

Explicit CN Soundness Proof

Dhruv Makwana

July 7, 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. $\mathcal{C}; \mathcal{L}; \mathcal{R} \vdash (\sigma) : (\mathcal{C}''; \mathcal{L}''; \mathcal{R}'')$.

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

2.2 Substitution Lemma

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

PROOF STRATEGY: Induction over the typing judgements.

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

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

(1)1. CASE: TY_PVAL_OBJ^* , $\text{TY_PVAL_}\{\text{OBJ, LOADED, UNIT, TRUE, FALSE, CTOR_NIL}\}$.
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: $\text{TY_PVAL_}\{\text{LIST}, \text{TUPLE}, \text{CTOR_CONS}, \text{CTOR_TUPLE}, \text{CTOR_ARRAY}, \text{CTOR_SPECIFIED}\}$.
 PROOF: By induction and then definition of substitution over values.

⟨1⟩3. CASE: TY_PVAL_ERROR .
 SUFFICES: $\text{smt}(\Phi \Rightarrow \text{false})$.
 PROOF: TODO

⟨1⟩4. CASE: TY_PVAL_STRUCT .

⟨1⟩5. CASE: TY_PVAL_VAR .
 $\mathcal{C}'; \mathcal{L}'; \Phi' \vdash x \Rightarrow \beta$

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

⟨2⟩2. So $\exists pval. \mathcal{C}; \mathcal{L}; \Phi \vdash pval \Rightarrow \beta$ by $\text{TY_SUBS_CONS_}\{\text{COMP}, \text{LOG}\}$.

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

⟨1⟩6. CASE: TY_TPE_LET .
 $\mathcal{C}'; \mathcal{L}'; \Phi' \vdash \text{let ident_or_pattern} = pexpr \text{ in } tpeexpr \Leftarrow y_2:\beta_2. term_2$.

⟨2⟩1. By induction,
 1. $\mathcal{C}; \mathcal{L}; \sigma(\Phi') \vdash \sigma(pexpr) \Rightarrow y_1:\beta. \sigma(term_1)$.
 2. $\mathcal{C}, \mathcal{C}_1; \mathcal{L}; \sigma(\Phi'), term/y_1, \sigma(term_1) \vdash \sigma(tpeexpr) \Leftarrow y_2:\beta_2. \sigma(term_2)$.

⟨2⟩2. $\mathcal{C}; \mathcal{L}; \sigma(\Phi') \vdash \sigma(\text{let ident_or_pattern} = pexpr \text{ in } tpeexpr) \Leftarrow y_2:\beta_2. \sigma(term_2)$ as required.

⟨1⟩7. CASE: TY_TVAL_LOG .
 $\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash \text{done } pval, \overline{spine_elem_i}^i \Leftarrow \exists y:\beta. ret$.

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

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

⟨1⟩8. 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 $\mathcal{C}; \mathcal{L}; \mathcal{R} \vdash (\sigma):(\mathcal{C}'; \mathcal{L}'; \mathcal{R}')$ then $\mathcal{C}; \mathcal{L}; \mathcal{R}_1, \mathcal{R} \vdash (\sigma, \text{id}):(\mathcal{C}, \mathcal{C}'; \mathcal{L}, \mathcal{L}'; \mathcal{R}_1, \mathcal{R}')$.

PROOF SKETCH: Induction over the substitution.

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

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

$\langle 1 \rangle 1.$ $\mathcal{C}; \mathcal{L}; \mathcal{R}_1 \vdash (\text{id}):(\mathcal{C}; \mathcal{L}; \mathcal{R}_1)$.

PROOF: By induction on each of $\mathcal{C}; \mathcal{L}; \mathcal{R}_1$.

$\langle 1 \rangle 2.$ $\mathcal{C}; \mathcal{L}; \mathcal{R}_1, \mathcal{R} \vdash (\sigma, \text{id}):(\mathcal{C}, \mathcal{C}'; \mathcal{L}, \mathcal{L}'; \mathcal{R}_1, \mathcal{R}')$

PROOF: By induction on σ with base case as above.

2.4 Let-friendly Substitution Lemma

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

2. $\mathcal{C}, \mathcal{C}'; \mathcal{L}, \mathcal{L}'; \Phi; \mathcal{R}_1, \mathcal{R}' \vdash J$.

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

$\langle 1 \rangle 1.$ $\mathcal{C}; \mathcal{L}; \mathcal{R} \vdash (\sigma, \text{id}):(\mathcal{C}, \mathcal{C}'; \mathcal{L}, \mathcal{L}'; \mathcal{R}_1, \mathcal{R}')$.

PROOF: Apply identity extension to 1.

$\langle 1 \rangle 2.$ $\mathcal{C}; \mathcal{L}; \sigma(\Phi); \mathcal{R}_1, \mathcal{R} \vdash (\sigma, \text{id})(J)$.

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: $\text{id}(J) = J$.

3 Progress

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

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

PROOF SKETCH: Induction over arg .

