Oxide: The Essence of Rust

ANONYMOUS AUTHOR(S)

Text of abstract . . . .

CCS Concepts: • **Theory of computation → Semantics and reasoning**; • **Software and its engineering → Formal language definitions**;

Additional Key Words and Phrases: Rust, type systems, semantics

ACM Reference Format:
Anonymous Author(s). 2018. Oxide: The Essence of Rust. *Proc. ACM Program. Lang.* 1, PL, Article 1 (October 2018), 25 pages.

1 INTRODUCTION

Programming languages have long been divided between “systems” languages, which enable low-level reasoning that has proven critical to writing performant software, and “high-level” languages which empower programmers with high-level abstractions to write software more safely and more quickly. For many language researchers then, a natural goal has been to try to enable both low-level reasoning and high-level abstractions in one language. To date, the Rust programming language from Mozilla Research has been the most successful endeavour toward such a goal.

2 LANGUAGE

2.1 Grammars

Identifiers	x	Region Identifiers	ρ	Loan Identifiers	ℓ
Struct Names	S	Enum Variants	E		

$$
\begin{array}{llcl}
\text{Mutability Quantifiers} & \mu & ::= & \texttt{imm} \mid \texttt{mut} \\
\text{Places} & \pi & ::= & x \mid *\,\pi \mid \pi.x \mid \pi.n \mid \pi[n] \\
\\
\text{Fractions} & f & ::= & n \mid {}^{f}/_{f} \mid f + f \\
\text{Regions} & r & ::= & \rho \mid \{\, \ell_1 \,\ldots\, \ell_n \,\} \\
\\
\text{Kinds} & \kappa & ::= & \star \mid \texttt{RGN} \\
\text{Base Types} & \tau_B & ::= & \texttt{bool} \mid \texttt{u32} \mid \texttt{unit} \\
\text{Types} & \tau & ::= & \tau_B \mid \alpha \mid \&r\,\mu\,\tau \\
 & & \mid & \texttt{fn<}\rho_1 \,\ldots\, \rho_n,\ \alpha_1 \,\ldots\, \alpha_n\texttt{>}(\tau_1 \,\ldots\, \tau_n) \xrightarrow{\varepsilon} \tau_r \\
 & & \mid & [\tau;\ n] \mid [\tau] \mid (\tau_1 \,\ldots\, \tau_n) \mid S :: \texttt{<}\tau_1 \,\ldots\, \tau_n\texttt{>} \\
\text{Effects} & \varepsilon & ::= & \texttt{borrow}\,\mu\,\pi\,\texttt{as}\,\ell \mid \texttt{drop}\,\pi
\end{array}
$$

2018. 2475-1421/2018/10-ART1 $15.00
https://doi.org/

Primitives	pv	$::=$	$() \mid n \mid \texttt{true} \mid \texttt{false}$
Expressions	e	$::=$	$pv \mid \pi \mid \&\ell\ \mu\ \pi \mid \texttt{drop}(\pi)$
		$\mid$	$\texttt{let}\ x\ :\ \tau = e_1;\ e_2 \mid \pi := e \mid e_1;\ e_2$
		$\mid$	$\texttt{forall<}\rho_1\ \ldots\ \rho_n,\ \alpha_1\ \ldots\ \alpha_n\texttt{>}\ \lvert x_1\ :\ \tau_1\ \ldots\ x_n\ :\ \tau_n \rvert\ \{\ e\ \}$
		$\mid$	$e_f :: \texttt{<}r_1\ \ldots\ r_n,\ \tau_1\ \ldots\ \tau_n\texttt{>}(e_1\ \ldots\ e_n)$
		$\mid$	$\texttt{if}\ e_1\ \{\ e_2\ \}\ \texttt{else}\ \{\ e_3\ \}$
		$\mid$	$\texttt{match}\ e_d\ \{\ pat_1\ \Rightarrow\ e_1\ \ldots\ pat_n\ \Rightarrow\ e_n\ \}$
		$\mid$	$\texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \}$
		$\mid$	$(e_1\ \ldots\ e_n) \mid [e_1\ \ldots\ e_n]$
		$\mid$	$S :: \texttt{<}r_1\ \ldots\ r_n,\ \tau_1\ \ldots\ \tau_n\texttt{>}\ \{\ x_1 :\ e_1\ \ldots\ x_n :\ e_n\ \}$
		$\mid$	$S :: \texttt{<}r_1\ \ldots\ r_n,\ \tau_1\ \ldots\ \tau_n\texttt{>}(e_1\ \ldots\ e_n)$
		$\mid$	$S :: \texttt{<}r_1\ \ldots\ r_n,\ \tau_1\ \ldots\ \tau_n\texttt{>} :: E\ \{\ x_1 :\ e_1\ \ldots\ x_n :\ e_n\ \}$
		$\mid$	$S :: \texttt{<}r_1\ \ldots\ r_n,\ \tau_1\ \ldots\ \tau_n\texttt{>} :: E(e_1\ \ldots\ e_n)$
Patterns	pat	$::=$	$\_$
		$\mid$	$(x_1\ \ldots\ x_n)$
		$\mid$	$S(x_1\ \ldots\ x_n)$
		$\mid$	$S\ \{\ x_{f_1}\ :\ x_1\ \ldots\ x_{f_n}\ :\ x_n\ \}$
		$\mid$	$S :: E(x_1\ \ldots\ x_n)$
		$\mid$	$S :: E\ \{\ x_{f_1}\ :\ x_1\ \ldots\ x_{f_n}\ :\ x_n\ \}$
Values	v	$::=$	pv
		$\mid$	$\texttt{forall<}\rho_1\ \ldots\ \rho_n,\ \alpha_1\ \ldots\ \alpha_n\texttt{>}\ \lvert x_1\ :\ \tau_1\ \ldots\ x_n\ :\ \tau_n \rvert\ \{\ e\ \}$
		$\mid$	$(v_1\ \ldots\ v_n) \mid [v_1\ \ldots\ v_n]$
		$\mid$	$S :: \texttt{<}r_1\ \ldots\ r_n,\ \tau_1\ \ldots\ \tau_n\texttt{>}\ \{\ x_1 :\ v_1\ \ldots\ x_n :\ v_n\ \}$
		$\mid$	$S :: \texttt{<}r_1\ \ldots\ r_n,\ \tau_1\ \ldots\ \tau_n\texttt{>}(v_1\ \ldots\ v_n)$
		$\mid$	$S :: \texttt{<}r_1\ \ldots\ r_n,\ \tau_1\ \ldots\ \tau_n\texttt{>} :: E\ \{\ x_1 :\ v_1\ \ldots\ x_n :\ v_n\ \}$
		$\mid$	$S :: \texttt{<}r_1\ \ldots\ r_n,\ \tau_1\ \ldots\ \tau_n\texttt{>} :: E(v_1\ \ldots\ v_n)$
Explicit Path Values	ν	$::=$	pv
		$\mid$	$\texttt{forall<}\rho_1\ \ldots\ \rho_n,\ \alpha_1\ \ldots\ \alpha_n\texttt{>}\ \lvert x_1\ :\ \tau_1\ \ldots\ x_n\ :\ \tau_n \rvert\ \{\ e\ \}$
		$\mid$	$(\square_1\ \ldots\ \square_n) \mid [\square_1\ \ldots\ \square_n]$
		$\mid$	$S :: \texttt{<}r_1\ \ldots\ r_n,\ \tau_1\ \ldots\ \tau_n\texttt{>}\ \{\ x_1 :\ \square_1\ \ldots\ x_n :\ \square_n\ \}$
		$\mid$	$Sr_1\ \ldots\ r_n,\ \tau_1\ \ldots\ \tau_n :: \texttt{<(>}\square_1\ \ldots\ \square_n)$
		$\mid$	$Sr_1\ \ldots\ r_n,\ \tau_1\ \ldots\ \tau_n :: \texttt{<} :: \texttt{>}E\ \{\ x_1 :\ \square_1\ \ldots\ x_n :\ \square_n\ \}$
		$\mid$	$S :: \texttt{<}r_1\ \ldots\ r_n,\ \tau_1\ \ldots\ \tau_n\texttt{>} :: E(\square_1\ \ldots\ \square_n)$
Enum Variants	ev	$::=$	$E(\tau_1\ \ldots\ \tau_n) \mid E\ \{\ x_1\ :\ \tau_1\ \ldots\ x_n\ :\ \tau_n\ \}$
Global Context	Σ	$::=$	$\bullet$
		$\mid$	$\Sigma,\ \texttt{struct}\ S\texttt{<}\overline{\rho},\ \overline{\alpha}\texttt{>}(\tau_1\ \ldots\ \tau_n)$
		$\mid$	$\Sigma,\ \texttt{struct}\ S\texttt{<}\overline{\rho},\ \overline{\alpha}\texttt{>}\ \{\ x_1\ :\ \tau_1\ \ldots\ x_n\ :\ \tau_n\ \}$
		$\mid$	$\Sigma,\ \texttt{enum}\ S\texttt{<}\overline{\rho},\ \overline{\alpha}\texttt{>}\ \{\ ev_1\ \ldots\ ev_n\ \}$
Type Variable Context	Δ	$::=$	$\bullet \mid \Delta,\ \rho : \texttt{RGN} \mid \Delta,\ \alpha : \star$
Variable Context	Γ	$::=$	$\bullet \mid \Gamma,\ \pi\ :_f\ \tau$
Loan Context	$\mathcal{L}$	$::=$	$\bullet \mid \mathcal{L},\ \ell \mapsto_f \pi$

2.2 Statics

$$\boxed{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e : \tau \Rightarrow \varepsilon}$$

$$\textsc{T-Move}\quad \frac{\pi \;:_1\; \tau \in \Gamma}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \pi : \tau \Rightarrow \texttt{drop}\ \pi} \qquad\qquad \textsc{T-Copy}\quad \frac{\pi \;:_f\; \tau \in \Gamma \qquad \texttt{copyable}\ \tau}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \pi : \tau \Rightarrow \diamond}$$

$$\textsc{T-BorrowImm}\quad \frac{\ell \notin \mathcal{L} \qquad \pi \;:_f\; \tau \in \Gamma \qquad f \neq 0}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \&\ell\ \texttt{imm}\ \pi : \&\{\ \ell\ \}\ \texttt{imm}\ \tau \Rightarrow \texttt{borrow imm}\ \pi\ \texttt{as}\ \ell}$$

$$\textsc{T-BorrowMut}\quad \frac{\ell \notin \mathcal{L} \qquad \pi \;:_1\; \tau \in \Gamma}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \&\ell\ \texttt{mut}\ \pi : \&\{\ \ell\ \}\ \texttt{mut}\ \tau \Rightarrow \texttt{borrow mut}\ \pi\ \texttt{as}\ \ell}$$

$$\textsc{T-Drop}\quad \frac{\pi \;:_1\; \tau \in \Gamma}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{drop}(\pi) : \texttt{unit} \Rightarrow \texttt{drop}\ \pi} \qquad\qquad \textsc{T-Seq}\quad \frac{\begin{array}{c}\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : \tau_1 \Rightarrow \varepsilon_1 \\ \Sigma; \Delta; \varepsilon_1(\Gamma; \mathcal{L}) \vdash e_2 : \tau_2 \Rightarrow \varepsilon_2\end{array}}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1;\ e_2 : \tau_2 \Rightarrow \varepsilon_1, \varepsilon_2}$$

$$\textsc{T-Branch}\quad \frac{\begin{array}{c}\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : \texttt{bool} \Rightarrow \varepsilon_1 \\ \Sigma; \Delta; \varepsilon_1(\Gamma; \mathcal{L}) \vdash e_2 : \tau_2 \Rightarrow \varepsilon_2 \\ \Sigma; \Delta; \varepsilon_1(\Gamma; \mathcal{L}) \vdash e_3 : \tau_3 \Rightarrow \varepsilon_3 \\ \tau_2 \sim \tau_3 \Rightarrow \tau\end{array}}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{if}\ e_1\ \{\ e_2\ \}\ \texttt{else}\ \{\ e_3\ \}\ : \tau \Rightarrow \varepsilon_1, \varepsilon_2, \varepsilon_3}$$

