Explicit CN Soundness Proof

Dhruv Makwana

July 14, 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 $J = (\sigma) : (\mathcal{C}''; \mathcal{L}''; \mathcal{R}'')$.

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

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

$\langle 1 \rangle 1$. 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 $\exists pval.\ \mathcal{C}; \mathcal{L}; \Phi \vdash pval \Rightarrow \beta$ by Ty_Subs_Cons_{Comp,Log}.

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

⟨1⟩4. Case: Ty_PVal_Error.
Proof: We know $\texttt{smt}\,(\Phi' \Rightarrow \texttt{false})$ and $\texttt{smt}\,(\Phi \Rightarrow term)$ for each $term \in \Phi'$ so $\texttt{smt}\,(\Phi \Rightarrow \texttt{false})$.

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

⟨2⟩1. $\overline{\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(pval_i) \Rightarrow \beta_{\tau_i}}^{\,i}$ 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_PE_Val
Proof: By induction.

⟨1⟩7. Case: Ty_PE_Array_Shift.
$\mathcal{C}'; \mathcal{L}'; \Phi' \vdash \texttt{array\_shift}\,(pval_1, \tau, pval_2) \Rightarrow y{:}\texttt{loc}.\ y = pval_1 +_{\text{ptr}} (pval_2 \times \text{size\_of}(\tau))$

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

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

⟨1⟩8. Case: Ty_PE_Member_Shift.
Proof: Similar to Ty_PE_Array_Shift.

⟨1⟩9. Case: Ty_PE_{Not,Arith_Binop,Rel_Binop,Bool_Binop}.
Proof: By induction.

⟨1⟩10. Case: Ty_PE_Call.
See Ty_Seq_E_CCall for more general case and proof.

⟨1⟩11. Case: Ty_PE_{Assert_Undef,Bool_To_Integer,WrapI}.
Proof: By induction.

⟨1⟩12. Case: Ty_TPVal_Undef
See Ty_TVal_Undef for a more general case and proof.

⟨1⟩13. Case: Ty_TPVal_Done

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

⟨2⟩1. By induction,
1. $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(pval) \Rightarrow \beta.$
2. $\texttt{smt}\,(\Phi \Rightarrow \sigma, pval/y, \cdot(term)).$

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

⟨1⟩14. Case: Ty_TPE_{Let,LetT}.
See Ty_Seq_TE_{Let,LetT} for a more general case and proof.

⟨1⟩15. Case: Ty_TPE_If.
Proof: By induction.

⟨1⟩16. Case: Ty_TPE_Case.
$\mathcal{C}'; \mathcal{L}'; \Phi' \vdash \texttt{case}\, pval\, \texttt{of} \overline{\mid pattern_i \Rightarrow tpexpr_i}^{\,i}\, \texttt{end} \Leftarrow y_2{:}\beta_2.\ term_2.$

⟨2⟩1. By induction,
1. $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(pval) \Rightarrow \beta_1.$
2. $\overline{pattern_i{:}\beta_1 \rightsquigarrow \mathcal{C}_i\, \texttt{with}\, term_i}^{\,i}.$
3. $\overline{\mathcal{C}, \mathcal{C}_i; \mathcal{L}; \Phi, term_i = pval \vdash \sigma(tpexpr_i) \Leftarrow y_2{:}\beta_2.\ \sigma(term_2)}^{\,i}.$

⟨2⟩2. $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(\texttt{case}\, pval\, \texttt{of} \overline{\mid pattern_i \Rightarrow tpexpr_i}^{\,i}\, \texttt{end}) \Leftarrow y_2{:}\beta_2.\ \sigma(term_2).$

⟨1⟩17. Case: Ty_{Action*,Memop*}.
Proof: By induction. Rules with SMT checks still hold because the same substitution is applied to the constraint context and the consequent.

⟨1⟩18. Case: Ty_TVal_I
Proof: Trivially (no free variables nor requirements on constraint context).

