}\left[\!\left[\mathsf{EmulTy}\right]\!\right]_{\nabla}$

By Lemma 13 (Related terms plugged in related contexts are still related) with HE we need to prove that $\forall W' \sqsupseteq W$, given IHV $(W', \mathsf{v}, \mathbf{v}) \in \mathcal{V}\left[\!\left[\mathsf{EmulTy}\right]\!\right]_{\nabla}$

- $(W', \begin{array}{l}\mathsf{let}\ \mathsf{x1} : \mathsf{Nat}{=}\mathsf{extract}_{\mathsf{Nat}}(\mathsf{v})\\ \mathsf{in}\ \mathsf{let}\ \mathsf{x2} : \mathsf{Nat}{=}\mathsf{extract}_{\mathsf{Nat}}(\mathsf{e}')\\ \mathsf{in}\ \mathsf{inject}_{\mathsf{Nat}}(\mathsf{x1} \oplus \mathsf{x2})\end{array}, \mathbf{v} \oplus \mathbf{e}') \in \mathcal{E}\left[\!\left[\mathsf{EmulTy}\right]\!\right]_{\nabla}$

By IHV we perform a case analysis on $\mathbf{v}$:

- $\mathbf{true}$/ $\mathbf{false}$ and thus v is $1/0$ respectively.

  We show the case for $\mathbf{true}$, 1 the other is analogous.

  In this case we have:

  $$\mathbf{P} \triangleright \mathbf{true} \oplus \mathbf{e}' \overset{\perp}{\Longrightarrow} \mathbf{fail}$$

  and

  $$\begin{aligned}&\mathsf{P} \triangleright \mathsf{extract}_{\mathsf{Nat}}(1)\\ \equiv&\mathsf{let}\ \mathsf{x} = 1\ \mathsf{in}\ \mathsf{if}\ \mathsf{x} \geq 2\ \mathsf{then}\ \mathsf{x} - 2\ \mathsf{else}\ \mathsf{fail}\\ \hookrightarrow&\ \mathsf{if}\ 1 \geq 2\ \mathsf{then}\ \mathsf{x} - 2\ \mathsf{else}\ \mathsf{fail}\\ \overset{\perp}{\Longrightarrow}&\ \mathsf{fail}\end{aligned}$$

  So this case follows from the definition of $O(W')_{\nabla}$ as both terms perform the same visible action ($\perp$).

- $\mathbf{n}$ and thus v is $\mathsf{n} + 2$.

  In this case we have:

  $$\begin{aligned}&\mathsf{P} \triangleright \mathsf{extract}_{\mathsf{Nat}}(\mathsf{n} + 2)\\ \equiv&\mathsf{let}\ \mathsf{x} = \mathsf{n} + 2\ \mathsf{in}\ \mathsf{if}\ \mathsf{x} \geq 2\ \mathsf{then}\ \mathsf{x} - 2\ \mathsf{else}\ \mathsf{fail}\\ \hookrightarrow&\ \mathsf{if}\ \mathsf{n} + 2 \geq 2\ \mathsf{then}\ \mathsf{x} - 2\ \mathsf{else}\ \mathsf{fail}\\ \hookrightarrow&\ \mathsf{n}\end{aligned}$$

And by Lemma 15 (If Extract reduces, it preserves relatedness) with IHV we know that IHN $(W', \mathsf{n}, \mathbf{n}) \in \mathcal{V}\,[\![\mathsf{Nat}]\!]_{\triangledown}$.

Analogously, e' and $\mathbf{e}'$ follow the same treatment. So we apply Lemma 13 (Related terms plugged in related contexts are still related) with HEP, perform a case analysis, in one case they fail and in the other they reduce to $\mathsf{n}'/\mathbf{n}'$ such that IHNP $(W', \mathsf{n}', \mathbf{n}') \in \mathcal{V}\,[\![\mathsf{Nat}]\!]_{\triangledown}$.

So the reductions are :

$$
\begin{aligned}
&\mathsf{P}\triangleright\mathsf{let\ x1 : Nat{=}extract_{Nat}(e)\ in\ let\ x2 : Nat{=}extract_{Nat}(e')}\\
&\quad\mathsf{in\ inject_{Nat}(x1 \oplus x2)}\\
\hookrightarrow^*\ &\mathsf{P}\triangleright\mathsf{let\ x1 : Nat{=}extract_{Nat}(n)\ in\ let\ x2 : Nat{=}extract_{Nat}(e')}\\
&\quad\mathsf{in\ inject_{Nat}(x1 \oplus x2)}\\
\hookrightarrow\ &\mathsf{P}\triangleright\mathsf{let\ x2 : Nat{=}extract_{Nat}(e')}\\
&\quad\mathsf{in\ inject_{Nat}(n \oplus x2)}\\
\hookrightarrow^*\ &\mathsf{P}\triangleright\mathsf{let\ x2 : Nat{=}extract_{Nat}(n')}\\
&\quad\mathsf{in\ inject_{Nat}(n \oplus x2)}\\
\hookrightarrow\ &\mathsf{P} \triangleright \mathsf{inject_{Nat}(n \oplus n')}
\end{aligned}
$$

and

$$\mathbf{P} \triangleright \mathbf{e} \oplus \mathbf{e}' \ \hookrightarrow^* \mathbf{P} \triangleright \mathbf{n} \oplus \mathbf{e}' \ \hookrightarrow^* \mathbf{P} \triangleright \mathbf{n} \oplus \mathbf{n}'$$

By Lemma 12 (Closedness under antireduction) the thesis becomes:

– $(W', \mathsf{inject_{Nat}(n \oplus n')}, \mathbf{n} \oplus \mathbf{n}') \in \mathcal{E}\,[\![\mathsf{EmulTy}]\!]_{\triangledown}$

If the $\mathit{lev}(W') = 0$ the thesis follows from Lemma 3 (No steps means relation), otherwise:

By Rule EL$^{\tau}$-op and Rule $\mathbf{EL^{u}}$-op we can apply Lemma 12 (Closedness under antireduction) (with IHN and IHNP in the term relation by Lemma 8 (Value relation implies term relation)) and the thesis becomes:

– $(W', \mathsf{inject_{Nat}(n'')}, \mathbf{n}'') \in \mathcal{E}\,[\![\mathsf{EmulTy}]\!]_{\triangledown}$

The reductions proceed as follows:

$$\mathsf{P} \triangleright \mathsf{inject_{Nat}(n'')} \ \hookrightarrow\ \mathsf{P} \triangleright \mathsf{n}'' + 2$$

By Lemma 12 (Closedness under antireduction) and then Lemma 8 (Value relation implies term relation) the thesis becomes:

– $(W', \mathsf{n}''' + 2, \mathbf{n}'') \in \mathcal{V}\,[\![\mathsf{EmulTy}]\!]_{\triangledown}$

By Lemma 16 (Inject reduces and preserves relatedness) the thesis becomes:

– $(W', \mathsf{n}'', \mathbf{n}'') \in \mathcal{V}[\![\mathsf{Nat}]\!]_{\triangledown}$

which follows from the definition of $\mathcal{V}[\![\mathsf{Nat}]\!]_{\triangledown}$.

□

Lemma 26 (Compatibility lemma for backtranslation of geq).

$$\begin{aligned}
\text{if } & \texttt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \triangledown_n \mathbf{e} : \mathsf{EmulTy} \\
& \texttt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{e}' \triangledown_n \mathbf{e}' : \mathsf{EmulTy} \\
\text{then } & \texttt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{let\ x1 : Nat{=}extract_{Nat}(e)} \quad \triangledown_n \mathbf{e} \geq \mathbf{e}' : \mathsf{EmulTy} \\
& \qquad \mathsf{in\ let\ x2 : Nat{=}extract_{Nat}(e')} \\
& \qquad \mathsf{in\ inject_{Bool}(x1 \geq x2)}
\end{aligned}$$

Proof. Analogous to the proof of Lemma 25. □

Lemma 27 (Compatibility lemma for backtranslation of letin).

$$\begin{aligned}
\text{if } & \texttt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \triangledown_n \mathbf{e} : \mathsf{EmulTy} \\
& \texttt{toEmul}\,(\mathbf{\Gamma}), \mathsf{x} : \mathsf{Nat}; \mathsf{P}; \mathbf{P} \vdash \mathsf{e}' \triangledown_n \mathbf{e}' : \mathsf{EmulTy} \\
\text{then } & \texttt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{let\ x : Nat = e\ in\ e'} \triangledown_n \mathbf{let\ x = e\ in\ e'} : \mathsf{EmulTy}
\end{aligned}$$

Proof. This is a trivial application of Lemma 13 (Related terms plugged in related contexts are still related) and Lemma 12 (Closedness under antireduction) and definitions. □

Lemma 28 (Compatibility lemma for backtranslation of if).

$$\begin{aligned}
\text{if } & (HE)\ \texttt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \triangledown_n \mathbf{e} : \mathsf{EmulTy} \\
& (HEP)\ \texttt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{e}' \triangledown_n \mathbf{e}' : \mathsf{EmulTy} \\
& \texttt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{e}'' \triangledown_n \mathbf{e}'' : \mathsf{EmulTy} \\
\text{then } & \texttt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{if\ extract_{Bool}(e)\ then\ e'\ else\ e''} \triangledown_n \mathbf{if\ e\ then\ e'\ else\ e''} : \mathsf{EmulTy}
\end{aligned}$$

