Explicit CN Soundness Proof

Dhruv Makwana

November 15, 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 exists, 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.

⟨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.

⟨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.

⟨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. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}_1 \vdash \sigma(\mathit{res\_term}) \Leftarrow \sigma(\mathit{res})$.
2. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}_2 \vdash \overline{x_i = \sigma(\mathit{spine\_elem}_i)}^{\,i} :: \sigma(\mathit{res}) \multimap \sigma(\mathit{arg}) \gg \sigma(\psi); \sigma(\mathit{ret})$.

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

⟨1⟩21. CASE: TY_SPINE_PHI.
$\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash \overline{x_i = \mathit{spine\_elem}_i}^{\,i} :: \mathit{term} \supset \mathit{arg} \gg \psi; \mathit{ret}$

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

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

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

⟨1⟩22. CASE: TY_PE_VAL
PROOF: By induction.

⟨1⟩23. CASE: TY_PE_ARRAY_SHIFT.
$\mathcal{C}'; \mathcal{L}'; \Phi' \vdash \mathtt{array\_shift}\,(\mathit{pval}_1, \tau, \mathit{pval}_2) \Rightarrow y{:}\mathtt{loc}.\ y = \mathit{pval}_1 +_{\text{ptr}} (\mathit{pval}_2 \times \text{size\_of}(\tau))$

⟨2⟩1. By induction,
1. $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(\mathit{pval}_1) \Rightarrow \mathtt{loc}$
2. $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(\mathit{pval}_2) \Rightarrow \mathtt{integer}$

⟨2⟩2. So, $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(\mathtt{array\_shift}\,(\mathit{pval}_1, \tau, \mathit{pval}_2)) \Rightarrow y{:}\mathtt{loc}.\ \sigma((y = \mathit{pval}_1 +_{\text{ptr}} (\mathit{pval}_2 \times \text{size\_of}(\tau))))$.

⟨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
$\mathcal{C}'; \mathcal{L}'; \Phi' \vdash \mathtt{done}\,\mathit{pval} \Leftarrow y{:}\beta.\ \mathit{term}$.

⟨2⟩1. $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(\mathit{pval}) \Rightarrow \beta$.
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. $ident{:}arg \equiv \overline{x_i}^{\,i} \mapsto texpr \in$ `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 patterns 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 6).

$\langle 1 \rangle 2.$ Case: Ty_PE_Array_Shift.
Proof: By inversion on $\cdot; \cdot; \cdot \vdash \mathit{pval}_1 \Rightarrow \mathtt{loc}$, 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 $\cdot;\cdot;\cdot;\cdot \vdash \overline{x_i = pval_i}^{\,i} :: pure\_arg \gg \sigma; \Sigma\, y{:}\beta.\ term \wedge \mathtt{I}$, with the latter implying $\overline{x_i = pval_i}^{\,i} :: pure\_arg \gg \sigma; \Sigma\, y{:}\beta.\ term \wedge \mathtt{I}$ (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 $\mathtt{smt}\,(\Phi \Rightarrow pval)$. 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 Type Preservation

4.1 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.

4.2 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 $term = pval$ holds.

PROOF SKETCH: Induction over $pattern$.

4.3 `strip_ifs` is idempotent

PROOF SKETCH: Induction over the definition.

4.4 Deconstructing a stripped resource produces the same environment

ASSUME: 1. $\Phi \vdash res\_pattern{:}res \rightsquigarrow \mathcal{L}; \Phi; \mathcal{R}$.
2. $\Phi \vdash res' = \texttt{strip\_ifs}\,(res)$.
PROVE: $\Phi \vdash res\_pattern{:}res' \rightsquigarrow \mathcal{L}; \Phi; \mathcal{R}$.

⟨1⟩1. SUFFICES: $\Phi \vdash res' = \texttt{strip\_ifs}\,(res')$.
PROOF: By `strip_ifs` idempotent and assumption 2.

⟨1⟩2. $\Phi \vdash res'\,\texttt{as}\,res\_pattern \rightsquigarrow \mathcal{L}; \Phi; \mathcal{R}$ by inversion on 1.

⟨1⟩3. By definiton of $\Phi \vdash res\_pattern{:}res \rightsquigarrow \mathcal{L}; \Phi; \mathcal{R}$ and ⟨1⟩1 and ⟨1⟩2 we are done.

4.5 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.

$\langle 1 \rangle 1.$ 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.

$\langle 1 \rangle 2.$ 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.

