Exploring Robust Property Preservation for Secure Compilation (Online Appendix)

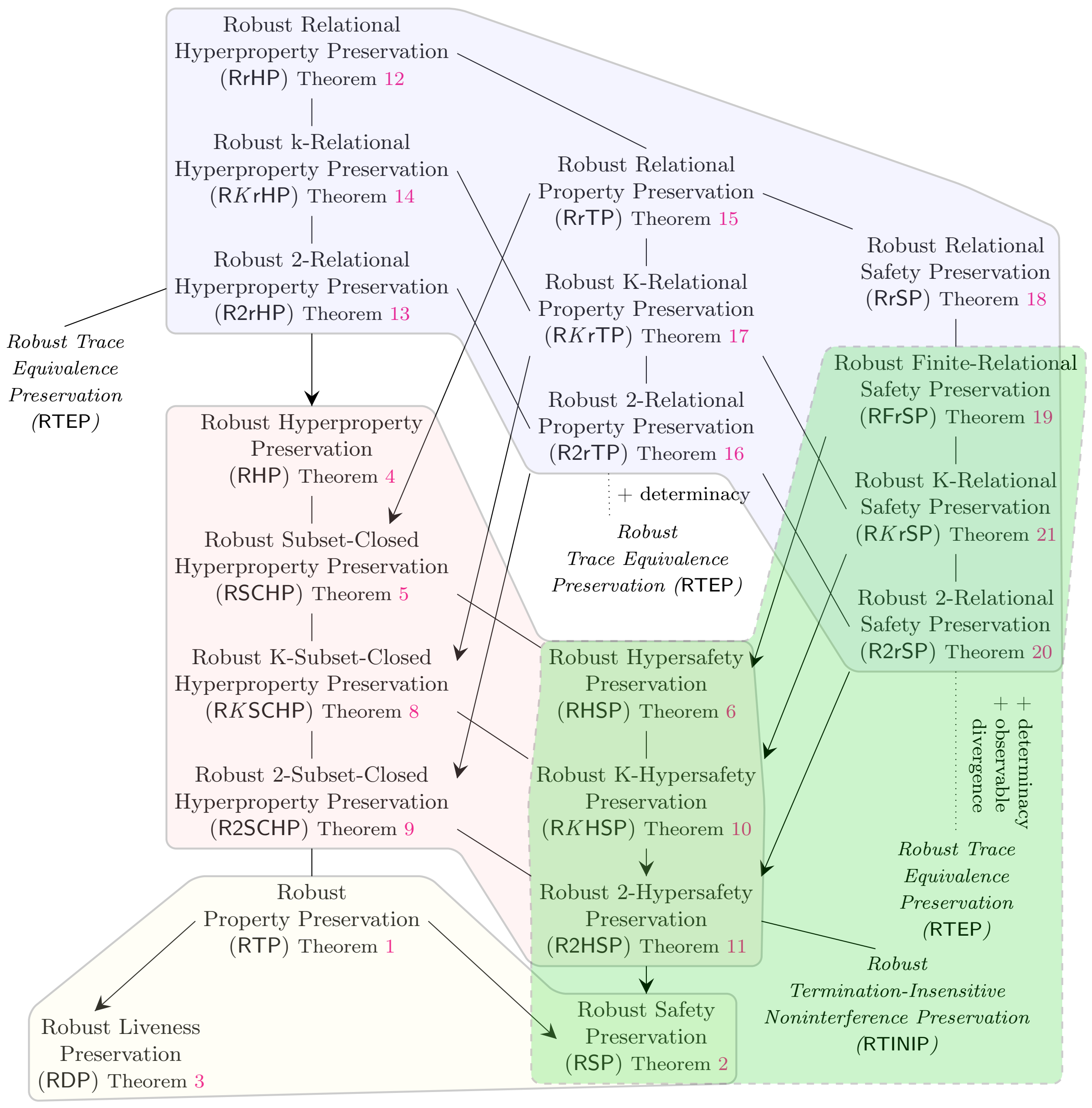
Anonymous Author(s)

July 12, 2018

Contents

Colour convention: We use $\textsf{blue, sans-serif}$ 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.

1 Secure Compilation Criteria


Robust Relational Hyperproperty Preservation (RrHP) Theorem 12
Robust k-Relational Hyperproperty Preservation (RKrHP) Theorem 14
Robust 2-Relational Hyperproperty Preservation (R2rHP) Theorem 13
Robust Trace Equivalence Preservation (RTEP)
Robust Relational Property Preservation (RrTP) Theorem 15
Robust K-Relational Property Preservation (RKrTP) Theorem 17
Robust 2-Relational Property Preservation (R2rTP) Theorem 16
+ determinacy
Robust Trace Equivalence Preservation (RTEP)
Robust Relational Safety Preservation (RrSP) Theorem 18
Robust Finite-Relational Safety Preservation (RFrSP) Theorem 19
Robust K-Relational Safety Preservation (RKrSP) Theorem 21
Robust 2-Relational Safety Preservation (R2rSP) Theorem 20
+ determinacy
+ observable divergence
Robust Trace Equivalence Preservation (RTEP)
Robust Hyperproperty Preservation (RHP) Theorem 4
Robust Subset-Closed Hyperproperty Preservation (RSCHP) Theorem 5
Robust K-Subset-Closed Hyperproperty Preservation (RKSCHP) Theorem 8
Robust 2-Subset-Closed Hyperproperty Preservation (R2SCHP) Theorem 9
Robust Hypersafety Preservation (RHSP) Theorem 6
Robust K-Hypersafety Preservation (RKHSP) Theorem 10
Robust 2-Hypersafety Preservation (R2HSP) Theorem 11
Robust Termination-Insensitive Noninterference Preservation (RTINIP)
Robust Property Preservation (RTP) Theorem 1
Robust Liveness Preservation (RDP) Theorem 3
Robust Safety Preservation (RSP) Theorem 2

1.1 Trace Property-based Criteria

1.1.1 Trace Property Preservation

Definition 1 (RTP)**.**

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

Definition 2 (RTP′)**.**

$$\mathsf{RTP}' : \forall \mathsf{P}.\ \forall \mathbf{C_T}.\ \forall t.\ \mathbf{C_T}\left[\mathsf{P}{\downarrow}\right] \rightsquigarrow t \Rightarrow \exists \mathsf{C_S}.\ \mathsf{C_S}\left[\mathsf{P}\right] \rightsquigarrow t$$

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

$$\forall \cdot{\downarrow}.\ \cdot{\downarrow} : \mathsf{RTP} \iff \cdot{\downarrow} : \mathsf{RTP}'$$

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

1.1.2 Safety

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

Definition 3 (RSP)**.**

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

Definition 4 (RSP′)**.**

$$\mathsf{RSP}' : \quad \forall \mathsf{P}.\ \forall \mathbf{C_T}.\ \forall m.\ \mathbf{C_T}\left[\mathsf{P}{\downarrow}\right] \rightsquigarrow m \Rightarrow \exists \mathsf{C_S}.\ \mathsf{C_S}\left[\mathsf{P}\right] \rightsquigarrow m$$

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

$$\forall \cdot{\downarrow}.\ \cdot{\downarrow} : \mathsf{RSP} \iff \cdot{\downarrow} : \mathsf{RSP}'$$

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

1.1.3 Dense Property Preservation

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

Definition 5 (RDP)**.**

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

Definition 6 (RDP′)**.**

$$\mathsf{RDP}' : \quad \forall \mathsf{P}.\ \forall \mathbf{C_T}.\ \forall t\ \mathit{infinite}.\ \mathbf{C_T}\left[\mathsf{P}{\downarrow}\right] \rightsquigarrow t \Rightarrow \exists \mathsf{C_S}.\ \mathsf{C_S}\left[\mathsf{P}\right] \rightsquigarrow t$$

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

$$\forall \cdot{\downarrow}.\ \cdot{\downarrow} : \mathsf{RDP} \iff \cdot{\downarrow} : \mathsf{RDP}'$$

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

1.2 Hyperproperty-based Criteria

1.2.1 Hyperproperties

Definition 7 (RHP)**.**

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

Definition 8 (RHP′)**.**

$$\begin{aligned}\mathsf{RHP}': \quad & \forall \mathsf{P}.\ \forall \mathbf{C_T}.\ \exists \mathsf{C_S}.\ \mathtt{Behav}\,(\mathbf{C_T}\,[\mathsf{P}{\downarrow}]) = \mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P}])\\ \iff & \forall \mathsf{P}.\ \forall \mathbf{C_T}.\ \exists \mathsf{C_S}.\ \forall t.\ \mathbf{C_T}\,[\mathsf{P}{\downarrow}] \rightsquigarrow t \iff \mathsf{C_S}\,[\mathsf{P}] \rightsquigarrow t\end{aligned}$$