3.2 Progress Statement and Proof

If $;; \mathcal{R} \vdash e \Leftrightarrow t$ 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 STRATEGY: Induction over the typing rules.

ASSUME: 1. $;; \mathcal{R} \vdash e \Leftrightarrow t$.

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).

<1>2. CASE: TY_PE_ARRAY_SHIFT.

PROOF: By inversion on $\cdot; \cdot; \cdot \vdash pval_1 \Rightarrow \text{loc}$, $pval_1$ 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_ints* so OP_PE_PE_{ARITH,REL}_BINOP respectively.

<1>6. CASE: TY_PE_BOOL_BINOP.

PROOF: $pval_1$ and $pval_2$ must be *bool_values* so OP_PE_PE_BOOL_BINOP.

<1>7. CASE: TY_PE_CALL.

PROOF: By inversion we have $\text{name:pure_arg} \equiv \overline{x_i}^i \mapsto tpe\text{expr} \in \mathbf{Globals}$ and $\cdot; \cdot; \cdot \vdash \overline{x_i = pval_i}^i :: \text{pure_arg} \gg \sigma; \Sigma y:\beta. \text{term} \wedge \mathbf{I}$, with the latter implying $\overline{x_i = pval_i}^i :: \text{pure_arg} \gg \sigma; \Sigma y:\beta. \text{term} \wedge \mathbf{I}$ (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 $\text{smt}(\Phi \Rightarrow pval)$. 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 $h = \cdot + \{pval_1 \mapsto_\tau pval_2\}$, 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 $\cdot + \{mem_ptr \xrightarrow{\tau} \cdot\}$ 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 $pval$ must be a mem_ptr , and $mem_ptr:arg \equiv \overline{x_i}^i \mapsto texpr \in \mathbf{Globals}$ and $\cdot; \cdot; \cdot \vdash \overline{x_i = spine_elem_i}^i :: arg \gg \sigma; ret$, with the latter implying $\overline{x_i = spine_elem_i}^i :: arg \gg \sigma; ret$ (lemma 3.1). Thus it can step with `OP_SEQ_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 it 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 $pt = _ \mapsto_\tau pval$, 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 Deconstructing a pattern leads to a well-typed substitution

First, computational part.

ASSUME: 1. $\cdot; \cdot; \cdot \vdash pval \Rightarrow \beta_1$.

2. $ident_or_pattern:\beta \rightsquigarrow \mathcal{C} \text{ with term.}$

3. $\text{ident_or_pattern} = \text{pval} \rightsquigarrow \sigma$.

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

PROOF SKETCH: By induction over 2.

$\langle 1 \rangle 1$. CASE: $\text{TY_PAT_SYM_OR_PATTERN_SYM}$ and $\text{TY_PAT_COMP_SYM_ANNOT}$.

$\sigma = \text{pval}/x, \cdot$ and $\mathcal{C} = \cdot, x:\beta$.

PROOF: By TY_SUBS_CONS_COMP and 1.

$\langle 1 \rangle 2$. CASE: $\text{TY_PAT_NO_SYM_ANNOT}$ and TY_PAT_COMP_NIL .

σ and \mathcal{C} are empty.

PROOF: By TY_SUBS_EMPTY , we are done.

$\langle 1 \rangle 3$. CASE: $\text{TY_PAT_COMP}\{\text{SPECIFIED}, \text{CONS}, \text{TUPLE}, \text{ARRAY}\}$.

PROOF: By induction (and concatenating well-typed substitutions).

Now, resource part.

ASSUME: 1. $\cdot; \cdot; \cdot; \mathcal{R} \vdash \text{res_term} \Leftarrow \text{res}$.

2. $\text{res_pattern}:\text{res} \rightsquigarrow \mathcal{L}; \Phi; \mathcal{R}'$.

3. $\text{res_pattern} = \text{res_term} \rightsquigarrow \sigma$.

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

PROOF SKETCH: By induction over 2.

$\langle 1 \rangle 1$. CASE: TY_PAT_RES_EMPTY .

$\text{res_pattern} = \text{res_term} = \text{res} = \text{emp}$. $\sigma, \mathcal{L}, \Phi, \mathcal{R}, \mathcal{R}'$ are all empty.

PROOF: By TY_SUBS_EMPTY , we are done.

$\langle 1 \rangle 2$. CASE: $\text{TY_PAT_RES_POINTSTO}$.

$\text{res_pattern} = \text{res_term} = \text{res} = \text{pt}$. $\sigma = \cdot, \mathcal{L} = \cdot, \Phi = \cdot, \mathcal{R} = \mathcal{R}' = \cdot, \text{pt}$.

PROOF: By $\text{TY_SUBS_CONS_RES_ANON}$.

$\langle 1 \rangle 3$. CASE: TY_PAT_RES_VAR .

$\text{res_pattern} = r, \sigma = \text{res_term}/x, \cdot, \mathcal{L} = \cdot, \Phi = \cdot, \mathcal{R}' = \cdot, x:\text{res}$.