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

Now, resource part (of deconstructing a pattern leads to a well-typed substitution).

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

PROOF SKETCH: By induction over 1.

$\langle 1 \rangle 1.$ CASE: TY_RES_EMPTY.
$\mathit{res\_pattern} = \mathit{res\_term} = \mathit{res} = \mathtt{emp}$. $\sigma, \mathcal{L}, \Phi, \mathcal{R}, \mathcal{R}'$ are all empty.
PROOF: By TY_SUBS_EMPTY, we are done.

$\langle 1 \rangle 2.$ CASE: TY_RES_POINTSTO.
$\mathit{res\_pattern} = r$, $\mathit{res\_term} = pt$, $\sigma = pt/r, \cdot$, $\mathcal{L} = \cdot$, $\Phi = \cdot$, $\mathcal{R} = \mathcal{R}' = \cdot, r{:}pt$.
PROOF: By TY_SUBS_CONS_RES.

$\langle 1 \rangle 3.$ CASE: TY_RES_VAR.
$\mathit{res\_pattern} = r$, $\sigma = \mathit{res\_term}/r, \cdot$, $\mathcal{L} = \cdot$, $\Phi = \cdot$, $\mathcal{R} = \mathcal{R}' = \cdot, r{:}\mathit{res}$.
PROOF: By TY_SUBS_CONS_RES.

$\langle 1 \rangle 4.$ CASE: TY_RES_SEPCONJ.
PROOF: By induction (and concatenating well-typed substitutions).

$\langle 1 \rangle 5.$ CASE: TY_RES_CONJ.
PROOF: By $\mathtt{smt}\,(\cdot \Rightarrow \mathit{term})$ (from 1) and induction with TY_SUB_CONS_PHI.

$\langle 1 \rangle 6.$ CASE: TY_RES_PACK.
$\mathit{res\_pattern} = \mathtt{pack}\,(x, \mathit{res\_pattern}')$, $\mathit{res\_term} = \mathtt{pack}\,(\mathit{pval}, \mathit{res\_term}')$, $\mathit{res} = \exists\, x{:}\beta.\ \mathit{res}'$.
$\sigma = \mathit{pval}/x, \sigma'$, $\mathcal{L} = \mathcal{L}', x{:}\beta$, $\mathcal{R} = \mathcal{R}'$.
PROOF: By induction and TY_SUBS_CONS_LOG.

$\langle 1 \rangle 7.$ CASE: TY_RES_FOLD.
$\mathit{res\_pattern} = \mathtt{fold}\,(\mathit{res\_pattern}')$, $\mathit{res\_term} = \mathtt{fold}\,(\mathit{res\_term}')$, $\mathit{res} = \alpha(\overline{\mathit{pval}_i}^{\,i})$.

$\langle 2 \rangle 1.$ 1. $\alpha \equiv \overline{x_i{:}\beta_i}^{\,i} \mapsto \mathit{res}' \in \mathtt{Globals}$.
2. $\Phi \vdash \mathit{res}'' = \mathtt{strip\_ifs}\,(\mathit{res}')$.
3. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term}' \Leftarrow \mathit{res}''$.
PROOF: Inversion on 1.

$\langle 2\rangle 2.$ $\Phi \vdash \mathit{res\_pattern}'{:}\overline{\mathit{pval}_i/x_i,\cdot}^{\,i}\,(\mathit{res}') \leadsto \mathcal{L}'; \Phi'; \mathcal{R}'$.
PROOF: Inversion on 2.

$\langle 2\rangle 3.$ $\Phi \vdash \mathit{res\_pattern}'{:}\mathit{res}'' \leadsto \mathcal{L}'; \Phi'; \mathcal{R}'$.
PROOF: By $\langle 2\rangle 1.2$, $\langle 2\rangle 2$ and deconstructing a stripped resource produces the same environment (lemma 4.4).

$\langle 2\rangle 4.$ $\mathit{res\_pattern}' = \mathit{res\_term}' \leadsto \sigma$.
PROOF: By inversion on 3.

$\langle 2\rangle 5.$ $\cdot; \cdot; \cdot; \mathcal{R} \vdash (\sigma){:}(\cdot; \mathcal{L}; \Phi; \mathcal{R}')$.
PROOF: By induction on $\langle 2\rangle 1.3$, $\langle 2\rangle 3$ and $\langle 2\rangle 4$.

Now, full proof (of deconstructing a pattern leads to a well-typed substitution).