Proof. We need to prove that

$$\texttt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{if\ extract_{Bool}(e)\ then\ e'\ else\ e''} \triangledown \mathbf{if\ e\ then\ e'\ else\ e''} : \mathsf{EmulTy}$$

Take W such that $lev(W) \leq n$ and $(W, \gamma, \gamma) \in \mathcal{G}[\![\texttt{toEmul}\,(\mathbf{\Gamma})]\!]_{\triangledown}$, the thesis is: (we omit substitutions as they don't play an active role)

- $(W, \mathsf{if\ extract_{Bool}(e)\ then\ e'\ else\ e''}, \mathbf{if\ e\ then\ e'\ else\ e''}) \in \mathcal{E}[\![\mathsf{EmulTy}]\!]_{\triangledown}$

By Lemma 13 (Related terms plugged in related contexts are still related) with HE, we have that for HW $W' \sqsupseteq W$, and HV $(W', \mathsf{v}, \mathbf{v}) \in \mathcal{V}[\![\mathsf{EmulTy}]\!]_{\triangledown}$, the thesis becomes:

- $(W', \mathsf{if\ extract_{Bool}(v)\ then\ e'\ else\ e''}, \mathbf{if\ v\ then\ e'\ else\ e''}) \in \mathcal{E}[\![\mathsf{EmulTy}]\!]_{\triangledown}$

We perform a case analysis based on HV:

- $\mathbf{v}=\mathbf{true}/\mathbf{false}$ and $\mathsf{v}=1/0$

  We consider the case $\mathbf{true}/1$ the other is analogous.

  The reductions proceed as follows:

$$\begin{aligned}
&\mathsf{P} \triangleright \mathsf{extract}_{\mathtt{Bool}}(1)\\
\equiv&\mathsf{P} \triangleright \mathsf{let}\ \mathsf{x} = 1\ \mathsf{in}\ \mathsf{if}\ \mathsf{x} \geq 2\ \mathsf{then}\ \mathsf{fail}\ \mathsf{else}\ \mathsf{if}\ \mathsf{x}+1 \geq 2\ \mathsf{then}\ \mathsf{true}\ \mathsf{else}\ \mathsf{false}\\
\hookrightarrow&\ \mathsf{P} \triangleright \mathsf{if}\ 1 \geq 2\ \mathsf{then}\ \mathsf{fail}\ \mathsf{else}\ \mathsf{if}\ 1+1 \geq 2\ \mathsf{then}\ \mathsf{true}\ \mathsf{else}\ \mathsf{false}\\
\hookrightarrow&\ \mathsf{P} \triangleright \mathsf{if}\ 1+1 \geq 2\ \mathsf{then}\ \mathsf{true}\ \mathsf{else}\ \mathsf{false}\\
\hookrightarrow\quad\hookrightarrow&\ \mathsf{P} \triangleright \mathsf{true}
\end{aligned}$$

  By Lemma 12 (Closedness under antireduction) the thesis becomes:

  – $(W', \mathsf{if\ true\ then\ e'\ else\ e''}, \mathbf{if\ true\ then\ e'\ else\ e''}) \in \mathcal{E}\,[\![\mathsf{EmulTy}]\!]_{\triangledown}$

  If the $lev(W') = 0$ the thesis follows from Lemma 3 (No steps means relation), otherwise:

  We can reduce based on Rules EL^{τ}-if-true and $\mathbf{EL^{u}}$-if-true. By Lemma 12 (Closedness under antireduction) the thesis becomes:

  – $(W', \mathsf{e'}, \mathbf{e'}) \in \mathcal{E}\,[\![\mathsf{EmulTy}]\!]_{\triangledown}$

  If the $lev(W') = 0$ the thesis follows from Lemma 3 (No steps means relation), otherwise by HEP.

- $\mathbf{v}=\mathbf{n}$ and $\mathsf{v}=\mathsf{n}+2$

  In this case we have that:

$$\begin{aligned}
&\mathsf{P} \triangleright \mathsf{extract}_{\mathtt{Bool}}(\mathsf{n}+2)\\
\equiv&\mathsf{P} \triangleright \mathsf{let}\ \mathsf{x} = \mathsf{n}+2\ \mathsf{in}\ \mathsf{if}\ \mathsf{x} \geq 2\ \mathsf{then}\ \mathsf{fail}\ \mathsf{else}\ \mathsf{if}\ \mathsf{x}+1 \geq 2\ \mathsf{then}\ \mathsf{true}\ \mathsf{else}\ \mathsf{false}\\
\hookrightarrow&\ \mathsf{P} \triangleright \mathsf{if}\ \mathsf{n}+2 \geq 2\ \mathsf{then}\ \mathsf{fail}\ \mathsf{else}\ \mathsf{if}\ \mathsf{x}+1 \geq 2\ \mathsf{then}\ \mathsf{true}\ \mathsf{else}\ \mathsf{false}\\
\stackrel{\perp}{\Longrightarrow}&\ \mathsf{fail}
\end{aligned}$$

  and

$$\mathbf{P} \triangleright \mathbf{if\ n\ then\ e'\ else\ e''} \stackrel{\perp}{\Longrightarrow} \mathbf{fail}$$

  So this case holds by definition of $O(W')_{\triangledown}$.

$\square$

Lemma 29 (Compatibility lemma for backtranslation of application)**.**

$$\begin{aligned}
\text{if}\quad & \mathtt{toEmul}(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \triangledown_n \mathbf{e} : \mathsf{EmulTy}\\
& \mathsf{f}(\mathsf{x} : \sigma') : \sigma \mapsto \mathsf{return\ e} \in \mathsf{P}\\
& (HP)\ \mathsf{P}; \mathbf{P} \vdash \mathsf{call\ f} \triangledown_n \mathbf{call\ f} : \sigma' \to \sigma\\
\text{then}\quad & \mathtt{toEmul}(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{inject}_{\tau'}(\mathsf{call\ f\ extract}_{\tau}(\mathsf{e})) \triangledown_n \mathbf{call\ f\ e} : \mathsf{EmulTy}
\end{aligned}$$

Proof. We need to prove that

$$\texttt{toEmul}\,(\boldsymbol{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{inject}_{\tau'}(\mathsf{call\ f\ extract}_{\tau}(\mathsf{e}))\ \nabla_n\ \mathbf{call\ f\ e} : \mathsf{EmulTy}$$

Take W such that $lev(W) \leq n$ and $(W, \gamma, \gamma) \in \mathcal{G}\,[\![\texttt{toEmul}\,(\boldsymbol{\Gamma})]\!]_{\triangledown}$, the thesis is: (we omit substitutions as they don't play an active role)

- $(W, \mathsf{inject}_{\tau'}(\mathsf{call\ f\ extract}_{\tau}(\mathsf{e})), \mathbf{call\ f\ e}) \in \mathcal{E}\,[\![\mathsf{EmulTy}]\!]_{\triangledown}$

By Lemma 13 (Related terms plugged in related contexts are still related) with HE we have that for HW $W' \sqsupseteq W$, and HV $(W', \mathsf{v}, \mathbf{v}) \in \mathcal{V}\,[\![\mathsf{EmulTy}]\!]_{\triangledown}$, the thesis becomes:

- $(W, \mathsf{inject}_{\tau'}(\mathsf{call\ f\ extract}_{\tau}(\mathsf{v})), \mathbf{call\ f\ v}) \in \mathcal{E}\,[\![\mathsf{EmulTy}]\!]_{\triangledown}$

We perform a case analysis based on HV:

- $\mathbf{v}{=}\mathbf{true}/\mathbf{false}$ and $\mathsf{v}{=}\mathsf{1}/\mathsf{0}$ (respectively).

  We consider the first case only, the other is analogous.

  We perform a case analysis on τ:

  - $\tau{=}\mathsf{Bool}$

    The thesis is:

    * $(W', \mathsf{inject}_{\tau'}(\mathsf{call\ f\ extract}_{\mathsf{Bool}}(\mathsf{v})), \mathbf{call\ f\ v}) \in \mathcal{E}\,[\![\mathsf{EmulTy}]\!]_{\triangledown}$

    By definition of $\mathsf{extract}_{\mathbf{Bool}}$ we have

    $$\begin{aligned} & \mathsf{P} \triangleright \mathsf{inject}_{\tau'}(\mathsf{call\ f\ extract}_{\mathsf{Bool}}(1)) \\ \equiv & \mathsf{P} \triangleright \mathsf{inject}_{\tau'}(\mathsf{call\ f\ let\ x} = 1\ \mathsf{in\ if\ x} \geq 2\ \mathsf{then\ fail\ else\ if\ x} + 1 \geq 2\ \mathsf{then\ true\ else\ false}) \\ \hookrightarrow & \mathsf{P} \triangleright \mathsf{inject}_{\tau'}(\mathsf{call\ f\ if}\ 1 \geq 2\ \mathsf{then\ fail\ else\ if}\ 1 + 1 \geq 2\ \mathsf{then\ true\ else\ false}) \\ \hookrightarrow & \mathsf{P} \triangleright \mathsf{inject}_{\tau'}(\mathsf{call\ f\ if}\ 1 + 1 \geq 2\ \mathsf{then\ true\ else\ false}) \\ \hookrightarrow & \mathsf{P} \triangleright \mathsf{inject}_{\tau'}(\mathsf{call\ f\ true}) \end{aligned}$$

    So by Lemma 12 (Closedness under antireduction) the thesis becomes:

    * $(W', \mathsf{inject}_{\tau'}(\mathsf{call\ f\ true}), \mathbf{call\ f\ true}) \in \mathcal{E}\,[\![\mathsf{EmulTy}]\!]_{\triangledown}$

    If the $lev(W') = 0$ the thesis follows from Lemma 3 (No steps means relation), otherwise:

    By HP and by the Hs on the function bodies, and by the relatedness of true and $\mathbf{true}$ and by the Lemma 7 (Monotonicity for value relation) we have that HF:

    $$(W', \mathsf{return\ e}[\mathsf{true}/\mathsf{x}], \mathbf{return\ e}[\mathbf{true}/\mathbf{x}]) \in \mathcal{E}\left[\!\!\left[\hat{\tau'}\right]\!\!\right]_{\triangledown}$$

    By Lemma 13 (Related terms plugged in related contexts are still related) with HF we have that for HW $W'' \sqsupseteq W'$, and HV $(W'', \mathsf{v}', \mathbf{v}') \in \mathcal{V}\,[\![\tau']\!]_{\triangledown}$, the thesis becomes:

* $(W', \mathsf{inject}_{\tau'}(\mathsf{v}'), \mathbf{v}') \in \mathcal{E}\,[\![\mathsf{EmulTy}]\!]_{\triangledown}$

  This case follows from Lemma 8 (Value relation implies term relation) and by Lemma 16 (Inject reduces and preserves relatedness) with HV.