⟨1⟩19. 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⟩20. Case: Ty_TVal_Phi
$\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash \overline{x_i = spine\_elem_i}^{\,i} :: term \supset arg \gg \psi; ret.$

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

⟨2⟩2. $\texttt{smt}\,(\Phi' \Rightarrow \sigma(term)).$
Proof: We $\texttt{smt}\,(\Phi' \Rightarrow term)$, which means it is true for all (well-typed) instantiations of its free variables.

⟨2⟩3. $\texttt{smt}\,(\Phi \Rightarrow \sigma(term)).$
Proof: From $\Phi \Rightarrow \Phi'$ and ⟨2⟩2.

⟨2⟩4. $\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \overline{x_i = \sigma(spine\_elem_i)}^{\,i} :: \sigma(term \supset arg) \gg \sigma(\psi);\sigma(ret)$ as required.

⟨1⟩21. Case: Ty_TVal_Res
Proof: Similar to Ty_TVal_Phi, except with resource environments being split.

⟨1⟩22. Case: Ty_TVal_Undef
Proof: *ret* can be anything, including $\sigma(ret)$.

⟨1⟩23. Case: Ty_Seq_TE_{TVal,If,Bound}.
Proof: By induction.

⟨1⟩24. 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. $\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash$ `ccall` $(\tau, ident, \overline{\sigma(spine\_elem_i)}^{\,i}) \Rightarrow \sigma, \psi(ret)$ as required.

⟨1⟩25. Case: Ty_Spine_Res.
$\mathcal{C}';\mathcal{L}';\Phi';\mathcal{R}'_1,\mathcal{R}_2 \vdash x = res\_term, \overline{x_i = spine\_elem_i}^{\,i} :: res \multimap arg \gg res\_term/x, \psi; ret$

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

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

2.3 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.

2.4 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 (2.2) to $\langle 1 \rangle 1$.

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

3 Progress

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

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

Proof sketch: Induction over *arg*.

3.2 Progress Statement and Proof

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

Proof sketch: Induction over the typing rules.

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

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

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

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

$\langle 1 \rangle 4$. Case: Ty_PE_Not.
Proof: *pval* must be a *bool_value* so Op_PE_PE_Not_{True,False}.

$\langle 1 \rangle 5$. Case: Ty_PE_{Arith,Rel}_Binop.

PROOF: $pval_1$ and $pval_2$ must be *mem_int*s so OP_PE_PE_{ARITH,REL}_BINOP respectively.

⟨1⟩6. CASE: TY_PE_BOOL_BINOP.
PROOF: $pval_1$ and $pval_2$ must be *bool_value*s so OP_PE_PE_BOOL_BINOP.

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

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

⟨1⟩9. CASE: TY_PE_BOOL_TO_INTEGER.
PROOF: *pval* must be a *bool_value* and so OP_PE_PE_BOOL_TO_INTEGER_{TRUE,FALSE}.

⟨1⟩10. CASE: TY_PE_WRAPI.
PROOF: *pval* must be a *mem_int* and so OP_PE_PE_WRAPI.

⟨1⟩11. CASE: TY_TPE_{IF,LET,LETT,CASE}.
PROOF: See TY_SEQ_TE_{IF,LET,LETT,CASE} cases for more general cases and proofs.

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

⟨1⟩13. CASE: TY_ACTION_LOAD.
PROOF: $pval_0$ must be a *mem_ptr* and $h = \cdot + \{pval_1 \overset{\checkmark}{\mapsto}_\tau pval_2\}$, so OP_ACTION_TVAL_LOAD.

⟨1⟩14. CASE: TY_ACTION_STORE.
PROOF: $pval_0$ and $pval_2$ must be the same *mem_ptr*, so OP_ACTION_TVAL_STORE.

⟨1⟩15. CASE: TY_ACTION_KILL_STATIC.
PROOF: $pval_0$ and $pval_1$ must be the same *mem_ptr*, so OP_ACTION_TVAL_KILL_STATIC.

⟨1⟩16. CASE: TY_MEMOP_REL_BINOP.
PROOF: Similar to TY_PE_{ARITH,REL}_BINOP.

⟨1⟩17. CASE: TY_MEMOP_INTFROMPTR.
PROOF: *pval* must be a *mem_ptr* so OP_MEMOP_TVAL_REL_INTFROMPTR.

