Explicit CN Soundness Proof

Dhruv Makwana

July 26, 2021

1 Weakening

If $\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \sqsubseteq \mathcal{C}';\mathcal{L}';\Phi';\mathcal{R}'$ and $\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash J$ then $\mathcal{C}';\mathcal{L}';\Phi';\mathcal{R}' \vdash J$.

ASSUME: 1. $\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \sqsubseteq \mathcal{C}';\mathcal{L}';\Phi';\mathcal{R}'$.
2. $\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash J$.

PROVE: $\mathcal{C}';\mathcal{L}';\Phi';\mathcal{R}' \vdash J$.

PROOF SKETCH: Consider only the below cases, the rest are functorial in the environment.

⟨1⟩1. CASE: TY_PVAL_VAR_{COMP,LOG}.
PROOF: By WEAK_CONS_{COMP,LOG}, if $x{:}\beta \in \mathcal{C}$ (or $x{:}\beta \in \mathcal{L}$) then $x{:}\beta \in \mathcal{C}'$ (or $x{:}\beta \in \mathcal{L}$).

⟨1⟩2. CASE: TY_PVAL_ERROR, TY_RES_EQ_{POINTSTO,TERM}, TY_RES_CONJ, TY_SPINE_RES_PHI, TY_PE_ASSERTUNDEF, TY_TPVAL_{UNDEF,DONE}, TY_ACTION_{LOAD,STORE,KILL}, TY_MEMOP_PTRVALIDFORDEREF, TY_TVAL_{PHI,UNDEF}.

ASSUME: $\texttt{smt}\,(\Phi \Rightarrow \mathit{term}')$.
PROVE: $\texttt{smt}\,(\Phi' \Rightarrow \mathit{term}')$.

⟨2⟩1. If $\mathit{term} \in \Phi$ then $\mathit{term} \in \Phi'$. PROOF: By WEAK_CONS_PHI.

⟨2⟩2. Any extra constraints in Φ' (by WEAK_SKIP_PHI) would either be irrelevant, redundant, or inconsistent.

⟨2⟩3. In all cases, $\texttt{smt}\,(\Phi' \Rightarrow \mathit{term}')$ as required.

⟨1⟩3. CASE: TY_RES_{EMP,POINTSTO,VAR,SEPCONJ}, TY_SPINE_{EMPTY,RES}, TY_ACTION_CREATE, TY_TVAL_RES, TY_MEMOP_{REL_BINOP, INTFROMPTR, PTRFROMINT, WELLALIGNED, PTRARRAYSHIFT}, TY_TVAL_{I,UNDEF}, TY_SEQ_TE_{LET,LETT,RUN}, TY_IS_TE_LETS.

⟨2⟩1. $\mathcal{R} = \mathcal{R}'$.
PROOF: Only WEAK_CONS_RES_{ANON,NAMED} exist, no WEAK_SKIP_RES.

⟨2⟩2. All the rules are otherwise functorial in $\mathcal{C}, \mathcal{L}, \Phi$, .

⟨2⟩3. So $\mathcal{C}';\mathcal{L}';\Phi';\mathcal{R}' \vdash J$ as required.

2 Substitution

2.1 Weakening for Substitution

Weakening for substitution: as above, but with $J = (\sigma) : (\mathcal{C}''; \mathcal{L}''; \Phi''; \mathcal{R}'')$.

ASSUME: 1. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \sqsubseteq \mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'$.
2. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash (\sigma) : (\mathcal{C}''; \mathcal{L}''; \Phi''; \mathcal{R}'')$.

PROVE: $\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash (\sigma) : (\mathcal{C}''; \mathcal{L}''; \Phi''; \mathcal{R}'')$.

PROOF SKETCH: By weakening and induction over the substitution.

2.2 Substitutions preserve SMT results

ASSUME: 1. $\texttt{smt}\,(\Phi' \Rightarrow \mathit{term})$.
2. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash (\sigma) : (\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}')$.

PROVE: $\texttt{smt}\,(\Phi \Rightarrow \sigma(\mathit{term}))$.

$\langle 1 \rangle 1$. $\texttt{smt}\,(\Phi' \Rightarrow \sigma(\mathit{term}))$.
PROOF: By assumption 1, which means it is true for all (well-typed) instantiations of its free variables.

$\langle 1 \rangle 2$. $\texttt{smt}\,(\Phi \Rightarrow \sigma(\mathit{term}))$.
PROOF: By $\texttt{smt}\,(\Phi \Rightarrow \mathit{term})$ for each $\mathit{term} \in \Phi'$ (from assumption 2) and $\langle 1 \rangle 1$.

2.3 Resource equality is an equivalence relation

PROOF SKETCH: By induction.

2.4 Resource typing subsumption

ASSUME: 1. $\Phi \vdash \mathit{res} \equiv \mathit{res}'$.
2. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term} \Leftarrow \mathit{res}$.

PROVE: $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term} \Leftarrow \mathit{res}'$.

PROOF SKETCH: Induction over $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term} \Leftarrow \mathit{res}$.

$\langle 1 \rangle 1$. CASE: TY_RES_EMP
PROOF: $\mathit{res} = \mathit{res}' = \mathit{res\_term} = \texttt{emp}$.

$\langle 1 \rangle 2$. CASE: TY_RES_POINTSTO
$\mathit{res} = \mathit{points\_to}''$, $\mathit{res\_term} = \mathit{points\_to}'$, $\mathit{res}' = \mathit{points\_to}_1$, $\mathcal{R} = \cdot, \mathit{points\_to}$.

$\langle 2 \rangle 1$. $\Phi \vdash \mathit{points\_to} \equiv \mathit{points\_to}'$ and $\Phi \vdash \mathit{points\_to}' \equiv \mathit{points\_to}''$ by inversion.

$\langle 2 \rangle 2$. $\Phi \vdash \mathit{points\_to}' \equiv \mathit{points\_to}_1$ by transitivity (lemma 2.3).

$\langle 2 \rangle 3$. $\mathcal{C}; \mathcal{L}; \Phi; \cdot, \mathit{points\_to} \vdash \mathit{points\_to}' \Leftarrow \mathit{points\_to}_1$ as required.

⟨1⟩3. Case: Ty_Res_Var
Proof: By transitivity (lemma 2.3).

⟨1⟩4. Case: Ty_Res_SepConj
Proof: By induction.

⟨1⟩5. Case: Ty_Res_Conj
Proof: We know $\texttt{smt}\,(\Phi \Rightarrow (\mathit{term} \rightarrow \mathit{term}'))$ (by inversion on the equality) and $\texttt{smt}\,(\Phi \Rightarrow \mathit{term})$ (by inversion on the typing rule) so $\texttt{smt}\,(\Phi \Rightarrow \mathit{term}')$. Rest follows by induction.

⟨1⟩6. Case: Ty_Res_Pack
$\mathit{res\_term} = \texttt{pack}\,(\mathit{pval}, \mathit{res\_term}')$, $\mathit{res} = \exists\, y{:}\beta.\ \mathit{res}_1$, $\mathit{res}' = \exists\, y{:}\beta.\ \mathit{res}'_1$.

⟨2⟩1. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term}' \Leftarrow \mathit{pval}/y, \cdot(\mathit{res}'_1)$ by induction.

⟨2⟩2. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \texttt{pack}\,(\mathit{pval}, \mathit{res\_term}') \Leftarrow \exists\, y{:}\beta.\ \mathit{res}'_1$ as required.

2.5 Substitution Lemma

If $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash (\sigma){:}(\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}')$ and $\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash J$ then $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(J)$.

Proof sketch: Induction over the typing judgements.

Assume: 1. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash (\sigma){:}(\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}')$.
2. $\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash J$.

Prove: $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(J)$.

⟨1⟩1. Case: Ty_PVal_Obj*, Ty_PVal_{Obj,Loaded,Unit,True,False,Ctor_Nil}.
Proof: No free variables in J so $\sigma(J) = J$ and the rules do not depend on the environment, so we are done.

⟨1⟩2. Case: Ty_PVal_{List,Tuple,Ctor_Cons,Ctor_Tuple,Ctor_Array,Ctor_Specified}.
Proof: By induction and then definition of substitution over values.

⟨1⟩3. Case: Ty_PVal_Var.
$\mathcal{C}'; \mathcal{L}'; \Phi' \vdash x \Rightarrow \beta$

⟨2⟩1. $x{:}\beta \in \mathcal{C}'$ (or $x{:}\beta \in \mathcal{L}'$) by inversion.

⟨2⟩2. So $\exists \mathit{pval}.\ \mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{pval} \Rightarrow \beta$ by Ty_Subs_Cons_{Comp,Log}.

⟨2⟩3. Since $\mathit{pval} = \sigma(x)$, we are done.

⟨1⟩4. Case: Ty_PVal_Error.
Proof: Substitutions preserve SMT results (lemma 2.2).

⟨1⟩5. Case: Ty_PVal_Struct.
$\mathcal{C}'; \mathcal{L}'; \Phi' \vdash (\,\texttt{struct}\,\mathit{tag})\{\,\overline{.\mathit{member}_i = \mathit{pval}_i}^{\,i}\,\} \Rightarrow \texttt{struct}\,\mathit{tag}$

⟨2⟩1. $\overline{\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(\mathit{pval}_i) \Rightarrow \beta_{\tau_i}}^{\,i}$ by induction.

⟨2⟩2. $\mathcal{C}; \mathcal{L}; \Phi \vdash (\,\texttt{struct}\, tag)\{\,\overline{.member_i = \sigma(pval_i)}^{\,i}\,\} \Rightarrow \texttt{struct}\, tag$

⟨1⟩6. Case: Ty_Eq_Emp
Proof: True trivially (no free variables).

⟨1⟩7. Case: Ty_Res_Eq_PointsTo.
Proof: Substitutions preserver SMT results (lemma 2.2).

⟨1⟩8. Case: Ty_Res_Eq_SepConj.
Proof: By induction.

⟨1⟩9. Case: Ty_Res_Eq_Exists.
Proof: By induction.

⟨1⟩10. Case: Ty_Res_Eq_Term.
Proof: By induction and substitutions preserving SMT results (lemma 2.2).

⟨1⟩11. Case: Ty_Res_Emp.
Proof:True trivially (no free variables).