- $\tau = \mathsf{Nat}$

  By definition of $\mathsf{extract}_{\mathsf{Nat}}$ we have:

$$
\begin{aligned}
& \mathsf{P} \triangleright \mathsf{inject}_{\tau'}(\mathsf{call\ f\ extract}_{\mathsf{Nat}}(1)) \\
\equiv & \mathsf{P} \triangleright \mathsf{inject}_{\tau'}(\mathsf{call\ f\ let\ x} = 1 \mathsf{\ in\ if\ x} \geq 2 \mathsf{\ then\ x} - 2 \mathsf{\ else\ fail}) \\
\hookrightarrow\ & \mathsf{P} \triangleright \mathsf{inject}_{\tau'}(\mathsf{call\ f\ if\ } 1 \geq 2 \mathsf{\ then\ } 1 - 2 \mathsf{\ else\ fail}) \\
\hookrightarrow\ & \mathsf{P} \triangleright \mathsf{inject}_{\tau'}(\mathsf{call\ f\ fail}) \\
\hookrightarrow\ & \mathsf{fail}
\end{aligned}
$$

  and by definition of the function bodies and Rule ($[\![\cdot]\!]^{\mathsf{L}^{\tau}}_{\mathbf{L^u}}$-Fun):

$$
\begin{aligned}
& \mathbf{P} \triangleright \mathbf{call\ f\ true} \\
\hookrightarrow\ & \mathbf{P} \triangleright \mathbf{return\ if\ true\ has\ } [\![\mathsf{Nat}]\!]^{\mathsf{L}^{\tau}}_{\mathbf{L^u}} \mathbf{\ then\ } [\![\mathsf{e}]\!]^{\mathsf{L}^{\tau}}_{\mathbf{L^u}} \mathbf{\ else\ fail} \\
\equiv & \mathbf{P} \triangleright \mathbf{return\ if\ true\ has\ } \mathbb{N} \mathbf{\ then\ } [\![\mathsf{e}]\!]^{\mathsf{L}^{\tau}}_{\mathbf{L^u}} \mathbf{\ else\ fail} \\
\hookrightarrow\ & \mathbf{P} \triangleright \mathbf{return\ if\ false\ then\ } [\![\mathsf{e}]\!]^{\mathsf{L}^{\tau}}_{\mathbf{L^u}} \mathbf{\ else\ fail} \\
\hookrightarrow\ & \mathbf{P} \triangleright \mathbf{return\ fail} \\
\hookrightarrow\ & \mathbf{fail}
\end{aligned}
$$

  So this case holds by definition of $O(W')_{\triangledown}$.

- $\mathbf{v}=\mathbf{n}$ and $\mathsf{v}=\mathsf{n}+2$

  Case analysis on τ

  - $\tau=\mathsf{Bool}$

    This is analogous to the case for naturals above.

  - $\tau = \mathsf{Nat}$

    This is analogous to the case for booleans above.

∎

Lemma 30 (Compatibility lemma for backtranslation of check)**.**

$$
\begin{aligned}
\text{if}\quad & (HE)\ \mathtt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \triangledown_n \mathbf{e} : \mathsf{EmulTy} \\
\text{then}\ 1\ & \mathtt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{let\ x : Nat = e\ in\ if\ x} \geq 2 \mathsf{\ then\ } 0 \mathsf{\ else\ } 1 \triangledown_n \mathbf{e\ has\ Bool} : \mathsf{EmulTy} \\
2\ & \mathtt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{let\ x : Nat = e\ in\ if\ x} \geq 2 \mathsf{\ then\ } 1 \mathsf{\ else\ } 0 \triangledown_n \mathbf{e\ has\ } \mathbb{N} : \mathsf{EmulTy}
\end{aligned}
$$

Proof. We need to prove that

$$
\begin{aligned}
1\ & \mathtt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{let\ x : Nat = e\ in\ if\ x} \geq 2 \mathsf{\ then\ } 0 \mathsf{\ else\ } 1 \triangledown_n \mathbf{e\ has\ Bool} : \mathsf{EmulTy} \\
2\ & \mathtt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{let\ x : Nat = e\ in\ if\ x} \geq 2 \mathsf{\ then\ } 1 \mathsf{\ else\ } 0 \triangledown_n \mathbf{e\ has\ } \mathbb{N} : \mathsf{EmulTy}
\end{aligned}
$$

We only show case 1, the other is analogous.

Take W such that $lev(W) \leq n$ and $(W, \gamma, \gamma) \in \mathcal{G}\llbracket \texttt{toEmul}(\mathbf{\Gamma}) \rrbracket_{\triangledown}$, the thesis is: (we omit substitutions as they don't play an active role)

1. $(W, \mathsf{let\ x : Nat = e\ in\ if\ x \geq 2\ then\ 0\ else\ 1}, \mathbf{e\ has\ Bool}) \in \mathcal{E}\llbracket \mathsf{EmulTy} \rrbracket_{\triangledown}$

By Lemma 13 (Related terms plugged in related contexts are still related) with HE we have that for HW $W' \sqsupseteq W$, and HV $(W', \mathsf{v}, \mathbf{v}) \in \mathcal{V}\llbracket \mathsf{EmulTy} \rrbracket_{\triangledown}$, the thesis becomes:

- $(W', \mathsf{let\ x : Nat = v\ in\ if\ x \geq 2\ then\ 0\ else\ 1}, \mathbf{v\ has\ Bool}) \in \mathcal{E}\llbracket \mathsf{EmulTy} \rrbracket_{\triangledown}$

We perform a case analysis based on HV:

- $\mathbf{v}{=}\mathbf{true}/\mathbf{false}$ and $\mathsf{v}{=}1/0$ (respectively).

  We consider only the first case, the other is analogous.

  We have that

  $$\begin{aligned} &\mathsf{P} \triangleright \mathsf{let\ x : Nat = 1\ in\ if\ x \geq 2\ then\ 0\ else\ 1} \\ \hookrightarrow\ &\mathsf{P} \triangleright \mathsf{if\ 1 \geq 2\ then\ 0\ else\ 1} \\ \hookrightarrow\ &\mathsf{P} \triangleright 1 \end{aligned}$$

  and

  $$\mathbf{P} \triangleright \mathbf{true\ has\ Bool} \ \hookrightarrow\ \mathbf{P} \triangleright \mathbf{true}$$

  This case holds by Lemma 12 (Closedness under antireduction) and Lemma 8 (Value relation implies term relation) and by the definition of $\mathcal{V}\llbracket \mathsf{EmulTy} \rrbracket_{\triangledown}$.

- $\mathbf{v}{=}\mathbf{n}$ and $\mathsf{v}{=}\mathsf{n}+2$

  In this case we have that:

  $$\begin{aligned} &\mathsf{P} \triangleright \mathsf{let\ x : Nat = n + 2\ in\ if\ x \geq 2\ then\ 0\ else\ 1} \\ \hookrightarrow\ &\mathsf{P} \triangleright \mathsf{if\ n + 2 \geq 2\ then\ 0\ else\ 1} \\ \hookrightarrow\ &\mathsf{P} \triangleright 0 \end{aligned}$$

  and

  $$\mathbf{P} \triangleright \mathbf{n\ has\ Bool} \ \hookrightarrow\ \mathbf{P} \triangleright \mathbf{false}$$

  This case holds by Lemma 12 (Closedness under antireduction) and Lemma 8 (Value relation implies term relation) and by the definition of $\mathcal{V}\llbracket \mathsf{EmulTy} \rrbracket_{\triangledown}$.

$\square$

Semantic Preservation of Backtranslation

Theorem 45 ($\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}$ is semantics preserving)**.**

$$\begin{aligned} \text{if } & \mathbf{\Gamma} \vdash \mathbf{e} \\ & (HP) \ \vdash \mathsf{P} \mathrel{\triangledown} \mathbf{P} \\ \text{then } & \texttt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \langle\!\langle \mathbf{e} \rangle\!\rangle \mathrel{\triangledown}_n \mathbf{e} : \mathsf{EmulTy} \end{aligned}$$

Proof. The proof preoceeds by induction on the derivation of $\mathbf{\Gamma} \vdash \mathbf{e}$.