⟨1⟩18. CASE: TY_MEMOP_PTRFROMINT.
PROOF: *pval* must be a *mem_int* so OP_MEMOP_TVAL_REL_PTRFROMINT.

⟨1⟩19. CASE: TY_MEMOP_PTRVALIDFORDEREF.

PROOF: *pval* must be a *mem_ptr* and h must be $\cdot + \{mem_ptr \overset{\checkmark}{\mapsto}_\tau _\}$so it can take a step with OP_MEMOP_TVAL_REL_PTRVALIDFORDEREF.

⟨1⟩20. CASE: TY_MEMOP_PTRWELLALIGNED.
PROOF: *pval* must be a *mem_ptr* and so OP_MEMOP_TVAL_PTRWELLALIGNED.

⟨1⟩21. CASE: TY_MEMOP_PTRARRAYSHIFT.
PROOF: $pval_1$ must be a *mem_ptr* and $pval_2$ must be a *mem_int* and so OP_MEMOP_TVAL_PTRARRAYSHIFT.

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

⟨1⟩23. CASE: TY_SEQ_E_PROC.
PROOF: Similar to TY_SEQ_E_CCALL.

⟨1⟩24. CASE: TY_IS_E_MEMOP.
PROOF: By induction, if *mem_op* is unreachable, then the whole expression is so. Memops are not values. Only stepping cases applies, so OP_ISE_ISE_MEMOP.

⟨1⟩25. CASE: TY_IS_E_{NEG_}ACTION.
PROOF: By induction, if *mem_action* is unreachable, then the whole expression is so. Actions are not values. Only stepping case applies, so OP_ISE_ISE_{NEG_}ACTION.

⟨1⟩26. CASE: TY_SEQ_TE_{LETP,LETPT}.
PROOF: See TY_SEQ_TE_{LET,LETT} for more general cases and proofs.

⟨1⟩27. CASE: TY_SEQ_TE_LET.
PROOF: By induction, since *seq_expr* is not value, if it is unreachable, the whole expression is so. If it takes a step, then OP_STE_TE_LET_LETT.

⟨1⟩28. CASE: TY_SEQ_TE_LETT.
PROOF: By induction, if *texpr* is unreachable, so is the whole expression. If if it a *tval* then OP_STE_TE_LETT_SUB. If if takes a step, then OP_STE_TE_LETT_LETT.

⟨1⟩29. CASE: TY_SEQ_TE_CASE.
PROOF: By assumption that all patterns are exhaustive, there is at least one pattern against which *pval* will match, so OP_STE_TE_CASE.

⟨1⟩30. CASE: TY_SEQ_TE_IF.
PROOF: *pval* must be a *bool_value* and so OP_STE_TE_IF_{TRUE,FALSE}.

⟨1⟩31. CASE: TY_SEQ_TE_RUN.
PROOF: Similar to TY_SEQ_E_CCALL.

⟨1⟩32. CASE: TY_SEQ_TE_BOUND.
PROOF: By OP_STE_TE_BOUND.

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

4 Framing

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

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

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

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

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

5 Type Preservation

5.1 Pointed-to values have type β_τ

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

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

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

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

Prove: The constraint $term_j = pval$ holds.

Proof sketch: Induction over *pattern*.

5.3 Deconstructing a pattern leads to a well-typed substitution

First, computational part.

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

Proof sketch: By induction over 2.

⟨1⟩1. Case: Ty_Pat_Sym_Or_Pattern_Sym and Ty_Pat_Comp_Sym_Annot.
$\sigma = pval/x, \cdot$ and $\mathcal{C} = \cdot, x{:}\beta$.
Proof: By Ty_Subs_Cons_Comp and 1.

⟨1⟩2. Case: Ty_Pat_No_Sym_Annot and Ty_Pat_Comp_Nil.
σ and $\mathcal{C}$ are empty.
Proof: By Ty_Subs_Empty, we are done.

⟨1⟩3. Case: Ty_Pat_Comp_{Specified,Cons,Tuple,Array}.
Proof: By induction (and concatenating well-typed substitutions).

Now, resource part.