PROOF: By $\text{TY_SUBS_CONS_RES_NAMED}$.

$\langle 1 \rangle 4$. CASE: $\text{TY_PAT_RES_SEPCONJ}$.

PROOF: By induction (and concatenating well-typed substitutions).

$\langle 1 \rangle 5$. CASE: TY_PAT_RES_CONJ .

PROOF: By induction.

$\langle 1 \rangle 6$. CASE: TY_PAT_RES_PACK .

$\text{res_pattern} = \text{pack}(x, \text{res_pattern}')$, $\text{res_term} = \text{pack}(\text{pval}, \text{res_term}')$, $\text{res} = \exists x:\beta. \text{res}'$.

$\sigma = \text{pval}/x, \sigma', \mathcal{L} = \mathcal{L}', x:\beta, \mathcal{R} = \mathcal{R}'$.

PROOF: By induction and TY_SUBS_CONS_LOG .

Now, full proof.

ASSUME: 1. $\overline{\text{ret_pattern}_i = \text{spine_elem}_i^i} \rightsquigarrow \sigma$.

2. $\cdot; \cdot; \cdot; \mathcal{R} \vdash \text{done spine_elem}_i^i \Leftarrow \text{ret}$.

3. $\overline{ret_pattern_i}^i : ret \rightsquigarrow \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}'$.

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

PROOF SKETCH: Induction on 3. Base case by `TY_SUBS_EMPTY`. `TY_RET_PAT_{COMP, RES}` by induction, well-typed computational / resource substitutions and concatenating well-typed substitutions. `TY_RET_PAT_{LOG, PHI}` by induction and `TY_SUBS_CONS_LOG`.

5.3 Type Preservation Statement and Proof

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

PROOF SKETCH: Induction over the typing rules.

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

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

(1)1. CASE: `TY_PE_ARRAY_SHIFT`.

LET: $term = mem_ptr +_{ptr} (mem_int \times size_of(\tau))$.

ASSUME: 1. $\cdot; \cdot; \cdot \vdash array_shift(mem_ptr, \tau, mem_int) \Rightarrow y : loc. y = term$.

2. $\langle array_shift(mem_ptr, \tau, mem_int) \rangle \longrightarrow \langle mem_ptr' \rangle$.

PROVE: $\cdot; \cdot; \cdot \vdash mem_ptr' \Rightarrow y : loc. y = term$.

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. $\cdot; \cdot; \cdot \vdash not(bool_value) \Rightarrow y : bool. y = \neg bool_value$.

2. $\langle not(True) \rangle \longrightarrow \langle False \rangle$ or $\langle not(False) \rangle \longrightarrow \langle True \rangle$.

PROVE: $\cdot; \cdot; \cdot \vdash bool_value' \Rightarrow y : bool. y = \neg bool_value$.

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. $\cdot; \cdot; \cdot \vdash mem_int_1 binop_{arith} mem_int_2 \Rightarrow y : integer. y = term$.

2. $\langle mem_int_1 binop_{arith} mem_int_2 \rangle \longrightarrow \langle mem_int \rangle$.

PROVE: $\cdot; \cdot; \cdot \vdash mem_int \Rightarrow y : integer. y = term$.

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. $\cdot; \cdot; \cdot \vdash \text{assert_undef}(\text{True}, UB_name) \Rightarrow y:\text{unit}. y = \text{unit}.$

2. $\langle \text{assert_undef}(\text{True}, UB_name) \rangle \longrightarrow \langle \text{Unit} \rangle.$

PROVE: $\cdot; \cdot; \cdot \vdash \text{Unit} \Rightarrow y:\text{unit}. y = \text{unit}.$

PROOF: By TY_PVAL_UNIT and TY_PE_VAL.

$\langle 1 \rangle 8.$ CASE: TY_PE_BOOL_TO_INTEGER.

LET: $term = \text{if } bool_value \text{ then } 1 \text{ else } 0.$

ASSUME: 1. $\cdot; \cdot; \cdot \vdash \text{bool_to_integer}(bool_value) \Rightarrow y:\text{integer}. y = term.$

2. $\langle \text{bool_to_integer}(\text{True}) \rangle \longrightarrow \langle 1 \rangle$ or $\langle \text{bool_to_integer}(\text{False}) \rangle \longrightarrow \langle 0 \rangle.$

PROVE: $\cdot; \cdot; \cdot \vdash mem_int \Rightarrow y:\text{integer}. y = term$

PROOF: By cases on $bool_value$, then applying TY_PVAL_{TRUE,FALSE} and TY_PE_VAL.

$\langle 1 \rangle 9.$ CASE: TY_PE_WRAPI.

PROOF SKETCH: Similar to TY_PE_BOOL_TO_INTEGER, except by cases on $abbrev_2 \leq \max_int_\tau$, then applying TY_PVAL_OBJ_INT, TY_PVAL_OBJ and TY_PE_VAL.