Base cases true,false,nat By definition of the $\mathcal{V}\,[\![\mathsf{EmulTy}]\!]_{\triangledown}$

var By definition of the $\mathcal{G}\,[\![\cdot]\!]_{\triangledown}$.

call This case cannot arise.

Inductive cases app By IH and HP and Lemma 29 (Compatibility lemma for backtranslation of application).

op By IH and Lemma 25 (Compatibility lemma for backtranslation of op).

geq Analogous to the case above.

if By IH and Lemma 28 (Compatibility lemma for backtranslation of if).

letin By IH and Lemma 27 (Compatibility lemma for backtranslation of letin).

check By IH and Lemma 30 (Compatibility lemma for backtranslation of check).

□

Theorems that Yield RRHP

Theorem 46 ($[\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}$ preserves behaviors)**.**

$$\begin{aligned} \text{if } & (HT) \ [\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright \mathbf{e} \overset{\beta}{\Longrightarrow} [\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright \mathbf{e}' \\ \text{then } & \mathsf{P} \triangleright \langle\!\langle \mathbf{e} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} \overset{\beta}{\Longrightarrow} \mathsf{P} \triangleright \mathsf{e}' \end{aligned}$$

Proof. By Theorem 44 ($[\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}$ is semantics preserving for programs) we have HPP:

- $\vdash \mathsf{P} \mathrel{\triangledown} [\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}$

Given that $\varnothing \vdash \mathbf{e}$, by Theorem 45 ($\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}$ is semantics preserving) with HPP we have HPE:

- $\texttt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; [\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \vdash \langle\!\langle \mathbf{e} \rangle\!\rangle \mathrel{\triangledown}_n \mathbf{e} : \mathsf{EmulTy}$

The thesis follows by Lemma 10 (Adequacy for $\gtrsim$) with HT. □

Theorem 47 ($[\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}$ reflects behaviors)**.**

$$\begin{aligned}\text{if } (HS)\ \mathsf{P} \triangleright \langle\!\langle \mathbf{e} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} &\overset{\beta}{\Longrightarrow} \mathsf{P} \triangleright \mathsf{e}' \\ \text{then } [\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright \mathbf{e} &\overset{\beta}{\Longrightarrow} [\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright \mathbf{e}'\end{aligned}$$

Proof. By Theorem 44 ($[\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}$ is semantics preserving for programs) we have HPP:

- $\vdash \mathsf{P} \triangledown [\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}$

Given that $\varnothing \vdash \mathbf{e}$, by Theorem 45 ($\langle\!\langle \cdot \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}$ is semantics preserving) with HPP we have HPE:

- $\texttt{toEmul}\,(\mathbf{\Gamma}); \mathsf{P}; [\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \vdash \langle\!\langle \mathbf{e} \rangle\!\rangle \ \triangledown_n\ \mathbf{e} : \mathsf{EmulTy}$

The thesis follows by Lemma 9 (Adequacy for $\lesssim$) with HS. ☐

8.4.3 Proof That $[\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}$ Satisfies Definition 20 (RrHC)

$$\begin{aligned}&\forall \mathbf{e}. \exists \mathsf{e}. \forall \mathsf{P}, \beta \\ &\qquad [\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright \mathbf{e} \overset{\beta}{\Longrightarrow} [\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright \mathbf{e}' \\ &\iff \mathsf{P} \triangleright \mathsf{e} \overset{\beta}{\Longrightarrow} \mathsf{P} \triangleright \mathsf{e}'\end{aligned}$$

We instantiate e with $\langle\!\langle \mathbf{e} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}$ then two cases arise.

$\Rightarrow$ **direction** By Theorem 46 ($[\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}$ preserves behaviors)

$\Leftarrow$ **direction** By Theorem 47 ($[\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}$ reflects behaviors).

8.5 Proof That $[\![\cdot]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}$ Is RFrSP

This section focuses on giving a high-level overview the proof technique that we use to prove that our compiler satisfies the criterion *robust finite-relational safety preservation*. The proof shows that for any k, the compiler satisfy *robust k-relational safety preservation*.

8.5.1 Overview of the Proof Technique

We have proved the following theorem for our instance:

Theorem 48 (k-Relational Robust Safety Preservation)**.** Let $\mathsf{P}_1 \ldots \mathsf{P}_k$ be k programs that share the same interface $\bar{\mathsf{I}}$ and $m_1 \ldots m_k$ be k finite trace prefixes. Then, for all target contexts $\mathbf{C_T}$, the following holds:

$$\begin{aligned}&\left(\forall i, \mathbf{C_T}\left[[\![\mathsf{P}_i]\!]^{\mathsf{S}}_{\mathbf{T}}\right] \rightsquigarrow m_i\right) \\ \Longrightarrow\ &(\exists \mathsf{C_S}, \forall i, \mathsf{C_S}[\mathsf{P}_i] \rightsquigarrow m_i)\end{aligned}$$

$$
\begin{tikzcd}
\forall i, \mathsf{C_S}[\mathsf{P}_i] \rightsquigarrow m_i & \forall i, \mathsf{C_S}[\mathsf{P}_i] \hookrightarrow \mu_i^s \arrow[l, "\text{Uninf}"'] & \forall i, \mathsf{P}_i \hookrightarrow_{\text{prg}} \mu_i \arrow[l, "\text{Comp}"'] & \forall i, \mathsf{C_S} \hookrightarrow_{\text{ctx}} \mu_i^s \arrow[ll, bend right, "\text{Comp}"'] \\
\forall i, \mathbf{C_T}\left[\llbracket \mathsf{P}_i \rrbracket_{\mathbf{T}}^{\mathsf{S}}\right] \rightsquigarrow m_i \arrow[r, "\text{Inf}"] & \forall i, \exists \mu_i \sqsupseteq m_i, \mathbf{C_T}\left[\llbracket \mathsf{P}_i \rrbracket_{\mathbf{T}}^{\mathsf{S}}\right] \hookrightarrow \mu_i \arrow[r, "\text{Dec}"] \arrow[rr, bend right, "\text{Dec}"'] & \forall i, \llbracket \mathsf{P}_i \rrbracket_{\mathbf{T}}^{\mathsf{S}} \hookrightarrow_{\text{prg}} \mu_i \arrow[u, "\text{BCC}"] & \forall i, \mathbf{C_T} \hookrightarrow_{\text{ctx}} \mu_i \arrow[u, "\text{BT}"]
\end{tikzcd}
$$

Figure 1: Proposed proof technique

Our proof technique for this is described in Figure 1. At the heart of this technique is the back-translation of a finite set of finite trace prefixes into a source context. In particular, this back-translation technique do not inspect the code of the target context. The first steps consist in transforming the trace prefixes into prefixes that can be back-translated easily, and separating the target context from the compiled programs. Then, we build a back-translation that provides us with a source context that can be composed with the initial source programs to generate the initial traces.

The reason for requiring all programs to share the same interface is that it allows us to produce a well-typed context. Otherwise, two programs could contain the same function, but one returning a natural number and the other a boolean. If these two functions would be called in different branches of the context, that could end up being badly typed.

8.5.2 Informative Traces

The first step of the proof is to augment the existing operational semantics with new events that allow to precisely track the behavior of the program and of the context. This new semantics are called *informative semantics* and produce *informative traces*. They are defined at both the source level and the target level. The relations $\hookrightarrow$ are the equivalent of $\rightsquigarrow$ for these informative semantics, and is defined as:

$$\mathsf{C}[\mathsf{P}] \hookrightarrow \mu \iff \exists \mathsf{e}, \mathsf{P} \triangleright \mathsf{C} \overset{\mu}{\Longrightarrow\!\!\!\!\Rightarrow} \mathsf{P} \triangleright \mathsf{e}$$

$$\mathbf{C}[\mathbf{P}] \hookrightarrow \mu \iff \exists \mathbf{e}, \mathbf{P} \triangleright \mathbf{C} \overset{\mu}{\Longrightarrow\!\!\!\!\Rightarrow} \mathbf{P} \triangleright \mathbf{e}$$

We can state the theorem for passing to informative traces as follow

Theorem 49 (Informative traces)**.** Let $\mathbf{C_T}$ be a target context and $\mathbf{P_T}$ a target program. Then,

$$\forall m, \mathbf{C_T}[\mathbf{P_T}] \rightsquigarrow m \implies \exists \mu \sqsupseteq m, \mathbf{C_T}[\mathbf{P_T}] \hookrightarrow \mu$$

where

$$\mu \sqsupseteq m \iff |\mu|_{\text{I/O/termination}} = m.$$

Proof. Let $\mathbf{C_T}$ be a target context, $\mathbf{P_T}$ a target program and m a finite prefix. We are going to show that if there exists $\mathbf{e}$ such that $\mathbf{P_T} \triangleright \mathbf{C_T} \overset{\mathbf{m}}{\Longrightarrow} \mathbf{P_T} \triangleright \mathbf{e}$, then there exists μ such that $|\mu|_{\text{I/O}} = m$ and $\mathbf{P_T} \triangleright \mathbf{C_T} \overset{\mu}{\Longrightarrow\!\!\!\!\!\Rightarrow} \mathbf{P_T} \triangleright \mathbf{e}$.

Let us proceed by induction on the relation $\mathbf{P_T} \triangleright \mathbf{C_T} \overset{\mathbf{m}}{\Longrightarrow} \mathbf{P_T} \triangleright \mathbf{e}$.

