Explicit CN Soundness Proof

Dhruv Makwana

July 23, 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 $\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash J$.

Proof Strategy: Induction over the typing judgements.

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

2 Substitution

2.1 Weakening for Substitution

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

Proof Strategy: Induction over the substitution.

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

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. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term} \Leftarrow \mathit{res}$.
3. $\Phi \vdash \sigma(\mathit{res}) \equiv \sigma(\mathit{res}')$.

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.

$\langle 1 \rangle 3$. CASE: TY_RES_VAR
PROOF: By transitivity (lemma 2.3).

$\langle 1 \rangle 4$. CASE: TY_RES_SEPCONJ
PROOF: By induction.

$\langle 1 \rangle 5$. CASE: TY_RES_CONJ
PROOF: We know $\texttt{smt}\,(\Phi \Rightarrow (\mathit{term} \to \mathit{term}'))$ (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.

$\langle 1 \rangle 6$. CASE: TY_RES_PACK
$\mathit{res\_term} = \texttt{pack}\,(\mathit{pval}, \mathit{res\_term}')$, $\mathit{res} = \exists\, y{:}\beta.\ \mathit{res}_1$, $\mathit{res}' = \exists\, y{:}\beta.\ \mathit{res}_1'$.

$\langle 2 \rangle 1$. $\Phi \vdash \mathit{res}_1 \equiv \mathit{res}_1'$ by inversion on the equality.

$\langle 2 \rangle 2$. $\Phi \vdash \mathit{pval}/y, \cdot(\mathit{res}_1) \equiv \mathit{pval}/y, \cdot(\mathit{res}_1')$ by inversion on substitution assumption.

$\langle 2 \rangle 3$. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term}' \Leftarrow \mathit{pval}/y, \cdot(\mathit{res}_1')$ by induction.

$\langle 2 \rangle 4$. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \texttt{pack}\,(\mathit{pval}, \mathit{res\_term}') \Leftarrow \exists\, y{:}\beta.\ \mathit{res}_1'$ 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. $\mathcal{C}; \mathcal{L}; \Phi \vdash (\,\texttt{struct}\, \mathit{tag})\{\, \overline{.\,\mathit{member}_i = \sigma(\mathit{pval}_i)}^{\,i}\, \} \Rightarrow \texttt{struct}\, \mathit{tag}$

⟨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.
$\mathcal{C}'; \mathcal{L}'; \Phi'; \cdot, \mathit{pt} \vdash \mathit{pt}' \Leftarrow \mathit{pt}''$.
PROVE: $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(\mathit{pt}') \Leftarrow \sigma(\mathit{pt}'')$.

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

⟨2⟩5. $\mathit{res\_term} = pt_3$ 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⟩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.
$\mathcal{C}'; \mathcal{L}'; \Phi'; \cdot, r{:}\mathit{res} \vdash r \Leftarrow \mathit{res}'$.

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

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

⟨2⟩3. $\Phi' \vdash \mathit{res} \equiv \mathit{res}'$ by inversion on TY_RES_VAR.

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

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

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

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

⟨2⟩8. SUFFICES: $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term} \Leftarrow \sigma'(\mathit{res}')$ 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.
$\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash \mathit{res\_term} \Leftarrow \mathit{term} \wedge \mathit{res}$.

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

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

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

⟨1⟩16. CASE: TY_RES_PACK.

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

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

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

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

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

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

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

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

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

⟨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. $\texttt{smt}\,(\Phi \Rightarrow \sigma(\mathit{term})).$
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.
$\mathcal{C}'; \mathcal{L}'; \Phi' \vdash \texttt{array\_shift}\,(\mathit{pval}_1, \tau, \mathit{pval}_2) \Rightarrow y{:}\texttt{loc}.\ y = \mathit{pval}_1 +_{\mathrm{ptr}} (\mathit{pval}_2 \times \mathrm{size\_of}(\tau))$

⟨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. Rules with SMT checks still hold because they are true for all (well-typed) instantiations of a term's free variables, and $\texttt{smt}\,(\Phi \Rightarrow term)$ for each $term \in \Phi'$.