Theorem 4 (RHP and RHP′ are equivalent)**.**

$$\forall \cdot{\downarrow}.\ \cdot{\downarrow} : \mathsf{RHP} \iff \cdot{\downarrow} : \mathsf{RHP}'$$

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

1.2.2 Subset-closed Hyperproperties

Definition 9 (RSCHP: Subset-closed Hyperproperty Robust Compilation)**.**

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

Definition 10 (RSCHP′)**.**

$$\mathsf{RSCHP}': \quad \forall \mathsf{P}.\ \forall \mathbf{C_T}.\ \exists \mathsf{C_S}.\ \forall t.\ \mathbf{C_T}\,[\mathsf{P}{\downarrow}] \rightsquigarrow t \Rightarrow \mathsf{C_S}\,[\mathsf{P}] \rightsquigarrow t$$

Theorem 5 (RSCHP and RSCHP′ are equivalent)**.**

$$\forall \cdot{\downarrow}.\ \cdot{\downarrow} : \mathsf{RSCHP} \iff \cdot{\downarrow} : \mathsf{RSCHP}'$$

Proof. See file Criteria.v,theorem SSC_criterium □

1.2.3 Hypersafety

$Obs = 2^{2^{FinPref}_{Fin}}$

$$Hypersafety \triangleq \{H \mid \forall b \notin H.\ (\exists o \in Obs.\ o \leq b \wedge (\forall b' \geq o.\ b' \notin H))\}$$

Definition 11 (RHSP)**.**

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

Definition 12 (RHSP′)**.**

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

Theorem 6 (RHSP and RHSP′ are equivalent)**.**

$$\forall \cdot{\downarrow}.\ \cdot{\downarrow} : \mathsf{RHSP} \iff \cdot{\downarrow} : \mathsf{RHSP}'$$

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

1.2.4 Hyperliveness Preservation

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

Definition 13 (RLHP: Liveness Hyperproperty Robust Preservation)**.**

$$\cdot\!\downarrow : \mathsf{RLHP} \overset{\mathsf{def}}{=} \forall H \in \mathit{Hyperliveness}.\ \forall \mathsf{P}.\ (\forall \mathsf{C_S}.\, \mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P}]) \in H) \Rightarrow (\forall \mathbf{C_T}.\, \mathtt{Behav}\,(\mathbf{C_T}\,[\mathsf{P}\!\downarrow]) \in H)$$

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

$$\forall \cdot\!\downarrow.\ \cdot\!\downarrow : \mathsf{RHP} \iff \cdot\!\downarrow : \mathsf{RLHP}$$

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

1.2.5 K- and 2- Subset-closed Hyperproperties

Notation: $\widehat{t}$ indicates a set of traces t.

Definition 14 (KSC)**.**

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

Definition 15 (RKSCHP)**.**

$$\cdot\!\downarrow : \mathsf{R}K\mathsf{SCHP} \overset{\mathsf{def}}{=} \forall H \in KSC.\ \forall \mathsf{P}.\ (\forall \mathsf{C_S}.\, \mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P}]) \in H) \Rightarrow (\forall \mathbf{C_T}.\, \mathtt{Behav}\,(\mathbf{C_T}\,[\mathsf{P}\!\downarrow]) \in H)$$

Definition 16 (RKSCHP′)**.**

$$\cdot\!\downarrow : \mathsf{R}K\mathsf{SCHP}' \overset{\mathsf{def}}{=} \forall \mathsf{P}, \mathbf{C_T}.\forall \widehat{t}.\|\widehat{t}\| = k.$$
$$(\widehat{t} \subseteq \mathtt{Behav}\,(\mathbf{C_T}\,[\mathsf{P}\!\downarrow])) \Rightarrow \exists \mathsf{C_S}.(\widehat{t} \subseteq \mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P}]))$$

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

$$\forall \cdot\!\downarrow.\ \cdot\!\downarrow : \mathsf{R}K\mathsf{SCHP} \iff \cdot\!\downarrow : \mathsf{R}K\mathsf{SCHP}'$$

Proof. Analogous to that of Theorem 5. □

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

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

$$\forall \cdot\!\downarrow.\ \cdot\!\downarrow : \mathsf{R2SCHP} \iff \cdot\!\downarrow : \mathsf{R2SCHP}'$$

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

1.2.6 K- and 2-Hypersafety

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

Definition 17 (RKHSP)**.**

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

Definition 18 (RKHSP′: K-Safety Hyperproperty Robust Preservation)**.**

$$\begin{aligned} \cdot{\downarrow} : \mathsf{R}K\mathsf{HSP}' \overset{\mathsf{def}}{=} \forall \mathsf{P}, \mathbf{C_T}. \forall \widehat{m}. \|\widehat{m}\| = k.& \\ (\widehat{m} \leq \mathtt{Behav}\,(\mathbf{C_T}\,[\mathsf{P}{\downarrow}])) \Rightarrow (\exists \mathsf{C_S}. \widehat{m} &\leq \mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P}])) \end{aligned}$$

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

$$\forall \cdot{\downarrow}.\ \cdot{\downarrow} : \mathsf{R}K\mathsf{HSP} \iff \cdot{\downarrow} : \mathsf{R}K\mathsf{HSP}'$$

Proof. See file Criteria.v, theorem R2HSP_H2SRC. □

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

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

$$\forall \cdot{\downarrow}.\ \cdot{\downarrow} : \mathsf{R2HSP} \iff \cdot{\downarrow} : \mathsf{R2HSP}'$$

Proof. Analogous to Theorem 6. □

1.3 Relational Criteria

1.3.1 Relational Hyperproperties Preservation

Definition 19 (RrHP)**.**

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

Definition 20 (RrHP′)**.**

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

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

$$\forall \cdot{\downarrow}.\ \cdot{\downarrow} : \mathsf{RrHP} \iff \cdot{\downarrow} : \mathsf{RrHP}'$$

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

1.3.2 2-Relational Hyperproperties Preservation

Definition 21 (R2rHP)**.**

$$\mathsf{R2rHP}: \quad \forall R \in 2^{(\mathtt{Behavs}^2)}.\ \forall \mathsf{P_1}\ \mathsf{P_2}.\ (\forall \mathsf{C_S}.\ (\mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P_1}]), \mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P_2}])) \in R) \\ \Rightarrow (\forall \mathbf{C_T}.\ (\mathtt{Behav}\,(\mathbf{C_T}\,[\mathsf{P_1}\!\downarrow]), \mathtt{Behav}\,(\mathbf{C_S}\,[\mathsf{P_2}\!\downarrow])) \in R)$$

Definition 22 (R2rHP′)**.**

$$\mathsf{R2rHP}': \quad \forall \mathsf{P_1}\ \mathsf{P_2}.\ \forall \mathbf{C_T}.\ \exists \mathsf{C_S}.\ \mathtt{Behav}\,(\mathbf{C_T}\,[\mathsf{P_1}\!\downarrow]) = \mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P_1}]) \wedge \\ \mathtt{Behav}\,(\mathbf{C_T}\,[\mathsf{P_2}\!\downarrow]) = \mathtt{Behav}\,(\mathsf{C_S}\,[\mathsf{P_2}])$$

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

$$\forall \cdot\!\downarrow.\ \cdot\!\downarrow : \mathsf{R2rHP} \iff \cdot\!\downarrow : \mathsf{R2rHP}'$$

Proof. See file Criteria.v, theorem r2HRP_r2HRC. □

K-Relational Hyperproperties Preservation For these defs, take the defs above and replace $\forall \mathsf{C_1}, \mathsf{C_2}$ with $\forall \mathsf{C_1}, \cdots, \mathsf{C_k}$.

