

# Explicit CN Soundness Proof

Dhruv Makwana

November 15, 2021

## 1 Weakening

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

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

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

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

$\langle 1 \rangle 1$ . CASE:  $\text{TY\_PVAL\_VAR}\_{\{\text{COMP}, \text{LOG}\}}$ .

PROOF: By  $\text{WEAK\_CONS}\_{\{\text{COMP}, \text{LOG}\}}$ , if  $x:\beta \in \mathcal{C}$  (or  $x:\beta \in \mathcal{L}$ ) then  $x:\beta \in \mathcal{C}'$  (or  $x:\beta \in \mathcal{L}$ ).

$\langle 1 \rangle 2$ . CASE:  $\text{TY\_PVAL\_ERROR}$ ,  $\text{TY\_RES\_EQ}\_{\{\text{POINTSTO}, \text{TERM}\}}$ ,  $\text{TY\_RES\_CONJ}$ ,  
 $\text{TY\_SPINE\_RES\_PHI}$ ,  $\text{TY\_PE\_ASSERTUNDEF}$ ,  $\text{TY\_TPVAL}\_{\{\text{UNDEF}, \text{DONE}\}}$ ,  
 $\text{TY\_ACTION}\_{\{\text{LOAD}, \text{STORE}, \text{KILL}\}}$ ,  $\text{TY\_MEMOP\_PTRVALIDFORDEREF}$ ,  
 $\text{TY\_TVAL}\_{\{\text{PHI}, \text{UNDEF}\}}$ .

ASSUME:  $\text{smt}(\Phi \Rightarrow \text{term}')$ .

PROVE:  $\text{smt}(\Phi' \Rightarrow \text{term}')$ .

$\langle 2 \rangle 1$ . If  $\text{term} \in \Phi$  then  $\text{term} \in \Phi'$ . PROOF: By  $\text{WEAK\_CONS\_PHI}$ .

$\langle 2 \rangle 2$ . Any extra constraints in  $\Phi'$  (by  $\text{WEAK\_SKIP\_PHI}$ ) would either be irrelevant, redundant, or inconsistent.

$\langle 2 \rangle 3$ . In all cases,  $\text{smt}(\Phi' \Rightarrow \text{term}')$  as required.

$\langle 1 \rangle 3$ . CASE:  $\text{TY\_RES}\_{\{\text{EMP}, \text{POINTSTO}, \text{VAR}, \text{SEPCONJ}\}}$ ,  $\text{TY\_SPINE}\_{\{\text{EMPTY}, \text{RES}\}}$ ,  
 $\text{TY\_ACTION\_CREATE}$ ,  $\text{TY\_TVAL\_RES}$ ,  $\text{TY\_MEMOP}\_{\{\text{REL\_BINOP},$   
 $\text{INTFROMPTR}, \text{PTRFROMINT}, \text{WELLALIGNED}, \text{PTRARRAYSHIFT}\}}$ ,  
 $\text{TY\_TVAL}\_{\{\text{I}, \text{UNDEF}\}}$ ,  $\text{TY\_SEQ\_TE}\_{\{\text{LET}, \text{LETT}, \text{RUN}\}}$ ,  $\text{TY\_IS\_TE\_LETS}$ .

$\langle 2 \rangle 1$ .  $\mathcal{R} = \mathcal{R}'$ .

PROOF: Only  $\text{WEAK\_CONS\_RES}$  exists, no  $\text{WEAK\_SKIP\_RES}$ .

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

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

## 2 Substitution

### 2.1 Weakening for Substitution

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

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

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

PROOF SKETCH: By weakening and induction over the substitution.

### 2.2 Substitutions preserve SMT results

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

PROVE:  $\text{smt}(\Phi \Rightarrow \sigma(\text{term}))$ .

$\langle 1 \rangle 1$ .  $\text{smt}(\Phi' \Rightarrow \sigma(\text{term}))$ .

PROOF: By assumption 1, which means it is true for all (well-typed) instantiations of its free variables.

$\langle 1 \rangle 2$ .  $\text{smt}(\Phi \Rightarrow \sigma(\text{term}))$ .