⟨1⟩12. Case: Ty_Res_PointsTo.
$\mathcal{C}'; \mathcal{L}'; \Phi'; \cdot, pt \vdash pt' \Leftarrow pt''$.
Prove: $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(pt') \Leftarrow \sigma(pt'')$.

⟨2⟩1. Since $\mathcal{R}' = \cdot, pt$, σ was derived using Ty_Subs_Cons_Res_Anon.

⟨2⟩2. $\Phi' \vdash pt \equiv pt'$ and $\Phi' \vdash pt' \equiv pt''$ by inversion on the case.

⟨2⟩3. So $\Phi \vdash \sigma(pt) \equiv \sigma(pt')$ and $\Phi \vdash \sigma(pt') \equiv \sigma(pt'')$ because substitutions preserve SMT results (lemma 2.2).

⟨2⟩4. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash res\_term \Leftarrow \sigma(pt)$ by inversion on ⟨2⟩1.

⟨2⟩5. $res\_term = pt_3$ for some pt_3 by inversion on ⟨2⟩4 (Ty_Res_PointsTo).

⟨2⟩6. $\Phi \vdash pt_3 \equiv \sigma(pt)$ by inversion on ⟨2⟩3.

⟨2⟩7. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(pt') \Leftarrow pt_3$.
Proof: Ty_Res_PointsTo is symmetric in all its pt arguments (because resource equality is an equivalence relation, lemma 2.3).

⟨2⟩8. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(pt') \Leftarrow \sigma(pt'')$.
Proof: By ⟨2⟩3, resource equality an equivalence relation (lemma 2.3) and resource typing subsumption (lemma 2.4).

⟨1⟩13. Case: Ty_Res_Var.
$\mathcal{C}'; \mathcal{L}'; \Phi'; \cdot, r{:}res \vdash r \Leftarrow res'$.

⟨2⟩1. From $\mathcal{R}' = \cdot, r{:}res$, we know σ was derived using Ty_Subs_Cons_Res_Named.

⟨2⟩2. $\sigma = res\_term/r, \sigma'$ and $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash res\_term \Leftarrow \sigma'(res)$ by inversion on ⟨2⟩1.

⟨2⟩3. $\Phi' \vdash res \equiv res'$ by inversion on Ty_Res_Var.

⟨2⟩4. $\Phi \vdash res \equiv res'$ and $\Phi \vdash \sigma(res) \equiv \sigma(res')$ by ⟨2⟩3 and substitution lemma over Ty_Res_Eq* cases.

⟨2⟩5. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash res\_term \Leftarrow \sigma'(res)$ by inversion on Ty_Subs_Cons_Res_Named.

⟨2⟩6. $\sigma(r) = res\_term$ by ⟨2⟩2.

⟨2⟩7. $\sigma'(res') = \sigma(res')$ (and same for res) because r cannot occur in either.

⟨2⟩8. Suffices: $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash res\_term \Leftarrow \sigma'(res')$ by ⟨2⟩3 and ⟨2⟩7.
Proof: Resource typing subsumption (lemma 2.4) and ⟨2⟩4.

⟨1⟩14. Case: Ty_Res_SepConj.
Proof: By induction.

⟨1⟩15. Case: Ty_Res_Conj.
$\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash res\_term \Leftarrow term \wedge res$.

⟨2⟩1. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(res\_term) \Leftarrow \sigma(res)$.
Proof: By induction.

⟨2⟩2. $\texttt{smt}\,(\Phi \Rightarrow \sigma(term))$.
Proof: Substitutions preserve SMT results (lemma 2.2).

⟨2⟩3. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(res\_term) \Leftarrow \sigma(term \wedge res)$ as required.

⟨1⟩16. Case: Ty_Res_Pack.
$\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash \texttt{pack}\,(pval, res\_term) \Leftarrow \exists\, y{:}\beta.\ res$.

⟨2⟩1. By induction,
1. $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(pval) \Rightarrow \beta$.
2. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(res\_term) \Leftarrow \sigma, pval/y, \cdot(res)$.

⟨2⟩2. So $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(\texttt{pack}\,(pval, res\_term)) \Leftarrow \sigma(\exists\, y{:}\beta.\ res)$.

⟨1⟩17. Case: Ty_Spine_Empty.
Proof: ret can be anything, including $\sigma(ret)$ and the rule does not depend on the environment, so we are done.

⟨1⟩18. Case: Ty_Spine_Comp.
$\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash x = pval, \overline{x_i = spine\_elem_i}^{\,i} :: \Pi\, x{:}\beta.\ arg \gg pval/x, \psi; ret$.

⟨2⟩1. By induction,
1. $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(pval) \Rightarrow \beta$.
2. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{x_i = \sigma(spine\_elem_i)}^{\,i} :: \sigma(arg) \gg \sigma(\psi); \sigma(ret)$.

⟨2⟩2. So $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash x = \sigma(pval), \overline{x_i = \sigma(spine\_elem_i)}^{\,i} :: \sigma(\Pi\, x{:}\beta.arg) \gg \sigma(pval/x, \psi); \sigma(ret)$.

⟨1⟩19. Case: Ty_Spine_Log.
Proof: Similar to Ty_Spine_Comp.

⟨1⟩20. Case: Ty_Spine_Res.
$\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'_1, \mathcal{R}_2 \vdash x = res\_term, \overline{x_i = spine\_elem_i}^{\,i} :: res \multimap arg \gg res\_term/x, \psi; ret$

⟨2⟩1. By inversion and then induction,
1. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}_1 \vdash \sigma(res\_term) \Leftarrow \sigma(res)$.
2. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}_2 \vdash \overline{x_i = \sigma(spine\_elem_i)}^{\,i} :: \sigma(res) \multimap \sigma(arg) \gg \sigma(\psi); \sigma(ret)$.

⟨2⟩2. Hence $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}_1, \mathcal{R}_2 \vdash x = \sigma(res\_term), \overline{x_i = \sigma(spine\_elem_i)}^{\,i} :: \sigma(res \multimap arg) \gg \sigma(res\_term/x, \psi); \sigma(ret)$ as required.

⟨1⟩21. Case: Ty_Spine_Phi.
$\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash \overline{x_i = spine\_elem_i}^{\,i} :: term \supset arg \gg \psi; ret$

⟨2⟩1. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{x_i = \sigma(spine\_elem_i)}^{\,i} :: \sigma(res) \multimap \sigma(arg) \gg \sigma(\psi); \sigma(ret)$.
Proof: By induction.

⟨2⟩2. $\texttt{smt}\,(\Phi \Rightarrow \sigma(term))$.
Proof: Substitutions preserve SMT results (lemma 2.2).

⟨2⟩3. Hence $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}_1, \mathcal{R}_2 \vdash x = \sigma(res\_term), \overline{x_i = \sigma(spine\_elem_i)}^{\,i} :: \sigma(res \multimap arg) \gg \sigma(res\_term/x, \psi); \sigma(ret)$ as required.

⟨1⟩22. Case: Ty_PE_Val
Proof: By induction.

⟨1⟩23. Case: Ty_PE_Array_Shift.
$\mathcal{C}'; \mathcal{L}'; \Phi' \vdash \texttt{array\_shift}\,(pval_1, \tau, pval_2) \Rightarrow y{:}\texttt{loc}.\ y = pval_1 +_{\text{ptr}} (pval_2 \times \text{size\_of}(\tau))$

⟨2⟩1. By induction,
1. $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(pval_1) \Rightarrow \texttt{loc}$
2. $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(pval_2) \Rightarrow \texttt{integer}$

⟨2⟩2. So, $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(\texttt{array\_shift}\,(pval_1, \tau, pval_2)) \Rightarrow y{:}\texttt{loc}.\ \sigma((y = pval_1 +_{\text{ptr}} (pval_2 \times \text{size\_of}(\tau))))$.

⟨1⟩24. Case: Ty_PE_Member_Shift.
Proof: Similar to Ty_PE_Array_Shift.

⟨1⟩25. Case: Ty_PE_{Not,Arith_Binop,Rel_Binop,Bool_Binop}.
Proof: By induction.

⟨1⟩26. Case: Ty_PE_Call.
See Ty_Seq_E_CCall for more general case and proof.

⟨1⟩27. Case: Ty_PE_{Assert_Undef,Bool_To_Integer,WrapI}.
Proof: By induction.

⟨1⟩28. Case: Ty_TPVal_Undef
See Ty_TVal_Undef for a more general case and proof.

⟨1⟩29. Case: Ty_TPVal_Done
$\mathcal{C}'; \mathcal{L}'; \Phi' \vdash \texttt{done}\, pval \Leftarrow y{:}\beta.\ term$.

⟨2⟩1. $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(pval) \Rightarrow \beta$.
Proof: By induction.

⟨2⟩2. $\texttt{smt}\,(\Phi \Rightarrow \sigma, pval/y, \cdot(term))$.
PROOF: Substitutions preserve SMT results (lemma 2.2).

⟨2⟩3. So $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(\texttt{done}\, pval) \Leftarrow y{:}\beta.\, \sigma(term)$.

⟨1⟩30. CASE: TY_TPE_{LET,LETT}.
See TY_SEQ_TE_{LET,LETT} for a more general case and proof.

⟨1⟩31. CASE: TY_TPE_IF.
PROOF: By induction.

⟨1⟩32. CASE: TY_TPE_CASE.
PROOF: See TY_SEQ_TE_CASE for more general case and proof.

⟨1⟩33. CASE: TY_{ACTION*,MEMOP*}.
PROOF: By induction and lemma 2.2 (substitutions preserve SMT results).

⟨1⟩34. CASE: TY_TVAL_I
PROOF: Trivially (no free variables nor requirements on constraint context).

⟨1⟩35. CASE: TY_TVAL_{COMP,LOG}.
Only focusing on logical case; computational one is similar.
$\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash \texttt{done}\, pval, \overline{spine\_elem_i}^{\,i} \Leftarrow \exists\, y{:}\beta.\, ret$.

⟨2⟩1. By inversion and then induction,
1. $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(pval) \Rightarrow \beta$
2. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(\texttt{done}\, \overline{spine\_elem_i}^{\,i}) \Leftarrow \sigma(pval/y, \cdot(ret))$.

