Functional Ownership through Fractional Uniqueness (Appendix)

DANIEL MARSHALL, University of Kent, United Kingdom
DOMINIC ORCHARD, University of Kent, United Kingdom and University of Cambridge, United Kingdom

131

Contents

A COLLECTED RULES

A.1 Typing

$$\frac{}{0 \cdot \Gamma, x : A \vdash x : A}\ \textsc{var} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \multimap B}\ \textsc{abs} \qquad \frac{\Gamma_1 \vdash t_1 : A \multimap B \qquad \Gamma_2 \vdash t_2 : A}{\Gamma_1 + \Gamma_2 \vdash t_1\ t_2 : B}\ \textsc{app}$$

$$\frac{\Gamma_1 \vdash t_1 : A \qquad \Gamma_2 \vdash t_2 : B}{\Gamma_1 + \Gamma_2 \vdash (t_1, t_2) : A \otimes B}\ \otimes_I \qquad \frac{\Gamma_1 \vdash t_1 : A \otimes B \qquad \Gamma_2, x : A, y : B \vdash t_2 : C}{\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\ (x, y) = t_1\ \mathbf{in}\ t_2 : C}\ \otimes_E$$

$$\frac{}{0 \cdot \Gamma \vdash () : \mathsf{unit}}\ 1_I \qquad \frac{\Gamma_1 \vdash t_1 : \mathsf{unit} \qquad \Gamma_2 \vdash t_2 : B}{\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\ () = t_1\ \mathbf{in}\ t_2 : B}\ 1_E$$

$$\frac{\Gamma \vdash t : A \qquad \neg\mathsf{resourceAllocator}(t)}{r \cdot \Gamma \vdash [t] : \Box_r A}\ \textsc{pr} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma, x : [A]_1 \vdash t : B}\ \textsc{der}$$

$$\frac{\Gamma_1 \vdash t_1 : \Box_r A \qquad \Gamma_2, x : [A]_r \vdash t_2 : B}{\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\ [x] = t_1\ \mathbf{in}\ t_2 : B}\ \textsc{elim} \qquad \frac{\Gamma, x : [A]_r, \Gamma' \vdash t : B \qquad r \sqsubseteq s}{\Gamma, x : [A]_s, \Gamma' \vdash t : B}\ \textsc{approx}$$

$$\frac{\Gamma \vdash t : *A}{\Gamma \vdash \mathbf{share}\ t : \Box_r A}\ \textsc{share} \qquad \frac{\begin{array}{c}1 \sqsubseteq r \qquad \mathsf{cloneable}(A) \\ \Gamma_1, \overline{id} \vdash t_1 : \Box_r A \qquad \Gamma_2, x : \exists \overline{id'}.*(A[\overline{id'}/\overline{id}]) \vdash t_2 : B\end{array}}{(\Gamma_1 + \Gamma_2), \overline{id} \vdash \mathbf{clone}\ t_1\ \mathbf{as}\ x\ \mathbf{in}\ t_2 : B}\ \textsc{clone}$$

$$\frac{\Gamma_1 \vdash t_1 : *A \qquad \Gamma_2 \vdash t_2 : \&_1 A \multimap \&_1 B}{\Gamma_1 + \Gamma_2 \vdash \mathbf{withBorrow}\ t_1\ t_2 : *B}\ \textsc{with}\&$$

$$\frac{\Gamma \vdash t : \&_p A}{\Gamma \vdash \mathbf{split}\ t : \&_{\frac{p}{2}} A \otimes \&_{\frac{p}{2}} A}\ \textsc{split} \qquad \frac{\Gamma_1 \vdash t_1 : \&_p A \qquad \Gamma_2 \vdash t_2 : \&_q A \qquad p + q \leq 1}{\Gamma_1 + \Gamma_2 \vdash \mathbf{join}\ t_1\ t_2 : \&_{p+q} A}\ \textsc{join}$$

$$\frac{\Gamma \vdash t : \&_p(A \otimes B)}{\Gamma \vdash \mathbf{push}\ t : (\&_p A) \otimes (\&_p B)}\ \textsc{push} \qquad \frac{\Gamma \vdash t : (\&_p A) \otimes (\&_p B)}{\Gamma \vdash \mathbf{pull}\ t : \&_p(A \otimes B)}\ \textsc{pull}$$

$$\frac{\Gamma \vdash t : A \qquad id \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash \mathbf{pack}\ \langle id', t\rangle : \exists id.A[id/id']}\ \textsc{pack} \qquad \frac{\begin{array}{c}\Gamma_1 \vdash t_1 : \exists id.A \\ \Gamma_2, id, x : A \vdash t_2 : B \qquad id \notin \mathsf{fv}(B)\end{array}}{\Gamma_1 + \Gamma_2 \vdash \mathbf{unpack}\ \langle id, x\rangle = t_1\ \mathbf{in}\ t_2 : B}\ \textsc{unpack}$$

Runtime typing.

$$\frac{\gamma \vdash t : A}{0 \cdot \Gamma, \gamma \vdash *t : \&_p A}\ \textsc{nec} \qquad \frac{}{0 \cdot \Gamma, \mathit{ref} : \mathit{Res}_{id}\ A \vdash \mathit{ref} : \mathit{Res}_{id}\ A}\ \textsc{ref}$$

$$\frac{\Gamma \vdash t : \&_1 A}{\Gamma \vdash \mathbf{unborrow}\ t : *A}\ \textsc{unborrow}$$

$$\frac{}{\emptyset \vdash \mathsf{init} : \mathsf{Array}_{id}\ \mathbb{F}}\ \textsc{arrayInit} \qquad \frac{\emptyset \vdash \mathbf{arr} : \mathsf{Array}_{id}\ \mathbb{F} \qquad \emptyset \vdash v : \mathbb{F}}{\emptyset \vdash \mathbf{arr}[n] = v : \mathsf{Array}_{id}\ \mathbb{F}}\ \textsc{arrayAt}$$

$$\frac{\gamma \vdash v : A}{\gamma \vdash \mathbf{ref}(v) : \mathsf{Ref}_{id}\ A}\ \textsc{refStore} \qquad \frac{\Gamma \vdash t : A \qquad \neg\mathsf{resourceAllocator}(t)}{r \cdot \Gamma \vdash [t]_r : \Box_r A}\ \textsc{pr}$$

We sometimes use an admissible rule to simplify some parts of the proofs:

$$\frac{}{0 \cdot \Gamma, ref : Res_{id}\ A \vdash *ref : *(Res_{id}\ A)}\ {*}\textsc{ref}^*$$

which has derivation:

$$\frac{\dfrac{}{ref : Res_{id}\ A \vdash ref : Res_{id}\ A}\ \textsc{ref}}{0 \cdot \Gamma, ref : Res_{id}\ A \vdash *ref : *(Res_{id}\ A)}\ \textsc{nec}$$

Primitives.

$$\frac{}{0 \cdot \Gamma \vdash \mathbf{newRef} : A \multimap \exists id.*(\mathsf{Ref}_{id}\ A)}\ \textsc{newRef}$$

$$\frac{p \equiv 1 \ \vee\ p \equiv *}{0 \cdot \Gamma \vdash \mathbf{swapRef} : \&_p(\mathsf{Ref}_{id}\ A) \multimap A \multimap A \otimes \&_p(\mathsf{Ref}_{id}\ A)}\ \textsc{swapRef}$$

$$\frac{}{0 \cdot \Gamma \vdash \mathbf{freezeRef} : *(\mathsf{Ref}_{id}\ A) \multimap A}\ \textsc{freezeRef}$$

$$\frac{}{0 \cdot \Gamma \vdash \mathbf{readRef} : \&_p(\mathsf{Ref}_{id}\ (\Box_{r+1} A)) \multimap A \otimes \&_p(\mathsf{Ref}_{id}\ (\Box_r A))}\ \textsc{readRef}$$

$$\frac{}{0 \cdot \Gamma \vdash \mathbf{newArray} : \mathbb{N} \multimap \exists id.*(\mathsf{Array}_{id}\ \mathbb{F})}\ \textsc{newArray}$$

$$\frac{}{0 \cdot \Gamma \vdash \mathbf{readArray} : \&_p(\mathsf{Array}_{id}\ \mathbb{F}) \multimap \mathbb{N} \multimap \mathbb{F} \otimes \&_p(\mathsf{Array}_{id}\ \mathbb{F})}\ \textsc{readArray}$$

$$\frac{p \equiv 1 \ \vee\ p \equiv *}{0 \cdot \Gamma \vdash \mathbf{writeArray} : \&_p(\mathsf{Array}_{id}\ \mathbb{F}) \multimap \mathbb{N} \multimap \mathbb{F} \multimap \&_p(\mathsf{Array}_{id}\ \mathbb{F})}\ \textsc{writeArray}$$

$$\frac{}{0 \cdot \Gamma \vdash \mathbf{deleteArray} : *(\mathsf{Array}_{id}\ \mathbb{F}) \multimap \mathsf{unit}}\ \textsc{delArray}$$

Definition A.1 (Graded contexts). $[\Gamma]$ classifies those contexts which contain only graded variables:

$$\frac{}{[\emptyset]} \qquad \frac{[\Gamma]}{[\Gamma, x : [A]_r]}$$

Definition A.2 (Copyable predicate). Predicate definition:

$$\frac{}{\mathsf{copyable}(\mathsf{unit})} \qquad \frac{}{\mathsf{copyable}(\mathbb{N})} \qquad \frac{}{\mathsf{copyable}(\mathbb{F})} \qquad \frac{\mathsf{copyable}(A) \quad \mathsf{copyable}(B)}{\mathsf{copyable}(A \otimes B)}$$

Definition A.3 (Cloneable predicate). Predicate definition:

$$\frac{}{\mathsf{cloneable}(\mathsf{Array}_{id}\ \mathbb{F})} \qquad \frac{\mathsf{cloneable}(A)\ \vee\ \mathsf{copyable}(A)}{\mathsf{cloneable}(\mathsf{Ref}_{id}\ A)} \qquad \frac{\mathsf{cloneable}(A) \quad \mathsf{cloneable}(B)}{\mathsf{cloneable}(A \otimes B)}$$

Definition A.4 (Resource allocating terms). Predicate definition:

$$\frac{}{\mathsf{resourceAllocator}(\mathbf{newRef})} \qquad \frac{}{\mathsf{resourceAllocator}(\mathbf{newArray})}$$

$$\frac{\mathsf{resourceAllocator}(t_1)}{\mathsf{resourceAllocator}(t_1\ t_2)} \qquad \frac{\mathsf{resourceAllocator}(t_2)}{\mathsf{resourceAllocator}(t_1\ t_2)} \qquad \frac{\mathsf{resourceAllocator}(t_1)}{\mathsf{resourceAllocator}((\lambda x.t_1)\ t_2)}$$

$$\frac{\mathsf{resourceAllocator}(t_1)}{\mathsf{resourceAllocator}(\mathbf{let}\ () = t_1\ \mathbf{in}\ t_2)} \qquad \frac{\mathsf{resourceAllocator}(t_2)}{\mathsf{resourceAllocator}(\mathbf{let}\ () = t_1\ \mathbf{in}\ t_2)}$$

$$\frac{\mathsf{resourceAllocator}(t_1)}{\mathsf{resourceAllocator}(\mathbf{let}\ (x, y) = t_1\ \mathbf{in}\ t_2)} \qquad \frac{\mathsf{resourceAllocator}(t_2)}{\mathsf{resourceAllocator}(\mathbf{let}\ (x, y) = t_1\ \mathbf{in}\ t_2)}$$

$$\frac{\mathsf{resourceAllocator}(t_1)}{\mathsf{resourceAllocator}((t_1, t_2))} \qquad \frac{\mathsf{resourceAllocator}(t_2)}{\mathsf{resourceAllocator}((t_1, t_2))}$$

$$\frac{\mathsf{resourceAllocator}(t_1)}{\mathsf{resourceAllocator}(\mathbf{let}\ [x] = t_1\ \mathbf{in}\ t_2)} \qquad \frac{\mathsf{resourceAllocator}(t_2)}{\mathsf{resourceAllocator}(\mathbf{let}\ [x] = t_1\ \mathbf{in}\ t_2)}$$

$$\frac{\mathsf{resourceAllocator}(t)}{\mathsf{resourceAllocator}([t])} \qquad \frac{\mathsf{resourceAllocator}(t_1)}{\mathsf{resourceAllocator}(\mathbf{share}\ t_1)}$$

$$\frac{\mathsf{resourceAllocator}(t_1)}{\mathsf{resourceAllocator}(\mathbf{clone}\ t_1\ \mathbf{as}\ x\ \mathbf{in}\ t_2)} \qquad \frac{\mathsf{resourceAllocator}(t_2)}{\mathsf{resourceAllocator}(\mathbf{clone}\ t_1\ \mathbf{as}\ x\ \mathbf{in}\ t_2)}$$

$$\frac{\mathsf{resourceAllocator}(t_1)}{\mathsf{resourceAllocator}(\mathbf{withBorrow}\ t_1\ t_2)} \qquad \frac{\mathsf{resourceAllocator}(t_2)}{\mathsf{resourceAllocator}(\mathbf{withBorrow}\ t_1\ t_2)}$$

$$\frac{\mathsf{resourceAllocator}(t_1)}{\mathsf{resourceAllocator}(\mathbf{split}\ t_1)} \qquad \frac{\mathsf{resourceAllocator}(t_1)}{\mathsf{resourceAllocator}(\mathbf{join}\ t_1\ t_2)} \qquad \frac{\mathsf{resourceAllocator}(t_2)}{\mathsf{resourceAllocator}(\mathbf{join}\ t_1\ t_2)}$$

$$\frac{\mathsf{resourceAllocator}(t_1)}{\mathsf{resourceAllocator}(\mathbf{push}\ t_1)} \qquad \frac{\mathsf{resourceAllocator}(t_1)}{\mathsf{resourceAllocator}(\mathbf{pull}\ t_1)} \qquad \frac{\mathsf{resourceAllocator}(t_1)}{\mathsf{resourceAllocator}(\mathbf{pack}\ \langle id', t_1\rangle)}$$

$$\frac{\mathsf{resourceAllocator}(t_1)}{\mathsf{resourceAllocator}(\mathbf{unpack}\ \langle id', x\rangle = t_1\ \mathbf{in}\ t_2)} \qquad \frac{\mathsf{resourceAllocator}(t_2)}{\mathsf{resourceAllocator}(\mathbf{unpack}\ \langle id', x\rangle = t_1\ \mathbf{in}\ t_2)}$$

A.2 Reduction rules for heap semantics

$$\frac{\exists r'.\ s + r' \sqsubseteq r}{H, x \mapsto_r v \vdash x \rightsquigarrow_s H, x \mapsto_r v \vdash v} \rightsquigarrow_{\text{VAR}} \qquad \frac{y \# \{H, v, t\}}{H \vdash (\lambda x.t)\ v \rightsquigarrow_s H, y \mapsto_s v \vdash t[y/x]} \rightsquigarrow_\beta$$

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash t_1\ t_2 \rightsquigarrow_s H' \vdash t_1'\ t_2} \rightsquigarrow_{\text{APPL}} \qquad \frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash v\ t_2 \rightsquigarrow_s H' \vdash v\ t_2'} \rightsquigarrow_{\text{APPR}}$$

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash (t_1, t_2) \rightsquigarrow_s H' \vdash (t_1', t_2)} \rightsquigarrow_{\otimes\text{L}} \qquad \frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash (v, t_2) \rightsquigarrow_s H' \vdash (v, t_2')} \rightsquigarrow_{\otimes\text{R}}$$

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash \textbf{let}\,(x, y) = t_1\ \textbf{in}\ t_2 \rightsquigarrow_s H' \vdash \textbf{let}\,(x, y) = t_1'\ \textbf{in}\ t_2} \rightsquigarrow_{\text{LET}\otimes}$$

$$\frac{x' \# \{H, v_1, v_2, t\} \qquad y' \# \{H, v_1, v_2, t\}}{H \vdash \textbf{let}\,(x, y) = (v_1, v_2)\ \textbf{in}\ t \rightsquigarrow_s H, x' \mapsto_s v_1, y' \mapsto_s v_2 \vdash t[y'/y][x'/x]} \rightsquigarrow_{\otimes\beta}$$

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash \textbf{let}\ () = t_1\ \textbf{in}\ t_2 \rightsquigarrow_s H' \vdash \textbf{let}\ () = t_1'\ \textbf{in}\ t_2} \rightsquigarrow_{\text{LETUNIT}} \qquad \frac{}{H \vdash \textbf{let}\ () = ()\ \textbf{in}\ t \rightsquigarrow_s H \vdash t} \rightsquigarrow_{\beta\text{UNIT}}$$

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash \textbf{let}\,[x] = t_1\ \textbf{in}\ t_2 \rightsquigarrow_s H' \vdash \textbf{let}\,[x] = t_1'\ \textbf{in}\ t_2} \rightsquigarrow_{\text{LET}\Box}$$

$$\frac{H \vdash t \rightsquigarrow_{s*r} H' \vdash t'}{H \vdash [t]_r \rightsquigarrow_s H' \vdash [t']_r} \rightsquigarrow_\Box$$

$$\frac{y \# \{H, v, t\}}{H \vdash \textbf{let}\,[x] = [v]_r\ \textbf{in}\ t \rightsquigarrow_s H, y \mapsto_{(s*r)} v \vdash t[y/x]} \rightsquigarrow_{\Box\beta}$$

$$\frac{y \# \{H, v, t\}}{H \vdash \textbf{unpack}\ \langle id, x\rangle = \textbf{pack}\ \langle id', v\rangle\ \textbf{in}\ t \rightsquigarrow_s H, y \mapsto_r v \vdash t[y/x]} \rightsquigarrow_{\exists\beta}$$

$$\frac{H \vdash t \rightsquigarrow_s H \vdash t'}{H \vdash \textbf{pack}\ \langle id, t\rangle \rightsquigarrow_s H \vdash \textbf{pack}\ \langle id, t'\rangle} \rightsquigarrow_{\text{PACK}}$$

$$\frac{H \vdash t_1 \rightsquigarrow_s H \vdash t_1'}{H \vdash \textbf{unpack}\ \langle id, x\rangle = t_1\ \textbf{in}\ t_2 \rightsquigarrow_s H \vdash \textbf{unpack}\ \langle id, x\rangle = t_1'\ \textbf{in}\ t_2} \rightsquigarrow_{\text{UNPACK}}$$

$$\frac{H \vdash t \rightsquigarrow_s H' \vdash t'}{H \vdash \textbf{share}\ t \rightsquigarrow_s H' \vdash \textbf{share}\ t'} \rightsquigarrow_{\text{SHARE}} \qquad \frac{\mathsf{dom}(H) \equiv \mathsf{refs}(v)}{H, H' \vdash \textbf{share}\ (*v) \rightsquigarrow_s ([H]_0), H' \vdash [v]} \rightsquigarrow_{\text{SHARE}\beta}$$

$$\frac{H \vdash t \rightsquigarrow_s H' \vdash t'}{H \vdash *t \rightsquigarrow_s H' \vdash *t'} \rightsquigarrow_* \qquad \frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash \textbf{clone}\ t_1\ \textbf{as}\ x\ \textbf{in}\ t_2 \rightsquigarrow_s H' \vdash \textbf{clone}\ t_1'\ \textbf{as}\ x\ \textbf{in}\ t_2} \rightsquigarrow_{\text{CLONE}}$$

$$\frac{\mathsf{dom}(H') \equiv \mathsf{refs}(v) \qquad (H'', \theta, \overline{id}) = \mathsf{copy}(H') \qquad y \# \{H, v, t\}}{H, H' \vdash \textbf{clone}\ [v]_r\ \textbf{as}\ x\ \textbf{in}\ t \leadsto_s H, H', H'', y \mapsto_s \textbf{pack}\ \langle \overline{id}, *(\theta(v)) \rangle \vdash t[y/x]} \leadsto_{\text{CLONE}\beta}$$

$$\frac{H \vdash t_1 \leadsto_s H' \vdash t_1'}{H \vdash \textbf{withBorrow}\ t_1\ t_2 \leadsto_s H' \vdash \textbf{withBorrow}\ t_1'\ t_2} \leadsto_{\text{WITH\&L}}$$

$$\frac{H \vdash t_2 \leadsto_s H' \vdash t_2'}{H \vdash \textbf{withBorrow}\ (\lambda x.t_1)\ t_2 \leadsto_s H' \vdash \textbf{withBorrow}\ (\lambda x.t_1)\ t_2'} \leadsto_{\text{WITH\&R}}$$

$$\frac{y \# \{H, v, t\}}{H \vdash \textbf{withBorrow}\ (\lambda x.t)\ (*v) \leadsto_s H, y \mapsto_s (*v) \vdash \textbf{unborrow}\ t[y/x]} \leadsto_{\text{WITH\&}}$$

$$\frac{H \vdash t \leadsto_s H' \vdash t'}{H \vdash \textbf{unborrow}\ t \leadsto_s H' \vdash \textbf{unborrow}\ t'} \leadsto_{\text{UNBORROW}}$$

$$\frac{}{H \vdash \textbf{unborrow}\ (*v) \leadsto_s H \vdash *v} \leadsto_{\text{UN\&}}$$

$$\frac{H \vdash t \leadsto_s H' \vdash t'}{H \vdash \textbf{split}\ t \leadsto_s H' \vdash \textbf{split}\ t'} \leadsto_{\text{SPLIT}}$$

$$\frac{ref_1 \# H \qquad ref_2 \# H}{H, ref \mapsto_p id, id \mapsto v \vdash \textbf{split}\ (*ref) \leadsto_s H, ref_1 \mapsto_{\frac{p}{2}} id, ref_2 \mapsto_{\frac{p}{2}} id, id \mapsto v \vdash (*ref_1, *ref_2)} \leadsto_{\text{SPLITREF}}$$

$$\frac{\begin{array}{c} H \vdash \textbf{split}\ (*v) \leadsto_s H' \vdash (*v_1, *v_2) \\ H' \vdash \textbf{split}\ (*w) \leadsto_s H'' \vdash (*w_1, *w_2) \end{array}}{H \vdash \textbf{split}\ (*(v, w)) \leadsto_s H'' \vdash (*(v_1, w_1), *(v_2, w_2))} \leadsto_{\text{SPLIT}\otimes}$$

$$\frac{H \vdash t_1 \leadsto_s H' \vdash t_1'}{H \vdash \textbf{join}\ t_1\ t_2 \leadsto_s H' \vdash \textbf{join}\ t_1'\ t_2} \leadsto_{\text{JOINL}} \qquad \frac{H \vdash t_2 \leadsto_s H' \vdash t_2'}{H \vdash \textbf{join}\ v\ t_2 \leadsto_s H' \vdash \textbf{join}\ v\ t_2'} \leadsto_{\text{JOINR}}$$

$$\frac{ref \# H}{H, ref_1 \mapsto_p id, ref_2 \mapsto_q id, id \mapsto v \vdash \textbf{join}\ (*ref_1)\ (*ref_2) \leadsto_s H, ref \mapsto_{(p+q)} id, id \mapsto v \vdash *ref} \leadsto_{\text{JOINREF}}$$

$$\frac{\begin{array}{c} H \vdash \textbf{join}\ (*v_1)\ (*v_2) \leadsto_s H' \vdash *v \\ H' \vdash \textbf{join}\ (*w_1)\ (*w_2) \leadsto_s H'' \vdash *w \end{array}}{H \vdash \textbf{join}\ (*(v_1, w_1))\ (*(v_2, w_2)) \leadsto_s H'' \vdash *(v, w)} \leadsto_{\text{JOIN}\otimes}$$

$$\frac{H \vdash t \leadsto_s H' \vdash t'}{H \vdash \textbf{push}\ t \leadsto_s H' \vdash \textbf{push}\ t'} \leadsto_{\text{PUSH}} \qquad \frac{H \vdash t \leadsto_s H' \vdash t'}{H \vdash \textbf{pull}\ t \leadsto_s H' \vdash \textbf{pull}\ t'} \leadsto_{\text{PULL}}$$

$$\frac{}{H \vdash \textbf{push}\ (*(v_1, v_2)) \leadsto_s H \vdash (*v_1, *v_2)} \leadsto_{\text{PUSH}*}$$

$$\frac{}{H \vdash \textbf{pull}\ (*v_1, *v_2) \leadsto_s H \vdash *(v_1, v_2)} \leadsto_{\text{PULL}*}$$

Primitive reduction rules.

$$\frac{ref\#H \qquad id\#H}{H \vdash \mathbf{newArray}\; n \rightsquigarrow_s H, ref{\mapsto}_1 id, id \mapsto \mathsf{init} \vdash \mathbf{pack}\;\langle id, *ref\rangle} \rightsquigarrow_{\text{NEWARRAY}}$$

$$\frac{}{H, ref{\mapsto}_p id, id \mapsto \mathbf{arr}[i] = v \vdash \mathbf{readArray}\;(*ref)\; i \rightsquigarrow_s H, ref{\mapsto}_p id, id \mapsto \mathbf{arr}[i] = v \vdash (v, *ref)} \rightsquigarrow_{\text{READARRAY}}$$

$$\frac{}{H, ref{\mapsto}_p id, id \mapsto \mathbf{arr} \vdash \mathbf{writeArray}\;(*ref)\; i\; v \rightsquigarrow_s H, ref{\mapsto}_p id, id \mapsto \mathbf{arr}[i] = v \vdash *ref} \rightsquigarrow_{\text{WRITEARRAY}}$$

$$\frac{}{H, ref{\mapsto}_p id, id \mapsto \mathbf{arr} \vdash \mathbf{deleteArray}\;(*ref) \rightsquigarrow_s H \vdash ()} \rightsquigarrow_{\text{DELETEARRAY}}$$

$$\frac{ref\#H \qquad id\#H}{H \vdash \mathbf{newRef}\; v \rightsquigarrow_s H, ref{\mapsto}_1 id, id \mapsto \mathbf{ref}(v) \vdash \mathbf{pack}\;\langle id, *ref\rangle} \rightsquigarrow_{\text{NEWREF}}$$

$$\frac{}{H, ref{\mapsto}_p id, id \mapsto \mathbf{ref}(v) \vdash \mathbf{swapRef}\;(*ref)\; v' \rightsquigarrow_s H, ref{\mapsto}_p id, id \mapsto \mathbf{ref}(v') \vdash v} \rightsquigarrow_{\text{SWAPREF}}$$

$$\frac{}{H, ref{\mapsto}_p id, id \mapsto \mathbf{ref}(v) \vdash \mathbf{freezeRef}\;(*ref) \rightsquigarrow_s H \vdash v} \rightsquigarrow_{\text{FREEZEREF}}$$

$$\frac{}{H, ref{\mapsto}_p id, id \mapsto \mathbf{ref}([v]_{r+1}) \vdash \mathbf{readRef}\;(*ref) \rightsquigarrow_s H, ref{\mapsto}_p id, id \mapsto \mathbf{ref}([v]_r) \vdash (v, *ref)} \rightsquigarrow_{\text{READREF}}$$

Multi-reduction rules.

$$\frac{}{H \vdash t \Rightarrow_s H \vdash t}\;\text{REFL} \qquad \frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_2 \qquad H' \vdash t_2 \Rightarrow_s H'' \vdash t_3}{H \vdash t_1 \Rightarrow_s H'' \vdash t_3}\;\text{EXT}$$

Heap-context compatibility.

$$\frac{}{\emptyset \bowtie \emptyset}\;\text{EMPTY} \qquad \frac{H \bowtie \emptyset}{H, ref{\mapsto}_p id \bowtie \emptyset}\;\text{GCARR} \qquad \frac{H, id \mapsto v_r \bowtie \Gamma + \gamma \qquad \gamma \vdash v_r : Res_{id}\; A}{H, ref{\mapsto}_p id, id \mapsto v_r \bowtie (\Gamma, ref : Res_{id}\; A)}\;\text{EXTRES}$$

$$\frac{H \bowtie \Gamma + s \cdot \Gamma' \qquad x \notin \mathsf{dom}(H) \qquad \Gamma' \vdash v : A \qquad \exists r'.\, s + r' \equiv r}{(H, x{\mapsto}_r v) \bowtie (\Gamma, x : [A]_s)}\;\text{EXT}$$

$$\frac{H \bowtie \Gamma + \Gamma' \qquad x \notin \mathsf{dom}(H) \qquad \Gamma' \vdash v : A \qquad \exists r'.\, 1 + r' \equiv r}{(H, x{\mapsto}_r v) \bowtie (\Gamma, x : A)}\;\text{EXTLIN}$$

A.3 Equational theory

$$
\begin{aligned}
(\lambda x.t_2)\ t_1 &\equiv t_2[t_1/x] && (\beta)\\
\lambda x.(t\ x) &\equiv t && (\eta)\\
\textbf{let}\ [x] = [t_1]\ \textbf{in}\ t_2 &\equiv t_2[t_1/x] && (\beta_\Box)\\
\textbf{let}\ [x] = t_1\ \textbf{in}\ [x] &\equiv t_1 && (\eta_\Box)\\
[\textbf{let}\ [x] = t_1\ \textbf{in}\ t_2] &\equiv \textbf{let}\ [x] = t_1\ \textbf{in}\ [t_2] && (\Box\text{distrib})\\
\textbf{let}\ () = ()\ \textbf{in}\ t &\equiv t && (\beta_{\mathsf{unit}})\\
\textbf{let}\ () = t\ \textbf{in}\ () &\equiv t && (\eta_{\mathsf{unit}})\\
\textbf{let}\ (x, y) = (t_1, t_2)\ \textbf{in}\ t_3 &\equiv t_3[t_2/y][t_1/x] && (\beta_\otimes)\\
\textbf{let}\ (x, y) = t_1\ \textbf{in}\ (x, y) &\equiv t_1 && (\eta_\otimes)\\
(\textbf{let}\ (x, y) = t_1\ \textbf{in}\ t_2, t_3) &\equiv \textbf{let}\ (x, y) = t_1\ \textbf{in}\ (t_2, t_3) && (\otimes\text{distribL})\\
(t_1, \textbf{let}\ (x, y) = t_2\ \textbf{in}\ t_3) &\equiv \textbf{let}\ (x, y) = t_2\ \textbf{in}\ (t_1, t_3) && (\otimes\text{distribR})\\
\textbf{unpack}\ \langle id, x\rangle = \textbf{pack}\ \langle id', t_1\rangle\ \textbf{in}\ t_2 &\equiv t_2[t_1/x] && (\beta_\exists)\\
\textbf{unpack}\ \langle id, x\rangle = t_1\ \textbf{in}\ \textbf{pack}\ \langle id, x\rangle &\equiv t_1 && (\eta_\exists)\\
\textbf{pack}\ \langle id, (\textbf{unpack}\ \langle id', x\rangle = t_1\ \textbf{in}\ t_2)\rangle &\equiv \textbf{unpack}\ \langle id', x\rangle = t_1\ \textbf{in}\ \textbf{pack}\ \langle id, t_2\rangle && (\exists\text{distrib})
\end{aligned}
$$

$$
\begin{aligned}
\textbf{clone}\ (\textbf{share}\ v)\ \textbf{as}\ x\ \textbf{in}\ t &\equiv t[\textbf{pack}\ \langle \overline{id}, v\rangle/x] && (\beta_*)\\
\textbf{clone}\ t_1\ \textbf{as}\ x\ \textbf{in}\ (\textbf{clone}\ t_2\ \textbf{as}\ y\ \textbf{in}\ t_3) &\equiv \textbf{clone}\ (\textbf{clone}\ t_1\ \textbf{as}\ x\ \textbf{in}\ t_2)\ \textbf{as}\ y\ \textbf{in}\ t_3 && (x \notin \mathsf{FV}(t_3))\\
&&& (*\text{assoc})
\end{aligned}
$$

$$
\begin{aligned}
\textbf{withBorrow}\ (\lambda x.x)\ t &\equiv t && (\&\text{unit})\\
\textbf{withBorrow}\ (\lambda x.f\ (g\ x))\ t &\equiv \textbf{withBorrow}\ f\ (\textbf{withBorrow}\ g\ t) && (\&\text{assoc})\\
(\textbf{let}\ (x, y) = (\textbf{split}\ t)\ \textbf{in}\ (\textbf{join}\ x\ y)) &\equiv t && (\&\text{rejoin})\\
\textbf{split}\ (\textbf{join}\ t_1\ t_2) &\equiv (t_1, t_2) && (\&\text{resplit})
\end{aligned}
$$

A.4 Parallel sum example in Granule

```
-- A more involved example summing two borrowed halves of a unique array in parallel.
-- Run `main` to see the result.

--- Sized vectors
data Vec (n : Nat) t where
  Nil  : Vec 0 t;
  Cons : t → Vec n t → Vec (n+1) t

-- Length of a `Vec` into an indexed `N`, preserving the elements
length' : ∀ {a : Type, n : Nat} . Vec n a → (Int, Vec n a)
length' Nil = (0, Nil);
length' (Cons x xs) = let (n, xs) = length' xs in (n + 1, Cons x xs)

-- Converts a vector of floats to a unique array of floats
toFloatArray : ∀ {n : Nat} . Vec n Float → ∃ {id : Name} . *(FloatArray id)
toFloatArray v =
```

```
  let (n', v) = length' v
  in unpack <id, arr> = newFloatArray n'
  in pack <id, (toFloatArrayAux arr [0] v)> as ∃ {id : Name} . *(FloatArray id)

-- Auxiliary function for `toFloatArray`
toFloatArrayAux : ∀ {n : Nat, id : Name} . *(FloatArray id) → Int [n] → Vec n Float
                                            → *(FloatArray id)
toFloatArrayAux a [n] Nil = a;
toFloatArrayAux a [n] (Cons x xs) =
  toFloatArrayAux (writeFloatArray a n x) [n + 1] xs

-- `sumFromTo a i n` sums the elements of a unique array `a` from index `i` to index `n`
sumFromTo : ∀ {id : Name, p : Fraction} . & p (FloatArray id) →  !Int → !Int
                                          → (Float, & p (FloatArray id))
sumFromTo array [i] [n] =
  if i == n then (0.0, array)
    else
      let (x, a) = readFloatArray array i;
          (y, arr) = sumFromTo a [i+1] [n]
      in  (x + y, arr)

-- Helper function `writeRef` for updating a reference where the old value is dropped
-- (A reference to a "Droppable" value can be written to without violating linearity)
writeRef : ∀ {id : Name, a : Type}     . {Droppable a} ⇒ a → & 1 (Ref id a)
                                         → & 1 (Ref id a)
writeRef x r = let
    (y, r') = swapRef r x;
    () = drop@a y in r'

-- Parallel sum of two halves of a unique array, storing the result in a mutable
-- reference after the parallel computation is done.
parSum : ∀ {id id' : Name} . *(FloatArray id) → *(Ref id’ Float)
                           → *(Ref id’ Float, FloatArray id)
parSum array ref = let
      ([n], array) : (!Int, *(FloatArray id))     = lengthFloatArray array;
      compIn                                      = pull (ref, array)
  in withBorrow (λcompIn →
                 let (ref, array)      = push compIn;
                     (array1, array2)  = split array;

              -- Compute in parallel
                     ((x, array1), (y, array2)) =
                                 par (λ() → sumFromTo array1 [0] [div n 2])
                                     (λ() → sumFromTo array2 [div n 2] [n]);

              -- Update the reference
                     ref'        = writeRef ((x : Float) + y) ref;
                     compOut     = pull (ref', join (array1, array2))

                   in compOut) compIn

-- Main function to sum the elements of a unique array in parallel
```

```
main : Float
main =
  -- Some example data
  unpack <id , arr> = toFloatArray (Cons 10.0 (Cons 20.0 (Cons 30.0 (Cons 40.0 Nil)))) in
  unpack <id', ref> = newRef 0.0 in
  let
      (result, array) = push (parSum arr ref);
      () = deleteFloatArray array
  in freezeRef result
```

B SUBSTITUTION PROOFS

Lemma B.1 (Linear substitution is admissible, extending [Orchard et al. 2019]). *If* $\Gamma_1 \vdash t_1 : A$ *and* $\Gamma_2, x : A \vdash t_2 : B$ *then* $\Gamma_2 + \Gamma_1 \vdash t_2[t_1/x] : B$.