Rule $\mathbf{EL^u}$**-refl** Immediate.

Rule $\mathbf{EL^u}$**-terminate** This is true by taking $\mu = \Downarrow$, because the informative semantics can progress if and only if the non-informative semantics can.

Rule $\mathbf{EL^u}$**-diverge** This is true by taking $\mu = \Uparrow$, because the informative semantics can only diverge when executing the program part (the context can not loop or do recursion), and calls from the program part do not generate any event.

Rule $\mathbf{EL^u}$**-silent** Then $\mathbf{P_T} \triangleright \mathbf{C_T} \xrightarrow{\epsilon} \mathbf{P_T} \triangleright \mathbf{e}$ according to the non-informative semantics. Since the semantics only differ on the events that are generated, we have two cases. Either $\mathbf{P_T} \triangleright \mathbf{C_T} \xrightarrow{\epsilon} \mathbf{P_T} \triangleright \mathbf{e}$ according to the informative semantics, in which case we can take $\mu = \epsilon$. Or $\mathbf{P_T} \triangleright \mathbf{C_T} \xrightarrow{\alpha} \mathbf{P_T} \triangleright \mathbf{e}$ according to the informative semantics, in which case we can take $\mu = \alpha$. This α must be a call or return event by definition of the informative semantics, hence the result.

Rule $\mathbf{EL^u}$**-single** Since $\mathbf{P_T} \triangleright \mathbf{C_T} \xrightarrow{\alpha} \mathbf{P_T} \triangleright \mathbf{e}$ according to the non-informative semantics, this is also the case according to the informative semantics, hence the result.

Rule $\mathbf{EL^u}$**-cons** Then $\mathbf{P_T} \triangleright \mathbf{C_T} \overset{\mathbf{m_1}}{\Longrightarrow} \mathbf{P_T} \triangleright \mathbf{e'}$ and $\mathbf{P_T} \triangleright \mathbf{e'} \overset{\mathbf{m_2}}{\Longrightarrow} \mathbf{P_T} \triangleright \mathbf{e}$ with $m = m_1 m_2$. By applying the induction hypothesis, there exists μ_1 and μ_2 such that $\mathbf{P_T} \triangleright \mathbf{C_T} \overset{\mu_1}{\Longrightarrow\!\!\!\!\!\Rightarrow} \mathbf{e'}$, $\mathbf{P_T} \triangleright \mathbf{e'} \overset{\mu_2}{\Longrightarrow\!\!\!\!\!\Rightarrow} \mathbf{e}$, $|\mu_1|_{\text{I/O/termination}} = m_1$, and $|\mu_2|_{\text{I/O/termination}} = m_2$.

Therefore by applying Rule $\mathbf{EL^u}$-cons, $\mathbf{P_T} \triangleright \mathbf{C_T} \overset{\mu_1 \mu_2}{\Longrightarrow\!\!\!\!\!\Rightarrow} \mathbf{e}$. It is easy to see that $|\mu_1 \mu_2|_{\text{I/O/termination}} = m_1 m_2$. We are done.

$\square$

8.5.3 Decomposition

This decomposition step relies on the definition of *partial semantics*, one for programs and one for contexts. These partial semantics describe the possible behaviors of a program in any context and of a context with respect to any program. Partial semantics can often be defined by abstracting away one part of the whole program (the context for the partial semantics of programs, and the program for the partial semantics of contexts), by introducing non-determinism for modeling the abstracted part.

We index our relations by either "ctx" or "prg" to denote the partial semantics. The partial semantics for contexts defined as:

$$\frac{(\mathsf{EL}^{\tau}\text{-ctx-call})}{\mathsf{call\ f\ v} \xrightarrow{\mathtt{call\ f\ v?}}_{\mathrm{ctx}} \mathsf{return\ e}} \qquad \frac{(\mathbf{EL^{u}}\text{-ctx-call})}{\mathbf{call\ f\ v} \xrightarrow{\mathtt{call\ \mathbf{f}\ \mathbf{v}?}}_{\mathrm{ctx}} \mathbf{return\ e}}$$

$$\frac{(\mathsf{EL}^{\tau}\text{-ctx-ret})}{\mathsf{return\ v} \xrightarrow{\epsilon}_{\mathrm{ctx}} \mathsf{v}} \qquad \frac{(\mathbf{EL^{u}}\text{-ctx-ret})}{\mathbf{return\ v} \xrightarrow{\epsilon}_{\mathrm{ctx}} \mathbf{v}}$$

and the relations $\overset{\cdot}{\Longrightarrow\!\!\!\!\!\!\twoheadrightarrow}_{\mathrm{ctx}}$ and $\overset{\cdot}{\Longrightarrow\!\!\!\!\!\!\twoheadrightarrow}_{\mathrm{ctx}}$ are defined in the same manner as the complete semantics.

The partial semantics for programs are defined in terms of the complete semantics, and are parameterized by the interface of the program $\bar{\mathsf{I}}$. Informally, we define $P \hookrightarrow_{\mathrm{prg}} \mu$ to mean that the program P is able to produce each part of the trace μ that comes from the program, i.e. each part that starts with a call event $\mathtt{call}\ f\ v?$ and ends before or with the corresponding return event, when it is put into the context that simply calls this function f with this value v. For every "subtrace" μ' of μ starting with a call event $\mathtt{call}\ f\ v?$ and stopping at the latest at the next (corresponding) return event, it must be that $P \rhd \mathit{call}\ f\ v \hookrightarrow \mu'$.

Definition 48 (Partial semantics for programs)**.** $\mathsf{P} \hookrightarrow_{\mathrm{prg}} \mu$ if and only if:

- for any trace $\mu_{f,v,v'} = \mathtt{call}\ f\ v?; \mu'; \mathtt{ret}\ v'!$ such that $\mu = \mu_1; \mu_{f,v,v'}; \mu_2$, such that there is no event $\mathit{return}\ \ldots$ in μ', and such that $\mathsf{f} : \tau \to \tau' \in \bar{\mathsf{I}}$ with $\mathsf{v} \in \tau$, we have

$$\mathsf{P_T} \rhd \mathsf{call\ f\ v} \overset{\mu_{\mathsf{f,v,v'}}}{\Longrightarrow\!\!\!\!\!\!\twoheadrightarrow} \mathsf{P} \rhd \mathsf{v}';$$

- for any trace $\mu_{f,v} = \mathtt{call}\ f\ v?; \mu'$ such that $\mu = \mu_1; \mu_{f,v}$, such that there is no event $\mathit{return}\ \ldots$ in μ', and such that $\mathsf{f} : \tau \to \tau' \in \bar{\mathsf{I}}$ with $\mathsf{v} \in \tau$, there exists e such that

$$\mathsf{P_T} \rhd \mathsf{call\ f\ v} \overset{\mu_{\mathsf{f,v}}}{\Longrightarrow\!\!\!\!\!\!\twoheadrightarrow} \mathsf{P} \rhd \mathsf{e}.$$

$\mathbf{P} \hookrightarrow_{\mathrm{prg}} \mu$ if and only if:

- for any trace $\mu_{f,v,v'} = \mathtt{call}\ f\ v?; \mu'; \mathtt{ret}\ v'!$ such that $\mu = \mu_1; \mu_{f,v,v'}; \mu_2$, such that there is no event $\mathit{return}\ \ldots$ in μ', and such that $\mathbf{f} \in \bar{\mathsf{I}}$ we have

$$\mathbf{P_T} \rhd \mathbf{call\ f\ v} \overset{\mu_{\mathbf{f,v,v'}}}{\Longrightarrow\!\!\!\!\!\!\twoheadrightarrow} \mathbf{P} \rhd \mathbf{v}';$$

- for any trace $\mu_{f,v} = \texttt{call}\ f\ v?; \mu'$ such that $\mu = \mu_1; \mu_{f,v}$, such that there is no event *return* ... in μ', and such that $\mathbf{f} \in \bar{\mathsf{I}}$ there exists $\mathbf{e}$ such that

$$\mathbf{P_T} \triangleright \mathbf{call\ f\ v} \xRightarrow{\mu_{\mathbf{f,v}}} \mathbf{P} \triangleright \mathbf{e}.$$

We must restrict this definition to the well-typed calls in the source level: indeed, a badly-typed call does not make sense in the source language.

Our decomposition theorem talks about both programs and contexts:

Theorem 50 (Decomposition)**.** Let $\mathbf{C_T}$ be a target context and $\mathbf{P_T}$ a target program. Then,

$$\forall \mu, \mathbf{C_T}\left[\mathbf{P_T}\right] \hookrightarrow \mu \implies \mathbf{C_T} \hookrightarrow_{\mathrm{ctx}} \mu \wedge \mathbf{P_T} \hookrightarrow_{\mathrm{prg}} \mu$$

We are going to prove two different lemmas, one for contexts and one for programs.

Lemma 31. Let $\mathbf{C_T}$ be a target context and $\mathbf{P_T}$ be a target program, μ an informative trace and $\mathbf{e}$ a target expression. Then,

$$\mathbf{C_T}\left[\mathbf{P_T}\right] \xRightarrow{\mu} \mathbf{e} \implies \mathbf{C_T} \xRightarrow{\mu}_{\mathrm{ctx}} \mathbf{e}$$

Proof. By induction on the relation $\mathbf{C_T}[\mathbf{P_T}] \xRightarrow{\mu} \mathbf{P_T} \triangleright \mathbf{e}$

Rule EL$^{\mathbf{u}}$-silent Therefore $\mathbf{C_T}[\mathbf{P_T}] \xrightarrow{\epsilon} \mathbf{P_T} \triangleright \mathbf{e}$. By case analysis, it is also the case that $\mathbf{C_T} \xrightarrow{\epsilon}_{\mathrm{ctx}} \mathbf{e}$ hence the result.