⟨2⟩2. Therefore $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(\texttt{done}\, pval, \overline{spine\_elem_i}^{\,i}) \Leftarrow \exists\, y{:}\beta.\, \sigma(ret)$.

⟨1⟩36. CASE: TY_TVAL_PHI
$\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash \texttt{done}\, spine \Leftarrow term \wedge ret$

⟨2⟩1. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(\texttt{done}\, spine) \Leftarrow \sigma(ret)$.
PROOF: By induction.

⟨2⟩2. $\texttt{smt}\,(\Phi \Rightarrow \sigma(term))$.
PROOF: Substitutions preserve SMT results (lemma 2.2).

⟨2⟩3. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(\texttt{done}\, spine) \Leftarrow \sigma(term \wedge ret)$ as required.

⟨1⟩37. CASE: TY_TVAL_RES
PROOF: Similar to TY_TVAL_PHI, except with resource environments being split.

⟨1⟩38. CASE: TY_TVAL_UNDEF
PROOF: *ret* can be anything, including $\sigma(ret)$.

⟨1⟩39. CASE: TY_SEQ_TE_{TVAL,IF,BOUND}.
PROOF: By induction.

⟨1⟩40. CASE: TY_SEQ_E_{CCALL,PROC,RUN}.
Only focusing on CCall, rest are similar.

⟨2⟩1. $\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \overline{x_i = \sigma(spine\_elem_i)}^{\,i} :: \sigma(arg) \gg \sigma(\psi);\sigma(ret)$.
PROOF: By induction.

⟨2⟩2. $ident{:}arg \equiv \overline{x_i}^{\,i} \mapsto texpr \in \texttt{Globals}$ is unaffected by the substitution.

⟨2⟩3. $\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \texttt{ccall}\,(\tau, ident, \overline{\sigma(spine\_elem_i)}^{\,i}) \Rightarrow \sigma, \psi(ret)$ as required.

⟨1⟩41. CASE: TY_IS_{MEMOP,NEG_ACTION,ACTION}
PROOF: By induction.

⟨1⟩42. CASE: TY_SEQ_TE_{LETP,LETPT}.
PROOF: See TY_SEQ_TE_{LET,LETT}.

⟨1⟩43. CASE: TY_SEQ_TE_{LET,LETT,LETS}.
Only doing LET case, LETT and LETS are similar.
$\mathcal{C}';\mathcal{L}';\Phi';\mathcal{R}''',\mathcal{R}'' \vdash \texttt{let}\,\overline{ret\_pattern_i}^{\,i} = seq\_expr\,\texttt{in}\,texpr \Leftarrow ret_2$.

⟨2⟩1. By induction,
1. $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}' \vdash \sigma(seq\_expr) \Rightarrow \sigma(ret_1)$.
2. $\mathcal{C},\mathcal{C}_1;\mathcal{L},\mathcal{L}_1;\Phi,\Phi_1;\mathcal{R},\mathcal{R}_1 \vdash \sigma(texpr) \Leftarrow \sigma(ret_2)$.

⟨2⟩2. $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}',\mathcal{R} \vdash \sigma(\texttt{let}\,\overline{ret\_pattern_i}^{\,i} = seq\_expr\,\texttt{in}\,texpr) \Leftarrow \sigma(ret_2)$ as required.

⟨1⟩44. CASE: TY_SEQ_TE_CASE.
$\mathcal{C}';\mathcal{L}';\Phi';\mathcal{R}' \vdash \texttt{case}\,pval\,\texttt{of}\,\overline{\mid pattern_i \Rightarrow texpr_i}^{\,i}\,\texttt{end} \Leftarrow ret$.

⟨2⟩1. By induction,
1. $\mathcal{C};\mathcal{L};\Phi \vdash \sigma(pval) \Rightarrow \beta_1$.
2. $\overline{\mathcal{C},\mathcal{C}_i;\mathcal{L};\Phi,term_i = \sigma(pval);\mathcal{R} \vdash \sigma(texpr_i) \Leftarrow \sigma(ret)}^{\,i}$.

⟨2⟩2. $\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \sigma(\texttt{case}\,pval\,\texttt{of}\,\overline{\mid pattern_i \Rightarrow texpr_i}^{\,i}\,\texttt{end}) \Leftarrow \sigma(ret)$ as required.

⟨1⟩45. CASE: TY_TE_{IS,SEQ}.
PROOF: By induction.

2.6 Identity Extension

If $\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash (\sigma){:}(\mathcal{C}';\mathcal{L}';\Phi';\mathcal{R}')$ then $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}_1,\mathcal{R} \vdash (\sigma,\mathrm{id}){:}(\mathcal{C},\mathcal{C}';\mathcal{L},\mathcal{L}';\Phi';\mathcal{R}_1,\mathcal{R}')$.

PROOF SKETCH: Induction over the substitution.

ASSUME: $\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash (\sigma){:}(\mathcal{C}';\mathcal{L}';\Phi';\mathcal{R}')$.

PROVE: $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}_1,\mathcal{R} \vdash (\sigma,\mathrm{id}){:}(\mathcal{C},\mathcal{C}';\mathcal{L},\mathcal{L}';\Phi';\mathcal{R}_1,\mathcal{R}')$.

⟨1⟩1. $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}_1 \vdash (\mathrm{id}){:}(\mathcal{C};\mathcal{L};\Phi';\mathcal{R}_1)$.
PROOF: By induction on each of $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}_1$.

⟨1⟩2. $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}_1,\mathcal{R} \vdash (\sigma,\mathrm{id}){:}(\mathcal{C},\mathcal{C}';\mathcal{L},\mathcal{L}';\Phi';\mathcal{R}_1,\mathcal{R}')$
PROOF: By induction on σ with base case as above.

2.7 Let-friendly Substitution Lemma

If $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash (\sigma):(\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}')$ and $\mathcal{C}, \mathcal{C}'; \mathcal{L}, \mathcal{L}'; \Phi; \mathcal{R}_1, \mathcal{R}' \vdash J$ then $\mathcal{C}; \mathcal{L}; \sigma(\Phi); \mathcal{R}_1, \mathcal{R} \vdash \sigma(J)$.

Proof sketch: Apply identity extension then substitution lemma.

Assume: 1. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash (\sigma):(\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}')$.
2. $\mathcal{C}, \mathcal{C}'; \mathcal{L}, \mathcal{L}'; \Phi'; \mathcal{R}_1, \mathcal{R}' \vdash J$.

Prove: $\mathcal{C}; \mathcal{L}; \sigma(\Phi); \mathcal{R}_1, \mathcal{R} \vdash \sigma(J)$.

$\langle 1 \rangle 1$. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash (\sigma, \mathrm{id}):(\mathcal{C}, \mathcal{C}'; \mathcal{L}, \mathcal{L}'; \Phi'; \mathcal{R}_1, \mathcal{R}')$.
Proof: Apply identity extension to 1.

$\langle 1 \rangle 2$. $\mathcal{C}; \mathcal{L}; \sigma(\Phi); \mathcal{R}_1, \mathcal{R} \vdash (\sigma, \mathrm{id})(J)$.
Proof: Apply substitution lemma (2.5) to $\langle 1 \rangle 1$.

$\langle 1 \rangle 3$. $\mathcal{C}; \mathcal{L}; \sigma(\Phi); \mathcal{R}_1, \mathcal{R} \vdash \sigma(J)$.
Proof: $\mathrm{id}(J) = J$.

3 Progress

3.1 Ty_Spine_* and Decons_Arg_* construct same substitution and return type

If $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{x_i = \mathit{spine\_elem}_i}^{\,i} :: \mathit{arg} \gg \sigma; \mathit{ret}$ and $\overline{x_i = \mathit{spine\_elem}_i}^{\,i} :: \mathit{arg} \gg \sigma'; \mathit{ret}'$ then $\sigma = \sigma'$ and $\mathit{ret} = \mathit{ret}'$.

Proof sketch: Induction over *arg*.

3.2 Progress Statement and Proof

If $\cdot; \cdot; \cdot; \mathcal{R} \vdash e \Leftrightarrow t$ and all pattern in e are exhaustive then either e is a value, or it is unreachable, or $\forall h : R.\ \exists e', h'.\ \langle h; e \rangle \longrightarrow \langle h'; e' \rangle$.

Proof sketch: Induction over the typing rules.

Assume: 1. $\cdot; \cdot; \cdot; \mathcal{R} \vdash e \Leftrightarrow t$.
2. All patterns in e are exhaustive.
Prove: Either e is a value, or it is unreachable, or $\forall h : R.\ \exists e', h'.\ \langle h; e \rangle \longrightarrow \langle h'; e' \rangle$.

$\langle 1 \rangle 1$. Case: Ty_PVal_Obj*, Ty_PVal*, Ty_PE_Val, Ty_TPVal*, Ty_TVal*, Ty_Seq_TE_TVal.
Proof: All these judgements/rules give types to syntactic values; and there are no operational rules corresponding to them (see Section 7).

$\langle 1 \rangle 2$. Case: Ty_PE_Array_Shift.
Proof: By inversion on $\cdot; \cdot; \cdot \vdash \mathit{pval}_1 \Rightarrow \texttt{loc}$, pval_1 must be a *mem_ptr* (Ty_PVal_Obj_Ptr). Similarly pval_2 must be a *mem_int*, so rule Op_PE_PE_ArrayShift applies.

$\langle 1 \rangle 3$. Case: Ty_PE_Member_Shift.
Proof: *pval* must be a *mem_ptr* so Op_PE_PE_MemberShift.

⟨1⟩4. Case: Ty_PE_Not.
Proof: *pval* must be a *bool_value* so Op_PE_PE_Not_{True,False}.

⟨1⟩5. Case: Ty_PE_{Arith,Rel}_Binop.
Proof: $pval_1$ and $pval_2$ must be *mem_int*s so Op_PE_PE_{Arith,Rel}_Binop respectively.

⟨1⟩6. Case: Ty_PE_Bool_Binop.
Proof: $pval_1$ and $pval_2$ must be *bool_value*s so Op_PE_PE_Bool_Binop.