PROOF: By  $\text{smt}(\Phi \Rightarrow \text{term})$  for each  $\text{term} \in \Phi'$  (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.  $\Phi \vdash \text{res} \equiv \text{res}'$ .  
 2.  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \text{res\_term} \Leftarrow \text{res}$ .

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

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

$\langle 1 \rangle 1$ . CASE: TY\_RES\_EMP

PROOF:  $\text{res} = \text{res}' = \text{res\_term} = \text{emp}$ .

$\langle 1 \rangle 2$ . CASE: TY\_RES\_POINTS\_TO

$\text{res} = \text{points\_to}'', \text{res\_term} = \text{points\_to}', \text{res}' = \text{points\_to}_1, \mathcal{R} = \cdot, \cdot : \text{points\_to}$ .

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

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

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

- ⟨1⟩3. CASE: TY\_RES\_VAR  
PROOF: By transitivity (lemma 2.3).
- ⟨1⟩4. CASE: TY\_RES\_SEPCONJ  
PROOF: By induction.
- ⟨1⟩5. CASE: TY\_RES\_CONJ  
PROOF: We know  $\text{smt}(\Phi \Rightarrow (term \rightarrow term'))$  (by inversion on the equality) and  $\text{smt}(\Phi \Rightarrow term)$  (by inversion on the typing rule) so  $\text{smt}(\Phi \Rightarrow term')$ . Rest follows by induction.
- ⟨1⟩6. CASE: TY\_RES\_PACK  
 $res\_term = \text{pack}(pval, res\_term'), res = \exists y:\beta. res_1, res' = \exists y:\beta. res'_1$ .
- ⟨2⟩1.  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash res\_term' \Leftarrow pval/y, \cdot (res'_1)$  by induction.
- ⟨2⟩2.  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \text{pack}(pval, res\_term') \Leftarrow \exists y:\beta. res'_1$  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  $\exists pval. \mathcal{C}; \mathcal{L}; \Phi \vdash pval \Rightarrow \beta$  by TY\_SUBS\_CONS\_{COMP,LOG}.
- ⟨2⟩3. Since  $pval = \sigma(x)$ , we are done.
- ⟨1⟩4. CASE: TY\_PVAL\_ERROR.  
PROOF: Substitutions preserve SMT results (lemma 2.2).
- ⟨1⟩5. CASE: TY\_PVAL\_STRUCT.  
 $\mathcal{C}'; \mathcal{L}'; \Phi' \vdash (\text{struct } tag) \{ \overline{member_i = pval_i}^i \} \Rightarrow \text{struct } tag$
- ⟨2⟩1.  $\overline{\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(pval_i) \Rightarrow \beta_{\tau_i}^i}$  by induction.