Proof. By induction on the typing derivation of t_2.

- (pr)

$$\frac{\Gamma \vdash t : A \qquad \neg\mathsf{resourceAllocator}(t)}{r \cdot \Gamma \vdash [t] : \Box_r A}\ \textsc{pr}$$

  where $t_2 = [t]$. Trivial since the form of the typing does not match here: no linear variable possible.
- (share)

$$\frac{\Gamma_2, x : A \vdash t : *A}{\Gamma_2, x : A \vdash \textbf{share}\ t : \Box_r A}\ \textsc{share}$$

  where $B = \Box_r A$.
  By induction on the premise then $\Gamma_1 + \Gamma_2 \vdash t[t_1/x] : *A$, from which we build the conclusion:

$$\frac{\Gamma_2 \vdash t[t_1/x] : *A}{\Gamma_2 \vdash \textbf{share}\ (t[t_1/x]) : \Box_r A}\ \textsc{share}$$

- (bind) Two possibilities:

(1) Linear variable x in the left premise:

$$\frac{\Gamma_1', x : A \vdash t_1' : \Box_r A' \qquad \Gamma_2', y : *(\#A') \vdash t_2' : \Box_r B \qquad r \sqsubseteq 1}{\Gamma_1', x : A + \Gamma_2' \vdash \textbf{clone}\ t_1'\ \textbf{as}\ y\ \textbf{in}\ t_2' : \Box_r B}\ \textsc{clone}'$$

By induction on the first premise: $\Gamma_1' + \Gamma_1 \vdash t_1'[t/x] : \Box_r A'$
Then we reconstruct the typing as:

$$\frac{\Gamma_1 + \Gamma_1 \vdash t_1'[t/x] : \Box_r A' \qquad \Gamma_2', y : *(\#A') \vdash t_2' : \Box_r B' \qquad r \sqsubseteq 1}{\Gamma_1' + \Gamma_1 + \Gamma_2' \vdash \textbf{clone}\ t_1'[t/x]\ \textbf{as}\ y\ \textbf{in}\ t_2' : \Box_r B'}\ \textsc{clone}'$$

satisfying the goal (by commutativity of $+$).

(2) Linear variable x in the right premise:

$$\frac{\Gamma_1' \vdash t_1' : \Box_r A' \qquad \Gamma_2', x : A, y : *(\#A') \vdash t_2' : \Box_r B' \qquad r \sqsubseteq 1}{\Gamma_1' + \Gamma_2', x : A \vdash \textbf{clone}\ t_1'\ \textbf{as}\ y\ \textbf{in}\ t_2' : \Box_r B'}\ \textsc{clone}'$$

By induction on the second premise: $(\Gamma_2' + \Gamma_1), y : *A' \vdash t_2'[t/x] : \Box_r B'$

Then we reconstruct the typing as:

$$\frac{\Gamma_1' \vdash t_1' : \Box_r A' \qquad (\Gamma_2' + \Gamma_1), y : *(\#A') \vdash t_2'[t/x] : \Box_r B'}{\Gamma_1' + \Gamma_2' + \Gamma_1 \vdash \textbf{clone } t_1' \textbf{ as } y \textbf{ in } t_2'[t/x] : \Box_r B'}\ \textsc{clone}$$

satisfying the goal.

- (withBorrow) Two possibilities:

(1) Linear variable in the first premise:

$$\frac{\Gamma_1', x : A \vdash t : *A' \qquad \Gamma_2' \vdash f : \&_1 A' \multimap \&_1 B'}{\Gamma_1', x : A + \Gamma_2' \vdash \textbf{withBorrow } f\ t : *B'}\ \textsc{with}\&$$

By induction on the first premise then $\Gamma_1' + \Gamma_1 \vdash t[t_1/x] : *A'$.
From this we construct the goal:

$$\frac{\Gamma_1' + \Gamma_1 \vdash t[t_1/x] : *A' \qquad \Gamma_2' \vdash f : \&_1 A' \multimap \&_1 B'}{\Gamma_1' + \Gamma_1 + \Gamma_2' \vdash \textbf{withBorrow } f\ t[t_1/x] : *B'}\ \textsc{with}\&$$

satisfying the goal by commutativity of $+$.

(2) Linear variable in the second premise:

$$\frac{\Gamma_1' \vdash t : *A' \qquad \Gamma_2', x : A \vdash f : \&_1 A' \multimap \&_1 B'}{\Gamma_1' + \Gamma_2', x : A \vdash \textbf{withBorrow } f\ t : *B'}\ \textsc{with}\&$$

By induction on the second premise then $\Gamma_2' + \Gamma_1 \vdash f[t_1/x] : \&_1 A' \multimap \&_1 B'$.
From this we construct the goal:

$$\frac{\Gamma_1' \vdash t : *A' \qquad \Gamma_2' + \Gamma_1 \vdash f[t_1/x] : \&_1 A' \multimap \&_1 B'}{\Gamma_1' + \Gamma_2' + \Gamma_1 \vdash \textbf{withBorrow } f[t_1/x]\ t : *B'}\ \textsc{with}\&$$

satisfying the goal.

- (split)

$$\frac{\Gamma, x : A \vdash t : \&_{p+q} A}{\Gamma, x : A \vdash \textbf{split } t : \&_p A \otimes \&_q A}\ \textsc{with}\&$$

Then by induction on the premise we have: $\Gamma_1 + \Gamma \vdash t[t_1/x] : B$ from which we construct the goal:

$$\frac{\Gamma_1 + \Gamma \vdash t[t_1/x] : \&_{p+q} A}{\Gamma_1 + \Gamma \vdash \textbf{split } (t[t_1/x]) : \&_p A \otimes \&_q A}\ \textsc{with}\&$$

- (join)

$$\frac{\Gamma_1 \vdash t_1' : \&_p A \qquad \Gamma_2 \vdash t_2' : \&_q A \qquad p + q \leq 1}{\Gamma_1 + \Gamma_2 \vdash \textbf{join } t_1'\ t_2' : \&_{p+q} A}\ \textsc{with}\&$$

Then there are two possibilities depending on the location of the linear typing variable:

(1) (on the left):

$$\frac{\Gamma_1', x : A \vdash t_1' : \&_p A \qquad \Gamma_2' \vdash t_2' : \&_q A \qquad p + q \leq 1}{\Gamma_1', x : A + \Gamma_2' \vdash \textbf{join } t_1'\ t_2' : \&_{p+q} A}\ \textsc{with}\&$$

Then by induction on the premise we have: $\Gamma_1' + \Gamma_1 \vdash t_1'[t_1/x] : \&_p A$ from which we construct the goal:

$$\frac{\Gamma_1', x : A \vdash t_1'[t/x] : \&_p A \qquad \Gamma_2' \vdash t_2' : \&_q A \qquad p + q \leq 1}{\Gamma_1' + \Gamma_1 + \Gamma_2' \vdash \mathbf{join}\ (t_1'[t_1/x])\ t_2' : \&_{p+q} A}\ \textsc{with\&}$$

(2) (on the right):

$$\frac{\Gamma_1' \vdash t_1' : \&_p A \qquad \Gamma_2', x : A \vdash t_2' : \&_q A \qquad p + q \leq 1}{\Gamma_1' + \Gamma_2', x : A \vdash \mathbf{join}\ t_1'\ t_2' : \&_{p+q} A}\ \textsc{with\&}$$

Then by induction on the premise we have: $\Gamma_2' + \Gamma_1 \vdash t_2'[t_1/x] : \&_q A$ from which we construct the goal:

$$\frac{\Gamma_1' \vdash t_1' : \&_p A \qquad \Gamma_2' + \Gamma_1 \vdash t_2'[t_1/x] : \&_q A \qquad p + q \leq 1}{\Gamma_1' + \Gamma_2' + \Gamma_1 \vdash \mathbf{join}\ t_1'\ (t_2'[t_1/x]) : \&_{p+q} A}\ \textsc{with\&}$$

- (push)

$$\frac{\Gamma, x : A \vdash t : \&_p (A \otimes B)}{\Gamma, x : A \vdash \mathbf{push}\, t : (\&_p A) \otimes (\&_p B)}\ \textsc{push}$$

Then by induction on the premise we have: $\Gamma + \Gamma_1 \vdash t[t_1/x] : \&_p (A \otimes B)$ from which we construct the goal:

$$\frac{\Gamma + \Gamma_1 \vdash t[t_1/x] : \&_p (A \otimes B)}{\Gamma + \Gamma_1 \vdash \mathbf{push}\, t[t_1/x] : \&_p A \otimes \&_q A}\ \textsc{push}$$

- (pull)

$$\frac{\Gamma, x : A \vdash t : (\&_p A) \otimes (\&_p B)}{\Gamma, x : A \vdash \mathbf{pull}\, t : \&_p (A \otimes B)}\ \textsc{pull}$$

Then by induction on the premise we have: $\Gamma + \Gamma_1 \vdash t[t_1/x] : (\&_p A) \otimes (\&_p B)$ from which we construct the goal:

$$\frac{\Gamma + \Gamma_1 \vdash t[t_1/x] : (\&_p A) \otimes (\&_p B)}{\Gamma + \Gamma_1 \vdash \mathbf{pull}\, t[t_1/x] : \&_p (A \otimes B)}\ \textsc{pull}$$

- (newRef), (swapRef), (freezeRef), (readRef), (newArray), (readArray), (writeArray), (deleteArray) all trivial as they are atomic with substitution having no effect.

□

Lemma B.2 (Graded substitution is admissible, extending [Orchard et al. 2019]). *If $[\Gamma_1] \vdash t_1 : A$ and $\Gamma_2, x : [A]_r \vdash t_2 : B$ (where $[\Gamma_1]$ represents a context Γ_1 containing only graded assumptions) and $\neg\mathsf{resourceAllocator}(t_1)$ then $\Gamma_2 + r \cdot \Gamma_1 \vdash t_2[t_1/x] : B$.*

Proof. By induction on the typing derivation of t_2.

- (pr)

$$\frac{\Gamma_2', x : [A]_{r_2} \vdash t : A \qquad \neg\mathsf{resourceAllocator}(t)}{r_1 \cdot (\Gamma_2', x : [A]_{r_2}) \vdash [t] : \Box_{r_1} A}\ \textsc{pr}$$

where $r = r_1 * r_2$ and $t_2 = [t]$.
By induction on the premise then we have $\Gamma_2', r_2 \cdot \Gamma_1 \vdash t[t_1/x] : A$.

Then we construct the goal:

$$\frac{\Gamma_2', r_2 \cdot \Gamma_1 \vdash t[t_1/x] : A \qquad \neg\mathsf{resourceAllocator}(t[t_1/x])}{r_1 \cdot (\Gamma_2', r_2 \cdot \Gamma_1) \vdash [t[t_1/x]] : \Box_{r_1} A}\ \textsc{pr}$$

where $\neg\mathsf{resourceAllocator}(t[t_1/x])$ follows from $\neg\mathsf{resourceAllocator}(t)$ and $\neg\mathsf{resourceAllocator}(t_1)$ and which equals $r_1 \cdot \Gamma_2' + r_1 \cdot r_2 \cdot \Gamma_1 \vdash [t[t_1/x]] : \Box_{r_1} A$ satisfying the goal here.

- (share)

$$\frac{\Gamma_2, x : [A]_r \vdash t : *A}{\Gamma_2, x : [A]_r \vdash \mathbf{share}\ t : \Box_s A}\ \textsc{share}$$

where $B = \Box_s A$.
By induction on the premise then $\Gamma_2 + r \cdot \Gamma_1 \vdash t[t_1/x] : *A$, from which we build the conclusion:

$$\frac{\Gamma_2 + r \cdot \Gamma_1 \vdash t[t_1/x] : *A}{\Gamma_2 + r \cdot \Gamma_1 \vdash \mathbf{share}\,(t[t_1/x]) : \Box_s A}\ \textsc{share}$$

- (bind)

$$\frac{\Gamma_1', x : [A]_{r_1} \vdash t_1' : \Box_s A' \qquad \Gamma_2', y : *(\#A'), x : [A]_{r_2} \vdash t_2' : \Box_s B \qquad r \sqsubseteq 1}{\Gamma_1' + \Gamma_2', x : [A]_{r_1+r_2} \vdash \mathbf{clone}\ t_1'\ \mathbf{as}\ y\ \mathbf{in}\ t_2' : \Box_s B}\ \textsc{clone}'$$

with $r = r_1 + r_2$ without loss of generality (since any context not including x can instead have weakening applied to have either $r_1 = 0$ and/or $r_2 = 0$).
By induction on the premises, we have: (1) $\Gamma_1' + r_1 \cdot \Gamma_1 \vdash t_1'[t/x] : \Box_s A'$ (2) $(\Gamma_2' + r_2 \cdot \Gamma_1), y : *A' \vdash t_2'[t/x] : \Box_s B'$
Then we reconstruct the typing as:

$$\frac{\Gamma_1' + r_1 \cdot \Gamma_1 \vdash t_1'[t/x] : \Box_s A' \qquad (\Gamma_2' + r_2 \cdot \Gamma_1), y : *(\#A') \vdash t_2'[t/x] : \Box_s B'}{\Gamma_1' + \Gamma_2' + (r_1 + r_2) \cdot \Gamma_1 \vdash \mathbf{clone}\ t_1'[t/x]\ \mathbf{as}\ y\ \mathbf{in}\ t_2'[t/x] : \Box_s B'}\ \textsc{clone}$$

satisfying the goal.

- (withBorrow)

$$\frac{\Gamma_1', x : [A]_{r_1} \vdash t : *A' \qquad \Gamma_2', x : [A]_{r_2} \vdash f : \&_1 A' \multimap \&_1 B'}{(\Gamma_1' + \Gamma_2'), x : [A]_{r_1+r_2} \vdash \mathbf{withBorrow}\ f\ t : *B'}\ \textsc{with}\&$$

By induction on the premises, we have: (1) $\Gamma_1' + r_1 \cdot \Gamma_1 \vdash t[t_1/x] : *A'$. (2) $\Gamma_2' + r_2 \cdot \Gamma_1 \vdash f[t_1/x] : \&_1 A' \multimap \&_1 B'$.
From this we construct the goal:

$$\frac{\Gamma_1' + r_1 \cdot \Gamma_1 \vdash t[t_1/x] : *A' \qquad \Gamma_2' + r_2 \cdot \Gamma_1 \vdash f[t_1/x] : \&_1 A' \multimap \&_1 B'}{\Gamma_1' + \Gamma_2' + (r_1 + r_2) \cdot \Gamma_1 \vdash \mathbf{withBorrow}\ f[t_1/x]\ t : *B'}\ \textsc{with}\&$$

satisfying the goal.

- (split)

$$\frac{\Gamma, x : [A]_r \vdash t : \&_{p+q} A}{\Gamma, x : [A]_r \vdash \mathbf{split}\ t : \&_p A \otimes \&_q A}\ \textsc{with}\&$$

Then by induction on the premise we have: $\Gamma_1 + r \cdot \Gamma \vdash t[t_1/x] : B$ from which we construct the goal:

$$\frac{\Gamma_1 + r \cdot \Gamma \vdash t[t_1/x] : \&_{p+q} A}{\Gamma_1 + r \cdot \Gamma \vdash \mathbf{split}\ (t[t_1/x]) : \&_p A \otimes \&_q A}\ \textsc{with}\&$$

- (join)

$$\frac{\Gamma_1', x : [A]_{r_1} \vdash t_1' : \&_p A \qquad \Gamma_2', x : [A]_{r_2} \vdash t_2' : \&_q A \qquad p + q \leq 1}{(\Gamma_1' + \Gamma_2'), x : [A]_{(r_1+r_2)} \vdash \textbf{join}\ t_1'\ t_2' : \&_{p+q} A} \textsc{with\&}$$

  Then by induction on the premises we have: (1) $\Gamma_1' + r_1 \cdot \Gamma_1 \vdash t_1'[t_1/x] : \&_p A$ (2) $\Gamma_2' + r_2 \cdot \Gamma_1 \vdash t_2'[t_1/x] : \&_q A$ from which we construct the goal:

$$\frac{\Gamma_1' + r_1 \cdot \Gamma_1 \vdash t_1'[t_1/x] : \&_p A \qquad \Gamma_2' + r_2 \cdot \Gamma_1 \vdash t_2'[t_1/x] : \&_q A \qquad p + q \leq 1}{\Gamma_1' + \Gamma_2' + (r_1 + r_2) \cdot \Gamma_1 \vdash \textbf{join}\ t_1'\ (t_2'[t_1/x]) : \&_{p+q} A} \textsc{with\&}$$

  satisfying the goal.
- (push)

$$\frac{\Gamma, x : [A]_r \vdash t : \&_p(A \otimes B)}{\Gamma, x : [A]_r \vdash \textbf{push}\ t : (\&_p A) \otimes (\&_p B)} \textsc{push}$$

  Then by induction on the premise we have: $\Gamma + r \cdot \Gamma_1 \vdash t[t_1/x] : \&_p(A \otimes B)$ from which we construct the goal:

$$\frac{\Gamma + r \cdot \Gamma_1 \vdash t[t_1/x] : \&_p(A \otimes B)}{\Gamma + r \cdot \Gamma_1 \vdash \textbf{push}\ t[t_1/x] : \&_p A \otimes \&_q A} \textsc{push}$$

- (pull)

$$\frac{\Gamma, x : [A]_r \vdash t : (\&_p A) \otimes (\&_p B)}{\Gamma, x : [A]_r \vdash \textbf{pull}\ t : \&_p(A \otimes B)} \textsc{pull}$$

  Then by induction on the premise we have: $\Gamma + r \cdot \Gamma_1 \vdash t[t_1/x] : (\&_p A) \otimes (\&_p B)$ from which we construct the goal:

$$\frac{\Gamma + r \cdot \Gamma_1 \vdash t[t_1/x] : (\&_p A) \otimes (\&_p B)}{\Gamma + r \cdot \Gamma_1 \vdash \textbf{pull}\ t[t_1/x] : \&_p(A \otimes B)} \textsc{pull}$$

- (newRef), (swapRef), (freezeRef), (readRef), (newArray), (readArray), (writeArray), (deleteArray) all trivial as they are atomic with substitution having no effect.

□

C TYPE SAFETY

C.1 Progress proof

LEMMA C.1. *Value lemma*
Given $\Gamma \vdash v : A$ then, depending on the type, the shape of v can be inferred:

- $A = A' \multimap B$ *then* $v = \lambda x.t$ *or a partially applied primitive term* p.
- $A = \Box_r A'$ *then* $v = [v']$.
- $A = A' \otimes B$ *then* $v = (v_1, v_2)$.
- $A = 1$ *then* $v = ()$.
- $A = *A'$ *then* $v = *v'$.
- $A = \&_p A'$ *then* $v = *v'$.
- $A = \mathbb{N}$ *then* $v = n$.
- $A = \mathbb{F}$ *then* $v = f$.
- $A = \mathsf{Ref}_{id}\ A'$ *then* $v = ref$.
- $A = \mathsf{Array}_{id}\ \mathbb{F}$ *then* $v = a$.
- $A = \exists id.A'$ *then* $v = \textbf{\textit{pack}}\ \langle id', v' \rangle$.

Proof. Recall that the value terms sub-grammar is:

$$v ::= (v_1, v_2) \mid () \mid *v \mid [v] \mid \lambda x.t \mid i \mid \mathit{ref} \mid a \mid p \mid \textbf{pack}\ \langle id', v' \rangle \qquad \text{(value terms sub-grammar)}$$

where p are partially-applied primitives:

$$\begin{aligned} p ::= {} & \textbf{newRef} \mid \textbf{swapRef} \mid \textbf{swapRef}\,(*\mathit{ref}) \\ & \mid \textbf{freezeRef} \mid \textbf{readRef} \\ & \mid \textbf{newArray} \mid \textbf{readArray} \mid \textbf{readArray}\,(*a) \\ & \mid \textbf{writeArray} \mid \textbf{writeArray}\,(*a) \mid \textbf{writeArray}\,(*a)\ n \mid \textbf{deleteArray} \end{aligned}$$

We then proceed by case analysis on the type A to match the structure of the lemma. In each case we must consider what possible values can be assigned the type A and by which rules.

In all cases, there exists additional derivations based on dereliction and approximation, e.g., for the case where $A = A' \multimap B$:

$$\frac{\Gamma, x : A'' \vdash t : A' \multimap B}{\Gamma, x : [A'']_1 \vdash t : A' \multimap B}\ \textsc{der}$$

$$\frac{\Gamma, y : [A'']_r, \Gamma' \vdash t : A' \multimap B \quad r \sqsubseteq s}{\Gamma, y : [A'']_s, \Gamma' \vdash t : A' \multimap B}\ \textsc{approx}$$

In all of these cases we can apply induction on the premise to get the result since the term is preserved between the premise and the conclusion.

We elide handling this separately each time in the cases that follow as the reasoning through dereliction is the same each time.

- $A = A' \multimap B$ then there are two classes of possible typing:
  - Abstract term:

    $$\frac{\Gamma, x : A' \vdash t : B}{\Gamma \vdash \lambda x.t : A' \multimap B}\ \textsc{abs}$$

    thus $v = \lambda x.t$ as in the lemma statement.
  - Primitive term p formed by an application of zero or more values to a primitive operation, of which there are then twelve possibilities:

(1)

$$\frac{}{0 \cdot \Gamma \vdash \textbf{newRef} : A \multimap \exists id.*(\mathsf{Ref}_{id}\ A)}\ \textsc{newRef}$$

thus $v = \textbf{newRef}$

(2)

$$\frac{p \equiv 1 \ \vee\ p \equiv *}{0 \cdot \Gamma \vdash \textbf{swapRef} : \&_p(\mathsf{Ref}_{id}\ A) \multimap A \multimap A \otimes \&_p(\mathsf{Ref}_{id}\ A)}\ \textsc{swapRef}$$

thus $v = \textbf{swapRef}$

(3)

$$\frac{\dfrac{\dfrac{}{0 \cdot \Gamma, \mathit{ref} : \mathit{Res}_{id}\ A \vdash \mathit{ref} : \mathit{Res}_{id}\ A}\ \textsc{ref}}{0 \cdot \Gamma, \mathit{ref} : \mathsf{Ref}_{id}\ A \vdash *\mathit{ref} : \&_1 \mathsf{Ref}_{id}\ A}\ \textsc{nec}}{0 \cdot \Gamma, \mathit{ref} : \mathsf{Ref}_{id}\ A \vdash \textbf{swapRef}\,(*\mathit{ref}) : A \otimes \&_1(\mathsf{Ref}_{id}\ A)}\ \textsc{app}$$

thus $v = \textbf{swapRef}\,(*\mathit{ref})$

(4)

$$\frac{}{0 \cdot \Gamma \vdash \mathbf{freezeRef} : *(\mathsf{Ref}_{id}\ A) \multimap A}\ \textsc{freezeRef}$$

thus $v = \mathbf{freezeRef}$

(5)

$$\frac{}{0 \cdot \Gamma \vdash \mathbf{readRef} : \&_p(\mathsf{Ref}_{id}\ (\Box_{r+1}A)) \multimap A \otimes \&_p(\mathsf{Ref}_{id}\ (\Box_r A))}\ \textsc{readRef}$$

thus $v = \mathbf{readRef}$

(6)

$$\frac{}{0 \cdot \Gamma \vdash \mathbf{newArray} : \mathbb{N} \multimap \exists id.*(\mathsf{Array}_{id}\ \mathbb{F})}\ \textsc{newArray}$$

thus $v = \mathbf{newArray}$

(7)

$$\frac{}{0 \cdot \Gamma \vdash \mathbf{readArray} : \&_p(\mathsf{Array}_{id}\ \mathbb{F}) \multimap \mathbb{N} \multimap \mathbb{F} \otimes \&_p(\mathsf{Array}_{id}\ \mathbb{F})}\ \textsc{readArray}$$

thus $v = \mathbf{readArray}$

(8)

$$\frac{\dfrac{\dfrac{}{0 \cdot \Gamma, \mathit{ref} : \mathit{Res}_{id}\ A \vdash \mathit{ref} : \mathit{Res}_{id}\ A}\ \textsc{ref}}{0 \cdot \Gamma, a : \mathsf{Array}_{id}\ A \vdash *a : \&_1 \mathsf{Array}_{id}\ \mathbb{F}}\ \textsc{nec}}{0 \cdot \Gamma, a : \mathsf{Array}_{id}\ A \vdash \mathbf{readArray}\ (*a) : \mathbb{N} \multimap \mathbb{F} \otimes \&_1(\mathsf{Array}_{id}\ \mathbb{F})}\ \textsc{app}$$

thus $v = \mathbf{readArray}\ (*a)$

(9)

$$\frac{p \equiv 1 \ \vee\ p \equiv *}{0 \cdot \Gamma \vdash \mathbf{writeArray} : \&_p(\mathsf{Array}_{id}\ \mathbb{F}) \multimap \mathbb{N} \multimap \mathbb{F} \multimap \&_p(\mathsf{Array}_{id}\ \mathbb{F})}\ \textsc{writeArray}$$

thus $v = \mathbf{writeArray}$

(10)

$$\frac{\dfrac{\dfrac{}{0 \cdot \Gamma, \mathit{ref} : \mathit{Res}_{id}\ A \vdash \mathit{ref} : \mathit{Res}_{id}\ A}\ \textsc{ref}}{0 \cdot \Gamma, a : \mathsf{Array}_{id}\ A \vdash *a : \&_1 \mathsf{Array}_{id}\ \mathbb{F}}\ \textsc{nec}}{0 \cdot \Gamma, a : \mathsf{Array}_{id}\ A \vdash \mathbf{writeArray}\ (*a) : \mathbb{N} \multimap \mathbb{F} \multimap \mathbb{F} \otimes \&_1(\mathsf{Array}_{id}\ \mathbb{F})}\ \textsc{app}$$

thus $v = \mathbf{writeArray}\ (*a)$

(11)

$$\frac{\dfrac{\dfrac{\dfrac{}{0 \cdot \Gamma, \mathit{ref} : \mathit{Res}_{id}\ A \vdash \mathit{ref} : \mathit{Res}_{id}\ A}\ \textsc{ref}}{0 \cdot \Gamma, a : \mathsf{Array}_{id}\ A \vdash *a : \&_1 \mathsf{Array}_{id}\ \mathbb{F}}\ \textsc{nec}}{0 \cdot \Gamma, a : \mathsf{Array}_{id}\ A \vdash \mathbf{writeArray}\ (*a) : \mathbb{F} \otimes \&_1(\mathsf{Array}_{id}\ \mathbb{F})}\ \textsc{app} \qquad \emptyset \vdash n : \mathbb{N}}{0 \cdot \Gamma, a : \mathsf{Array}_{id}\ A \vdash \mathbf{writeArray}\ (*a)\ n : \mathbb{F} \multimap \mathbb{F} \otimes \&_1(\mathsf{Array}_{id}\ \mathbb{F})}\ \textsc{app}$$

thus $v = \mathbf{writeArray}\ (*a)\ n$

(12)

$$\frac{}{0 \cdot \Gamma \vdash \mathbf{deleteArray} : *(\mathsf{Array}_{id}\ \mathbb{F}) \multimap \mathsf{unit}}\ \textsc{delArray}$$

thus $v = \mathbf{deleteArray}$

- $A = \Box_r A'$ then there is only one possible non-dereliction/non-approximation typing of a value at that type:

$$\frac{\Gamma \vdash v' : A'}{r \cdot \Gamma \vdash [v'] : \Box_r A'}\ \textsc{pr}$$

thus $v = [v']$ as in the lemma statement.

- $A = A' \otimes B$ then there is only one possible non-dereliction/non-approximation typing of a value at that type:

$$\frac{\Gamma_1 \vdash v_1 : A' \quad \Gamma_2 \vdash v_2 : B}{\Gamma_1 + \Gamma_2 \vdash (v_1, v_2) : A' \otimes B}\ \otimes_I$$

thus $v = (v_1, v_2)$ as in the lemma statement.

- $A = 1$ then there is only one possible non-dereliction/non-approximation typing of a value at that type:

$$\frac{}{0 \cdot \Gamma \vdash () : \mathsf{unit}}\ 1_I$$

thus $v = ()$ as in the lemma statement.

- $A = *A'$ then there is only one possible non-dereliction/non-approximation typing of a value at that type:

$$\frac{\emptyset \vdash v' : A'}{0 \cdot \Gamma \vdash *v' : *A'}\ \textsc{nec}$$

thus $v = *t$ as in the lemma statement.

- $A = \&_p A'$ then there is only one possible non-dereliction/non-approximation typing of a value at that type:

$$\frac{\emptyset \vdash v' : A'}{0 \cdot \Gamma \vdash *v' : \&_1 A'}\ \textsc{nec}$$

thus $v = *v'$ as in the lemma statement.

- $A = \mathbb{N}$ then $v = n$ Trivial case on typing of constants which is elided in this paper for brevity (but covered by the core type theory of Granule for example).
- $A = \mathbb{F}$ then $v = f$ Trivial case on typing of constants which is elided in this paper for brevity (but covered by the core type theory of Granule for example).
- $A = \mathsf{Ref}_{id}\ A$ then $v = ref$ then the only possible typing that is a value is given by:

$$\frac{}{0 \cdot \Gamma, ref : Res_{id}\ A \vdash ref : Res_{id}\ A}\ \textsc{ref}$$

- $A = \mathsf{Array}_{id}\ \mathbb{F}$ then $v = a$ then the only possible typing that is a value is given by:

$$\frac{}{0 \cdot \Gamma, ref : Res_{id}\ A \vdash ref : Res_{id}\ A}\ \textsc{ref}$$

(and since the only type of arrays is $\mathbb{F}$ currently).

- $A = \exists id.A'$ then there is only one possible non-dereliction/non-approximation typing of a value at that type:

$$\frac{\Gamma \vdash v' : A \quad id \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash \textbf{pack}\ \langle id', t\rangle : \exists id.A[id/id']}\ \textsc{pack}$$

thus $v = \textbf{pack}\ \langle id', v'\rangle$ as in the lemma statement.

□

Lemma C.2 (Closed value lemma). *Given $\Gamma \vdash v : A$ where A does not comprise a function type, then there exists a runtime only context γ such that $\gamma \vdash v : A$, i.e., v is closed with respect to normal variables.*

Proof. Similar to the value lemma proof structure, and where:

- $A = A' \multimap B'$ is excluded by the lemma statement.
- $A = \square_r A'$ then $v = [v']$ by induction.
- $A = A' \otimes B$ then $v = (v_1, v_2)$ by induction.
- $A = 1$ then $v = ()$ is closed.
- $A = *A'$ then $v = *v'$ by induction.
- $A = \&_p A'$ then $v = *v'$ by induction.
- $A = \mathbb{N}$ then $v = n$ is closed.
- $A = \mathbb{F}$ then $v = f$ is closed.
- $A = \mathsf{Ref}_{id}\ A'$ then $v = \mathit{ref}$ has only a runtime context.
- $A = \mathsf{Array}_{id}\ \mathbb{F}$ then $v = a$ has only a runtime context.
- $A = \exists id.A'$ then $v = \textbf{pack}\ \langle id', v' \rangle$ by induction.

□

Lemma C.3 (Unique value lemma). *Given $\Gamma \vdash *v : *A$ then, depending on the type A, the shape of v can be inferred:*

- *$A = A' \otimes B'$ then $v = (v_1, v_2)$.*
- *$A = \mathsf{Ref}_{id}\ A$ then $v = \mathit{ref}$.*
- *$A = \mathsf{Array}_{id}\ \mathbb{F}$ then $v = a$.*

and there are no other possible typings for $*v$. Furthermore, $\exists \Gamma', \gamma$ such that $0 \cdot \Gamma', \gamma \vdash *v : *A$, i.e., it can be type in a runtime context only.

Proof. There are only three possible typings for $*v$.

- $A = A' \otimes B'$ where there is only one possible non-dereliction/non-approximation typing of a value at the type $*(A' \otimes B')$:

$$\dfrac{\dfrac{\dfrac{}{0 \cdot \Gamma_1, \gamma_1 \vdash *v_1 : *A'}\ \textsc{induction.} \qquad \dfrac{}{0 \cdot \Gamma_2, \gamma_2 \vdash *v_2 : *B'}\ \textsc{induction.}}{0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + \gamma_1 + \gamma_2 \vdash (*v_1, *v_2) : *A' \otimes *B'}\ \otimes_I}{0 \cdot (\Gamma_1 + \Gamma_2) + \gamma_1 + \gamma_2 \vdash *(v_1, v_2) : *(A' \otimes B')}\ \textsc{pull}$$

  thus $v = (v_1, v_2)$ as in the lemma statement and $\Gamma' = \Gamma_1 + \Gamma_2$ and $\gamma = \gamma_1 + \gamma_2$.
- $A = \mathsf{Ref}_{id}\ A$ where there is only one possible non-dereliction/non-approximation typing of a value at the type $*(\mathsf{Ref}_{id}\ A)$:

$$\dfrac{}{0 \cdot \Gamma, \mathit{ref} : \mathit{Res}_{id}\ A \vdash *\mathit{ref} : *(\mathit{Res}_{id}\ A)}\ *\textsc{ref}^*$$

  thus $v = \mathit{ref}$ as in the lemma statement and $\Gamma' = \Gamma$ and $\gamma = \mathit{ref} : \mathsf{Ref}_{id}\ A$.
- $A = \mathsf{Array}_{id}\ \mathbb{F}$ where there is only one possible non-dereliction/non-approximation typing of a value at the type $*(\mathsf{Array}_{id}\ \mathbb{F})$:

$$\dfrac{}{0 \cdot \Gamma, \mathit{ref} : \mathit{Res}_{id}\ A \vdash *\mathit{ref} : *(\mathit{Res}_{id}\ A)}\ *\textsc{ref}^*$$

  thus $v = a$ as in the lemma statement (and since the only type of arrays is $\mathbb{F}$ currently) and $\Gamma' = \Gamma$ and $\gamma = a : \mathsf{Array}_{id}\ A$.