$$\textsc{T-Let}\quad \frac{\begin{array}{c}\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : \tau_1 \Rightarrow \varepsilon_1 \qquad \tau = \tau_1 \\ \tau \neq \&r\ \mu\ \tau' \qquad \tau_1 \neq \&r\ \mu\ \tau' \\ \texttt{compute-places}(x,\ \tau) = \overline{\pi :_1 \tau} \\ \Sigma; \Delta; \varepsilon_1(\Gamma,\ \overline{\pi :_1 \tau}; \mathcal{L}) \vdash e_2 : \tau_2 \Rightarrow \varepsilon_2\end{array}}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{let}\ x\ :\ \tau = e_1;\ e_2 : \tau_2 \Rightarrow \varepsilon_1, \varepsilon_2} \qquad \textsc{T-Assign}\quad \frac{\pi \;:_1\; \tau \in \Gamma \qquad \Sigma; \Delta; \Gamma; \mathcal{L} \vdash e : \tau \Rightarrow \varepsilon}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \pi \coloneqq e : \texttt{unit} \Rightarrow \varepsilon}$$

$$\textsc{T-LetRef}\quad \frac{\begin{array}{c}\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : \tau_1 \Rightarrow \varepsilon_1 \qquad \tau = \&\rho\ \mu\ \tau' \qquad \tau_1 = \&\{\ \overline{\ell}\ \}\ \mu\ \tau' \\ \texttt{compute-places}(x,\ \tau) = \overline{\pi :_1 \tau} \\ \Sigma; \Delta; \varepsilon_1(\Gamma,\ \overline{\pi :_1 \tau}; \mathcal{L}) \vdash e_2[{}^{\{\ \overline{\ell}\ \}}/_{\rho}] : \tau_2 \Rightarrow \varepsilon_2\end{array}}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{let}\ x\ :\ \tau = e_1;\ e_2 : \tau_2 \Rightarrow \varepsilon_1, \varepsilon_2}$$

$$\textsc{T-ForArray}\quad \frac{\begin{array}{c}\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : [\tau;\ n] \Rightarrow \varepsilon_1 \qquad \varepsilon_2 = \diamond \\ \Sigma; \Delta; \varepsilon_1(\Gamma,\ x\ :_1\ \tau; \mathcal{L}) \vdash e_2 : \tau_2 \Rightarrow \varepsilon_2\end{array}}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \}\ : \texttt{unit} \Rightarrow \varepsilon_1} \qquad \textsc{T-ForSlice}\quad \frac{\begin{array}{c}\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : \&r\ \mu\ [\tau] \Rightarrow \varepsilon_1 \qquad \varepsilon_2 = \diamond \\ \Sigma; \Delta; \varepsilon_1(\Gamma,\ x\ :_\mu\ \tau; \mathcal{L}) \vdash e_2 : \tau_2 \Rightarrow \varepsilon_2\end{array}}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \}\ : \texttt{unit} \Rightarrow \varepsilon_1}$$

T-Closure

$$\frac{\begin{array}{c}\Sigma; \Delta,\ \overline{\rho : \texttt{RGN}}, \overline{\alpha : \star}; \Gamma,\ \texttt{compute-places}(x_1,\ \tau_1)\ \ldots\ \texttt{compute-places}(x_n,\ \tau_n); \mathcal{L} \vdash e : \tau_f \Rightarrow \varepsilon_f \\ \texttt{fvars}(e) - \{\ x_1\ \ldots\ x_n\ \} = \overline{x}_{cap}\end{array}}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{forall<}\overline{\rho},\ \overline{\alpha}\texttt{>}\ |x_1\ :\ \tau_1\ \ldots\ x_n\ :\ \tau_n|\ \{\ e\ \}\ : \texttt{fn<}\overline{\rho},\ \overline{\alpha}\texttt{>}(\tau_1\ \ldots\ \tau_n) \xrightarrow{\varepsilon_f} \tau_f \Rightarrow \texttt{drop}\ \overline{x}_{cap}}$$

T-App

$$\frac{\begin{array}{c}\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_f : \texttt{fn<}\overline{\rho},\ \overline{\alpha}\texttt{>}(\tau_1\ \ldots\ \tau_n) \xrightarrow{\varepsilon_f} \tau_f \Rightarrow \varepsilon \\ \Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : \tau_1[\overline{r}/\overline{\rho}][\overline{\tau}/\overline{\alpha}] \Rightarrow \varepsilon_1 \\ \ldots \\ \Sigma; \Delta; (\varepsilon_1,\ \ldots, \varepsilon_{n-1})(\Gamma; \mathcal{L}) \vdash e_n : \tau_n[\overline{r}/\overline{\rho}][\overline{\tau}/\overline{\alpha}] \Rightarrow \varepsilon_n\end{array}}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_f :: \texttt{<}\overline{r},\ \overline{\tau}\texttt{>}(e_1\ \ldots\ e_n) : \tau_f[\overline{r}/\overline{\rho}][\overline{\tau}/\overline{\alpha}] \Rightarrow \varepsilon, \varepsilon_1,\ \ldots, \varepsilon_n, \varepsilon_f}$$

T-Unit

$$\frac{}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{()} : \texttt{unit} \Rightarrow \diamond}$$

T-u32

$$\frac{}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash n : \texttt{u32} \Rightarrow \diamond}$$

T-True

$$\frac{}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{true} : \texttt{bool} \Rightarrow \diamond}$$

T-False

$$\frac{}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{false} : \texttt{bool} \Rightarrow \diamond}$$

T-Tuple

$$\frac{\begin{array}{c}\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : \tau_1 \Rightarrow \varepsilon_1 \qquad \ldots \\ \Sigma; \Delta; (\varepsilon_1,\ \ldots, \varepsilon_{n-1})(\Gamma; \mathcal{L}) \vdash e_n : \tau_n \Rightarrow \varepsilon_n\end{array}}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash (e_1\ \ldots\ e_n) : (\tau_1\ \ldots\ \tau_n) \Rightarrow \varepsilon_1,\ \ldots, \varepsilon_n}$$

T-Array

$$\frac{\begin{array}{c}\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : \tau \Rightarrow \varepsilon_1 \qquad \ldots \\ \Sigma; \Delta; (\varepsilon_1,\ \ldots, \varepsilon_{n-1})(\Gamma; \mathcal{L}) \vdash e_n : \tau_n \Rightarrow \varepsilon_n\end{array}}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash [e_1\ \ldots\ e_n] : [\tau;\ n] \Rightarrow \varepsilon_1,\ \ldots, \varepsilon_n}$$

2.3 Additional Typing Rules for the Dynamics

T-PointerImm

$$\frac{f \neq 0}{\Sigma; \Delta; \Gamma,\ \pi\ :_{f_\pi}\ \tau; \mathcal{L},\ \ell \mapsto_f \pi \vdash \texttt{ptr}\ \ell : \&\{\ \ell\ \}\ \texttt{imm}\ \tau \Rightarrow \diamond}$$

T-PointerMut

$$\frac{}{\Sigma; \Delta; \Gamma,\ \pi\ :_0\ \tau; \mathcal{L},\ \ell \mapsto_1 \pi \vdash \texttt{ptr}\ \ell : \&\{\ \ell\ \}\ \texttt{mut}\ \tau \Rightarrow \diamond}$$

T-FatPointerImm

$$\frac{f \neq 0 \qquad n_1 \leq n_2}{\Sigma; \Delta; \Gamma,\ \pi\ :_{f_\pi}\ \tau; \mathcal{L},\ \ell \mapsto_f \pi \vdash \texttt{fatptr}\ \ell[n_1..n_2]) : \&\{\ \ell\ \}\ \texttt{imm}\ [\tau] \Rightarrow \diamond}$$

T-FatPointerMut

$$\frac{n_1 \leq n_2}{\Sigma; \Delta; \Gamma,\ \pi\ :_0\ \tau; \mathcal{L},\ \ell \mapsto_1 \pi \vdash \texttt{fatptr}\ \ell[n_1..n_2]) : \&\{\ \ell\ \}\ \texttt{mut}\ [\tau] \Rightarrow \diamond}$$

2.4 Type Unification

$\boxed{\tau_1 \sim \tau_2 \Rightarrow \tau}$

$$\textsc{U-Symmetry}\ \dfrac{\tau_2 \sim \tau_1 \Rightarrow \tau'}{\tau_1 \sim \tau_2 \Rightarrow \tau'} \qquad \textsc{U-BaseTypes}\ \dfrac{}{\tau_B \sim \tau_B \Rightarrow \tau_B}$$

$$\textsc{U-ConcreteRegionRef}\ \dfrac{\tau_1 \sim \tau_2 \Rightarrow \tau}{\&\{\ \overline{\ell_1}\ \}\ \mu\ \tau_1 \sim \&\{\ \overline{\ell_2}\ \}\ \mu\ \tau_2 \Rightarrow \&\{\ \overline{\ell_1},\ \overline{\ell_2}\ \}\ \mu\ \tau}$$

$$\textsc{U-RegionVarRef}\ \dfrac{\tau_1 \sim \tau_2 \Rightarrow \tau}{\&\rho_1\ \mu\ \tau_1 \sim \&\{\ \overline{\ell_2}\ \}\ \mu\ \tau_2 \Rightarrow \&\rho_1\ \mu\ \tau}$$

$$\textsc{U-Array}\ \dfrac{\tau_1 \sim \tau_2 \Rightarrow \tau}{[\tau_1;\ n] \sim [\tau_2;\ n] \Rightarrow [\tau;\ n]} \qquad \textsc{U-Slice}\ \dfrac{\tau_1 \sim \tau_2 \Rightarrow \tau}{[\tau_1] \sim [\tau_2] \Rightarrow [\tau]}$$

$$\textsc{U-Tuple}\ \dfrac{\overline{\tau_1} \sim \overline{\tau_2} \Rightarrow \overline{\tau}}{(\overline{\tau_1}) \sim (\overline{\tau_2}) \Rightarrow (\overline{\tau})} \qquad \textsc{U-Struct}\ \dfrac{\overline{\tau_1} \sim \overline{\tau_2} \Rightarrow \overline{\tau}}{S :: <\overline{\tau_1}> \sim S :: <\overline{\tau_2}> \Rightarrow S :: <\overline{\tau}>}$$

2.5 Additional Judgments

$\boxed{\Gamma \vdash \mathcal{L}}$

$$\textsc{V-EmptyLoanContext}\ \dfrac{}{\Gamma \vdash \bullet} \qquad \textsc{V-Loan}\ \dfrac{\Gamma \vdash \mathcal{L}}{\Gamma,\ \pi\ :_{f_\pi}\ \tau \vdash \mathcal{L},\ \ell \mapsto_{f_\ell} \pi}$$

$\boxed{\Sigma; \Gamma; \mathcal{L} \vdash \sigma}$

$$\textsc{V-EmptyStore}\ \dfrac{}{\Sigma; \bullet; \bullet \vdash \bullet} \qquad \textsc{V-StoreEntry}\ \dfrac{\begin{array}{c}\texttt{compute-value}(\sigma,\ \pi) = v \\ \Sigma; \bullet; \Gamma,\ \pi\ :_f\ \tau; \mathcal{L} \vdash \bullet : v \Rightarrow \tau\diamond \qquad \Sigma; \Gamma; \mathcal{L} \vdash \sigma\end{array}}{\Sigma; \Gamma,\ \pi\ :_f\ \tau; \mathcal{L} \vdash \sigma,\ \pi \mapsto_f v}$$