- $\langle 2 \rangle 2. \mathcal{C}; \mathcal{L}; \Phi \vdash (\mathbf{struct\ tag})\{ \overline{.member_i = \sigma(pval_i)}^i \} \Rightarrow \mathbf{struct\ tag}$
- $\langle 1 \rangle 6. \text{ CASE: TY\_EQ\_EMP}$   
PROOF: True trivially (no free variables).
- $\langle 1 \rangle 7. \text{ CASE: TY\_RES\_EQ\_POINTSTO.}$   
PROOF: Substitutions preserver SMT results (lemma 2.2).
- $\langle 1 \rangle 8. \text{ CASE: TY\_RES\_EQ\_SEP\_CONJ.}$   
PROOF: By induction.
- $\langle 1 \rangle 9. \text{ CASE: TY\_RES\_EQ\_EXISTS.}$   
PROOF: By induction.
- $\langle 1 \rangle 10. \text{ CASE: TY\_RES\_EQ\_TERM.}$   
PROOF: By induction and substitutions preserving SMT results (lemma 2.2).
- $\langle 1 \rangle 11. \text{ CASE: TY\_RES\_EMP.}$   
PROOF: True trivially (no free variables).
- $\langle 1 \rangle 12. \text{ CASE: TY\_RES\_POINTSTO.}$   
 $\mathcal{C}'; \mathcal{L}'; \Phi'; \cdot, \cdot : pt \vdash pt' \Leftarrow pt''.$   
PROVE:  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(pt') \Leftarrow \sigma(pt'').$
- $\langle 2 \rangle 1. \text{ Since } \mathcal{R}' = \cdot, \cdot : pt, \sigma \text{ was derived using TY\_SUBS\_CONS\_RES.}$
- $\langle 2 \rangle 2. \Phi' \vdash pt \equiv pt' \text{ and } \Phi' \vdash pt' \equiv pt'' \text{ by inversion on the case.}$
- $\langle 2 \rangle 3. \text{ So } \Phi \vdash \sigma(pt) \equiv \sigma(pt') \text{ and } \Phi \vdash \sigma(pt') \equiv \sigma(pt'') \text{ because substitutions preserve SMT results (lemma 2.2).}$
- $\langle 2 \rangle 4. \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash res\_term \Leftarrow \sigma(pt) \text{ by inversion on } \langle 2 \rangle 1.$
- $\langle 2 \rangle 5. res\_term = pt_3 \text{ for some } pt_3 \text{ by inversion on } \langle 2 \rangle 4 \text{ (TY\_RES\_POINTSTO).}$
- $\langle 2 \rangle 6. \Phi \vdash pt_3 \equiv \sigma(pt) \text{ by inversion on } \langle 2 \rangle 3.$
- $\langle 2 \rangle 7. \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(pt') \Leftarrow pt_3.$   
PROOF: TY\\_RES\\_POINTSTO is symmetric in all its  $pt$  arguments (because resource equality is an equivalence relation, lemma 2.3).
- $\langle 2 \rangle 8. \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(pt') \Leftarrow \sigma(pt'').$   
PROOF: By  $\langle 2 \rangle 3$ , resource equality an equivalence relation (lemma 2.3) and resource typing subsumption (lemma 2.4).
- $\langle 1 \rangle 13. \text{ CASE: TY\_RES\_VAR.}$   
 $\mathcal{C}'; \mathcal{L}'; \Phi'; \cdot, r : res \vdash r \Leftarrow res'.$
- $\langle 2 \rangle 1. \text{ From } \mathcal{R}' = \cdot, r : res, \text{ we know } \sigma \text{ was derived using TY\_SUBS\_CONS\_RES.}$
- $\langle 2 \rangle 2. \sigma = res\_term / r, \sigma' \text{ and } \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash res\_term \Leftarrow \sigma'(res) \text{ by inversion on } \langle 2 \rangle 1.$
- $\langle 2 \rangle 3. \Phi' \vdash res \equiv res' \text{ by inversion on TY\_RES\_VAR.}$