⟨1⟩7. Case: Ty_PE_Call.
Proof: By inversion we have $name{:}pure\_arg \equiv \overline{x_i}^{\,i} \mapsto tpexpr \in \texttt{Globals}$ and $\cdot;\cdot;\cdot;\cdot \vdash \overline{x_i = pval_i}^{\,i} :: pure\_arg \gg \sigma; \Sigma\, y{:}\beta.\ term \wedge \texttt{I}$, with the latter implying $\overline{x_i = pval_i}^{\,i} :: pure\_arg \gg \sigma; \Sigma\, y{:}\beta.\ term \wedge \texttt{I}$ (lemma 3.1. Thus it can step with Op_PE_TPE_Call.

⟨1⟩8. Case: Ty_PE_Assert_Undef.
Proof: *pval* must be a *bool_value* and $\texttt{smt}\,(\Phi \Rightarrow pval)$. If it is `False`, then by the latter, we have an inconsistent constraints context, meaning the code is unreachable. If it is `True`, we may step with Op_PE_PE_Assert_Undef.

⟨1⟩9. Case: Ty_PE_Bool_To_Integer.
Proof: *pval* must be a *bool_value* and so Op_PE_PE_Bool_To_Integer_{True,False}.

⟨1⟩10. Case: Ty_PE_WrapI.
Proof: *pval* must be a *mem_int* and so Op_PE_PE_WrapI.

⟨1⟩11. Case: Ty_TPE_{If,Let,LetT,Case}.
Proof: See Ty_Seq_TE_{If,Let,LetT,Case} cases for more general cases and proofs.

⟨1⟩12. Case: Ty_Action_Create.
Proof: *pval* must be a *mem_int* and h must be $\cdot$, so Op_Action_TVal_Create (*mem_ptr* and $pval{:}\beta_\tau$ are free in the premises and so can be constructed to satisfy the requirements).

⟨1⟩13. Case: Ty_Action_Load.
Proof: $pval_0$ must be a *mem_ptr* and $h = \cdot + \{pval_1 \overset{\checkmark}{\mapsto}_\tau pval_2\}$, so Op_Action_TVal_Load.

⟨1⟩14. Case: Ty_Action_Store.
Proof: $pval_0$ and $pval_2$ must be the same *mem_ptr*, so Op_Action_TVal_Store.

⟨1⟩15. Case: Ty_Action_Kill_Static.
Proof: $pval_0$ and $pval_1$ must be the same *mem_ptr*, so Op_Action_TVal_Kill_Static.

⟨1⟩16. Case: Ty_Memop_Rel_Binop.
Proof: Similar to Ty_PE_{Arith,Rel}_Binop.

⟨1⟩17. Case: Ty_Memop_IntFromPtr.
Proof: *pval* must be a *mem_ptr* so Op_Memop_TVal_Rel_IntFromPtr.

⟨1⟩18. Case: Ty_Memop_PtrFromInt.
Proof: *pval* must be a *mem_int* so Op_Memop_TVal_Rel_PtrFromInt.

⟨1⟩19. Case: Ty_Memop_PtrValidForDeref.
Proof: *pval* must be a *mem_ptr* and h must be $\cdot + \{mem_ptr \overset{\checkmark}{\mapsto}_{\tau} _\}$so it can take a step with Op_Memop_TVal_Rel_PtrValidForDeref.

⟨1⟩20. Case: Ty_Memop_PtrWellAligned.
Proof: *pval* must be a *mem_ptr* and so Op_Memop_TVal_PtrWellAligned.

⟨1⟩21. Case: Ty_Memop_PtrArrayShift.
Proof: $pval_1$ must be a *mem_ptr* and $pval_2$ must be a *mem_int* and so Op_Memop_TVal_PtrArrayShift.

⟨1⟩22. Case: Ty_Seq_E_CCall.
Proof: By inversion we have $ident{:}arg \equiv \overline{x_i}^{\,i} \mapsto texpr \in \texttt{Globals}$ and $\cdot;\cdot;\cdot;\cdot \vdash \overline{x_i = spine\_elem_i}^{\,i} :: arg \gg \sigma; ret$, with the latter implying $\overline{x_i = spine\_elem_i}^{\,i} :: arg \gg \sigma; ret$ (lemma 3.1. Thus it can step with Op_SE_TE_CCall.

⟨1⟩23. Case: Ty_Seq_E_Proc.
Proof: Similar to Ty_Seq_E_CCall.

⟨1⟩24. Case: Ty_Is_E_Memop.
Proof: By induction, if *mem_op* is unreachable, then the whole expression is so. Memops are not values. Only stepping cases applies, so Op_IsE_IsE_Memop.

⟨1⟩25. Case: Ty_Is_E_{Neg_}Action.
Proof: By induction, if *mem_action* is unreachable, then the whole expression is so. Actions are not values. Only stepping case applies, so Op_IsE_IsE_{Neg_}Action.

⟨1⟩26. Case: Ty_Seq_TE_{LetP,LetPT}.
Proof: See Ty_Seq_TE_{Let,LetT} for more general cases and proofs.

⟨1⟩27. Case: Ty_Seq_TE_Let.
Proof: By induction, since *seq_expr* is not value, if it is unreachable, the whole expression is so. If it takes a step, then Op_STE_TE_Let_LetT.

⟨1⟩28. Case: Ty_Seq_TE_LetT.
Proof: By induction, if *texpr* is unreachable, so is the whole expression. If if it a *tval* then Op_STE_TE_LetT_Sub. If if takes a step, then Op_STE_TE_LetT_LetT.

⟨1⟩29. Case: Ty_Seq_TE_Case.
Proof: By assumption that all patterns are exhaustive, there is at least one pattern against which *pval* will match, so Op_STE_TE_Case.

⟨1⟩30. Case: Ty_Seq_TE_If.
Proof: *pval* must be a *bool_value* and so Op_STE_TE_If_{True,False}.

⟨1⟩31. Case: Ty_Seq_TE_Run.
Proof: Similar to Ty_Seq_E_CCall.

⟨1⟩32. Case: Ty_Seq_TE_Bound.
Proof: By Op_STE_TE_Bound.

⟨1⟩33. Case: Ty_Is_TE_LetS.
Proof: Similar to Ty_Seq_TE_LetT.

4 Framing

If $\langle h; e\rangle \longrightarrow \langle h'; e'\rangle$ and $\exists h_1, h_2.\ \text{disjoint}(h_1, h_2) \wedge h = h_1 + h_2 \wedge \langle h_1; e\rangle \longrightarrow \langle h_1'; e'\rangle$ then $h' = h_1' + h_2$.

Assume: 1. $\langle h; e\rangle \longrightarrow \langle h'; e'\rangle$,
2. $h = h_1 + h_2$ where h_1, h_2 disjoint,
3. and $\langle h_1; e\rangle \longrightarrow \langle h_1'; e'\rangle$.
Prove: $h' = h_1' + h_2$.

Proof sketch:Induction over the operational rules. Only covering ones which modify the heap; rest are trivially true.

⟨1⟩1. Case: Op_Action_TVal_Create
Proof: Because *mem_ptr* is fresh.

⟨1⟩2. Case: Op_Action_TVal_{Store,Kill}.
Proof: By assumption of disjointness, $mem\_ptr \in h_1$ implies $mem\_ptr \notin h_2$.

5 Type Preservation

5.1 Pointed-to values have type β_τ

For $pt = \_ \overset{\checkmark}{\mapsto}_\tau pval$, if $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash pt \Leftarrow pt$ then $\mathcal{C}; \mathcal{L}; \Phi \vdash pval \Rightarrow \beta_\tau$.

Proof sketch: Induction over the typing judgements. Only Ty_Action_Store create such permissions, and its premise $\mathcal{C}; \mathcal{L}; \Phi \vdash pval_1 \Rightarrow \beta_\tau$ ensures the desired property. Ty_Action_Load simply preserves the property.

5.2 Terms derived from patterns are "equal to" matching values

Assume: 1. $pattern{:}\beta \rightsquigarrow \mathcal{C}\ \texttt{with}\ term$.
2. $pattern = pval \rightsquigarrow \sigma$.

Prove: The constraint $term_j = pval$ holds.

Proof sketch: Induction over *pattern*.

5.3 Deconstructing a pattern leads to a well-typed substitution

First, computational part.

Assume: 1. $\cdot; \cdot; \cdot \vdash pval \Rightarrow \beta_1$.
2. $ident\_or\_pattern{:}\beta \rightsquigarrow \mathcal{C}\ \texttt{with}\ term$.
3. $ident\_or\_pattern = pval \rightsquigarrow \sigma$.

PROVE: $\cdot;\cdot;\cdot;\cdot \vdash (\sigma){:}(\mathcal{C};\cdot;\cdot;\cdot)$.

PROOF SKETCH: By induction over 2.

⟨1⟩1. CASE: TY_PAT_SYM_OR_PATTERN_SYM and TY_PAT_COMP_SYM_ANNOT.
$\sigma = pval/x, \cdot$ and $\mathcal{C} = \cdot, x{:}\beta$.
PROOF: By TY_SUBS_CONS_COMP and 1.

⟨1⟩2. CASE: TY_PAT_NO_SYM_ANNOT and TY_PAT_COMP_NIL.
σ and $\mathcal{C}$ are empty.
PROOF: By TY_SUBS_EMPTY, we are done.

⟨1⟩3. CASE: TY_PAT_COMP_{SPECIFIED,CONS,TUPLE,ARRAY}.
PROOF: By induction (and concatenating well-typed substitutions).

Now, resource part.

ASSUME: 1. $\cdot;\cdot;\cdot;\mathcal{R} \vdash res\_term \Leftarrow res$.
2. $res\_pattern{:}res \rightsquigarrow \mathcal{L};\Phi;\mathcal{R}'$.
3. $res\_pattern = res\_term \rightsquigarrow \sigma$.
PROVE: $\cdot;\cdot;\cdot;\mathcal{R} \vdash (\sigma){:}(\cdot;\mathcal{L};\Phi;\mathcal{R}')$.

PROOF SKETCH: By induction over 2.

⟨1⟩1. CASE: TY_PAT_RES_EMPTY.
$res\_pattern = res\_term = res = \texttt{emp}$. $\sigma, \mathcal{L}, \Phi, \mathcal{R}, \mathcal{R}'$ are all empty.
PROOF: By TY_SUBS_EMPTY, we are done.