2.6 Dynamics

$\boxed{(\sigma;\ \mathcal{L};\ \boxed{e}) \to (\sigma';\ \mathcal{L}';\ \boxed{e'})}$

$$\textsc{E-Move}\ \dfrac{\texttt{compute-value}(\sigma,\ \pi) = v}{(\sigma,\ \pi \mapsto_1 v;\ \mathcal{L};\ \boxed{\pi}) \to (\sigma;\ \mathcal{L};\ \boxed{v})}$$

$$\textsc{E-Copy}\ \dfrac{f \neq 0 \qquad \texttt{compute-value}(\sigma,\ \pi) = v}{(\sigma,\ \pi \mapsto_f v;\ \mathcal{L};\ \boxed{\pi}) \to (\sigma,\ \pi \mapsto_f v;\ \mathcal{L};\ \boxed{v})}$$

$$\textsc{E-BorrowImm}\ \dfrac{f \neq 0 \qquad \ell \notin \mathcal{L}}{(\sigma,\ \pi \mapsto_f v;\ \mathcal{L};\ \boxed{\&\ell\ \texttt{imm}\ \pi}) \to (\sigma,\ \pi \mapsto_{f/_2} v;\ \mathcal{L},\ \ell \mapsto_{f/_2} \pi;\ \boxed{\texttt{ptr}\ \ell})}$$

$$\textsc{E-BorrowMut}\ \dfrac{\ell \notin \mathcal{L}}{(\sigma,\ \pi \mapsto_1 v;\ \mathcal{L};\ \boxed{\&\ell\ \texttt{mut}\ \pi}) \to (\sigma,\ \pi \mapsto_0 v;\ \mathcal{L},\ \ell \mapsto_1 \pi;\ \boxed{\texttt{ptr}\ \ell})}$$

$$\textsc{E-Drop}\ \dfrac{}{(\sigma,\ \pi \mapsto_1 v;\ \mathcal{L};\ \boxed{\pi}) \to (\sigma;\ \mathcal{L};\ \boxed{()})} \qquad \textsc{E-Seq}\ \dfrac{}{(\sigma;\ \mathcal{L};\ \boxed{v;\ e}) \to (\sigma;\ \mathcal{L};\ \boxed{e})}$$

E-IfTrue

$$\frac{}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{if true}\ \{\ e_1\ \}\ \texttt{else}\ \{\ e_2\ \}}\,) \rightarrow (\sigma;\ \mathcal{L};\ \boxed{e_1}\,)}$$

E-IfFalse

$$\frac{}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{if false}\ \{\ e_1\ \}\ \texttt{else}\ \{\ e_2\ \}}\,) \rightarrow (\sigma;\ \mathcal{L};\ \boxed{e_2}\,)}$$

E-Let

$$\frac{\texttt{compute-places}(x,\ v) = \overline{\pi \mapsto_1 v} \qquad v \neq \texttt{ptr}\ \ell}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{let}\ x\ :\ \tau = v;\ e}\,) \rightarrow (\sigma,\ \overline{\pi \mapsto_1 v};\ \mathcal{L};\ \boxed{e}\,)}$$

E-Assign

$$\frac{\texttt{compute-places}(\pi,\ v) = \overline{\pi \mapsto_1 v}}{(\sigma,\ \pi \mapsto_1 v;\ \mathcal{L};\ \boxed{\pi := v}\,) \rightarrow (\sigma,\ \overline{\pi \mapsto_1 v};\ \mathcal{L};\ \boxed{(\,)}\,)}$$

E-LetRef

$$\frac{\texttt{compute-places}(x,\ \texttt{ptr}\ \ell) = \overline{\pi \mapsto_1 v}}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{let}\ x\ :\ \&\rho\ \mu\ \tau = \texttt{ptr}\ \ell;\ e}\,) \rightarrow (\sigma,\ \overline{\pi \mapsto_1 v};\ \mathcal{L};\ \boxed{e[^{\{\ \ell\ \}}/_{\rho}]}\,)}$$

E-ForArray

$$\frac{\texttt{compute-places}(x,\ v_1) = \overline{\pi \mapsto_1 v}}{\begin{array}{c}(\sigma;\ \mathcal{L};\ \boxed{\texttt{for}\ x\ \texttt{in}\ [v_1\ \ldots\ v_n]\ \{\ e\ \}}\,) \rightarrow \\ (\sigma,\ \overline{\pi \mapsto_1 v};\ \mathcal{L};\ \boxed{e;\ \texttt{for}\ x\ \texttt{in}\ [v_2\ \ldots\ v_n]\ \{\ e\ \}}\,)\end{array}}$$

E-ForSlice

$$\frac{\begin{array}{c} n_1 < n_2 \qquad \sigma(\pi[n_1]) =_f v \qquad f \neq 0 \\ \texttt{compute-value}(\sigma,\ \pi[n_1]) = v \qquad \texttt{compute-places}(x,\ v) = \overline{\pi \mapsto_f v} \qquad n_1 + 1 = n_1' \end{array}}{\begin{array}{c}(\sigma;\ \mathcal{L};\ \boxed{\texttt{for}\ x\ \texttt{in fatptr}\ \ell[n_1..n_2])\ \{\ e\ \}}\,) \rightarrow \\ (\sigma,\ \overline{\pi \mapsto_f v};\ \mathcal{L};\ \boxed{e;\ \texttt{for}\ x\ \texttt{in fatptr}\ \ell[n_1'..n_2])\ \{\ e\ \}}\,)\end{array}}$$

E-ForEmptyArray

$$\frac{}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{for}\ x\ \texttt{in}\ [\,]\ \{\ e\ \}}\,) \rightarrow (\sigma;\ \mathcal{L};\ \boxed{(\,)}\,)}$$

E-ForEmptySlice

$$\frac{}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{for}\ x\ \texttt{in fatptr}\ \ell[n..n])\ \{\ e\ \}}\,) \rightarrow (\sigma;\ \mathcal{L};\ \boxed{(\,)}\,)}$$

E-App

$$\frac{\texttt{compute-places}(x_1,\ v_1) = \overline{\pi_1 \mapsto_1 v_1} \qquad \ldots \qquad \texttt{compute-places}(x_n,\ v_n) = \overline{\pi_n \mapsto_1 v_n}}{\begin{array}{c}(\sigma;\ \mathcal{L};\ \boxed{(\texttt{forall<}\overline{\rho},\ \overline{\alpha}\texttt{>}\ |x_1\ :\ \tau_1\ \ldots\ x_n\ :\ \tau_n|\ \{\ e\ \}\ )\ ::\ \texttt{<}\overline{r},\ \overline{\tau}\texttt{>}(v_1\ \ldots\ v_n)}\,) \rightarrow \\ (\sigma,\ \overline{\pi_1 \mapsto_1 v_1}\ \ldots\ \overline{\pi_n \mapsto_1 v_n};\ \mathcal{L};\ \boxed{e[^{\overline{r}}/_{\overline{\rho}}][^{\overline{\tau}}/_{\overline{\alpha}}]}\,)\end{array}}$$

3 METATHEORY

3.1 Progress

Theorem. If $\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e : \tau \Rightarrow \varepsilon$ and $\Gamma \vdash \mathcal{L}$ and $\Sigma; \Gamma; \mathcal{L} \vdash \sigma$, then either e is a value or $\exists\, \sigma', \mathcal{L}', e'.\ (\sigma;\ \mathcal{L};\ \boxed{e}) \rightarrow (\sigma';\ \mathcal{L}';\ \boxed{e'})$.

Proof. By induction on the derivation $\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e : \tau \Rightarrow \varepsilon$

Case T-Move:

From premise:

$$\textsc{T-Move}\quad \frac{\pi\ :_1\ \tau \in \Gamma}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \pi : \tau \Rightarrow \texttt{drop}\ \pi}$$

We want to step with:

$$\textsc{E-Move}\quad \frac{\texttt{compute-value}(\sigma,\ \pi) = v}{(\sigma,\ \pi \mapsto_1 v;\ \mathcal{L};\ \boxed{\pi}) \rightarrow (\sigma;\ \mathcal{L};\ \boxed{v})}$$

From the premise, we know that $\Sigma; \Gamma; \mathcal{L} \vdash \sigma$. Then, by inspecting V-StoreEntry, we know that if $\pi\ :_1\ \tau \in \Gamma$, then $\pi \mapsto_1 v \in \sigma$. Combining this with the knowledge from T-Move that $\pi\ :_1\ \tau \in \Gamma$, we can indeed conclude $\pi \mapsto_1 v \in \sigma$, and thus know that `compute-value` will succeed, allowing us to step with E-Move.

Case T-Copy:

From premise:

$$\textsc{T-Copy}\quad \frac{\pi\ :_f\ \tau \in \Gamma \qquad \texttt{copyable}\ \tau}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \pi : \tau \Rightarrow \diamond}$$

We want to step with:

$$\textsc{E-Copy}\quad \frac{f \neq 0 \qquad \texttt{compute-value}(\sigma,\ \pi) = v}{(\sigma,\ \pi \mapsto_f v;\ \mathcal{L};\ \boxed{\pi}) \rightarrow (\sigma,\ \pi \mapsto_f v;\ \mathcal{L};\ \boxed{v})}$$

From the premise, we know that $\Sigma; \Gamma; \mathcal{L} \vdash \sigma$. Then, by inspecting V-StoreEntry, we know that if $\pi\ :_f\ \tau \in \Gamma$, then $\pi \mapsto_f v \in \sigma$. Combining this with the knowledge from T-Copy that $\pi\ :_f\ \tau \in \Gamma$, we can indeed conclude $\pi \mapsto_f v \in \sigma$, and thus know that `compute-value` will succeed, allowing us to step with E-Copy.

Case T-Drop:

From premise:

$$\textsc{T-Drop}\quad \frac{\pi\ :_1\ \tau \in \Gamma}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{drop}(\pi) : \texttt{unit} \Rightarrow \texttt{drop}\ \pi}$$

We want to step with:

$$\textsc{E-Drop}\quad \frac{}{(\sigma,\ \pi \mapsto_1 v;\ \mathcal{L};\ \boxed{\pi}) \rightarrow (\sigma;\ \mathcal{L};\ \boxed{()})}$$

From the premise, we know that $\Sigma; \Gamma; \mathcal{L} \vdash \sigma$. Then, by inspecting V-StoreEntry, we know that if $\pi\ :_1\ \tau \in \Gamma$, then $\pi \mapsto_1 v \in \sigma$. Combining this with the knowledge from T-Drop that $\pi\ :_1\ \tau \in \Gamma$, we can indeed conclude $\pi \mapsto_1 v \in \sigma$, allowing us to step with E-Drop.

Case T-BorrowImm:

From premise:

$$\boxed{\dfrac{\ell \notin \mathcal{L} \qquad \pi :_f \tau \in \Gamma \qquad f \neq 0}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \&\ell\ \mathtt{imm}\ \pi : \&\{\ \ell\ \}\ \mathtt{imm}\ \tau \Rightarrow \mathtt{borrow\ imm}\ \pi\ \mathtt{as}\ \ell}\ \text{T-BorrowImm}}$$

We want to step with:

$$\boxed{\dfrac{f \neq 0 \qquad \ell \notin \mathcal{L}}{(\sigma,\ \pi \mapsto_f v;\ \mathcal{L};\ \boxed{\&\ell\ \mathtt{imm}\ \pi}) \to (\sigma,\ \pi \mapsto_{f/2} v;\ \mathcal{L},\ \ell \mapsto_{f/2} \pi;\ \boxed{\mathtt{ptr}\ \ell})}\ \text{E-BorrowImm}}$$

From the premise, we know that $\Sigma; \Gamma; \mathcal{L} \vdash \sigma$. Then, by inspecting V-StoreEntry, we know that if $\pi :_f \tau \in \Gamma$, then $\pi \mapsto_f v \in \sigma$. Combining this with the knowledge from T-BorrowImm that $\pi :_f \tau \in \Gamma$, we can indeed conclude $\pi \mapsto_f v \in \sigma$, allowing us to step with E-BorrowImm.