Theorem 14 (R2rHP and R2rHP′ are equivalent)**.**

$$\forall \cdot\!\downarrow.\ \cdot\!\downarrow : \mathsf{R2rHP} \iff \cdot\!\downarrow : \mathsf{R2rHP}'$$

Proof. Analogous to Theorem 12. □

1.3.3 Relational Property Preservation

Definition 23 (RrTP)**.**

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

Definition 24 (RrTP′)**.**

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

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

$$\forall \cdot\!\downarrow.\ \cdot\!\downarrow : \mathsf{RrTP} \iff \cdot\!\downarrow : \mathsf{RrTP}'$$

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

1.3.4 2-Relational Property Preservation

Definition 25 (R2rTP)**.**

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

Definition 26 (R2rTP′)**.**

$$\mathsf{R2rTP}': \ \forall \mathsf{P}_1\ \mathsf{P}_2.\ \forall \mathbf{C_T}.\ \forall t_1\ t_2.\ (\mathbf{C_T}\,[\mathsf{P}_1{\downarrow}] \rightsquigarrow t_1 \wedge \mathbf{C_T}\,[\mathsf{P}_2{\downarrow}] \rightsquigarrow t_2)$$
$$\Rightarrow \exists \mathsf{C_S}.\ (\mathsf{C_S}\,[\mathsf{P}_1] \rightsquigarrow t_1 \wedge \mathsf{C_S}\,[\mathsf{P}_2] \rightsquigarrow t_2)$$

Theorem 16 (R2rTP and R2rTP′ are equivalent)**.**

$$\forall \cdot{\downarrow}.\ \cdot{\downarrow} : \mathsf{R2rTP} \iff \cdot{\downarrow} : \mathsf{R2rTP}'$$

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

1.3.5 K-Relational Property Preservation

For these defs, take the defs above and replace $\forall \mathsf{C}_1, \mathsf{C}_2$ with $\forall \mathsf{C}_1, \cdots, \mathsf{C}_\mathsf{k}$. and the two implications/conjunctions with k of them.

Theorem 17 (RKrTP′and RKrTP are equivalent)**.**

$$\forall \cdot{\downarrow}.\ \cdot{\downarrow} : \mathsf{R}K\mathsf{rTP}' \iff \cdot{\downarrow} : \mathsf{R}K\mathsf{rTP}$$

Proof. Analogous to Theorem 15. □

1.3.6 Relational Safety Preservation

Definition 27 (RrSP)**.**

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

Definition 28 (RrSP′)**.**

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

Theorem 18 (RrSP′and RrSP are equivalent)**.**

$$\forall \cdot{\downarrow}.\ \cdot{\downarrow} : \mathsf{RrSP}' \iff \cdot{\downarrow} : \mathsf{RrSP}$$

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

1.3.7 Finite-Relational Safety Preservation

$m \prec S$ indicates that prefix m is extended by at least one trace in the set S.

Definition 29 (Robust Finite-Relational Safety Compilation)**.**

$$\begin{aligned}\mathsf{RFrSP}' :\ \forall K.\ \forall \mathsf{P_1} \ldots \mathsf{P_K}.\ \forall \mathbf{C_T}.\ \forall m_1 \ldots m_K.\ (\mathbf{C_T}\left[\mathsf{P_1}{\downarrow}\right] \rightsquigarrow m_1 \wedge \ldots \wedge \mathbf{C_T}\left[\mathsf{P_K}{\downarrow}\right] \rightsquigarrow m_K) \\ \Rightarrow \exists \mathsf{C_S}.\ (\mathsf{C_S}\left[\mathsf{P_1}\right] \rightsquigarrow m_1 \wedge \ldots \wedge \mathsf{C_S}\left[\mathsf{P_K}\right] \rightsquigarrow m_K)\end{aligned}$$

Definition 30 (Finite-Relational Safety Robust Preservation)**.**

$$\begin{aligned}\cdot{\downarrow} : \mathsf{RFrSP} \overset{\mathsf{def}}{=} \forall k, \mathsf{P_1}, \cdots, \mathsf{P_k}, R \in 2^{(FinPref^k)}. \\ (\forall \mathsf{C_S}, m_1, \cdots, m_k, (\mathsf{C_S}\left[\mathsf{P_1}\right] \rightsquigarrow m_1 \wedge \cdots \wedge \mathsf{C_S}\left[\mathsf{P_k}\right] \rightsquigarrow m_k) \\ \Rightarrow (m_1, \cdots, m_k) \in R) \\ \Rightarrow (\forall \mathbf{C_T}.(\mathbf{C_T}\left[\mathsf{P_1}{\downarrow}\right] \rightsquigarrow m_1 \wedge \cdots \wedge \mathbf{C_T}\left[\mathsf{P_k}{\downarrow}\right] \rightsquigarrow m_k) \\ \Rightarrow (m_1, \cdots, m_k) \in R)\end{aligned}$$

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

$$\forall \cdot{\downarrow}.\ \cdot{\downarrow} : \mathsf{RFrSP} \iff \cdot{\downarrow} : \mathsf{RFrSP}'$$

Proof. Analogous to Theorem 20. □

1.3.8 2-Relational Safety Preservation

Definition 31 ($\mathsf{R2rSP}$)**.**

$$\begin{aligned}\mathsf{R2rSP} : \forall R \in 2^{(FinPref^2)}.\ \forall \mathsf{P_1}\ \mathsf{P_2}.\ (\forall \mathsf{C_S}\ m_1\ m_2.\ (\mathsf{C_S}\left[\mathsf{P_1}\right] \rightsquigarrow m_1 \wedge \mathsf{C_S}\left[\mathsf{P_2}\right] \rightsquigarrow m_2) \Rightarrow (m_1, m_2) \in R) \\ \Rightarrow (\forall \mathbf{C_T}\ m_1\ m_2.\ (\mathbf{C_T}\left[\mathsf{P_1}{\downarrow}\right] \rightsquigarrow m_1 \wedge \mathbf{C_T}\left[\mathsf{P_2}{\downarrow}\right] \rightsquigarrow m_2) \Rightarrow (m_1, m_2) \in R)\end{aligned}$$

Definition 32 ($\mathsf{R2rSP}'$)**.**

$$\begin{aligned}\mathsf{R2rSP}' :\ \forall \mathsf{P_1}\ \mathsf{P_2}.\ \forall \mathbf{C_T}.\ \forall m_1\ m_2.\ (\mathbf{C_T}\left[\mathsf{P_1}{\downarrow}\right] \rightsquigarrow m_1 \wedge \mathbf{C_T}\left[\mathsf{P_2}{\downarrow}\right] \rightsquigarrow m_2) \\ \Rightarrow \exists \mathsf{C_S}.\ (\mathsf{C_S}\left[\mathsf{P_1}\right] \rightsquigarrow m_1 \wedge \mathsf{C_S}\left[\mathsf{P_2}\right] \rightsquigarrow m_2)\end{aligned}$$

Theorem 20 ($\mathsf{R2rSP}$ and $\mathsf{R2rSP}'$ are equivalent)**.**

$$\forall \cdot{\downarrow}.\ \cdot{\downarrow} : \mathsf{R2rSP} \iff \cdot{\downarrow} : \mathsf{R2rSP}'$$

Proof. See file Criteria.v, r2RSP_r2RSC. □

1.3.9 K-Relational Safety Preservation

For these defs, take the defs above and replace $\forall \mathsf{C_1}, \mathsf{C_2}$ with $\forall \mathsf{C_1}, \cdots, \mathsf{C_k}$. and the two implications/conjunctions with k of them.

Theorem 21 ($\mathsf{R}K\mathsf{rSP}'$and $\mathsf{R}K\mathsf{rSP}$ are equivalent)**.**

$$\forall \cdot{\downarrow}.\ \cdot{\downarrow} : \mathsf{R}K\mathsf{rSP}' \iff \cdot{\downarrow} : \mathsf{R}K\mathsf{rSP}$$

Proof. Analogous to Theorem 20. □

2 Separation Results

2.1 RSP and RDP do not imply RTP

In this section we show that the robust preservation of all safety (dense respectively) properties is not enough to guarantee the robust preservation of all properties. In particular 22 shows that the robust preservation of all safety properties does not imply the robust preservation of all dense property, while **??** shows the viceversa.

Consider an arbitrary language $\mathcal{L}$ described by a small-step semantics, assume it is possible to write a non terminating program in $\mathcal{L}$, i.e. a program that produces some infinite trace. Assume moreover that such a program is indipendent from the context with which it is linked. For a more concrete example consider a while language and the program P_Ω

```
WHILE true

  output(n);

END
```

For some $n \in \mathbb{N}$.

We define $\phi(\mathcal{L})$ to be identical to $\mathcal{L}$, except that it uses *"fuel"* to bound its executions. 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 $(n, C)[P]_{\phi(\mathcal{L})} \equiv (n, C[P]_{\mathcal{L}})$, we will omit subscripts from now on.
- the semantics of $\phi(\mathcal{L})$ extends the semantics of $\mathcal{L}$ as following. Every time a step is taken the fuel is decremented of one unit, if the fuel is 0 then no step is 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 the dense ones are.

Let $S \in \mathit{Safety}$, assume the premises for robust preservation of all safety properties, i.e. that for every program P, for every source context (n, C) and every trace t,

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

and assume by contradiction that there exists some target context C' and a trace t' such that

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

with $P\downarrow = P$.
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 with the same events of m followed by stopping. 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.