ASSUME: 1. $\overline{\mathit{ret\_pattern}_i = \mathit{spine\_elem}_i}^{\,i} \leadsto \sigma$.
2. $\cdot; \cdot; \cdot; \mathcal{R} \vdash \texttt{done}\ \overline{\mathit{spine\_elem}_i}^{\,i} \Leftarrow \mathit{ret}$.
3. $\Phi \vdash \overline{\mathit{ret\_pattern}_i}^{\,i}{:}\mathit{ret} \leadsto \mathcal{C}; \mathcal{L}'; \Phi'; \mathcal{R}'$.
PROVE: $\cdot; \cdot; \cdot; \mathcal{R} \vdash (\sigma){:}(\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}')$.

PROOF SKETCH: Induction on 3.
$\langle 1\rangle 1.$ CASE: TY_RET_PAT_EMPTY
PROOF: By TY_SUBS_EMPTY.

$\langle 1\rangle 2.$ CASE: TY_RET_PAT_{COMP,RES}
PROOF: By induction, well-typed computational / resource substitutions and concatenating well-typed substitutions.

$\langle 1\rangle 3.$ CASE: TY_RET_PATH_LOG.
PROOF: By induction.

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

4.6 Type Preservation Statement and Proof

If $\cdot; \cdot; \cdot; \mathcal{R}_1 \vdash e \Leftrightarrow t$ then $\forall h{:}\mathcal{R}, f, e', h'.\ \langle h + f; e\rangle \longrightarrow \langle h'; e'\rangle \implies \exists h''{:}\mathcal{R}'.\ h' = h'' + f \wedge \cdot; \cdot; \cdot; \mathcal{R}' \vdash e' \Leftrightarrow t$.

PROOF SKETCH: Induction over the typing rules.

ASSUME: 1. $\cdot; \cdot; \cdot; \mathcal{R}_1 \vdash e \Leftrightarrow t$
2. arbitrary $h{:}\mathcal{R}_1, f, e', h'$
3. $\langle h + f; e\rangle \longrightarrow \langle h'; e'\rangle$.

PROVE: $\exists h'{:}\mathcal{R}_1'.\ h' = h' + f \wedge \cdot; \cdot; \cdot; \mathcal{R}_1' \vdash e' \Leftrightarrow t$.

$\langle 1\rangle 1.$ CASE: TY_PE_ARRAY_SHIFT.
LET: $\mathit{term} = \mathit{mem\_ptr} +_{\mathrm{ptr}} (\mathit{mem\_int} \times \mathrm{size\_of}(\tau))$.
ASSUME: 1. $\cdot; \cdot; \cdot \vdash \texttt{array\_shift}\,(\mathit{mem\_ptr}, \tau, \mathit{mem\_int}) \Rightarrow y{:}\texttt{loc}.\ y = \mathit{term}$.

2. $\langle \texttt{array\_shift}\,(\mathit{mem\_ptr}, \tau, \mathit{mem\_int}) \rangle \longrightarrow \langle \mathit{mem\_ptr}' \rangle$.
PROVE: $\cdot;\cdot;\cdot \vdash \mathit{mem\_ptr}' \Rightarrow y{:}\texttt{loc}.\ y = \mathit{term}$
(because this is a pure expression, heaps are irrelevant).
PROOF: By TY_PVAL_OBJ_INT, TY_PVAL_OBJ, TY_PE_VAL and construction of $\mathit{mem\_ptr}'$ (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. $\cdot;\cdot;\cdot \vdash \texttt{not}\,(\mathit{bool\_value}) \Rightarrow y{:}\texttt{bool}.\ y = \neg\,\mathit{bool\_value}$.
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: $\cdot;\cdot;\cdot \vdash \mathit{bool\_value}' \Rightarrow y{:}\texttt{bool}.\ y = \neg\,\mathit{bool\_value}$
(because this is a pure expression, heaps are irrelevant).
PROOF: By TY_PVAL_{TRUE,FALSE}, TY_PE_VAL and 2.