⟨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. $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(\mathit{pval}) \Rightarrow \beta$
2. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(\texttt{done}\, \overline{\mathit{spine\_elem}_i}^{\,i}) \Leftarrow \sigma(\mathit{pval}/y, \cdot(\mathit{ret}))$.

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

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

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

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

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

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

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

⟨2⟩3. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \texttt{ccall}\,(\tau, \mathit{ident}, \overline{\sigma(\mathit{spine\_elem}_i)}^{\,i}) \Rightarrow \sigma, \psi(\mathit{ret})$ 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.
$\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}''', \mathcal{R}'' \vdash \texttt{let}\, \overline{\mathit{ret\_pattern}_i}^{\,i} = \mathit{seq\_expr}\, \texttt{in}\, \mathit{texpr} \Leftarrow \mathit{ret}_2$.

⟨2⟩1. By induction,
1. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}' \vdash \sigma(\mathit{seq\_expr}) \Rightarrow \sigma(\mathit{ret}_1)$.
2. $\mathcal{C}, \mathcal{C}_1; \mathcal{L}, \mathcal{L}_1; \Phi, \Phi_1; \mathcal{R}, \mathcal{R}_1 \vdash \sigma(\mathit{texpr}) \Leftarrow \sigma(\mathit{ret}_2)$.

⟨2⟩2. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}', \mathcal{R} \vdash \sigma(\texttt{let}\, \overline{\mathit{ret\_pattern}_i}^{\,i} = \mathit{seq\_expr}\, \texttt{in}\, \mathit{texpr}) \Leftarrow \sigma(\mathit{ret}_2)$ as required.

⟨1⟩44. Case: Ty_Seq_TE_Case.

$\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash \texttt{case}\, \mathit{pval}\, \texttt{of}\, \overline{\mid \mathit{pattern}_i \Rightarrow \mathit{texpr}_i}^{\,i}\, \texttt{end} \Leftarrow \mathit{ret}.$

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

⟨2⟩2. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(\texttt{case}\, \mathit{pval}\, \texttt{of}\, \overline{\mid \mathit{pattern}_i \Rightarrow \mathit{texpr}_i}^{\,i}\, \texttt{end}) \Leftarrow \sigma(\mathit{ret})$ 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: $\mathcal{C}; \mathcal{L}; \sigma(\Phi); \mathcal{R}_1, \mathcal{R} \vdash \sigma(J)$.

⟨1⟩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.

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

⟨1⟩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 $\forall h : R.\ \exists e', h'.\ \langle h; e \rangle \longrightarrow \langle h'; e' \rangle$.

⟨1⟩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).

⟨1⟩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.

⟨1⟩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 $\mathit{name}{:}\mathit{pure\_arg} \equiv \overline{x_i}^{\,i} \mapsto \mathit{tpexpr} \in \texttt{Globals}$ and $\cdot; \cdot; \cdot; \cdot \vdash \overline{x_i = \mathit{pval}_i}^{\,i} :: \mathit{pure\_arg} \gg \sigma; \Sigma\, y{:}\beta.\ \mathit{term} \wedge \texttt{I}$, with the latter implying $\overline{x_i = \mathit{pval}_i}^{\,i} :: \mathit{pure\_arg} \gg \sigma; \Sigma\, y{:}\beta.\ \mathit{term} \wedge \texttt{I}$ (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 $\texttt{smt}\,(\Phi \Rightarrow \mathit{pval})$. 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 $ident{:}arg \equiv \overline{x_i}^{\,i} \mapsto texpr \in$ `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.
$\mathit{res\_pattern} = \mathit{res\_term} = \mathit{res} = \mathtt{emp}$. $\sigma, \mathcal{L}, \Phi, \mathcal{R}, \mathcal{R}'$ are all empty.
PROOF: By TY_SUBS_EMPTY, we are done.