⟨1⟩2. CASE: TY_PAT_RES_POINTSTO.
$res\_pattern = res\_term = res = pt$. $\sigma = \cdot$, $\mathcal{L} = \cdot$, $\Phi = \cdot$, $\mathcal{R} = \mathcal{R}' = \cdot, pt$.
PROOF: By TY_SUBS_CONS_RES_ANON.

⟨1⟩3. CASE: TY_PAT_RES_VAR.
$res\_pattern = r$, $\sigma = res\_term/x, \cdot$, $\mathcal{L} = \cdot$, $\Phi = \cdot$, $\mathcal{R}' = \cdot, x{:}res$.
PROOF: By TY_SUBS_CONS_RES_NAMED.

⟨1⟩4. CASE: TY_PAT_RES_SEPCONJ.
PROOF: By induction (and concatenating well-typed substitutions).

⟨1⟩5. CASE: TY_PAT_RES_CONJ.
PROOF: By $\texttt{smt}\,(\cdot \Rightarrow term)$ (from 1) and induction with TY_SUB_CONS_PHI.

⟨1⟩6. CASE: TY_PAT_RES_PACK.
$res\_pattern = \texttt{pack}\,(x, res\_pattern')$, $res\_term = \texttt{pack}\,(pval, res\_term')$, $res = \exists\, x{:}\beta.\ res'$.
$\sigma = pval/x, \sigma'$, $\mathcal{L} = \mathcal{L}', x{:}\beta$, $\mathcal{R} = \mathcal{R}'$.
PROOF: By induction and TY_SUBS_CONS_LOG.

Now, full proof.

ASSUME: 1. $\overline{ret\_pattern_i = spine\_elem_i}^{\,i} \rightsquigarrow \sigma$.
2. $\cdot;\cdot;\cdot;\mathcal{R} \vdash \texttt{done}\ \overline{spine\_elem_i}^{\,i} \Leftarrow ret$.
3. $\overline{ret\_pattern_i}^{\,i} {:} ret \rightsquigarrow \mathcal{C};\mathcal{L};\Phi;\mathcal{R}'$.

PROVE: $\cdot;\cdot;\cdot;\mathcal{R} \vdash (\sigma) : (\mathcal{C};\mathcal{L};\Phi;\mathcal{R}')$.

PROOF SKETCH: Induction on 3.

⟨1⟩1. CASE: TY_RET_PAT_EMPTY
PROOF: By TY_SUBS_EMPTY.

⟨1⟩2. CASE: TY_RET_PAT_{COMP,RES}
PROOF: By induction, well-typed computational / resource substitutions and concatenating well-typed substitutions.

⟨1⟩3. CASE: TY_RET_PATH_LOG.
PROOF: By induction.

⟨1⟩4. CASE: TY_RET_PAT_PHI
PROOF: By induction and inversion on 2 to conclude $\texttt{smt}\,(\cdot \Rightarrow term)$
(required by TY_SUBS_CONS_PHI).

5.4 Type Preservation Statement and Proof

If $\cdot;\cdot;\cdot;\mathcal{R} \vdash e \Leftrightarrow t$ then $\forall h : \mathcal{R}, e', h' : \mathcal{R}'.\ \langle h; e\rangle \longrightarrow \langle h'; e'\rangle \implies \cdot;\cdot;\cdot;\mathcal{R}' \vdash e' \Leftrightarrow t$.

PROOF SKETCH: Induction over the typing rules.

ASSUME: 1. $\cdot;\cdot;\cdot;\mathcal{R} \vdash e \Leftrightarrow t$
2. arbitrary $h : \mathcal{R}, e', h' : \mathcal{R}'$
3. $\langle h; e\rangle \longrightarrow \langle h'; e'\rangle$.

PROVE: $\cdot;\cdot;\cdot;\mathcal{R}' \vdash e' \Leftrightarrow t$.

⟨1⟩1. CASE: TY_PE_ARRAY_SHIFT.
LET: $term = mem\_ptr +_{\text{ptr}} (mem\_int \times \text{size\_of}(\tau))$.
ASSUME: 1. $\cdot;\cdot;\cdot \vdash \texttt{array\_shift}\,(mem\_ptr, \tau, mem\_int) \Rightarrow y{:}\texttt{loc}.\ y = term$.
2. $\langle \texttt{array\_shift}\,(mem\_ptr, \tau, mem\_int)\rangle \longrightarrow \langle mem\_ptr'\rangle$.
PROVE: $\cdot;\cdot;\cdot \vdash mem\_ptr' \Rightarrow y{:}\texttt{loc}.\ y = term$.
PROOF: By TY_PVAL_OBJ_INT, TY_PVAL_OBJ, TY_PE_VAL and construction of mem_ptr' (inversion on 2).

⟨1⟩2. CASE: TY_PE_MEMBER_SHIFT.
PROOF SKETCH: Similar to TY_ARRAY_SHIFT.

⟨1⟩3. CASE: TY_PE_NOT.
ASSUME: 1. $\cdot;\cdot;\cdot \vdash \texttt{not}\,(bool\_value) \Rightarrow y{:}\texttt{bool}.\ y = \neg\, bool\_value$.
2. $\langle \texttt{not}\,(\texttt{True})\rangle \longrightarrow \langle \texttt{False}\rangle$ or $\langle \texttt{not}\,(\texttt{False})\rangle \longrightarrow \langle \texttt{True}\rangle$.
PROVE: $\cdot;\cdot;\cdot \vdash bool\_value' \Rightarrow y{:}\texttt{bool}.\ y = \neg\, bool\_value$.
PROOF: By TY_PVAL_{TRUE,FALSE}, TY_PE_VAL and 2.

⟨1⟩4. CASE: TY_PE_ARITH_BINOP.
LET: $term = mem\_int_1\ binop_{arith}\ mem\_int_2$.
ASSUME: 1. $\cdot;\cdot;\cdot \vdash mem\_int_1\ binop_{arith}\ mem\_int_2 \Rightarrow y{:}\texttt{integer}.\ y = term$.
2. $\langle mem\_int_1\ binop_{arith}\ mem\_int_2\rangle \longrightarrow \langle mem\_int\rangle$.
PROVE: $\cdot;\cdot;\cdot \vdash mem\_int \Rightarrow y{:}\texttt{integer}.\ y = term$.

PROOF: By TY_PVAL_OBJ_INT, TY_PVAL_OBJ, TY_PE_VAL and construction of *mem_int* (inversion on 2).

⟨1⟩5. CASE: TY_PE_{REL,BOOL}_BINOP.
PROOF SKETCH: Similar to TY_PE_ARITH_BINOP.

⟨1⟩6. CASE: TY_PE_CALL.
PROOF: See TY_SEQ_E_CALL for a more general case and proof.

⟨1⟩7. CASE: TY_PE_ASSERT_UNDEF.
ASSUME: 1. $\cdot;\cdot;\cdot \vdash \texttt{assert\_undef}\,(\texttt{True},\ \mathit{UB\_name}) \Rightarrow y{:}\texttt{unit}.\ y = \texttt{unit}$.
2. $\langle \texttt{assert\_undef}\,(\texttt{True},\ \mathit{UB\_name})\rangle \longrightarrow \langle \texttt{Unit}\rangle$.
PROVE: $\cdot;\cdot;\cdot \vdash \texttt{Unit} \Rightarrow y{:}\texttt{unit}.\ y = \texttt{unit}$.
PROOF: By TY_PVAL_UNIT and TY_PE_VAL.

⟨1⟩8. CASE: TY_PE_BOOL_TO_INTEGER.
LET: $\mathit{term} = \texttt{if}\ \mathit{bool\_value}\ \texttt{then}\ 1\ \texttt{else}\ 0$.
ASSUME: 1. $\cdot;\cdot;\cdot \vdash \texttt{bool\_to\_integer}\,(\mathit{bool\_value}) \Rightarrow y{:}\texttt{integer}.\ y = \mathit{term}$.
2. $\langle \texttt{bool\_to\_integer}\,(\texttt{True})\rangle \longrightarrow \langle 1\rangle$ or $\langle \texttt{bool\_to\_integer}\,(\texttt{False})\rangle \longrightarrow \langle 0\rangle$.
PROVE: $\cdot;\cdot;\cdot \vdash \mathit{mem\_int} \Rightarrow y{:}\texttt{integer}.\ y = \mathit{term}$
PROOF: By cases on *bool_value*, then applying TY_PVAL_{TRUE,FALSE} and TY_PE_VAL.

⟨1⟩9. CASE: TY_PE_WRAPI.
PROOF SKETCH: Similar to TY_PE_BOOL_TO_INTEGER, except by cases on $\mathit{abbrev}_2 \leq \mathrm{max\_int}_\tau$, then applying TY_PVAL_OBJ_INT, TY_PVAL_OBJ and TY_PE_VAL.

⟨1⟩10. CASE: TY_TPE_IF.
PROOF: See TY_SEQ_TE_IF for a more general case and proof.

⟨1⟩11. CASE: TY_TPE_LET.
PROOF: See TY_SEQ_TE_LET for a more general case and proof.

⟨1⟩12. CASE: TY_TPE_LETT.
PROOF: See TY_SEQ_TE_LETT for a more general case and proof.

⟨1⟩13. CASE: TY_TPE_CASE.
PROOF: See TY_SEQ_TE_CASE for a more general case and proof.

⟨1⟩14. CASE: TY_ACTION_CREATE.
LET: $\mathit{pt} = \mathit{mem\_ptr} \overset{\times}{\mapsto}_\tau \mathit{pval}$.
$\mathit{term} = \texttt{representable}\,(\tau*, y_p) \wedge \texttt{alignedI}\,(\mathit{mem\_int}, y_p)$.
$\mathit{ret} = \Sigma\, y_p{:}\texttt{loc}.\ \mathit{term} \wedge \exists\, y{:}\beta_\tau.\ y_p \overset{\times}{\mapsto}_\tau y \otimes \texttt{I}$.
ASSUME: 1. $\cdot;\cdot;\cdot;\cdot \vdash \texttt{create}\,(\mathit{mem\_int}, \tau) \Rightarrow \mathit{ret}$.
2. $\langle \cdot; \texttt{create}\,(\mathit{mem\_int}, \tau)\rangle \longrightarrow \langle \cdot + \{\mathit{pt}\}; \texttt{done}\,\mathit{mem\_ptr}, \mathit{pval}, \mathit{pt}\rangle$.
PROVE: $\cdot;\cdot;\cdot;\cdot, \mathit{pt} \vdash \texttt{done}\,\mathit{mem\_ptr}, \mathit{pval}, \mathit{pt} \Leftarrow \mathit{ret}$.