Case T-BorrowMut:

From premise:

$$\boxed{\dfrac{\ell \notin \mathcal{L} \qquad \pi :_1 \tau \in \Gamma}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \&\ell\ \mathtt{mut}\ \pi : \&\{\ \ell\ \}\ \mathtt{mut}\ \tau \Rightarrow \mathtt{borrow\ mut}\ \pi\ \mathtt{as}\ \ell}\ \text{T-BorrowMut}}$$

We want to step with:

$$\boxed{\dfrac{\ell \notin \mathcal{L}}{(\sigma,\ \pi \mapsto_1 v;\ \mathcal{L};\ \boxed{\&\ell\ \mathtt{mut}\ \pi}) \to (\sigma,\ \pi \mapsto_0 v;\ \mathcal{L},\ \ell \mapsto_1 \pi;\ \boxed{\mathtt{ptr}\ \ell})}\ \text{E-BorrowMut}}$$

From the premise, we know that $\Sigma; \Gamma; \mathcal{L} \vdash \sigma$. Then, by inspecting V-StoreEntry, we know that if $\pi :_1 \tau \in \Gamma$, then $\pi \mapsto_1 v \in \sigma$. Combining this with the knowledge from T-BorrowMut that $\pi :_1 \tau \in \Gamma$, we can indeed conclude $\pi \mapsto_1 v \in \sigma$, allowing us to step with E-BorrowMut.

Case T-Seq:

From premise:

$$\boxed{\dfrac{\begin{array}{c}\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : \tau_1 \Rightarrow \varepsilon_1 \\ \Sigma; \Delta; \varepsilon_1(\Gamma; \mathcal{L}) \vdash e_2 : \tau_2 \Rightarrow \varepsilon_2\end{array}}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1;\ e_2 : \tau_2 \Rightarrow \varepsilon_1, \varepsilon_2}\ \text{T-Seq}}$$

We want to step with:

$$\boxed{\dfrac{}{(\sigma;\ \mathcal{L};\ \boxed{v;\ e}) \to (\sigma;\ \mathcal{L};\ \boxed{e})}\ \text{E-Seq}}$$

If we have an expression of the form $e_1;\ e_2$, then either e_1 is a value or it can take a step (by the inductive hypothesis). If e_1 is a value, then we know that we can step using E-Seq.

Case T-Branch:

From premise:

$$\boxed{\textsc{T-Branch}\quad \frac{\begin{array}{c}\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : \texttt{bool} \Rightarrow \varepsilon_1 \\ \Sigma; \Delta; \varepsilon_1(\Gamma; \mathcal{L}) \vdash e_2 : \tau_2 \Rightarrow \varepsilon_2 \\ \Sigma; \Delta; \varepsilon_1(\Gamma; \mathcal{L}) \vdash e_3 : \tau_3 \Rightarrow \varepsilon_3 \\ \tau_2 \sim \tau_3 \Rightarrow \tau\end{array}}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{if}\ e_1\ \{\ e_2\ \}\ \texttt{else}\ \{\ e_3\ \}\ : \tau \Rightarrow \varepsilon_1, \varepsilon_2, \varepsilon_3}}$$

We want to step with:

$$\boxed{\begin{array}{c}\textsc{E-IfTrue}\quad \frac{}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{if true}\ \{\ e_1\ \}\ \texttt{else}\ \{\ e_2\ \}}) \to (\sigma;\ \mathcal{L};\ \boxed{e_1})} \\ \\ \textsc{E-IfFalse}\quad \frac{}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{if false}\ \{\ e_1\ \}\ \texttt{else}\ \{\ e_2\ \}}) \to (\sigma;\ \mathcal{L};\ \boxed{e_2})}\end{array}}$$

If we have an expression of the form `if` e_1 { e_2 } `else` { e_3 } , then either e_1 is a value or it can take a step (by the inductive hypothesis). If e_1 is a value, then we know that, given its type (`bool`) from the premise, by Canonical Forms, it must be either `true` or `false`. If it is `true`, then we can step using E-IfTrue. If it is `false`, then we can step using E-IfFalse.

Case T-Let:

From premise:

$$\boxed{\textsc{T-Let}\quad \frac{\begin{array}{c}\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : \tau_1 \Rightarrow \varepsilon_1 \qquad \tau = \tau_1 \\ \tau \neq \&r\ \mu\ \tau' \qquad \tau_1 \neq \&r\ \mu\ \tau' \\ \texttt{compute-places}(x,\ \tau) = \overline{\pi :_1 \tau} \\ \Sigma; \Delta; \varepsilon_1(\Gamma,\ \overline{\pi :_1 \tau}; \mathcal{L}) \vdash e_2 : \tau_2 \Rightarrow \varepsilon_2\end{array}}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{let}\ x\ :\ \tau = e_1;\ e_2 : \tau_2 \Rightarrow \varepsilon_1, \varepsilon_2}}$$

We want to step with:

$$\boxed{\textsc{E-Let}\quad \frac{\texttt{compute-places}(x,\ v) = \overline{\pi \mapsto_1 v} \qquad v \neq \texttt{ptr}\ \ell}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{let}\ x\ :\ \tau = v;\ e}) \to (\sigma,\ \overline{\pi \mapsto_1 v};\ \mathcal{L};\ \boxed{e})}}$$

If we have an expression of the form `let` $x\ :\ \tau = e_1;\ e_2$, then either e_1 is a value or it can take a step (by the inductive hypothesis). If e_1 is a value, then we know that we can step using E-Let since `compute-places` must always succeed.

Case T-Assign:

From premise:

$$\textsc{T-Assign}\quad \frac{\pi \;:_1\; \tau \in \Gamma \qquad \Sigma;\Delta;\Gamma;\mathcal{L} \vdash e : \tau \Rightarrow \varepsilon}{\Sigma;\Delta;\Gamma;\mathcal{L} \vdash \pi := e : \texttt{unit} \Rightarrow \varepsilon}$$

We want to step with:

$$\textsc{E-Assign}\quad \frac{\texttt{compute-places}(\pi,\ v) = \overline{\pi \mapsto_1 v}}{(\sigma,\ \pi \mapsto_1 v;\ \mathcal{L};\ \boxed{\pi := v}) \longrightarrow (\sigma,\ \overline{\pi \mapsto_1 v};\ \mathcal{L};\ \boxed{()})}$$

If we have an expression of the form $\pi := e$, then either e is a value or it can take a step (by the inductive hypothesis). If e is a value, then we know that we can step using E-Assign since `compute-places` must always succeed.

Case T-LetRef:

From premise:

$$\textsc{T-LetRef}\quad \frac{\begin{array}{c}\Sigma;\Delta;\Gamma;\mathcal{L} \vdash e_1 : \tau_1 \Rightarrow \varepsilon_1 \qquad \tau = \&\rho\ \mu\ \tau' \qquad \tau_1 = \&\{\ \overline{\ell}\ \}\ \mu\ \tau' \\ \texttt{compute-places}(x,\ \tau) = \overline{\pi :_1 \tau} \\ \Sigma;\Delta;\varepsilon_1(\Gamma,\ \overline{\pi :_1 \tau};\mathcal{L}) \vdash e_2[^{\{\ \overline{\ell}\ \}}/_{\rho}] : \tau_2 \Rightarrow \varepsilon_2\end{array}}{\Sigma;\Delta;\Gamma;\mathcal{L} \vdash \texttt{let}\ x\ :\ \tau = e_1;\ e_2 : \tau_2 \Rightarrow \varepsilon_1, \varepsilon_2}$$

We want to step with:

$$\textsc{E-LetRef}\quad \frac{\texttt{compute-places}(x,\ \texttt{ptr}\ \ell) = \overline{\pi \mapsto_1 v}}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{let}\ x\ :\ \&\rho\ \mu\ \tau = \texttt{ptr}\ \ell;\ e}) \longrightarrow (\sigma,\ \overline{\pi \mapsto_1 v};\ \mathcal{L};\ \boxed{e[^{\{\ \ell\ \}}/_{\rho}]})}$$

If we have an expression of the form $\texttt{let}\ x\ :\ \tau = e_1;\ e_2$, then either e_1 is a value or it can take a step (by the inductive hypothesis). If e_1 is a value, then we know that we can step using E-LetRef since `compute-places` must always succeed.

Case T-ForArray:

From premise:

$$\textsc{T-ForArray}\quad \frac{\Sigma;\Delta;\Gamma;\mathcal{L} \vdash e_1 : [\tau;\ n] \Rightarrow \varepsilon_1 \qquad \varepsilon_2 = \diamond \qquad \Sigma;\Delta;\varepsilon_1(\Gamma,\ x\ :_1\ \tau;\mathcal{L}) \vdash e_2 : \tau_2 \Rightarrow \varepsilon_2}{\Sigma;\Delta;\Gamma;\mathcal{L} \vdash \texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \}\ : \texttt{unit} \Rightarrow \varepsilon_1}$$

We want to step with:

$$\textsc{E-ForArray}\quad \frac{\texttt{compute-places}(x,\ v_1) = \overline{\pi \mapsto_1 v}}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{for}\ x\ \texttt{in}\ [v_1\ \ldots\ v_n]\ \{\ e\ \}}) \rightarrow (\sigma,\ \overline{\pi \mapsto_1 v};\ \mathcal{L};\ \boxed{e;\ \texttt{for}\ x\ \texttt{in}\ [v_2\ \ldots\ v_n]\ \{\ e\ \}})}$$

$$\textsc{E-ForEmptyArray}\quad \frac{}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{for}\ x\ \texttt{in}\ []\ \{\ e\ \}}) \rightarrow (\sigma;\ \mathcal{L};\ \boxed{()})}$$

If we have an expression of the form $\texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \}$, then either e_1 is a value or it can take a step (by the inductive hypothesis). If e_1 is a value, then we know that, given its type ($[\tau;\ n]$) from the premise, by Canonical Forms, it must be of the form $[v_1\ \ldots\ v_n]$. Thus, if $n \neq 0$ then, we can step with E-ForArray (since compute-places must always succeed), and if $n = 0$, we can step with E-ForEmptyArray.