Rule EL$^{\mathbf{u}}$-action $\mathbf{C_T}[\mathbf{P_T}] \xrightarrow{\alpha} \mathbf{P_T} \triangleright \mathbf{e}$. We proceed by case analysis on this relation: if α is an I/O operation, correct termination or failure event, then we indeed have $\mathbf{C_T} \xrightarrow{\alpha}_{\mathrm{ctx}} \mathbf{e}$.

Otherwise, $\alpha = \Uparrow$. Therefore, $\forall n, \exists \mathbf{e_n}, \mathbf{C_T}[\mathbf{P_T}] \xrightarrow{\epsilon}^{\mathbf{n}} \mathbf{P_T} \triangleright \mathbf{e_n}$. Now, by induction on n, we can prove that $\forall n, \exists \mathbf{e_n}, \mathbf{C_T} \xrightarrow{\epsilon}_{\mathrm{ctx}}{}^{\mathbf{n}} \mathbf{e_n}$. Hence the result.

Rule EL$^{\mathbf{u}}$-single Then $\mathbf{C_T}[\mathbf{P_T}] \xrightarrow{\beta} \mathbf{P_T} \triangleright \mathbf{e}$. We proceed by case analysis on this relation:

- If $\beta = \texttt{call}\ f\ v?$, then $\mathbf{C_T} = \mathbb{E}\left[\mathbf{call\ f\ v}\right]$ and $\mathbf{e} = \mathbb{E}\left[\mathbf{return\ e'}\right]$ for some evaluation context $\mathbb{E}$ and some expression $\mathbf{e'}$. Therefore, $\mathbf{e} \xrightarrow{\texttt{call f v?}}_{\mathrm{ctx}} \mathbb{E}\left[\mathbf{return\ e'}\right]$ by the partial semantics, hence the result.
- If $\beta = \texttt{ret}\ f!v$, then $\mathbf{C_T} = \mathbb{E}\left[\mathbf{return\ v}\right]$ for some evaluation context $\mathbb{E}$. Therefore, $\mathbf{e} \xrightarrow{\texttt{ret f!v}}_{\mathrm{ctx}} \mathbb{E}\left[\mathbf{v}\right]$ according to the partial semantics, hence the result.

Rule EL$^{\mathbf{u}}$-cons We have that $\mathbf{P_T} \triangleright \mathbf{C_t} \xRightarrow{\mu_1}_{\mathrm{ctx}} \mathbf{e'}$ and $\mathbf{P_T} \triangleright \mathbf{e'} \xRightarrow{\mu_2}_{\mathrm{ctx}} \mathbf{e}$. Then, by applying the induction hypothesis to the two relations, we are done.

□

Then, we prove a similar lemma for programs:

Lemma 32. Let $\mathbf{P_T}$ be a target program, $\mathbf{C_T}$ a target context and μ an informative trace. Suppose that $\mathbf{C_T}[\mathbf{P_T}] \hookrightarrow \mu$. Then:

- for any trace $\mu_{f,v,v'} = \texttt{call } f\ v?;\mu';\texttt{ret } v'!$ such that $\mu = \mu_1;\mu_{f,v,v'};\mu_2$ and such that there is no event *return* ... in μ', $\mathbf{P_T} \triangleright \mathbf{call\ f\ v} \xRightarrow{\mu_{\mathbf{f,v,v'}}} \mathbf{v'}$
- for any trace $\mu_{f,v} = \texttt{call } f\ v?;\mu'$ such that $\mu = \mu_1;\mu_{f,v}$ and such that there is no event *return* ... in μ', there exists $\mathbf{e}$ such that $\mathbf{P_T} \triangleright \mathbf{call\ f\ v} \xRightarrow{\mu_{\mathbf{f,v}}} \mathbf{e}$.

Proof. Consider the first case for instance. From the fact that $\mu_{f,v,v'}$ appears in μ, we can deduce the fact that there exists an evaluation context $\mathbb{E}$ such that $\mathbf{P} \triangleright \mathbb{E}[\mathbf{call\ f\ v}] \xRightarrow{\mu_{f,v,v'}}\!\!\!\!\!\!\Rightarrow_{\mathrm{ctx}} \mathbb{E}[\mathbf{v'}]$.

From this, we can reason by induction and use Rule $\mathbf{EL^u}$-ctx to obtain the result. □

8.5.4 Backward Compiler Correctness for Programs

Theorem 51 (Backward Compiler Correctness)**.** Let P be a source program. Then,

$$\forall \mu, [\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \hookrightarrow_{\mathrm{prg}} \mu \implies \mathsf{P} \hookrightarrow_{\mathrm{prg}} \mu.$$

Before proving the theorem, we state a preliminary lemma:

Lemma 33. Suppose that $[\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright \mathbf{call\ f\ v} \xRightarrow{\texttt{call f v}?;\mu}\!\!\!\!\!\!\Rightarrow [\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright \mathbf{e'}$ where the call is well-typed.

Then, $[\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright \mathbf{call\ f\ v} \xRightarrow{\texttt{call f v}?}\!\!\!\!\!\!\Rightarrow [\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright [\![\mathsf{e}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}[\mathbf{x}/\mathbf{v}]$ and:

- $[\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright [\![\mathsf{e}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}[\mathbf{x}/\mathbf{v}] \xRightarrow{\mu}\!\!\!\!\!\!\Rightarrow [\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright \mathbf{e'}$,
- or, $\mu = \epsilon$ and $[\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright \mathbf{call\ f\ v} \xRightarrow{\texttt{call f v}?}\!\!\!\!\!\!\Rightarrow [\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright \mathbf{e'}$

where e is the code of the function f in the source program.

Proof. By induction on $[\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright \mathbf{call\ f\ v} \xRightarrow{\texttt{call f v}?;\mu}\!\!\!\!\!\!\Rightarrow \mathbf{e'}$.

Rule $\mathbf{EL^u}$-single In this case, $\mu = \epsilon$. The result is obtained by direct application of the semantics.

Rule $\mathbf{EL^u}$-cons There exists μ_1 and μ_2 such that $\mu_1\mu_2 = \mu$ and

$$[\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright \mathbf{call\ f\ v} \xRightarrow{\texttt{call f v}?;\mu_1}\!\!\!\!\!\!\Rightarrow \mathbf{e_1}$$

and

$$[\![\mathsf{P}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright \mathbf{e_1} \xRightarrow{\mu_2}\!\!\!\!\!\!\Rightarrow \mathbf{e}.$$

By applying the induction hypothesis to the first relation, we obtain the result.

Other cases: these cases are impossible

$\square$

We can now prove the backward compiler correctness theorem:

Theorem 51 (Backward Compiler Correctness)**.** Let P be a source program. Then,

$$\forall \mu, [\![\mathsf{P}]\!]^{\mathsf{L}^{\tau}}_{\mathbf{L^u}} \hookrightarrow_{\mathrm{prg}} \mu \implies \mathsf{P} \hookrightarrow_{\mathrm{prg}} \mu.$$

Proof. Let P be a source program and μ an informative trace. Suppose that $[\![\mathsf{P}]\!]^{\mathsf{L}^{\tau}}_{\mathbf{L^u}} \hookrightarrow_{\mathrm{prg}} \mu$, we will prove that $\mathsf{P} \hookrightarrow_{\mathrm{prg}} \mu$.

Let $\mu_{f,v,v'} = \texttt{call}\ f\ v?; \mu'; \texttt{ret}\ v'!$ be a trace as defined by the source partial semantics. Let us show that

$$\mathsf{P} \triangleright \mathsf{call\ f\ v} \xRightarrow{\mu_{\mathsf{f,v,v'}}} \mathsf{v}',$$

knowing that

$$[\![\mathsf{P}]\!]^{\mathsf{L}^{\tau}}_{\mathbf{L^u}} \triangleright \mathbf{call\ f\ v} \xRightarrow{\mu_{\mathbf{f,v,v'}}} \mathbf{v}'.$$

By the preliminary lemma, and since $\mu' \neq \epsilon$, we have that

$$[\![\mathsf{P}]\!]^{\mathsf{L}^{\tau}}_{\mathbf{L^u}} \triangleright \mathbf{call\ f\ v} \xRightarrow{\texttt{call f v?}} [\![\mathsf{e}]\!]^{\mathsf{L}^{\tau}}_{\mathbf{L^u}}[\mathbf{x}/\mathbf{v}]$$

where e is the source of f in the source program, because the call is well-typed and $[\![\mathsf{P}]\!]^{\mathsf{L}^{\tau}}_{\mathbf{L^u}} \triangleright [\![\mathsf{e}]\!]^{\mathsf{L}^{\tau}}_{\mathbf{L^u}}[\mathbf{x}/\mathbf{v}] \xRightarrow{\mu';\texttt{ret v}'!} \mathbf{v}'$.

Now, we can conclude by induction on e.