- $\langle 2 \rangle 4.$   $\Phi \vdash res \equiv res'$  and  $\Phi \vdash \sigma(res) \equiv \sigma(res')$  by  $\langle 2 \rangle 3$  and substitution lemma over  $TY\_RES\_EQ^*$  cases.
- $\langle 2 \rangle 5.$   $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash res\_term \Leftarrow \sigma'(res)$  by inversion on  $TY\_SUBS\_CONS\_RES$ .
- $\langle 2 \rangle 6.$   $\sigma(r) = res\_term$  by  $\langle 2 \rangle 2$ .
- $\langle 2 \rangle 7.$   $\sigma'(res') = \sigma(res')$  (and same for  $res$ ) because  $r$  cannot occur in either.
- $\langle 2 \rangle 8.$  SUFFICES:  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash res\_term \Leftarrow \sigma'(res')$  by  $\langle 2 \rangle 3$  and  $\langle 2 \rangle 7$ .  
 PROOF: Resource typing subsumption (lemma 2.4) and  $\langle 2 \rangle 4$ .
- $\langle 1 \rangle 14.$  CASE:  $TY\_RES\_SEP\_CONJ$ .  
 PROOF: By induction.
- $\langle 1 \rangle 15.$  CASE:  $TY\_RES\_CONJ$ .  
 $\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash res\_term \Leftarrow term \wedge res$ .
- $\langle 2 \rangle 1.$   $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(res\_term) \Leftarrow \sigma(res)$ .  
 PROOF: By induction.
- $\langle 2 \rangle 2.$   $smt(\Phi \Rightarrow \sigma(term))$ .  
 PROOF: Substitutions preserve SMT results (lemma 2.2).
- $\langle 2 \rangle 3.$   $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(res\_term) \Leftarrow \sigma(term \wedge res)$  as required.
- $\langle 1 \rangle 16.$  CASE:  $TY\_RES\_PACK$ .  
 $\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash pack(pval, res\_term) \Leftarrow \exists y:\beta. res$ .
- $\langle 2 \rangle 1.$  By induction,  
 1.  $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(pval) \Rightarrow \beta$ .  
 2.  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(res\_term) \Leftarrow \sigma, pval/y, \cdot(res)$ .
- $\langle 2 \rangle 2.$  So  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(pack(pval, res\_term)) \Leftarrow \sigma(\exists y:\beta. res)$ .
- $\langle 1 \rangle 17.$  CASE:  $TY\_SPINE\_EMPTY$ .  
 PROOF:  $ret$  can be anything, including  $\sigma(ret)$  and the rule does not depend on the environment, so we are done.
- $\langle 1 \rangle 18.$  CASE:  $TY\_SPINE\_COMP$ .  
 $\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash x = pval, \overline{x_i = spine\_elem_i}^i :: \Pi x:\beta. arg \gg pval/x, \psi; ret$ .
- $\langle 2 \rangle 1.$  By induction,  
 1.  $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(pval) \Rightarrow \beta$ .  
 2.  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{x_i = \sigma(spine\_elem_i)}^i :: \sigma(arg) \gg \sigma(\psi); \sigma(ret)$ .
- $\langle 2 \rangle 2.$  So  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash x = \sigma(pval), \overline{x_i = \sigma(spine\_elem_i)}^i :: \sigma(\Pi x:\beta. arg) \gg \sigma(pval/x, \psi); \sigma(ret)$ .
- $\langle 1 \rangle 19.$  CASE:  $TY\_SPINE\_LOG$ .  
 PROOF: Similar to  $TY\_SPINE\_COMP$ .
- $\langle 1 \rangle 20.$  CASE:  $TY\_SPINE\_RES$ .  
 $\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'_1, \mathcal{R}_2 \vdash x = res\_term, \overline{x_i = spine\_elem_i}^i :: res \multimap arg \gg res\_term/x, \psi; ret$

- (2)1. By inversion and then induction,  
 1.  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}_1 \vdash \sigma(res\_term) \Leftarrow \sigma(res)$ .  
 2.  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}_2 \vdash \overline{x_i = \sigma(spine\_elem_i)}^i :: \sigma(res) \multimap \sigma(arg) \gg \sigma(\psi); \sigma(ret)$ .
- (2)2. Hence  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}_1, \mathcal{R}_2 \vdash x = \sigma(res\_term), \overline{x_i = \sigma(spine\_elem_i)}^i :: \sigma(res \multimap arg) \gg \sigma(res\_term/x, \psi); \sigma(ret)$  as required.
- (1)21. CASE: TY\_SPINE\_PHI.  
 $\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash \overline{x_i = spine\_elem_i}^i :: term \supset arg \gg \psi; ret$
- (2)1.  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{x_i = \sigma(spine\_elem_i)}^i :: \sigma(res) \multimap \sigma(arg) \gg \sigma(\psi); \sigma(ret)$ .  
 PROOF: By induction.
- (2)2.  $smt(\Phi \Rightarrow \sigma(term))$ .  
 PROOF: Substitutions preserve SMT results (lemma 2.2).
- (2)3. Hence  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}_1, \mathcal{R}_2 \vdash x = \sigma(res\_term), \overline{x_i = \sigma(spine\_elem_i)}^i :: \sigma(res \multimap arg) \gg \sigma(res\_term/x, \psi); \sigma(ret)$  as required.
- (1)22. CASE: TY\_PE\_VAL  
 PROOF: By induction.
- (1)23. CASE: TY\_PE\_ARRAY\_SHIFT.  
 $\mathcal{C}'; \mathcal{L}'; \Phi' \vdash \text{array\_shift}(pval_1, \tau, pval_2) \Rightarrow y:\text{loc}. y = pval_1 +_{\text{ptr}} (pval_2 \times \text{size\_of}(\tau))$
- (2)1. By induction,  
 1.  $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(pval_1) \Rightarrow \text{loc}$   
 2.  $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(pval_2) \Rightarrow \text{integer}$
- (2)2. So,  $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(\text{array\_shift}(pval_1, \tau, pval_2)) \Rightarrow y:\text{loc}. \sigma((y = pval_1 +_{\text{ptr}} (pval_2 \times \text{size\_of}(\tau))))$ .
- (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\_CALL for more general case and proof.
- (1)27. CASE: TY\_PE\_{ASSERT\_UNDEF, BOOL\_TO\_INTEGER, WRAP\_I}.  
 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  
 $\mathcal{C}'; \mathcal{L}'; \Phi' \vdash \text{done } pval \Leftarrow y:\beta. term$ .
- (2)1.  $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(pval) \Rightarrow \beta$ .  
 PROOF: By induction.