Case T-ForSlice:

From premise:

$$\textsc{T-ForSlice}\quad \frac{\Sigma;\Delta;\Gamma;\mathcal{L} \vdash e_1 : \&r\ \mu\ [\tau] \Rightarrow \varepsilon_1 \qquad \varepsilon_2 = \diamond \qquad \Sigma;\Delta;\varepsilon_1(\Gamma,\ x\ :_\mu\ \tau;\mathcal{L}) \vdash e_2 : \tau_2 \Rightarrow \varepsilon_2}{\Sigma;\Delta;\Gamma;\mathcal{L} \vdash \texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \}\ : \texttt{unit} \Rightarrow \varepsilon_1}$$

We want to step with:

$$\textsc{E-ForSlice}\quad \frac{n_1 < n_2 \qquad \sigma(\pi[n_1]) =_f v \qquad f \neq 0 \qquad \texttt{compute-value}(\sigma,\ \pi[n_1]) = v \qquad \texttt{compute-places}(x,\ v) = \overline{\pi \mapsto_f v} \qquad n_1 + 1 = n_1'}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{for}\ x\ \texttt{in}\ \texttt{fatptr}\ \ell[n_1..n_2])\ \{\ e\ \}}) \rightarrow (\sigma,\ \overline{\pi \mapsto_f v};\ \mathcal{L};\ \boxed{e;\ \texttt{for}\ x\ \texttt{in}\ \texttt{fatptr}\ \ell[n_1'..n_2])\ \{\ e\ \}})}$$

$$\textsc{E-ForEmptySlice}\quad \frac{}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{for}\ x\ \texttt{in}\ \texttt{fatptr}\ \ell[n..n])\ \{\ e\ \}}) \rightarrow (\sigma;\ \mathcal{L};\ \boxed{()})}$$

If we have an expression of the form $\texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \}$, then either e_1 is a value or it can take a step (by the inductive hypothesis). If e_1 is a value, then we know that, given its type ($[\tau]$) from the premise, by Canonical Forms, it must be of the form $\texttt{fatptr}\ \pi[n_1..n_2]$). Thus, if $n_1 < n_2$ then, we can step with E-ForSlice (since compute-places and addition must always succeed), and if $n_1 = n_2$, we can step with E-ForEmptySlice. FIXME: there's something that needs to be done (abort!) about out-of-bounds slicing, but it also probably should've happened when we created the fatptr value (which we don't actually *have* a way to do right now). – Aaron

Case T-App:

From premise:

$$\textsc{T-App}\quad \frac{\begin{array}{c}\Sigma;\Delta;\Gamma;\mathcal{L} \vdash e_f : \texttt{fn<}\overline{\rho},\ \overline{\alpha}\texttt{>}(\tau_1\ \ldots\ \tau_n) \xrightarrow{\varepsilon_f} \tau_f \Rightarrow \varepsilon \\ \Sigma;\Delta;\Gamma;\mathcal{L} \vdash e_1 : \tau_1[\overline{r}/\overline{\rho}][\overline{\tau}/\overline{\alpha}] \Rightarrow \varepsilon_1 \\ \ldots \\ \Sigma;\Delta;(\varepsilon_1,\ \ldots,\varepsilon_{n-1})(\Gamma;\mathcal{L}) \vdash e_n : \tau_n[\overline{r}/\overline{\rho}][\overline{\tau}/\overline{\alpha}] \Rightarrow \varepsilon_n\end{array}}{\Sigma;\Delta;\Gamma;\mathcal{L} \vdash e_f :: \texttt{<}\overline{r},\ \overline{\tau}\texttt{>}(e_1\ \ldots\ e_n) : \tau_f[\overline{r}/\overline{\rho}][\overline{\tau}/\overline{\alpha}] \Rightarrow \varepsilon, \varepsilon_1,\ \ldots,\varepsilon_n, \varepsilon_f}$$

We want to step with:

$$\textsc{E-App}\quad \frac{\texttt{compute-places}(x_1,\ v_1) = \overline{\pi_1 \mapsto_1 v_1} \qquad \ldots \qquad \texttt{compute-places}(x_n,\ v_n) = \overline{\pi_n \mapsto_1 v_n}}{\begin{array}{c}(\sigma;\ \mathcal{L};\ \boxed{(\texttt{forall<}\overline{\rho},\ \overline{\alpha}\texttt{>}\ |x_1\ :\ \tau_1\ \ldots\ x_n\ :\ \tau_n|\ \{\ e\ \})::\texttt{<}\overline{r},\ \overline{\tau}\texttt{>}(v_1\ \ldots\ v_n)}) \rightarrow \\ (\sigma,\ \overline{\pi_1 \mapsto_1 v_1}\ \ldots\ \overline{\pi_n \mapsto_1 v_n};\ \mathcal{L};\ \boxed{e[\overline{r}/\overline{\rho}][\overline{\tau}/\overline{\alpha}]})\end{array}}$$

If we have an expression of the form $e_f :: \texttt{<}\overline{r},\ \overline{\tau}\texttt{>}(e_1\ \ldots\ e_n)$, then either e_f is a value or it can take a step (by the inductive hypothesis). Similarly, either every e_i is a value or it can take a step (by the inductive hypothesis). If e_f is a value, then we know that, given its type $(\texttt{fn<}\overline{\rho},\ \overline{\alpha}\texttt{>}(\tau_1\ \ldots\ \tau_n)\ \xrightarrow{\varepsilon_f}\ \tau_r)$ from the premise, by Canonical Forms, it must be of the form $\texttt{forall<}\overline{\rho},\ \overline{\alpha}\texttt{>}\ |x_1\ :\ \tau_1\ \ldots\ x_n\ :\ \tau_n|\ \{\ e\ \}$. Then, if every e_i is a value, we can step with E-App since we know that `compute-places` must always succeed.

Case T-Tuple:

From premise:

$$\textsc{T-Tuple}\quad \frac{\begin{array}{c}\Sigma;\Delta;\Gamma;\mathcal{L} \vdash e_1 : \tau_1 \Rightarrow \varepsilon_1 \qquad \ldots \\ \Sigma;\Delta;(\varepsilon_1,\ \ldots,\varepsilon_{n-1})(\Gamma;\mathcal{L}) \vdash e_n : \tau_n \Rightarrow \varepsilon_n\end{array}}{\Sigma;\Delta;\Gamma;\mathcal{L} \vdash (e_1\ \ldots\ e_n) : (\tau_1\ \ldots\ \tau_n) \Rightarrow \varepsilon_1,\ \ldots,\varepsilon_n}$$

If we have an expression of the form $(e_1\ \ldots\ e_n)$, then either each e_i is a value or it can take a step (by the inductive hypothesis). If every e_i is a value, then we know (from the grammar for values) that $(e_1\ \ldots\ e_n)$ is also a value.

Case T-Array:

From premise:

$$\textsc{T-Array}\quad \frac{\begin{array}{c}\Sigma;\Delta;\Gamma;\mathcal{L} \vdash e_1 : \tau \Rightarrow \varepsilon_1 \qquad \ldots \\ \Sigma;\Delta;(\varepsilon_1,\ \ldots,\varepsilon_{n-1})(\Gamma;\mathcal{L}) \vdash e_n : \tau_n \Rightarrow \varepsilon_n\end{array}}{\Sigma;\Delta;\Gamma;\mathcal{L} \vdash [e_1\ \ldots\ e_n] : [\tau;\ n] \Rightarrow \varepsilon_1,\ \ldots,\varepsilon_n}$$

If we have an expression of the form $[e_1\ \ldots\ e_n]$, then either each e_i is a value or it can take a step (by the inductive hypothesis). If every e_i is a value, then we know (from the grammar for values) that $[e_1 \ \ldots \ e_n]$ is also a value.

Finally, for T-Closure, T-Unit, T-u32, T-True, and T-False, it is trivial to check the grammar value and determine that they are all values.

3.2 Preservation

Theorem. If $\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e : \tau \Rightarrow \varepsilon$ and $\Gamma \vdash \mathcal{L}$ and $\Sigma; \Gamma; \mathcal{L} \vdash \sigma$ and $(\sigma;\ \mathcal{L};\ \boxed{e}) \rightarrow (\sigma';\ \mathcal{L}_1;\ \boxed{e'})$ then $\exists\, \varepsilon_1, \varepsilon_2.\ \varepsilon_1, \varepsilon_2 \leq \varepsilon \wedge \exists\, \Gamma' \supseteq \Gamma,\ \mathcal{L}' \supseteq \mathcal{L}.\ \varepsilon_1(\Gamma'; \mathcal{L}') = \Gamma_1; \mathcal{L}_1 \wedge \Sigma; \Delta; \Gamma_1; \mathcal{L}_1 \vdash e' : \tau \Rightarrow \varepsilon_2 \wedge \Gamma_1 \vdash \mathcal{L}_1 \ \wedge\ \Sigma; \Gamma_1; \mathcal{L}_1 \vdash \sigma'$.

Proof. By induction on the reduction $(\sigma;\ \mathcal{L};\ \boxed{e}) \rightarrow (\sigma';\ \mathcal{L}_1;\ \boxed{e'})$.

Case E-Move:

From premise:

E-Move
$$\frac{\texttt{compute-value}(\sigma,\ \pi) = v}{(\sigma,\ \pi \mapsto_1 v;\ \mathcal{L};\ \boxed{\pi}) \rightarrow (\sigma;\ \mathcal{L};\ \boxed{v})}$$

From premise and knowledge of the form of e, we know that the last rule in the typing derivation must be one of the following:

T-Move
$$\frac{\pi \ :_1 \ \tau \in \Gamma}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \pi : \tau \Rightarrow \texttt{drop}\ \pi}$$

Let $\varepsilon_1 = \texttt{drop}\ \pi$ and $\varepsilon_2 = \diamond$.

Pick contexts Γ' to be $\boxed{\Gamma}$ and $\mathcal{L}'$ to be $\boxed{\mathcal{L}}$ such that $\varepsilon_1(\Gamma'; \mathcal{L}') = \Gamma_1; \mathcal{L}_1$ and …

$\Gamma_1 \vdash \mathcal{L}_1$	Since applying the effect $\texttt{drop}\ \pi$ removes π from Γ' and none of the loans in $\mathcal{L}_1$ could have possibly referred to π (since it had a capability of 1), then this judgment still holds.
$\Sigma; \Gamma_1; \mathcal{L}_1 \vdash \sigma'$	The store σ' has only changed by removing π from it. Similarly, the variable context Γ_1 has only changed by removing π from it. The other environments remain unchanged. Thus, the only difference between the derivation tree of validity for σ' and σ is that it skips the application of V-StoreEntry to the π entry.
$\Sigma; \Delta; \Gamma_1; \mathcal{L}_1 \vdash e' : \tau \Rightarrow \varepsilon_2$	Since π was bound to v and that π was well-typed, we know that v (our e' in this case) must be of the type τ. Further, we know that, since it is a value, typechecking it must yield the effect $\diamond$ which is our expected value for ε_2.

Case E-Copy:

From premise:

$$\dfrac{f \neq 0 \qquad \texttt{compute-value}(\sigma,\ \pi) = v}{(\sigma,\ \pi \mapsto_f v;\ \mathcal{L};\ \boxed{\pi}) \longrightarrow (\sigma,\ \pi \mapsto_f v;\ \mathcal{L};\ \boxed{v})}\ \textsc{E-Copy}$$

From premise and knowledge of the form of e, we know that the last rule in the typing derivation must be one of the following:

$$\dfrac{\pi\ :_f\ \tau \in \Gamma \qquad \texttt{copyable}\ \tau}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \pi : \tau \Rightarrow \diamond}\ \textsc{T-Copy}$$

Let $\varepsilon_1 = \diamond$ and $\varepsilon_2 = \diamond$.

Pick contexts Γ' to be $\boxed{\Gamma}$ and $\mathcal{L}'$ to be $\boxed{\mathcal{L}}$ such that $\varepsilon_1(\Gamma'; \mathcal{L}') = \Gamma_1; \mathcal{L}_1$ and …

$\Gamma_1 \vdash \mathcal{L}_1$	Since applying the effect $\diamond$ has no effect, then this judgment still holds.
$\Sigma; \Gamma_1; \mathcal{L}_1 \vdash \sigma'$	Since E-Copy has not changed any of the contexts at all, we know that the judgment holds for σ' since we know that it holds for σ and that they are identical.
$\Sigma; \Delta; \Gamma_1; \mathcal{L}_1 \vdash e' : \tau \Rightarrow \varepsilon_2$	Since π was bound to v and that π was well-typed, we know that v (our e' in this case) must be of the type τ. Further, we know that, since it is a value, typechecking it must yield the effect $\diamond$ which is our expected value for ε_2.

Case E-Drop:

From premise:

$$\dfrac{}{(\sigma,\ \pi \mapsto_1 v;\ \mathcal{L};\ \boxed{\pi}) \longrightarrow (\sigma;\ \mathcal{L};\ \boxed{()})}\ \textsc{E-Drop}$$

From premise and knowledge of the form of e, we know that the last rule in the typing derivation must be one of the following:

$$\dfrac{\pi\ :_1\ \tau \in \Gamma}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{drop}(\pi) : \texttt{unit} \Rightarrow \texttt{drop}\ \pi}\ \textsc{T-Drop}$$

Let $\varepsilon_1 = \texttt{drop}\ \pi$ and $\varepsilon_2 = \diamond$.