Now we are going to 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 obliged to stop after a finite number of steps and produces a trace that is infinite and different from $\textit{output}(42)^{\omega}$. □

Theorem 23. RDP $\not\Rightarrow$ 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 ones.
Let L be a dense property. Every trace t produced by a program in the target is finite so that we might think at it as a finite prefix with a $\bullet$ and denote it by m_t. By definition of *Diveness*, 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 the following property

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

S is a safety property because for every trace $t \notin S$, t starts with some successive occurrences of $\textit{output}(42)$ but then it contains some other event $e \neq \textit{output}(42)$ or stops, i.e.

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

and every continuation of $output(42)^n; e$ is different from $output(42)^\omega$, as well as every finite trace.

Finally consider the program $P = P\downarrow$

```
WHILE true

  output(42);

END
```

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

Theorem 24. RSP nor RDP implies RTP.

Proof. Consequence of 22 and 23 □

2.2 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`. □

2.3 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`. □

2.4 Robust Non-Relational Preservation Does Not Imply Robust Relational 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`. □

2.5 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`. □

3 Relational Criteria and Robust Trace Equivalence Preservation

Theorem 29. R2rHP ⇒ RTEP.

Proof. See file Criteria.v, Theorem r2HRP_teq. □

Theorem 30. For deterministic source languages R2rTP ⇒ RTEP.

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

Theorem 31. Assuming :

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

then R2rTP ⇒ RTEP.

Proof. See file TEP.txt. □

Theorem 32. Assuming :

1. the source language is determined
2. the target language respects 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 R2rSP ⇒ RTEP.

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

4 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. Decomposition theorem holds, i.e. that ever property can be written as intersection of a safety property and a dense one.

2. $\mathit{Safety} \cap \mathit{Dense} = \{\mathit{True}\}$

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

Proof. We prove mutual inclusion of the class *Dense* into the class of the dense sets in the topology.
Let $D \in \mathit{Dense}$. The smallest safety property that includes D is *True*, or in topological terms the closure of D is *True*, that is a sufficient condition for D to be a dense set in the topology.
Vice versa assume by contradiction there is a dense set Δ in the topology that is not a dense property. Then apply the decomposition theorem and get

$$\Delta = S \cap D \tag{1}$$

for some S safety property and D dense property. If $S = \mathit{True}$ we immediately have a contradiction. If there exists $t \notin S$ then the complement of S, $\sim S$ is a non empty open set in the topology, so that it has non empty intersection with Δ,

$$\Delta \cap \sim S \neq \varnothing \tag{2}$$

on the other hand by equation 1 we get

$$\Delta \cap \sim S = S \cap D \cap \sim S = \varnothing \tag{3}$$

a contradiction.

□

Finally recall that if a set is dense then every set containing 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 fact 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 roles 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.

5 Instances

This section presents a compiler and two proof techniques.

5.1 The Source Language L^τ

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

5.1.1 Syntax

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

5.1.2 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.

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

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

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

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

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

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

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

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

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

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

5.1.3 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 behaviour β.

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

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

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

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

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

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

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

$$\text{(EL}^{\tau}\text{-ret-internal)}\quad \frac{}{P \triangleright_{\overline{f},f,f'} \text{return } v \xrightarrow{\epsilon} P \triangleright_{\overline{f},f} v}$$

$$\text{(EL}^{\tau}\text{-ret-out)}\quad \frac{}{P \triangleright_{f} \text{return } v \xrightarrow{\texttt{ret v!}} P \triangleright v}$$

$$\text{(EL}^{\tau}\text{-read)}\quad \frac{}{P \triangleright \text{read} \xrightarrow{\texttt{read n}} P \triangleright n}$$

$$\text{(EL}^{\tau}\text{-write)}\quad \frac{}{P \triangleright \text{write } n \xrightarrow{\texttt{write n}} P \triangleright n}$$

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

$$\text{(EL}^{\tau}\text{-fail)}\quad \frac{}{P \triangleright \text{fail} \xrightarrow{\perp} \text{fail}}$$

$$\boxed{P \triangleright e \xRightarrow{\beta} P \triangleright e'}$$

$$\text{(EL}^{\tau}\text{-refl)}\quad \frac{}{\Omega \Rightarrow \Omega}$$

$$\text{(EL}^{\tau}\text{-terminate)}\quad \frac{\Omega \not\Rightarrow \_}{\Omega \xRightarrow{\Downarrow} \Omega}$$

$$\text{(EL}^{\tau}\text{-diverge)}\quad \frac{\forall n.\ \Omega \xrightarrow{\epsilon}{}^{n}\ \Omega'_n}{\Omega \xRightarrow{\Uparrow} \Omega}$$

$$\text{(EL}^{\tau}\text{-silent)}\quad \frac{\Omega \xrightarrow{\epsilon} \Omega'}{\Omega \Rightarrow \Omega'}$$

$$\text{(EL}^{\tau}\text{-single)}\quad \frac{\Omega \xrightarrow{\alpha} \Omega'}{\Omega \xRightarrow{\alpha} \Omega'}$$

$$\text{(EL}^{\tau}\text{-silent-act)}\quad \frac{\Omega \xrightarrow{\gamma} \Omega'}{\Omega \Rightarrow \Omega'}$$

$$\text{(EL}^{\tau}\text{-cons)}\quad \frac{\Omega \xRightarrow{\beta} \Omega'' \qquad \Omega'' \xRightarrow{\beta'} \Omega'}{\Omega \xRightarrow{\beta\beta'} \Omega'}$$

$$\boxed{P \triangleright e \overset{t}{\Rrightarrow} P \triangleright e'}$$

$$\text{(EL}^{\tau}\text{-silent)}\quad \frac{\Omega \xrightarrow{\epsilon} \Omega'}{\Omega \Rrightarrow \Omega'}$$

$$\text{(EL}^{\tau}\text{-action)}\quad \frac{\Omega \xRightarrow{\alpha} \Omega'}{\Omega \overset{\alpha}{\Rrightarrow} \Omega'}$$

$$\text{(EL}^{\tau}\text{-single)}\quad \frac{\Omega \xrightarrow{\gamma} \Omega'}{\Omega \overset{\gamma}{\Rrightarrow} \Omega'}$$

$$\text{(EL}^{\tau}\text{-cons)}\quad \frac{\Omega \overset{\sigma}{\Rrightarrow} \Omega'' \qquad \Omega'' \overset{\sigma'}{\Rrightarrow} \Omega'}{\Omega \overset{\sigma\sigma'}{\Rrightarrow} \Omega'}$$

5.1.4 Auxiliaries and Definitions

$$\boxed{\text{Helpers}}$$

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

Definition 33 (Program Behaviours)**.**

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

Theorem 34 (Progress)**.**

Theorem 35 (Preservation)**.**

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

5.2.1 Syntax

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

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.

5.2.2 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 behaviour β.
$\Omega \overset{\sigma}{\Longrightarrow\!\!\!\!\Rightarrow} \Omega'$	Program state Ω steps to Ω' with trace σ.

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

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

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

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

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

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

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

$$\frac{}{\mathbf{P} \triangleright \mathbf{read} \xrightarrow{\mathtt{read}\ \mathbf{n}} \mathbf{P} \triangleright \mathbf{n}}\ (\mathbf{EL^u}\text{-read})$$

$$\frac{}{\mathbf{P} \triangleright \mathbf{write\ n} \xrightarrow{\mathtt{write}\ \mathbf{n}} \mathbf{P} \triangleright \mathbf{n}}\ (\mathbf{EL^u}\text{-write})$$

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

$$\frac{}{\mathbf{P} \triangleright \mathbf{fail} \xrightarrow{\epsilon} \mathbf{fail}}\ (\mathbf{EL^u}\text{-fail})$$

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

$$\frac{}{\mathbf{P} \triangleright \mathbf{n\ has}\ \mathtt{Bool} \xrightarrow{\epsilon} \mathbf{P} \triangleright \mathbf{false}}\ (\mathbf{EL^u}\text{-check-bool-false})$$

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

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

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

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

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

$$\frac{}{\mathbf{P} \triangleright_{\mathbf{f}} \mathbf{return\ v} \xrightarrow{\mathtt{ret}\ \mathbf{v}!} \mathbf{P} \triangleright \mathbf{v}}\ (\mathbf{EL^u}\text{-ret-out})$$

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

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

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

$$\boxed{\mathbf{P} \triangleright \mathbf{e} \xRightarrow{\beta} \mathbf{P} \triangleright \mathbf{e'}}$$