□

Theorem C.4 (Progress). *Given $\Gamma \vdash t : A$, then t is either a value, or for all grades s and contexts Γ_0 then if $H \bowtie \Gamma_0 + s \cdot \Gamma$ there exists a heap H' and term t' such that $H \vdash t \rightsquigarrow_s H' \vdash t'$.*

Proof. By induction on typing.

- (var)

$$\frac{}{0 \cdot \Gamma, x : A \vdash x : A}\ \text{VAR}$$

Here, $H \bowtie s \cdot [\Gamma], x : [A]_s$, which by inversion of heap compatibility implies that $H = H', x \mapsto_r v$ and $\exists r'.\ s + r' \equiv r$. Hence, we can reduce by the following rule:

$$\frac{\exists r'.\ s + r' \sqsubseteq r}{H, x \mapsto_r v \vdash x \rightsquigarrow_s H, x \mapsto_r v \vdash v}\ \rightsquigarrow_{\text{VAR}}$$

where $\exists r'.\ s + r' \equiv r$ implies $\exists r'.\ s + r' \sqsubseteq r$ by reflexivity, satisfying the premise.

- (abs)

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \multimap B}\ \text{ABS}$$

A value.

- (app)

$$\frac{\Gamma_1 \vdash t_1 : A \multimap B \qquad \Gamma_2 \vdash t_2 : A}{\Gamma_1 + \Gamma_2 \vdash t_1\ t_2 : B}\ \text{APP}$$

By induction on the first premise, there are two possibilities.

(1) t_1 is a value and therefore by the value lemma there are a number of choices:
  - $t_1 = \lambda x.t_1'$. Therefore we induct on the second premise providing two possibilities:
    * If $t_2 = v$ for some value v, then we can reduce by the following rule:

$$\frac{y \# \{H, v, t\}}{H \vdash (\lambda x.t)\ v \rightsquigarrow_s H, y \mapsto_s v \vdash t[y/x]}\ \rightsquigarrow_\beta$$

    * If t_2 is not a value, then there exists heap H', term t_1' and context Γ' such that $H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'$. Therefore we can reduce by the following rule:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash v\, t_2 \rightsquigarrow_s H' \vdash v\, t_2'}\ \rightsquigarrow_{\text{APPR}}$$

  - $t_1 = \mathbf{newRef}$

    Therefore we induct on the second argument:
    * t_2 is a value v and thus we can reduce:

$$\frac{\mathit{ref} \# H \qquad \mathit{id} \# H}{H \vdash \mathbf{newRef}\ v \rightsquigarrow_s H, \mathit{ref} \mapsto_1 \mathit{id}, \mathit{id} \mapsto \mathbf{ref}(v) \vdash \mathbf{pack}\ \langle \mathit{id}, *\mathit{ref} \rangle}\ \rightsquigarrow_{\text{NEWREF}}$$

    * t_2 is not a value and thus has a reduction, therefore we can build the compound reduction:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \mathbf{newRef}\ t_2 \rightsquigarrow_s H' \vdash \mathbf{newRef}\ t_2'}\ \rightsquigarrow_{\text{PRIM}}$$

  - $t_1 = \mathbf{swapRef}$
    * t_2 is a value and therefore by the value lemma on $\&_p(\mathrm{Ref}_{id}\ A)$ (Lemma C.1) $t_2 = *\mathit{ref}$ and thus $t_1\ t_2 = \mathbf{swapRef}\ (*\mathit{ref})$ which is also a value.

* t_2 is not a value and thus has a reduction, therefore we can build the compound reduction:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \mathbf{swapRef}\ t_2 \rightsquigarrow_s H' \vdash \mathbf{swapRef}\ t_2'} \rightsquigarrow_{\text{PRIM}}$$

– $t_1 = \mathbf{swapRef}\ (*ref)$
Therefore we induct on the second argument:
  * t_2 is a value v and thus we can reduce:

$$\frac{}{H, ref \mapsto_p id, id \mapsto \mathbf{ref}(v) \vdash \mathbf{swapRef}\ (*ref)\ v' \rightsquigarrow_s H, ref \mapsto_p id, id \mapsto \mathbf{ref}(v') \vdash v} \rightsquigarrow_{\text{SWAPREF}}$$

  * t_2 is not a value and thus has a reduction, therefore we can build the compound reduction:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \mathbf{swapRef}\ (*ref)\ t_2 \rightsquigarrow_s H' \vdash \mathbf{swapRef}\ (*ref)\ t_2'} \rightsquigarrow_{\text{PRIM}}$$

– $t_1 = \mathbf{freezeRef}$
Therefore we induct on the second argument:
  * t_2 is a value which by the value and unique value lemmas has the form $*ref$, and thus we can reduce:

$$\frac{}{H, ref \mapsto_p id, id \mapsto \mathbf{ref}(v) \vdash \mathbf{freezeRef}\ (*ref) \rightsquigarrow_s H \vdash v} \rightsquigarrow_{\text{FREEZEREF}}$$

  * t_2 is not a value and thus has a reduction, therefore we can build the compound reduction:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \mathbf{freezeRef}\ t_2 \rightsquigarrow_s H' \vdash \mathbf{freezeRef}\ t_2'} \rightsquigarrow_{\text{PRIM}}$$

– $t_1 = \mathbf{readRef}$
Therefore we induct on the second argument:
  * t_2 is a value which by the value and unique value lemmas has the form $*ref$, and thus we can reduce:

$$\frac{}{H, ref \mapsto_p id, id \mapsto \mathbf{ref}([v]_{r+1}) \vdash \mathbf{readRef}\ (*ref) \rightsquigarrow_s H, ref \mapsto_p id, id \mapsto \mathbf{ref}([v]_r) \vdash (v, *ref)} \rightsquigarrow_{\text{READREF}}$$

  * t_2 is not a value and thus has a reduction, therefore we can build the compound reduction:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \mathbf{readRef}\ t_2 \rightsquigarrow_s H' \vdash \mathbf{readRef}\ t_2'} \rightsquigarrow_{\text{PRIM}}$$

– $t_1 = \mathbf{newArray}$ therefore $A = \mathbb{N}$
Therefore we induct on the second argument:
  * t_2 is a value and therefore by the value lemma (Lemma C.1) $t_2 = n$ and thus the typing is:

$$\frac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathbf{newArray}\ n : *(\mathsf{Array}_{id}\ \mathbb{F})}\text{TyDerivednewArray}$$

with $H \bowtie (\Gamma_0 + s \cdot \Gamma)$.
Thus there is a reduction as follows:

$$\frac{ref \# H \qquad id \# H}{H \vdash \mathbf{newArray}\ n \rightsquigarrow_s H, ref \mapsto_1 id, id \mapsto \mathsf{init} \vdash \mathbf{pack}\ \langle id, *ref \rangle} \rightsquigarrow_{\text{NEWARRAY}}$$

  * t_2 is not a value and thus has a reduction, therefore we can build the compound reduction:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \textbf{newArray}\ t_2 \rightsquigarrow_s H' \vdash \textbf{newArray}\ t_2'} \rightsquigarrow_{\text{PRIM}}$$

- $t_1 = \textbf{readArray}$ therefore $A = \&_p(\mathsf{Array}_{id}\ \mathbb{F}) \multimap \mathbb{N} \multimap \mathbb{F} \otimes \&_p(\mathsf{Array}_{id}\ \mathbb{F})$
  * t_2 is a value and therefore by the value lemma on $\&_p(\mathsf{Array}_{id}\ \mathbb{F})$ (Lemma C.1) $t_2 = *a$ and thus $t_1\ t_2 = \textbf{readArray}\ (*a)$ which is also a value.
  * t_2 is not a value and thus has a reduction, therefore we can build the compound reduction:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \textbf{readArray}\ t_2 \rightsquigarrow_s H' \vdash \textbf{readArray}\ t_2'} \rightsquigarrow_{\text{PRIM}}$$

- $t_1 = \textbf{readArray}\ (*a)$ therefore $A = \mathbb{N} \multimap \mathbb{F} \otimes \&_p(\mathsf{Array}_{id}\ \mathbb{F})$
  * t_2 is a value and therefore by the value lemma on $\Gamma_2 \vdash t_2 : \mathbb{N}$ (Lemma C.1) implies $t_2 = n$ and thus the typing is refined at runtime as follows:

$$\frac{\dfrac{}{[\Gamma_1], a : \mathsf{Array}_{id}\ \mathbb{F} \vdash (*a) : \&_p(\mathsf{Array}_{id}\ \mathbb{F})}\ \text{NEC} \qquad \Gamma_2 \vdash n : \mathbb{N}}{[\Gamma_1] + \Gamma_2, a : \mathsf{Array}_{id}\ \mathbb{F} \vdash \textbf{readArray}\ (*a)\ n : \mathbb{F} \otimes \&_p(\mathsf{Array}_{id}\ \mathbb{F})}\ \text{TyDerivedreadArray}$$

with $H' \bowtie \Gamma_0 + s \cdot ([\Gamma_1] + \Gamma_2), a : \mathsf{Array}_{id}\ \mathbb{F}$, and by the heap compatibility rule for array references there exists some H such that $H' = H, a \mapsto_p id, id \mapsto \textbf{arr}$.
Then there is a reduction as follows:

$$\frac{}{H, ref \mapsto_p id, id \mapsto \textbf{arr}[i] = v \vdash \textbf{readArray}\ (*ref)\ i \rightsquigarrow_s H, ref \mapsto_p id, id \mapsto \textbf{arr}[i] = v \vdash (v, *ref)} \rightsquigarrow_{\text{READARRAY}}$$

  * t_2 is not a value and thus has a reduction, therefore we can build the compound reduction:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \textbf{readArray}\ (*a)\ t_2 \rightsquigarrow_s H' \vdash \textbf{readArray}\ (*a)\ t_2'} \rightsquigarrow_{\text{PRIM}}$$

- $t_1 = \textbf{writeArray}$ therefore $A = \&_1(\mathsf{Array}_{id}\ \mathbb{F}) \multimap \mathbb{N} \multimap \mathbb{F} \multimap \&_1(\mathsf{Array}_{id}\ \mathbb{F})$
  * t_2 is a value therefore by the value lemma (Lemma C.1) $t_2 = (*a)$ and thus $t_1\ t_2 = \textbf{writeArray}\ (*a)$ which is also a value.
  * t_2 is not a value and thus has a reduction, therefore we can build the compound reduction:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \textbf{writeArray}\ t_2 \rightsquigarrow_s H' \vdash \textbf{writeArray}\ t_2'} \rightsquigarrow_{\text{PRIM}}$$

- $t_1 = \textbf{writeArray}\ (*a)$ therefore $A = \mathbb{N} \multimap \mathbb{F} \multimap \&_1(\mathsf{Array}_{id}\ \mathbb{F})$
  * t_2 is a value therefore by the value lemma (Lemma C.1) $t_2 = n$ and thus $t_1\ t_2 = \textbf{writeArray}\ (*a)\ n$ which is also a value.
  * t_2 is not a value and thus has a reduction, therefore we can build the compound reduction:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \textbf{writeArray}\ (*a)\ t_2 \rightsquigarrow_s H' \vdash \textbf{writeArray}\ (*a)\ t_2'} \rightsquigarrow_{\text{PRIM}}$$

- $t_1 = \textbf{writeArray}\ (*a)\ n$ therefore $A = \mathbb{F} \otimes \&_1(\mathsf{Array}_{id}\ \mathbb{F})$

* t_2 is a value and therefore the value lemma on $\Gamma_2 \vdash t_2 : \mathbb{F}$ (Lemma C.1) implies $t_2 = f$ and thus the typing is refined at runtime as follows:

$$\dfrac{\dfrac{[\Gamma_1], a : \mathsf{Array}_{id}\,\mathbb{F} \vdash a : (\mathsf{Array}_{id}\,\mathbb{F})\ \textsc{ref}}{[\Gamma_1], a : \mathsf{Array}_{id}\,\mathbb{F} \vdash *a : \&_1(\mathsf{Array}_{id}\,\mathbb{F})}\ \textsc{nec} \qquad \Gamma_2 \vdash n : \mathbb{N} \qquad \Gamma_3 \vdash f : \mathbb{F}}{[\Gamma_1] + \Gamma_2 + \Gamma_3, a : \mathsf{Array}_{id}\,\mathbb{F} \vdash \mathbf{writeArray}\ (*a)\ n\, f : \&_1(\mathsf{Array}_{id}\,\mathbb{F})}\ \textsc{TyDerivedwriteArray}$$

with $H' \bowtie (\Gamma_0 + [\Gamma_1] + \Gamma_2 + \Gamma_3, a : \mathsf{Array}_{id}\,\mathbb{F})$, and by the heap compatibility rule for array references there exists some H such that $H' = H, a \mapsto_p id, id \mapsto \mathbf{arr}$.
Then there is a reduction as follows:

$$\dfrac{}{H, \mathit{ref} \mapsto_p id, id \mapsto \mathbf{arr} \vdash \mathbf{writeArray}\ (*\mathit{ref})\ i\ v \rightsquigarrow_s H, \mathit{ref} \mapsto_p id, id \mapsto \mathbf{arr}[i] = v \vdash *\mathit{ref}}\ \rightsquigarrow_{\textsc{writeArray}}$$

* t_2 is not a value and thus has a reduction, therefore we can build the compound reduction:

$$\dfrac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \mathbf{writeArray}\ (*a)\ n\ t_2 \rightsquigarrow_s H' \vdash \mathbf{writeArray}\ (*a)\ n\ t_2'}\ \rightsquigarrow_{\textsc{prim}}$$

– $t_1 = \mathbf{deleteArray}$ therefore $A = *(\mathsf{Array}_{id}\,\mathbb{F}) \multimap \mathsf{unit}$

* t_2 is a value and therefore by the value lemma (Lemma C.1) $t_2 = *a$ thus the typing is refined at runtime as follows:

$$\dfrac{\dfrac{[\Gamma], a : \mathsf{Array}_{id}\,\mathbb{F} \vdash a : (\mathsf{Array}_{id}\,\mathbb{F})\ \textsc{ref}}{[\Gamma], a : \mathsf{Array}_{id}\,\mathbb{F} \vdash *a : *(\mathsf{Array}_{id}\,\mathbb{F})}\ \textsc{nec}}{[\Gamma], a : \mathsf{Array}_{id}\,\mathbb{F} \vdash \mathbf{deleteArray}\ *a : \mathsf{unit}}\ \textsc{TyDeriveddeleteArray}$$

with $H' \bowtie (\Gamma_0 + [\Gamma], a : \mathsf{Array}_{id}\,\mathbb{F})$, and by the heap compatibility rule for array references there exists some H such that $H' = H, a \mapsto_p id, id \mapsto \mathbf{arr}$.
There there is a reduction as follows:

$$\dfrac{}{H, \mathit{ref} \mapsto_p id, id \mapsto \mathbf{arr} \vdash \mathbf{deleteArray}\ (*\mathit{ref}) \rightsquigarrow_s H \vdash ()}\ \rightsquigarrow_{\textsc{deleteArray}}$$

* t_2 is not a value and thus has a reduction, therefore we can build the compound reduction:

$$\dfrac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \mathbf{deleteArray}\ t_2 \rightsquigarrow_s H' \vdash \mathbf{deleteArray}\ t_2'}\ \rightsquigarrow_{\textsc{prim}}$$

(2) Otherwise, by induction on the premise we then have that there exists heap H', term t_1', and context Γ' such that $H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'$. Therefore we can reduce by the following rule:

$$\dfrac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash t_1\ t_2 \rightsquigarrow_s H' \vdash t_1'\ t_2}\ \rightsquigarrow_{\textsc{appL}}$$

- (pr)

$$\dfrac{\Gamma \vdash t : A \qquad \neg\mathsf{resourceAllocator}(t)}{r \cdot \Gamma \vdash [t]_r : \Box_r A}\ \textsc{pr}$$

with heap compatibility $H \bowtie \Gamma_0 + s \cdot (r \cdot \Gamma)$, which by associativity of $*$ is equal to $H \bowtie \Gamma_0 + (s * r) \cdot \Gamma$. Then, by induction on the premise (with $s' = s * r$), there are two possibilities:

(1) If t is a value, say v, then $[t]$ is also a value, $[v]$.

(2) Otherwise, there exists H' and t' such that $H \vdash t \rightsquigarrow_{s*r} H' \vdash t'$, from which we can then reduce by the following rule:

$$\frac{H \vdash t \rightsquigarrow_{s*r} H' \vdash t'}{H \vdash [t]_r \rightsquigarrow_s H' \vdash [t']_r} \rightsquigarrow_{\Box}$$

- (elim)

$$\frac{\Gamma_1 \vdash t_1 : \Box_r A \qquad \Gamma_2, x : [A]_r \vdash t_2 : B}{\Gamma_1 + \Gamma_2 \vdash \textbf{let}\,[x] = t_1 \textbf{ in } t_2 : B} \text{ ELIM}$$

By induction on the first premise, there are two possibilities.

(1) t_1 is a value and therefore by the value lemma $t_1 = [v]$. This refines the typing as follows:

$$\frac{\dfrac{\Gamma_1 \vdash v : A}{r \cdot \Gamma_1 \vdash [v]_r : \Box_r A} \text{ PR} \qquad \Gamma_2, x : \Box_r A \vdash t_2 : B}{r \cdot \Gamma_1 + \Gamma_2 \vdash \textbf{let}\,[x] = [v]_r \textbf{ in } t_2 : B} \text{ ELIM}$$

Therefore we can reduce by the following rule:

$$\frac{y \# \{H, v, t\}}{H \vdash \textbf{let}\,[x] = [v]_r \textbf{ in } t \rightsquigarrow_s H, y \mapsto_{s*r} v \vdash t[y/x]} \rightsquigarrow_{\Box\beta}$$

(2) Otherwise by induction on the premise we then have that there exists heap H', term t_1' and context Γ' such that $H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'$. Therefore we can reduce by the following rule:

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash \textbf{let}\,[x] = t_1 \textbf{ in } t_2 \rightsquigarrow_s H' \vdash \textbf{let}\,[x] = t_1' \textbf{ in } t_2} \rightsquigarrow_{\text{LET}\Box}$$

- (der)

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma, x : [A]_1 \vdash t : B} \text{ DER}$$

Goal achieved immediately by induction on the premise.

- (approx)

$$\frac{\Gamma, x : [A]_r, \Gamma' \vdash t : B \qquad r \sqsubseteq s}{\Gamma, x : [A]_s, \Gamma' \vdash t : B} \text{ APPROX}$$

Goal achieved immediately by induction on the premise.

- (pairIntro)

$$\frac{\Gamma_1 \vdash t_1 : A \qquad \Gamma_2 \vdash t_2 : B}{\Gamma_1 + \Gamma_2 \vdash (t_1, t_2) : A \otimes B} \otimes_I$$

By induction on the premises there are three possible cases.

(1) If both t_1 and t_2 are values then (t_1, t_2) is also a value.

(2) If only t_1 is a value then by induction on the second premise we then have that there exists heap H', term t_2' and context Γ' such that $H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'$. Therefore we can reduce by the following rule:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash (v, t_2) \rightsquigarrow_s H' \vdash (v, t_2')} \rightsquigarrow_{\otimes\text{R}}$$

(3) If neither t_1 nor t_2 are values then by induction on the first premise we then have that there exists heap H', term t_1' and context Γ' such that $H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'$. Therefore we can reduce by the following rule:

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash (t_1, t_2) \rightsquigarrow_s H' \vdash (t_1', t_2)} \rightsquigarrow_{\otimes \mathrm{L}}$$

- (pairElim)

$$\frac{\Gamma_1 \vdash t_1 : A \otimes B \qquad \Gamma_2, x : A, y : B \vdash t_2 : C}{\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\,(x, y) = t_1\,\mathbf{in}\,t_2 : C} \otimes_E$$

By induction on the first premise, there are two possibilities.

(1) t_1 is a value and therefore by the value lemma $t_1 = (v_1, v_2)$. This refines the typing as follows:

$$\frac{\dfrac{\Gamma_1 \vdash v_1 : A \qquad \Gamma_2 \vdash v_2 : B}{\Gamma_1 + \Gamma_2 \vdash (v_1, v_2) : A \otimes B} \otimes_I \qquad \Gamma_3, x : A, y : B \vdash t_2 : C}{\Gamma_1 + \Gamma_2 + \Gamma_3 \vdash \mathbf{let}\,(x, y) = (v_1, v_2)\,\mathbf{in}\,t_2 : C} \otimes_E$$

Therefore, we can reduce by the following rule:

$$\frac{x' \# \{H, v_1, v_2, t\} \qquad y' \# \{H, v_1, v_2, t\}}{H \vdash \mathbf{let}\,(x, y) = (v_1, v_2)\,\mathbf{in}\,t \rightsquigarrow_s H, x' \mapsto_s v_1, y' \mapsto_s v_2 \vdash t[y'/y][x'/x]} \rightsquigarrow_{\otimes \beta}$$

(2) Otherwise, by induction on the premise we then have that there exists heap H', term t_1' and context Γ' such that $H \vdash t \rightsquigarrow_s H' \vdash t_1'$. Therefore we can reduce by the following rule:

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash \mathbf{let}\,(x, y) = t_1\,\mathbf{in}\,t_2 \rightsquigarrow_s H' \vdash \mathbf{let}\,(x, y) = t_1'\,\mathbf{in}\,t_2} \rightsquigarrow_{\mathrm{LET}\otimes}$$

- (unitIntro)

$$\frac{}{0 \cdot \Gamma \vdash () : \mathsf{unit}} 1_I$$

A value.

- (unitElim)

$$\frac{\Gamma_1 \vdash t_1 : \mathsf{unit} \qquad \Gamma_2 \vdash t_2 : B}{\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\,() = t_1\,\mathbf{in}\,t_2 : B} 1_E$$

By induction on the first premise, there are two possibilities.

(1) t is a value and therefore by the value lemma $t = ()$. Therefore we can reduce by the following rule:

$$\frac{}{H \vdash \mathbf{let}\,() = ()\,\mathbf{in}\,t \rightsquigarrow_s H \vdash t} \rightsquigarrow_{\beta \mathrm{UNIT}}$$

(2) Otherwise by induction on the premise we then have that there exists heap H', term t_1' and context Γ' such that $H \vdash t \rightsquigarrow_s H' \vdash t_1'$. Therefore we can reduce by the following rule:

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash \mathbf{let}\,() = t_1\,\mathbf{in}\,t_2 \rightsquigarrow_s H' \vdash \mathbf{let}\,() = t_1'\,\mathbf{in}\,t_2} \rightsquigarrow_{\mathrm{LETUNIT}}$$

- (returnGen)

$$\frac{\Gamma \vdash t : *A}{\Gamma \vdash \textbf{share}\ t : \Box_r A}\ \textsc{share}$$

By induction on the premise, there are two possibilities.

(1) t is a value and therefore by the value lemma $t = *v$. Therefore we can reduce by the following rule:

$$\frac{\mathsf{dom}(H) \equiv \mathsf{refs}(v)}{H, H' \vdash \textbf{share}\ (*v) \rightsquigarrow_s ([H]_0), H' \vdash [v]}\ \rightsquigarrow_{\textsc{share}\beta}$$

(2) Otherwise by induction on the premise we then have that there exists heap H', term t' and context Γ' such that $H \vdash t \rightsquigarrow_s H' \vdash t'$. Therefore we can reduce by the following rule:

$$\frac{H \vdash t \rightsquigarrow_s H' \vdash t'}{H \vdash \textbf{share}\ t \rightsquigarrow_s H' \vdash \textbf{share}\ t'}\ \rightsquigarrow_{\textsc{share}}$$

- (bindGen)

$$\frac{\Gamma_1 \vdash t_1 : \Box_r A \qquad \Gamma_2, x : *A \vdash t_2 : \Box_r B \qquad 1 \sqsubseteq r}{\Gamma_1 + \Gamma_2 \vdash \textbf{clone'}\ t_1\ \textbf{as}\ x\ \textbf{in}\ t_2 : \Box_r B}\ \textsc{clone'}$$

By induction on the first premise, there are two possibilities.

(1) t_1 is a value and therefore by the value lemma $t_1 = [v]$. This refines the typing as follows:

$$\frac{\dfrac{\Gamma_1 \vdash v : A}{r \cdot \Gamma_1 \vdash [v] : \Box_r A}\ \textsc{pr} \qquad \Gamma_2, x : *A \vdash t_2 : \Box_r B}{r \cdot \Gamma_1 + \Gamma_2 \vdash \textbf{clone}\ [v]\ \textbf{as}\ x\ \textbf{in}\ t_2 : \Box_r B}\ \textsc{clone'}$$

Therefore we can reduce using the following rule (with $t = t_2$):

$$\frac{\mathsf{dom}(H') \equiv \mathsf{refs}(v) \qquad (H'', \theta, \overline{id}) = \mathsf{copy}(H') \qquad y\#\{H, v, t\}}{H, H' \vdash \textbf{clone}\ [v]_r\ \textbf{as}\ x\ \textbf{in}\ t \rightsquigarrow_s H, H', H'', y \mapsto_s \textbf{pack}\ \langle \overline{id}, *(\theta(v)) \rangle \vdash t[y/x]}\ \rightsquigarrow_{\textsc{clone}\beta}$$

(2) Otherwise by induction on the premise we then have that there exists heap H', term t_1' and context Γ' such that $H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'$. Therefore we can reduce by the following rule:

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash \textbf{clone}\ t_1\ \textbf{as}\ x\ \textbf{in}\ t_2 \rightsquigarrow_s H' \vdash \textbf{clone}\ t_1'\ \textbf{as}\ x\ \textbf{in}\ t_2}\ \rightsquigarrow_{\textsc{clone}}$$

- (withBorrow)

$$\frac{\Gamma_1 \vdash t_1 : *A \qquad \Gamma_2 \vdash t_2 : \&_1 A \multimap \&_1 B}{\Gamma_1 + \Gamma_2 \vdash \textbf{withBorrow}\ t_1\ t_2 : *B}\ \textsc{with}\&$$

By induction on the first premise, there are two possibilities.

(1) t_1 is a value and therefore by the value lemma $t_1 = (\lambda x.t)$. Then, again, we have two possibilities:
  - t_2 is also a value, and therefore by the value lemma $t_2 = (*v)$ for some v. Then we can reduce by the following rule:

$$\frac{y\#\{H, v, t\}}{H \vdash \textbf{withBorrow}\ (\lambda x.t)\ (*v) \rightsquigarrow_s H, y \mapsto_s (*v) \vdash \textbf{unborrow}\ t[y/x]}\ \rightsquigarrow_{\textsc{with}\&}$$

– t_2 is not a value. Then, by induction on the premise we have that there exists heap H', term t_2' and context Γ' such that $H \vdash t_1 \rightsquigarrow_s H' \vdash t_2'$. Therefore we can reduce by the following rule:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \textbf{withBorrow}\ (\lambda x.t_1)\ t_2 \rightsquigarrow_s H' \vdash \textbf{withBorrow}\ (\lambda x.t_1)\ t_2'} \rightsquigarrow_{\text{WITH\&R}}$$

(2) t_1 is not a value. Then, by induction on the premise we have that there exists heap H', term t_1' and context Γ' such that $H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'$. Therefore we can reduce by the following rule:

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash \textbf{withBorrow}\ t_1\ t_2 \rightsquigarrow_s H' \vdash \textbf{withBorrow}\ t_1'\ t_2} \rightsquigarrow_{\text{WITH\&L}}$$

- (split)

$$\frac{\Gamma \vdash t : \&_p A}{\Gamma \vdash \textbf{split}\ t : \&_{\frac{p}{2}} A \otimes \&_{\frac{p}{2}} A}\ \text{SPLIT}$$

By induction on the premise, there are two cases.

(1) If t is a value, then by the value lemma and the unique value lemma there are three possibilities for the form of t.

– t could have the form $(*ref)$. This refines the typing as follows:

$$\frac{[\Gamma], ref : \mathsf{Ref}_{id}\ A \vdash (*ref) : \&_p(\mathsf{Ref}_{id}\ A)\ \text{REF + NEC}}{[\Gamma], ref : \mathsf{Ref}_{id}\ A \vdash \textbf{split}\ (*ref) : \&_{\frac{p}{2}}(\mathsf{Ref}_{id}\ A) \otimes \&_{\frac{p}{2}}(\mathsf{Ref}_{id}\ A)}\ \text{SPLIT}$$

Then we can reduce by the following rule:

$$\frac{ref_1 \# H \qquad ref_2 \# H}{H, ref \mapsto_p id, id \mapsto v \vdash \textbf{split}\ (*ref) \rightsquigarrow_s H, ref_1 \mapsto_{\frac{p}{2}} id, ref_2 \mapsto_{\frac{p}{2}} id, id \mapsto v \vdash (*ref_1, *ref_2)} \rightsquigarrow_{\text{SPLITREF}}$$

– t could have the form $(*a)$. This refines the typing as follows:

$$\frac{[\Gamma], a : \mathsf{Array}_{id}\ \mathbb{F} \vdash (*a) : \&_p(\mathsf{Array}_{id}\ \mathbb{F})\ \text{REF + NEC}}{[\Gamma], a : \mathsf{Array}_{id}\ \mathbb{F} \vdash \textbf{split}\ (*a) : \&_{\frac{p}{2}}(\mathsf{Array}_{id}\ \mathbb{F}) \otimes \&_{\frac{p}{2}}(\mathsf{Array}_{id}\ \mathbb{F})}\ \text{SPLIT}$$

Then we can reduce by the following rule:

$$\frac{a_1 \# H \qquad a_2 \# H}{H, a \mapsto_p id, id \mapsto \textbf{arr} \vdash \textbf{split}\ (*a) \rightsquigarrow_s H, a_1 \mapsto_{\frac{p}{2}} id, a_2 \mapsto_{\frac{p}{2}} id, id \mapsto \textbf{arr} \vdash (*a_1, *a_2)} \rightsquigarrow_{\text{SPLITARR}}$$

– t could have the form $(*(v, w))$. There are two possible typings in this case:

$$\frac{\dfrac{\dfrac{\gamma_1 \vdash v_1 : A' \qquad \gamma_2 \vdash v_2 : B}{\gamma_1 + \gamma_2 \vdash (v_1, v_2) : A' \otimes B}\ \otimes_I}{\gamma_1 + \gamma_2 \vdash *(v_1, v_2) : *(A' \otimes B)}\ \text{NEC}}{\gamma_1 + \gamma_2 \vdash \textbf{split}\ (*(v_1, v_2)) : (\&_{\frac{1}{2}}(A' \otimes B) \otimes \&_{\frac{1}{2}}(A' \otimes B))}\ \text{SPLIT}$$

In either instance, we can reduce by the following rule:

$$\frac{\begin{array}{c} H \vdash \textbf{split}\ (*v) \ \leadsto_s\ H' \vdash (*v_1, *v_2) \\ H' \vdash \textbf{split}\ (*w) \ \leadsto_s\ H'' \vdash (*w_1, *w_2) \end{array}}{H \vdash \textbf{split}\ (*(v,w)) \ \leadsto_s\ H'' \vdash (*(v_1,w_1), *(v_2,w_2))}\ \leadsto_{\text{SPLIT}\otimes}$$

(2) If t is not a value, then by induction on the premise we have that there exists heap H', term t' and context Γ' such that $H \vdash t \ \leadsto_s\ H' \vdash t'$. Therefore we can reduce by the following rule:

$$\frac{H \vdash t \ \leadsto_s\ H' \vdash t'}{H \vdash \textbf{split}\ t \ \leadsto_s\ H' \vdash \textbf{split}\ t'}\ \leadsto_{\text{SPLIT}}$$

- (join)

$$\frac{\Gamma_1 \vdash t_1 : \&_p A \qquad \Gamma_2 \vdash t_2 : \&_q A \qquad p + q \leq 1}{\Gamma_1 + \Gamma_2 \vdash \textbf{join}\ t_1\ t_2 : \&_{p+q} A}\ \text{JOIN}$$

By induction on the first premise, there are two cases.

(1) If t_1 is a value, then first we should consider whether t_2 is also a value.
  - If t_2 is also a value, then by the value lemma and the unique value lemma there are three possibilities for the form of t_1.