⟨2⟩1. $\cdot;\cdot;\cdot \vdash \mathit{mem\_ptr} \Rightarrow \texttt{loc}$ by TY_PVAL_OBJ_INT and TY_PVAL_OBJ.

⟨2⟩2. $\texttt{smt}\,(\cdot \Rightarrow \mathit{term})$ by construction of *mem_ptr*.

⟨2⟩3. $\cdot;\cdot;\cdot \vdash \mathit{pval} \Rightarrow \beta_\tau$ by construction of *pval*.

⟨2⟩4. $\cdot;\cdot;\cdot;\cdot, pt \vdash pt \Leftarrow pt$ by Ty_Res_PointsTo.

⟨2⟩5. By Ty_TVal_I and then ⟨2⟩4 – ⟨2⟩1 with Ty_TVal_{Res,Log,Phi,Comp} respectively, we are done.

⟨1⟩15. Case: Ty_Action_Load.

Let: $pt = mem\_ptr \overset{\checkmark}{\mapsto}_\tau pval$.
$ret = \Sigma\, y{:}\beta_\tau.\ y = pval \wedge pt \otimes \mathtt{I}$.
Assume: 1. $\cdot;\cdot;\cdot;\mathcal{R} \vdash \mathtt{load}\,(\tau, mem\_ptr, \_, pt) \Rightarrow ret$.
2. $\langle \cdot + \{pt\}; \mathtt{load}\,(\tau, mem\_ptr, \_, pt)\rangle \longrightarrow \langle \cdot + \{pt\}; \mathtt{done}\, pval, pt\rangle$.
Prove: $\cdot;\cdot;\cdot;\mathcal{R} \vdash \mathtt{done}\, pval, pt \Leftarrow ret$

⟨2⟩1. $\mathcal{R} = \cdot, pt'$ where $\cdot \vdash pt' \equiv pt$ by inversion on 1.

⟨2⟩2. $\mathtt{smt}\,(\cdot \Rightarrow pval = pval)$ trivially.

⟨2⟩3. $\cdot;\cdot;\cdot \vdash pval \Rightarrow \beta_\tau$ by ⟨2⟩1 and pointed-values have the right type (lemma 5.1).

⟨2⟩4. By Ty_TVal_I and then ⟨2⟩1 – ⟨2⟩3 with Ty_TVal_{Res,Phi,Comp} respectively, we are done.

⟨1⟩16. Case: Ty_Action_Store.

Let: $pt = mem\_ptr \overset{\checkmark}{\mapsto}_\tau \_$.
$pt' = mem\_ptr \overset{\checkmark}{\mapsto}_\tau pval$.
$ret = \Sigma\, \_{:}\mathtt{unit}.\ pt' \otimes \mathtt{I}$.
Assume: 1. $\cdot;\cdot;\cdot;\mathcal{R} \vdash \mathtt{store}\,(\_, \tau, pval_0, pval_1, \_, pt) \Rightarrow ret$.
2. $\langle \cdot + \{pt\}; \mathtt{store}\,(\_, \tau, mem\_ptr, pval, \_, pt)\rangle \longrightarrow \langle \cdot + \{pt'\}; \mathtt{done}\,\mathtt{Unit}, pt'\rangle$.
Prove: $\cdot;\cdot;\cdot;\cdot, pt' \vdash \mathtt{done}\,\mathtt{Unit}, pt' \Leftarrow ret$.

⟨2⟩1. $\mathcal{R} = \cdot, pt_2$ where $\cdot \vdash pt'' \equiv pt$, by inversion on the typing assumption.

⟨2⟩2. $\cdot;\cdot;\cdot \vdash \mathtt{Unit} \Rightarrow \mathtt{unit}$ by Ty_PVal_Unit.

⟨2⟩3. $\cdot;\cdot;\cdot;\cdot, pt' \vdash pt' \Leftarrow pt'$ by Ty_Res_PointsTo.

⟨2⟩4. By Ty_TVal_I and ⟨2⟩2 and ⟨2⟩3 with Ty_TVal_{Res,Comp} respectively, we are done.

⟨1⟩17. Case: Ty_Action_Kill_Static.

Let: $pt = mem\_ptr \mapsto_\tau \_$.
Assume: 1. $\cdot;\cdot;\cdot;\mathcal{R} \vdash \mathtt{kill}\,(\mathtt{static}\,\tau, pval_0, pt) \Rightarrow \Sigma\, \_{:}\mathtt{unit}.\ \mathtt{I}$.
2. $\langle \cdot + \{pt\}; \mathtt{kill}\,(\mathtt{static}\,\tau, mem\_ptr, pt)\rangle \longrightarrow \langle h; \mathtt{done}\,\mathtt{Unit}\rangle$.
Prove: $\cdot;\cdot;\cdot;\cdot \vdash \mathtt{done}\,\mathtt{Unit} \Leftarrow \Sigma\, \_{:}\mathtt{unit}.\ \mathtt{I}$
Proof: By Ty_TVal_I, Ty_PVal_Unit and then Ty_TVal_Comp. Note: $\mathcal{R} = \cdot, pt'$ where $\cdot \vdash pt' \equiv pt$.

⟨1⟩18. Case: Ty_Memop_Rel_Binop.
Proof: Similar Ty_PE_Rel_Binop, except with Ty_TVal_{I,Phi,Comp} at the end.

⟨1⟩19. Case: Ty_Memop_IntFromPtr.

Let: $ret = \Sigma\, y{:}\mathtt{integer}.\ y = \mathtt{cast\_ptr\_to\_int}\, mem\_ptr \wedge \mathtt{I}$.
Assume: 1. $\cdot;\cdot;\cdot;\cdot \vdash \mathtt{intFromPtr}\,(\tau_1, \tau_2, mem\_ptr) \Rightarrow ret$.
2. $\langle \cdot; \mathtt{intFromPtr}\,(\tau_1, \tau_2, mem\_ptr)\rangle \longrightarrow \langle \cdot; \mathtt{done}\, mem\_int\rangle$.
Prove: $\cdot;\cdot;\cdot;\cdot \vdash \mathtt{done}\, mem\_int \Leftarrow ret$

⟨2⟩1. $\texttt{smt}\,(\cdot \Rightarrow \mathit{mem\_int} = \texttt{cast\_ptr\_to\_int}\,\mathit{mem\_ptr})$ by construction of *mem_int* (inversion on 2).

⟨2⟩2. $\cdot;\cdot;\cdot \vdash \mathit{mem\_int} \Rightarrow \texttt{integer}$ by Ty_PVal_Obj_Int and Ty_PVal_Obj.

⟨2⟩3. By Ty_TVal_I and ⟨2⟩1 and ⟨2⟩2 with Ty_TVal_{Phi,Comp} respectively, we are done.

⟨1⟩20. Case: Ty_Memop_PtrFromInt.
Proof: Similar to Ty_Memop_IntFromPtr, swapping base types `integer` and `loc`.

⟨1⟩21. Case: Ty_Memop_PtrValidForDeref.
Let: $\mathit{pt} = \mathit{mem\_ptr} \overset{\checkmark}{\mapsto}_{\tau} \_$.
$\mathit{ret} = \Sigma\, y{:}\texttt{bool}.\ y = \texttt{aligned}\,(\tau, \mathit{mem\_ptr}) \wedge \mathit{pt} \otimes \texttt{I}$.
Assume: 1. $\cdot;\cdot;\cdot;\mathcal{R} \vdash \texttt{ptrValidForDeref}\,(\tau, \mathit{mem\_ptr}, \mathit{pt}) \Rightarrow \mathit{ret}$.
2. $\langle \cdot + \{\mathit{pt}\}; \texttt{ptrValidForDeref}\,(\tau, \mathit{mem\_ptr}, \mathit{pt})\rangle \longrightarrow \langle \cdot + \{\mathit{pt}\}; \texttt{done}\,\mathit{bool\_value}, \mathit{pt}\rangle$.
Prove: $\cdot;\cdot;\cdot;\cdot, \mathit{pt} \vdash \texttt{done}\,\mathit{bool\_value}, \mathit{pt} \Leftarrow \mathit{ret}$.

⟨2⟩1. $\cdot;\cdot;\cdot;\cdot, \mathit{pt} \vdash \mathit{pt} \Leftarrow \mathit{pt}$, by inversion on 1.

⟨2⟩2. $\mathit{bool\_value} = \texttt{aligned}\,(\tau, \mathit{mem\_ptr})$ by construction of *bool_value* (inversion on 2).

⟨2⟩3. $\cdot;\cdot;\cdot \vdash \mathit{bool\_value} \Rightarrow \texttt{bool}$ by Ty_PVal_{True,False}.

⟨2⟩4. By Ty_TVal_I, and then ⟨2⟩1 – ⟨2⟩3 with Ty_TVal_{Res,Phi,Comp} respectively, we are done.

⟨1⟩22. Case: Ty_Memop_PtrWellAligned.
Let: $\mathit{ret} = \Sigma\, y{:}\texttt{bool}.\ y = \texttt{aligned}\,(\tau, \mathit{mem\_ptr}) \wedge \texttt{I}$.
Assume: 1. $\cdot;\cdot;\cdot;\cdot \vdash \texttt{ptrWellAligned}\,(\tau, \mathit{mem\_ptr}) \Rightarrow \mathit{ret}$.
2. $\langle \cdot; \texttt{ptrWellAligned}\,(\tau, \mathit{mem\_ptr})\rangle \longrightarrow \langle \cdot; \texttt{done}\,\mathit{bool\_value}\rangle$.
Prove: $\cdot;\cdot;\cdot;\cdot \vdash \texttt{done}\,\mathit{bool\_value} \Rightarrow \mathit{ret}$.

⟨2⟩1. $\texttt{smt}\,(\cdot \Rightarrow \mathit{bool\_value} = \texttt{aligned}\,(\tau, \mathit{mem\_ptr}))$ by construction of *bool_value* (inversion on 2).

⟨2⟩2. $\cdot;\cdot;\cdot \vdash \mathit{bool\_value} \Rightarrow \texttt{bool}$ by Ty_PVal_{True,False}.