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

$$\frac{\mathbf{\Omega} \xrightarrow{\gamma} \mathbf{\Omega'}}{\mathbf{\Omega} \Rightarrow \mathbf{\Omega'}}\;(\mathsf{E}\mathbf{L}^{\mathbf{u}}\text{-silent-act}) \qquad \frac{\mathbf{\Omega} \xrightarrow{\alpha} \mathbf{\Omega'}}{\mathbf{\Omega} \xRightarrow{\alpha} \mathbf{\Omega'}}\;(\mathsf{E}\mathbf{L}^{\mathbf{u}}\text{-single}) \qquad \frac{\mathbf{\Omega} \xRightarrow{\beta} \mathbf{\Omega''} \qquad \mathbf{\Omega''} \xRightarrow{\beta'} \mathbf{\Omega'}}{\mathbf{\Omega} \xRightarrow{\beta\beta'} \mathbf{\Omega'}}\;(\mathsf{E}\mathbf{L}^{\mathbf{u}}\text{-cons})$$

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

$$\frac{\mathbf{\Omega} \xrightarrow{\epsilon} \mathbf{\Omega'}}{\mathbf{\Omega} \Longrightarrow\!\!\!\!\!\Rightarrow \mathbf{\Omega'}}\;(\mathsf{E}\mathbf{L}^{\mathbf{u}}\text{-silent}) \qquad \frac{\mathbf{\Omega} \xRightarrow{\alpha} \mathbf{\Omega'}}{\mathbf{\Omega} \overset{\alpha}{\Longrightarrow\!\!\!\!\!\Rightarrow} \mathbf{\Omega'}}\;(\mathsf{E}\mathbf{L}^{\mathbf{u}}\text{-action}) \qquad \frac{\mathbf{\Omega} \xrightarrow{\gamma} \mathbf{\Omega'}}{\mathbf{\Omega} \overset{\gamma}{\Longrightarrow\!\!\!\!\!\Rightarrow} \mathbf{\Omega'}}\;(\mathsf{E}\mathbf{L}^{\mathbf{u}}\text{-single}) \qquad \frac{\begin{array}{c}\mathbf{\Omega} \overset{\sigma}{\Longrightarrow\!\!\!\!\!\Rightarrow} \mathbf{\Omega''} \\ \mathbf{\Omega''} \overset{\sigma'}{\Longrightarrow\!\!\!\!\!\Rightarrow} \mathbf{\Omega'}\end{array}}{\mathbf{\Omega} \overset{\sigma\sigma'}{\Longrightarrow\!\!\!\!\!\Rightarrow} \mathbf{\Omega'}}\;(\mathsf{E}\mathbf{L}^{\mathbf{u}}\text{-cons})$$

5.2.3 Auxiliaries and Definitions

$$\boxed{\text{Helpers}}$$

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

Definition 34 (Program Behaviours)**.**

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

Definition 35 (Program Traces)**.**

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

5.3 $\cdot\downarrow^{\mathsf{L}^\tau}_{\mathbf{L^u}}$: a Compiler from L^τ to $\mathbf{L^u}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5.4 Proof that $\cdot\downarrow^{\mathsf{L}^\tau}_{\mathbf{L^u}}$ is RrSP′

5.4.1 $\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}$: 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.

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

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

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

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

$$\begin{aligned}\langle\!\langle\mathbf{e}\oplus\mathbf{e}'\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} &= \mathsf{let}\ \mathsf{x1}:\mathsf{Nat}{=}\mathsf{extract}_{\mathsf{Nat}}(\langle\!\langle\mathbf{e}\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}) \quad (\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\text{-Op})\\ &\quad \mathsf{in\ let}\ \mathsf{x2}:\mathsf{Nat}{=}\mathsf{extract}_{\mathsf{Nat}}(\langle\!\langle\mathbf{e}'\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau})\\ &\quad \mathsf{in}\ \mathsf{inject}_{\mathsf{Nat}}(\mathsf{x1}\oplus\mathsf{x2})\end{aligned}$$

$$\langle\!\langle \mathbf{e} \geq \mathbf{e}' \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} = \mathsf{let\ x1 : Nat{=}extract_{Nat}}(\langle\!\langle \mathbf{e} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}) \quad (\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\text{-Geq})$$
$$\mathsf{in\ let\ x2 : Nat{=}extract_{Nat}}(\langle\!\langle \mathbf{e}' \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau})$$
$$\mathsf{in\ inject_{Bool}(x1 \geq x2)}$$
$$\langle\!\langle \mathbf{let\ x = e\ in\ e'} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} = \mathsf{let\ x : Nat} = \langle\!\langle \mathbf{e} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\ \mathsf{in}\ \langle\!\langle \mathbf{e}' \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} \quad (\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\text{-Let})$$
$$\langle\!\langle \mathbf{if\ e\ then\ e'\ else\ e''} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} = \mathsf{if\ extract_{Bool}}(\langle\!\langle \mathbf{e} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau})\ \mathsf{then}\ \langle\!\langle \mathbf{e}' \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\ \mathsf{else}\ \langle\!\langle \mathbf{e}'' \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} \quad (\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\text{-If})$$
$$\langle\!\langle \mathbf{call\ f\ e} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} = \mathsf{inject}_{\tau'}(\mathsf{call\ f\ extract}_{\tau}(\langle\!\langle \mathbf{e} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau})) \quad (\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\text{-Call})$$
$$\text{if } \mathsf{f} : \tau \to \tau' \in \bar{\mathsf{I}}$$
$$\langle\!\langle \mathbf{e\ has}\ \tau \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} = \begin{cases} \mathsf{let\ x : Nat} = \langle\!\langle \mathbf{e} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\ \mathsf{in\ if\ x \geq 2\ then\ 0\ else\ 1} & \text{if } \tau \equiv \mathtt{Bool} \\ \mathsf{let\ x : Nat} = \langle\!\langle \mathbf{e} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\ \mathsf{in\ if\ x \geq 2\ then\ 1\ else\ 0} & \text{if } \tau \equiv \mathbb{N} \end{cases} \quad (\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}\text{-Check})$$

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.

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

5.4.2 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 TL$^\tau$-function-call with these below.

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

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

Context well-formedness ensures that expressions are never turned into **call f** values.

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

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

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

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

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

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

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

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

Worlds

$$\begin{aligned}
World\ W &::= (n, (\mathsf{P}, \mathbf{P}))\\
lev((n, \_)) &= n\\
progs((\_, (\mathsf{P}, \mathbf{P}))) &= (\mathsf{P}, \mathbf{P})\\
srcprog((\_, (\mathsf{P}, \mathbf{P}))) &= \mathsf{P}\\
trgprog((\_, (\mathsf{P}, \mathbf{P}))) &= \mathbf{P}\\
\triangleright((0, \_)) &= (0, \_)\\
\triangleright((n+1, \_)) &= (n, \_)\\
W \sqsupseteq W' &= lev(W') \leq lev(W)\\
W \sqsupset_\triangleright W' &= lev(W') < lev(W)
\end{aligned}$$

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

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

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

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

$$\nearrow(R) \stackrel{\mathsf{def}}{=} \{(W, \mathsf{v}_1, \mathbf{v_2}) \mid \forall W' \sqsupseteq W.(W', \mathsf{v}_1, \mathbf{v_2}) \in R\}$$

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 Γ that has the same domain but where variables all have type Nat.

Value, Context, Expression and Environment relation

$$\mathcal{V}[\![\mathsf{Bool}]\!]_{\triangledown} \stackrel{\mathsf{def}}{=} \{(W, \mathsf{true}, \mathbf{true}), (W, \mathsf{false}, \mathbf{false})\}$$

$$\mathcal{V}[\![\mathsf{Nat}]\!]_{\triangledown} \stackrel{\mathsf{def}}{=} \{(W, \mathsf{n}, \mathbf{n})\}$$

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

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

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

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

$$\mathcal{G}[\![\varnothing]\!]_{\triangledown} \stackrel{\mathsf{def}}{=} \{(W, \varnothing, \varnothing)\}$$

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

Relation for Open and Closed Terms and Programs

Definition 36 (Logical relation up to n steps)**.**

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

Definition 37 (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 38 (Logical relation for programs)**.**

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

Auxiliary Lemmas from Existing Work

Lemma 1 (No observation with 0 steps)**.**

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

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

Lemma 2 (No steps means relation)**.**

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

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

Lemma 3 (Later preserves monotonicity)**.**

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

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

Lemma 4 (Monotonicity for environment relation)**.**

$$\begin{aligned}\text{if} \quad & W' \sqsupseteq W \\ & (W, \gamma, \gamma) \in \mathcal{G}\llbracket \Gamma \rrbracket_{\triangledown} \\ \text{then} \quad & (W', \gamma, \gamma) \in \mathcal{G}\llbracket \Gamma \rrbracket_{\triangledown}\end{aligned}$$

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

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

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

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

Lemma 6 (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 7 (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 8 (Adequacy for $\lesssim$)**.**