⟨1⟩4. CASE: TY_PE_ARITH_BINOP.
LET: $\mathit{term} = \mathit{mem\_int}_1\, \mathit{binop}_{\mathit{arith}}\, \mathit{mem\_int}_2$.
ASSUME: 1. $\cdot;\cdot;\cdot \vdash \mathit{mem\_int}_1\ \mathit{binop}_{\mathit{arith}}\ \mathit{mem\_int}_2 \Rightarrow y{:}\texttt{integer}.\ y = \mathit{term}$.
2. $\langle \mathit{mem\_int}_1\ \mathit{binop}_{\mathit{arith}}\ \mathit{mem\_int}_2 \rangle \longrightarrow \langle \mathit{mem\_int} \rangle$.
PROVE: $\cdot;\cdot;\cdot \vdash \mathit{mem\_int} \Rightarrow y{:}\texttt{integer}.\ y = \mathit{term}$
(because this is a pure expression, heaps are irrelevant).
PROOF: By TY_PVAL_OBJ_INT, TY_PVAL_OBJ, TY_PE_VAL and construction of $\mathit{mem\_int}$ (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: $\cdot;\cdot;\cdot \vdash \texttt{Unit} \Rightarrow y{:}\texttt{unit}.\ y = \texttt{unit}$
(because this is a pure expression, heaps are irrelevant).
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: $\cdot;\cdot;\cdot \vdash \mathit{mem\_int} \Rightarrow y{:}\texttt{integer}.\ y = \mathit{term}$
(because this is a pure expression, heaps are irrelevant).
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: $pt = mem\_ptr \overset{\times}{\mapsto}_\tau pval$.
$term = \texttt{representable}\,(\tau*, y_p) \wedge \texttt{alignedI}\,(mem\_int, y_p)$.
$ret = \Sigma\, y_p{:}\texttt{loc}.\ term \wedge \exists\, y{:}\beta_\tau.\ y_p \overset{\times}{\mapsto}_\tau y \otimes \texttt{I}$.
$h = \cdot$ so $h' = \cdot + \{pt\}$.
ASSUME: 1. $\cdot; \cdot; \cdot; \cdot \vdash \texttt{create}\,(mem\_int, \tau) \Rightarrow ret$.
2. $\langle f; \texttt{create}\,(mem\_int, \tau)\rangle \longrightarrow \langle f + \{pt\}; \texttt{done}\, mem\_ptr, pval, pt\rangle$.
PROVE: $\cdot; \cdot; \cdot; \cdot, \_{:}pt \vdash \texttt{done}\, mem\_ptr, pval, pt \Leftarrow ret$.

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

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

⟨2⟩3. $\cdot; \cdot; \cdot \vdash pval \Rightarrow \beta_\tau$ 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: $pt = mem\_ptr \overset{\checkmark}{\mapsto}_\tau pval$.
$ret = \Sigma\, y{:}\beta_\tau.\ y = pval \wedge pt \otimes \texttt{I}$.
$h = h' = \cdot + \{pt\}$.
ASSUME: 1. $\cdot; \cdot; \cdot; \mathcal{R} \vdash \texttt{load}\,(\tau, mem\_ptr, \_, pt) \Rightarrow ret$.
2. $\langle f + \{pt\}; \texttt{load}\,(\tau, mem\_ptr, \_, pt)\rangle \longrightarrow \langle f + \{pt\}; \texttt{done}\, pval, pt\rangle$.
PROVE: $\cdot; \cdot; \cdot; \mathcal{R} \vdash \texttt{done}\, pval, pt \Leftarrow ret$

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

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

⟨2⟩3. $\cdot; \cdot; \cdot \vdash pval \Rightarrow \beta_\tau$ by ⟨2⟩1 and pointed-values have the right type (lemma 4.1).

⟨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: $pt = mem\_ptr \overset{\checkmark}{\mapsto}_\tau \_$.
$pt' = mem\_ptr \overset{\checkmark}{\mapsto}_\tau pval$.
$ret = \Sigma\, \_{:}\texttt{unit}.\ pt' \otimes \texttt{I}$.

$h = h' = \cdot + \{pt\}$.

ASSUME: 1. $\cdot;\cdot;\cdot;\mathcal{R} \vdash \texttt{store}\,(\_, \tau, pval_0, pval_1, \_, pt) \Rightarrow ret$.

2. $\langle f + \{pt\}; \texttt{store}\,(\_, \tau, mem\_ptr, pval, \_, pt)\rangle \longrightarrow \langle f + \{pt'\}; \texttt{done}\,\texttt{Unit}, pt'\rangle$.

PROVE: $\cdot;\cdot;\cdot;\cdot, \_{:}pt' \vdash \texttt{done}\,\texttt{Unit}, pt' \Leftarrow ret$.

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

⟨2⟩2. $\cdot;\cdot;\cdot \vdash \texttt{Unit} \Rightarrow \texttt{unit}$ 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: $pt = mem\_ptr \mapsto_\tau \_$.

$\mathcal{R} = \cdot, \_{:}pt'$ where $\cdot \vdash pt' \equiv pt$.