$\square$

8.5.5 Back-Translation of a Finite Set of Finite Trace Prefixes

The theorem we wish to prove in this section is the following theorem:

Theorem 52. Let $\mathbf{C_T}$ be a target context and $\{\mu_i\}$ be a finite set of trace prefixes such that $\forall i, \mathbf{C_T} \hookrightarrow_{\mathrm{ctx}} \mu_i$. Then,

$$\exists \mathsf{C_S}, \forall i, \mathsf{C_S} \hookrightarrow_{\mathrm{ctx}} \mu_i^s$$

where the relation between μ_i and μ_i^s is explicited later.

We will construct a function $\uparrow$ such that if F is a set of finite prefixes, $F\uparrow$ is a source context such that:

$$\forall \mu \in F, F\uparrow \hookrightarrow_{\mathrm{ctx}} \mu^s.$$

where μ^s, defined later, is the trace μ with the possibility of swapping failure and calls events, as described previously.

We only consider traces that do not have any I/O. Indeed, I/O is produced only by the programs in these languages, hence do not affect the backtranslation of a source context. First, we explicit the tree structure that is found in F by defining the following inductive construction:

$$T ::= \epsilon \mid \Downarrow \mid \bot \mid \Uparrow \\ \mid (\texttt{call } f\ v?, (v_1, T_1), (v_2, T_2), \ldots, (v_i, T_i))$$

From a set of trace F, we define a relation $F \vDash T$ as follow:

$$\frac{F = \varnothing \vee \forall \mu \in F, \mu = \epsilon}{F \vDash \epsilon}\ \text{(Tree-Empty)} \qquad \frac{\forall \mu \in F, \mu \neq \epsilon \implies \mu = \Downarrow}{F \vDash \Downarrow}\ \text{(Tree-Term)}$$

$$\frac{\forall \mu \in F, \mu \neq \epsilon \implies \mu = \Uparrow}{F \vDash \Uparrow}\ \text{(Tree-Divr)} \qquad \frac{\forall \mu \in F, \mu \neq \epsilon \implies \mu = \bot}{F \vDash \bot}\ \text{(Tree-Fail)}$$

$$\frac{\forall \mu \in F, \mu \neq \epsilon \implies \mu = \texttt{call } f\ v?; \mu' \wedge \mathsf{f} : \tau \to \tau' \wedge v \notin \tau}{F \vDash \bot}\ \text{(Tree-Fail-Type)}$$

$$\frac{\begin{array}{c}\forall i, \exists \mu \in F, \mu = \texttt{call } f\ v?; \texttt{ret } v_i!; \mu' \\ \{\mu' \mid \texttt{call } f\ v?; \texttt{ret } v_i!; \mu' \in F\} \vDash T_i \\ \bigcup_{1 \leq j \leq i} \{\texttt{call } f\ v?; \texttt{ret } v_j!; \mu' \in F\} \cup \{\texttt{call } f\ v?; \Uparrow\} \cup \{\texttt{call } f\ v?\} \cup \{\epsilon\} \supseteq F\end{array}}{F \vDash (\texttt{call } f\ v?, (v_1, T_1), \ldots, (v_i, T_i))}\ \text{(Tree-Call-Ret)}$$

This relation means that the tree T represents the set of traces F. The first five rules represent the base cases from the point of view of the context: Rule Tree-Empty is the case where every trace is empty or there are no trace in F. Rule Tree-Term represent the case where all traces terminate. Rule Tree-Divr is a case that should never happen, because the context should never diverge. Rule Tree-Fail is the case where all traces fail in the context. Rule Tree-Fail-Type represent the case where all traces call a function with an incorrect argument and must fail.

The last rule, Rule Tree-Call-Ret, represent the case where some traces may be cut, and the others shall call a function. The next event must be either divergence, which is ignored because it is part of the program, or a return event. Then, the remaining traces are separated into groups receiving the same return value: these traces are then considered on their own to construct subtrees T_i. The third condition is required to ensure that no trace is forgotten.

The fact that this object is indeed defined is directly derived from the determinacy of the context. Indeed, let F be a set of informative traces produced by the same context. They must either be empty, or start by the same event, by determinacy, and this event has to be a call event. If this call in not correctly typed, then we are in the fifth case. Otherwise, we are necessarily in the last case, and the T_i exist by induction.

The back-translation of F is defined by induction on the tree T such that $F \vDash T$:

Definition 49 (Backtranslation of the tree T)**.**

$$T\uparrow = \begin{cases} \mathsf{fail} & \text{if } T = \epsilon \text{ or } T = \bot \\ \mathsf{0} & \text{if } T = \Downarrow \\ \mathsf{fail} & \text{if } T = \Uparrow \\ \mathsf{let\ x = call\ f\ v\ in} \begin{pmatrix} \mathsf{if\ x = v_1\ then\ T_1\uparrow} \\ \mathsf{else\ if\ x = v_2\ then\ \ldots} \\ \mathsf{else\ if\ x = v_i\ then\ T_i\uparrow\ else\ fail} \end{pmatrix} & \begin{array}{l} \text{if } T = \\ (\texttt{call}\ f\ v?, (v_1, T_1), \ldots, (v_i, T_i)) \\ \text{and } \mathsf{f} : \tau \rightarrow \tau' \text{ and } v \in \tau \end{array} \\ \mathsf{fail} & \text{otherwise} \end{cases}$$

Lemma 34. The back-translation of a set of traces F generated by a single context is well-typed.

Proof. By induction on the relation $F \vDash T$. □

We define what it means for a trace to be "part" of such a tree:

Definition 50 (Trace extract from a tree)**.** We say that a trace μ is extracted from a tree T if:

1. $\mu = \epsilon$
2. $\mu = \Downarrow$ and $T = \Downarrow$
3. $\mu = \bot$ and $T = \bot$
4. $\mu = \texttt{call}\ f\ v? :: \epsilon$, $\text{type}(v) \neq \text{input\_type}(f)$ and $T = \bot$
5. $\mu = \texttt{call}\ f\ v? :: \bot$, $\text{type}(v) \neq \text{input\_type}(f)$ and $T = \bot$
6. $\mu = \texttt{call}\ f\ v? :: \epsilon$ or $\mu = \texttt{call}\ f\ v?; \Uparrow$, $T = (\texttt{call}\ f\ v?, \ldots)$ and $\text{type}(v) = \text{input\_type}(f)$
7. $\mu = \texttt{call}\ f\ v? :: \texttt{ret}\ v'! :: \mu'$, $T = (\texttt{call}\ f\ v?, (v_1, T_1), \ldots, (v_i, T_i))$, and $\exists j$, such that $v_j = v'$ and μ' is extracted from T_j
8. $\mu = \texttt{call}\ f\ v? :: \epsilon$ or $\mu = \texttt{call}\ f\ v?; \bot$, $T = \bot$ and $\text{type}(v) \neq \text{input\_type}(f)$

We are going to prove that any such trace extracted from a tree can be produced by the back-translated context, modulo the behaviors allowed at the target level but not at the source level.

Definition 51.

$$\mu^s = \begin{cases} \mu'\bot & \text{if } \mu = \mu'\texttt{call}\ f\ v? \text{ such that input\_type}(f) \neq \text{type}(v) \\ \mu'\bot & \text{if } \mu = \mu'\texttt{call}\ f\ v?\bot \text{ such that input\_type}(f) \neq \text{type}(v) \\ \mu & \text{otherwise} \end{cases}$$

Theorem 53 (Correction of the backtranslation)**.** Let T be a tree and μ a trace extracted from T. Then, $T\uparrow \rightsquigarrow \mu^s$.

Proof. We are going to prove by induction on the relation "μ is extracted from T" that there exists e such that $T\!\uparrow \xRightarrow{\mu^s} \mathsf{e}$.

1. $\mu = \epsilon$: OK.
2. $\mu = \Downarrow$ and $T = \Downarrow$: $T\!\uparrow = 0$. OK.
3. $\mu = \bot$ and $T = \bot$: $T\!\uparrow = \mathsf{fail}$. OK.
4. $\mu = \texttt{call}\ f\ v?; \epsilon$, $\text{type}(v) \neq \text{input\_type}(f)$ and $T = \bot$ We are in the first case for μ^s: OK.
5. $\mu = \texttt{call}\ f\ v?; \bot$, $\text{type}(v) \neq \text{input\_type}(f)$ and $T = \bot$ We are in the second case for μ^s: OK.
6. $\mu = \texttt{call}\ f\ v?; \epsilon$, $T = (\texttt{call}\ f\ v?, \dots)$ and $\text{type}(v) = \text{input\_type}(f)$: $T\!\uparrow = \mathsf{let\ x = call\ f\ v\ in\ \dots}$. OK. Idem with $\Uparrow$ instead of ϵ.
7. $t = \texttt{call}\ f\ v?; \texttt{ret}\ v'!; \mu'$, $T = (\texttt{call}\ f\ v?, (v_1, T_1), \dots, (v_i, T_i))$, and $\exists j$, such that $v_j = v'$ and μ' is extracted from T_j: Then:

   $$T\!\uparrow = \mathsf{let\ x = call\ f\ v\ in\ if\ \dots\ then\ if\ x = v_j\ then\ T_j\!\uparrow\ else\ \dots\ else\ \dots}.$$

   By application of the partial semantics:

   $$T\!\uparrow \xRightarrow{\texttt{call}\ f\ v?;\texttt{ret}\ v_j!}_{\text{ctx}} \mathsf{if\ x = v_j\ then\ T_j\!\uparrow\ else\ \dots [v_j/x]}$$

   and therefore by substituting and application of the partial semantics:

   $$T\!\uparrow \xRightarrow{\texttt{call}\ f\ v?;\texttt{ret}\ v_j!}_{\text{ctx}} T_j\!\uparrow.$$

   By induction hypothesis, we are done.
8. $\mu = \texttt{call}\ f\ v? :: \epsilon$ or $\mu = \texttt{call}\ f\ v?; \bot$, $T = \bot$ and $\text{type}(v) \neq \text{input\_type}(f)$. The result is immediate

□

Now, we can prove that any of the initial traces that are used to construct the tree can be found in this tree, and then the theorem applies to them.