  (a) t_1 could have the form $(*ref_1)$. This refines the typing as follows:

$$\frac{[\Gamma], ref_1 : \mathsf{Ref}_{id}\ A \vdash (*ref_1) : \&_p(\mathsf{Ref}_{id}\ A)\ \text{REF+NEC} \qquad [\Gamma], ref_2 : \mathsf{Ref}_{id}\ A \vdash (*ref_2) : \&_q(\mathsf{Ref}_{id}\ A)\ \text{REF+NEC}}{[\Gamma], ref_1 : \mathsf{Ref}_{id}\ A, ref_2 : \mathsf{Ref}_{id}\ A \vdash \textbf{join}\ (*ref_1)\ (*ref_2) : \&_{p+q}(\mathsf{Ref}_{id}\ A)}\ \text{JOIN}$$

Note that the typing in this case restricts t_2 to also have the form $*ref_2$. Then we can reduce by the following rule:

$$\frac{ref \# H}{H, ref_1 \mapsto_p id, ref_2 \mapsto_q id, id \mapsto v \vdash \textbf{join}\ (*ref_1)\ (*ref_2) \ \leadsto_s\ H, ref \mapsto_{(p+q)} id, id \mapsto v \vdash *ref}\ \leadsto_{\text{JOINREF}}$$

  (b) t_1 could have the form $(*a_1)$. This refines the typing as follows:

$$\frac{[\Gamma], a_1 : \mathsf{Array}_{id}\ \mathbb{F} \vdash (*a_1) : \&_p(\mathsf{Array}_{id}\ \mathbb{F})\ \text{REF+NEC} \qquad [\Gamma], a_2 : \mathsf{Array}_{id}\ \mathbb{F} \vdash (*a_2) : \&_q(\mathsf{Array}_{id}\ \mathbb{F})\ \text{REF+NEC}}{[\Gamma], a_1 : \mathsf{Array}_{id}\ \mathbb{F}, a_2 : \mathsf{Array}_{id}\ \mathbb{F} \vdash \textbf{join}\ (*a_1)\ (*a_2) : \&_{p+q}(\mathsf{Array}_{id}\ \mathbb{F})}\ \text{JOIN}$$

Note that the typing in this case restricts t_2 to also have the form $*a_2$. Then we can reduce by the following rule:

$$\frac{a \# H}{H, a_1 \mapsto_p id, a_2 \mapsto_q id, id \mapsto \textbf{arr} \vdash \textbf{join}\ *a_1\ *a_2 \ \leadsto_s\ H, a \mapsto_{(p+q)} id, id \mapsto \textbf{arr} \vdash *a}\ \leadsto_{\text{JOINARR}}$$

  (c) t_1 could have the form $(*(v_1, w_1))$. There are two possible typings in this case:

$$\frac{\dfrac{\dfrac{\gamma_1 \vdash v_1 : A' \quad \gamma_2 \vdash w_1 : B}{\gamma_1 + \gamma_2 \vdash (v_1, w_1) : A' \otimes B}\ \otimes_I}{\gamma_1 + \gamma_2 \vdash *(v_1, w_1) : *(A' \otimes B)}\ \text{NEC} \qquad \dfrac{\dfrac{\gamma_3 \vdash v_2 : A' \quad \gamma_4 \vdash w_2 : B}{\gamma_3 + \gamma_4 \vdash (v_2, w_2) : A' \otimes B}\ \otimes_I}{\gamma_3 + \gamma_4 \vdash *(v_2, w_2) : *(A' \otimes B)}\ \text{NEC}}{\gamma_1 + \gamma_2 + \gamma_3 + \gamma_4 \vdash \textbf{join}\ (*(v_1, w_1)}\ \text{JOIN}$$

Note that the typing in both situations restricts t_2 to also have the form $(*(v_2, w_2))$. In either instance we can reduce by the following rule:

$$\frac{H \vdash \textbf{join}\ (*v_1)\ (*v_2)\ \rightsquigarrow_s\ H' \vdash *v \qquad H' \vdash \textbf{join}\ (*w_1)\ (*w_2)\ \rightsquigarrow_s\ H'' \vdash *w}{H \vdash \textbf{join}\ (*(v_1, w_1))\ (*(v_2, w_2))\ \rightsquigarrow_s\ H'' \vdash *(v, w)} \rightsquigarrow_{\text{JOIN}\otimes}$$

- If t_2 is not a value, then by induction on the second premise we have that there exists heap H', term t_2' and context Γ' such that $H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'$. Therefore we can reduce by the following rule:

$$\frac{H \vdash t_2\ \rightsquigarrow_s\ H' \vdash t_2'}{H \vdash \textbf{join}\ v\ t_2\ \rightsquigarrow_s\ H' \vdash \textbf{join}\ v\ t_2'} \rightsquigarrow_{\text{JOINR}}$$

(2) If t_1 is not a value, then by induction on the second premise we have that there exists heap H', term t_1' and context Γ' such that $H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'$. Therefore we can reduce by the following rule:

$$\frac{H \vdash t_2\ \rightsquigarrow_s\ H' \vdash t_2'}{H \vdash \textbf{join}\ v\ t_2\ \rightsquigarrow_s\ H' \vdash \textbf{join}\ v\ t_2'} \rightsquigarrow_{\text{JOINR}}$$

- (nec)

$$\frac{\gamma \vdash t : A \qquad \neg\mathsf{resourceAllocator}(t)}{0 \cdot \Gamma, \gamma \vdash *t : *A} \ \text{NEC}$$

By induction on the premise, there are two possibilities.

(1) If t is a value, then $*t$ is also a value.

(2) Otherwise by induction on the premise we then have that there exists heap H', term t' and context Γ' such that $H \vdash t \rightsquigarrow_s H' \vdash t'$. Therefore we can reduce by the following rule:

$$\frac{H \vdash t\ \rightsquigarrow_s\ H' \vdash t'}{H \vdash *t\ \rightsquigarrow_s\ H' \vdash *t'} \rightsquigarrow_{*}$$

- (push)

$$\frac{\Gamma \vdash t : \&_p(A \otimes B)}{\Gamma \vdash \textbf{push}\ t : (\&_p A) \otimes (\&_p B)} \ \text{PUSH}$$

By induction on the premise, there are two possibilities.

(1) If t is a value, then by the value lemma and the unique value lemma there are two possibilities for the form of t.
   - t could have the form $*(v_1, v_2)$. Then we can reduce by the following rule:

$$\frac{}{H \vdash \textbf{push}\ (*(v_1, v_2))\ \rightsquigarrow_s\ H \vdash (*v_1, *v_2)} \rightsquigarrow_{\text{PUSH}*}$$

(2) If t is not a value, by induction on the premise we then have that there exists heap H', term t' and context Γ' such that $H \vdash t \rightsquigarrow_s H' \vdash t'$. Therefore we can reduce by the following rule:

$$\frac{H \vdash t\ \rightsquigarrow_s\ H' \vdash t'}{H \vdash \textbf{push}\ t\ \rightsquigarrow_s\ H' \vdash \textbf{push}\ t'} \rightsquigarrow_{\text{PUSH}}$$

- (pull)

$$\frac{\Gamma \vdash t : (\&_p A) \otimes (\&_p B)}{\Gamma \vdash \textbf{pull}\ t : \&_p(A \otimes B)} \ \text{PULL}$$

By induction on the premise, there are two possibilities.

(1) If t is a value, then by the value lemma and the unique value lemma there are two possibilities for the form of t.
   - t could have the form $(*v_1, *v_2)$. Then we can reduce by the following rule:

$$\frac{}{H \vdash \textbf{pull}\,(*v_1, *v_2) \ \rightsquigarrow_s\ H \vdash *(v_1, v_2)} \ \rightsquigarrow_{\text{PULL}*}$$

(2) If t is not a value, by induction on the premise we then have that there exists heap H', term t' and context Γ' such that $H \vdash t \ \rightsquigarrow_s\ H' \vdash t'$. Therefore we can reduce by the following rule:

$$\frac{H \vdash t \ \rightsquigarrow_s\ H' \vdash t'}{H \vdash \textbf{pull}\, t \ \rightsquigarrow_s\ H' \vdash \textbf{pull}\, t'} \ \rightsquigarrow_{\text{PULL}}$$

- (ref)

$$\frac{}{0 \cdot \Gamma, \mathit{ref} : \mathit{Res}_{id}\ A \vdash \mathit{ref} : \mathit{Res}_{id}\ A} \ \text{REF}$$

  A value.
- (array)

$$\frac{}{0 \cdot \Gamma, \mathit{ref} : \mathit{Res}_{id}\ A \vdash \mathit{ref} : \mathit{Res}_{id}\ A} \ \text{REF}$$

  A value.
- (unborrow)

$$\frac{\Gamma \vdash t : \&_1 A}{\Gamma \vdash \textbf{unborrow}\ t : *A} \ \text{UNBORROW}$$

  By induction on the premise, there are two possibilities.

(1) t is a value, and therefore by the value lemma $t = (*v)$ for some value v. Then we can reduce by the following rule:

$$\frac{}{H \vdash \textbf{unborrow}\ (*v) \ \rightsquigarrow_s\ H \vdash *v} \ \rightsquigarrow_{\text{UN}\&}$$

(2) t is not a value. By induction on the premise we then have that there exists heap H', term t' and context Γ' such that $H \vdash t \ \rightsquigarrow_s\ H' \vdash t'$. Therefore we can reduce by the following rule:

$$\frac{H \vdash t \ \rightsquigarrow_s\ H' \vdash t'}{H \vdash \textbf{unborrow}\ t \ \rightsquigarrow_s\ H' \vdash \textbf{unborrow}\ t'} \ \rightsquigarrow_{\text{UNBORROW}}$$

- (pack)

$$\frac{\Gamma \vdash t : A \qquad id \notin \text{dom}(\Gamma)}{\Gamma \vdash \textbf{pack}\ \langle id', t\rangle : \exists id.A[id/id']} \ \text{PACK}$$

  By induction on the premise, there are two possibilities.

(1) If t is a value, then $\textbf{pack}\ \langle id', t\rangle$ is also a value.
(2) Otherwise by induction on the premise we then have that there exists heap H', term t' and context Γ' such that $H \vdash t \ \rightsquigarrow_s\ H' \vdash t'$. Therefore we can reduce by the following rule:

$$\frac{H \vdash t \ \rightsquigarrow_s\ H \vdash t'}{H \vdash \textbf{pack}\ \langle id, t\rangle \ \rightsquigarrow_s\ H \vdash \textbf{pack}\ \langle id, t'\rangle} \ \rightsquigarrow_{\text{PACK}}$$

- (unpack)

$$\frac{\Gamma_1 \vdash t_1 : \exists id.A \qquad \Gamma_2, id, x : A \vdash t_2 : B \qquad id \notin \mathsf{fv}(B)}{\Gamma_1 + \Gamma_2 \vdash \textbf{unpack}\ \langle id, x\rangle = t_1\ \textbf{in}\ t_2 : B}\ \textsc{unpack}$$

By induction on the premise, there are two possibilities.

(1) If t_1 is a value, then by the value lemma it has the form **pack** $\langle id', v\rangle$ for some value v. Then we can reduce by the following rule:

$$\frac{y\#\{H, v, t\}}{H \vdash \textbf{unpack}\ \langle id, x\rangle = \textbf{pack}\ \langle id', v\rangle\ \textbf{in}\ t \rightsquigarrow_s H, y \mapsto_r v \vdash t[y/x]} \rightsquigarrow_{\exists\beta}$$

(2) Otherwise by induction on the premise we then have that there exists heap H', term t_1' and context Γ' such that $H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'$. Therefore we can reduce by the following rule:

$$\frac{H \vdash t_1 \rightsquigarrow_s H \vdash t_1'}{H \vdash \textbf{unpack}\ \langle id, x\rangle = t_1\ \textbf{in}\ t_2 \rightsquigarrow_s H \vdash \textbf{unpack}\ \langle id, x\rangle = t_1'\ \textbf{in}\ t_2} \rightsquigarrow_{\textsc{unpack}}$$

□

C.2 Type preservation proof

Lemma C.5 (Admissibility of weakening).

$$\Gamma \vdash t : A \implies (0 \cdot \Gamma'), \Gamma \vdash t : A$$

Proof. By induction on the structure of typing and since all ‘leaf’ nodes of a typing derivation permit weakening (e.g., var rule, primitive rules). □

Lemma C.6 (Renaming array refs). *Given an array reference renaming θ (generated from a clone) then $\Gamma \vdash t : A \implies \theta(\Gamma) \vdash \theta(t) : A$.*

Proof. By trivial induction, with the only action happening in the use of the ref runtime typing rule for array references, in which case this acts just like alpha renaming via θ. □

Theorem C.7 (Type Preservation). *For a well-typed term $\Gamma \vdash t : A$, under a restriction that polymorphic reference resources are restricted to non-function types, and for all s, Γ_0, and H such that $H \bowtie (\Gamma_0 + s \cdot \Gamma)$ and a reduction $H \vdash t \rightsquigarrow_s H' \vdash t'$, then we have:*

$$\exists \Gamma', H'.\ \ \Gamma' \vdash t' : A \ \ \wedge \ \ H' \bowtie (\Gamma_0 + s \cdot \Gamma')$$

Note the caveat to preservation: references $\mathsf{Ref}_{id}\ A$ are restricted such that A cannot be of function type, or some other composite type involving functions. The restriction is needed for preservation since it works at the granularity of a single reduction, and so cannot rule out the possibility that a reference is storing a λ term with free variables. This considerably complicates reasoning about resources and heaps, so we rule it out for this theorem. Importantly, this is not a restriction that needs to be made on the calculus and its implementation as a whole: for deterministic CBV reduction starting from a closed term (i.e., a complete program) then all β-redexes are on closed values and hence this problem does not exist in the context of an overall reduction sequence. However, to make preservation work for a single reduction, on potentially open terms, this minor restriction is needed locally. This does not affect any of the examples discussed in the paper.

Proof. By induction on the structure of typing and reductions.

- (var)

$$\frac{}{0 \cdot \Gamma, x : A \vdash x : A}\ \text{VAR}$$

and one possible reduction:

$$\frac{\exists r'.\ s + r' \sqsubseteq r}{H, x \mapsto_r v \vdash x \rightsquigarrow_s H, x \mapsto_r v \vdash v}\ \rightsquigarrow_{\text{VAR}}$$

with incoming heap compatibility:

$$(H, x \mapsto_r v) \bowtie (\Gamma_0 + s \cdot (0 \cdot \Gamma, x : A))$$

with derivation:

$$\frac{H \bowtie (\Gamma_0 + s * 0 \cdot \Gamma + s \cdot \Gamma') \qquad x \notin \mathsf{dom}(H) \qquad \Gamma' \vdash v : A \qquad \exists r''.\ s + r'' \equiv r}{(H, x \mapsto_r v) \bowtie (\Gamma_0 + s * 0 \cdot \Gamma, x : [A]_s)}\ \text{EXT}$$

Goal 1:

$$\exists \Gamma''.\ \Gamma'' \vdash v : A$$

Which is provided by the third premise of the head compatibility derivation here, but with weakening (Lemma C.5) such that we have:

$$\frac{\Gamma' \vdash v : A}{0 \cdot \Gamma, \Gamma', x : [A]_0 \vdash v : A}\ \text{Lemma C.5}$$

Goal 2

$$(H, x \mapsto_r v) \bowtie (\Gamma_0 + s \cdot (0 \cdot \Gamma, \Gamma', x : [A]_0))$$

given by the following derivation from the premise of the incoming heap compatibility:

$$\frac{\dfrac{H \bowtie (\Gamma_0 + s * 0 \cdot \Gamma + s \cdot \Gamma')}{H \bowtie ((\Gamma_0 + s * 0 \cdot \Gamma + s \cdot \Gamma') + 0 \cdot \Gamma')}\ 0\ \text{unitality} \qquad x \notin \mathsf{dom}(H) \qquad \Gamma' \vdash v : A \qquad \exists r'''.\ 0 + r''' \equiv r}{H, x \mapsto_r v \bowtie ((\Gamma_0 + s * 0 \cdot \Gamma + s \cdot \Gamma'), x : [A]_0)}\ \text{EXT}$$

where the premise $\exists r''''.\ 0 + r''' \equiv r$ is fulfilled by $r''' = r$ by unitality and since $0 * s = 0$.

- (abs)

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \multimap B}\ \text{ABS}$$

Has no reduction so the case is trivial here.

- (app)

$$\frac{\Gamma_1 \vdash t_1 : A \multimap B \qquad \Gamma_2 \vdash t_2 : A}{\Gamma_1 + \Gamma_2 \vdash t_1\ t_2 : B}\ \text{APP}$$

with incoming heap compatibility $H \bowtie \Gamma_0 + s \cdot (\Gamma_1 + \Gamma_2)$.
There are four possible general reductions (and then further reductions for the operation of primitives, separated our below).

General reductions (app).

(1) (beta)

$$\frac{y\#\{H, v, t\}}{H \vdash (\lambda x.t)\ v \rightsquigarrow_s H, y \mapsto_s v \vdash t[y/x]} \rightsquigarrow_\beta$$

i.e. $t_1 = (\lambda x.t)\ v$ with the refined typing (where $\Gamma = \Gamma_2$):

$$\frac{\dfrac{\Gamma_1, x : A \vdash t : B}{\Gamma_1 \vdash \lambda x.t : A \multimap B}\ \text{ABS} \qquad \Gamma_2 \vdash v : A}{\Gamma_1 + \Gamma_2 \vdash (\lambda x.t)\ v : B}\ \text{APP}$$

Therefore the resulting typing judgment is (goal) $\Gamma_1, x : A \vdash t : B$ given by the first premise here.

The goal heap compatibility is: (goal 2) $(H, x \mapsto_1 v) \bowtie (\Gamma_0 + s \cdot (\Gamma_1, x : A))$

We construct the goal compatibility judgment as follows, using the incoming compatibility assumption:

$$\frac{H \bowtie (\Gamma_0 + s \cdot \Gamma_1 + s \cdot \Gamma_2) \quad x \notin \text{dom}(H) \quad \Gamma_2 \vdash v : A \quad \exists r'.\ s + r' \equiv 1}{H, x \mapsto_1 v \bowtie (\Gamma_0 + s \cdot \Gamma_1), x : [A]_s}\ \text{EXT}$$

where $r' = 0$ here.

(2) (appL)

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash t_1\ t_2 \rightsquigarrow_s H' \vdash t_1'\ t_2} \rightsquigarrow_{\text{APPL}}$$

with incoming heap compatibility $H \bowtie (\Gamma_0' + s \cdot (\Gamma_1 + \Gamma_2))$.

Applying induction on the premise reduction with heap compatibility given by the incoming heap compatibility but with $\Gamma_0 = \Gamma_0' + s \cdot \Gamma_2$ then yields:

(a) $\exists \Gamma_1'.\ \Gamma_1' \vdash t_1' : A \multimap B$

(b) $H' \bowtie (\Gamma_0' + s \cdot \Gamma_2 + s \cdot \Gamma_1')$

(Goal 1) is then given by the reconstructed application type $\Gamma_1' + \Gamma_2 \vdash t_1'\ t_2 : B$

(Goal 2) is $H' \bowtie (\Gamma_0' + s \cdot (\Gamma_1' + \Gamma_2))$ which is given by the second conjunct above by commutativity of + and distributivity of $*$ over +.

(3) (appR)

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash v\ t_2 \rightsquigarrow_s H' \vdash v\ t_2'} \rightsquigarrow_{\text{APPR}}$$

Same as (appL) but by induction on the premise with t_2.

(4) (prim)

$$\frac{H \vdash t \rightsquigarrow_s H' \vdash t'}{H \vdash pr\ t \rightsquigarrow_s H' \vdash pr\ t'} \rightsquigarrow_{\text{PRIM}}$$

Same as (appL) but by induction on the premise with t.

Primitives (app).

(1) (newRef)

$$\frac{ref\#H \qquad id\#H}{H \vdash \textbf{newRef}\ v \rightsquigarrow_s H, ref \mapsto_1 id, id \mapsto \textbf{ref}(v) \vdash \textbf{pack}\ \langle id, *ref \rangle} \rightsquigarrow_{\text{NEWREF}}$$

Thus typing refines to:

$$\frac{0 \cdot \Gamma_1 \vdash \textbf{newRef} : A \multimap \exists id.*(\text{Ref}_{id}\ A) \quad \Gamma_2 \vdash v : A}{0 \cdot \Gamma_1 + \Gamma_2 \vdash \textbf{newRef}\ v : \exists id.*(\text{Ref}_{id}\ A)}\ \text{APP}$$

with incoming heap compatibility $H \bowtie \Gamma_0 + s \cdot (0 \cdot \Gamma_1 + \Gamma_2)$ which is equal to $H \bowtie \Gamma_0 + 0 \cdot \Gamma_1 + s \cdot \Gamma_2$ by distributivity of $*$ over $+$ and absorption.
By the closed value lemma (Lemma C.2) then $\Gamma_2 = \gamma$ and thus $s \cdot \gamma = \gamma$ since multiplication has no effect on runtime reference type assumptions, therefore incoming heap compatibility is $H \bowtie \Gamma_0 + 0 \cdot \Gamma_1 + \gamma$.
Goal 1 (typing) is thus given by:

$$\dfrac{\dfrac{}{0 \cdot \Gamma_1, \mathit{ref} : \mathsf{Ref}_{id}\ A \vdash *\mathit{ref} : *(\mathsf{Ref}_{id}\ A)}\ {*\textsc{ref}^*}}{0 \cdot \Gamma_1, \mathit{ref} : \mathsf{Ref}_{id}\ A \vdash \mathbf{pack}\ \langle id, *\mathit{ref} \rangle : \exists id.\mathsf{Ref}_{id}\ A}\ \textsc{pack}$$

i.e., we set $\Gamma' = 0 \cdot \Gamma_1, \mathit{ref} : \mathsf{Ref}_{id}\ A$ (which notably has runtime typing of ref).
Goal 2 (compatibility) is $(H, \mathit{ref} \mapsto_1 id, id \mapsto v) \bowtie (\Gamma_0 + s \cdot (0 \cdot \Gamma_1, \mathit{ref} : \mathsf{Ref}_{id}\ A))$ which is equal to: $(H, \mathit{ref} \mapsto_1 id, id \mapsto v) \bowtie (\Gamma_0 + 0 \cdot \Gamma_1, \mathit{ref} : \mathsf{Ref}_{id}\ A)$
We construct this goal as follows (leveraging distributivity of $\cdot$ over $+$):

$$\dfrac{H \bowtie \Gamma_0 + 0 \cdot \Gamma_1 + \gamma \qquad \dfrac{\gamma \vdash v : A}{\gamma \vdash \mathbf{ref}(v) : \mathsf{Ref}_{id}\ A}\ \textsc{refStore}}{H, \mathit{ref} \mapsto_1 id, id \mapsto \mathbf{ref}(v) \bowtie (\Gamma_0 + 0 \cdot \Gamma_1, \mathit{ref} : \mathsf{Ref}_{id}\ A)}\ \textsc{extRes}$$

(2) (swapRef)

$$\dfrac{}{H, \mathit{ref} \mapsto_p id, id \mapsto \mathbf{ref}(v) \vdash \mathbf{swapRef}\ (*\mathit{ref})\ v' \leadsto_s H, \mathit{ref} \mapsto_p id, id \mapsto \mathbf{ref}(v') \vdash v}\ \leadsto_{\textsc{swapRef}}$$

Thus typing refines to

$$\dfrac{\begin{array}{c}0 \cdot \Gamma_1 \vdash \mathbf{swapRef} : \&_p(\mathsf{Ref}_{id}\ A) \multimap A \multimap A \otimes \&_p(\mathsf{Ref}_{id}\ A) \\ 0 \cdot \Gamma_2, \mathit{ref} : \mathsf{Ref}_{id}\ A \vdash *\mathit{ref} : *(\mathsf{Ref}_{id}\ A) \qquad \Gamma_3 \vdash v' : A\end{array}}{0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + \Gamma_3, \mathit{ref} : \mathsf{Ref}_{id}\ A \vdash \mathbf{swapRef}\ (*\mathit{ref})\ v' : A \otimes \&_p(\mathsf{Ref}_{id}\ A)}\ \textsc{app}{\times}2$$

i.e. where $p = *$.
By the closed value lemma (Lemma C.2) then $\Gamma_3 = \gamma'$ and thus $s \cdot \gamma' = \gamma'$ since multiplication has no effect on runtime reference type assumptions, therefore incoming heap compatibility is:

$$\dfrac{H \bowtie \Gamma_0 + 0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + \gamma + \gamma' \qquad \dfrac{\gamma \vdash v : A}{\gamma \vdash \mathbf{ref}(v) : \mathsf{Ref}_{id}\ A}\ \textsc{refStore}}{(H, \mathit{ref} \mapsto_p id, id \mapsto \mathbf{ref}(v)) \bowtie (\Gamma_0 + s \cdot (0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + \gamma', \mathit{ref} : \mathsf{Ref}_{id}\ A))}\ \textsc{extRes}$$

Goal 1 (typing) is thus given directly by the heap compatibility's second premise: $\gamma \vdash v : A$
Goal 2 (heap compatibility) is

$$(H, \mathit{ref} \mapsto_p id, id \mapsto \mathbf{ref}(v')) \bowtie (\Gamma_0 + s \cdot \gamma, \mathit{ref} : \mathsf{Ref}_{id}\ A)$$

By absorption then the premise of incoming heap compatibility $H \bowtie \Gamma_0 + 0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + \gamma + \gamma'$ is equal to: $H \bowtie \Gamma_0 + \gamma + \gamma'$, and since $s \cdot \gamma = \gamma$ then we can provide this goal by:

$$\dfrac{H \bowtie \Gamma_0 + s \cdot \gamma + \gamma' \qquad \dfrac{\gamma' \vdash v : A}{\gamma' \vdash \mathbf{ref}(v') : \mathsf{Ref}_{id}\ A}\ \textsc{refStore}}{(H, \mathit{ref} \mapsto_p id, id \mapsto \mathbf{ref}(v')) \bowtie (\Gamma_0 + s \cdot (\gamma, \mathit{ref} : \mathsf{Ref}_{id}\ A))}\ \textsc{extRes}$$

(3) (freezeRef)

$$\frac{}{H, ref \mapsto_p id, id \mapsto \mathbf{ref}(v) \vdash \mathbf{freezeRef}\,(*ref) \rightsquigarrow_s H \vdash v} \rightsquigarrow_{\textsc{freezeRef}}$$

Thus typing refines to

$$\frac{0 \cdot \Gamma_1 \vdash \mathbf{freezeRef} : \forall\, id.*(\mathsf{Ref}_{id}\ A) \multimap A \qquad 0 \cdot \Gamma_2, ref : \mathsf{Ref}_{id}\ A \vdash *ref : *(\mathsf{Ref}_{id}\ A)}{0 \cdot \Gamma_1 + 0 \cdot \Gamma_2, ref : \mathsf{Ref}_{id}\ A \vdash \mathbf{freezeRef}\,(*ref) : A}\ \textsc{app}$$

and we have incoming heap compatibility:

$$\frac{H \bowtie \Gamma_0 + 0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + \gamma \qquad \dfrac{\gamma \vdash v : A}{\gamma \vdash \mathbf{ref}(v) : \mathsf{Ref}_{id}\ A}\ \textsc{refStore}}{(H, a \mapsto_p id, id \mapsto v) \bowtie (\Gamma_0 + 0 \cdot \Gamma_1 + 0 \cdot \Gamma_2, ref : \mathsf{Ref}_{id}\ A)}\ \textsc{extRes}$$

Goal 1 (typing) is thus given directly by the second premise of heap compatibility: $\gamma \vdash v : A$
Goal 2 (heap compatibility) is then $H \bowtie (\Gamma_0 + s \cdot \gamma)$ given by the premise of incoming heap compatibility since $\Gamma_0 + 0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 = \Gamma_0$ and $s \cdot \gamma = \gamma$:

$$\begin{aligned} & H \bowtie \Gamma_0 + 0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + \gamma \\ \equiv\ & H \bowtie \Gamma_0 + s \cdot \gamma \end{aligned}$$

(4) (readRef)

$$\frac{}{H, ref \mapsto_p id, id \mapsto \mathbf{ref}([v]_{r+1}) \vdash \mathbf{readRef}\,(*ref) \rightsquigarrow_s H, ref \mapsto_p id, id \mapsto \mathbf{ref}([v]_r) \vdash (v, *ref)} \rightsquigarrow_{\textsc{readRef}}$$

Thus typing refines to

$$\frac{\begin{array}{c} 0 \cdot \Gamma_1 \vdash \mathbf{readRef} : \&_p(\mathsf{Ref}_{id}\ \Box_{r+1} A) \multimap A \otimes \&_p(\mathsf{Ref}_{id}\ \Box_r A) \\ 0 \cdot \Gamma_2, ref : \mathsf{Ref}_{id}\ \Box_{r+1} A \vdash *ref : *(\mathsf{Ref}_{id}\ \Box_{r+1} A) \end{array}}{0 \cdot \Gamma_1 + 0 \cdot \Gamma_2, ref : \mathsf{Ref}_{id}\ A \vdash \mathbf{readRef}\,(*ref) : A \otimes \&_p(\mathsf{Ref}_{id}\ \Box_r A)}\ \textsc{app}$$

i.e. where $p = *$ and we have incoming heap compatibility:

$$\frac{H \bowtie \Gamma_0 + 0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + \gamma \qquad \dfrac{\dfrac{\gamma \vdash v : A}{\gamma \vdash [v]_{r+1} : \Box_{r+1} A}\ \textsc{pr}}{\gamma \vdash \mathbf{ref}([v]_{r+1}) : \mathsf{Ref}_{id}\ (\Box_{r+1} A)}\ \textsc{refStore}}{(H, ref \mapsto_p id, id \mapsto \mathbf{ref}([v]_{r+1})) \bowtie (\Gamma_0 + s \cdot (0 \cdot \Gamma_1 + 0 \cdot \Gamma_2, ref : \mathsf{Ref}_{id}\ A))}\ \textsc{extRes}$$

where recall $r + 1 \cdot \gamma = \gamma$ and thus compatibility is which is equal to $(H, ref \mapsto_p id, id \mapsto \mathbf{ref}([v]_{r+1})) \bowtie (\Gamma_0 + 0 \cdot \Gamma_1 + 0 \cdot \Gamma_2, ref : \mathsf{Ref}_{id}\ A)$ by absorption.
Goal 1 (typing) is thus given by:

$$\frac{\gamma \vdash v : A \qquad 0 \cdot \Gamma_2, ref : \mathsf{Ref}_{id}\ \Box_r A \vdash *ref : *(\mathsf{Ref}_{id}\ \Box_r A)}{\gamma + 0 \cdot \Gamma_2, ref : \mathsf{Ref}_{id}\ A \vdash (v, *ref) : A \otimes \&_p(\mathsf{Ref}_{id}\ \Box_r A)}\ \otimes_I$$

Goal 2 (heap compatibility) is

$$(H, ref \mapsto_p id, id \mapsto \mathbf{ref}([v]_r)) \bowtie (\Gamma_0 + 0 \cdot (\Gamma_1 + \Gamma_2), ref : \mathsf{Ref}_{id}\ A)$$

which is provided exactly by the incoming heap compatibility but re-deriving promotion at the grade r.

(5) (newArray)

$$\frac{ref \# H \qquad id \# H}{H \vdash \mathbf{newArray}\ n \rightsquigarrow_s H, ref \mapsto_1 id, id \mapsto \mathsf{init} \vdash \mathbf{pack}\ \langle id, *ref \rangle}\ \rightsquigarrow_{\text{NEWARRAY}}$$

Thus typing refines to:

$$\frac{0 \cdot \Gamma_1 \vdash \mathbf{newArray} : \mathbb{N} \multimap \exists id.*(\mathsf{Array}_{id}\ \mathbb{F}) \qquad 0 \cdot \Gamma_2 \vdash n : \mathbb{N}}{0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 \vdash \mathbf{newArray}\ n : \exists id.*(\mathsf{Array}_{id}\ \mathbb{F})}\ \text{APP}$$

with incoming heap compatibility $H \bowtie \Gamma_0 + s \cdot (0 \cdot \Gamma_1 + 0 \cdot \Gamma_2)$ which is equal to $H \bowtie \Gamma_0 + 0 \cdot \Gamma_1 + 0 \cdot \Gamma_2$ by distributivity of $*$ over $+$ and absorption.

Goal 1 (typing) is thus given by:

$$\frac{\dfrac{}{0 \cdot (\Gamma_1 + \Gamma_2), a : \mathsf{Array}_{id}\ \mathbb{F} \vdash *a : *(\mathsf{Array}_{id}\ \mathbb{F})}\ *\text{REF}^*}{0 \cdot (\Gamma_1 + \Gamma_2), a : \mathsf{Array}_{id}\ A \vdash \mathbf{pack}\ \langle id, *a \rangle : \exists id.\mathsf{Array}_{id}\ \mathbb{F}}\ \text{PACK}$$

i.e., we set $\Gamma' = 0 \cdot (\Gamma_1 + \Gamma_2), a : \mathsf{Array}_{id}\ A$ (which notably has runtime typing of a).

Goal 2 (compatibility) is $(H, a \mapsto_1 id, id \mapsto \mathsf{init}) \bowtie (\Gamma_0 + s \cdot (0 \cdot (\Gamma_1 + \Gamma_2), a : \mathsf{Array}_{id}\ A))$

We construct this goal as follows (leveraging distributivity of $\cdot$ over $+$ and absorption and since $s \cdot \gamma = \gamma$):

$$\frac{H \bowtie \Gamma_0 + 0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 \qquad \dfrac{}{\emptyset \vdash \mathsf{init} : \mathsf{Array}_{id}\ \mathbb{F}}\ \text{ARRAYINIT}}{H, a \mapsto_1 id, id \mapsto \mathsf{init} \bowtie (\Gamma_0 + 0 \cdot (\Gamma_1 + \Gamma_2), a : \mathsf{Array}_{id}\ A)}\ \text{EXTRES}$$

(6) (readArray)

$$\frac{}{H, ref \mapsto_p id, id \mapsto \mathbf{arr}[i] = v \vdash \mathbf{readArray}\ (*ref)\ i \rightsquigarrow_s H, ref \mapsto_p id, id \mapsto \mathbf{arr}[i] = v \vdash (v, *ref)}\ \rightsquigarrow_{\text{READARRAY}}$$

Thus typing refines to

$$\frac{\begin{array}{c} 0 \cdot \Gamma_1 \vdash \mathbf{readArray} : \&_p(\mathsf{Array}_{id}\ \mathbb{F}) \multimap \mathbb{N} \multimap \mathbb{F} \otimes \&_p(\mathsf{Array}_{id}\ \mathbb{F}) \\ 0 \cdot \Gamma_2, a : \mathsf{Array}_{id}\ \mathbb{F} \vdash *a : *(\mathsf{Array}_{id}\ A) \qquad 0 \cdot \Gamma_3 \vdash i : \mathbb{N} \end{array}}{0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + 0 \cdot \Gamma_3, a : \mathsf{Array}_{id}\ \mathbb{F} \vdash \mathbf{readArray}\ (*a)\ i : \mathbb{F} \otimes \&_p(\mathsf{Array}_{id}\ \mathbb{F})}\ \text{APP}\times 2$$

i.e. where $p = *$ and we have incoming heap compatibility (simplified below by absorption since $(\Gamma_0 + s \cdot (0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + 0 \cdot \Gamma_3, a : \mathsf{Array}_{id}\ \mathbb{F})) = (\Gamma_0 + 0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + 0 \cdot \Gamma_3, a : \mathsf{Array}_{id}\ \mathbb{F})$):

$$\frac{H \bowtie \Gamma_0 + 0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + 0 \cdot \Gamma_3 \qquad \dfrac{\emptyset \vdash \mathbf{arr} : \mathsf{Array}_{id}\ \mathbb{F} \qquad \emptyset \vdash v : \mathbb{F}}{\emptyset \vdash \mathbf{arr}[i] = v : \mathsf{Array}_{id}\ \mathbb{F}}\ \text{ARRAYAT}}{(H, a \mapsto_p id, id \mapsto \mathbf{arr}[i] = v) \bowtie (\Gamma_0 + 0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + 0 \cdot \Gamma_3, a : \mathsf{Array}_{id}\ \mathbb{F})}\ \text{EXTRES}$$

Goal 1 (typing) is thus given by:

$$\frac{\emptyset \vdash v : \mathbb{F} \qquad 0 \cdot (\Gamma_1 + \Gamma_2 + \Gamma_3), a : \mathsf{Array}_{id}\ \mathbb{F} \vdash *a : \&_p(\mathsf{Array}_{id}\ \mathbb{F})}{0 \cdot (\Gamma_1 + \Gamma_2 + \Gamma_3), a : \mathsf{Array}_{id}\ \mathbb{F} \vdash (v, *a) : \mathbb{F} \otimes \&_p(\mathsf{Array}_{id}\ \mathbb{F})}\ \otimes_I$$

i.e. $\Gamma' = 0 \cdot (\Gamma_1 + \Gamma_2 + \Gamma_3), a : \mathsf{Array}_{id}\ \mathbb{F}$.