$\langle 1 \rangle 10.$ CASE: TY_TPE_IF.

PROOF: See TY_SEQ_TE_IF for a more general case and proof.

$\langle 1 \rangle 11.$ CASE: TY_TPE_LET.

PROOF: See TY_SEQ_TE_LET for a more general case and proof.

$\langle 1 \rangle 12.$ CASE: TY_TPE_LETT.

PROOF: See TY_SEQ_TE_LETT for a more general case and proof.

$\langle 1 \rangle 13.$ CASE: TY_TPE_CASE.

PROOF: See TY_SEQ_TE_CASE for a more general case and proof.

$\langle 1 \rangle 14.$ CASE: TY_ACTION_CREATE.

LET: $pt = mem_ptr \overset{\times}{\mapsto}_\tau pval.$

$term = \text{representable}(\tau*, y_p) \wedge \text{alignedI}(mem_int, y_p).$

$ret = \Sigma y_p:\text{loc}. term \wedge \exists y:\beta_\tau. y_p \overset{\times}{\mapsto}_\tau y \otimes \mathbf{I}.$

ASSUME: 1. $\cdot; \cdot; \cdot; \cdot \vdash \text{create}(mem_int, \tau) \Rightarrow ret.$

2. $\langle \cdot; \text{create}(mem_int, \tau) \rangle \longrightarrow \langle \cdot + \{pt\}; \text{done } mem_ptr, pval, pt \rangle.$

PROVE: $\cdot; \cdot; \cdot; \cdot, pt \vdash \text{done } mem_ptr, pval, pt \Leftarrow ret.$

$\langle 2 \rangle 1.$ $\cdot; \cdot; \cdot \vdash mem_ptr \Rightarrow \text{loc}$ by TY_PVAL_OBJ_INT and TY_PVAL_OBJ.

$\langle 2 \rangle 2.$ $\text{smt}(\cdot \Rightarrow term)$ by construction of mem_ptr .

$\langle 2 \rangle 3.$ $\cdot; \cdot; \cdot \vdash pval \Rightarrow \beta_\tau$ by construction of $pval$.

$\langle 2 \rangle 4.$ $\cdot; \cdot; \cdot; \cdot, pt \vdash pt \Leftarrow pt$ by TY_RES_POINTS_TO.

$\langle 2 \rangle 5.$ By TY_TVAL_I and then $\langle 2 \rangle 4 - \langle 2 \rangle 1$ with TY_TVAL_{RES,LOG,PHI,COMP} respectively, we are done.

$\langle 1 \rangle 15.$ CASE: TY_ACTION_LOAD.

LET: $pt = mem_ptr \overset{\checkmark}{\mapsto}_\tau pval.$

$ret = \Sigma y:\beta_\tau. y = pval \wedge pt \otimes \mathbf{I}.$

ASSUME: 1. $\cdot; \cdot; \cdot; \cdot, pt \vdash \text{load}(\tau, mem_ptr, -, pt) \Rightarrow ret.$