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

Proof sketch: By induction over 2.
⟨1⟩1. Case: Ty_Pat_Res_Empty.
$res\_pattern = res\_term = res = \texttt{emp}$. $\sigma, \mathcal{L}, \Phi, \mathcal{R}, \mathcal{R}'$ are all empty.
Proof: By Ty_Subs_Empty, we are done.

⟨1⟩2. Case: Ty_Pat_Res_PointsTo.
$res\_pattern = res\_term = res = pt$. $\sigma = \cdot$, $\mathcal{L} = \cdot$, $\Phi = \cdot$, $\mathcal{R} = \mathcal{R}' = \cdot, pt$.
Proof: By Ty_Subs_Cons_Res_Anon.

⟨1⟩3. Case: Ty_Pat_Res_Var.
$res\_pattern = r$, $\sigma = res\_term/x, \cdot$, $\mathcal{L} = \cdot$, $\Phi = \cdot$, $\mathcal{R}' = \cdot, x{:}res$.
Proof: By Ty_Subs_Cons_Res_Named.

⟨1⟩4. Case: Ty_Pat_Res_SepConj.
Proof: By induction (and concatenating well-typed substitutions).

⟨1⟩5. Case: Ty_Pat_Res_Conj.
Proof: By $\texttt{smt}\,(\cdot \Rightarrow term)$ (from 1) and induction with Ty_Sub_Cons_Phi.

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

Now, full proof.

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

Proof sketch: Induction on 3.

⟨1⟩1. Case: Ty_Ret_Pat_Empty
Proof: By Ty_Subs_Empty.

⟨1⟩2. Case: Ty_Ret_Pat_{Comp,Res}
Proof: By induction, well-typed computational / resource substitutions and concatenating well-typed substitutions.

⟨1⟩3. Case: Ty_Ret_Path_Log.
Proof: By induction.

⟨1⟩4. Case: Ty_Ret_Pat_Phi
Proof: By induction and inversion on 2 to conclude $\texttt{smt}\,(\cdot \Rightarrow term)$
(required by Ty_Subs_Cons_Phi).

5.4 Type Preservation Statement and Proof

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

Proof sketch: Induction over the typing rules.

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

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

⟨1⟩1. Case: Ty_PE_Array_Shift.
Let: $term = mem\_ptr +_{\text{ptr}} (mem\_int \times \text{size\_of}(\tau))$.
Assume: 1. $\cdot;\cdot;\cdot \vdash \texttt{array\_shift}\,(mem\_ptr, \tau, mem\_int) \Rightarrow y{:}\texttt{loc}.\ y = term$.
2. $\langle \texttt{array\_shift}\,(mem\_ptr, \tau, mem\_int)\rangle \longrightarrow \langle mem\_ptr'\rangle$.
Prove: $\cdot;\cdot;\cdot \vdash mem\_ptr' \Rightarrow y{:}\texttt{loc}.\ y = term$.
Proof: By Ty_PVal_Obj_Int, Ty_PVal_Obj, Ty_PE_Val and construction of mem_ptr' (inversion on 2).

⟨1⟩2. Case: Ty_PE_Member_Shift.
Proof sketch: Similar to Ty_Array_Shift.

⟨1⟩3. Case: Ty_PE_Not.
Assume: 1. $\cdot;\cdot;\cdot \vdash \texttt{not}\,(bool\_value) \Rightarrow y{:}\texttt{bool}.\ y = \neg\, bool\_value$.
2. $\langle \texttt{not}\,(\texttt{True})\rangle \longrightarrow \langle \texttt{False}\rangle$ or $\langle \texttt{not}\,(\texttt{False})\rangle \longrightarrow \langle \texttt{True}\rangle$.
Prove: $\cdot;\cdot;\cdot \vdash bool\_value' \Rightarrow y{:}\texttt{bool}.\ y = \neg\, bool\_value$.
Proof: By Ty_PVal_{True,False}, Ty_PE_Val and 2.