Pick contexts Γ' to be $\boxed{\Gamma}$ and $\mathcal{L}'$ to be $\boxed{\mathcal{L}}$ such that $\varepsilon_1(\Gamma'; \mathcal{L}') = \Gamma_1; \mathcal{L}_1$ and …

$\Gamma_1 \vdash \mathcal{L}_1$	Since applying the effect $\texttt{drop}\ \pi$ removes π from Γ' and none of the loans in $\mathcal{L}_1$ could have possibly referred to π (since it had a capability of 1), then this judgment still holds.
$\Sigma; \Gamma_1; \mathcal{L}_1 \vdash \sigma'$	The store σ' has only changed by removing π from it. Similarly, the variable context Γ_1 has only changed by removing π from it. The other environments remain unchanged. Thus, the only difference between the derivation tree of validity for σ' and σ is that it skips the application of V-StoreEntry to the π entry.
$\Sigma; \Delta; \Gamma_1; \mathcal{L}_1 \vdash e' : \tau \Rightarrow \varepsilon_2$	E-Drop always yields the expression $\texttt{()}$ which we can always typecheck using T-Unit to get the type $\texttt{unit}$ with the effect $\diamond$ (which is our expected ε_2).

Case E-BorrowImm:

From premise:

$$\textsc{E-BorrowImm}\quad \frac{f \neq 0 \qquad \ell \notin \mathcal{L}}{(\sigma,\ \pi \mapsto_f v;\ \mathcal{L};\ \boxed{\&\ell\ \texttt{imm}\ \pi}) \longrightarrow (\sigma,\ \pi \mapsto_{f/2} v;\ \mathcal{L},\ \ell \mapsto_{f/2} \pi;\ \boxed{\texttt{ptr}\ \ell})}$$

From premise and knowledge of the form of e, we know that the last rule in the typing derivation must be one of the following:

$$\textsc{T-BorrowImm}\quad \frac{\ell \notin \mathcal{L} \qquad \pi :_f \tau \in \Gamma \qquad f \neq 0}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \&\ell\ \texttt{imm}\ \pi : \&\{\ \ell\ \}\ \texttt{imm}\ \tau \Rightarrow \texttt{borrow imm}\ \pi\ \texttt{as}\ \ell}$$

Let $\varepsilon_1 = \texttt{borrow imm}\ \pi\ \texttt{as}\ \ell$ and $\varepsilon_2 = \diamond$.

Pick contexts Γ' to be $\boxed{\Gamma}$ and $\mathcal{L}'$ to be $\boxed{\mathcal{L}}$ such that $\varepsilon_1(\Gamma'; \mathcal{L}') = \Gamma_1; \mathcal{L}_1$ and …

$\Gamma_1 \vdash \mathcal{L}_1$	Applying the effect $\texttt{borrow imm}\ \pi\ \texttt{as}\ \ell$ adds a new loan ℓ to the loan context, but otherwise leaves it unchanged. Further, it updates fraction values in the variable context, but otherwise keeps the collection of places in tact. Thus, we can extend the derivation of $\Gamma \vdash \mathcal{L}$ with V-Loan for the new loan to get $\Gamma_1 \vdash \mathcal{L}_1$.
$\Sigma; \Gamma_1; \mathcal{L}_1 \vdash \sigma'$	While the fractions associated with π have changed, the domain of both σ' and Γ_1 have remained the same. Thus, the same derivation tree for validity of σ still works for σ'.
$\Sigma; \Delta; \Gamma_1; \mathcal{L}_1 \vdash e' : \tau \Rightarrow \varepsilon_2$	We know that E-BorrowImm yields an expression of the form $\texttt{ptr}\ \ell$ corresponding to the newly-created loan ℓ from the borrow. Then, we know that this is valid by T-PointerImm which yields the type $\&\{\ \ell\ \}\ \texttt{imm}\ \tau$ with the effect $\diamond$ (which is our expected ε_2).

Case E-BorrowMut:

From premise:

$$\textsc{E-BorrowMut}\quad \frac{\ell \notin \mathcal{L}}{(\sigma,\ \pi \mapsto_1 v;\ \mathcal{L};\ \boxed{\&\ell\ \texttt{mut}\ \pi}) \rightarrow (\sigma,\ \pi \mapsto_0 v;\ \mathcal{L},\ \ell \mapsto_1 \pi;\ \boxed{\texttt{ptr}\ \ell})}$$

From premise and knowledge of the form of e, we know that the last rule in the typing derivation must be one of the following:

$$\textsc{T-BorrowMut}\quad \frac{\ell \notin \mathcal{L} \qquad \pi\ :_1\ \tau \in \Gamma}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \&\ell\ \texttt{mut}\ \pi : \&\{\ \ell\ \}\ \texttt{mut}\ \tau \Rightarrow \texttt{borrow mut}\ \pi\ \texttt{as}\ \ell}$$

Let ε_1 = $\texttt{borrow mut}\ \pi\ \texttt{as}\ \ell$ and $\varepsilon_2 = \diamond$.

Pick contexts Γ' to be $\boxed{\Gamma}$ and $\mathcal{L}'$ to be $\boxed{\mathcal{L}}$ such that $\varepsilon_1(\Gamma'; \mathcal{L}') = \Gamma_1; \mathcal{L}_1$ and ...

$\Gamma_1 \vdash \mathcal{L}_1$	Applying the effect $\texttt{borrow mut}\ \pi\ \texttt{as}\ \ell$ adds a new loan ℓ to the loan context, but otherwise leaves it unchanged. Further, it updates fraction values in the variable context, but otherwise keeps the collection of places in tact. Thus, we can extend the derivation of $\Gamma \vdash \mathcal{L}$ with V-Loan for the new loan to get $\Gamma_1 \vdash \mathcal{L}_1$.
$\Sigma; \Gamma_1; \mathcal{L}_1 \vdash \sigma'$	While the fractions associated with π have changed, the domain of both σ' and Γ_1 have remained the same. Thus, the same derivation tree for validity of σ still works for σ'.
$\Sigma; \Delta; \Gamma_1; \mathcal{L}_1 \vdash e' : \tau \Rightarrow \varepsilon_2$	We know that E-BorrowMut yields an expression of the form $\texttt{ptr}\ \ell$ corresponding to the newly-created loan ℓ from the borrow. Then, we know that this is valid by T-PointerMut which yields the type $\&\{\ \ell\ \}\ \texttt{mut}\ \tau$ with the effect $\diamond$ (which is our expected ε_2).

Case E-Seq:

From premise:

$$\textsc{E-Seq}\quad \frac{}{(\sigma;\ \mathcal{L};\ \boxed{v;\ e}) \rightarrow (\sigma;\ \mathcal{L};\ \boxed{e})}$$

From premise and knowledge of the form of e, we know that the last rule in the typing derivation must be one of the following:

$$\textsc{T-Seq}\quad \frac{\begin{array}{c}\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : \tau_1 \Rightarrow \varepsilon_1 \\ \Sigma; \Delta; \varepsilon_1(\Gamma; \mathcal{L}) \vdash e_2 : \tau_2 \Rightarrow \varepsilon_2\end{array}}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1;\ e_2 : \tau_2 \Rightarrow \varepsilon_1, \varepsilon_2}$$

Let $\varepsilon_1 = \diamond$ and $\varepsilon_2 = \varepsilon_2$.

Pick contexts Γ' to be $\boxed{\Gamma}$ and $\mathcal{L}'$ to be $\boxed{\mathcal{L}}$ such that $\varepsilon_1(\Gamma'; \mathcal{L}') = \Gamma_1; \mathcal{L}_1$ and ...

$\Gamma_1 \vdash \mathcal{L}_1$	Since applying the effect $\diamond$ has no effect and we selected the same contexts as we originally had, then this judgment still holds.
$\Sigma; \Gamma_1; \mathcal{L}_1 \vdash \sigma'$	Since E-Seq has not changed any of the contexts at all, we know that the judgment holds for σ' since we know that it holds for σ and that they are identical.
$\Sigma; \Delta; \Gamma_1; \mathcal{L}_1 \vdash e' : \tau \Rightarrow \varepsilon_2$	Since none of our contexts have changed and the resulting expression was already typed in T-Seq, we can use exactly that derivation to derive the type τ_2 and effect ε_2.

Case E-IfTrue:

From premise:

$$\frac{\text{E-IfTrue}}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{if true}\ \{\ e_1\ \}\ \texttt{else}\ \{\ e_2\ \}}) \rightarrow (\sigma;\ \mathcal{L};\ \boxed{e_1})}$$

From premise and knowledge of the form of e, we know that the last rule in the typing derivation must be one of the following:

$$\text{T-Branch}\ \frac{\begin{array}{c}\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : \texttt{bool} \Rightarrow \varepsilon_1 \\ \Sigma; \Delta; \varepsilon_1(\Gamma; \mathcal{L}) \vdash e_2 : \tau_2 \Rightarrow \varepsilon_2 \\ \Sigma; \Delta; \varepsilon_1(\Gamma; \mathcal{L}) \vdash e_3 : \tau_3 \Rightarrow \varepsilon_3 \\ \tau_2 \sim \tau_3 \Rightarrow \tau\end{array}}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{if}\ e_1\ \{\ e_2\ \}\ \texttt{else}\ \{\ e_3\ \}\ : \tau \Rightarrow \varepsilon_1, \varepsilon_2, \varepsilon_3}$$

Let $\varepsilon_1 = \diamond$ and $\varepsilon_2 = \varepsilon_2$.

Pick contexts Γ' to be $\boxed{\Gamma}$ and $\mathcal{L}'$ to be $\boxed{\mathcal{L}}$ such that $\varepsilon_1(\Gamma'; \mathcal{L}') = \Gamma_1; \mathcal{L}_1$ and ...

$\Gamma_1 \vdash \mathcal{L}_1$	Since applying the empty effect $\diamond$ has no effect and we selected the same contexts as we originally had, then this judgment still holds.
$\Sigma; \Gamma_1; \mathcal{L}_1 \vdash \sigma'$	Since E-IfTrue has not changed any of the contexts at all, we know that the judgment holds for σ' since we know that it holds for σ and that they are identical.
$\Sigma; \Delta; \Gamma_1; \mathcal{L}_1 \vdash e' : \tau \Rightarrow \varepsilon_2$	We know by the inductive hypothesis that our original expression `if true` $\{\ e_1\ \}$ `else` $\{\ e_2\ \}$ typechecks via T-Branch. Moreover, we know that the predicate `true` typechecks with the effect $\diamond$ via T-True. We can use this information to then conclude (using the premise of T-Branch) that e_1 is well-typed with the effect ε_2.

Case E-IfFalse:

From premise:

$$\textsc{E-IfFalse}\quad \frac{}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{if false}\ \{\ e_1\ \}\ \texttt{else}\ \{\ e_2\ \}\ })\longrightarrow(\sigma;\ \mathcal{L};\ \boxed{e_2})}$$

From premise and knowledge of the form of e, we know that the last rule in the typing derivation must be one of the following:

$$\textsc{T-Branch}\quad \frac{\begin{array}{c}\Sigma;\Delta;\Gamma;\mathcal{L}\vdash e_1 : \texttt{bool} \Rightarrow \varepsilon_1\\ \Sigma;\Delta;\varepsilon_1(\Gamma;\mathcal{L})\vdash e_2 : \tau_2 \Rightarrow \varepsilon_2\\ \Sigma;\Delta;\varepsilon_1(\Gamma;\mathcal{L})\vdash e_3 : \tau_3 \Rightarrow \varepsilon_3\\ \tau_2 \sim \tau_3 \Rightarrow \tau\end{array}}{\Sigma;\Delta;\Gamma;\mathcal{L}\vdash \texttt{if}\ e_1\ \{\ e_2\ \}\ \texttt{else}\ \{\ e_3\ \}\ : \tau \Rightarrow \varepsilon_1,\varepsilon_2,\varepsilon_3}$$

Let $\varepsilon_1 = \diamond$ and $\varepsilon_2 = \varepsilon_3$.