2. $\langle \cdot + \{pt\}; \text{load}(\tau, \text{mem_ptr}, _, pt) \rangle \longrightarrow \langle \cdot + \{pt\}; \text{done } pval, pt \rangle$.
- PROVE: $\cdot; \cdot; \cdot, pt \vdash \text{done } pval, pt \Leftarrow \text{ret}$
- $\langle 2 \rangle 1$. $\cdot; \cdot; \cdot, pt \vdash pt \Leftarrow pt$, by inversion on 1.
- $\langle 2 \rangle 2$. $\text{smt}(\cdot \Rightarrow pval = pval)$ trivially.
- $\langle 2 \rangle 3$. $\cdot; \cdot; \cdot \vdash pval \Rightarrow \beta_\tau$ by $\langle 2 \rangle 1$ and lemma 5.1.
- $\langle 2 \rangle 4$. By TY_TVAL_I and then $\langle 2 \rangle 1 - \langle 2 \rangle 3$ with $\text{TY_TVAL_}\{\text{RES}, \text{PHI}, \text{COMP}\}$ respectively, we are done.
- $\langle 1 \rangle 16$. CASE: TY_ACTION_STORE .
- LET: $pt = \text{mem_ptr} \xrightarrow{\check{\tau}} _$.
- $pt' = \text{mem_ptr} \xrightarrow{\check{\tau}} pval$.
- $\text{ret} = \Sigma _:\text{unit}. pt' \otimes \text{I}$.
- ASSUME: 1. $\cdot; \cdot; \cdot, pt \vdash \text{store}(_, \tau, pval_0, pval_1, _, pt) \Rightarrow \text{ret}$.
2. $\langle \cdot + \{pt\}; \text{store}(_, \tau, \text{mem_ptr}, pval, _, pt) \rangle \longrightarrow \langle \cdot + \{pt'\}; \text{done Unit}, pt' \rangle$.
- PROVE: $\cdot; \cdot; \cdot, pt' \vdash \text{done Unit}, pt' \Leftarrow \text{ret}$.
- $\langle 2 \rangle 1$. $\cdot; \cdot; \cdot \vdash \text{Unit} \Rightarrow \text{unit}$ by TY_PVAL_UNIT .
- $\langle 2 \rangle 2$. $\cdot; \cdot; \cdot, pt' \vdash pt' \Leftarrow pt'$ by TY_RES_POINTSTO .
- $\langle 2 \rangle 3$. By TY_TVAL_I and $\langle 2 \rangle 1$ and $\langle 2 \rangle 2$ with $\text{TY_TVAL_}\{\text{RES}, \text{COMP}\}$ respectively, we are done.
- $\langle 1 \rangle 17$. CASE: $\text{TY_ACTION_KILL_STATIC}$.
- LET: $pt = \text{mem_ptr} \mapsto_\tau _$.
- ASSUME: 1. $\cdot; \cdot; \cdot, pt \vdash \text{kill}(\text{static } \tau, pval_0, pt) \Rightarrow \Sigma _:\text{unit}. \text{I}$.
2. $\langle \cdot + \{pt\}; \text{kill}(\text{static } \tau, \text{mem_ptr}, pt) \rangle \longrightarrow \langle h; \text{done Unit} \rangle$.
- PROVE: $\cdot; \cdot; \cdot \vdash \text{done Unit} \Leftarrow \Sigma _:\text{unit}. \text{I}$
- PROOF: By TY_TVAL_I , TY_PVAL_UNIT and then TY_TVAL_COMP .
- $\langle 1 \rangle 18$. CASE: $\text{TY_MEMOP_REL_BINOP}$.
- PROOF: Similar TY_PE_REL_BINOP , except with $\text{TY_TVAL_}\{\text{I}, \text{PHI}, \text{COMP}\}$ at the end.
- $\langle 1 \rangle 19$. CASE: $\text{TY_MEMOP_INTFROMPTR}$.
- LET: $\text{ret} = \Sigma y:\text{integer}. y = \text{cast_ptr_to_int } \text{mem_ptr} \wedge \text{I}$.
- ASSUME: 1. $\cdot; \cdot; \cdot \vdash \text{intFromPtr}(\tau_1, \tau_2, \text{mem_ptr}) \Rightarrow \text{ret}$.
2. $\langle \cdot; \text{intFromPtr}(\tau_1, \tau_2, \text{mem_ptr}) \rangle \longrightarrow \langle \cdot; \text{done mem_int} \rangle$.
- PROVE: $\cdot; \cdot; \cdot \vdash \text{done mem_int} \Leftarrow \text{ret}$
- $\langle 2 \rangle 1$. $\text{smt}(\cdot \Rightarrow \text{mem_int} = \text{cast_ptr_to_int } \text{mem_ptr})$ by construction of mem_int (inversion on 2).
- $\langle 2 \rangle 2$. $\cdot; \cdot; \cdot \vdash \text{mem_int} \Rightarrow \text{integer}$ by TY_PVAL_OBJ_INT and TY_PVAL_OBJ .
- $\langle 2 \rangle 3$. By TY_TVAL_I and $\langle 2 \rangle 1$ and $\langle 2 \rangle 2$ with $\text{TY_TVAL_}\{\text{PHI}, \text{COMP}\}$ respectively, we are done.
- $\langle 1 \rangle 20$. CASE: $\text{TY_MEMOP_PTRFROMINT}$.
- PROOF: Similar to $\text{TY_MEMOP_INTFROMPTR}$, swapping base types integer and loc .

⟨1⟩21. CASE: TY_MEMOP_PTRVALIDFORDEREF.

LET: $pt = mem_ptr \xrightarrow{\check{\tau}} _$.

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

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

2. $\langle \cdot + \{pt\}; \text{ptrValidForDeref}(\tau, mem_ptr, pt) \rangle \longrightarrow \langle \cdot + \{pt\}; \text{done } bool_value, pt \rangle.$

PROVE: $\cdot; \cdot; \cdot; \mathcal{R} \vdash \text{done } bool_value, pt \Leftarrow ret.$

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

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

⟨2⟩3. $bool_value = \text{aligned}(\tau, mem_ptr)$ by construction of $bool_value$ (inversion on 2).

⟨2⟩4. $\cdot; \cdot; \cdot \vdash bool_value \Rightarrow \text{bool}$ 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: $ret = \Sigma y:\text{bool}. y = \text{aligned}(\tau, mem_ptr) \wedge \text{I}.$

ASSUME: 1. $\cdot; \cdot; \cdot; \cdot \vdash \text{ptrWellAligned}(\tau, mem_ptr) \Rightarrow ret.$

2. $\langle \cdot; \text{ptrWellAligned}(\tau, mem_ptr) \rangle \longrightarrow \langle \cdot; \text{done } bool_value \rangle.$

PROVE: $\cdot; \cdot; \cdot; \cdot \vdash \text{done } bool_value \Rightarrow ret.$

⟨2⟩1. $\text{smt}(\cdot \Rightarrow bool_value = \text{aligned}(\tau, mem_ptr))$ by construction of $bool_value$ (inversion on 2).