Goal 2 (heap compatibility) is

$$(H, a \mapsto_p id, id \mapsto \mathbf{arr}[i] = v) \bowtie (\Gamma_0 + s \cdot 0 \cdot (\Gamma_1 + \Gamma_2 + \Gamma_3), a : \mathsf{Array}_{id}\ \mathbb{F})$$

which is provided exactly by the incoming heap compatibility.

(7) (writeArray)

$$\frac{}{H, \mathit{ref} \mapsto_p \mathit{id}, \mathit{id} \mapsto \mathbf{arr} \vdash \mathbf{writeArray}\ (*\mathit{ref})\ i\ v \leadsto_s H, \mathit{ref} \mapsto_p \mathit{id}, \mathit{id} \mapsto \mathbf{arr}[i] = v \vdash *\mathit{ref}} \leadsto_{\text{WRITEARRAY}}$$

Thus typing refines to

$$\frac{\begin{array}{c}0 \cdot \Gamma_1 \vdash \mathbf{writeArray} : \&_p(\mathsf{Array}_{id}\ \mathbb{F}) \multimap \mathbb{N} \multimap \mathbb{F} \otimes \&_p(\mathsf{Array}_{id}\ \mathbb{F}) \\ 0 \cdot \Gamma_2, a : \mathsf{Array}_{id}\ \mathbb{F} \vdash *a : \&_p(\mathsf{Array}_{id}\ A) \quad 0 \cdot \Gamma_3 \vdash n : \mathbb{N} \quad 0 \cdot \Gamma_4 \vdash v : \mathbb{F}\end{array}}{0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + 0 \cdot \Gamma_3 + 0 \cdot \Gamma_4, a : \mathsf{Array}_{id}\ \mathbb{F} \vdash \mathbf{writeArray}\ (*a)\ n\ v : \&_p(\mathsf{Array}_{id}\ \mathbb{F})} \text{APP}{\times}3$$

and we have incoming heap compatibility (simplified by absorption as : $(\Gamma_0 + 0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + 0 \cdot \Gamma_3 + 0 \cdot \Gamma_4, a : \mathsf{Array}_{id}\ \mathbb{F}) = (\Gamma_0 + s \cdot (0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + 0 \cdot \Gamma_3 + 0 \cdot \Gamma_4, a : \mathsf{Array}_{id}\ \mathbb{F}))$):

$$\frac{H \bowtie \Gamma_0 + 0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + 0 \cdot \Gamma_3 + 0 \cdot \Gamma_4 \quad \emptyset \vdash \mathbf{arr} : \mathsf{Array}_{id}\ \mathbb{F}}{(H, a \mapsto_p \mathit{id}, \mathit{id} \mapsto \mathbf{arr}) \bowtie (\Gamma_0 + 0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + 0 \cdot \Gamma_3 + 0 \cdot \Gamma_4, a : \mathsf{Array}_{id}\ \mathbb{F})} \text{EXTRES}$$

Goal 1 (typing) is thus given by:

$$\frac{0 \cdot \Gamma_2, a : \mathsf{Array}_{id}\ \mathbb{F} \vdash (*a) : \&_p(\mathsf{Array}_{id}\ A)}{0 \cdot (\Gamma_1 + \Gamma_2 + \Gamma_3), a : \mathsf{Array}_{id}\ \mathbb{F} \vdash (*a) : \&_p(\mathsf{Array}_{id}\ \mathbb{F})} \text{LEMMA C.5}$$

i.e. $\Gamma' = 0 \cdot (\Gamma_1 + \Gamma_2 + \Gamma_3), a : \mathsf{Array}_{id}\ \mathbb{F}$ and where we can strengthen $0 \cdot \Gamma_4 \vdash v : \mathbb{F}$ to $\emptyset \vdash v : \mathbb{F}$ by inversion on float values.

Goal 2 (heap compatibility) is

$$(H, a \mapsto_p \mathit{id}, \mathit{id} \mapsto \mathbf{arr}[i] = v) \bowtie (\Gamma_0 + s \cdot 0 \cdot (\Gamma_1 + \Gamma_2 + \Gamma_3), a : \mathsf{Array}_{id}\ \mathbb{F})$$

which is provided exactly by the incoming heap compatibility:

$$\frac{H \bowtie \Gamma_0 + 0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + 0 \cdot \Gamma_3 + 0 \cdot \Gamma_4 \quad \dfrac{\emptyset \vdash \mathbf{arr} : \mathsf{Array}_{id}\ \mathbb{F} \quad \emptyset \vdash v : \mathbb{F}}{\emptyset \vdash \mathbf{arr}[i] = v : \mathsf{Array}_{id}\ \mathbb{F}} \text{ARRAYAT}}{(H, a \mapsto_p \mathit{id}, \mathit{id} \mapsto \mathbf{arr}) \bowtie (\Gamma_0 + 0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + 0 \cdot \Gamma_3, a : \mathsf{Array}_{id}\ \mathbb{F})} \text{EXTRES}$$

where $(\Gamma_0 + s \cdot 0 \cdot (\Gamma_1 + \Gamma_2 + \Gamma_3), a : \mathsf{Array}_{id}\ \mathbb{F}) = (\Gamma_0 + 0 \cdot \Gamma_1 + 0 \cdot \Gamma_2 + 0 \cdot \Gamma_3, a : \mathsf{Array}_{id}\ \mathbb{F})$.

(8) (deleteArray)

$$\frac{}{H, \mathit{ref} \mapsto_p \mathit{id}, \mathit{id} \mapsto \mathbf{arr} \vdash \mathbf{deleteArray}\ (*\mathit{ref}) \leadsto_s H \vdash ()} \leadsto_{\text{DELETEARRAY}}$$

Thus typing refines to

$$\frac{\begin{array}{c}0 \cdot \Gamma_1 \vdash \mathbf{deleteArray} : \forall\, \mathit{id}.*(\mathsf{Array}_{id}\ \mathbb{F}) \multimap \mathsf{unit} \\ 0 \cdot \Gamma_2, a : \mathsf{Array}_{id}\ \mathbb{F} \vdash *a : *(\mathsf{Array}_{id}\ A)\end{array}}{0 \cdot \Gamma_1 + 0 \cdot \Gamma_2, a : \mathsf{Array}_{id}\ \mathbb{F} \vdash \mathbf{deleteArray}\ (*a) : \mathsf{unit}} \text{APP}$$

and we have incoming heap compatibility:

$$\frac{H \bowtie \Gamma_0 + s \cdot (0 \cdot \Gamma_1 + 0 \cdot \Gamma_2) \quad \emptyset \vdash \mathbf{arr} : \mathsf{Array}_{id}\ \mathbb{F}}{(H, a \mapsto_p \mathit{id}, \mathit{id} \mapsto \mathbf{arr}) \bowtie (\Gamma_0 + s \cdot (0 \cdot \Gamma_1 + 0 \cdot \Gamma_2), a : \mathsf{Array}_{id}\ \mathbb{F})} \text{EXTRES}$$

Goal 1 (typing) is thus given by:

$$\frac{}{0 \cdot (\Gamma_1 + \Gamma_2) \vdash () : \mathsf{unit}} 1_I$$

with $\Gamma' = 0 \cdot (\Gamma_1 + \Gamma_2)$

Goal 2 (heap compatibility) is $H \bowtie (\Gamma_0 + s * 0 \cdot (\Gamma_1 + \Gamma_2))$ given by the premise of incoming heap compatibility.

- (boxElim)

$$\frac{\Gamma_1 \vdash t_1 : \Box_r A \qquad \Gamma_2, x : [A]_r \vdash t_2 : B}{\Gamma_1 + \Gamma_2 \vdash \textbf{let}\,[x] = t_1\,\textbf{in}\,t_2 : B}\ \textsc{elim}$$

And two possible reductions:

(1) (betaBoxElim)

$$\frac{y\#\{H, v, t\}}{H \vdash \textbf{let}\,[x] = [v]_r\,\textbf{in}\,t \rightsquigarrow_s H, y \mapsto_{(s*r)} v \vdash t[y/x]}\ \rightsquigarrow_{\Box\beta}$$

refining the typing to:

$$\frac{\dfrac{[\Gamma_1'] \vdash v : A}{r \cdot \Gamma_1' \vdash [v]_r : \Box_r A}\ \textsc{pr} \qquad \Gamma_2, x : [A]_r \vdash t : B}{r \cdot \Gamma_1' + \Gamma_2 \vdash \textbf{let}\,[x] = [v]_r\,\textbf{in}\,t : B}\ \textsc{elim}$$

with incoming heap compatibility:

$$H \bowtie \Gamma_0 + s \cdot (r \cdot \Gamma_1' + \Gamma_2)$$

Goal 1: typing Provided by the premise here as:

$$\Gamma_2, x : [A]_r \vdash t : A$$

under renaming to:

$$\Gamma_2, y : [A]_r \vdash t[y/x] : A$$

i.e., we set the goal $\Gamma' = \Gamma_2, y : [A]_r$.

Goal 2: heap compatibility is $(H, y \mapsto_{s*r} v) \bowtie (\Gamma_0 + s \cdot (\Gamma_2, y : [A]_r))$ which refines to: $(H, y \mapsto_{s*r} v) \bowtie (\Gamma_0 + s \cdot \Gamma_2), y : [A]_{s*r}$ by the disjointness of x

We construct this goal via the premise heap compatibility by

$$\frac{H \bowtie \Gamma_0 + s * r \cdot \Gamma_1' + s \cdot \Gamma_2}{(H, y \mapsto_{s*r} v) \bowtie (\Gamma_0 + s \cdot \Gamma_2), y : [A]_{s*r}}\ \textsc{ext}$$

(2) (congBoxElim)

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash \textbf{let}\,[x] = t_1\,\textbf{in}\,t_2 \rightsquigarrow_s H' \vdash \textbf{let}\,[x] = t_1'\,\textbf{in}\,t_2}\ \rightsquigarrow_{\textsc{let}\Box}$$

with incoming heap compatibility:

$$H \bowtie \Gamma_0' + \Gamma_1 + \Gamma_2$$

Applying induction on the premise reduction with heap compatibility given by the incoming heap compatibility but with $\Gamma_0 = \Gamma_0' + \Gamma_2$ then yields:

(a) $\exists \Gamma_1'.\ \Gamma_1' \vdash t_1' : \Box_r A$

(b) $H' \bowtie (\Gamma_0' + s \cdot \Gamma_2 + s \cdot \Gamma_1')$

(Goal 1) is then given by the reconstructed application type $\Gamma_1' + \Gamma_2 \vdash \textbf{let}\,[x] = t_1'\,\textbf{in}\,t_2 : B$

(Goal 2) is $H' \bowtie (\Gamma_0' + s \cdot (\Gamma_1' + \Gamma_2))$ which is given by the second conjunct above by commutativity of $+$ and distributivity of $*$ over $+$.

- (boxIntro)

$$\frac{\Gamma \vdash t : A \qquad \neg\mathsf{resourceAllocator}(t)}{r \cdot \Gamma \vdash [t]_r : \Box_r A}\ \textsc{pr}$$

and one possible reduction

$$\frac{H \vdash t \rightsquigarrow_{s*r} H' \vdash t'}{H \vdash [t]_r \rightsquigarrow_s H' \vdash [t']_r} \rightsquigarrow_\Box$$

Aside: Here we see the need for the Church-style graded annotation here in the term as otherwise we could have a reduction derivation that uses a different grade in its premise:

$$\frac{H \vdash t \rightsquigarrow_{s*r'} H' \vdash t'}{H \vdash [t] \rightsquigarrow_s H' \vdash [t']} \rightsquigarrow_\Box\text{-ALTERNATE-NONCHURCH}$$

In which case we could not induct on the first premise as we would not have the required heap compatibility $H \bowtie \Gamma_0 + s' * r' \cdot \Gamma$. Thus this motivates the need for the Church style typing here.
Thus, instead we have the incoming heap compatibility:

$$H \bowtie \Gamma_0' + s' * r \cdot \Gamma$$

Applying induction on the premise reduction with $\Gamma_0 = \Gamma_0'$ and $s = s' * r$ then yields:
(1) $\exists \Gamma_1'. \Gamma_1' \vdash t_1' : A$
(2) $H' \bowtie (\Gamma_0' + (s' * r) \cdot \Gamma_1')$
(Goal 1) is then given by the reconstructed application type $r \cdot \Gamma_1' \vdash [t_1'] : \Box_r A$

$$\frac{\Gamma_1' \vdash t_1' : A}{r \cdot \Gamma_1' \vdash [t_1'] : \Box_r A} \rightsquigarrow_\Box$$

(Goal 2) is $H' \bowtie (\Gamma_0' + s' * r \cdot \Gamma_1')$ which is given by the second conjunct above by commutativity of + and associativity of $*$.

- (der)

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma, x : [A]_1 \vdash t : B} \text{DER}$$

with incoming heap compatibility $H \bowtie \Gamma_0 + s \cdot (\Gamma, x : [A]_1)$ which refines to $H \bowtie \Gamma_0 + s \cdot \Gamma, x : [A]_s$ and a reduction:

$$H \vdash t \rightsquigarrow_s H' \vdash t'$$

Induction requires $H \bowtie \Gamma_0 + s \cdot (\Gamma, x : A)$ which by the definition of scalar multiplication is just $H \bowtie \Gamma_0 + s \cdot \Gamma, x : [A]_s$, thus we can apply induction and get the result of $\Gamma' \vdash t' : B$ and $H' \bowtie \Gamma_0 + s \cdot \Gamma_1'$ satisfying the goal here.

- (tensorIntro)

$$\frac{\Gamma_1 \vdash t_1 : A \qquad \Gamma_2 \vdash t_2 : B}{\Gamma_1 + \Gamma_2 \vdash (t_1, t_2) : A \otimes B} \otimes_I$$

with incoming heap compatibility $H \bowtie \Gamma_0 + s \cdot (\Gamma_1 + \Gamma_2)$.
And two possible reductions:
(1) (congPairL)

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash (t_1, t_2) \rightsquigarrow_s H' \vdash (t_1', t_2)} \rightsquigarrow_{\otimes\text{L}}$$

By induction on the premise with $\Gamma_0' = \Gamma_0 + s \cdot \Gamma_2$ then we have: (i) $\Gamma_1' \vdash t_1' : A$ and (ii) $H' \bowtie \Gamma_0 + s \cdot \Gamma_2 + s \cdot \Gamma_1'$. From which we construct the resulting typing, via applying $\otimes_I$ again to get $\Gamma_1' + \Gamma_2 \vdash (t_1', t_2) : A \otimes B$ and with (ii) providing the required heap compatibility (by commutativity of +).

(2) (congPairR)

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash (v, t_2) \rightsquigarrow_s H' \vdash (v, t_2')} \rightsquigarrow_{\otimes\mathrm{R}}$$

Essentially the same as the preceding case (congPairL) but by induction on the second premise.

- (tensorElim)

$$\frac{\Gamma_1 \vdash t_1 : A \otimes B \qquad \Gamma_2, x : A, y : B \vdash t_2 : C}{\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\,(x, y) = t_1 \,\mathbf{in}\, t_2 : C} \otimes_E$$

with incoming heap compatibility $H \bowtie \Gamma_0 + s \cdot (\Gamma_1 + \Gamma_2)$.
And two possible reductions:

(1) (pairBeta)

$$\frac{x'\#\{H, v_1, v_2, t\} \qquad y'\#\{H, v_1, v_2, t\}}{H \vdash \mathbf{let}\,(x, y) = (v_1, v_2)\,\mathbf{in}\, t \rightsquigarrow_s H, x' \mapsto_s v_1, y' \mapsto_s v_2 \vdash t[y'/y][x'/x]} \rightsquigarrow_{\otimes\beta}$$

with typing $\Gamma_2, x : A, y : B \vdash t_2 : C$ as the result i.e., with $\Gamma' = \Gamma_2, x : A, y : B$ and outgoing heap compatibility

(2) (pairElim)

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash \mathbf{let}\,(x, y) = t_1 \,\mathbf{in}\, t_2 \rightsquigarrow_s H' \vdash \mathbf{let}\,(x, y) = t_1' \,\mathbf{in}\, t_2} \rightsquigarrow_{\mathrm{LET}\otimes}$$

By induction on the premise with $\Gamma_0' = \Gamma_0 + s \cdot \Gamma_2$ then we have: (i) $\Gamma_1' \vdash t_1' : A \otimes B$ and (ii) $H' \bowtie \Gamma_0 + s \cdot \Gamma_2 + s \cdot \Gamma_1'$. From which we construct the resulting typing, via applying $\otimes_E$ again to get $\Gamma_1' + \Gamma_2 \vdash \mathbf{let}\,(x, y) = t_1' \,\mathbf{in}\, t_2 : C$ and with (ii) providing the required heap compatibility (by commutativity of $+$).

- (unitIntro)

$$\frac{}{0 \cdot \Gamma \vdash () : \mathsf{unit}} 1_I$$

Trivial case since there is no heap semantics rule as this is already a normal form value.

- (unitElim)

$$\frac{\Gamma_1 \vdash t_1 : \mathsf{unit} \qquad \Gamma_2 \vdash t_2 : B}{\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\ () = t_1 \,\mathbf{in}\, t_2 : B} 1_E$$

And two possible reductions:

(1) (betaUnit)

$$\frac{}{H \vdash \mathbf{let}\ () = ()\ \mathbf{in}\ t \rightsquigarrow_s H \vdash t} \rightsquigarrow_{\beta\mathrm{UNIT}}$$

which refines the typing to:

$$\frac{0 \cdot \Gamma_1 \vdash () : \mathsf{unit} \quad \Gamma_2 \vdash t_2 : B}{0 \cdot \Gamma_1 + \Gamma_2 \vdash \mathbf{let}\ () = ()\ \mathbf{in}\ t_2 : B} 1_E$$

with incoming heap compatibility $H \bowtie \Gamma_0 + s \cdot (0 \cdot \Gamma_1 + \Gamma_2)$ which refines to $H \bowtie \Gamma_0 + 0 \cdot \Gamma_1 + s \cdot \Gamma_2$. The resulting typing is thus given by:

$$\frac{\Gamma_2 \vdash t_2 : B}{0 \cdot \Gamma_1 + \Gamma_2 \vdash t_2 : B} \text{LEMMA C.5}$$

i.e., $\Gamma' = 0 \cdot \Gamma_1 + \Gamma_2$ and outgoing heap compatibility is provided exactly by the incoming heap compatibility.

(2) (congUnitElim)

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash \textbf{let}\ () = t_1\ \textbf{in}\ t_2 \rightsquigarrow_s H' \vdash \textbf{let}\ () = t_1'\ \textbf{in}\ t_2} \rightsquigarrow_{\text{LETUNIT}}$$

By induction on the premise with $\Gamma_0' = \Gamma_0 + s \cdot \Gamma_2$ then we have: (i) $\Gamma_1' \vdash t_1' : \mathsf{unit}$ and (ii) $H' \bowtie \Gamma_0 + s \cdot \Gamma_2 + s \cdot \Gamma_1'$. From which we construct the resulting typing, via applying 1_E again to get $\Gamma_1' + \Gamma_2 \vdash \textbf{let}\ () = t_1'\ \textbf{in}\ t_2 : B$ and with (ii) providing the required heap compatibility (by commutativity of $+$).

- (share)

$$\frac{\Gamma \vdash t : *A}{\Gamma \vdash \textbf{share}\ t : \Box_r A}\ \text{SHARE}$$

And two possible reductions:

(1) (share)

$$\frac{\mathsf{dom}(H) \equiv \mathsf{refs}(v)}{H, H' \vdash \textbf{share}\ (*v) \rightsquigarrow_s ([H]_0), H' \vdash [v]} \rightsquigarrow_{\text{SHARE}\beta}$$

with refined (runtime) typing, and by Lemma C.1,

$$\dfrac{\dfrac{0 \cdot \Gamma_1', \gamma \vdash v : A}{0 \cdot (\Gamma_1', \Gamma_1''), \gamma \vdash *v : *A}\ \text{NEC}}{0 \cdot (\Gamma_1', \Gamma_1''), \gamma \vdash \textbf{share}\ (*v) : \Box_r A}\ \text{SHARE}$$

and thus incoming heap compatibility $H, H' \bowtie \Gamma_0 + s \cdot (0 \cdot (\Gamma_1', \Gamma_1''), \gamma)$ which refines to: $H, H' \bowtie \Gamma_0 + 0 \cdot (\Gamma_1', \Gamma_1''), \gamma$.
The resulting type is given by:

$$\dfrac{\dfrac{0 \cdot \Gamma_1', \gamma \vdash v : A}{0 \cdot (\Gamma_1', \Gamma_1''), \gamma \vdash v : A}\ \text{Lemma C.5}}{r \cdot (0 \cdot (\Gamma_1', \Gamma_1''), \gamma) \vdash [v] : \Box_r A}\ \text{PR}$$

where $r \cdot (0 \cdot (\Gamma_1', \Gamma_1''), \gamma) = 0 \cdot (\Gamma_1', \Gamma_1''), \gamma$ and thus $\Gamma' = 0 \cdot (\Gamma_1', \Gamma_1''), \gamma$ and the outgoing heap compatibility is provided by the incoming heap compatibility but where the heap has zeroed out H (which does not affect the heap compatibility; it can be re-derived with different fractions).

(2) (congShare)

$$\frac{H \vdash t \rightsquigarrow_s H' \vdash t'}{H \vdash \textbf{share}\ t \rightsquigarrow_s H' \vdash \textbf{share}\ t'} \rightsquigarrow_{\text{SHARE}}$$

By induction on the premise with $\Gamma_0' = \Gamma_0$ then we have: (i) $\Gamma' \vdash t' : *A$ and (ii) $H' \bowtie \Gamma_0 + s \cdot \Gamma'$. From which we construct the resulting typing, via applying SHARE again to get $\Gamma' \vdash \textbf{share}\ t' : \Box_r A$ and with (ii) providing the required heap compatibility.

- (clone)

$$\frac{\Gamma_1 \vdash t_1 : \Box_r A \qquad \Gamma_2, x : *A \vdash t_2 : \Box_r B \qquad 1 \sqsubseteq r}{\Gamma_1 + \Gamma_2 \vdash \textbf{clone'}\ t_1\ \textbf{as}\ x\ \textbf{in}\ t_2 : \Box_r B}\ \text{CLONE'}$$

And two possible reductions:

(1) (cloneCongr)

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash \textbf{clone } t_1 \textbf{ as } x \textbf{ in } t_2 \rightsquigarrow_s H' \vdash \textbf{clone } t_1' \textbf{ as } x \textbf{ in } t_2} \rightsquigarrow_{\text{CLONE}}$$

Follows by induction similar to other congruences.

(2) (cloneBeta)

$$\frac{\text{dom}(H') \equiv \text{refs}(v) \qquad (H'', \theta, \overline{id}) = \text{copy}(H') \qquad y\#\{H, v, t\}}{H, H' \vdash \textbf{clone } [v]_r \textbf{ as } x \textbf{ in } t \rightsquigarrow_s H, H', H'', y \mapsto_s \textbf{pack } \langle \overline{id}, *(\theta(v)) \rangle \vdash t[y/x]} \rightsquigarrow_{\text{CLONE}\beta}$$

refining the typing to:

$$\frac{\dfrac{\Gamma_1, \overline{id_1} \vdash v : A}{r \cdot \Gamma_1, \overline{id_1} \vdash [v]_r : \Box_r A}\ \text{PR} \qquad \Gamma_2, x : \exists \overline{id'}.*(A[\overline{id'}/\overline{id_1}]) \vdash t : \Box_r B \qquad 1 \sqsubseteq r}{(r \cdot \Gamma_1 + \Gamma_2), \overline{id} \vdash \textbf{clone } [v]_r \textbf{ as } x \textbf{ in } t : \Box_r B}\ \text{CLONE'}$$

with incoming heap compatibility $H, H' \bowtie \Gamma_0 + s \cdot (r \cdot \Gamma_1 + \Gamma_2)$.
Goal typing is then given by:

$$\Gamma_2, x : \exists \overline{id'}.*(A[\overline{id'}/\overline{id_1}]) \vdash t : \Box_r B$$

thus with $\Gamma' = \Gamma_2, x : \exists \overline{id'}.*(A[\overline{id'}/\overline{id_1}])$.
The typing of the extended heap is given by the value:

$$\frac{\dfrac{\dfrac{\Gamma_1, \overline{id_1} \vdash v : A}{\theta(\Gamma_1, \overline{id_1}) \vdash \theta(v) : \theta(A)}\ \text{Lemma C.6}}{\theta(\Gamma_1, \overline{id_1}) \vdash *(\theta(v)) : *\theta(A)}\ \text{NEC}}{\theta(\Gamma_1), \overline{id} \vdash \textbf{pack } \langle \overline{id}, *(\theta(v)) \rangle : \exists \overline{id'}.*\theta(A)[\overline{id'}/\overline{id}]}\ \text{PACK} \tag{1}$$

since θ maps each $\overline{id_1}$ to $\overline{id}$ then $\exists \overline{id'}.*\theta(A)[\overline{id'}/\overline{id}] = \exists \overline{id'}.*A[\overline{id'}/\overline{id_1}]$.
Goal heap compatibility: $(H, H', H'', x \mapsto_s *(\theta(v))) \bowtie (\Gamma_0 + s \cdot (\Gamma_2, x : *(\#A)))$ given by:

$$\frac{\dfrac{\dfrac{H, H' \bowtie (\Gamma_0 + s \cdot \Gamma_2 + r * s \cdot \Gamma_1)}{H, H', H'' \bowtie (\Gamma_0 + s \cdot \Gamma_2 + r * s \cdot \theta(\Gamma_1))} \quad 1 \sqsubseteq r}{H, H', H'' \bowtie (\Gamma_0 + s \cdot \Gamma_2 + s \cdot \theta(\Gamma_1))} \qquad x \notin \text{dom}(H) \quad (1) \qquad \exists r'.\, r + r' \equiv s}{(H, H', H'', x \mapsto_s \textbf{pack } \langle \overline{id}, *(\theta(v)) \rangle) \bowtie (\Gamma_0 + s \cdot \Gamma_2, x : [\exists \overline{id'}.*(A[\overline{id'}/\overline{id_1}])]_s)}\ \text{EXT}$$

- (withBorrow)

$$\frac{\Gamma_1 \vdash t_1 : *A \qquad \Gamma_2 \vdash t_2 : \&_1 A \multimap \&_1 B}{\Gamma_1 + \Gamma_2 \vdash \textbf{withBorrow } t_1\ t_2 : *B}\ \text{WITH}\&$$

And three possible reductions:

(1) (congWithBorrowL)

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash \textbf{withBorrow } t_1\ t_2 \rightsquigarrow_s H' \vdash \textbf{withBorrow } t_1'\ t_2} \rightsquigarrow_{\text{WITH\&L}}$$

Which follows by induction similar to other inductive cases.

(2) (congWithBorrowR)

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \mathbf{withBorrow}\ (\lambda x.t_1)\ t_2 \rightsquigarrow_s H' \vdash \mathbf{withBorrow}\ (\lambda x.t_1)\ t_2'} \rightsquigarrow_{\text{WITH\&R}}$$

Which follows by induction similar to other inductive cases.

(3) (withBorrow)

$$\frac{y\#\{H, v, t\}}{H \vdash \mathbf{withBorrow}\ (\lambda x.t)\ (*v) \rightsquigarrow_s H, y \mapsto_s (*v) \vdash \mathbf{unborrow}\ t[y/x]} \rightsquigarrow_{\text{WITH\&}}$$

which refines the typing to:

$$\frac{\dfrac{\Gamma_2, x : \&_1 A \vdash t : \&_1 B}{\Gamma_2 \vdash \lambda x.t : \&_1 A \multimap \&_1 B}\ \text{WITH\&} \qquad \dfrac{\Gamma_1 \vdash v : A}{\Gamma_1 \vdash *v : *A}\ \text{NEC}}{\Gamma_1 + \Gamma_2 \vdash \mathbf{withBorrow}\ (\lambda x.t)\ (*v) : *B}\ \text{WITH\&}$$

with incoming heap compatibility $H \bowtie (\Gamma_0 + s \cdot (\Gamma_1 + \Gamma_2))$.
We then get the resulting typing as:

$$\frac{\Gamma_2, y : \&_1 A \vdash t[y/x] : \&_1 B}{\Gamma_2, y : \&_1 A \vdash \mathbf{unborrow}\ (t[y/x]) : *A}\ \text{UNBORROW}$$

thus: $\Gamma' = \Gamma_2, y : \&_1 A$ and construct the goal heap compatibility as:

$$\frac{H \bowtie \Gamma_0 + s \cdot (\Gamma_1 + \Gamma_2) \qquad \Gamma_1 \vdash *v : \&_1 A}{(H, y \mapsto_s (*v)) \bowtie (\Gamma_0 + s \cdot \Gamma_2), y : [\&_1 A]_s}\ \text{EXTLIN}$$

- (unborrow)

$$\frac{\Gamma \vdash t : \&_1 A}{\Gamma \vdash \mathbf{unborrow}\ t : *A}\ \text{UNBORROW}$$

And two possible reductions:

(1) (congUnborrow)

$$\frac{H \vdash t \rightsquigarrow_s H' \vdash t'}{H \vdash \mathbf{unborrow}\ t \rightsquigarrow_s H' \vdash \mathbf{unborrow}\ t'} \rightsquigarrow_{\text{UNBORROW}}$$

By induction as in the other inductive cases.

(2) (unborrowBorrw)

$$\frac{}{H \vdash \mathbf{unborrow}\ (*v) \rightsquigarrow_s H \vdash *v} \rightsquigarrow_{\text{UN\&}}$$

with the refined typing:

$$\frac{\dfrac{\Gamma \vdash v : A}{\Gamma \vdash (*v) : \&_1 A}\ \text{NEC}}{\Gamma \vdash \mathbf{unborrow}\ (*v) : *A}\ \text{UNBORROW}$$

and thus heap compatibility is $H \bowtie \Gamma_0 + s \cdot \Gamma$
The resulting typing matches the goal as:

$$\frac{\Gamma \vdash v : A}{\Gamma \vdash (*v) : *A}\ \text{NEC}$$

thus $\Gamma' = \Gamma$ and outgoing heap compatibility is provided by the incoming compatibility.

- (push)

$$\frac{\Gamma \vdash t : \&_p(A \otimes B)}{\Gamma \vdash \textbf{push}\ t : (\&_p A) \otimes (\&_p B)}\ \textsc{push}$$

And three possible reductions:

(1) (congPush)

$$\frac{H \vdash t \rightsquigarrow_s H' \vdash t'}{H \vdash \textbf{push}\ t \rightsquigarrow_s H' \vdash \textbf{push}\ t'}\ \rightsquigarrow_{\textsc{push}}$$

Inductive case as in other inductive rules.

(2) (pushUnique)

$$\frac{}{H \vdash \textbf{push}\ (*(v_1, v_2)) \rightsquigarrow_s H \vdash (*v_1, *v_2)}\ \rightsquigarrow_{\textsc{push}*}$$

with the refined typing:

$$\frac{\dfrac{\dfrac{\Gamma_1 \vdash v_1 : A \qquad \Gamma_2 \vdash v_2 : B}{\Gamma_1 + \Gamma_2 \vdash (v_1, v_2) : (A \otimes B)}\ \otimes_I}{\Gamma_1 + \Gamma_2 \vdash *(v_1, v_2) : *(A \otimes B)}\ \textsc{nec}}{\Gamma_1 + \Gamma_2 \vdash \textbf{push}\ (*(v_1, v_2)) : (*A) \otimes (*B)}\ \textsc{push}$$

i.e., in the original typing $p = *$ and thus heap compatibility is $H \bowtie \Gamma_0 + s \cdot (\Gamma_1 + \Gamma_2)$ The goal typing is then provided by:

$$\frac{\dfrac{\Gamma_1 \vdash v_1 : A}{\Gamma_1 \vdash *v_1 : *A}\ \textsc{nec} \qquad \dfrac{\Gamma_2 \vdash v_2 : B}{\Gamma_2 \vdash *v_2 : *B}\ \textsc{nec}}{\Gamma_1 + \Gamma_2 \vdash (*v_1, *v_2) : (*A \otimes *B)}\ \otimes_I$$

with the goal heap compatibility $H \bowtie \Gamma_0 + s \cdot (\Gamma_1 + \Gamma_2)$ then provided by the incoming heap compatibility.

- (pull)

$$\frac{\Gamma \vdash t : (\&_p A) \otimes (\&_p B)}{\Gamma \vdash \textbf{pull}\ t : \&_p(A \otimes B)}\ \textsc{pull}$$

And three possible reductions:

(1) (congPull)

$$\frac{H \vdash t \rightsquigarrow_s H' \vdash t'}{H \vdash \textbf{pull}\ t \rightsquigarrow_s H' \vdash \textbf{pull}\ t'}\ \rightsquigarrow_{\textsc{pull}}$$

Inductive case as in other inductive rules.