⟨1⟩4. Case: Ty_PE_Arith_Binop.
Let: $term = mem\_int_1\ binop_{arith}\ mem\_int_2$.
Assume: 1. $\cdot;\cdot;\cdot \vdash mem\_int_1\ binop_{arith}\ mem\_int_2 \Rightarrow y{:}\texttt{integer}.\ y = term$.
2. $\langle mem\_int_1\ binop_{arith}\ mem\_int_2\rangle \longrightarrow \langle mem\_int\rangle$.
Prove: $\cdot;\cdot;\cdot \vdash mem\_int \Rightarrow y{:}\texttt{integer}.\ y = term$.
Proof: By Ty_PVal_Obj_Int, Ty_PVal_Obj, Ty_PE_Val and construction of mem_int (inversion on 2).

⟨1⟩5. Case: Ty_PE_{Rel,Bool}_Binop.
Proof sketch: Similar to Ty_PE_Arith_Binop.

⟨1⟩6. Case: Ty_PE_Call.
Proof: See Ty_Seq_E_Call for a more general case and proof.

⟨1⟩7. Case: Ty_PE_Assert_Undef.
Assume: 1. $\cdot;\cdot;\cdot \vdash \texttt{assert\_undef}\,(\texttt{True},\ \mathit{UB\_name}) \Rightarrow y{:}\texttt{unit}.\ y = \texttt{unit}$.
2. $\langle \texttt{assert\_undef}\,(\texttt{True},\ \mathit{UB\_name})\rangle \longrightarrow \langle \texttt{Unit}\rangle$.
Prove: $\cdot;\cdot;\cdot \vdash \texttt{Unit} \Rightarrow y{:}\texttt{unit}.\ y = \texttt{unit}$.
Proof: By Ty_PVal_Unit and Ty_PE_Val.

⟨1⟩8. Case: Ty_PE_Bool_To_Integer.
Let: $\mathit{term} = \texttt{if}\ \mathit{bool\_value}\ \texttt{then}\ 1\ \texttt{else}\ 0$.
Assume: 1. $\cdot;\cdot;\cdot \vdash \texttt{bool\_to\_integer}\,(\mathit{bool\_value}) \Rightarrow y{:}\texttt{integer}.\ y = \mathit{term}$.
2. $\langle \texttt{bool\_to\_integer}\,(\texttt{True})\rangle \longrightarrow \langle 1\rangle$ or $\langle \texttt{bool\_to\_integer}\,(\texttt{False})\rangle \longrightarrow \langle 0\rangle$.
Prove: $\cdot;\cdot;\cdot \vdash \mathit{mem\_int} \Rightarrow y{:}\texttt{integer}.\ y = \mathit{term}$
Proof: By cases on *bool_value*, then applying Ty_PVal_{True,False} and Ty_PE_Val.

⟨1⟩9. Case: Ty_PE_WrapI.
Proof sketch: Similar to Ty_PE_Bool_To_Integer, except by cases on $\mathit{abbrev}_2 \leq \mathrm{max\_int}_\tau$, then applying Ty_PVal_Obj_Int, Ty_PVal_Obj and Ty_PE_Val.

⟨1⟩10. Case: Ty_TPE_If.
Proof: See Ty_Seq_TE_If for a more general case and proof.

⟨1⟩11. Case: Ty_TPE_Let.
Proof: See Ty_Seq_TE_Let for a more general case and proof.

⟨1⟩12. Case: Ty_TPE_LetT.
Proof: See Ty_Seq_TE_LetT for a more general case and proof.

⟨1⟩13. Case: Ty_TPE_Case.
Proof: See Ty_Seq_TE_Case for a more general case and proof.

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

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

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

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

⟨2⟩4. $\cdot;\cdot;\cdot;\cdot, \mathit{pt} \vdash \mathit{pt} \Leftarrow \mathit{pt}$ 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: $pt = mem\_ptr \overset{\checkmark}{\mapsto}_\tau pval$.
$ret = \Sigma\, y{:}\beta_\tau.\ y = pval \wedge pt \otimes \mathtt{I}$.
Assume: 1. $\cdot;\cdot;\cdot;\cdot, pt \vdash \mathtt{load}\,(\tau, mem\_ptr, \_, pt) \Rightarrow ret$.
2. $\langle \cdot + \{pt\}; \mathtt{load}\,(\tau, mem\_ptr, \_, pt)\rangle \longrightarrow \langle \cdot + \{pt\}; \mathtt{done}\, pval, pt\rangle$.
Prove: $\cdot;\cdot;\cdot;\cdot, pt \vdash \mathtt{done}\, pval, pt \Leftarrow ret$