⟨2⟩2. $\cdot; \cdot; \cdot \vdash bool_value \Rightarrow \text{bool}$ 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_CALL.

ASSUME: 1. $\cdot; \cdot; \cdot; \mathcal{R} \vdash \text{ccall}(\tau, pval, \overline{spine_elem_i}^i) \Rightarrow \sigma(ret).$

2. $\langle h; \text{ccall}(\tau, pval, \overline{spine_elem_i}^i) \rangle \longrightarrow \langle h; \sigma'(texpr); \sigma'(ret) \rangle.$

PROVE: $\cdot; \cdot; \cdot; \mathcal{R} \vdash \sigma(texpr) \Leftarrow \sigma(ret)$

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

⟨2⟩2. $\cdot; \cdot; \cdot; \mathcal{R} \vdash \overline{x_i}^i = \overline{spine_elem_i}^i :: arg \gg \sigma; ret$ 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 $\sigma: \cdot; \cdot; \mathcal{R} \vdash (\sigma):(\mathcal{C}; \mathcal{L}; \mathcal{R}')$.

PROOF: Constructing such a substitution requires $\mathcal{C}; \mathcal{L}; \Phi \vdash pval_i \Rightarrow \beta_i$ for each $x_i:\beta_i \in \mathcal{C}$ or $x_i:\beta_i \in \mathcal{L}$ and $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}' \vdash res_term_i \Leftarrow res_i$ for each $res_i \in \mathcal{R}'$ which can be deduced from ⟨2⟩2.

⟨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: $\text{TY_IS_E_}\{\text{NEG_}\}\text{ACTION}$.

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 $\langle pexpr \rangle \longrightarrow \langle tpxpr:(y:\beta. term) \rangle$ case.

ASSUME: 1. $\cdot; \cdot; \cdot \vdash \text{let } ident_or_pattern = pexpr \text{ in } tpxpr \Leftarrow y_2:\beta_2. term_2$.

2. $\langle \text{let } ident_or_pattern = pexpr \text{ in } tpxpr \rangle \longrightarrow \langle \text{let } ident_or_pattern = pexpr' \text{ in } tpxpr \rangle$.

PROVE: $\cdot; \cdot; \cdot \vdash \text{let } ident_or_pattern = pexpr' \text{ in } tpxpr \Leftarrow y_2:\beta_2. term_2$.

⟨2⟩1. 1. $\cdot; \cdot; \cdot \vdash pexpr \Rightarrow y:\beta. term$.

2. $ident_or_pattern:\beta \rightsquigarrow \mathcal{C}_1 \text{ with } term_1$.

3. $\mathcal{C}_1; \cdot; \cdot, term_1/y, \cdot(term), \Phi_1; \mathcal{R} \vdash tpxpr \Leftarrow ret$.

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. $\cdot; \cdot; \cdot \vdash \text{let } ident_or_pattern = pexpr' \text{ in } tpxpr \Leftarrow y_2:\beta_2. term_2$.

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. $\cdot; \cdot; \cdot; \mathcal{R}' \vdash \text{let } \overline{ret_pattern_i}^i = seq_expr \text{ in } texpr_2 \Leftarrow ret_2$.

2. $\langle h; \text{let } \overline{ret_pattern_i}^i = seq_expr \text{ in } texpr_2 \rangle \longrightarrow \langle h; \text{let } \overline{ret_pattern_i}^i : ret'_1 = texpr_1 \text{ in } texpr_2 \rangle$.

PROVE: $\cdot; \cdot; \cdot; \mathcal{R}' \vdash \text{let } \overline{ret_pattern_i}^i : ret_1 = texpr_1 \text{ in } texpr_2 \Leftarrow ret_2$.

⟨2⟩1. 1. $\cdot; \cdot; \cdot; \mathcal{R}' \vdash seq_expr \Rightarrow ret_1$.

2. $\overline{ret_pattern_i}^i : ret_1 \rightsquigarrow \mathcal{C}_1; \mathcal{L}_1; \Phi_1; \mathcal{R}_1$.

3. $\mathcal{C}_1; \mathcal{L}_1; \Phi_1; \mathcal{R}, \mathcal{R}_1 \vdash texpr \Leftarrow ret_2$.

PROOF: By inversion on 1.

⟨2⟩2. $\langle h; seq_expr \rangle \longrightarrow \langle h; texpr_1 : ret'_1 \rangle$.

PROOF: By inversion on 2.