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

⟨1⟩3. CASE: TY_PAT_RES_VAR.
$\mathit{res\_pattern} = r$, $\sigma = \mathit{res\_term}/x, \cdot$, $\mathcal{L} = \cdot$, $\Phi = \cdot$, $\mathcal{R}' = \cdot, x{:}\mathit{res}$.
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 $\mathtt{smt}\,(\cdot \Rightarrow \mathit{term})$ (from 1) and induction with TY_SUB_CONS_PHI.

⟨1⟩6. CASE: TY_PAT_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.

Now, full proof.

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

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 $\mathtt{smt}\,(\cdot \Rightarrow \mathit{term})$
(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: $\mathit{term} = \mathit{mem\_ptr} +_{\text{ptr}} (\mathit{mem\_int} \times \text{size\_of}(\tau))$.
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: $\cdot;\cdot;\cdot \vdash \mathit{mem\_ptr}' \Rightarrow y{:}\texttt{loc}.\ y = \mathit{term}$.
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: $\cdot;\cdot;\cdot \vdash \mathit{bool\_value}' \Rightarrow y{:}\texttt{bool}.\ y = \neg\,\mathit{bool\_value}$.
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: $\cdot;\cdot;\cdot \vdash \mathit{mem\_int} \Rightarrow y{:}\texttt{integer}.\ y = \mathit{term}$.
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: $\cdot;\cdot;\cdot \vdash \texttt{Unit} \Rightarrow y{:}\texttt{unit}.\ y = \texttt{unit}$.
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: $\mathit{pt} = \mathit{mem\_ptr} \overset{\times}{\mapsto}_\tau \mathit{pval}$.
$\mathit{term} = \texttt{representable}\,(\tau*, y_p) \land \texttt{alignedI}\,(\mathit{mem\_int}, y_p)$.
$\mathit{ret} = \Sigma\, y_p{:}\texttt{loc}.\ \mathit{term} \land \exists\, y{:}\beta_\tau.\ y_p \overset{\times}{\mapsto}_\tau y \otimes \texttt{I}$.
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. $\cdot;\cdot;\cdot;\cdot, \mathit{pt} \vdash \mathit{pt} \Leftarrow \mathit{pt}$ 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: $\mathit{pt} = \mathit{mem\_ptr} \overset{\checkmark}{\mapsto}_\tau \mathit{pval}$.
$\mathit{ret} = \Sigma\, y{:}\beta_\tau.\ y = \mathit{pval} \land \mathit{pt} \otimes \texttt{I}$.
ASSUME: 1. $\cdot;\cdot;\cdot;\cdot, \mathit{pt} \vdash \texttt{load}\,(\tau, \mathit{mem\_ptr}, \_, \mathit{pt}) \Rightarrow \mathit{ret}$.
2. $\langle \cdot + \{\mathit{pt}\}; \texttt{load}\,(\tau, \mathit{mem\_ptr}, \_, \mathit{pt})\rangle \longrightarrow \langle \cdot + \{\mathit{pt}\}; \texttt{done}\,\mathit{pval}, \mathit{pt}\rangle$.
PROVE: $\cdot;\cdot;\cdot;\cdot, \mathit{pt} \vdash \texttt{done}\,\mathit{pval}, \mathit{pt} \Leftarrow \mathit{ret}$

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

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

⟨2⟩3. $\cdot;\cdot;\cdot \vdash \mathit{pval} \Rightarrow \beta_\tau$ 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. $\cdot; \cdot; \cdot; \cdot, pt \vdash \mathtt{store}\,(\_, \tau, pval_0, pval_1, \_, pt) \Rightarrow ret$.
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. $\cdot; \cdot; \cdot \vdash \mathtt{Unit} \Rightarrow \mathtt{unit}$ by TY_PVAL_UNIT.

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

⟨2⟩3. By TY_TVAL_I and ⟨2⟩1 and ⟨2⟩2 with TY_TVAL_{RES,COMP} respectively, we are done.