Lemma 35. Let F be a set of traces and T such that $F \vDash T$. Then, any trace $\mu \in F$ is extracted from the tree T.

Proof. Let us prove by induction on T that if there exists F such that $T = T(F)$, then $\forall \mu \in F$, μ is extracted from T. Since the trace ϵ is always extracted from any tree, we ignore this case.

$T = \epsilon$: OK.

$T = \Downarrow$: Then $\mu = \Downarrow$. OK.

$T = \Uparrow$: Then $\mu = \Uparrow$. OK.

$T = (\texttt{call}\ f\ v?, (v_1, T_1), \dots, (v_i, T_i))$: By induction hypothesis.

□

8.5.6 Composition

The composition theorem states that if a context and a program can partially produce two related informative traces, then plugging the program into the context gives a whole program that can produce one of the traces. The relation between the two traces captures the fact that the way things fail in the source is not the same as in the target, as seen in the back-translation section. The theorem is stated as follows:

Theorem 54 (Composition)**.** Let $\mathsf{C_S}$ be a source context, $\mathsf{P_S}$ be a source program, $\mu_i \sim \mu_i^s$ two related traces. Then, if $\mathsf{C_S} \hookrightarrow_{\text{ctx}} \mu_i^s$ and $\mathsf{P_S} \hookrightarrow_{\text{prg}} \mu_i$, then $\mathsf{C_S}\left[\mathsf{P_S}\right] \hookrightarrow \mu_i^s$.

We state a preliminary lemma:

Lemma 36. If $\mathsf{P} \hookrightarrow_{\text{prg}} \mu_i$, then $\mathsf{P} \hookrightarrow_{\text{prg}} \mu_i^s$.

Proof. This is by definition of μ_i^s. □

Lemma 37. Let $\mathsf{C_S}$ be a source context, $\mathsf{P_S}$ be a source program, $\mu_i \sim \mu_i^s$ two related traces such that μ_i was produced by $[\![\mathsf{P_S}]\!]^{\mathsf{L}^\tau}_{\mathbf{L^u}}$ and some target context, and e an expression. Then, if $\mathsf{C_S} \xRightarrow{\mathsf{\mu_i^s}} \mathsf{e}$ and $\mathsf{P_s} \hookrightarrow_{\text{prg}} \mu_i^s$, then $\mathsf{P_S} \triangleright \mathsf{C_S} \xRightarrow{\mathsf{\mu_i^s}} \mathsf{e'}$ where $\mathsf{C_S} \xRightarrow{\mathsf{\mu_i^s}} \mathsf{e'}$.

Proof. We will prove by induction n that $\forall n, \forall \mu, |\mu| = n, \forall \mathsf{e}, \forall \mathsf{P_S}, \mathsf{e} \hookrightarrow_{\text{ctx}} \mu \wedge \mathsf{P_S} \hookrightarrow_{\text{prg}} \mu \implies \exists \mathsf{e'}, \mathsf{e} \xRightarrow{\mu} \mathsf{e'} \wedge \mathsf{P_S} \triangleright \mathsf{e} \xRightarrow{\mu} \mathsf{e'}$

Base case If $n = 0$, this is trivially true.

Inductive case Let $n \in \mathbb{N}$, μ of length n, e and $\mathsf{P_S}$ such that $\mathsf{e} \hookrightarrow_{\text{ctx}} \mu$ and $\mathsf{P_S} \hookrightarrow_{\text{prg}} \mu$.

We consider only one case, but the other cases are similar:

$$\mu = \mu_1\mu_2\mu_3$$

where $\mu_2 = \texttt{call}\ f\ v?\mu_2'\texttt{ret}\ v'!$ is defined as in the definition of $\hookrightarrow_{\text{prg}}$.

- First, $\mathsf{e} \hookrightarrow_{\text{ctx}} \mu_1$ and $\mathsf{e} \hookrightarrow_{\text{prg}} \mu_1$, by definition of these relations. Therefore, by induction hypothesis, $\exists \mathsf{e'}, \mathsf{e} \xRightarrow{\mu_1} \mathsf{e'}$ and $\mathsf{P_S} \triangleright \mathsf{e} \xRightarrow{\mu_1} \mathsf{e'}$. In particular, $\mathsf{e'}$ is of the form $\mathbb{E}\left[\mathsf{call\ f\ v}\right]$ by determinism of the execution of the context (since the read/writes are set by the trace), such that $\mathsf{e'} \hookrightarrow_{\text{ctx}} \mu_2\mu_3$.
- We have that $\mathsf{P_S} \triangleright \mathsf{e'} \xRightarrow{\mu_2} \mathbb{E}\left[\mathsf{v'}\right]$ by definition of the partial semnatics for programs, and the rules of evaluations inside contexts.
- We can again apply the induction hypothesis to μ_3.

Hence, we obtain the result: $\mathsf{P_S} \triangleright \mathsf{e} \xRightarrow{\mu_1\mu_2\mu_3} \mathsf{e''}$ where $\mathsf{e} \xRightarrow{\mu_1\mu_2\mu_3} \mathsf{e''}$.

□

By using these two lemmas, we can prove the composition theorem.

8.5.7 Back to Non-Informative Traces

The last step of the proof is to go back to the non-informative trace model. In particular, we must take into account that the trace μ_i^s that is generated by the whole program is not exactly equal to the original trace μ_i.

Theorem 55 (Back to non-informative traces)**.** Let $\mathsf{C_S}$ be a source context, $\mathsf{P_S}$ be a source program, m a non-informative trace and μ an informative trace such that $\mu \sqsupseteq m$.

Then, $\mathsf{C_S}\left[\mathsf{P_S}\right] \hookrightarrow \mu^s \implies \mathsf{C_S}\left[\mathsf{P_S}\right] \rightsquigarrow m$.

The proof is immediate by definition of μ^s.

8.5.8 Proving the Secure Compilation Criterion

Now that we have all the necessary theorems, we can finally prove that our compiler satisfy the criterion:

Theorem 48 (k-Relational Robust Safety Preservation)**.** Let $\mathsf{P}_1 \ldots \mathsf{P}_k$ be k programs that share the same interface $\bar{\mathsf{I}}$ and $m_1 \ldots m_k$ be k finite trace prefixes. Then, for all target contexts $\mathbf{C_T}$, the following holds:

$$\left(\forall i, \mathbf{C_T}\left[\llbracket \mathsf{P}_i \rrbracket_{\mathbf{T}}^{\mathsf{S}}\right] \rightsquigarrow m_i\right)$$
$$\implies (\exists \mathsf{C_S}, \forall i, \mathsf{C_S}[\mathsf{P}_i] \rightsquigarrow m_i)$$

The proof follows the scheme depicted by Figure 1.

Proof. Let $\mathsf{P}_1 \ldots \mathsf{P}_k$ be k programs and $m_1 \ldots m_k$ be k finite trace prefixes. Let $\mathbf{C_T}$ be a target context and suppose the following holds:

$$\forall i, \mathbf{C_T}\left[\llbracket \mathsf{P}_i \rrbracket_{\mathbf{T}}^{\mathsf{S}}\right] \rightsquigarrow m_i$$

We can pass to informative traces by applying Theorem 49 to each m_i

$$\forall i, \exists \mu_i \sqsupseteq m, \mathbf{C_T}\left[\llbracket \mathsf{P}_i \rrbracket_{\mathbf{T}}^{\mathsf{S}}\right] \hookrightarrow \mu_i.$$

From here, we can apply the decomposition theorem (Theorem 50) to each μ_i:

$$\forall i, \mathbf{C_T} \hookrightarrow_{\text{ctx}} \mu_i \wedge \llbracket \mathsf{P}_i \rrbracket_{\mathbf{T}}^{\mathsf{S}} \hookrightarrow_{\text{prg}} \mu_i.$$

By the backward compiler correctness theorem (Theorem 51) for programs applied to each program, we obtain that:

$$\forall i, \mathsf{P}_i \hookrightarrow_{\text{prg}} \mu_i.$$

Also, by applying the back-translation theorem, we can produce a source context:

$$\exists \mathsf{C_S}, \forall i, \mathsf{C_S} \hookrightarrow_{\text{ctx}} \mu_i^s.$$

Now, we are able to apply the composition theorem (Theorem 54) to each program:

$$\forall i, \mathsf{C_S}[\mathsf{P}_i] \hookrightarrow \mu_i^s$$

Finally, we can go back to the non-informative traces by the last theorem (Theorem 55):

$$\forall i, \mathsf{C_S}[\mathsf{P}_i] \rightsquigarrow m_i.$$

$\square$

Remarks on the proof technique This proof technique should be fairly generic and could be adapted to other languages. if needed, it is possible to change the top-level statement by introducing a more complex relation between source and target, that could for instance model the exchange between failure and calls that might happen in our instance, or to model non-determinism in a non-deterministic language. While decomposition and composition are natural properties that we expect to hold for most languages, and while backward correctness can reasonably be expected from a secure compiler, the back-translation seems to be the hardest part of the proof and the most subject to change between languages.