(2) (pullUnique)

$$\frac{}{H \vdash \textbf{pull}\ (*v_1, *v_2) \rightsquigarrow_s H \vdash *(v_1, v_2)}\ \rightsquigarrow_{\textsc{pull}*}$$

with the refined typing:

$$\frac{\dfrac{\dfrac{\Gamma_1 \vdash v_1 : A}{\Gamma_1 \vdash *v_1 : *A}\ \textsc{nec} \qquad \dfrac{\Gamma_2 \vdash v_2 : B}{\Gamma_2 \vdash *v_2 : *B}\ \textsc{nec}}{\Gamma_1 + \Gamma_2 \vdash (*v_1, *v_2) : (*A \otimes *B)}\ \otimes_I}{\Gamma_1 + \Gamma_2 \vdash \textbf{pull}\ (*v_1, *v_2) : *(A \otimes B)}\ \textsc{pull}$$

thus with $p = *$ in the original typing and thus heap compatibility is $H \bowtie \Gamma_0 + s \cdot (\Gamma_1 + \Gamma_2)$ The goal typing is then provided by:

$$\dfrac{\dfrac{\Gamma_1 \vdash v_1 : A \qquad \Gamma_2 \vdash v_2 : B}{\Gamma_1 + \Gamma_2 \vdash (v_1, v_2) : (A \otimes B)}\ \otimes_I}{\Gamma_1 + \Gamma_2 \vdash *(v_1, v_2) : *(A \otimes B)}\ \textsc{nec}$$

with the goal heap compatibility $H \bowtie \Gamma_0 + s \cdot (\Gamma_1 + \Gamma_2)$ then provided by the incoming heap compatibility.

- (split)

$$\dfrac{\Gamma \vdash t : \&_p A}{\Gamma \vdash \textbf{split}\ t : \&_{\frac{p}{2}} A \otimes \&_{\frac{p}{2}} A}\ \textsc{split}$$

And three possible reductions:

(1) (congSplit)

$$\dfrac{H \vdash t \rightsquigarrow_s H' \vdash t'}{H \vdash \textbf{split}\ t \rightsquigarrow_s H' \vdash \textbf{split}\ t'}\ \rightsquigarrow_{\textsc{split}}$$

Inductive case as in other inductive rules.

(2) (splitArr)

$$\dfrac{a_1 \# H \qquad a_2 \# H}{H, a \mapsto_p id, id \mapsto \textbf{arr} \vdash \textbf{split}\ (*a) \rightsquigarrow_s H, a_1 \mapsto_{\frac{p}{2}} id, a_2 \mapsto_{\frac{p}{2}} id, id \mapsto \textbf{arr} \vdash (*a_1, *a_2)}\ \rightsquigarrow_{\textsc{splitArr}}$$

with the refined typing:

$$\dfrac{\dfrac{0 \cdot \Gamma, a : \mathsf{Array}_{id}\ A \vdash a : \mathsf{Array}_{id}\ A}{0 \cdot \Gamma, a : \mathsf{Array}_{id}\ A \vdash *a : \&_p B}\ \textsc{nec}}{0 \cdot \Gamma, a : \mathsf{Array}_{id}\ A \vdash \textbf{split}\ (*a) : \&_{\frac{p}{2}} A \otimes \&_{\frac{p}{2}} A}\ \textsc{split}$$

and thus heap compatibility is $(H, id \mapsto \textbf{arr}) \bowtie \Gamma_0 + s \cdot (0 \cdot \Gamma, a : \mathsf{Array}_{id}\ A)$ The resulting typing is then given by:

$$\dfrac{\dfrac{0 \cdot \Gamma_1, a_1 : \mathsf{Array}_{id}\ A \vdash a_1 : \mathsf{Array}_{id}\ A}{0 \cdot \Gamma_1, a_1 : \mathsf{Array}_{id}\ A \vdash *a_1 : \&_p(\mathsf{Array}_{id}\ A)}\ \textsc{nec} \quad \dfrac{0 \cdot \Gamma_2, a_2 : \mathsf{Array}_{id'}\ B \vdash a_2 : \mathsf{Array}_{id'}\ B}{0 \cdot \Gamma_2, a_2 : \mathsf{Array}_{id'}\ B \vdash *a_2 : \&_p(*(\mathsf{Array}_{id'}\ B))}\ \textsc{nec}}{0 \cdot \Gamma_1 + 0 \cdot \Gamma_2, a_1 : \mathsf{Array}_{id}\ A, a_2 : \mathsf{Array}_{id'}\ B \vdash (*v_1, *v_2) : (\&_p(\mathsf{Array}_{id}\ A) \otimes \&_p(\mathsf{Array}_{id}\ B))}\ \otimes_I$$

Goal compatibility is $(H, id \mapsto \textbf{arr}, a_1 \mapsto_{\frac{p}{2}} id, a_2 \mapsto_{\frac{p}{2}} id) \bowtie (\Gamma_0 + s \cdot (0 \cdot \Gamma_1 + 0 \cdot \Gamma_2, a_1 : \mathsf{Array}_{id}\ A, a_2 : \mathsf{Array}_{id'}\ B))$ which refines to

$$(H, id \mapsto \textbf{arr}, a_1 \mapsto_{\frac{p}{2}} id, a_2 \mapsto_{\frac{p}{2}} id) \bowtie (\Gamma_0 + 0 \cdot \Gamma_1 + 0 \cdot \Gamma_2, a_1 : \mathsf{Array}_{id}\ A, a_2 : \mathsf{Array}_{id'}\ B)$$

which is constructed by:

$$\dfrac{\dfrac{(H, id \mapsto \textbf{arr}) \bowtie \Gamma_0 + s \cdot (0 \cdot \Gamma, a : \mathsf{Array}_{id}\ A)}{(H, id \mapsto \textbf{arr}, a_1 \mapsto_{\frac{p}{2}} id) \bowtie \Gamma_0 + s \cdot (0 \cdot \Gamma, a : \mathsf{Array}_{id}\ A), a_1 : \mathsf{Array}_{id}\ A}\ \textsc{extRes}}{((H, id \mapsto \textbf{arr}, a_1 \mapsto_{\frac{p}{2}} id), a_2 \mapsto_{\frac{p}{2}} id) \bowtie \Gamma_0 + s \cdot (0 \cdot \Gamma, a : \mathsf{Array}_{id}\ A), a_1 : \mathsf{Array}_{id}\ A, a_2 : \mathsf{Array}_{id'}\ B}\ \textsc{extRes}$$

satisfying the goal here.

(3) (splitPair)

$$\frac{\begin{array}{c} H \vdash \textbf{split}\ (*v)\ \rightsquigarrow_s\ H' \vdash (*v_1, *v_2) \\ H' \vdash \textbf{split}\ (*w)\ \rightsquigarrow_s\ H'' \vdash (*w_1, *w_2) \end{array}}{H \vdash \textbf{split}\ (*(v, w))\ \rightsquigarrow_s\ H'' \vdash (*(v_1, w_1), *(v_2, w_2))} \rightsquigarrow_{\text{SPLIT}\otimes}$$

with the refined typing:

$$\frac{\dfrac{\dfrac{\Gamma_1 \vdash v : A \quad \Gamma_2 \vdash w : B}{\Gamma_1 + \Gamma_2 \vdash (v, w) : A \otimes B} \otimes_I}{\Gamma_1 + \Gamma_2 \vdash *(v, w) : \&_p(A \otimes B)} \text{NEC}}{\Gamma_1 + \Gamma_2 \vdash \textbf{split}\ (*(v, w)) : \&_{\frac{p}{2}}(A \otimes B) \otimes \&_{\frac{p}{2}}(A \otimes B)} \text{SPLIT}$$

and thus heap compatibility is $H \bowtie \Gamma_0 + s \cdot (\Gamma_1 + \Gamma_2)$ By induction on the first premise with $\Gamma_0' = \Gamma_0 + s \cdot \Gamma_2$ and second premise with $\Gamma_0'' = \Gamma_0 + s \cdot \Gamma_2 + s \cdot \Gamma_1' + s \cdot \Gamma_1$, providing:

(a) Γ_1' with $\Gamma_1' \vdash (*v_1, *v_2) : \&_{\frac{p}{2}} A \otimes \&_{\frac{p}{2}} A$ and $H' \bowtie \Gamma_0 + s \cdot \Gamma_2 + s \cdot \Gamma_1'$

(b) Γ_2' with $\Gamma_2' \vdash (*w_1, *w_2) : \&_{\frac{p}{2}} B \otimes \&_{\frac{p}{2}} B$ and $H'' \bowtie \Gamma_0 + s \cdot \Gamma_2 + s \cdot \Gamma_1' + s \cdot \Gamma_1 + s \cdot \Gamma_2'$

The resulting goal type derivation is then:

$$\frac{\dfrac{\dfrac{\Gamma_1 \vdash v_1 : A \quad \Gamma_1' \vdash w_1 : B}{\Gamma_1 + \Gamma_1' \vdash (v_1, w_1) : A \otimes B} \otimes_I}{\Gamma_1 + \Gamma_1' \vdash *(v_1, w_1) : \&_{\frac{p}{2}}(A \otimes B)} \text{NEC} \qquad \dfrac{\dfrac{\Gamma_2 \vdash v_2 : A \quad \Gamma_2' \vdash w_2 : B}{\Gamma_2 + \Gamma_2' \vdash (v_2, w_2) : A \otimes B} \otimes_I}{\Gamma_2 + \Gamma_2' \vdash *(v_2, w_2) : \&_{\frac{p}{2}}(A \otimes B)} \text{NEC}}{\Gamma_1 + \Gamma_1' + \Gamma_2 + \Gamma_2' \vdash (*(v_1, w_1), *(v_2, w_2)) : \&_{\frac{p}{2}}(A \otimes B) \otimes \&_{\frac{p}{2}}(A \otimes B)} \otimes_I$$

The goal compatibility is then $H'' \bowtie \Gamma_0 + s \cdot (\Gamma_1 + \Gamma_1' + \Gamma_2 + \Gamma_2')$ which is provided by the second induction with distributivity and commutativity of $+$.

- (join)

$$\frac{\Gamma_1 \vdash t_1 : \&_p A \qquad \Gamma_2 \vdash t_2 : \&_q A \qquad p + q \leq 1}{\Gamma_1 + \Gamma_2 \vdash \textbf{join}\ t_1\ t_2 : \&_{p+q} A} \text{JOIN}$$

And four possible reductions:

(1) (congJoinL)

$$\frac{H \vdash t_1\ \rightsquigarrow_s\ H' \vdash t_1'}{H \vdash \textbf{join}\ t_1\ t_2\ \rightsquigarrow_s\ H' \vdash \textbf{join}\ t_1'\ t_2} \rightsquigarrow_{\text{JOINL}}$$

Inductive case as in other inductive rules.

(2) (congJoinR)

$$\frac{H \vdash t_2\ \rightsquigarrow_s\ H' \vdash t_2'}{H \vdash \textbf{join}\ v\ t_2\ \rightsquigarrow_s\ H' \vdash \textbf{join}\ v\ t_2'} \rightsquigarrow_{\text{JOINR}}$$

Inductive case as in other inductive rules.

(3) (joinArr)

$$\frac{a \# H}{H, a_1 \mapsto_p id, a_2 \mapsto_q id, id \mapsto \textbf{arr} \vdash \textbf{join}\ *a_1\ *a_2\ \rightsquigarrow_s\ H, a \mapsto_{(p+q)} id, id \mapsto \textbf{arr} \vdash *a} \rightsquigarrow_{\text{JOINARR}}$$

Dualising the splitArr cases exactly.

(4) (joinPair)

$$\dfrac{\begin{array}{c}H \vdash \textbf{join}\ (*v_1)\ (*v_2) \ \rightsquigarrow_s\ H' \vdash *v \\ H' \vdash \textbf{join}\ (*w_1)\ (*w_2) \ \rightsquigarrow_s\ H'' \vdash *w\end{array}}{H \vdash \textbf{join}\ (*(v_1, w_1))\ (*(v_2, w_2)) \ \rightsquigarrow_s\ H'' \vdash *(v, w)} \rightsquigarrow_{\text{JOIN}\otimes}$$

Dualising the joinPair cases exactly.

□

D UNIQUENESS AND BORROW SAFETY PROOFS

Proposition D.1 (Relevant references in a well-typed term are in a compatible heap). *Given* $\Gamma \vdash t : A$ *and* $H \bowtie \Gamma_0 + s \cdot \Gamma$ *then if* $ref \in \mathsf{refs}(t)$ *then* $ref \in \mathsf{dom}(H)$.

Proof. Trivial induction on typing and inversion of heap compatibility; ref must receive a (runtime) type in Γ and thus must feature in H for the heap to be compatible. □

Lemma D.2 (Irrelevant references are preserved by reduction). *For* $\Gamma \vdash t : A$ *and* Γ_0 *and* H *such that* $H \bowtie (\Gamma_0 + s \cdot \Gamma)$ *and* $H \vdash t \rightsquigarrow_s H' \vdash t'$, *then for all references* $ref \in \mathsf{dom}(H) \wedge ref \notin \mathsf{refs}(t)$ *then* $\forall id.(ref \mapsto_p id \in H \implies ref \mapsto_p id \in H')$.

Proof.

- (var)

$$\dfrac{\exists r'.\ s + r' \sqsubseteq r}{H, x \mapsto_r v \vdash x \ \rightsquigarrow_s\ H, x \mapsto_r v \vdash v} \rightsquigarrow_{\text{VAR}}$$

For all $ref \in \mathsf{dom}(H) \wedge ref \notin \mathsf{refs}(x)$, then assume $ref \mapsto_p id \in (H, x \mapsto_r v)$. Since the output heap is the same as the input heap, then the assumption provides the goal.

- (app)
  - (appR)

$$\dfrac{H \vdash t_2 \ \rightsquigarrow_s\ H' \vdash t_2'}{H \vdash v\, t_2 \ \rightsquigarrow_s\ H' \vdash v\, t_2'} \rightsquigarrow_{\text{APPR}}$$

By induction since $ref \notin \mathsf{refs}(v\, t_2)$ implies $ref \notin \mathsf{refs}(t_2)$, and the heaps of the premise are preserved in the conclusion.

  - (appL)

$$\dfrac{H \vdash t_1 \ \rightsquigarrow_s\ H' \vdash t_1'}{H \vdash t_1\, t_2 \ \rightsquigarrow_s\ H' \vdash t_1'\, t_2} \rightsquigarrow_{\text{APPL}}$$

By induction since $ref \notin \mathsf{refs}(t_1\, t_2)$ implies $ref \notin \mathsf{refs}(t_2)$, and the heaps of the premise are preserved in the conclusion.

  - (beta)

$$\dfrac{y \# \{H, v, t\}}{H \vdash (\lambda x.t)\ v \ \rightsquigarrow_s\ H, y \mapsto_s v \vdash t[y/x]} \rightsquigarrow_{\beta}$$

Trivial since the reduction does not affect any references or identifiers in the heap.

  - (congPrim)

$$\dfrac{H \vdash t \ \rightsquigarrow_s\ H' \vdash t'}{H \vdash pr\ t \ \rightsquigarrow_s\ H' \vdash pr\ t'} \rightsquigarrow_{\text{PRIM}}$$

Trivial since $ref \notin \mathsf{refs}(pr\ t_2)$ implies $ref \notin \mathsf{refs}(t_2)$, and the heaps of the premise are preserved in the conclusion

- (existsBeta)

$$\frac{y\#\{H, v, t\}}{H \vdash \textbf{unpack}\ \langle id, x\rangle = \textbf{pack}\ \langle id', v\rangle\ \textbf{in}\ t \rightsquigarrow_s H, y \mapsto_r v \vdash t[y/x]} \rightsquigarrow_{\exists\beta}$$

Then for $ref \in \mathrm{dom}(H) \wedge ref \notin \mathrm{refs}(\textbf{unpack}\ \langle id, x\rangle = \textbf{pack}\ \langle id', v\rangle\ \textbf{in}\ t)$ and with antecedent $ref \mapsto_p id'' \in H$ goal is $ref \mapsto_p id'' \in H'$.
Consider two possibilities:
  - $id'' = id$. If this were the case then $ref : Res_{id''}\ A$ in the typing of the term and thus $ref \in \mathrm{refs}(\textbf{unpack}\ \langle id, x\rangle = \textbf{pack}\ \langle id', v\rangle\ \textbf{in}\ t)$, contradicting the premise and so this trivially holds.
  - $id'' \neq id$, then the renaming here does not change id'' and thus $ref \mapsto_p id'' \in H[id'/id]$.
- (packCong)

$$\frac{H \vdash t \rightsquigarrow_s H \vdash t'}{H \vdash \textbf{pack}\ \langle id, t\rangle \rightsquigarrow_s H \vdash \textbf{pack}\ \langle id, t'\rangle} \rightsquigarrow_{\text{PACK}}$$

Trivial since $ref \notin \mathrm{refs}(\textbf{pack}\ \langle id, t\rangle)$ implies $ref \notin \mathrm{refs}(t)$, and the heaps of the premise are preserved in the conclusion
- (unpackCong),(congBoxElim),(congPromotion),(congPairL),(congPairR),(congPairElim),(congShare),(cong
Trivial like the other congruence rules (see above)
- (betaBox)

$$\frac{y\#\{H, v, t\}}{H \vdash \textbf{let}\ [x] = [v]_r\ \textbf{in}\ t \rightsquigarrow_s H, y \mapsto_{(s*r)} v \vdash t[y/x]} \rightsquigarrow_{\Box\beta}$$

Trivially holds since no new references are added to the outgoing heap.
- (pairBeta)

$$\frac{x'\#\{H, v_1, v_2, t\} \qquad y'\#\{H, v_1, v_2, t\}}{H \vdash \textbf{let}\ (x, y) = (v_1, v_2)\ \textbf{in}\ t \rightsquigarrow_s H, x' \mapsto_s v_1, y' \mapsto_s v_2 \vdash t[y'/y][x'/x]} \rightsquigarrow_{\otimes\beta}$$

Trivially holds since no new references are added to the outgoing heap.
- (betaUnit)

$$\frac{}{H \vdash \textbf{let}\ () = ()\ \textbf{in}\ t \rightsquigarrow_s H \vdash t} \rightsquigarrow_{\beta\text{UNIT}}$$

Trivially holds since the outgoing heap is the same as incoming heap.
- (share)

$$\frac{\mathrm{dom}(H) \equiv \mathrm{refs}(v)}{H, H' \vdash \textbf{share}\ (*v) \rightsquigarrow_s ([H]_0), H' \vdash [v]} \rightsquigarrow_{\text{SHARE}\beta}$$

For $ref \in \mathrm{dom}(H)$ and $ref \notin \mathrm{refs}(\textbf{share}\ (*v))$ then this implies ref in $\mathrm{dom}(H')$ (as otherwise ref would be in H and get zeroed) therefore $\forall id.(ref \mapsto_p id \in H' \implies ref \mapsto_p id \in H')$ providing the goal here.
- (splitRef)

$$\frac{ref_1\#H \qquad ref_2\#H}{H, ref \mapsto_p id, id \mapsto v \vdash \textbf{split}\ (*ref) \rightsquigarrow_s H, ref_1 \mapsto_{\frac{p}{2}} id, ref_2 \mapsto_{\frac{p}{2}} id, id \mapsto v \vdash (*ref_1, *ref_2)} \rightsquigarrow_{\text{SPLITREF}}$$

Let the incoming heap be $H_0 = H, id \mapsto v, ref \mapsto_p id$ For $ref' \in \mathrm{dom}(H_0)$ and $ref' \notin \mathrm{refs}(\textbf{split}\ (*ref))$ then this implies $ref' \neq ref$ and thus $ref' \in \mathrm{dom}(H)$, therefore $\forall id.(ref' \mapsto_p id \in H_0 \implies ref' \mapsto_p id \in H_0)$ trivially providing the goal here.
- (joinRef)

$$\frac{ref\#H}{H, ref_1 \mapsto_p id, ref_2 \mapsto_q id, id \mapsto v \vdash \textbf{join}\ (*ref_1)\ (*ref_2) \rightsquigarrow_s H, ref \mapsto_{(p+q)} id, id \mapsto v \vdash *ref} \rightsquigarrow_{\text{JOINREF}}$$

Let the incoming heap be $H_0 = \mathsf{dom}(H, id \mapsto v, ref_1 \mapsto_p id, ref_2 \mapsto_q id)$. For $ref' \in H_0$ and $ref' \notin \mathsf{refs}(\mathbf{join}\ (*ref_1)\ (*ref_2))$ then this implies $ref' \neq ref$ and thus $ref' \in \mathsf{dom}(H)$, therefore $\forall id.(ref' \mapsto_p id \in H_0 \implies ref' \mapsto_p id \in H_0)$ trivially providing the goal here.

- (splitPair)

$$\frac{\begin{array}{c} H \vdash \mathbf{split}\ (*v) \rightsquigarrow_s H' \vdash (*v_1, *v_2) \\ H' \vdash \mathbf{split}\ (*w) \rightsquigarrow_s H'' \vdash (*w_1, *w_2) \end{array}}{H \vdash \mathbf{split}\ (*(v, w)) \rightsquigarrow_s H'' \vdash (*(v_1, w_1), *(v_2, w_2))} \rightsquigarrow_{\text{SPLIT}\otimes}$$

By induction over the first premise, and transitivity with induction on the second premise.

- (joinPair) Similarly to (splitPair).
- (withBorrow)

$$\frac{y \# \{H, v, t\}}{H \vdash \mathbf{withBorrow}\ (\lambda x.t)\ (*v) \rightsquigarrow_s H, y \mapsto_s (*v) \vdash \mathbf{unborrow}\ t[y/x]} \rightsquigarrow_{\text{WITH}\&}$$

Trivially holds since no new references are added to the outgoing heap.

- (unborrowBorrow)

$$\frac{}{H \vdash \mathbf{unborrow}\ (*v) \rightsquigarrow_s H \vdash *v} \rightsquigarrow_{\text{UN}\&}$$

Trivial since the input heap and output heap are the same.

- (newRef)

$$\frac{ref \# H \qquad id \# H}{H \vdash \mathbf{newRef}\ v \rightsquigarrow_s H, ref \mapsto_1 id, id \mapsto \mathbf{ref}(v) \vdash \mathbf{pack}\ \langle id, *ref \rangle} \rightsquigarrow_{\text{NEWREF}}$$

Trivially holds since heap references not changed.

- (swapRef)

$$\frac{}{H, ref \mapsto_p id, id \mapsto \mathbf{ref}(v) \vdash \mathbf{swapRef}\ (*ref)\ v' \rightsquigarrow_s H, ref \mapsto_p id, id \mapsto \mathbf{ref}(v') \vdash v} \rightsquigarrow_{\text{SWAPREF}}$$

Trivially holds since heap H whose references are not in the term is preserved into the outgoing heap.

- (freezeRef)

$$\frac{}{H, ref \mapsto_p id, id \mapsto \mathbf{ref}(v) \vdash \mathbf{freezeRef}\ (*ref) \rightsquigarrow_s H \vdash v} \rightsquigarrow_{\text{FREEZEREF}}$$

Trivially holds since heap H whose references are not in the term is preserved into the outgoing heap.

- (readRef)

$$\frac{}{H, ref \mapsto_p id, id \mapsto \mathbf{ref}([v]_{r+1}) \vdash \mathbf{readRef}\ (*ref) \rightsquigarrow_s H, ref \mapsto_p id, id \mapsto \mathbf{ref}([v]_r) \vdash (v, *ref)} \rightsquigarrow_{\text{READREF}}$$

Trivially holds since heap H whose references are not in the term is preserved into the outgoing heap.

- (newArray), (readArray), (writeArray), (deleteArray) follow in a similar way to the polymorphic reference counterparts above.
- (copyBeta)

$$\frac{\mathsf{dom}(H') \equiv \mathsf{refs}(v) \qquad (H'', \theta, \overline{id}) = \mathsf{copy}(H') \qquad y \# \{H, v, t\}}{H, H' \vdash \mathbf{clone}\ [v]_r\ \mathbf{as}\ x\ \mathbf{in}\ t \rightsquigarrow_s H, H', H'', y \mapsto_s \mathbf{pack}\ \langle \overline{id}, *(\theta(v)) \rangle \vdash t[y/x]} \rightsquigarrow_{\text{CLONE}\beta}$$

then for $ref \in \mathsf{dom}(H,, H')$ and $ref \notin \mathsf{refs}(\mathbf{clone}\ [v]\ \mathbf{as}\ x\ \mathbf{in}\ t)$ then for id where $(ref \mapsto_p id \in H, H'$ then we have $ref \mapsto_p id \in (H, H'), H'')$ since the heap is only extended not changed.

- (pushUnique), (pullUnique) Trivial since the incoming and outgoing heaps are the same.

□

Lemma D.3 (Borrow safety). *For a well-typed term $\Gamma \vdash t : A$ and all Γ_0, s, and heaps H such that $H \bowtie (\Gamma_0 + s \cdot \Gamma)$, and given a single-step reduction $H \vdash t \rightsquigarrow_s H' \vdash t'$ then for all $id \in \mathsf{dom}(H)$:*

$$\sum_{\substack{\forall ref \in \mathsf{refs}(t).\\ ref \mapsto_p id \in H}} p = 1 \quad \Longrightarrow \quad \sum_{\substack{\forall ref \in \mathsf{refs}(t').\\ ref \mapsto_{p'} id \in H'}} p' \in \{0, 1\}$$

i.e., for all resources with identifier id in the incoming heap and all references in the term pointing to this resource, if the sum of all permissions pointing to this resource are 1 *in the incoming heap then either this is preserved in the outgoing heap* or *the total permissions in the output heap is* 0*, i.e., this resource has now been fully shared and has no ownership tracking now.*

Furthermore, any resources in the outgoing heap that did not appear in the initial heap with references in the final term should have permissions summing to 1*. That is, for all* $id' \in \mathsf{dom}(H') \wedge id' \notin \mathsf{dom}(H)$*:*

$$\sum_{\substack{\forall ref' \in \mathsf{refs}(t').\\ ref' \mapsto_q id' \in H'}} q = 1$$

Proof. By induction on typing $\Gamma \vdash t : A$.

- (var)

$$\frac{}{0 \cdot \Gamma, x : A \vdash x : A}\ \textsc{var}$$

Then we also have a heap H such that $H \bowtie (0 \cdot \Gamma, x : A)$.
By inversion of heap compatibility, which implies that there exists a subheap H_1 such that $H = H_1, x \mapsto_r (\Gamma' \vdash v : A)$ (where there exists some r' such that $r' + 1 \sqsubseteq r$, and since $\downarrow v = \emptyset$):

$$\frac{H_1 \bowtie 0 \cdot \Gamma + \Gamma' + \downarrow v \qquad x \notin \mathsf{dom}(H_1) \qquad \Gamma' \vdash v : A \qquad \exists r'.\, r' + s \equiv r}{(H_1, x \mapsto_r (\Gamma' \vdash v : A)) \bowtie (0 \cdot \Gamma, x : [A]_s)}\ \textsc{ext}$$

and we have the reduction:

$$\frac{\exists r'.\, r' + s \sqsubseteq r}{H_1, x \mapsto_r v \vdash x \rightsquigarrow_s H_1, x \mapsto_r v \vdash v}\ \rightsquigarrow_{\textsc{var}}$$

For part 1 of the lemma, for all $id \in \mathsf{dom}(H)$ and for all $ref \in \mathsf{refs}(t)$ then we assume the antecedent:

$$\sum_{ref \mapsto_p id \in H} p = 1$$

Since the heap is unchanged by the reduction, $H' = H$ and then we trivially conclude with the antecedent evidence:

$$\sum_{ref \mapsto_p id \in H'} p = 1$$

For part 2, we have:

$$\forall id' \in \mathsf{dom}(H'), ref \in \mathsf{refs}(v). \left(ref \notin \mathsf{dom}(H) \implies \sum_{ref \mapsto_q id' \in H'} q = 1 \right)$$

trivially since the premise must always be false by heap compatibility and that the heap is preserved by the reduction here.

- (abs)

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \multimap B}\ \text{ABS}$$

Is trivial since there are no possible reductions.

- (app)

$$\frac{\Gamma_1 \vdash t_1 : A \multimap B \qquad \Gamma_2 \vdash t_2 : A}{\Gamma_1 + \Gamma_2 \vdash t_1\ t_2 : B}\ \text{APP}$$

with a heap H such that $H \bowtie (\Gamma_1 + \Gamma_2)$ and three reductions (beta and application congruence on the left and right) then several possible reductions following from primitive applications:

(1)

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash t_1\ t_2 \rightsquigarrow_s H' \vdash t_1'\ t_2} \rightsquigarrow_{\text{APPL}}$$

For part 1, for $id \in \text{dom}(H)$ and $ref \in \text{refs}(t_1\ t_2)$, assuming the antecedent

$$\sum_{ref \mapsto_p id \in H} p = 1$$

From this, since $\text{refs}(t_1) \subseteq \text{refs}(t_1\ t_2)$, we induct on the premise reduction to attain that:

$$(1) \sum_{ref \mapsto_p id \in H'} p \in \{0, 1\}$$

$$(2) \forall id' \in \text{dom}(H'), ref \in \text{refs}(t_2').\left(ref \notin \text{dom}(H) \implies \sum_{ref \mapsto_q id' \in H'} q = 1\right)$$

To conclude then we use (1) but also need that $\forall ref' \in \text{refs}(t_2).\ \sum_{ref' \mapsto_p id \in H'} p \in \{0, 1\}$. We consider two cases depending on whether $ref' \in \text{refs}(t_1)$ or not:

– $ref' \in \text{refs}(t_1)$, therefore by (1) we also have that $\sum_{a' \mapsto_p id \in H'} p \in \{0, 1\}$, satisfying the goal.

– $ref' \notin \text{refs}(t_1)$ therefore $\sum_{ref' \mapsto_p id \in H'} p = 1$ satisfying the goal.

Second goal is that:

$$\forall id' \in \text{dom}(H'), ref \in \text{refs}(t_1\ t_2').\left(ref \notin \text{dom}(H) \implies \sum_{ref \mapsto_q id' \in H'} q = 1\right)$$

Thus we can conclude with (2) along with the cases for $\forall ref' \in \text{refs}(t_1)$, for which we discriminate based on whether ref' is contained also in t_2':

(a) $ref' \in \text{refs}(t_2')$ therefore we have from (2) that $\left(ref' \notin \text{dom}(H) \implies \sum_{ref' \mapsto_q id' \in H'} q = 1\right)$ satisfying the goal.

(b) $ref' \notin \text{refs}(t_2')$ then by Proposition D.1 with $\Gamma_1 + \Gamma_2 \vdash t_1\ t_2' : B$ (Type preservation) and $ref' \in \text{refs}(t_1\ t_2')$ and $H \bowtie \Gamma_0 + s \cdot (\Gamma_1 + \Gamma_2)$ then $ref' \in \text{dom}(H)$ and thus the antecedent of condition (2) is false and so is trivially true.

(2)

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash v\ t_2 \rightsquigarrow_s H' \vdash v\ t_2'} \rightsquigarrow_{\text{APPR}}$$

By the same reasoning as above for the (appL) case *mutatis mutandis*: with t_1 being a value and induction happening on the right-hand term t_2's reduction.

(3) Alternatively $t_1 = \lambda x.t_1'$ such that typing is:

$$\dfrac{\dfrac{\Gamma_1, x : A \vdash t' : B}{\Gamma_1 \vdash t' : A \multimap B}\;\text{ABS} \qquad \Gamma_2 \vdash v : A}{\Gamma_1 + \Gamma_2 \vdash (\lambda x.t')\; v : B}\;\text{APP}$$

and we have reduction:

$$\dfrac{y \# \{H, v, t\}}{H \vdash (\lambda x.t)\; v \rightsquigarrow_s H, y \mapsto_s v \vdash t[y/x]} \rightsquigarrow_\beta$$

(Part 1) Then for $id \in \mathrm{dom}(H)$ and $ref \in \mathrm{refs}((\lambda x.t')\; v)$, we assume the antecedent: $\sum_{ref \mapsto_p id \in H} p = 1$.
Since the right-hand side of reduction does not introduce any new references then the goal follows from the antecedent: $\sum_{ref \mapsto_p id \in H'} = 1$.
(Part 2) Trivial since the antecedent is always false as $\mathrm{refs}(t[y/x]) \subseteq \mathrm{refs}((\lambda x.t)\; v)$.

(4) $t_1 = \mathbf{newRef}$
Therefore we induct on the second argument:
- t_2 is a value and thus the typing is:

$$\dfrac{}{0 \cdot \Gamma \vdash \mathbf{newRef} : A \multimap \exists id.*(\mathrm{Ref}_{id}\; A)}\;\text{NEWREF}$$

with $H \bowtie (\Gamma_0 + \Gamma)$.
Thus there is a reduction as follows:

$$\dfrac{ref \# H \qquad id \# H}{H \vdash \mathbf{newRef}\; v \rightsquigarrow_s H, ref \mapsto_1 id, id \mapsto \mathbf{ref}(v) \vdash \mathbf{pack}\; \langle id, *ref \rangle} \rightsquigarrow_{\text{NEWREF}}$$

Part 1 follows as all id in the input heap and references pointing to them are preserved in the output heap.
Part 2 follows since where $ref \notin \mathrm{dom}(H)$ and we have that $ref \mapsto_1 id \in H, ref \mapsto_1 id$ satisfying the goal.
- t_2 is not a value and thus has a reduction, therefore we can build the compound reduction:

$$\dfrac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \mathbf{newRef}\; t_2 \rightsquigarrow_s H' \vdash \mathbf{newRef}\; t_2'} \rightsquigarrow_{\text{PRIM}}$$

Then the result holds by induction.

(5) $t_1 = \mathbf{swapRef}$
and there is a reduction:

$$\dfrac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \mathbf{swapRef}\; t_2 \rightsquigarrow_s H' \vdash \mathbf{swapRef}\; t_2'} \rightsquigarrow_{\text{PRIM}}$$

Therefore the borrow safety result holds by induction.

(6) $t_1 = \mathbf{swapRef}\;(*ref)$ Case on progress for t_2:
- t_2 is a value and we have a reduction:

$$\dfrac{}{H, ref \mapsto_p id, id \mapsto \mathbf{ref}(v) \vdash \mathbf{swapRef}\;(*ref)\; v' \rightsquigarrow_s H, ref \mapsto_p id, id \mapsto \mathbf{ref}(v') \vdash v} \rightsquigarrow_{\text{SWAPREF}}$$

(Part 1) Trivially true since the only change to the heap is the value that id is pointing to.
(Part 2) No new references are created so trivially true.

– t_2 is not a value and therefore has a reduction from which we form the congruence:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \mathbf{swapRef}\,(*ref)\; t_2 \rightsquigarrow_s H' \vdash \mathbf{swapRef}\,(*ref)\; t_2'} \rightsquigarrow_{\text{PRIM}}$$

(7) $t_1 = \mathbf{freezeRef}$ Case on progress for t_2:

– t_2 is a value therefore by the value lemma $t_2 = *ref$ and we have a reduction:

$$\frac{}{H, ref \mapsto_p id, id \mapsto \mathbf{ref}(v) \vdash \mathbf{freezeRef}\,(*ref) \rightsquigarrow_s H \vdash v} \rightsquigarrow_{\text{FREEZEREF}}$$

(Part 1) Thus, for $id' \in \mathrm{dom}(H)$, $ref' \in \mathrm{refs}(t)$ with assumption $\sum_{ref' \mapsto_p id' \in H} p = 1$. If $id' = id$ and $ref' = ref$ then we can conclude with: $\sum_{ref' \mapsto_p id' \in H'} p = 0$ since the reference and identifier assignments are removed from the output heap. Otherwise, the heap is preserved so $\sum_{ref' \mapsto_p id' \in H'} p = 1$

(Part 2) No new references are created so trivially true.