$h = \cdot + \{pt\}$ so $h' = \cdot$.

ASSUME: 1. $\cdot;\cdot;\cdot;\mathcal{R} \vdash \texttt{kill}\,(\texttt{static}\,\tau, pval_0, pt) \Rightarrow \Sigma\,\_{:}\texttt{unit}.\,\texttt{I}$.

2. $\langle f + \{pt\}; \texttt{kill}\,(\texttt{static}\,\tau, mem\_ptr, pt)\rangle \longrightarrow \langle f; \texttt{done}\,\texttt{Unit}\rangle$.

PROVE: $\cdot;\cdot;\cdot;\cdot \vdash \texttt{done}\,\texttt{Unit} \Leftarrow \Sigma\,\_{:}\texttt{unit}.\,\texttt{I}$

PROOF: By TY_TVAL_I, TY_PVAL_UNIT and then TY_TVAL_COMP.

⟨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: $ret = \Sigma\,y{:}\texttt{integer}.\,y = \texttt{cast\_ptr\_to\_int}\,mem\_ptr \wedge \texttt{I}$.

$h = \cdot$ so $h' = \cdot$.

ASSUME: 1. $\cdot;\cdot;\cdot;\cdot \vdash \texttt{intFromPtr}\,(\tau_1, \tau_2, mem\_ptr) \Rightarrow ret$.

2. $\langle f; \texttt{intFromPtr}\,(\tau_1, \tau_2, mem\_ptr)\rangle \longrightarrow \langle f; \texttt{done}\,mem\_int\rangle$.

PROVE: $\cdot;\cdot;\cdot;\cdot \vdash \texttt{done}\,mem\_int \Leftarrow ret$

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

⟨2⟩2. $\cdot;\cdot;\cdot \vdash mem\_int \Rightarrow \texttt{integer}$ 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: $pt = mem\_ptr \overset{\checkmark}{\mapsto}_\tau \_$.

$ret = \Sigma\,y{:}\texttt{bool}.\,y = \texttt{aligned}\,(\tau, mem\_ptr) \wedge pt \otimes \texttt{I}$.

$h = \cdot + \{pt\}$ so $h' = h$.

ASSUME: 1. $\cdot;\cdot;\cdot;\mathcal{R} \vdash \texttt{ptrValidForDeref}\,(\tau, mem\_ptr, pt) \Rightarrow ret$.

2. $\langle f{+}\{pt\}; \texttt{ptrValidForDeref}\,(\tau, mem\_ptr, pt)\rangle \longrightarrow \langle f{+}\{pt\}; \texttt{done}\,bool\_value, pt\rangle$.

PROVE: $\cdot;\cdot;\cdot;\cdot, \_{:}pt \vdash \texttt{done}\,bool\_value, pt \Leftarrow ret$.

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

Note: $\mathcal{R} = \cdot, \_{:}pt'$ where $\cdot \vdash pt' \equiv pt$.

$\langle 2 \rangle 2$. $bool\_value = \mathtt{aligned}\,(\tau, mem\_ptr)$ by construction of $bool\_value$ (inversion on 2).

$\langle 2 \rangle 3$. $\cdot;\cdot;\cdot \vdash bool\_value \Rightarrow \mathtt{bool}$ by Ty_PVal_{True,False}.

$\langle 2 \rangle 4$. By Ty_TVal_I, and then $\langle 2 \rangle 1$ – $\langle 2 \rangle 3$ with Ty_TVal_{Res,Phi,Comp} respectively, we are done.

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

$\langle 2 \rangle 1$. $\mathtt{smt}\,(\cdot \Rightarrow bool\_value = \mathtt{aligned}\,(\tau, mem\_ptr))$ by construction of $bool\_value$ (inversion on 2).

$\langle 2 \rangle 2$. $\cdot;\cdot;\cdot \vdash bool\_value \Rightarrow \mathtt{bool}$ by Ty_PVal_{True,False}.

$\langle 2 \rangle 3$. By Ty_TVal_I and $\langle 2 \rangle 1$ and $\langle 2 \rangle 2$ with Ty_TVal_{Phi,Comp} respectively, we are done.

$\langle 1 \rangle 23$. Case: Ty_Memop_PtrArrayShift.
Proof: Similiar to Ty_PE_Array_Shift, except with Ty_TVal_{I,Phi,Comp} at the end.