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

Proof. By Definition 37 (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 9 (Adequacy for $\gtrsim$)**.**

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

Proof. By Definition 37 (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 10 (Observation relation is closed under antireduction)**.**

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

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

Lemma 11 (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 12 (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 13 (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 14 (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 15 (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 $\mathcal{V}\,[\![\mathsf{Bool}]\!]_{\triangledown}$ 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:

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

So we need to prove that $(W, 1, \mathbf{true}) \in \mathcal{V}\,[\![\mathsf{EmulTy}]\!]_{\triangledown}$ 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:

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

(we keep the value as a sum for simplicity)

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

□

Compatibility Lemmas for τ Types

Lemma 16 (Compatibility lemma for calls)**.**

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

Proof. We need to prove that

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

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 7 (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 11 the thesis becomes:

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

  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 4 and by HV and Lemma 6 and by the definition of $\mathcal{G}[\![\cdot]\!]_{\triangledown}$. □

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

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

Proof. This is standard using Lemma 7, Lemma 6, Lemma 12 and Lemma 11. □

Lemma 18 (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 17. □

Lemma 19 (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 17. □

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

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

Proof. This is standard and analogous to the proof of Lemma 17. □

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

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

Proof. This is standard and analogous to the proof of Lemma 17. □

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

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

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

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

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

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

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

By Lemma 12 (Related terms plugged in related contexts are still related) with HE, we have that for HW $W' \sqsupseteq W$, and HV $(W', \mathsf{n}, \mathbf{n}) \in \mathcal{V}\left[\!\left[\mathsf{Nat}\right]\!\right]_\triangledown$, the thesis becomes:

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

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 11 (Closedness under antireduction) the thesis is:

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

So the theorem holds by Lemma 7 (Value relation implies term relation) with HV. □

Semantic Preservation Results

Theorem 36 ($\cdot{\downarrow}^{\mathsf{L}^\tau}_{\mathbf{L^u}}$ is semantics preserving for expressions)**.**

$$\begin{array}{rl} \text{if} & \mathsf{P}; \Gamma \vdash \mathsf{e} : \tau \\ & \vdash \mathsf{P} \ \nabla_n \ \mathbf{P} \\ \text{then} & \forall n.\ \Gamma; \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \ \nabla_n \ \mathsf{e}{\downarrow}^{\mathsf{L}^\tau}_{\mathbf{L^u}} : \tau \end{array}$$

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

true, false, nat By definition of $\mathcal{V}\llbracket\cdot\rrbracket_{\triangledown}$.

var By definition of $\mathcal{G}\llbracket\cdot\rrbracket_{\triangledown}$.

call By Lemma 16 (Compatibility lemma for calls).

app By IH with Lemma 17 (Compatibility lemma for application).

op By IH with Lemma 18 (Compatibility lemma for op).

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

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

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

read By Lemma 22 (Compatibility lemma for read).

write By IH with Lemma 23 (Compatibility lemma for write).

□

Theorem 37 ($\cdot\downarrow^{\mathsf{L}^{\tau}}_{\mathbf{L^u}}$ is semantics preserving for programs)**.**

$$\begin{array}{rl} \text{if} & \vdash \mathsf{P} \\ \text{then} & \vdash \mathsf{P} \triangledown \mathsf{P}\downarrow^{\mathsf{L}^{\tau}}_{\mathbf{L^u}} \end{array}$$

Proof. By induction on the size of P and then Rule ($\cdot\downarrow^{\mathsf{L}^{\tau}}_{\mathbf{L^u}}$-Prog) and with Theorem 36 ($\cdot\downarrow^{\mathsf{L}^{\tau}}_{\mathbf{L^u}}$ is semantics preserving for expressions) on each function body. □

Compatibility Lemmas for Pseudo Types

Lemma 24 (Compatibility lemma for backtranslation of op)**.**

$$\begin{array}{rl} \text{if} & (HE)\ \texttt{toEmul}(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \triangledown_n \mathbf{e} : \mathsf{EmulTy} \\ & (HEP)\ \texttt{toEmul}(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{e}' \triangledown_n \mathbf{e}' : \mathsf{EmulTy} \\ \text{then} & \texttt{toEmul}(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \begin{array}{l}\mathsf{let\ x1 : Nat{=}extract_{Nat}(e)} \\ \quad \mathsf{in\ let\ x2 : Nat{=}extract_{Nat}(e')} \\ \quad \mathsf{in\ inject_{Nat}(x1 \oplus x2)}\end{array} \triangledown_n \mathbf{e} \oplus \mathbf{e}' : \mathsf{EmulTy} \end{array}$$

Proof. We need to prove that

$$\texttt{toEmul}(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \begin{array}{l}\mathsf{let\ x1 : Nat{=}extract_{Nat}(e)} \\ \quad \mathsf{in\ let\ x2 : Nat{=}extract_{Nat}(e')} \\ \quad \mathsf{in\ inject_{Nat}(x1 \oplus x2)}\end{array} \triangledown \mathbf{e} \oplus \mathbf{e}' : \mathsf{EmulTy}$$

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:

- $(W, \begin{array}{l}\mathsf{let\ x1 : Nat{=}extract_{Nat}(e)}\\ \quad\mathsf{in\ let\ x2 : Nat{=}extract_{Nat}(e')}\\ \quad\mathsf{in\ inject_{Nat}(x1 \oplus x2)}\end{array}, \mathbf{e \oplus e'}) \in \mathcal{E}\,[\![\mathsf{EmulTy}]\!]_{\triangledown}$

By Lemma 12 (Related terms plugged in related contexts are still related) with HE we need to prove that $\forall W' \sqsupseteq W$, given IHV $(W', \mathsf{v}, \mathbf{v}) \in \mathcal{V}\,[\![\mathsf{EmulTy}]\!]_{\triangledown}$

- $(W', \begin{array}{l}\mathsf{let\ x1 : Nat{=}extract_{Nat}(v)}\\ \quad\mathsf{in\ let\ x2 : Nat{=}extract_{Nat}(e')}\\ \quad\mathsf{in\ inject_{Nat}(x1 \oplus x2)}\end{array}, \mathbf{v \oplus e'}) \in \mathcal{E}\,[\![\mathsf{EmulTy}]\!]_{\triangledown}$

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

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

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

  In this case we have:

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

  and

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

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

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

  In this case we have:

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

  And by Lemma 14 (If Extract reduces, it preserves relatedness) with IHV we know that IHN $(W', \mathsf{n}, \mathbf{n}) \in \mathcal{V}\,[\![\mathsf{Nat}]\!]_{\triangledown}$.

  Analogously, $\mathsf{e'}$ and $\mathbf{e'}$ follow the same treatment. So we apply Lemma 12 (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 11 (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 $lev(W') = 0$ the thesis follows from Lemma 2 (No steps means relation), otherwise:

By Rule EL$^{\tau}$-op and Rule $\mathbf{EL^{u}}$-op we can apply Lemma 11 (Closedness under antireduction) (with IHN and IHNP in the term relation by Lemma 7 (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 11 (Closedness under antireduction) and then Lemma 7 (Value relation implies term relation) the thesis becomes:

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

By Lemma 15 (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 $\mathcal{V}\,[\![\mathsf{Nat}]\!]_{\triangledown}$.

$\square$

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

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

Proof. Analogous to the proof of Lemma 24. □

Lemma 26 (Compatibility lemma for backtranslation of letin)**.**

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

Proof. This is a trivial application of Lemma 12 (Related terms plugged in related contexts are still related) and Lemma 11 (Closedness under antireduction) and definitions. □

Lemma 27 (Compatibility lemma for backtranslation of if)**.**

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

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 12 (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:

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

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

The reductions proceed as follows:

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

By Lemma 11 (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 2 (No steps means relation), otherwise:

We can reduce based on Rules EL^{τ}-if-true and $\mathbf{EL^{u}}$-if-true. By Lemma 11 (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 2 (No steps means relation), otherwise by HEP.