– t_2 is not a value and therefore has a reduction from which we form the congruence:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \mathbf{freezeRef}\; t_2 \rightsquigarrow_s H' \vdash \mathbf{freezeRef}\; t_2'} \rightsquigarrow_{\text{PRIM}}$$

(8) $t_1 = \mathbf{readRef}$ Case on progress for t_2:

– t_2 is a value therefore by the value lemma $t_2 = *ref$ and we have a reduction:

$$\frac{}{H, ref \mapsto_p id, id \mapsto \mathbf{ref}([v]_{r+1}) \vdash \mathbf{readRef}\,(*ref) \rightsquigarrow_s H, ref \mapsto_p id, id \mapsto \mathbf{ref}([v]_r) \vdash (v, *ref)} \rightsquigarrow_{\text{READREF}}$$

(Part 1) Trivially true since the heap is preserved.

(Part 2) No new references are created so trivially true.

– t_2 is not a value and therefore has a reduction from which we form the congruence:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \mathbf{readRef}\; t_2 \rightsquigarrow_s H' \vdash \mathbf{readRef}\; t_2'} \rightsquigarrow_{\text{PRIM}}$$

(9) $t_1 = \mathbf{newArray}$ therefore $A = \mathbb{N}$

Therefore we induct on the second argument:

– t_2 is a value and therefore by the value lemma (Lemma C.1) $t_2 = n$ and thus the typing is:

$$\frac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathbf{newArray}\; n : *(\mathsf{Array}_{id}\; \mathbb{F})} \textsc{TyDerivedNewArray}$$

with $H \bowtie (\Gamma_0 + s \cdot \Gamma)$.

Thus there is a reduction as follows:

$$\frac{ref \# H \qquad id \# H}{H \vdash \mathbf{newArray}\; n \rightsquigarrow_s H, ref \mapsto_1 id, id \mapsto \mathsf{init} \vdash \mathbf{pack}\,\langle id, *ref \rangle} \rightsquigarrow_{\text{NEWARRAY}}$$

Part 1 follows as all id in the input heap and references pointing to them are preserved in the output heap.

Part 2 follows since where $ref \notin \mathrm{dom}(H)$ and we have that $ref \mapsto_1 id \in H, ref \mapsto_1 id$ satisfying the goal.

– t_2 is not a value and thus has a reduction, therefore we can build the compound reduction:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \mathbf{newArray}\; t_2 \rightsquigarrow_s H' \vdash \mathbf{newArray}\; t_2'} \rightsquigarrow_{\text{PRIM}}$$

Then the result holds by induction.

(10) $t_1 = \textbf{readArray}$ therefore $A = \&_p(\mathsf{Array}_{id}\ \mathbb{F}) \multimap \mathbb{N} \multimap \mathbb{F} \otimes \&_p(\mathsf{Array}_{id}\ \mathbb{F})$
and there is a reduction:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \textbf{readArray}\ t_2 \rightsquigarrow_s H' \vdash \textbf{readArray}\ t_2'} \rightsquigarrow_{\text{PRIM}}$$

Therefore the borrow safety result holds by induction.

(11) $t_1 = \textbf{readArray}\ (*a)$ therefore $A = \mathbb{N} \multimap \mathbb{F} \otimes *(\mathsf{Array}_{id}\ \mathbb{F})$

– t_2 is a value and therefore by the value lemma on $\Gamma_2 \vdash t_2 : \mathbb{N}$ (Lemma C.1) implies $t_2 = n$ and thus the typing is refined at runtime as follows:

$$\dfrac{\dfrac{[\Gamma_1], a : \mathsf{Array}_{id}\ \mathbb{F} \vdash a : (\mathsf{Array}_{id}\ \mathbb{F})\ \text{REF}}{[\Gamma_1], a : \mathsf{Array}_{id}\ \mathbb{F} \vdash *a : *(\mathsf{Array}_{id}\ \mathbb{F})} *\text{REF}^* \qquad \Gamma_2 \vdash n : \mathbb{N}}{[\Gamma_1] + \Gamma_2, a : \mathsf{Array}_{id}\ \mathbb{F} \vdash \textbf{readArray}\ (*a)\ n : \mathbb{F} \otimes *(\mathsf{Array}_{id}\ \mathbb{F})} \text{TyDerivedreadArray}$$

with $H' \bowtie (\Gamma_0 + s \cdot ([\Gamma_1] + \Gamma_2, a : \mathsf{Array}_{id}\ \mathbb{F}))$, and by the heap compatibility rule for array references there exists some H such that $H' = H, a \mapsto_p id, id \mapsto \textbf{arr}$.
Then there is a reduction as follows:

$$\frac{}{H, ref \mapsto_p id, id \mapsto \textbf{arr}[i] = v \vdash \textbf{readArray}\ (*ref)\ i \rightsquigarrow_s H, ref \mapsto_p id, id \mapsto \textbf{arr}[i] = v \vdash (v, *ref)} \rightsquigarrow_{\text{READARRAY}}$$

As the entirety of the heap is preserved (including the array reference a being read from here), the goal follows trivially.

– t_2 is not a value and thus has a reduction, therefore we can build the compound reduction:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \textbf{readArray}\ (*a)\ t_2 \rightsquigarrow_s H' \vdash \textbf{readArray}\ (*a)\ t_2'} \rightsquigarrow_{\text{PRIM}}$$

And therefore the borrow safety result holds by induction.

(12) $t_1 = \textbf{writeArray}$ therefore $A = \&_p(\mathsf{Array}_{id}\ \mathbb{F}) \multimap \mathbb{N} \multimap \mathbb{F} \multimap \&_p(\mathsf{Array}_{id}\ \mathbb{F})$
with reduction

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \textbf{writeArray}\ t_2 \rightsquigarrow_s H' \vdash \textbf{writeArray}\ t_2'} \rightsquigarrow_{\text{PRIM}}$$

Therefore the borrow safety result holds by induction.

(13) $t_1 = \textbf{writeArray}\ (*a)$ therefore $A = \mathbb{N} \multimap \mathbb{F} \multimap *(\mathsf{Array}_{id}\ \mathbb{F})$
with reduction:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \textbf{writeArray}\ (*a)\ t_2 \rightsquigarrow_s H' \vdash \textbf{writeArray}\ (*a)\ t_2'} \rightsquigarrow_{\text{PRIM}}$$

Therefore the borrow safety result holds by induction.

(14) $t_1 = \textbf{writeArray}\ (*a)\ i$ therefore $A = \mathbb{F} \otimes \&_1(\mathsf{Array}_{id}\ \mathbb{F})$
Then we case on progress for t_2:

– t_2 is a value and therefore by the value lemma on $\Gamma_2 \vdash t_2 : \mathbb{F}$ (Lemma C.1) implies $t_2 = f$ and thus the typing is refined at runtime as follows:

$$\dfrac{\dfrac{[\Gamma_1], a : \mathsf{Array}_{id}\ \mathbb{F} \vdash a : (\mathsf{Array}_{id}\ \mathbb{F})\ \text{REF}}{[\Gamma_1], a : \mathsf{Array}_{id}\ \mathbb{F} \vdash (*a) : \&_p(\mathsf{Array}_{id}\ \mathbb{F})} \text{NEC} \qquad \Gamma_2 \vdash i : \mathbb{N} \quad \Gamma_3 \vdash f : \mathbb{F}}{[\Gamma_1] + \Gamma_2 + \Gamma_3, a : \mathsf{Array}_{id}\ \mathbb{F} \vdash \textbf{writeArray}\ (*a)\ i\ f : \&_p(\mathsf{Array}_{id}\ \mathbb{F})} \text{TyDerivedwriteArray}$$

with $H' \bowtie (\Gamma_0 + s \cdot [\Gamma_1] + \Gamma_2 + \Gamma_3, a : \mathsf{Array}_{id}\ \mathbb{F})$, and by the heap compatibility rule for array references there exists some H such that $H' = H, a \mapsto_p id, id \mapsto \textbf{arr}$.

Then there is a reduction as follows:

$$\frac{}{H, ref \mapsto_p id, id \mapsto \mathbf{arr} \vdash \mathbf{writeArray}\ (*ref)\ i\ v \ \rightsquigarrow_s\ H, ref \mapsto_p id, id \mapsto \mathbf{arr}[i] = v \vdash *ref} \rightsquigarrow_{\text{WRITEARRAY}}$$

Here, the only change in the heap is to the array value that id is pointing to. As the remainder of the heap is preserved, both parts of the goal follow trivially.

– t_2 is not a value and thus has a reduction, therefore we can build the compound reduction:

$$\frac{H \vdash t_2 \ \rightsquigarrow_s\ H' \vdash t_2'}{H \vdash \mathbf{writeArray}\ (*a)\ i\ t_2 \ \rightsquigarrow_s\ H' \vdash \mathbf{writeArray}\ (*a)\ i\ t_2'} \rightsquigarrow_{\text{PRIM}}$$

Therefore the borrow safety result holds by induction.

(15) $t_1 = \mathbf{deleteArray}$ therefore $A = *(\mathsf{Array}_{id}\ \mathbb{F}) \multimap \mathsf{unit}$

Case on progress for t_2:

– t_2 is a value therefore by the value lemma $t_2 = *ref$ and we have a reduction:

$$\frac{}{H, ref \mapsto_p id, id \mapsto \mathbf{arr} \vdash \mathbf{deleteArray}\ (*ref) \ \rightsquigarrow_s\ H \vdash ()} \rightsquigarrow_{\text{DELETEARRAY}}$$

(Part 1) Thus, for $id' \in \mathrm{dom}(H)$, $ref' \in \mathrm{refs}(t)$ with assumption $\sum_{ref' \mapsto_p id' \in H} p = 1$. If $id' = id$ and $ref' = ref$ then we can conclude with: $\sum_{ref' \mapsto_p id' \in H'} p = 0$ since the reference and identifier assignments are removed from the output heap. Otherwise, the heap is preserved so $\sum_{ref' \mapsto_p id' \in H'} p = 1$

(Part 2) No new references are created so trivially true.

– t_2 is not a value and therefore has a reduction from which we form the congruence:

$$\frac{H \vdash t_2 \ \rightsquigarrow_s\ H' \vdash t_2'}{H \vdash \mathbf{deleteArray}\ t_2 \ \rightsquigarrow_s\ H' \vdash \mathbf{deleteArray}\ t_2'} \rightsquigarrow_{\text{PRIM}}$$

- (pr)

$$\frac{\Gamma \vdash t : A \qquad \neg\mathsf{resourceAllocator}(t)}{r \cdot \Gamma \vdash [t]_r : \Box_r A} \text{PR}$$

with a heap H such that $H \bowtie (\Gamma_0 + s \cdot (r \cdot \Gamma))$ and reduction:

$$H \vdash [t]_r \ \rightsquigarrow_s\ H' \vdash t''$$

which has only one possible derivation:

$$\frac{H \vdash t \ \rightsquigarrow_{s*r}\ H' \vdash t'}{H \vdash [t]_r \ \rightsquigarrow_s\ H' \vdash [t']_r} \rightsquigarrow_{\Box}$$

Thus, induction provides the goal, where the incoming heap compatibility by provides the inductive heap compatibility as $H \bowtie \Gamma_0 + (s * r) \cdot \Gamma_0$ by associativity of $*$.

- (elim)

$$\frac{\Gamma_1 \vdash t_1 : \Box_r A \qquad \Gamma_2, x : [A]_r \vdash t_2 : B}{\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\ [x] = t_1\ \mathbf{in}\ t_2 : B} \text{ELIM}$$

with a heap H such that $H \bowtie \Gamma_0 + s \cdot (\Gamma_1 + \Gamma_2)$ and reduction:

$$H \vdash \mathbf{let}\ [x] = t_1\ \mathbf{in}\ t_2 \ \rightsquigarrow_s\ H' \vdash t'$$

which has two possible derivations:

(1)

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash \textbf{let}\,[x] = t_1\ \textbf{in}\ t_2 \rightsquigarrow_s H' \vdash \textbf{let}\,[x] = t_1'\ \textbf{in}\ t_2} \rightsquigarrow_{\text{LET}\square}$$

Then induction provides the goal following the same scheme of generalising the inductive evidence as for the $\rightsquigarrow_{\text{APPL}}$ case.

(2) Alternatively $t_1 = [v]$ such that the typing is:

$$\frac{\dfrac{\Gamma_1 \vdash v : A}{r \cdot \Gamma_1 \vdash [v]_r : \square_r A}\ \text{PR} \qquad \Gamma_2, x : [A]_r \vdash t_2 : B}{r \cdot \Gamma_1 + \Gamma_2 \vdash \textbf{let}\,[x] = [v]_r\ \textbf{in}\ t_2 : B}\ \text{ELIM}$$

and we have reduction:

$$\frac{y\#\{H, v, t\}}{H \vdash \textbf{let}\,[x] = [v]_r\ \textbf{in}\ t \rightsquigarrow_s H, y \mapsto_{(s*r)} v \vdash t[y/x]} \rightsquigarrow_{\square\beta}$$

(where $\Gamma = r \cdot \Gamma_1$ in the above.
Which trivially satisfies the goal since all references are then in H and we get the conditions trivially (since no references are manipulated) similar to the β proof above.

- (der)

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma, x : [A]_1 \vdash t : B}\ \text{DER}$$

with a heap H such that $H \bowtie \Gamma_0 + s \cdot (\Gamma, x : [A]_1)$ and reduction:

$$H \vdash t \rightsquigarrow_s H' \vdash t'$$

As the term t is unchanged from the premise, the goal holds by induction regardless of how this reduction is derived.

- (approx)

$$\frac{\Gamma, x : [A]_r, \Gamma' \vdash t : B \qquad r \sqsubseteq s}{\Gamma, x : [A]_s, \Gamma' \vdash t : B}\ \text{APPROX}$$

with a heap H such that $H \bowtie (\Gamma, x : [A]_s, \Gamma')$ and reduction:

$$H \vdash t \rightsquigarrow_s H' \vdash t'$$

As the term t is unchanged from the premise, the goal holds by induction regardless of how this reduction is derived.

- (pairIntro)

$$\frac{\Gamma_1 \vdash t_1 : A \qquad \Gamma_2 \vdash t_2 : B}{\Gamma_1 + \Gamma_2 \vdash (t_1, t_2) : A \otimes B}\ \otimes_I$$

with a heap H such that $H \bowtie \Gamma_0 + s \cdot (\Gamma_1 + \Gamma_2)$ and reduction:

$$H \vdash (t_1, t_2) \rightsquigarrow_s H' \vdash t''$$

which has two possible derivations:

(1)

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash (t_1, t_2) \rightsquigarrow_s H' \vdash (t_1', t_2)} \rightsquigarrow_{\otimes\text{L}}$$

Here, induction on t_1 provides the goal following the same scheme of generalising the inductive evidence as for the $\rightsquigarrow_{\text{APPL}}$ case.

(2) Otherwise, $t_1 = v$ for some value v and we can perform the reduction:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash (v, t_2) \rightsquigarrow_s H' \vdash (v, t_2')} \rightsquigarrow_{\otimes\text{R}}$$

Here, induction on t_2 provides the goal, following the same scheme of generalising the inductive evidence as for the $\rightsquigarrow_{\text{APPL}}$ case.

- (pairElim)

$$\frac{\Gamma_1 \vdash t_1 : A \otimes B \qquad \Gamma_2, x : A, y : B \vdash t_2 : C}{\Gamma_1 + \Gamma_2 \vdash \textbf{let}\,(x, y) = t_1\,\textbf{in}\,t_2 : C} \otimes_E$$

Two possible reductions:

(1)

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash \textbf{let}\,(x, y) = t_1\,\textbf{in}\,t_2 \rightsquigarrow_s H' \vdash \textbf{let}\,(x, y) = t_1'\,\textbf{in}\,t_2} \rightsquigarrow_{\text{LET}\otimes}$$

and induction provides the goal, following the same scheme of generalising the inductive evidence as for the $\rightsquigarrow_{\text{APPL}}$ case.

(2) otherwise $t_1 = (v_1, v_2)$ such that the typing is:

$$\frac{\dfrac{\Gamma_3 \vdash v_1 : A \quad \Gamma_4 \vdash v_2 : B}{\Gamma_3 + \Gamma_4 \vdash (v_1, v_2) : (A \otimes B)} \otimes_I \qquad \Gamma_2, x : A, y : B \vdash t_2 : C}{\Gamma_3 + \Gamma_4 + \Gamma_2 \vdash \textbf{let}\,(x, y) = (v_1, v_2)\,\textbf{in}\,t_2 : C} \otimes_E$$

and we have reduction:

$$\frac{\Gamma_1 \vdash t_1' : A \quad \Gamma_2 \vdash t_1'' : B}{H \vdash \textbf{let}\,(x, y) = (t_1', t_1'')\,\textbf{in}\,t_3 \rightsquigarrow_s H, x \mapsto_s (\Gamma_1 \vdash t_1' : A), y \mapsto_s (\Gamma_2 \vdash t_1'' : B) \vdash t_3} \rightsquigarrow_{\otimes\beta}$$

Then (part 1) for all $id \in \text{refs}(H)$ and all $ref \in \text{refs}(t_3)$ we have:

$$\sum_{\substack{\forall ref \,\in\, \text{refs}(t). \\ ref \mapsto_p id \,\in\, H}} p \quad = \sum_{\substack{\forall ref \,\in\, \text{refs}(t'). \\ ref \mapsto_{p'} id \,\in\, H'}} p'$$

satisfying the goal and (part 2) for all $id' \in \text{dom}(H') \land id' \notin \text{dom}(H)$:

$$\sum_{\substack{\forall ref' \,\in\, \text{refs}(t'). \\ ref' \mapsto_q id' \,\in\, H'}} q \quad = 1$$

since no references are introduced.

- (unitIntro)

$$\frac{}{0 \cdot \Gamma \vdash () : \text{unit}} 1_I$$

The result type here is 1, so cannot contain any types of the form $\&_p A$, and therefore the result holds trivially.

- (unitElim)

$$\frac{\Gamma_1 \vdash t_1 : \text{unit} \qquad \Gamma_2 \vdash t_2 : B}{\Gamma_1 + \Gamma_2 \vdash \textbf{let}\,() = t_1\,\textbf{in}\,t_2 : B} 1_E$$

Following essentially the same structure as the tensor proof where array reference counting is not used so induction provides the goal.

- (share)

$$\frac{\Gamma \vdash t : {*}A}{\Gamma \vdash \textbf{share}\, t : \Box_r A}\ \textsc{share}$$

with a heap H such that $H \bowtie \Gamma_0 + s \cdot \Gamma$ and reduction:

$$H \vdash \textbf{share}\, t \rightsquigarrow_s H' \vdash t''$$

which has two possible derivations:

(1)

$$\frac{H \vdash t \rightsquigarrow_s H' \vdash t'}{H \vdash \textbf{share}\, t \rightsquigarrow_s H' \vdash \textbf{share}\, t'} \rightsquigarrow_{\textsc{share}}$$

Here, induction provides the goal following the same scheme of generalising the inductive evidence as for the $\rightsquigarrow_{\textsc{appL}}$ case.

(2) t has the form $*v$ for some value v, and we can reduce:

$$\frac{\mathsf{dom}(H) \equiv \mathsf{refs}(v)}{H, H' \vdash \textbf{share}\,({*}v) \rightsquigarrow_s ([H]_0), H' \vdash [v]} \rightsquigarrow_{\textsc{share}\beta}$$

(Part 1) Then for $id \in \mathsf{dom}(H, H')$, $ref \in \mathsf{refs}(\textbf{share}\,({*}v))$, we assume the antecedent $\sum_{ref \mapsto_p id \in H, H'} p = 1$.

Then we have two cases:

- $ref \in \mathsf{dom}(H)$ therefore in the output heap $\sum_{ref \mapsto_p id \in ([H]_0), H'} p = 0$ by the zeroing of H here (thus $p = 0$), satisfying the goal.
- $ref \in \mathsf{dom}(H')$ then $\sum_{ref \mapsto_p id \in ([H]_0), H'} p = 1$ following from the antecedent.

(Part 2) No new references are introduced in the output term, so this trivially holds.

- (clone)

$$\frac{\Gamma_1 \vdash t_1 : \Box_r A \qquad \Gamma_2, x : {*}A \vdash t_2 : \Box_r B \qquad 1 \sqsubseteq r}{\Gamma_1 + \Gamma_2 \vdash \textbf{clone'}\ t_1\ \textbf{as}\ x\ \textbf{in}\ t_2 : \Box_r B}\ \textsc{clone'}$$

with a heap H such that $H \bowtie (\Gamma_1 + \Gamma_2)$ and reduction:

$$H \vdash \textbf{clone}\ t_1\ \textbf{as}\ x\ \textbf{in}\ t_2 \rightsquigarrow_s H' \vdash t''$$

which has two possible derivations:

(1)

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash \textbf{clone}\ t_1\ \textbf{as}\ x\ \textbf{in}\ t_2 \rightsquigarrow_s H' \vdash \textbf{clone}\ t_1'\ \textbf{as}\ x\ \textbf{in}\ t_2} \rightsquigarrow_{\textsc{clone}}$$

Here, induction provides the goal, following the same scheme of generalising the inductive evidence as for the $\rightsquigarrow_{\textsc{appL}}$ case.

(2) t_1 has the form $[v]$ for some value v, and we can reduce:

$$\frac{\mathsf{dom}(H') \equiv \mathsf{refs}(v) \qquad (H'', \theta, \overline{id}) = \mathsf{copy}(H') \qquad y \# \{H, v, t\}}{H, H' \vdash \textbf{clone}\ [v]_r\ \textbf{as}\ x\ \textbf{in}\ t \rightsquigarrow_s H, H', H'', y \mapsto_s \textbf{pack}\ \langle \overline{id}, {*}(\theta(v)) \rangle \vdash t[y/x]} \rightsquigarrow_{\textsc{clone}\beta}$$

Here, all of the references from the original heap H (separated out as H') are copied and given a new identifier, to form H''. In other words, for every $ref \mapsto_p id$, $id \mapsto v$ in H, there now exists a new ref' and id' in H'' such that $ref' \mapsto_p id'$, $id' \mapsto v$.

Therefore, for all $id \in \mathsf{dom}(H)$ we have:

$$\sum_{\substack{\forall ref \in \mathsf{refs}(t). \\ ref \mapsto_p id \in H}} p = 1 \implies \sum_{\substack{\forall ref \in \mathsf{refs}(t'). \\ ref \mapsto_{p'} id \in H'}} p' = 1$$

because none of the references in the preexisting heap have been modified in the new heap, and there are no new references with the same identifiers as any references appearing in H'' have fresh identifiers.

We also have that for all $id' \in \mathrm{dom}(H') \wedge id' \notin \mathrm{dom}(H)$:

$$\sum_{\substack{\forall ref' \in \mathsf{refs}(t'). \\ ref' \mapsto_q id' \in H'}} q \quad = 1$$

because any $ref' \notin \mathrm{dom}(H)$ must be in H'', and so it matches up with another reference ref appearing in H, for which we have $\sum_{\substack{\forall ref \in \mathsf{refs}(t). \\ ref \mapsto_p id \in H}} p \quad = 1$ by the above argument.

- (withBorrow)

$$\frac{\Gamma_1 \vdash t_1 : *A \qquad \Gamma_2 \vdash t_2 : \&_1 A \multimap \&_1 B}{\Gamma_1 + \Gamma_2 \vdash \textbf{withBorrow}\ t_1\ t_2 : *B}\ \textsc{with}\&$$

with a heap H such that $H \bowtie (\Gamma_1 + \Gamma_2)$ and reduction:

$$H \vdash \textbf{withBorrow}\ f\ t \rightsquigarrow_s H' \vdash t''$$

which has three possible derivations:

(1)

$$\frac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash \textbf{withBorrow}\ t_1\ t_2 \rightsquigarrow_s H' \vdash \textbf{withBorrow}\ t_1'\ t_2}\ \rightsquigarrow_{\textsc{with\&L}}$$

Here, induction on t_1 provides the goal, following the same scheme of generalising the inductive evidence as for the $\rightsquigarrow_{\textsc{appL}}$ case.

(2) f has the form $(\lambda x.t_1)$, and we can reduce:

$$\frac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \textbf{withBorrow}\ (\lambda x.t_1)\ t_2 \rightsquigarrow_s H' \vdash \textbf{withBorrow}\ (\lambda x.t_1)\ t_2'}\ \rightsquigarrow_{\textsc{with\&R}}$$

Here, induction on t_2 provides the goal, following the same scheme of generalising the inductive evidence as for the $\rightsquigarrow_{\textsc{appL}}$ case.

(3) As above, but t also has the form $(*v)$, and we can reduce:

$$\frac{y \# \{H, v, t\}}{H \vdash \textbf{withBorrow}\ (\lambda x.t)\ (*v) \rightsquigarrow_s H, y \mapsto_s (*v) \vdash \textbf{unborrow}\ t[y/x]}\ \rightsquigarrow_{\textsc{with\&}}$$

Here, the goal is trivial since the heap H is preserved modulo new variable bindings.

- (split)

$$\frac{\Gamma \vdash t : \&_p A}{\Gamma \vdash \textbf{split}\ t : \&_{\frac{p}{2}} A \otimes \&_{\frac{p}{2}} A}\ \textsc{split}$$

with a heap H such that $H \bowtie \Gamma$ and reduction:

$$H \vdash \textbf{split}\ t \rightsquigarrow_s H' \vdash t''$$

which has four possible derivations:

(1)

$$\frac{H \vdash t \rightsquigarrow_s H' \vdash t'}{H \vdash \textbf{split}\ t \rightsquigarrow_s H' \vdash \textbf{split}\ t'}\ \rightsquigarrow_{\textsc{split}}$$

Here, induction provides the goal, following the same scheme of generalising the inductive evidence as for the $\rightsquigarrow_{\textsc{appL}}$ case.

(2) t has the form $(*\mathit{ref})$ such that the typing is:

$$\dfrac{[\Gamma], \mathit{ref} : \mathsf{Ref}_{id}\ A \vdash (*\mathit{ref}) : \&_p(\mathsf{Ref}_{id}\ A)\ {*\textsc{ref}^*}}{[\Gamma], \mathit{ref} : \mathsf{Ref}_{id}\ A \vdash \mathbf{split}\ (*\mathit{ref}) : \&_{\frac{p}{2}}(\mathsf{Ref}_{id}\ A) \otimes \&_{\frac{p}{2}}(\mathsf{Ref}_{id}\ A)}\ \textsc{split}$$

and we have reduction:

$$\dfrac{\mathit{ref}_1 \# H \qquad \mathit{ref}_2 \# H}{H, \mathit{ref} \mapsto_p id, id \mapsto v \vdash \mathbf{split}\ (*\mathit{ref}) \rightsquigarrow_s H, \mathit{ref}_1 \mapsto_{\frac{p}{2}} id, \mathit{ref}_2 \mapsto_{\frac{p}{2}} id, id \mapsto v \vdash (*\mathit{ref}_1, *\mathit{ref}_2)}\ \rightsquigarrow_{\textsc{splitRef}}$$

Thus for $id' \in \mathrm{dom}(H, id \mapsto v, \mathit{ref} \mapsto_p id)$ and $\mathit{ref}' \in \mathrm{refs}(\mathbf{split}\ (*\mathit{ref}))$ we assume the antecedent $\sum_{\mathit{ref}' \mapsto_{p'} id' \in H, id \mapsto v, \mathit{ref} \mapsto_p id}\, p = 1$.
Since there is only one reference in the term then we have $\mathit{ref}' = \mathit{ref}$ and $id' = id$ and $p' = p$. However in the output heap we now have $\mathit{ref}_1 \mapsto_{\frac{p}{2}} id$ and $\mathit{ref}_2 \mapsto_{\frac{p}{2}} id$ in the output heap and thus:

$$\sum_{\mathit{ref} \mapsto_p id \in H'} p = \frac{p}{2} + \frac{p}{2} = p = 1$$

Satisfying the goal.

(3) t has the form $(*(v, w))$ with reduction:

$$\dfrac{\begin{array}{l} H \vdash \mathbf{split}\ (*v) \rightsquigarrow_s H' \vdash (*v_1, *v_2) \\ H' \vdash \mathbf{split}\ (*w) \rightsquigarrow_s H'' \vdash (*w_1, *w_2) \end{array}}{H \vdash \mathbf{split}\ (*(v, w)) \rightsquigarrow_s H'' \vdash (*(v_1, w_1), *(v_2, w_2))}\ \rightsquigarrow_{\textsc{split}\otimes}$$

Induction over the two premises gives us the result here, by threading the theorem's implications from H to H' via the first premise and then H' to H'' via the second.

- (join)

$$\dfrac{\Gamma_1 \vdash t_1 : \&_p A \qquad \Gamma_2 \vdash t_2 : \&_q A \qquad p + q \leq 1}{\Gamma_1 + \Gamma_2 \vdash \mathbf{join}\ t_1\ t_2 : \&_{p+q} A}\ \textsc{join}$$

with a heap H such that $H \bowtie (\Gamma_1 + \Gamma_2)$ and reduction:

$$H \vdash \mathbf{join}\ t_1\ t_2 \rightsquigarrow_s H' \vdash t''$$

which has four possible derivations:

(1)

$$\dfrac{H \vdash t_1 \rightsquigarrow_s H' \vdash t_1'}{H \vdash \mathbf{join}\ t_1\ t_2 \rightsquigarrow_s H' \vdash \mathbf{join}\ t_1'\ t_2}\ \rightsquigarrow_{\textsc{joinL}}$$

Here, induction over t_1 provides the goal, following the same scheme of generalising the inductive evidence as for the $\rightsquigarrow_{\textsc{appL}}$ case.

(2) $t_1 = v$ for some value v, and we can reduce:

$$\dfrac{H \vdash t_2 \rightsquigarrow_s H' \vdash t_2'}{H \vdash \mathbf{join}\ v\ t_2 \rightsquigarrow_s H' \vdash \mathbf{join}\ v\ t_2'}\ \rightsquigarrow_{\textsc{joinR}}$$

Here, induction over t_2 provides the goal, following the same scheme of generalising the inductive evidence as for the $\rightsquigarrow_{\textsc{appL}}$ case.

(3) As above, but t_2 has the form $(*\mathit{ref}_2)$, which restricts v to have the form $(*\mathit{ref}_1)$ such that the typing is:

$$\dfrac{[\Gamma], \mathit{ref}_1 : \mathsf{Ref}_{id}\ A \vdash (*\mathit{ref}_1) : \&_p(\mathsf{Ref}_{id}\ A)\ \textsc{ref+nec} \qquad [\Gamma], \mathit{ref}_2 : \mathsf{Ref}_{id}\ A \vdash (*\mathit{ref}_2) : \&_q(\mathsf{Ref}_{id}\ A)\ \textsc{ref+nec}}{[\Gamma], \mathit{ref}_1 : \mathsf{Ref}_{id}\ A, \mathit{ref}_2 : \mathsf{Ref}_{id}\ A \vdash \mathbf{join}\ (*\mathit{ref}_1)\ (*\mathit{ref}_2) : \&_{p+q}(\mathsf{Ref}_{id}\ A)}\ \textsc{join}$$

and we have reduction:

$$\frac{ref\#H}{H, ref_1 \mapsto_p id, ref_2 \mapsto_q id, id \mapsto v \vdash \textbf{join}\ (*ref_1)\ (*ref_2) \ \rightsquigarrow_s\ H, ref \mapsto_{(p+q)} id, id \mapsto v \vdash *ref} \rightsquigarrow_{\text{JOINREF}}$$

Here, for all $id \in \mathrm{dom}(H, id \mapsto v, ref_1 \mapsto_p id, ref_2 \mapsto_q id)$ and all $ref' \in \mathrm{refs}(\textbf{join}\ (*ref_1)\ (*ref_2))$ then we assume the antecedent $\sum_{ref' \mapsto_p id \in H} p = 1$ then we have: $\sum_{\substack{\forall ref\ \in\ \mathrm{refs}(t').\\ ref \mapsto_{p'} id\ \in\ H'}} = p+q = 1$.

Part 2 is then follows trivially as there are no new resources created.

(4) t_2 has the form $(*(v_2, w_2))$, which restricts t_1 to have the form $(*(v_1, w_1))$, allowing the reduction:

$$\frac{\begin{array}{c} H \vdash \textbf{join}\ (*v_1)\ (*v_2) \ \rightsquigarrow_s\ H' \vdash *v \\ H' \vdash \textbf{join}\ (*w_1)\ (*w_2) \ \rightsquigarrow_s\ H'' \vdash *w \end{array}}{H \vdash \textbf{join}\ (*(v_1, w_1))\ (*(v_2, w_2)) \ \rightsquigarrow_s\ H'' \vdash *(v, w)} \rightsquigarrow_{\text{JOIN}\otimes}$$

Induction over the two premises gives us the result here, by threading the theorem's implications from H to H' via the first premise and then H' to H'' via the second.

- (push)

$$\frac{\Gamma \vdash t : \&_p (A \otimes B)}{\Gamma \vdash \textbf{push}\, t : (\&_p A) \otimes (\&_p B)} \text{ PUSH}$$

with a heap H such that $H \bowtie \Gamma$ and reduction:

$$H \vdash \textbf{push}\, t \ \rightsquigarrow_s\ H' \vdash t''$$

which has three possible derivations:

(1)

$$\frac{H \vdash t \ \rightsquigarrow_s\ H' \vdash t'}{H \vdash \textbf{push}\, t \ \rightsquigarrow_s\ H' \vdash \textbf{push}\, t'} \rightsquigarrow_{\text{PUSH}}$$

Here, induction provides the goal, following the same scheme of generalising the inductive evidence as for the $\rightsquigarrow_{\text{APPL}}$ case.

(2) t has the form $*(v_1, v_2)$ and we can reduce:

$$\frac{}{H \vdash \textbf{push}\, (*(v_1, v_2)) \ \rightsquigarrow_s\ H \vdash (*v_1, *v_2)} \rightsquigarrow_{\text{PUSH}*}$$

Here, the goal is trivial since the heap H is preserved.