$\langle 1 \rangle 24$. Case: Ty_Seq_E_CCall.
Assume: 1. $\cdot;\cdot;\cdot;\mathcal{R} \vdash \mathtt{ccall}\,(\tau, ident, \overline{spine\_elem_i}^{\,i}) \Rightarrow \sigma(ret)$.
2. $\langle h + f; \mathtt{ccall}\,(\tau, ident, \overline{spine\_elem_i}^{\,i})\rangle \longrightarrow \langle h + f; \sigma'(texpr){:}\sigma'(ret)\rangle$.
Prove: $\cdot;\cdot;\cdot;\mathcal{R} \vdash \sigma(texpr) \Leftarrow \sigma(ret)$
(because the heap does not change).

$\langle 2 \rangle 1$. $ident{:}arg \equiv \overline{x_i}^{\,i} \mapsto texpr \in \mathtt{Globals}$ by inversion (on either assumption).

$\langle 2 \rangle 2$. $\cdot;\cdot;\cdot;\mathcal{R} \vdash \overline{x_i = spine\_elem_i}^{\,i} :: arg \gg \sigma; ret$ by inversion on 1.

$\langle 2 \rangle 3$. $\sigma = \sigma'$ and $ret = ret'$ by induction on arg.
Proof: Ty_Spine_* and Decons_Arg_* construct same substitution and return type (lemma 3.1).

$\langle 2 \rangle 4$. 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 $\langle 2 \rangle 2$ 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. $\mathtt{smt}\,(\cdot \Rightarrow term)$ for each $term \in \Phi$.

$\langle 2 \rangle 5$. $\mathcal{C}''; \mathcal{L}''; \Phi''; \mathcal{R}'' \vdash texpr \Leftarrow ret''$ 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.

$\langle 2 \rangle 6$. $\mathcal{C} = \mathcal{C}''$, $\Phi = \Phi''$, $\mathcal{L} = \mathcal{L}''$, $\mathcal{R}' = \mathcal{R}''$ and $ret = ret''$.
Proof: By induction on arg.

$\langle 2 \rangle 7$. Apply substitution lemma (2.5) to $\langle 2 \rangle 4$ and $\langle 2 \rangle 5$ 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. $\cdot;\cdot;\cdot \vdash \mathtt{let}\, ident\_or\_pattern = pexpr\, \mathtt{in}\, tpexpr \Leftarrow y_2{:}\beta_2.\, term_2$.
2. $\langle \mathtt{let}\, ident\_or\_pattern = pexpr\, \mathtt{in}\, tpexpr \rangle \longrightarrow \langle \mathtt{let}\, ident\_or\_pattern = pexpr'\, \mathtt{in}\, tpexpr \rangle$.
Prove: $\cdot;\cdot;\cdot \vdash \mathtt{let}\, ident\_or\_pattern = pexpr'\, \mathtt{in}\, tpexpr \Leftarrow y_2{:}\beta_2.\, term_2$
(because this is a pure expression, heaps are irrelevant).

⟨2⟩1. 1. $\cdot;\cdot;\cdot \vdash pexpr \Rightarrow y{:}\beta.\, term$.
2. $ident\_or\_pattern{:}\beta \rightsquigarrow \mathcal{C}_1\, \mathtt{with}\, term_1$.
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. $\cdot;\cdot;\cdot \vdash \mathtt{let}\, ident\_or\_pattern = pexpr'\, \mathtt{in}\, tpexpr \Leftarrow y_2{:}\beta_2.\, term_2$.
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. $\cdot;\cdot;\cdot;\mathcal{R}',\mathcal{R} \vdash \mathtt{let}\, \overline{ret\_pattern_i}^{\,i} = seq\_expr\, \mathtt{in}\, texpr_2 \Leftarrow ret_2$.
2. $\langle h{+}f; \mathtt{let}\, \overline{ret\_pattern_i}^{\,i} = seq\_expr\, \mathtt{in}\, texpr_2 \rangle \longrightarrow \langle h{+}f; \mathtt{let}\, \overline{ret\_pattern_i}^{\,i}{:}ret'_1 = texpr_1\, \mathtt{in}\, texpr_2 \rangle$.
Prove: $\cdot;\cdot;\cdot;\mathcal{R}',\mathcal{R} \vdash \mathtt{let}\, \overline{ret\_pattern_i}^{\,i}{:}ret_1 = texpr_1\, \mathtt{in}\, texpr_2 \Leftarrow ret_2$
(because the heap does not change).