⟨1⟩17. CASE: TY_ACTION_KILL_STATIC.
LET: $pt = mem\_ptr \mapsto_\tau \_$.
ASSUME: 1. $\cdot; \cdot; \cdot; \cdot, pt \vdash \mathtt{kill}\,(\mathtt{static}\,\tau, pval_0, pt) \Rightarrow \Sigma \_{:}\mathtt{unit}.\ \mathtt{I}$.
2. $\langle \cdot + \{pt\}; \mathtt{kill}\,(\mathtt{static}\,\tau, mem\_ptr, pt)\rangle \longrightarrow \langle h; \mathtt{done}\,\mathtt{Unit}\rangle$.
PROVE: $\cdot; \cdot; \cdot \vdash \mathtt{done}\,\mathtt{Unit} \Leftarrow \Sigma \_{:}\mathtt{unit}.\ \mathtt{I}$
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: $ret = \Sigma\, y{:}\mathtt{integer}.\ y = \mathtt{cast\_ptr\_to\_int}\, mem\_ptr \wedge \mathtt{I}$.
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. $\mathtt{smt}\,(\cdot \Rightarrow mem\_int = \mathtt{cast\_ptr\_to\_int}\, mem\_ptr)$ by construction of mem_int (inversion on 2).

⟨2⟩2. $\cdot; \cdot; \cdot \vdash mem\_int \Rightarrow \mathtt{integer}$ 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: $pt = mem\_ptr \overset{\checkmark}{\mapsto}_\tau \_$.
$ret = \Sigma\, y{:}\mathtt{bool}.\ y = \mathtt{aligned}\,(\tau, mem\_ptr) \wedge pt \otimes \mathtt{I}$.
ASSUME: 1. $\cdot; \cdot; \cdot; \mathcal{R} \vdash \mathtt{ptrValidForDeref}\,(\tau, mem\_ptr, pt) \Rightarrow ret$.
2. $\langle \cdot + \{pt\}; \mathtt{ptrValidForDeref}\,(\tau, mem\_ptr, pt)\rangle \longrightarrow \langle \cdot + \{pt\}; \mathtt{done}\, bool\_value, pt\rangle$.
PROVE: $\cdot; \cdot; \cdot; \mathcal{R} \vdash \mathtt{done}\, bool\_value, pt \Leftarrow ret$.

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

$\langle 2\rangle 2.$ $R = \cdot, pt$, by TY_RES_POINTSTO.

$\langle 2\rangle 3.$ $\mathit{bool\_value} = \texttt{aligned}\,(\tau, \mathit{mem\_ptr})$ by construction of $\mathit{bool\_value}$ (inversion on 2).

$\langle 2\rangle 4.$ $\cdot; \cdot; \cdot \vdash \mathit{bool\_value} \Rightarrow \texttt{bool}$ by TY_PVAL_{TRUE,FALSE}.

$\langle 2\rangle 5.$ By TY_TVAL_I, and then $\langle 2\rangle 2$ – $\langle 2\rangle 4$ with TY_TVAL_{RES,PHI,COMP} respectively, we are done.

$\langle 1\rangle 22.$ 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: $\cdot; \cdot; \cdot; \cdot \vdash \texttt{done}\,\mathit{bool\_value} \Rightarrow \mathit{ret}$.

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

$\langle 2\rangle 2.$ $\cdot; \cdot; \cdot \vdash \mathit{bool\_value} \Rightarrow \texttt{bool}$ 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. $\cdot; \cdot; \cdot; \mathcal{R} \vdash \texttt{ccall}\,(\tau, \mathit{ident}, \overline{\mathit{spine\_elem}_i}^{\,i}) \Rightarrow \sigma(\mathit{ret})$.
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: $\cdot; \cdot; \cdot; \mathcal{R} \vdash \sigma(\mathit{texpr}) \Leftarrow \sigma(\mathit{ret})$

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

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

$\langle 2\rangle 3.$ $\sigma = \sigma'$ and $\mathit{ret} = \mathit{ret}'$ 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. $\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{pval}_i \Rightarrow \beta_i$ for each $x_i{:}\beta_i \in \mathcal{C}$ or $x_i{:}\beta_i \in \mathcal{L}$.
2. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}' \vdash \mathit{res\_term}_i \Leftarrow \mathit{res}_i$ for each $\mathit{res}_i \in \mathcal{R}'$.
3. $\texttt{smt}\,(\cdot \Rightarrow \mathit{term})$ for each $\mathit{term} \in \Phi$.