$\langle 2 \rangle 3.$ $\cdot; \cdot; \cdot; \mathcal{R}' \vdash \text{texpr}_1 \Leftarrow \text{ret}_1.$

PROOF: By induction on $\langle 2 \rangle 1.1$ and $\langle 2 \rangle 2.$

$\langle 2 \rangle 4.$ $\text{ret}_1 = \text{ret}'_1.$

PROOF: By cases $\text{TY_SEQ_E_}\{\text{CCALL}, \text{PCALL}\}.$

$\langle 2 \rangle 5.$ 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: $\text{TY_SEQ_TE_LETT}.$

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

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

PROVE: $\cdot; \cdot; \cdot; \mathcal{R}', \mathcal{R} \vdash \sigma(\text{texpr}_2) \Leftarrow \sigma(\text{ret}_2).$

$\langle 2 \rangle 1.$ 1. $\cdot; \cdot; \cdot; \mathcal{R}' \vdash \text{done } \overline{\text{spine_elem}_i}^i \Leftarrow \text{ret}_1.$

2. $\overline{\text{ret_pattern}_i}^i : \text{ret}_1 \rightsquigarrow \mathcal{C}_1; \mathcal{L}_1; \Phi_1; \mathcal{R}_1.$

3. $\mathcal{C}_1; \mathcal{L}_1; \Phi_1; \mathcal{R}_1, \mathcal{R} \vdash \text{texpr}_2 \Leftarrow \text{ret}_2.$

PROOF: By inversion on 1.

$\langle 2 \rangle 2.$ $\overline{\text{ret_pattern}_i}^i = \overline{\text{spine_elem}_i}^i \rightsquigarrow \sigma.$

PROOF: By inversion on 2.

$\langle 2 \rangle 3.$ $\cdot; \cdot; \cdot; \mathcal{R}' \vdash (\sigma)(\mathcal{C}_1; \mathcal{L}_1; \mathcal{R}_1).$

PROOF: By $\langle 2 \rangle 1.1, 2$ and $\langle 2 \rangle 2$ using lemma 5.2.

$\langle 2 \rangle 4.$ By $\langle 2 \rangle 1.3$ and $\langle 2 \rangle 3$ and lemma 2.4, we are done.

$\langle 1 \rangle 32.$ CASE: $\text{TY_SEQ_TE_LETT}.$

ASSUME: 1. $\cdot; \cdot; \cdot; \mathcal{R}', \mathcal{R} \vdash \text{let } \overline{\text{ret_pattern}_i}^i : \text{ret}_1 = \text{texpr}_1 \text{ in } \text{texpr}_2 \Leftarrow \text{ret}_2.$

2. $\langle h; \text{let } \overline{\text{ret_pattern}_i}^i : \text{ret} = \text{texpr}_1 \text{ in } \text{texpr}_2 \rangle \longrightarrow \langle h'; \text{let } \overline{\text{ret_pattern}_i}^i : \text{ret} = \text{texpr}'_1 \text{ in } \text{texpr}_2 \rangle.$

PROVE: $\cdot; \cdot; \cdot; \mathcal{R}'', \mathcal{R} \vdash \text{let } \overline{\text{ret_pattern}_i}^i : \text{ret}_1 = \text{texpr}'_1 \text{ in } \text{texpr}_2 \Leftarrow \text{ret}_2.$

$\langle 2 \rangle 1.$ 1. $\cdot; \cdot; \cdot; \mathcal{R}' \vdash \text{texpr}_1 \Leftarrow \text{ret}_1.$

2. $\overline{\text{ret_pattern}_i}^i : \text{ret}_1 \rightsquigarrow \mathcal{C}_1; \mathcal{L}_1; \Phi_1; \mathcal{R}_1.$

3. $\mathcal{C}_1; \mathcal{L}_1; \Phi_1; \mathcal{R}_1, \mathcal{R} \vdash \text{texpr}_2 \Leftarrow \text{ret}_2.$

PROOF: By inversion on 1.

$\langle 2 \rangle 2.$ $\langle h; \text{texpr}_1 \rangle \longrightarrow \langle h'; \text{texpr}'_1 \rangle.$

PROOF: By inversion on 2.

$\langle 2 \rangle 3.$ $\cdot; \cdot; \cdot; \mathcal{R}'' \vdash \text{texpr}'_1 \Leftarrow \text{ret}_1.$

PROOF: By induction on $\langle 2 \rangle 1.1$ and $\langle 2 \rangle 2.$

$\langle 2 \rangle 4.$ By $\langle 2 \rangle 3$, $\langle 2 \rangle 1.2, 3$ using TY_SEQ_TE_LETT , we are done.

$\langle 1 \rangle 33.$ CASE: $\text{TY_SEQ_TE_CASE}.$

ASSUME: 1. $\cdot; \cdot; \cdot; \mathcal{R} \vdash \text{case pval of } \overline{\text{pattern}_i \Rightarrow \text{texpr}_i}^i \text{ end} \Leftarrow \text{ret}.$

2. $\langle h; \text{case pval of } \overline{\text{pattern}_i \Rightarrow \text{texpr}_i}^i \text{ end} \rangle \longrightarrow \langle h; \sigma_j(\text{texpr}_j) \rangle.$

PROVE: $\cdot; \cdot; \cdot; \mathcal{R} \vdash \sigma_j(\text{texpr}_j) \Leftarrow \text{ret}.$

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

$\langle 2 \rangle 2.$ 1. $pattern_j = pval \rightsquigarrow \sigma_j.$
 2. $\forall i < j. \text{ not } (pattern_i = pval \rightsquigarrow \sigma_i).$
 PROOF: By inversion on 2.