Pick contexts Γ' to be $\boxed{\Gamma}$ and $\mathcal{L}'$ to be $\boxed{\mathcal{L}}$ such that $\varepsilon_1(\Gamma';\mathcal{L}') = \Gamma_1;\mathcal{L}_1$ and …

$\Gamma_1 \vdash \mathcal{L}_1$	Since applying the empty effect $\diamond$ has no effect and we selected the same contexts as we originally had, then this judgment still holds.
$\Sigma;\Gamma_1;\mathcal{L}_1 \vdash \sigma'$	Since E-IfFalse has not changed any of the contexts at all, we know that the judgment holds for σ' since we know that it holds for σ and that they are identical.
$\Sigma;\Delta;\Gamma_1;\mathcal{L}_1 \vdash e' : \tau \Rightarrow \varepsilon_2$	We know by the inductive hypothesis that our original expression `if false` $\{\ e_1\ \}$ `else` $\{\ e_2\ \}$ typechecks via T-Branch. Moreover, we know that the predicate `false` typechecks with the effect $\diamond$ via T-False. We can use this information to then conclude (using the premise of T-Branch) that e_2 is well-typed with the effect ε_3.

Case E-Let:

From premise:

$$\textsc{E-Let}\quad \frac{\texttt{compute-places}(x,\ v) = \overline{\pi \mapsto_1 v} \qquad v \neq \texttt{ptr}\ \ell}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{let}\ x\ :\ \tau = v;\ e})\longrightarrow(\sigma,\ \overline{\pi \mapsto_1 v};\ \mathcal{L};\ \boxed{e})}$$

From premise and knowledge of the form of e, we know that the last rule in the typing derivation must be one of the following:

$$\textsc{T-Let}\quad \frac{\begin{array}{c}\Sigma;\Delta;\Gamma;\mathcal{L}\vdash e_1 : \tau_1 \Rightarrow \varepsilon_1 \qquad \tau = \tau_1\\ \tau \neq \&r\ \mu\ \tau' \qquad \tau_1 \neq \&r\ \mu\ \tau'\\ \texttt{compute-places}(x,\ \tau) = \overline{\pi :_1 \tau}\\ \Sigma;\Delta;\varepsilon_1(\Gamma,\ \overline{\pi :_1 \tau};\mathcal{L})\vdash e_2 : \tau_2 \Rightarrow \varepsilon_2\end{array}}{\Sigma;\Delta;\Gamma;\mathcal{L}\vdash \texttt{let}\ x\ :\ \tau = e_1;\ e_2 : \tau_2 \Rightarrow \varepsilon_1,\varepsilon_2}$$

Let $\varepsilon_1 = \diamond$ and $\varepsilon_2 = \varepsilon_2$.

Pick contexts Γ' to be $\boxed{\Gamma,\ \overline{\pi\ :_1\ \tau}}$ and $\mathcal{L}'$ to be $\boxed{\mathcal{L}}$ such that $\varepsilon_1(\Gamma'; \mathcal{L}') = \Gamma_1; \mathcal{L}_1$ and …

$\Gamma_1 \vdash \mathcal{L}_1$	Since applying the empty effect $\diamond$ has no effect and we selected the same loan context while *extending* Γ with additional places, we know that $\mathcal{L}$ will remain valid by the same judgment. There's a bit of trickiness here with shadowing. Technically, we should only allow shadowing if the identifier being shadowed has a capability of 1. – Aaron
$\Sigma; \Gamma_1; \mathcal{L}_1 \vdash \sigma'$	We intentionally chose Γ_1 in a way that mirrors the way we changed σ in E-Let. Thus, if the relationship between `compute-places`$(x,\ v)$ and `compute-places`$(x,\ \tau)$ is correct, then the new store σ' should still be valid.
$\Sigma; \Delta; \Gamma_1; \mathcal{L}_1 \vdash e' : \tau \Rightarrow \varepsilon_2$	Since our expression is of the form `let` $x\ :\ \tau = v;\ e$, we know that it was typechecked using either T-Let or T-LetRef. In both cases, we know from the hypothesis that e_2 has to be well-typed.

Case E-Assign:

From premise:

$$\textsc{E-Assign}\quad \frac{\texttt{compute-places}(\pi,\ v) = \overline{\pi \mapsto_1 v}}{(\sigma,\ \pi \mapsto_1 v;\ \mathcal{L};\ \boxed{\pi := v}) \rightarrow (\sigma,\ \overline{\pi \mapsto_1 v};\ \mathcal{L};\ \boxed{()})}$$

From premise and knowledge of the form of e, we know that the last rule in the typing derivation must be one of the following:

$$\textsc{T-Assign}\quad \frac{\pi\ :_1\ \tau \in \Gamma \qquad \Sigma; \Delta; \Gamma; \mathcal{L} \vdash e : \tau \Rightarrow \varepsilon}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \pi := e : \texttt{unit} \Rightarrow \varepsilon}$$

Let $\varepsilon_1 = \diamond$ and $\varepsilon_2 = \diamond$.

Pick contexts Γ' to be $\boxed{\Gamma}$ and $\mathcal{L}'$ to be $\boxed{\mathcal{L}}$ such that $\varepsilon_1(\Gamma'; \mathcal{L}') = \Gamma_1; \mathcal{L}_1$ and …

$\Gamma_1 \vdash \mathcal{L}_1$	Since applying the empty effect $\diamond$ has no effect and we selected the same contexts as we originally had, then this judgment still holds.
$\Sigma; \Gamma_1; \mathcal{L}_1 \vdash \sigma'$	In E-Assign, we changed the value of some π entries in σ, but importantly we know from T-Assign that they must have the same types as the original values. Thus, the newly-updated store remains valid.
$\Sigma; \Delta; \Gamma_1; \mathcal{L}_1 \vdash e' : \tau \Rightarrow \varepsilon_2$	The result of E-Assign is `()` which is always well-typed by the rule T-Unit.

Case E-LetRef:

From premise:

$$\textsc{E-LetRef}\quad \frac{\texttt{compute-places}(x,\ \texttt{ptr}\ \ell) = \overline{\pi \mapsto_1 v}}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{let}\ x\ :\ \&\rho\ \mu\ \tau = \texttt{ptr}\ \ell;\ e}) \longrightarrow (\sigma,\ \overline{\pi \mapsto_1 v};\ \mathcal{L};\ \boxed{e[^{\{\ell\}}/_{\rho}]})}$$

From premise and knowledge of the form of e, we know that the last rule in the typing derivation must be one of the following:

$$\textsc{T-LetRef}\quad \frac{\begin{array}{c}\Sigma;\Delta;\Gamma;\mathcal{L} \vdash e_1 : \tau_1 \Rightarrow \varepsilon_1 \qquad \tau = \&\rho\ \mu\ \tau' \qquad \tau_1 = \&\{\ \overline{\ell}\ \}\ \mu\ \tau' \\ \texttt{compute-places}(x,\ \tau) = \overline{\pi :_1 \tau} \\ \Sigma;\Delta;\varepsilon_1(\Gamma,\ \overline{\pi :_1 \tau};\mathcal{L}) \vdash e_2[^{\{\overline{\ell}\}}/_{\rho}] : \tau_2 \Rightarrow \varepsilon_2\end{array}}{\Sigma;\Delta;\Gamma;\mathcal{L} \vdash \texttt{let}\ x\ :\ \tau = e_1;\ e_2 : \tau_2 \Rightarrow \varepsilon_1, \varepsilon_2}$$

Let $\varepsilon_1 = \diamond$ and $\varepsilon_2 = \varepsilon_2$.

Pick contexts Γ' to be $\boxed{\Gamma,\ \overline{\pi\ :_1\ \tau}}$ and $\mathcal{L}'$ to be $\boxed{\mathcal{L}}$ such that $\varepsilon_1(\Gamma';\mathcal{L}') = \Gamma_1;\mathcal{L}_1$ and …

$\Gamma_1 \vdash \mathcal{L}_1$	Since applying the empty effect $\diamond$ has no effect and we selected the same loan context while *extending* Γ with additional places, we know that $\mathcal{L}$ will remain valid by the same judgment. There's a bit of trickiness here with shadowing. Technically, we should only allow shadowing if the identifier being shadowed has a capability of 1. – Aaron
$\Sigma;\Gamma_1;\mathcal{L}_1 \vdash \sigma'$	We intentionally chose Γ_1 in a way that mirrors the way we changed σ in E-LetRef. Thus, if the relationship between $\texttt{compute-places}(x,\ v)$ and $\texttt{compute-places}(x,\ \tau)$ is correct, then the new store σ' should still be valid.
$\Sigma;\Delta;\Gamma_1;\mathcal{L}_1 \vdash e' : \tau \Rightarrow \varepsilon_2$	Since our expression is of the form $\texttt{let}\ x\ :\ \tau = v;\ e$, we know that it was typechecked using either T-Let or T-LetRef. In both cases, we know from the hypothesis that e_2 has to be well-typed. Substitution Lemma? – Aaron

Case E-ForArray:

From premise:

$$\textsc{E-ForArray}\quad \frac{\texttt{compute-places}(x,\ v_1) = \overline{\pi \mapsto_1 v}}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{for}\ x\ \texttt{in}\ [v_1\ \ldots\ v_n]\ \{\ e\ \}}) \rightarrow (\sigma,\ \overline{\pi \mapsto_1 v};\ \mathcal{L};\ \boxed{e;\ \texttt{for}\ x\ \texttt{in}\ [v_2\ \ldots\ v_n]\ \{\ e\ \}})}$$

From premise and knowledge of the form of e, we know that the last rule in the typing derivation must be one of the following:

$$\textsc{T-ForArray}\quad \frac{\begin{array}{c}\Sigma;\Delta;\Gamma;\mathcal{L} \vdash e_1 : [\tau;\ n] \Rightarrow \varepsilon_1 \qquad \varepsilon_2 = \diamond \\ \Sigma;\Delta;\varepsilon_1(\Gamma,\ x\ :_1\ \tau;\mathcal{L}) \vdash e_2 : \tau_2 \Rightarrow \varepsilon_2\end{array}}{\Sigma;\Delta;\Gamma;\mathcal{L} \vdash \texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \}\ : \texttt{unit} \Rightarrow \varepsilon_1}$$

Let $\varepsilon_1 = \diamond$ and $\varepsilon_2 = \diamond$.

Pick contexts Γ' to be $\boxed{\Gamma,\ \texttt{compute-places}(x,\ \tau)}$ and $\mathcal{L}'$ to be $\boxed{\mathcal{L}}$ such that $\varepsilon_1(\Gamma';\mathcal{L}') = \Gamma_1;\mathcal{L}_1$ and …

$\Gamma_1 \vdash \mathcal{L}_1$	Since applying the empty effect $\diamond$ has no effect and we selected the same loan context while *extending* Γ with additional places, we know that $\mathcal{L}$ will remain valid by the same judgment. There's a bit of trickiness here with shadowing. Technically, we should only allow shadowing if the identifier being shadowed has a capability of 1. – Aaron
$\Sigma;\Gamma_1;\mathcal{L}_1 \vdash \sigma'$	We intentionally chose Γ_1 in a way that mirrors the way we changed σ in E-ForArray. Thus, if the relationship between $\texttt{compute-places}(x,\ v)$ and $\texttt{compute-places}(x,\ \tau)$ is correct, then the new store σ' should still be valid.
$\Sigma;\Delta;\Gamma_1;\mathcal{L}_1 \vdash e' : \tau \Rightarrow \varepsilon_2$	We know from T-ForArray that e is well-typed with an effect that is equivalent to the empty effect $\diamond$. We want to then typecheck the output of E-ForArray using T-Seq. Fortunately, since we know that the effect of e is $\diamond$, then we know that to use T-Seq we'll need to show $\Sigma;\Delta;\Gamma;\mathcal{L} \vdash \texttt{for}\ x\ \texttt{in}\ [v_2\ \ldots\ v_n]\ \{\ e\ \}\ : \texttt{unit} \Rightarrow \varepsilon_2$. This is almost identical to the well-typedness for our original form, except that we've dropped the first value from the array. Since this does not effect the type of the array, the whole expression remains well-typed.