$\langle 2\rangle 5.$ $\mathcal{C}''; \mathcal{L}''; \Phi''; \mathcal{R}'' \vdash \mathit{texpr} \Leftarrow \mathit{ret}''$ where $\overline{x_i}^{\,i} :: \mathit{arg} \rightsquigarrow \mathcal{C}''; \mathcal{L}''; \Phi''; \mathcal{R}'' \mid \mathit{ret}''$ 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 $\mathit{ret} = \mathit{ret}''$.
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.

$\langle 1\rangle 25.$ 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 $\langle \mathit{pexpr} \rangle \longrightarrow \langle \mathit{pexpr}' \rangle$ here.
See TY_SEQ_TE_LET for a more general version and proof for the remaining $\langle \mathit{pexpr} \rangle \longrightarrow \langle \mathit{tpexpr}{:}(y{:}\beta.\ \mathit{term}) \rangle$ case.
ASSUME: 1. $\cdot;\cdot;\cdot \vdash \texttt{let}\,\mathit{ident\_or\_pattern} = \mathit{pexpr}\,\texttt{in}\,\mathit{tpexpr} \Leftarrow y_2{:}\beta_2.\ \mathit{term}_2$.
2. $\langle \texttt{let}\,\mathit{ident\_or\_pattern} = \mathit{pexpr}\,\texttt{in}\,\mathit{tpexpr} \rangle \longrightarrow \langle \texttt{let}\,\mathit{ident\_or\_pattern} = \mathit{pexpr}'\,\texttt{in}\,\mathit{tpexpr} \rangle$.
PROVE: $\cdot;\cdot;\cdot \vdash \texttt{let}\,\mathit{ident\_or\_pattern} = \mathit{pexpr}'\,\texttt{in}\,\mathit{tpexpr} \Leftarrow y_2{:}\beta_2.\ \mathit{term}_2$.

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

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

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

⟨2⟩4. $\cdot;\cdot;\cdot \vdash \texttt{let}\,\mathit{ident\_or\_pattern} = \mathit{pexpr}'\,\texttt{in}\,\mathit{tpexpr} \Leftarrow y_2{:}\beta_2.\ \mathit{term}_2$.
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. $\cdot;\cdot;\cdot;\mathcal{R}',\mathcal{R} \vdash \texttt{let}\,\overline{\mathit{ret\_pattern}_i}^{\,i} = \mathit{seq\_expr}\,\texttt{in}\,\mathit{texpr}_2 \Leftarrow \mathit{ret}_2$.
2. $\langle h; \texttt{let}\,\overline{\mathit{ret\_pattern}_i}^{\,i} = \mathit{seq\_expr}\,\texttt{in}\,\mathit{texpr}_2 \rangle \longrightarrow \langle h; \texttt{let}\,\overline{\mathit{ret\_pattern}_i}^{\,i}{:}\mathit{ret}_1' = \mathit{texpr}_1\,\texttt{in}\,\mathit{texpr}_2 \rangle$.
PROVE: $\cdot;\cdot;\cdot;\mathcal{R}',\mathcal{R} \vdash \texttt{let}\,\overline{\mathit{ret\_pattern}_i}^{\,i}{:}\mathit{ret}_1 = \mathit{texpr}_1\,\texttt{in}\,\mathit{texpr}_2 \Leftarrow \mathit{ret}_2$.