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

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

⟨2⟩3. $\cdot;\cdot;\cdot \vdash pval \Rightarrow \beta_\tau$ by ⟨2⟩1 and 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: $\cdot;\cdot;\cdot;\cdot \vdash \mathtt{done}\,\mathtt{Unit} \Leftarrow \Sigma\, \_{:}\mathtt{unit}.\ \mathtt{I}$
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: $pt = mem\_ptr \overset{\checkmark}{\mapsto}_{\tau} \_$.
$ret = \Sigma\, y{:}\texttt{bool}.\ y = \texttt{aligned}\,(\tau, mem\_ptr) \wedge pt \otimes \texttt{I}$.
Assume: 1. $\cdot;\cdot;\cdot;\mathcal{R} \vdash \texttt{ptrValidForDeref}\,(\tau, mem\_ptr, pt) \Rightarrow ret$.
2. $\langle \cdot + \{pt\}; \texttt{ptrValidForDeref}\,(\tau, mem\_ptr, pt)\rangle \longrightarrow \langle \cdot + \{pt\}; \texttt{done}\, bool\_value, pt\rangle$.
Prove: $\cdot;\cdot;\cdot;\mathcal{R} \vdash \texttt{done}\, bool\_value, pt \Leftarrow ret$.

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

⟨2⟩2. $R = \cdot, pt$, by Ty_Res_PointsTo.

⟨2⟩3. $bool\_value = \texttt{aligned}\,(\tau, mem\_ptr)$ by construction of *bool_value* (inversion on 2).

⟨2⟩4. $\cdot;\cdot;\cdot \vdash bool\_value \Rightarrow \texttt{bool}$ by Ty_PVal_{True,False}.

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

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

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

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

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

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

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

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

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

⟨2⟩3. $\sigma = \sigma'$ and $ret = ret'$ by induction on *arg*.
Proof: Follows from lemma 3.1.

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

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

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

⟨2⟩7. Apply substitution lemma (2.2) to ⟨2⟩4 and ⟨2⟩5 to finish proof.

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

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

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

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

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

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

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

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

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

⟨1⟩30. CASE: TY_SEQ_TE_LET.
ASSUME: 1. $\cdot;\cdot;\cdot;\mathcal{R}', \mathcal{R} \vdash \texttt{let}\, \overline{ret\_pattern_i}^{\,i} = seq\_expr\, \texttt{in}\, texpr_2 \Leftarrow ret_2$.
2. $\langle h; \texttt{let}\, \overline{ret\_pattern_i}^{\,i} = seq\_expr\, \texttt{in}\, texpr_2\rangle \longrightarrow \langle h; \texttt{let}\, \overline{ret\_pattern_i}^{\,i} {:} ret_1' = texpr_1\, \texttt{in}\, texpr_2\rangle$.
PROVE: $\cdot;\cdot;\cdot;\mathcal{R}', \mathcal{R} \vdash \texttt{let}\, \overline{ret\_pattern_i}^{\,i} {:} ret_1 = texpr_1\, \texttt{in}\, texpr_2 \Leftarrow ret_2$.

⟨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}.

⟨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.
NOTE: $h : \mathcal{R}', \mathcal{R}$ and $h : \mathcal{R}_1, \mathcal{R}$.

ASSUME: 1. $\cdot;\cdot;\cdot;\mathcal{R}', \mathcal{R} \vdash \mathtt{let}\ \overline{\mathit{ret\_pattern}_i}^{\,i} : \mathit{ret}_1 = \mathtt{done}\ \overline{\mathit{spine\_elem}_i}^{\,i}\ \mathtt{in}\ \mathit{texpr}_2 \Leftarrow \mathit{ret}_2$.
2. $\langle h; \mathtt{let}\ \overline{\mathit{ret\_pattern}_i}^{\,i} : \mathit{ret}_1 = \mathtt{done}\ \overline{\mathit{spine\_elem}_i}^{\,i}\ \mathtt{in}\ \mathit{texpr}\rangle \longrightarrow \langle h; \sigma(\mathit{texpr}_2)\rangle$.
PROVE: $\cdot;\cdot;\cdot;\mathcal{R}', \mathcal{R} \vdash \sigma(\mathit{texpr}_2) \Leftarrow \sigma(\mathit{ret}_2)$.