⟨2⟩3. By Ty_TVal_I and ⟨2⟩1 and ⟨2⟩2 with Ty_TVal_{Phi,Comp} respectively, we are done.

⟨1⟩23. Case: Ty_Memop_PtrArrayShift.
Proof: Similiar to Ty_PE_Array_Shift, except with Ty_TVal_{I,Phi,Comp} at the end.

⟨1⟩24. Case: Ty_Seq_E_CCall.
Assume: 1. $\cdot;\cdot;\cdot;\mathcal{R} \vdash \texttt{ccall}\,(\tau, \mathit{ident}, \overline{\mathit{spine\_elem}_i}^{\,i}) \Rightarrow \sigma(\mathit{ret})$.
2. $\langle h; \texttt{ccall}\,(\tau, \mathit{ident}, \overline{\mathit{spine\_elem}_i}^{\,i})\rangle \longrightarrow \langle h; \sigma'(\mathit{texpr}){:}\sigma'(\mathit{ret})\rangle$.
Prove: $\cdot;\cdot;\cdot;\mathcal{R} \vdash \sigma(\mathit{texpr}) \Leftarrow \sigma(\mathit{ret})$

⟨2⟩1. $\mathit{ident}{:}\mathit{arg} \equiv \overline{x_i}^{\,i} \mapsto \mathit{texpr} \in \texttt{Globals}$ by inversion (on either assumption).

⟨2⟩2. $\cdot;\cdot;\cdot;\mathcal{R} \vdash \overline{x_i = \mathit{spine\_elem}_i}^{\,i} :: \mathit{arg} \gg \sigma; \mathit{ret}$ by inversion on 1.

⟨2⟩3. $\sigma = \sigma'$ and $ret = ret'$ by induction on arg.
PROOF: TY_SPINE_* and DECONS_ARG_* construct same substitution and return type (lemma 3.1).

⟨2⟩4. LET: $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}'$ be the the type of substitution σ: $\cdot;\cdot;\cdot;\mathcal{R} \vdash (\sigma):(\mathcal{C};\mathcal{L};\Phi;\mathcal{R}')$.
PROOF: From ⟨2⟩2 we may deduce
1. $\mathcal{C};\mathcal{L};\Phi \vdash pval_i \Rightarrow \beta_i$ for each $x_i{:}\beta_i \in \mathcal{C}$ or $x_i{:}\beta_i \in \mathcal{L}$.
2. $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}' \vdash res\_term_i \Leftarrow res_i$ for each $res_i \in \mathcal{R}'$.
3. $\texttt{smt}\,(\cdot \Rightarrow term)$ for each $term \in \Phi$.

⟨2⟩5. $\mathcal{C}'';\mathcal{L}'';\Phi'';\mathcal{R}'' \vdash texpr \Leftarrow ret''$ where $\overline{x_i}^{\,i} :: arg \rightsquigarrow \mathcal{C}'';\mathcal{L}'';\Phi'';\mathcal{R}'' \mid ret''$ formalises the assumption that all global functions and labels are well-typed.

⟨2⟩6. $\mathcal{C} = \mathcal{C}''$, $\Phi = \Phi''$, $\mathcal{L} = \mathcal{L}''$, $\mathcal{R}' = \mathcal{R}''$ and $ret = ret''$.
PROOF: By induction on arg.

⟨2⟩7. Apply substitution lemma (2.5) to ⟨2⟩4 and ⟨2⟩5 to finish proof.

⟨1⟩25. CASE: TY_SEQ_E_PROC.
PROOF: Similar to TY_SEQ_E_CCALL.

⟨1⟩26. CASE: TY_IS_E_MEMOP.
PROOF: By induction on TY_MEMOP* cases.

⟨1⟩27. CASE: TY_IS_E_{NEG_}ACTION.
PROOF: By induction on TY_ACTION* cases.

⟨1⟩28. CASE: TY_SEQ_TE_LETP.
PROOF SKETCH: Only covering case $\langle pexpr\rangle \longrightarrow \langle pexpr'\rangle$ here.
See TY_SEQ_TE_LET for a more general version and proof for the remaining $\langle pexpr\rangle \longrightarrow \langle tpexpr{:}(y{:}\beta.\, term)\rangle$ case.
ASSUME: 1. $\cdot;\cdot;\cdot \vdash \texttt{let}\, ident\_or\_pattern = pexpr\, \texttt{in}\, tpexpr \Leftarrow y_2{:}\beta_2.\, term_2$.
2. $\langle \texttt{let}\, ident\_or\_pattern = pexpr\, \texttt{in}\, tpexpr\rangle \longrightarrow \langle \texttt{let}\, ident\_or\_pattern = pexpr'\, \texttt{in}\, tpexpr\rangle$.
PROVE: $\cdot;\cdot;\cdot \vdash \texttt{let}\, ident\_or\_pattern = pexpr'\, \texttt{in}\, tpexpr \Leftarrow y_2{:}\beta_2.\, term_2$.

⟨2⟩1. 1. $\cdot;\cdot;\cdot \vdash pexpr \Rightarrow y{:}\beta.\, term$.
2. $ident\_or\_pattern{:}\beta \rightsquigarrow \mathcal{C}_1\, \texttt{with}\, term_1$.
3. $\mathcal{C}_1;\cdot;\cdot, term_1/y, \cdot(term), \Phi_1;\mathcal{R} \vdash texpr \Leftarrow ret$.
PROOF: Invert assumption 1.

⟨2⟩2. $\langle pexpr\rangle \longrightarrow \langle pexpr'\rangle$.
PROOF: Invert assumption 2.

⟨2⟩3. $\cdot;\cdot;\cdot \vdash pexpr' \Rightarrow y{:}\beta.\, term$.
PROOF: By induction on ⟨2⟩1.1 and ⟨2⟩2.

⟨2⟩4. $\cdot;\cdot;\cdot \vdash \texttt{let}\, ident\_or\_pattern = pexpr'\, \texttt{in}\, tpexpr \Leftarrow y_2{:}\beta_2.\, term_2$.
PROOF: By TY_SEQ_TE_LETP using ⟨2⟩1.2,3 and ⟨2⟩3.

⟨1⟩29. CASE: TY_SEQ_TE_LETPT.
PROOF: See TY_SEQ_TE_LETT for a more general case and proof.

⟨1⟩30. CASE: TY_SEQ_TE_LET.

ASSUME: 1. $\cdot;\cdot;\cdot;\mathcal{R}', \mathcal{R} \vdash \texttt{let}\ \overline{ret\_pattern_i}^{\,i} = seq\_expr\ \texttt{in}\ texpr_2 \Leftarrow ret_2$.
2. $\langle h; \texttt{let}\ \overline{ret\_pattern_i}^{\,i} = seq\_expr\ \texttt{in}\ texpr_2\rangle \longrightarrow \langle h; \texttt{let}\ \overline{ret\_pattern_i}^{\,i} : ret_1' = texpr_1\ \texttt{in}\ texpr_2\rangle$.
PROVE: $\cdot;\cdot;\cdot;\mathcal{R}', \mathcal{R} \vdash \texttt{let}\ \overline{ret\_pattern_i}^{\,i} : ret_1 = texpr_1\ \texttt{in}\ texpr_2 \Leftarrow ret_2$.

$\langle 2\rangle$1. 1. $\cdot;\cdot;\cdot;\mathcal{R}' \vdash seq\_expr \Rightarrow ret_1$.
2. $\overline{ret\_pattern_i}^{\,i} : ret_1 \rightsquigarrow \mathcal{C}_1; \mathcal{L}_1; \Phi_1; \mathcal{R}_1$.
3. $\mathcal{C}_1; \mathcal{L}_1; \Phi_1; \mathcal{R}, \mathcal{R}_1 \vdash texpr \Leftarrow ret_2$.
PROOF: By inversion on 1.

$\langle 2\rangle$2. $\langle h; seq\_expr\rangle \longrightarrow \langle h; texpr_1 : ret_1'\rangle$.
PROOF: By inversion on 2.

$\langle 2\rangle$3. $\cdot;\cdot;\cdot;\mathcal{R}' \vdash texpr_1 \Leftarrow ret_1$.
PROOF: By induction on $\langle 2\rangle$1.1 and $\langle 2\rangle$2.

$\langle 2\rangle$4. $ret_1 = ret_1'$.
PROOF: By cases TY_SEQ_E_{CCALL,PCALL}.

$\langle 2\rangle$5. By TY_SEQ_TE_LET with $\langle 2\rangle$1.2,3 and $\langle 2\rangle$3, we are done.

$\langle 1\rangle$31. CASE: TY_SEQ_TE_LETT.
NOTE: $h : \mathcal{R}', \mathcal{R}$ and $h : \mathcal{R}_1, \mathcal{R}$.

ASSUME: 1. $\cdot;\cdot;\cdot;\mathcal{R}', \mathcal{R} \vdash \texttt{let}\ \overline{ret\_pattern_i}^{\,i} : ret_1 = \texttt{done}\ \overline{spine\_elem_i}^{\,i}\ \texttt{in}\ texpr_2 \Leftarrow ret_2$.
2. $\langle h; \texttt{let}\ \overline{ret\_pattern_i}^{\,i} : ret_1 = \texttt{done}\ \overline{spine\_elem_i}^{\,i}\ \texttt{in}\ texpr\rangle \longrightarrow \langle h; \sigma(texpr_2)\rangle$.
PROVE: $\cdot;\cdot;\cdot;\mathcal{R}', \mathcal{R} \vdash \sigma(texpr_2) \Leftarrow \sigma(ret_2)$.

$\langle 2\rangle$1. 1. $\cdot;\cdot;\cdot;\mathcal{R}' \vdash \texttt{done}\ \overline{spine\_elem_i}^{\,i} \Leftarrow ret_1$.
2. $\overline{ret\_pattern_i}^{\,i} : ret_1 \rightsquigarrow \mathcal{C}_1; \mathcal{L}_1; \Phi_1; \mathcal{R}_1$.
3. $\mathcal{C}_1; \mathcal{L}_1; \Phi_1; \mathcal{R}_1, \mathcal{R} \vdash texpr_2 \Leftarrow ret_2$.
PROOF: By inversion on 1.

$\langle 2\rangle$2. $\overline{ret\_pattern_i = spine\_elem_i}^{\,i} \rightsquigarrow \sigma$.
PROOF: By inversion on 2.