⟨2⟩1. 1. $\cdot;\cdot;\cdot;\mathcal{R}' \vdash \mathit{seq\_expr} \Rightarrow \mathit{ret}_1$.
2. $\overline{\mathit{ret\_pattern}_i}^{\,i}{:}\mathit{ret}_1 \rightsquigarrow \mathcal{C}_1;\mathcal{L}_1;\Phi_1;\mathcal{R}_1$.
3. $\mathcal{C}_1;\mathcal{L}_1;\Phi_1;\mathcal{R},\mathcal{R}_1 \vdash \mathit{texpr} \Leftarrow \mathit{ret}_2$.
PROOF: By inversion on 1.

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

⟨2⟩3. $\cdot;\cdot;\cdot;\mathcal{R}' \vdash \mathit{texpr}_1 \Leftarrow \mathit{ret}_1$.
PROOF: By induction on ⟨2⟩1.1 and ⟨2⟩2.

⟨2⟩4. $\mathit{ret}_1 = \mathit{ret}_1'$.
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. $\cdot;\cdot;\cdot;\mathcal{R}',\mathcal{R} \vdash \texttt{let}\,\overline{\mathit{ret\_pattern}_i}^{\,i}{:}\mathit{ret}_1 = \texttt{done}\,\overline{\mathit{spine\_elem}_i}^{\,i}\,\texttt{in}\,\mathit{texpr}_2 \Leftarrow \mathit{ret}_2$.
2. $\langle h;\texttt{let}\,\overline{\mathit{ret\_pattern}_i}^{\,i}{:}\mathit{ret}_1 = \texttt{done}\,\overline{\mathit{spine\_elem}_i}^{\,i}\,\texttt{in}\,\mathit{texpr}\rangle \longrightarrow \langle h;\sigma(\mathit{texpr}_2)\rangle$.
PROVE: $\cdot;\cdot;\cdot;\mathcal{R}',\mathcal{R} \vdash \sigma(\mathit{texpr}_2) \Leftarrow \sigma(\mathit{ret}_2)$.

$\langle 2\rangle 1$. 1. $\cdot;\cdot;\cdot;\mathcal{R}' \vdash \texttt{done}\,\overline{\mathit{spine\_elem}_i}^{\,i} \Leftarrow \mathit{ret}_1$.
2. $\overline{\mathit{ret\_pattern}_i}^{\,i}{:}\mathit{ret}_1 \rightsquigarrow \mathcal{C}_1;\mathcal{L}_1;\Phi_1;\mathcal{R}_1$.
3. $\mathcal{C}_1;\mathcal{L}_1;\Phi_1;\mathcal{R}_1,\mathcal{R} \vdash \mathit{texpr}_2 \Leftarrow \mathit{ret}_2$.
PROOF: By inversion on 1.

$\langle 2\rangle 2$. $\overline{\mathit{ret\_pattern}_i = \mathit{spine\_elem}_i}^{\,i} \rightsquigarrow \sigma$.
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. $\cdot;\cdot;\cdot;\mathcal{R}',\mathcal{R} \vdash \texttt{let}\,\overline{\mathit{ret\_pattern}_i}^{\,i}{:}\mathit{ret}_1 = \mathit{texpr}_1\,\texttt{in}\,\mathit{texpr}_2 \Leftarrow \mathit{ret}_2$.
2. $\langle h;\texttt{let}\,\overline{\mathit{ret\_pattern}_i}^{\,i}{:}\mathit{ret} = \mathit{texpr}_1\,\texttt{in}\,\mathit{texpr}_2\rangle \longrightarrow \langle h';\texttt{let}\,\overline{\mathit{ret\_pattern}_i}^{\,i}{:}\mathit{ret} = \mathit{texpr}'_1\,\texttt{in}\,\mathit{texpr}_2\rangle$.
PROVE: $\cdot;\cdot;\cdot;\mathcal{R}'',\mathcal{R} \vdash \texttt{let}\,\overline{\mathit{ret\_pattern}_i}^{\,i}{:}\mathit{ret}_1 = \mathit{texpr}'_1\,\texttt{in}\,\mathit{texpr}_2 \Leftarrow \mathit{ret}_2$.