- (pull)

$$\frac{\Gamma \vdash t : (\&_p A) \otimes (\&_p B)}{\Gamma \vdash \textbf{pull}\, t : \&_p (A \otimes B)} \text{ PULL}$$

with a heap H such that $H \bowtie \Gamma$ and reduction:

$$H \vdash \textbf{pull}\, t \ \rightsquigarrow_s\ H' \vdash t''$$

which has three possible derivations:

(1)

$$\frac{H \vdash t \ \rightsquigarrow_s\ H' \vdash t'}{H \vdash \textbf{pull}\, t \ \rightsquigarrow_s\ H' \vdash \textbf{pull}\, t'} \rightsquigarrow_{\text{PULL}}$$

Here, induction provides the goal, following the same scheme of generalising the inductive evidence as for the $\rightsquigarrow_{\text{APPL}}$ case.

(2) t has the form $(*v_1, *v_2)$ and we can reduce:

$$\frac{}{H \vdash \textbf{pull}\ (*v_1, *v_2) \ \rightsquigarrow_s\ H \vdash *(v_1, v_2)} \rightsquigarrow_{\text{PULL*}}$$

Here, the goal is trivial since the heap H is preserved.

- (newArray) (readArray) (writeArray) (deleteArray) (newRef) (swapRef) (freezeRef) (readRef) All trivial as they have no reductions.
- (pack)

$$\frac{\Gamma \vdash t : A \qquad id \notin \mathsf{dom}(\Gamma)}{\Gamma \vdash \textbf{pack}\ \langle id', t\rangle : \exists id.A[id/id']} \text{PACK}$$

with a heap H such that $H \bowtie \Gamma$ and reduction:

$$H \vdash \textbf{pack}\ \langle id, t\rangle \ \rightsquigarrow_s\ H' \vdash t''$$

which has one possible derivation:

$$\frac{H \vdash t \ \rightsquigarrow_s\ H \vdash t'}{H \vdash \textbf{pack}\ \langle id, t\rangle \ \rightsquigarrow_s\ H \vdash \textbf{pack}\ \langle id, t'\rangle} \rightsquigarrow_{\text{PACK}}$$

Here, the goal is trivial since the heap H is preserved.

- (unpack)

$$\frac{\begin{array}{c}\Gamma_1 \vdash t_1 : \exists id.A \\ \Gamma_2, id, x : A \vdash t_2 : B \qquad id \notin \mathsf{fv}(B)\end{array}}{\Gamma_1 + \Gamma_2 \vdash \textbf{unpack}\ \langle id, x\rangle = t_1\ \textbf{in}\ t_2 : B} \text{UNPACK}$$

with a heap H such that $H \bowtie \Gamma$ and reduction:

$$H \vdash \textbf{unpack}\ \langle id, x\rangle = t_1\ \textbf{in}\ t_2 \ \rightsquigarrow_s\ H' \vdash t''$$

which has two possible derivations:

(1)

$$\frac{y\#\{H, v, t\}}{H \vdash \textbf{unpack}\ \langle id, x\rangle = \textbf{pack}\ \langle id', v\rangle\ \textbf{in}\ t \ \rightsquigarrow_s\ H, y \mapsto_r v \vdash t[y/x]} \rightsquigarrow_{\exists\beta}$$

Here, the heap is preserved with the exception of y. Since no references are affected, the goal is achieved directly.

(2)

$$\frac{H \vdash t_1 \ \rightsquigarrow_s\ H \vdash t_1'}{H \vdash \textbf{unpack}\ \langle id, x\rangle = t_1\ \textbf{in}\ t_2 \ \rightsquigarrow_s\ H \vdash \textbf{unpack}\ \langle id, x\rangle = t_1'\ \textbf{in}\ t_2} \rightsquigarrow_{\text{UNPACK}}$$

Here, the goal is trivial since the heap H is preserved.

□

Theorem D.4 (Multi-reduction borrow safety). *For a well-typed term $\Gamma \vdash t : A$ and all Γ_0, s, and H such that $H \bowtie (\Gamma_0 + s \cdot \Gamma)$, and multi-step reduction $H \vdash t \ \Rightarrow_s\ H' \vdash v$, then for all $id \in \mathsf{dom}(H)$:*

$$\sum_{\substack{\forall ref \in \mathsf{refs}(t).\\ ref \mapsto_p id \in H}} p \ = 1 \implies \exists! ref'.ref' \mapsto_1 id \in H'$$

i.e., for all resources with identifier id in the incoming heap and all references in the term pointing to this resource, if the sum of all permissions pointing to this resource are 1 *in the incoming heap then their total permission of* 1 *is preserved from the incoming heap to the resulting term, with this permission now contained in a single reference ref′.*

Furthermore, any new references in the final term should uniquely point to an identifier, and thus have permission 1*. That is, for all* $id' \in \mathrm{dom}(H') \wedge id' \notin \mathrm{dom}(H)$ *then:*

$$\forall ref \in \mathrm{refs}(v).\ \exists! ref'.\ \ ref' \mapsto_1 id' \in H'$$

Proof. By induction on the structure of the multi-reduction $H \vdash t_1 \ \Rightarrow_s \ H' \vdash v$.

- (refl)

$$\frac{}{H \vdash v \ \Rightarrow_s \ H \vdash v}\ \textsc{refl}$$

Since v is a value which we know has type $*A$, then by the unique value lemma there are two possibilities for the form of v.

(1) $A = Res_{id}\ A'$, and so v has the form $*ref$. This restricts the typing as follows:

$$\frac{}{0 \cdot \Gamma, ref : Res_{id}\ A \vdash *ref : *(Res_{id}\ A)}\ *\textsc{ref}^*$$

Then we also have a heap H such that $H \bowtie (0 \cdot \Gamma, ref \ : \ Res_{id}\ A)$ which by inversion of heap-compatibility for references implies that there exists a subheap H_1 such that $H = H_1, ref \mapsto_1 id, id \mapsto v$.

As we have a single reference in the heap annotated with 1, both the premise and the goal of the implication in the theorem hold.

(2) $A = A' \otimes B$, and so v has the form (v_1, v_2). This restricts the typing to two possible derivations:

$$\frac{\dfrac{\gamma_1 \vdash v_1 : A' \quad \gamma_2 \vdash v_2 : B}{\gamma_1 + \gamma_2 \vdash (v_1, v_2) : A' \otimes B}\ \otimes_I}{\gamma_1 + \gamma_2 \vdash *(v_1, v_2) : *(A' \otimes B)}\ \textsc{nec} \qquad \frac{\dfrac{\dfrac{\gamma_1 \vdash v_1 : A'}{\gamma_1 \vdash *v_1 : *A'}\ \textsc{nec} \quad \dfrac{\gamma_2 \vdash v_2 : B}{\gamma_2 \vdash *v_2 : *B}\ \textsc{nec}}{\gamma_1 + \gamma_2 \vdash (*v_1, *v_2) : *A' \otimes *B}\ \otimes_I}{\gamma_1 + \gamma_2 \vdash *(v_1, v_2) : *(A' \otimes B)}\ \textsc{pull}$$

In either case, by the unique value lemma we know that v_1 and v_2 are both restricted: they can either be of the form a, meaning that one of the types in the product is $Res_{id}\ A$, or they can be of the form (v_3, v_4), meaning that one of the types is $A'' \otimes B'$.

Proceed by inspecting each of v_1 and v_2 in turn. If the value is of the form ref, then as above by inversion of heap compatibility the heap must contain a unique reference $ref \mapsto_1 id, id \mapsto v$, which satisfies the theorem as it is annotated with 1.

If the value is of the form (v_3, v_4), then we can restrict the typing of this subpart of the derivation by exactly the above argument, and then inspect the form of v_3 and v_4 using the unique value lemma following the same logic. This proceeds inductively; as typing derivations are finite trees, this must eventually terminate in an reference which satisfies the theorem at every leaf of the tree.

- (ext)

$$\frac{H \vdash t_1 \ \rightsquigarrow_s \ H' \vdash t_2 \qquad H' \vdash t_2 \ \Rightarrow_s \ H'' \vdash t_3}{H \vdash t_1 \ \Rightarrow_s \ H'' \vdash t_3}\ \textsc{ext}$$

First, consider the first premise, which is a single-step reduction of the form $H \vdash t_1 \ \rightsquigarrow_s \ H' \vdash t_2$.

From Theorem D.3, we know that for all Γ_0 and heaps H such that $H \bowtie (\Gamma_0 + s \cdot \Gamma)$ then for all $id \in \mathrm{dom}(H)$:

$$\sum_{\substack{\forall ref \in \mathrm{refs}(t). \\ ref \mapsto_p id \in H}} p = 1 \implies \sum_{\substack{\forall ref \in \mathrm{refs}(t). \\ ref \mapsto_p id \in H'}} p \in \{0, 1\}$$

and also that for all $id' \in \mathrm{dom}(H') \wedge id' \notin \mathrm{dom}(H)$ (new resources) we have:

$$\sum_{\substack{\forall ref' \in \mathrm{refs}(t').\\ ref' \mapsto_q id' \in H'}} q = 1$$

Now, we induct over the second premise, which is a multi-reduction of the form $H' \vdash t_2 \Rightarrow_s H'' \vdash t_3$. Induction using the present theorem tells us that for all Γ_0 and H such that $H \bowtie (\Gamma_0 + s \cdot \Gamma)$, then for all $id \in \mathrm{dom}(H)$ we have:

$$\sum_{\substack{\forall ref \in \mathrm{refs}(t).\\ ref \mapsto_p id \in H'}} p = 1 \implies \exists! ref'.ref' \mapsto_1 id \in H''$$

From our knowledge about the first premise (the single-step reduction), there are two cases.

- $\sum_{\substack{\forall ref \in \mathrm{refs}(t).\\ ref \mapsto_p id \in H}} p = \sum_{\substack{\forall ref \in \mathrm{refs}(t).\\ ref \mapsto_p id \in H'}} p = 1.$
  Then we know that fractions must sum to 1 for both references preserved from t_1 and also new references preserved from t_2, but via the implication we obtained from induction on the second premise, we know $\exists! ref'.ref' \mapsto_1 id \in H''$, which is exactly the required result.
- $\sum_{\substack{\forall ref \in \mathrm{refs}(t).\\ ref \mapsto_p id \in H'}} p = 0.$
  Then we only need to concern ourselves with new references preserved from t_2, but via the same implication we obtained via induction on the second premise, we know $\exists! ref'.ref' \mapsto_1 id \in H''$, again exactly as required.

□

Corollary D.5 (Uniqueness). *For a well-typed term $\Gamma \vdash t : *A$ and all Γ_0, s, and H such that $H \bowtie (\Gamma_0 + s \cdot \Gamma)$ and multi-reduction to a value $H \vdash t \Rightarrow_s H' \vdash *v$, for all $id \in \mathrm{dom}(H)$ then:*

$$\begin{aligned} &\forall ref \in \mathrm{refs}(t).(ref \mapsto_1 id \in H \implies ref \mapsto_1 id \in H') \\ \wedge\ & \forall id' \in \mathrm{dom}(H') \wedge id' \notin \mathrm{dom}(H).\ \forall ref \in \mathrm{refs}(v).\ \exists! ref'.ref \mapsto_1 id' \in H' \end{aligned}$$

i.e., any references contributing to the final term that are unique in the incoming heap stay unique in the resulting term, and any new references contributing to the final term are also unique.

Proof. Follows directly from Lemma D.4, in the subcase where only one reference exists in the initial heap (since $\sum_{ref \mapsto_p id \in H} p = 1$ must hold if there exists a single reference such that $ref \mapsto_1 id \in H$). □

E SOUNDNESS OF HEAP MODEL WRT. EQUATIONAL THEORY

Theorem E.1 (Soundness with respect to the equational theory). *For all t_1, t_2 such that $\Gamma \vdash t_1 : A$ and $\Gamma \vdash t_2 : A$ and $t_1 \equiv t_2$ and given H such that $H \bowtie \Gamma$, there exist multi-reductions to values that are equal under full β-reduction and evaluating any references to the value they point to in the resulting heaps (written $H'(v_1)$ and $H'(v_2)$):*

$$H \vdash t_1 \Rightarrow_1 H' \vdash v_1 \quad \wedge \quad H \vdash t_2 \Rightarrow_1 H'' \vdash v_2 \quad \wedge \quad H'(v_1) \equiv H''(v_2)$$

Proof. • (β_*)

$$\frac{}{\textbf{clone}\ (\textbf{share}\ v)\ \textbf{as}\ x\ \textbf{in}\ t' \equiv t'[\textbf{pack}\ \langle \overline{id}, v\rangle / x]}\ \beta_*$$

With typing derivation for the LHS:

$$\dfrac{\dfrac{\Gamma_1, \overline{id} \vdash v : *A}{\Gamma_1, \overline{id} \vdash \textbf{share}\ v : \Box_r A}\ \text{SHARE} \qquad \mathsf{noIDs}(\Gamma_1) \quad \Gamma_2, x : \exists id'.*(A[\overline{id'}/\overline{id}]) \vdash t : \Box_r B \quad 1 \sqsubseteq r}{(\Gamma_1 + \Gamma_2), \overline{id} \vdash \textbf{clone}\ (\textbf{share}\ v)\ \textbf{as}\ x\ \textbf{in}\ t : \Box_r B}\ \text{CLONE}$$

And typing derivation for the RHS:

$$(\Gamma_1 + \Gamma_2), \overline{id} \vdash t[\textbf{pack}\ \langle \overline{id}, v \rangle / x] : \Box_r B$$

By the value lemma (Lemma C.1) we know that $v = *v'$ therefore we know that we can perform the following reduction (where we elide the output binding context and usage context as they are not needed in the proof):

$$\begin{array}{rcl} & & H_1, H_2 \vdash \textbf{clone}\ (\textbf{share}\ (*v'))\ \textbf{as}\ x\ \textbf{in}\ t \\ \rightsquigarrow_{\text{CLONE}} + \rightsquigarrow_{\text{SHARE}\beta} & \rightarrow & [H_1]_0, H_2 \vdash \textbf{clone}\ [v']\ \textbf{as}\ x\ \textbf{in}\ t \\ \rightsquigarrow_{\text{CLONE}\beta} & \rightsquigarrow & [H_1]_0, H_2, H_3, x \mapsto_r \textbf{pack}\ \langle \overline{id}, \theta(v) \rangle \vdash t \end{array}$$

where $\mathsf{dom}(H_1) \equiv \mathsf{refs}(v)$ and $(H_3, \theta, \overline{id}) = \mathsf{copy}(H_1)$

- $(*\text{assoc})$

$$\dfrac{x \# t_3}{\textbf{clone}\ t_1\ \textbf{as}\ x\ \textbf{in}\ (\textbf{clone}\ t_2\ \textbf{as}\ y\ \textbf{in}\ t_3) \equiv \textbf{clone}\ (\textbf{clone}\ t_1\ \textbf{as}\ x\ \textbf{in}\ t_2)\ \textbf{as}\ y\ \textbf{in}\ t_3}\ *\text{ASSOC}$$

With typing for the LHS:

$$\dfrac{\Gamma_1, \overline{id_1} \vdash t_1 : \Box_{r_1} A_1 \qquad \dfrac{\Gamma_2, \overline{id_2}, x : \exists \overline{id_1'}.*(A_1[\overline{id_1'}/\overline{id_1}]) \vdash t_2 : \Box_{r_2} A_2 \quad \Gamma_3, y : \exists \overline{id_2'}.*(A_2[\overline{id_2'}/\overline{id_2}]) \vdash t_3 : \Box_{r_3} A_3}{\Gamma_2, x : \exists \overline{id_1'}.*(A_1[\overline{id_1'}/\overline{id_1}]) + \Gamma_3 \vdash \textbf{clone}\ t_2\ \textbf{as}\ y\ \textbf{in}\ t_3 : \Box_{r_3} A_3}\ \text{CLONE}}{(\Gamma_1 + \Gamma_2 + \Gamma_3), \overline{id_1}, \overline{id_2} \vdash \textbf{clone}\ t_1\ \textbf{as}\ x\ \textbf{in}\ (\textbf{clone}\ t_2\ \textbf{as}\ y\ \textbf{in}\ t_3) : \Box_{r_3} A_3}\ \text{CLONE}$$

where $\mathsf{noIDs}(\Gamma_1)$ and $\mathsf{noIDs}(\Gamma_2)$ and $1 \sqsubseteq r_1$ and $1 \sqsubseteq r_2$.
And with RHS typing, with the same conditions:

$$\dfrac{\dfrac{\Gamma_1, \overline{id_1} \vdash t_1 : \Box_{r_1} A_1 \quad \Gamma_2, \overline{id_2}, x : \exists \overline{id_1'}.*(A_1[\overline{id_1'}/\overline{id_1}]) \vdash t_2 : \Box_{r_2} A_2}{(\Gamma_1 + \Gamma_2), \overline{id_1}, \overline{id_2} \vdash \textbf{clone}\ t_1\ \textbf{as}\ x\ \textbf{in}\ t_2 : \Box_{r_2} A_2}\ \text{CLONE} \qquad \Gamma_3, y : \exists \overline{id_2'}.*(A_2[\overline{id_2'}/\overline{id_2}]) \vdash t_3 : \Box_{r_3} A_3}{(\Gamma_1 + \Gamma_2 + \Gamma_3), \overline{id_1}, \overline{id_2} \vdash \textbf{clone}\ (\textbf{clone}\ t_1\ \textbf{as}\ x\ \textbf{in}\ t_2)\ \textbf{as}\ y\ \textbf{in}\ t_3 : \Box_{r_3} A_3}\ \text{CLONE}$$

There are four possibilities depending on the reduction of t_1 and t_2.

(1) Divergence: $t_1 \rightarrow^{\omega}$ (i.e., t_1 diverges). In which case then $H \vdash \textbf{clone}\ t_1\ \textbf{as}\ x\ \textbf{in}\ (\textbf{clone}\ t_2\ \textbf{as}\ y\ \textbf{in}\ t_3) \rightsquigarrow_{\text{CLONE}}{}^{\omega}$ (diverging) and also $H \vdash \textbf{clone}\ (\textbf{clone}\ t_1\ \textbf{as}\ x\ \textbf{in}\ t_2)\ \textbf{as}\ y\ \textbf{in}\ t_3 \rightsquigarrow_{\text{CLONE}}{}^{\omega}$ and so the equation is trivially satisfied as both sides diverge.
If t_1 converges, but t_2 diverges then both sides diverge by similar reasoning. Similarly if both diverge then overall both sides diverge.

(2) Convergence: t_1 reduces to a value v_1 and t_2 reduces to a value v_2. By the typing and the value lemma (Lemma C.1) then $\exists v_1'.\ v_1 = [v_1']$ and then $\exists v_2'.\ v_2 = [v_2']$.
Then we can reduce as follows on the LHS:

$$\begin{array}{rcll} & & H \vdash \textbf{clone}\ t_1\ \textbf{as}\ x\ \textbf{in}\ (\textbf{clone}\ t_2\ \textbf{as}\ y\ \textbf{in}\ t_3) & \\ & \rightsquigarrow * & H' \vdash \textbf{clone}\ [v_1']\ \textbf{as}\ x\ \textbf{in}\ (\textbf{clone}\ t_2\ \textbf{as}\ y\ \textbf{in}\ t_3) & \\ \rightsquigarrow_{\text{CLONE}\beta} & \rightsquigarrow & H_1, H_1', H_1'', x \mapsto_r \textbf{pack}\ \langle \overline{id_1}, *\theta(v_1) \rangle \vdash \textbf{clone}\ t_2\ \textbf{as}\ y\ \textbf{in}\ t_3 & \mathsf{dom}(H_1') \equiv \mathsf{refs}(v_1) \ \wedge\ (H_1'', \theta, \overline{id_1}) = \mathsf{copy}(H_1') \\ & \rightsquigarrow * & H'' \vdash \textbf{clone}\ [v_2]\ \textbf{as}\ y\ \textbf{in}\ t_3 & \\ \rightsquigarrow_{\text{CLONE}\beta} & \rightsquigarrow & H_2, H_2', H_2'', y \mapsto_r \textbf{pack}\ \langle \overline{id_2}, *\theta'(v_2) \rangle \vdash t_3 & \mathsf{dom}(H_2') \equiv \mathsf{refs}(v_2) \ \wedge\ (H_2'', \theta, \overline{id_2}) = \mathsf{copy}(H_2') \end{array}$$

and on the RHS:

$$
\begin{array}{rll}
 & H \vdash \textbf{clone}\,(\textbf{clone}\ t_1\ \textbf{as}\ x\ \textbf{in}\ t_2)\ \textbf{as}\ y\ \textbf{in}\ t_3 & \\
\rightsquigarrow * & H' \vdash \textbf{clone}\,(\textbf{clone}\ [v_1]\ \textbf{as}\ x\ \textbf{in}\ t_2)\ \textbf{as}\ y\ \textbf{in}\ t_3 & \\
\rightsquigarrow & H_1, H_1', H_1'', x \mapsto_r \textbf{pack}\ \langle \overline{id_1}, *\theta(v_1)\rangle \vdash \textbf{clone}\ t_2\ \textbf{as}\ y\ \textbf{in}\ t_3 & \mathsf{dom}(H_1') \equiv \mathsf{refs}(v_1)\ \wedge\ (H_1'', \theta, \overline{id_1'}) = \mathsf{copy}(H_1') \\
\rightsquigarrow * & H'' \vdash \textbf{clone}\ [v_2]\ \textbf{as}\ y\ \textbf{in}\ t_3 & \\
\rightsquigarrow_{\textsc{clone}\beta} & H_2, H_2', H_2'', y \mapsto_r \textbf{pack}\ \langle \overline{id_2}, *\theta'(v_2)\rangle \vdash t_3 & \mathsf{dom}(H_2') \equiv \mathsf{refs}(v_2)\ \wedge\ (H_2'', \theta, \overline{id_2'}) = \mathsf{copy}(H_2')
\end{array}
$$

matching in both sides.

- (&unit)

$$
\frac{}{\textbf{withBorrow}\ (\lambda x.x)\ t \equiv t}\ \&\textsc{unit}
$$

With typing derivation for the LHS:

$$
\dfrac{\Gamma_1 \vdash t : *A \qquad \dfrac{\dfrac{}{0 \cdot \Gamma_2, x : \&_1 A \vdash x : \&_1 A}\ \textsc{var}}{0 \cdot \Gamma_2 \vdash (\lambda x.x) : \&_1 A \multimap \&_1 A}\ \textsc{abs}}{\Gamma_1 + 0 \cdot \Gamma_2 \vdash \textbf{withBorrow}\ (\lambda x.x)\ t : *A}\ \textsc{with}\&
$$

and typing derivation for the RHS:

$$
\Gamma_1 + 0 \cdot \Gamma_2 \vdash t : *A
$$

There are two possibilities depending on the reduction of t.

(1) If reduction of t diverges, then reduction on the LHS also diverges, since we repeatedly apply the $\rightsquigarrow_{\textsc{with}\&\textsc{R}}$ rule. Hence, the equation is trivially satisfied as both sides diverge.

(2) Otherwise, t reduces to a value v. By the typing and the value lemma (Lemma C.1) then $\exists v'.\ v = *v'$.
Then we can reduce as follows on the LHS:

$$
\begin{array}{rrl}
 & & H \vdash \textbf{withBorrow}\ (\lambda x.x)\ t \\
 & \rightsquigarrow * & H' \vdash \textbf{withBorrow}\ (\lambda x.x)\ (*v') \\
\rightsquigarrow_{\textsc{with}\&} & \rightsquigarrow & H', y \mapsto_r *v' \vdash \textbf{unborrow}\ (x[y/x]) \\
 & \rightsquigarrow * & H', y \mapsto_r *v' \vdash \textbf{unborrow}\ y \\
\rightsquigarrow_{\textsc{un}\&} & \rightsquigarrow & H', y \mapsto_r *v' \vdash y \\
 & \rightsquigarrow * & H', y \mapsto_r *v' \vdash *v'
\end{array}
$$

and as follows on the RHS:

$$
\begin{array}{rl}
 & H \vdash t \\
\rightsquigarrow * & H' \vdash *v'
\end{array}
$$

- (&assoc)

$$
\frac{}{\textbf{withBorrow}\ (\lambda x.(f\ (g\ x)))\ t \equiv \textbf{withBorrow}\ f\ (\textbf{withBorrow}\ g\ t)}\ \&\textsc{assoc}
$$

If any of the terms f, g or t diverge, then the LHS and RHS both diverge, and so the equation is trivially satisfied.
Otherwise, all three terms must reduce to values v_1, v_2 and v_3.
By typing and the value lemma (Lemma C.1) then $v_1 = \lambda x_1.t_1$, $v_2 = \lambda x_2.t_2$, and $\exists v_3'.\ v_3 = *v_3'$.

Then we can reduce as follows on the LHS:

$$\begin{array}{rll}
 & & H \vdash \textbf{withBorrow}\ (\lambda x.(f\ (g\ x)))\ t \\
 & \rightsquigarrow * & H' \vdash \textbf{withBorrow}\ (\lambda x.(f\ (g\ x)))\ (*v_3') \\
\rightsquigarrow_{\text{WITH\&}} & \rightsquigarrow & H', y{\mapsto_r} * v_3' \vdash \textbf{unborrow}\ ((f\ (g\ x))[y/x]) \\
 & \rightsquigarrow * & H', y{\mapsto_r} * v_3' \vdash \textbf{unborrow}\ (((\lambda x_1.t_1)\ ((\lambda x_2.t_2)\ y))) \\
\rightsquigarrow_\beta & \rightsquigarrow & H', y{\mapsto_r} * v_3', x_2{\mapsto_s}(*v_3') \vdash \textbf{unborrow}\ (((\lambda x_1.t_1)\ t_2))
\end{array}$$

If t_2 diverges, then again both sides diverge. Otherwise, t_2 must reduce to a value v_4, and by the value lemma $\exists v_4'.\ v_4 = (*v_4')$. Then we can continue reducing as follows:

$$\begin{array}{rll}
 & \rightsquigarrow * & H'', y{\mapsto_r} * v_3', x_2{\mapsto_s}(*v_3') \vdash \textbf{unborrow}\ (((\lambda x_1.t_1)\ (*v_4'))) \\
\rightsquigarrow_\beta & \rightsquigarrow & H'', y{\mapsto_r} * v_3', x_2{\mapsto_s}(*v_3'), x_1{\mapsto_{s'}}(*v_4') \vdash \textbf{unborrow}\ (t_1)
\end{array}$$

If t_1 diverges, then again both sides diverge. Otherwise, t_1 must reduce to a value v_5, and by the value lemma $\exists v_5'.\ v_5 = (*v_5')$. Then we can continue reducing as follows:

$$\begin{array}{rll}
 & \rightsquigarrow * & H''', y{\mapsto_r} * v_3', x_2{\mapsto_s}(*v_3'), x_1{\mapsto_{s'}}((*v_4')) \vdash \textbf{unborrow}\ ((*v_5')) \\
\rightsquigarrow_{\text{UN\&}} & \rightsquigarrow & H''', y{\mapsto_r} * v_3', x_2{\mapsto_s}(*v_3'), x_1{\mapsto_{s'}}(*v_4') \vdash *v_5'
\end{array}$$

and on the RHS:

$$\begin{array}{rll}
 & & H \vdash \textbf{withBorrow}\ f\ (\textbf{withBorrow}\ g\ t) \\
 & \rightsquigarrow * & H' \vdash \textbf{withBorrow}\ (\lambda x_1.t_1)\ (\textbf{withBorrow}\ (\lambda x_2.t_2)\ (*v_3')) \\
\rightsquigarrow_{\text{WITH\&}} & \rightsquigarrow & H', y{\mapsto_r} * v_3' \vdash \textbf{withBorrow}\ (\lambda x_1.t_1)\ (\textbf{unborrow}\ (t_2[y/x_2])) \\
 & \sim & H', y{\mapsto_r} * v_3', x_2{\mapsto_s}(*v_3') \vdash \textbf{withBorrow}\ (\lambda x_1.t_1)\ (\textbf{unborrow}\ (t_2))
\end{array}$$

If t_2 diverges, then again both sides diverge. Otherwise, t_2 must reduce to a value v_4, and by the value lemma $\exists v_4'.\ v_4 = (*v_4')$. Then we can continue reducing as follows:

$$\begin{array}{rll}
 & \rightsquigarrow * & H'', y{\mapsto_r} * v_3', x_2{\mapsto_s}(*v_3') \vdash \textbf{withBorrow}\ (\lambda x_1.t_1)\ (\textbf{unborrow}\ ((*v_4'))) \\
\rightsquigarrow_{\text{UN\&}} & \rightsquigarrow & H'', y{\mapsto_r} * v_3, x_2{\mapsto_s}(*v_3') \vdash \textbf{withBorrow}\ (\lambda x_1.t_1)\ (*v_4') \\
\rightsquigarrow_{\text{WITH\&}} & \rightsquigarrow & H'', y{\mapsto_r} * v_3', x_2{\mapsto_s}(*v_3'), z{\mapsto_{r'}} * v_4' \vdash \textbf{unborrow}\ (t_1[z/x_1]) \\
 & \sim & H'', y{\mapsto_r} * v_3', x_2{\mapsto_s}(*v_3'), z{\mapsto_{r'}} * v_4', x_1{\mapsto_{s'}}((*v_4')) \vdash \textbf{unborrow}\ (t_1)
\end{array}$$

If t_1 diverges, then again both sides diverge. Otherwise, t_1 must reduce to a value v_5, and by the value lemma $\exists v_5'.\ v_5 = (*v_5')$. Then we can continue reducing as follows:

$$\begin{array}{rll}
 & \rightsquigarrow * & H'', y{\mapsto_r} * v_3', x_2{\mapsto_s}(*v_3'), z{\mapsto_{r'}} * v_4', x_1{\mapsto_{s'}}((*v_4')) \vdash \textbf{unborrow}\ (*v_5') \\
\rightsquigarrow_{\text{UN\&}} & \rightsquigarrow & H'', y{\mapsto_r} * v_3', x_2{\mapsto_s}(*v_3'), z{\mapsto_{r'}} * v_4', x_1{\mapsto_{s'}}((*v_4')) \vdash *v_5'
\end{array}$$

- $(\textbf{let}\ (x, y) = (\textbf{split}\ t)\ \textbf{in}\ (\textbf{join}\ x\ y)) \equiv t$
  If t diverges, then both sides diverge.
  Otherwise t reduces to a value v which by (Lemma C.1) is over the form $*ref$.
  Thus, we reduce by:

$$\begin{array}{rll}
 & & H \vdash \textbf{let}\ (x, y) = \textbf{split}\ t\ \textbf{in}\ (\textbf{join}\ x\ y) \\
 & \rightsquigarrow * & H', id \mapsto v, ref{\mapsto_p} id \vdash \textbf{let}\ (x, y) = \textbf{split}\ (*ref)\ \textbf{in}\ (\textbf{join}\ x\ y) \\
\rightsquigarrow_{\text{LET}\otimes} + \rightsquigarrow_{\text{SPLITREF}} & \rightsquigarrow^* & H', id \mapsto v, ref_1{\mapsto_{\frac{p}{2}}} id, ref_2{\mapsto_{\frac{p}{2}}} id \vdash \textbf{let}\ (x, y) = (*ref_1, *ref_2)\ \textbf{in}\ (\textbf{join}\ x\ y) \\
\rightsquigarrow_{\otimes\beta} & \rightsquigarrow & H', id \mapsto v, ref_1{\mapsto_{\frac{p}{2}}} id, ref_2{\mapsto_{\frac{p}{2}}} id, x'{\mapsto_1} * ref_1, y'{\mapsto_1} * ref_2 \vdash \textbf{join}\ x'\ y' \\
(\rightsquigarrow_{\text{PRIM}} + \rightsquigarrow_{\text{VAR}})^*2 & \rightsquigarrow^* & H', id \mapsto v, ref_1{\mapsto_{\frac{p}{2}}} id, ref_2{\mapsto_{\frac{p}{2}}} id, x'{\mapsto_1} * ref_1, y'{\mapsto_1} * ref_2 \vdash \textbf{join}\ (*ref_1')\ (*ref_2') \\
\rightsquigarrow_{\text{JOINREF}} & \rightsquigarrow^* & H', id \mapsto v, ref'{\mapsto_p} id, x'{\mapsto_1} * ref_1, y'{\mapsto_1} * ref_2 \vdash *ref'
\end{array}$$

  Evaluating under the heap $(H', id \mapsto v, ref'{\mapsto_p} id, x'{\mapsto_1} * ref_1, y'{\mapsto_1} * ref_2)(*ref') = v$.
  The RHS then reduces to: $H', id \mapsto v, ref{\mapsto_p} id \vdash (*ref)$
  Evaluating under the heap $(H', id \mapsto v, ref{\mapsto_p} id)(ref) = v$, satisfying the goal.

- **split** (**join** t_1 t_2) $\equiv$ (t_1, t_2)
  If either t_1 or t_2 diverge then both sides diverge. Otherwise, we assume reduction to values, using the value lemma to ascertain their form:

$$
\begin{array}{rrl}
 & & H \vdash \textbf{split}\ (\textbf{join}\ t_1\ \ t_2) \\
 & \rightsquigarrow * & H', id \mapsto v, ref_1 \mapsto_p id, ref_2 \mapsto_q id \vdash \textbf{split}\ (\textbf{join}\ (*ref_1)\ (*ref_2)) \\
\rightsquigarrow_{\textsc{joinRef}} & \rightsquigarrow & H', id \mapsto v, ref \mapsto_{(p+q)} id \vdash \textbf{split}\ (*ref) \\
\rightsquigarrow_{\textsc{splitRef}} & \rightsquigarrow & H', id \mapsto v, ref_1' \mapsto_{\frac{(p+q)}{2}} id, ref_2 \mapsto_{\frac{(p+q)}{2}} id \vdash (*ref_1', *ref_2')
\end{array}
$$

  Evaluating under the heap $(H', id \mapsto v, ref_1' \mapsto_{\frac{(p+q)}{2}} id, ref_2 \mapsto_{\frac{(p+q)}{2}} id)((*ref_1', *ref_2')) = (v, v)$.
  The RHS then reduces to $H', id \mapsto v, ref_1 \mapsto_p id, ref_2 \mapsto_q id \vdash (*ref_1, *ref_2)$.
  Evaluating under the heap $(H', id \mapsto v, ref_1 \mapsto_p id, ref_2 \mapsto_q id)((*ref_1, *ref_2)) = (v, v)$, satisfying the goal.

$\square$

REFERENCES

Dominic Orchard, Vilem-Benjamin Liepelt, and Harley Eades III. 2019. Quantitative Program Reasoning with Graded Modal Types. *Proceedings of the ACM on Programming Languages* 3, ICFP (2019), 1–30.