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

  In this case we have that:

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

  and

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

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

□

Lemma 28 (Compatibility lemma for backtranslation of application)**.**

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

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

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

By Lemma 12 (Related terms plugged in related contexts are still related) with HE we have that for HW $W' \sqsupseteq W$, and HV $(W', \mathsf{v}, \mathbf{v}) \in \mathcal{V}\,[\![\mathsf{EmulTy}]\!]_{\nabla}$, the thesis becomes:

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

We perform a case analysis based on HV:

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

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

  We perform a case analysis on τ:

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

    The thesis is:

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

    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 11 (Closedness under antireduction) the thesis becomes:

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

    If the $lev(W') = 0$ the thesis follows from Lemma 2 (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 6 (Monotonicity for value relation) we have that HF:

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

    By Lemma 12 (Related terms plugged in related contexts are still related) with HF we have that for HW $W'' \sqsupseteq W'$, and HV $(W'', \mathsf{v}', \mathbf{v}') \in \mathcal{V}\,[\![\tau']\!]_{\nabla}$, the thesis becomes:

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

  This case follows from Lemma 7 (Value relation implies term relation) and by Lemma 15 (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 $(\cdot\downarrow^{\mathsf{L}^{\tau}}_{\mathbf{L^u}}$-Fun):

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

  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 29 (Compatibility lemma for backtranslation of check)**.**

if (HE) `toEmul` $(\mathbf{\Gamma}); \mathsf{P}; \mathbf{P} \vdash \mathsf{e} \triangledown_n \mathbf{e} : \mathsf{EmulTy}$

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

Proof. We need to prove that

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

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 12 (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 v=$\mathsf{1}/\mathsf{0}$ (respectively).

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

  We have that

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

  and

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

  This case holds by Lemma 11 (Closedness under antireduction) and Lemma 7 (Value relation implies term relation) and by the definition of $\mathcal{V}\llbracket \mathsf{EmulTy} \rrbracket_{\triangledown}$.

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

  In this case we have that:

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

  and

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

  This case holds by Lemma 11 (Closedness under antireduction) and Lemma 7 (Value relation implies term relation) and by the definition of $\mathcal{V}\llbracket \mathsf{EmulTy} \rrbracket_{\triangledown}$.

□

Semantic Preservation of Backtranslation

Theorem 38 ($\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L^\tau}}$ is semantics preserving)**.**

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

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 28 (Compatibility lemma for backtranslation of application).

op By IH and Lemma 24 (Compatibility lemma for backtranslation of op).

geq Analogous to the case above.

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

letin By IH and Lemma 26 (Compatibility lemma for backtranslation of letin).

check By IH and Lemma 29 (Compatibility lemma for backtranslation of check).

□

Theorems that Yield RRHP

Theorem 39 ($\cdot\downarrow^{\mathsf{L^\tau}}_{\mathbf{L^u}}$ preserves behaviours)**.**

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

Proof. By Theorem 37 ($\cdot\downarrow^{\mathsf{L^\tau}}_{\mathbf{L^u}}$ is semantics preserving for programs) we have HPP:

- $\vdash \mathsf{P} \triangledown \mathsf{P}\downarrow^{\mathsf{L^\tau}}_{\mathbf{L^u}}$

Given that $\varnothing \vdash \mathbf{e}$, by Theorem 38 ($\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L^\tau}}$ is semantics preserving) with HPP we have HPE:

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

The thesis follows by Lemma 9 (Adequacy for $\gtrsim$) with HT. □

Theorem 40 ($\cdot\downarrow^{\mathsf{L}^\tau}_{\mathbf{L^u}}$ reflects behaviours)**.**

$$\text{if } (HS)\ \mathsf{P} \triangleright \langle\!\langle \mathbf{e} \rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau} \overset{\beta}{\Longrightarrow} \mathsf{P} \triangleright \mathsf{e}'$$
$$\text{then } \mathsf{P}\downarrow^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright \mathbf{e} \overset{\beta}{\Longrightarrow} \mathsf{P}\downarrow^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright \mathbf{e}'$$

Proof. By Theorem 37 ($\cdot\downarrow^{\mathsf{L}^\tau}_{\mathbf{L^u}}$ is semantics preserving for programs) we have HPP:

- $\vdash \mathsf{P} \triangledown \mathsf{P}\downarrow^{\mathsf{L}^\tau}_{\mathbf{L^u}}$

Given that $\varnothing \vdash \mathbf{e}$, by Theorem 38 ($\langle\!\langle\cdot\rangle\!\rangle^{\mathbf{L^u}}_{\mathsf{L}^\tau}$ is semantics preserving) with HPP we have HPE:

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

The thesis follows by Lemma 8 (Adequacy for $\lesssim$) with HS. □

5.4.3 Proof that $\cdot\downarrow^{\mathsf{L}^\tau}_{\mathbf{L^u}}$ satisfies Definition 20 (RrHP′)

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

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

$\Rightarrow$ **direction** By Theorem 39 ($\cdot\downarrow^{\mathsf{L}^\tau}_{\mathbf{L^u}}$ preserves behaviours)

$\Leftarrow$ **direction** By Theorem 40 ($\cdot\downarrow^{\mathsf{L}^\tau}_{\mathbf{L^u}}$ reflects behaviours).

5.5 Proof that $\cdot\downarrow^{\mathsf{L}^\tau}_{\mathbf{L^u}}$ 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.*

5.5.1 Overview of the proof technique

We have proved the following theorem for our instance:

Theorem 41 (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:

$$\begin{aligned}
&(\forall i, \mathbf{C_T}[\mathsf{P}_i\downarrow] \rightsquigarrow m_i)\\
\Longrightarrow\ &(\exists \mathsf{C_S}, \forall i, \mathsf{C_S}[\mathsf{P}_i] \rightsquigarrow m_i)
\end{aligned}$$

$$\begin{tikzcd}
\forall i, \mathsf{C_S}[\mathsf{P}_i] \rightsquigarrow m_i & \forall i, \mathsf{C_S}[\mathsf{P}_i] \hookrightarrow \mu_i^s \arrow[l, "\text{Uninf}"'] & \forall i, \mathsf{P}_i \hookrightarrow_{\text{prg}} \mu_i \arrow[l, "\text{Comp}"'] & \forall i, \mathsf{C_S} \hookrightarrow_{\text{ctx}} \mu_i^s \arrow[ll, bend right, "\text{Comp}"'] \\
\forall i, \mathbf{C_T}[\mathsf{P}_i{\downarrow}] \rightsquigarrow m_i \arrow[r, "\text{Inf}"] & \forall i, \exists \mu_i \sqsupseteq m_i, \mathbf{C_T}[\mathsf{P}_i{\downarrow}] \hookrightarrow \mu_i \arrow[r, "\text{Dec}"] \arrow[rr, bend right, "\text{Dec}"'] & \forall i, \mathsf{P}_i{\downarrow} \hookrightarrow_{\text{prg}} \mu_i \arrow[u, "\text{BCC}"] & \forall i, \mathbf{C_T} \hookrightarrow_{\text{ctx}} \mu_i \arrow[u, "\text{BT"}]
\end{tikzcd}$$

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.

5.5.2 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:

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

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

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

Theorem 42 (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

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

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 $\mathbf{P_T} \triangleright \mathbf{C_T} \xRightarrow{\mathbf{m}} \mathbf{P_T} \triangleright \mathbf{e}$, then there exists μ such that $|\mu|_{\mathrm{I/O}} = m$ and $\mathbf{P_T} \triangleright \mathbf{C_T} \overset{\mu}{\Rrightarrow} \mathbf{P_T} \triangleright \mathbf{e}$.

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

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 $\mathbf{P_T} \triangleright \mathbf{C_T} \xRightarrow{\mathbf{m_1}} \mathbf{P_T} \triangleright \mathbf{e'}$ and $\mathbf{P_T} \triangleright \mathbf{e'} \xRightarrow{\mathbf{m_2}} \mathbf{P_T} \triangleright \mathbf{e}$ with $m = m_1 m_2$. By applying the induction hypothesis, there exists μ_1 and μ_2 such that $\mathbf{P_T} \triangleright \mathbf{C_T} \overset{\mu_1}{\Rrightarrow} \mathbf{e'}$, $\mathbf{P_T} \triangleright \mathbf{e'} \overset{\mu_2}{\Rrightarrow} \mathbf{e}$, $|\mu_1|_{\mathrm{I/O/termination}} = m_1$, and $|\mu_2|_{\mathrm{I/O/termination}} = m_2$.

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

□

5.5.3 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:

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

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

and the relations $\overset{\cdot}{\Longrightarrow}_{\text{ctx}}$ and $\overset{\cdot}{\Longrightarrow}_{\text{ctx}}$ 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 $P \hookrightarrow_{\text{prg}} \mu$ 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 `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 `call` $f\ v?$ and stopping at the latest at the next (corresponding) return event, it must be that $P \triangleright \mathit{call}\ f\ v \hookrightarrow \mu'$.