⟨2⟩1. 1. $\cdot;\cdot;\cdot;\mathcal{R}' \vdash \mathtt{done}\ \overline{\mathit{spine\_elem}_i}^{\,i} \Leftarrow \mathit{ret}_1$.
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.

⟨2⟩2. $\overline{\mathit{ret\_pattern}_i = \mathit{spine\_elem}_i}^{\,i} \rightsquigarrow \sigma$.
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⟩3, ⟨2⟩2 using lemma 5.3 (deconstructing a pattern produces a well-typed substitution).

⟨2⟩4. By ⟨2⟩1.3 and ⟨2⟩3 and the let-friendly substitution lemma 2.4, we are done.

⟨1⟩32. CASE: TY_SEQ_TE_LETT.
ASSUME: 1. $\cdot;\cdot;\cdot;\mathcal{R}', \mathcal{R} \vdash \mathtt{let}\ \overline{\mathit{ret\_pattern}_i}^{\,i} : \mathit{ret}_1 = \mathit{texpr}_1\ \mathtt{in}\ \mathit{texpr}_2 \Leftarrow \mathit{ret}_2$.
2. $\langle h; \mathtt{let}\ \overline{\mathit{ret\_pattern}_i}^{\,i} : \mathit{ret} = \mathit{texpr}_1\ \mathtt{in}\ \mathit{texpr}_2\rangle \longrightarrow \langle h'; \mathtt{let}\ \overline{\mathit{ret\_pattern}_i}^{\,i} : \mathit{ret} = \mathit{texpr}_1'\ \mathtt{in}\ \mathit{texpr}_2\rangle$.
PROVE: $\cdot;\cdot;\cdot;\mathcal{R}'', \mathcal{R} \vdash \mathtt{let}\ \overline{\mathit{ret\_pattern}_i}^{\,i} : \mathit{ret}_1 = \mathit{texpr}_1'\ \mathtt{in}\ \mathit{texpr}_2 \Leftarrow \mathit{ret}_2$.

⟨2⟩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.

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

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

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

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

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

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

⟨2⟩3. $term_j = pval$.
Proof: By ⟨1⟩32.2 and lemma 5.2.

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

⟨2⟩5. By ⟨2⟩4, ⟨1⟩32.3 and 2.2, we are done.

⟨1⟩34. Case: Ty_Seq_TE_If.
Only covering `True` case, `False` is almost identical.
Assume: 1. $\cdot;\cdot;\cdot;\mathcal{R} \vdash \texttt{if True then}\, texpr_1\, \texttt{else}\, texpr_2 \Leftarrow ret$.
2. $\langle h; \texttt{if True then}\, texpr_1\, \texttt{else}\, texpr_2\rangle \longrightarrow \langle h; texpr_1\rangle$.
Prove: $\cdot;\cdot;\cdot;\mathcal{R} \vdash texpr_1 \Leftarrow ret$.
Proof: Invert 1, note $\cdot;\cdot;\cdot;\mathcal{R} \vdash (\mathrm{id}){:}(\cdot;\cdot;\cdot, \texttt{true} = \texttt{true}; \mathcal{R})$ and then apply substitution lemma (2.2).

⟨1⟩35. Case: Ty_Seq_TE_Run.
Proof sketch: Similar to case Ty_Seq_E_{CCall,PCall}.

⟨1⟩36. Case: Ty_Seq_TE_Bound.
Proof: By inversion on the typing rule.

⟨1⟩37. Case: Ty_Is_TE_LetS.
Proof sketch: Similar to Ty_Seq_TE_LetT.

6 Typing Judgements

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

7 Opsem Judgements

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

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