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

⟨2⟩2. $\langle h; seq\_expr \rangle \longrightarrow \langle h; texpr_1{:}ret'_1 \rangle$.
Proof: By inversion on 2.

⟨2⟩3. $\cdot;\cdot;\cdot;\mathcal{R}' \vdash texpr_1 \Leftarrow ret_1$.
Proof: By induction on ⟨2⟩1.1 and ⟨2⟩2.

⟨2⟩4. $ret_1 = ret_1'$.
PROOF: By cases TY_SEQ_E_{CCALL,PCALL}.

⟨2⟩5. By TY_SEQ_TE_LET with ⟨2⟩1.2,3 and ⟨2⟩3, we are done.

⟨1⟩31. CASE: TY_SEQ_TE_LETT.
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. $\langle h{+}f; \texttt{let}\, \overline{ret\_pattern_i}^{\,i} : ret_1 = \texttt{done}\, \overline{spine\_elem_i}^{\,i}\, \texttt{in}\, texpr \rangle \longrightarrow \langle h{+}f; \sigma(texpr_2) \rangle$.
PROVE: $\cdot; \cdot; \cdot; \mathcal{R}', \mathcal{R} \vdash \sigma(texpr_2) \Leftarrow \sigma(ret_2)$
(because the heap does not change).

⟨2⟩1. 1. $\cdot; \cdot; \cdot; \mathcal{R}' \vdash \texttt{done}\, \overline{spine\_elem_i}^{\,i} \Leftarrow ret_1$.
2. $\Phi \vdash \overline{ret\_pattern_i}^{\,i} : ret_1 \leadsto \mathcal{C}_1; \mathcal{L}_1; \Phi_1; \mathcal{R}_1$.
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.

⟨2⟩2. $\overline{ret\_pattern_i = spine\_elem_i}^{\,i} \leadsto \sigma$.
PROOF: By inversion on 2.

⟨2⟩3. $\cdot; \cdot; \cdot; \mathcal{R}' \vdash (\sigma) : (\mathcal{C}_1; \mathcal{L}_1; \Phi_1; \mathcal{R}_1)$.
PROOF: By ⟨2⟩1.1,2 and ⟨2⟩2 using lemma 4.5 (deconstructing a pattern produces a well-typed substitution).

⟨2⟩4. By ⟨2⟩1.3 and ⟨2⟩3 and the let-friendly substitution lemma 2.7, we are done.

⟨1⟩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. $\langle h{+}f; \texttt{let}\, \overline{ret\_pattern_i}^{\,i} : ret = texpr_1\, \texttt{in}\, texpr_2 \rangle \longrightarrow \langle h'; \texttt{let}\, \overline{ret\_pattern_i}^{\,i} : ret = texpr_1'\, \texttt{in}\, texpr_2 \rangle$.
PROVE: $\exists h'' : \mathcal{R}'', \mathcal{R}.\ h' = h'' + f$
$\wedge\ \cdot; \cdot; \cdot; \mathcal{R}'', \mathcal{R} \vdash \texttt{let}\, \overline{ret\_pattern_i}^{\,i} : ret_1 = texpr_1'\, \texttt{in}\, texpr_2 \Leftarrow ret_2$.

⟨2⟩1. 1. $\cdot; \cdot; \cdot; \mathcal{R}' \vdash texpr_1 \Leftarrow ret_1$.
2. $\Phi \vdash \overline{ret\_pattern_i}^{\,i} : ret_1 \leadsto \mathcal{C}_1; \mathcal{L}_1; \Phi_1; \mathcal{R}_1$.
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.

⟨2⟩2. $\langle h + f; texpr_1 \rangle \longrightarrow \langle h'; texpr_1' \rangle$.
PROOF: By inversion on 2.

⟨2⟩3. $h = h_1 + h_2$ where $h_1 : \mathcal{R}'$ and $h_2 : \mathcal{R}$.
PROOF: By induction on $\mathcal{R}$.

⟨2⟩4. $\exists h_1' : R''.\ h' = h_1' + h_2 + f \wedge \cdot; \cdot; \cdot; \mathcal{R}'' \vdash texpr_1' \Leftarrow ret_1$.
PROOF: By induction with $h_1 : \mathcal{R}'$ and $h2 + f$ as the frame, using ⟨2⟩1.1 and ⟨2⟩2.