Definition 39 (Partial semantics for programs)**.** $\mathsf{P} \hookrightarrow_{\text{prg}} \mu$ if and only if:

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

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

- 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 $\mathsf{f} : \tau \to \tau' \in \bar{\mathsf{I}}$ with $\mathsf{v} \in \tau$, there exists e such that

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

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

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

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

- 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

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

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 43 (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 30. 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,

$$\mathbf{C_T}\left[\mathbf{P_T}\right] \overset{\mu}{\Longrightarrow\!\!\!\!\!\!\twoheadrightarrow} \mathbf{e} \implies \mathbf{C_T} \overset{\mu}{\Longrightarrow\!\!\!\!\!\!\twoheadrightarrow}_{\mathrm{ctx}} \mathbf{e}$$

Proof. By induction on the relation $\mathbf{C_T}[\mathbf{P_T}] \overset{\mu}{\Longrightarrow\!\!\!\!\!\!\twoheadrightarrow} \mathbf{P_T} \triangleright \mathbf{e}$

Rule $\mathbf{EL^u}$**-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 $\mathbf{EL^u}$**-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, $\forall n, \exists \mathbf{e_n}, \mathbf{C_T}[\mathbf{P_T}] \xrightarrow{\epsilon}^{n} \mathbf{P_T} \triangleright \mathbf{e_n}$. 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 $\mathbf{EL^u}$**-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 $\mathbf{EL^u}$**-cons** We have that $\mathbf{P_T} \triangleright \mathbf{C_t} \overset{\mu_1}{\Longrightarrow\!\!\!\!\!\!\twoheadrightarrow}_{\mathrm{ctx}} \mathbf{e'}$ and $\mathbf{P_T} \triangleright \mathbf{e'} \overset{\mu_2}{\Longrightarrow\!\!\!\!\!\!\twoheadrightarrow}_{\mathrm{ctx}} \mathbf{e}$. Then, by applying the induction hypothesis to the two relations, we are done.

□

Then, we prove a similar lemma for programs:

Lemma 31. 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 $\mathbf{P} \triangleright \mathbb{E}[\mathbf{call\ f\ v}] \xRightarrow{\mu_{f,v,v'}}_{\text{ctx}} \mathbb{E}[\mathbf{v'}]$.

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

5.5.4 Backward Compiler Correctness for Programs

Theorem 44 (Backward Compiler Correctness)**.** Let P be a source program. Then,

$$\forall \mu, \mathsf{P}{\downarrow}^{\mathsf{L}^\tau}_{\mathbf{L^u}} \hookrightarrow_{\text{prg}} \mu \implies \mathsf{P} \hookrightarrow_{\text{prg}} \mu.$$

Before proving the theorem, we state a preliminary lemma:

Lemma 32. Suppose that $\mathsf{P}{\downarrow}^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright\ \mathbf{call\ f\ v} \xRightarrow{\texttt{call f v?};\mu} \mathsf{P}{\downarrow}^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright \mathbf{e'}$ where the call is well-typed.

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

- $\mathsf{P}{\downarrow}^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright\ \mathsf{e}{\downarrow}^{\mathsf{L}^\tau}_{\mathbf{L^u}}[\mathbf{x}/\mathbf{v}] \xRightarrow{\mu} \mathsf{P}{\downarrow}^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright \mathbf{e'}$,

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

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

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

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

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

and

$$\mathsf{P}{\downarrow}^{\mathsf{L}^\tau}_{\mathbf{L^u}} \triangleright\ \mathbf{e_1} \xRightarrow{\mu_2} \mathbf{e}.$$

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

Other cases: these cases are impossible

□

We can now prove the backward compiler correctness theorem:

Theorem 44 (Backward Compiler Correctness)**.** Let P be a source program. Then,

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

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

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

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

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

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

Now, we can conclude by induction on e.

□

5.5.5 Back-Translation of a finite set of finite trace prefixes

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

Theorem 45. 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:

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

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

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

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

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

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

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 40 (Backtranslation of the tree T)**.**

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

Lemma 33. 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 41 (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 $\mu = \texttt{call}\ f\ v?; \Uparrow$, $T = (\texttt{call}\ f\ v?, \dots)$ and $\text{type}(v) = \text{input\_type}(f)$
7. $\mu = \texttt{call}\ f\ v? :: \texttt{ret}\ v'! :: \mu'$, $T = (\texttt{call}\ f\ v?, (v_1, T_1), \dots, (v_i, T_i))$, 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 42.

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

Theorem 46 (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:

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

and therefore by substituting and application of the partial semantics:

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

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 34. 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.

□

5.5.6 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 47 (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 $\mathsf{C_S} \hookrightarrow_{\mathrm{ctx}} \mu_i^s$ and $\mathsf{P_S} \hookrightarrow_{\mathrm{prg}} \mu_i$, then $\mathsf{C_S}\left[\mathsf{P_S}\right] \hookrightarrow \mu_i^s$.

We state a preliminary lemma:

Lemma 35. If $\mathsf{P} \hookrightarrow_{\mathrm{prg}} \mu_i$, then $\mathsf{P} \hookrightarrow_{\mathrm{prg}} \mu_i^s$.

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

Lemma 36. 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 $\mathsf{P_S}{\downarrow}^{\mathsf{L}^\tau}_{\mathbf{L^u}}$ and some target context, and e an expression. Then, if $\mathsf{C_S} \xrightarrow{\mu_i^s}\!\!\!\!\twoheadrightarrow \mathsf{e}$ and $\mathsf{P_s} \hookrightarrow_{\mathrm{prg}} \mu_i^s$, then $\mathsf{P_S} \triangleright \mathsf{C_S} \xrightarrow{\mu_i^s}\!\!\!\!\twoheadrightarrow \mathsf{e}'$ where $\mathsf{C_S} \xrightarrow{\mu_i^s}\!\!\!\!\twoheadrightarrow \mathsf{e}'$.

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

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 $\mathsf{e} \hookrightarrow_{\mathrm{ctx}} \mu$ and $\mathsf{P_S} \hookrightarrow_{\mathrm{prg}} \mu$.

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 $\hookrightarrow_{\mathrm{prg}}$.

- First, $\mathsf{e} \hookrightarrow_{\mathrm{ctx}} \mu_1$ and $\mathsf{e} \hookrightarrow_{\mathrm{prg}} \mu_1$, by definition of these relations. Therefore, by induction hypothesis, $\exists \mathsf{e}', \mathsf{e} \xrightarrow{\mu_1}\!\!\!\!\twoheadrightarrow \mathsf{e}'$ and $\mathsf{P_S} \triangleright \mathsf{e} \xrightarrow{\mu_1}\!\!\!\!\twoheadrightarrow \mathsf{e}'$. In particular, 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 $\mathsf{e}' \hookrightarrow_{\mathrm{ctx}} \mu_2\mu_3$.
- We have that $\mathsf{P_S} \triangleright \mathsf{e}' \xrightarrow{\mu_2}\!\!\!\!\twoheadrightarrow \mathbb{E}\left[\mathsf{v}'\right]$ 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: $\mathsf{P_S} \triangleright \mathsf{e} \xrightarrow{\mu_1\mu_2\mu_3}\!\!\!\!\twoheadrightarrow \mathsf{e}''$ where $\mathsf{e} \xrightarrow{\mu_1\mu_2\mu_3}\!\!\!\!\twoheadrightarrow \mathsf{e}''$.

□

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

5.5.7 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 48 (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, $\mathsf{C_S}\,[\mathsf{P_S}] \hookrightarrow \mu^s \implies \mathsf{C_S}\,[\mathsf{P_S}] \rightsquigarrow m$.

The proof is immediate by definition of μ^s.

5.5.8 Proving the secure compilation criterion

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

Theorem 41 (k-Relational Robust Safety Preservation)**.** Let $\mathsf{P}_1 \dots \mathsf{P}_k$ be k programs that share the same interface $\bar{\mathsf{I}}$ and $m_1 \dots m_k$ be k finite trace prefixes. Then, for all target contexts $\mathbf{C_T}$, the following holds:

$$(\forall i, \mathbf{C_T}[\mathsf{P}_i{\downarrow}] \rightsquigarrow m_i)$$
$$\implies (\exists \mathsf{C_S}, \forall i, \mathsf{C_S}[\mathsf{P}_i] \rightsquigarrow m_i)$$

The proof follows the scheme depicted by Figure 1.

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

$$\forall i, \mathbf{C_T}[\mathsf{P}_i{\downarrow}] \rightsquigarrow m_i$$

We can pass to informative traces by applying Theorem 42 to each m_i

$$\forall i, \exists \mu_i \sqsupseteq m, \mathbf{C_T}[\mathsf{P}_i{\downarrow}] \hookrightarrow \mu_i.$$

From here, we can apply the decomposition theorem (Theorem 43) to each μ_i:

$$\forall i, \mathbf{C_T} \hookrightarrow_{\text{ctx}} \mu_i \wedge \mathsf{P}_i{\downarrow} \hookrightarrow_{\text{prg}} \mu_i.$$

By the backward compiler correctness theorem (Theorem 44) 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 47) 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 48):

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

□

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.