$\langle 2\rangle$3. $\cdot;\cdot;\cdot;\mathcal{R}' \vdash (\sigma) : (\mathcal{C}_1; \mathcal{L}_1; \Phi_1; \mathcal{R}_1)$.
PROOF: By $\langle 2\rangle$1.1,2 and $\langle 2\rangle$3, $\langle 2\rangle$2 using lemma 5.3 (deconstructing a pattern produces a well-typed substitution).

$\langle 2\rangle$4. By $\langle 2\rangle$1.3 and $\langle 2\rangle$3 and the let-friendly substitution lemma 2.7, we are done.

$\langle 1\rangle$32. CASE: TY_SEQ_TE_LETT.
ASSUME: 1. $\cdot;\cdot;\cdot;\mathcal{R}', \mathcal{R} \vdash \texttt{let}\ \overline{ret\_pattern_i}^{\,i} : ret_1 = texpr_1\ \texttt{in}\ texpr_2 \Leftarrow ret_2$.
2. $\langle h; \texttt{let}\ \overline{ret\_pattern_i}^{\,i} : ret = texpr_1\ \texttt{in}\ texpr_2\rangle \longrightarrow \langle h'; \texttt{let}\ \overline{ret\_pattern_i}^{\,i} : ret = texpr_1'\ \texttt{in}\ texpr_2\rangle$.
PROVE: $\cdot;\cdot;\cdot;\mathcal{R}'', \mathcal{R} \vdash \texttt{let}\ \overline{ret\_pattern_i}^{\,i} : ret_1 = texpr_1'\ \texttt{in}\ texpr_2 \Leftarrow ret_2$.

$\langle 2\rangle$1. 1. $\cdot;\cdot;\cdot;\mathcal{R}' \vdash texpr_1 \Leftarrow ret_1$.
2. $\overline{ret\_pattern_i}^{\,i} : ret_1 \rightsquigarrow \mathcal{C}_1; \mathcal{L}_1; \Phi_1; \mathcal{R}_1$.
3. $\mathcal{C}_1; \mathcal{L}_1; \Phi_1; \mathcal{R}_1, \mathcal{R} \vdash texpr_2 \Leftarrow ret_2$.
PROOF: By inversion on 1.

$\langle 2\rangle$2. $\langle h; texpr_1\rangle \longrightarrow \langle h'; texpr_1'\rangle$.

PROOF: By inversion on 2.

⟨2⟩3. $\cdot;\cdot;\cdot;\mathcal{R}'' \vdash texpr'_1 \Leftarrow ret_1$.
PROOF: By induction on ⟨1⟩32.1 and ⟨2⟩2.

⟨2⟩4. By ⟨2⟩3, ⟨1⟩32.2,3 using TY_SEQ_TE_LETT, we are done.

⟨1⟩33. CASE: TY_SEQ_TE_CASE.
ASSUME: 1. $\cdot;\cdot;\cdot;\mathcal{R} \vdash \texttt{case}\, pval\, \texttt{of}\, \overline{\mid pattern_i \Rightarrow texpr_i}^{\,i}\, \texttt{end} \Leftarrow ret$.
2. $\langle h; \texttt{case}\, pval\, \texttt{of}\, \overline{\mid pattern_i \Rightarrow texpr_i}^{\,i}\, \texttt{end}\rangle \longrightarrow \langle h; \sigma_j(texpr_j)\rangle$.
PROVE: $\cdot;\cdot;\cdot;\mathcal{R} \vdash \sigma_j(texpr_j) \Leftarrow ret$.

⟨2⟩1. 1. $\cdot;\cdot;\cdot \vdash pval \Rightarrow \beta_1$.
2. $\overline{pattern_i{:}\beta_1 \rightsquigarrow \mathcal{C}_i\, \texttt{with}\, term_i}^{\,i}$.
3. $\overline{\mathcal{C}_i;\cdot;\cdot, term_i = pval; \mathcal{R} \vdash texpr_i \Leftarrow ret}^{\,i}$.
PROOF: By inversion on 1.

⟨2⟩2. 1. $pattern_j = pval \rightsquigarrow \sigma_j$.
2. $\forall\, i < j.\ \texttt{not}\, (pattern_i = pval \rightsquigarrow \sigma_i)$.
PROOF: By inversion on 2.

⟨2⟩3. $term_j = pval$.
PROOF: By ⟨1⟩32.2 and terms derived from patterns are "equal to" matching values (lemma 5.2).

⟨2⟩4. $\cdot;\cdot;\cdot;\cdot \vdash (\sigma_j){:}(\mathcal{C}_j;\cdot;\cdot, term_j = pval;\cdot)$.
PROOF: By ⟨2⟩3 and lemma 5.3 (deconstructing a pattern produces a well-typed substitution).

⟨2⟩5. By ⟨2⟩4, ⟨1⟩32.3 and substitution lemma 2.5, we are done.

⟨1⟩34. CASE: TY_SEQ_TE_IF.
Only covering `True` case, `False` is almost identical.
ASSUME: 1. $\cdot;\cdot;\cdot;\mathcal{R} \vdash \texttt{if True then}\, texpr_1\, \texttt{else}\, texpr_2 \Leftarrow ret$.
2. $\langle h; \texttt{if True then}\, texpr_1\, \texttt{else}\, texpr_2\rangle \longrightarrow \langle h; texpr_1\rangle$.
PROVE: $\cdot;\cdot;\cdot;\mathcal{R} \vdash texpr_1 \Leftarrow ret$.
PROOF: Invert 1, note $\cdot;\cdot;\cdot;\mathcal{R} \vdash (\mathrm{id}){:}(\cdot;\cdot;\cdot, \texttt{true} = \texttt{true}; \mathcal{R})$ and then apply substitution lemma (2.5).

⟨1⟩35. CASE: TY_SEQ_TE_RUN.
PROOF SKETCH: Similar to case TY_SEQ_E_{CCALL,PCALL}.

⟨1⟩36. CASE: TY_SEQ_TE_BOUND.
PROOF: By inversion on the typing rule.

⟨1⟩37. CASE: TY_IS_TE_LETS.
PROOF SKETCH: Similar to TY_SEQ_TE_LETT.

6 Typing Judgements

$$
\begin{array}{lll}
\mathit{object\_value\_jtype} & ::= & \\
 & \mid & \mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{object\_value} \Rightarrow \texttt{obj}\ \beta \\
 & & \\
\mathit{pval\_jtype} & ::= & \\
 & \mid & \mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{pval} \Rightarrow \beta \\
 & & \\
\mathit{res\_jtype} & ::= & \\
 & \mid & \Phi \vdash \mathit{res} \equiv \mathit{res}' \\
 & \mid & \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term} \Leftarrow \mathit{res} \\
 & \mid & h{:}\mathcal{R} \\
 & & \\
\mathit{spine\_jtype} & ::= & \\
 & \mid & \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{x_i = \mathit{spine\_elem}_i}^{\,i} :: \mathit{arg} \gg \sigma; \mathit{ret} \\
 & & \\
\mathit{pexpr\_jtype} & ::= & \\
 & \mid & \mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{pexpr} \Rightarrow \mathit{ident}{:}\beta.\, \mathit{term} \\
 & & \\
\mathit{tpval\_jtype} & ::= & \\
 & \mid & \mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{tpval} \Leftarrow \mathit{ident}{:}\beta.\, \mathit{term} \\
 & & \\
\mathit{tpexpr\_jtype} & ::= & \\
 & \mid & \mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{tpexpr} \Leftarrow \mathit{ident}{:}\beta.\, \mathit{term} \\
 & & \\
\mathit{action\_jtype} & ::= & \\
 & \mid & \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{mem\_action} \Rightarrow \mathit{ret} \\
 & & \\
\mathit{memop\_jtype} & ::= & \\
 & \mid & \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{mem\_op} \Rightarrow \mathit{ret} \\
 & & \\
\mathit{seq\_expr\_jtype} & ::= & \\
 & \mid & \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{seq\_expr} \Rightarrow \mathit{ret} \\
 & & \\
\mathit{is\_expr\_jtype} & ::= & \\
 & \mid & \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{is\_expr} \Rightarrow \mathit{ret} \\
 & & \\
\mathit{tval\_jtype} & ::= & \\
 & \mid & \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{tval} \Leftarrow \mathit{ret} \\
 & & \\
\mathit{texpr\_jtype} & ::= & \\
 & \mid & \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{seq\_texpr} \Leftarrow \mathit{ret} \\
 & \mid & \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{is\_texpr} \Leftarrow \mathit{ret} \\
 & \mid & \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{texpr} \Leftarrow \mathit{ret}
\end{array}
$$

7 Opsem Judgements

$$
\begin{array}{lll}
\mathit{pure\_opsem\_jtype} & ::= & \\
 & | & \langle \mathit{pexpr} \rangle \longrightarrow \langle \mathit{pexpr}' \rangle \\
 & | & \langle \mathit{pexpr} \rangle \longrightarrow \langle \mathit{tpexpr}{:}(y{:}\beta.\ \mathit{term}) \rangle \\
 & | & \langle \mathit{tpexpr} \rangle \longrightarrow \langle \mathit{tpexpr}' \rangle
\end{array}
$$

$$
\begin{array}{lll}
\mathit{opsem\_jtype} & ::= & \\
 & | & \langle h; \mathit{seq\_expr} \rangle \longrightarrow \langle h'; \mathit{texpr}{:}\mathit{ret} \rangle \\
 & | & \langle h; \mathit{seq\_texpr} \rangle \longrightarrow \langle h'; \mathit{texpr} \rangle \\
 & | & \langle h; \mathit{mem\_op} \rangle \longrightarrow \langle h'; \mathit{tval} \rangle \\
 & | & \langle h; \mathit{mem\_action} \rangle \longrightarrow \langle h'; \mathit{tval} \rangle \\
 & | & \langle h; \mathit{is\_expr} \rangle \longrightarrow \langle h'; \mathit{is\_expr}' \rangle \\
 & | & \langle h; \mathit{is\_texpr} \rangle \longrightarrow \langle h'; \mathit{texpr} \rangle \\
 & | & \langle h; \mathit{texpr} \rangle \longrightarrow \langle h'; \mathit{texpr}' \rangle
\end{array}
$$