⟨2⟩5. By ⟨2⟩3, ⟨2⟩2.2,3 using TY_SEQ_TE_LETT, and $h'' = h_1' + h_2$ (so $h'' : \mathcal{R}'', \mathcal{R}$) 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. $\langle h + f; \texttt{case}\, pval\, \texttt{of}\, \overline{\mid pattern_i \Rightarrow texpr_i}^{\,i}\, \texttt{end} \rangle \longrightarrow \langle h + f; \sigma_j(texpr_j) \rangle$.
PROVE: $\cdot; \cdot; \cdot; \mathcal{R} \vdash \sigma_j(texpr_j) \Leftarrow ret$

(because the heap does not change).

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

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

⟨2⟩3. $\mathit{term}_j = \mathit{pval}$.
PROOF: By ⟨2⟩1.2 and terms derived from patterns are "equal to" matching values (lemma 4.2).

⟨2⟩4. $\cdot;\cdot;\cdot;\cdot \vdash (\sigma_j){:}(\mathcal{C}_j;\cdot;\cdot,\mathit{term}_j = \mathit{pval};\cdot)$.
PROOF: By ⟨2⟩3 and lemma 4.5 (deconstructing a pattern produces a well-typed substitution).

⟨2⟩5. By ⟨2⟩4, ⟨2⟩1.3 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. $\cdot;\cdot;\cdot;\mathcal{R} \vdash \texttt{if}\,\texttt{True}\,\texttt{then}\,\mathit{texpr}_1\,\texttt{else}\,\mathit{texpr}_2 \Leftarrow \mathit{ret}$.
2. $\langle h+f;\texttt{if}\,\texttt{True}\,\texttt{then}\,\mathit{texpr}_1\,\texttt{else}\,\mathit{texpr}_2\rangle \longrightarrow \langle h+f;\mathit{texpr}_1\rangle$.
PROVE: $\cdot;\cdot;\cdot;\mathcal{R} \vdash \mathit{texpr}_1 \Leftarrow \mathit{ret}$
(because the heap does not change).
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.

5 Typing Judgements

object_value_jtype	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{object\_value} \Rightarrow \mathtt{obj}\,\beta$
pval_jtype	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{pval} \Rightarrow \beta$
res_jtype	::=	
	|	$\Phi \vdash \mathit{res} \equiv \mathit{res}'$
	|	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term} \Leftarrow \mathit{res}$
	|	$h{:}\mathcal{R}$
spine_jtype	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{x_i = \mathit{spine\_elem}_i}^{\,i} :: \mathit{arg} \gg \sigma; \mathit{ret}$
pexpr_jtype	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{pexpr} \Rightarrow \mathit{ident}{:}\beta.\,\mathit{term}$
tpval_jtype	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{tpval} \Leftarrow \mathit{ident}{:}\beta.\,\mathit{term}$
tpexpr_jtype	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{tpexpr} \Leftarrow \mathit{ident}{:}\beta.\,\mathit{term}$
action_jtype	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{mem\_action} \Rightarrow \mathit{ret}$
memop_jtype	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{mem\_op} \Rightarrow \mathit{ret}$
seq_expr_jtype	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{seq\_expr} \Rightarrow \mathit{ret}$
is_expr_jtype	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{is\_expr} \Rightarrow \mathit{ret}$
tval_jtype	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{tval} \Leftarrow \mathit{ret}$
texpr_jtype	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{seq\_texpr} \Leftarrow \mathit{ret}$
	|	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{is\_texpr} \Leftarrow \mathit{ret}$
	|	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{texpr} \Leftarrow \mathit{ret}$

6 Opsem Judgements

pure_opsem_jtype	::=	
	|	$\langle pexpr \rangle \longrightarrow \langle pexpr' \rangle$
	|	$\langle pexpr \rangle \longrightarrow \langle tpexpr{:}(y{:}\beta.\, term) \rangle$
	|	$\langle tpexpr \rangle \longrightarrow \langle tpexpr' \rangle$
opsem_jtype	::=	
	|	$\langle h; seq\_expr \rangle \longrightarrow \langle h'; texpr{:}ret \rangle$
	|	$\langle h; seq\_texpr \rangle \longrightarrow \langle h'; texpr \rangle$
	|	$\langle h; mem\_op \rangle \longrightarrow \langle h'; tval \rangle$
	|	$\langle h; mem\_action \rangle \longrightarrow \langle h'; tval \rangle$
	|	$\langle h; is\_expr \rangle \longrightarrow \langle h'; is\_expr' \rangle$
	|	$\langle h; is\_texpr \rangle \longrightarrow \langle h'; texpr \rangle$
	|	$\langle h; texpr \rangle \longrightarrow \langle h'; texpr' \rangle$