$\langle 2 \rangle 3.$ $\cdot; \cdot; \cdot \vdash (\sigma_j); (\mathcal{C}_i; \cdot; \cdot).$
 PROOF: By lemma 5.2.

$\langle 2 \rangle 4.$ By ??3 and 2.2, we are done.

$\langle 1 \rangle 34.$ CASE: TY_SEQ_TE_IF.

Only covering **True** case, **False** is almost identical.

ASSUME: 1. $\cdot; \cdot; \cdot; \mathcal{R} \vdash \text{if True then expr}_1 \text{ else expr}_2 \Leftarrow ret.$

2. $\langle h; \text{if True then expr}_1 \text{ else expr}_2 \rangle \longrightarrow \langle h; \text{expr}_1 \rangle.$

PROVE: $\cdot; \cdot; \cdot; \mathcal{R} \vdash \text{expr}_1 \Leftarrow ret.$

PROOF: Invert 1, note $\cdot; \cdot; \mathcal{R} \vdash (\text{id}); (\cdot; \cdot; \mathcal{R})$ and then apply substitution lemma (2.2).

$\langle 1 \rangle 35.$ CASE: TY_SEQ_TE_RUN.

PROOF SKETCH: Similar to case TY_SEQ_E_{CCALL, PCALL}.

$\langle 1 \rangle 36.$ CASE: TY_SEQ_TE_BOUND.

PROOF: By inversion on the typing rule.

$\langle 1 \rangle 37.$ CASE: TY_IS_TE_LETS.

PROOF SKETCH: Similar to TY_SEQ_TE_LETT.

6 Typing Judgements

$object_value_jtype$	$::=$ $\mathcal{C}; \mathcal{L}; \Phi \vdash object_value \Rightarrow \mathbf{obj} \beta$
$pval_jtype$	$::=$ $\mathcal{C}; \mathcal{L}; \Phi \vdash pval \Rightarrow \beta$
res_jtype	$::=$ $\Phi \vdash res \equiv res'$ $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash res_term \Leftarrow res$
$spine_jtype$	$::=$ $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{x_i = spine_elem_i}^i :: arg \gg \sigma; ret$
$pexpr_jtype$	$::=$ $\mathcal{C}; \mathcal{L}; \Phi \vdash pexpr \Rightarrow ident:\beta. term$
$tpval_jtype$	$::=$ $\mathcal{C}; \mathcal{L}; \Phi \vdash tpval \Leftarrow ident:\beta. term$
$tpexpr_jtype$	$::=$ $\mathcal{C}; \mathcal{L}; \Phi \vdash tpexpr \Leftarrow ident:\beta. term$
$action_jtype$	$::=$ $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash mem_action \Rightarrow ret$
$memop_jtype$	$::=$ $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash mem_op \Rightarrow ret$
seq_expr_jtype	$::=$ $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash seq_expr \Rightarrow ret$
is_expr_jtype	$::=$ $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash is_expr \Rightarrow ret$
$tval_jtype$	$::=$ $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash tval \Leftarrow ret$
$texpr_jtype$	$::=$ $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash seq_texpr \Leftarrow ret$ $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash is_texpr \Leftarrow ret$ $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash texpr \Leftarrow ret$

7 Opsem Judgements

$$\begin{array}{lcl}
 \text{pure_opsem_jtype} & ::= & \\
 & | & \langle pexpr \rangle \longrightarrow \langle pexpr' \rangle \\
 & | & \langle pexpr \rangle \longrightarrow \langle tpepr:(y:\beta. term) \rangle \\
 & | & \langle tpepr \rangle \longrightarrow \langle tpepr' \rangle
 \end{array}$$

$$\begin{array}{lcl}
 \text{opsem_jtype} & ::= & \\
 & | & \langle h; seq_expr \rangle \longrightarrow \langle h'; texpr:ret \rangle \\
 & | & \langle h; seq_texpr \rangle \longrightarrow \langle h'; texpr \rangle \\
 & | & \langle h; mem_op \rangle \longrightarrow \langle h'; tval \rangle \\
 & | & \langle h; mem_action \rangle \longrightarrow \langle h'; tval \rangle \\
 & | & \langle h; is_expr \rangle \longrightarrow \langle h'; is_expr' \rangle \\
 & | & \langle h; is_texpr \rangle \longrightarrow \langle h'; texpr \rangle \\
 & | & \langle h; texpr \rangle \longrightarrow \langle h'; texpr' \rangle
 \end{array}$$