$\langle 2\rangle 1$. 1. $\cdot;\cdot;\cdot;\mathcal{R}' \vdash \mathit{texpr}_1 \Leftarrow \mathit{ret}_1$.
2. $\overline{\mathit{ret\_pattern}_i}^{\,i}{:}\mathit{ret}_1 \rightsquigarrow \mathcal{C}_1;\mathcal{L}_1;\Phi_1;\mathcal{R}_1$.
3. $\mathcal{C}_1;\mathcal{L}_1;\Phi_1;\mathcal{R}_1,\mathcal{R} \vdash \mathit{texpr}_2 \Leftarrow \mathit{ret}_2$.
PROOF: By inversion on 1.

$\langle 2\rangle 2$. $\langle h;\mathit{texpr}_1\rangle \longrightarrow \langle h';\mathit{texpr}'_1\rangle$.
PROOF: By inversion on 2.

$\langle 2\rangle 3$. $\cdot;\cdot;\cdot;\mathcal{R}'' \vdash \mathit{texpr}'_1 \Leftarrow \mathit{ret}_1$.
PROOF: By induction on $\langle 1\rangle 32.1$ and $\langle 2\rangle 2$.

$\langle 2\rangle 4$. By $\langle 2\rangle 3$, $\langle 1\rangle 32.2,3$ using TY_SEQ_TE_LETT, we are done.

$\langle 1\rangle 33$. CASE: TY_SEQ_TE_CASE.
ASSUME: 1. $\cdot;\cdot;\cdot;\mathcal{R} \vdash \texttt{case}\,\mathit{pval}\,\texttt{of}\,\overline{\mid \mathit{pattern}_i \Rightarrow \mathit{texpr}_i}^{\,i}\,\texttt{end} \Leftarrow \mathit{ret}$.
2. $\langle h;\texttt{case}\,\mathit{pval}\,\texttt{of}\,\overline{\mid \mathit{pattern}_i \Rightarrow \mathit{texpr}_i}^{\,i}\,\texttt{end}\rangle \longrightarrow \langle h;\sigma_j(\mathit{texpr}_j)\rangle$.
PROVE: $\cdot;\cdot;\cdot;\mathcal{R} \vdash \sigma_j(\mathit{texpr}_j) \Leftarrow \mathit{ret}$.

$\langle 2\rangle 1$. 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.

$\langle 2\rangle 2$. 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 ⟨1⟩32.2 and terms derived from patterns are "equal to" matching values (lemma 5.2).

⟨2⟩4. $\cdot; \cdot; \cdot; \cdot \vdash (\sigma_j){:}(\mathcal{C}_j; \cdot; \cdot, \mathit{term}_j = \mathit{pval}; \cdot)$.
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. $\cdot; \cdot; \cdot; \mathcal{R} \vdash \texttt{if}\ \texttt{True}\ \texttt{then}\ \mathit{texpr}_1\ \texttt{else}\ \mathit{texpr}_2 \Leftarrow \mathit{ret}$.
2. $\langle h; \texttt{if}\ \texttt{True}\ \texttt{then}\ \mathit{texpr}_1\ \texttt{else}\ \mathit{texpr}_2\rangle \longrightarrow \langle h; \mathit{texpr}_1\rangle$.
PROVE: $\cdot; \cdot; \cdot; \mathcal{R} \vdash \mathit{texpr}_1 \Leftarrow \mathit{ret}$.
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

object_value_jtype	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{object\_value} \Rightarrow \texttt{obj}\, \beta$
pval_jtype	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{pval} \Rightarrow \beta$
res_jtype	::=	
	|	$\Phi \vdash \mathit{res} \equiv \mathit{res}'$
	|	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term} \Leftarrow \mathit{res}$
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	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{seq\_texpr} \Leftarrow \mathit{ret}$
	|	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{is\_texpr} \Leftarrow \mathit{ret}$
	|	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{texpr} \Leftarrow \mathit{ret}$

7 Opsem Judgements

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

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