Case E-ForSlice:

From premise:

$$\boxed{\dfrac{\textsc{E-ForSlice}\quad n_1 < n_2 \qquad \sigma(\pi[n_1]) =_f v \qquad f \neq 0 \\ \texttt{compute-value}(\sigma,\ \pi[n_1]) = v \qquad \texttt{compute-places}(x,\ v) = \overline{\pi \mapsto_f v} \qquad n_1 + 1 = n_1'}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{for}\ x\ \texttt{in}\ \texttt{fatptr}\ \ell[n_1..n_2])\ \{\ e\ \}}) \rightarrow \\ (\sigma,\ \overline{\pi \mapsto_f v};\ \mathcal{L};\ \boxed{e;\ \texttt{for}\ x\ \texttt{in}\ \texttt{fatptr}\ \ell[n_1'..n_2])\ \{\ e\ \}})}}$$

From premise and knowledge of the form of e, we know that the last rule in the typing derivation must be one of the following:

$$\boxed{\dfrac{\textsc{T-ForSlice} \\ \Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : \&r\ \mu\ [\tau] \Rightarrow \varepsilon_1 \qquad \varepsilon_2 = \diamond \\ \Sigma; \Delta; \varepsilon_1(\Gamma,\ x\ :_\mu\ \tau; \mathcal{L}) \vdash e_2 : \tau_2 \Rightarrow \varepsilon_2}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \}\ : \texttt{unit} \Rightarrow \varepsilon_1}}$$

Let $\varepsilon_1 = \diamond$ and $\varepsilon_2 = \diamond$.

Pick contexts Γ' to be $\boxed{\Gamma,\ \texttt{compute-places}(x,\ \tau)}$ and $\mathcal{L}'$ to be $\boxed{\mathcal{L}}$ such that $\varepsilon_1(\Gamma'; \mathcal{L}') = \Gamma_1; \mathcal{L}_1$ and ...

$\Gamma_1 \vdash \mathcal{L}_1$	Since applying the empty effect $\diamond$ has no effect and we selected the same loan context while *extending* Γ with additional places, we know that $\mathcal{L}$ will remain valid by the same judgment. There's a bit of trickiness here with shadowing. Technically, we should only allow shadowing if the identifier being shadowed has a capability of 1. – Aaron
$\Sigma; \Gamma_1; \mathcal{L}_1 \vdash \sigma'$	We intentionally chose Γ_1 in a way that mirrors the way we changed σ in E-ForSlice. Thus, if the relationship between $\texttt{compute-places}(x,\ v)$ and $\texttt{compute-places}(x,\ \tau)$ is correct, then the new store σ' should still be valid.
$\Sigma; \Delta; \Gamma_1; \mathcal{L}_1 \vdash e' : \tau \Rightarrow \varepsilon_2$	We know from T-ForSlice that e is well-typed with an effect that is equivalent to the empty effect $\diamond$. We want to then typecheck the output of E-ForSlice using T-Seq. Fortunately, since we know that the effect of e is $\diamond$, then we know that to use T-Seq we'll need to show $\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{for}\ x\ \texttt{in}\ \texttt{fatptr}\ \ell[n_1'..n_2])\ \{\ e\ \}\ : \texttt{unit} \Rightarrow \varepsilon_2$. This is almost identical to the well-typedness for our original form, except that we've dropped the first entry from the slice. Since this does not effect the type of the slice, the whole expression remains well-typed.

Case E-ForEmptyArray:

From premise:

$$\frac{\textsc{E-ForEmptyArray}}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{for}\ x\ \texttt{in}\ []\ \{\ e\ \}})\to(\sigma;\ \mathcal{L};\ \boxed{()})}$$

From premise and knowledge of the form of e, we know that the last rule in the typing derivation must be one of the following:

$$\textsc{T-ForArray}\quad \frac{\Sigma;\Delta;\Gamma;\mathcal{L}\vdash e_1:[\tau;\ n]\Rightarrow\varepsilon_1 \qquad \varepsilon_2=\diamond \qquad \Sigma;\Delta;\varepsilon_1(\Gamma,\ x\ :_1\ \tau;\mathcal{L})\vdash e_2:\tau_2\Rightarrow\varepsilon_2}{\Sigma;\Delta;\Gamma;\mathcal{L}\vdash \texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \}\ :\texttt{unit}\Rightarrow\varepsilon_1}$$

Let $\varepsilon_1 = \diamond$ and $\varepsilon_2 = \diamond$.

Pick contexts Γ' to be $\boxed{\Gamma}$ and $\mathcal{L}'$ to be $\boxed{\mathcal{L}}$ such that $\varepsilon_1(\Gamma';\mathcal{L}')=\Gamma_1;\mathcal{L}_1$ and …

$\Gamma_1\vdash\mathcal{L}_1$	Since applying the empty effect $\diamond$ has no effect and we selected the same contexts as we originally had, then this judgment still holds.
$\Sigma;\Gamma_1;\mathcal{L}_1\vdash\sigma'$	Since E-ForEmptyArray has not changed any of the contexts at all, we know that the judgment holds for σ' since we know that it holds for σ and that they are identical.
$\Sigma;\Delta;\Gamma_1;\mathcal{L}_1\vdash e':\tau\Rightarrow\varepsilon_2$	The result of E-ForEmptyArray is () which is always well-typed by the rule T-Unit.

Case E-ForEmptySlice:

From premise:

$$\frac{\textsc{E-ForEmptySlice}}{(\sigma;\ \mathcal{L};\ \boxed{\texttt{for}\ x\ \texttt{in}\ \texttt{fatptr}\ \ell[n..n])\ \{\ e\ \}})\to(\sigma;\ \mathcal{L};\ \boxed{()})}$$

From premise and knowledge of the form of e, we know that the last rule in the typing derivation must be one of the following:

$$\textsc{T-ForSlice}\quad \frac{\Sigma;\Delta;\Gamma;\mathcal{L}\vdash e_1:\&r\ \mu\ [\tau]\Rightarrow\varepsilon_1 \qquad \varepsilon_2=\diamond \qquad \Sigma;\Delta;\varepsilon_1(\Gamma,\ x\ :_\mu\ \tau;\mathcal{L})\vdash e_2:\tau_2\Rightarrow\varepsilon_2}{\Sigma;\Delta;\Gamma;\mathcal{L}\vdash \texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \}\ :\texttt{unit}\Rightarrow\varepsilon_1}$$

Let $\varepsilon_1 = \diamond$ and $\varepsilon_2 = \diamond$.

Pick contexts Γ' to be $\boxed{\Gamma}$ and $\mathcal{L}'$ to be $\boxed{\mathcal{L}}$ such that $\varepsilon_1(\Gamma';\mathcal{L}')=\Gamma_1;\mathcal{L}_1$ and …

$\Gamma_1 \vdash \mathcal{L}_1$	Since applying the empty effect $\diamond$ has no effect and we selected the same contexts as we originally had, then this judgment still holds.
$\Sigma; \Gamma_1; \mathcal{L}_1 \vdash \sigma'$	Since E-ForEmptySlice has not changed any of the contexts at all, we know that the judgment holds for σ' since we know that it holds for σ and that they are identical.
$\Sigma; \Delta; \Gamma_1; \mathcal{L}_1 \vdash e' : \tau \Rightarrow \varepsilon_2$	The result of E-ForEmptySlice is `()` which is always well-typed by the rule T-Unit.

Case E-App:

From premise:

E-App

$$\frac{\texttt{compute-places}(x_1,\ v_1) = \overline{\pi_1 \mapsto_1 v_1} \qquad \dots \qquad \texttt{compute-places}(x_n,\ v_n) = \overline{\pi_n \mapsto_1 v_n}}{\begin{array}{c}(\sigma;\ \mathcal{L};\ \boxed{(\texttt{forall<}\overline{\rho},\ \overline{\alpha}\texttt{>}\ |x_1\ :\ \tau_1\ \dots\ x_n\ :\ \tau_n|\ \{\ e\ \})\ \texttt{::}\ \texttt{<}\overline{r},\ \overline{\tau}\texttt{>}(v_1\ \dots\ v_n)}) \rightarrow \\ (\sigma,\ \overline{\pi_1 \mapsto_1 v_1}\ \dots\ \overline{\pi_n \mapsto_1 v_n};\ \mathcal{L};\ \boxed{e[^{\overline{r}}/_{\overline{\rho}}][^{\overline{\tau}}/_{\overline{\alpha}}]})\end{array}}$$

From premise and knowledge of the form of e, we know that the last rule in the typing derivation must be one of the following:

T-App

$$\frac{\begin{array}{c}\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_f : \texttt{fn<}\overline{\rho},\ \overline{\alpha}\texttt{>}(\tau_1\ \dots\ \tau_n) \xrightarrow{\varepsilon_f} \tau_f \Rightarrow \varepsilon \\ \Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : \tau_1[^{\overline{r}}/_{\overline{\rho}}][^{\overline{\tau}}/_{\overline{\alpha}}] \Rightarrow \varepsilon_1 \\ \dots \\ \Sigma; \Delta; (\varepsilon_1,\ \dots, \varepsilon_{n-1})(\Gamma; \mathcal{L}) \vdash e_n : \tau_n[^{\overline{r}}/_{\overline{\rho}}][^{\overline{\tau}}/_{\overline{\alpha}}] \Rightarrow \varepsilon_n\end{array}}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_f\ \texttt{::}\ \texttt{<}\overline{r},\ \overline{\tau}\texttt{>}(e_1\ \dots\ e_n) : \tau_f[^{\overline{r}}/_{\overline{\rho}}][^{\overline{\tau}}/_{\overline{\alpha}}] \Rightarrow \varepsilon, \varepsilon_1,\ \dots, \varepsilon_n, \varepsilon_f}$$

Let $\varepsilon_1 = \diamond$ and $\varepsilon_2 = \varepsilon_f$.

Pick contexts Γ' to be $\boxed{\Gamma,\ \texttt{compute-places}(x_1,\ \tau_1)\ \dots\ \texttt{compute-places}(x_n,\ \tau_n)}$ and $\mathcal{L}'$ to be $\boxed{\mathcal{L}}$ such that $\varepsilon_1(\Gamma'; \mathcal{L}') = \Gamma_1; \mathcal{L}_1$ and ...

$\Gamma_1 \vdash \mathcal{L}_1$	Since applying the empty effect $\diamond$ has no effect and we selected the same loan context while *extending* Γ with additional places, we know that $\mathcal{L}$ will remain valid by the same judgment. There's a bit of trickiness here with shadowing. Technically, we should only allow shadowing if the identifier being shadowed has a capability of 1. – Aaron
$\Sigma; \Gamma_1; \mathcal{L}_1 \vdash \sigma'$	We intentionally chose Γ_1 in a way that mirrors the way we changed σ in E-App. Thus, if the relationship between `compute-places`$(x,\ v)$ and `compute-places`$(x,\ \tau)$ is correct, then the new store σ' should still be valid.
$\Sigma; \Delta; \Gamma_1; \mathcal{L}_1 \vdash e' : \tau \Rightarrow \varepsilon_2$	T-Closure. Substitution Lemma? – Aaron

A APPENDIX

Text of appendix . . .