- ⟨2⟩2.  $\text{smt}(\Phi \Rightarrow \sigma, pval/y, \cdot(term))$ .  
 PROOF: Substitutions preserve SMT results (lemma 2.2).
- ⟨2⟩3. So  $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(\text{done } pval) \Leftarrow y:\beta. \sigma(term)$ .
- ⟨1⟩30. CASE:  $\text{TY\_TPE\_}\{\text{LET}, \text{LETT}\}$ .  
 See  $\text{TY\_SEQ\_TE\_}\{\text{LET}, \text{LETT}\}$  for a more general case and proof.
- ⟨1⟩31. CASE:  $\text{TY\_TPE\_IF}$ .  
 PROOF: By induction.
- ⟨1⟩32. CASE:  $\text{TY\_TPE\_CASE}$ .  
 PROOF: See  $\text{TY\_SEQ\_TE\_CASE}$  for more general case and proof.
- ⟨1⟩33. CASE:  $\text{TY\_}\{\text{ACTION}^*, \text{MEMOP}^*\}$ .  
 PROOF: By induction and lemma 2.2 (substitutions preserve SMT results).
- ⟨1⟩34. CASE:  $\text{TY\_TVAL\_I}$   
 PROOF: Trivially (no free variables nor requirements on constraint context).
- ⟨1⟩35. CASE:  $\text{TY\_TVAL\_}\{\text{COMP}, \text{LOG}\}$ .  
 Only focusing on logical case; computational one is similar.  
 $\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash \text{done } pval, \overline{\text{spine\_elem}_i}^i \Leftarrow \exists y:\beta. ret$ .
- ⟨2⟩1. By inversion and then induction,  
 1.  $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(pval) \Rightarrow \beta$   
 2.  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(\text{done } \overline{\text{spine\_elem}_i}^i) \Leftarrow \sigma(pval/y, \cdot(ret))$ .
- ⟨2⟩2. Therefore  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(\text{done } pval, \overline{\text{spine\_elem}_i}^i) \Leftarrow \exists y:\beta. \sigma(ret)$ .
- ⟨1⟩36. CASE:  $\text{TY\_TVAL\_PHI}$   
 $\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash \text{done } spine \Leftarrow term \wedge ret$
- ⟨2⟩1.  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(\text{done } spine) \Leftarrow \sigma(ret)$ .  
 PROOF: By induction.
- ⟨2⟩2.  $\text{smt}(\Phi \Rightarrow \sigma(term))$ .  
 PROOF: Substitutions preserve SMT results (lemma 2.2).
- ⟨2⟩3.  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(\text{done } spine) \Leftarrow \sigma(term \wedge ret)$  as required.
- ⟨1⟩37. CASE:  $\text{TY\_TVAL\_RES}$   
 PROOF: Similar to  $\text{TY\_TVAL\_PHI}$ , except with resource environments being split.
- ⟨1⟩38. CASE:  $\text{TY\_TVAL\_UNDEF}$   
 PROOF:  $ret$  can be anything, including  $\sigma(ret)$ .
- ⟨1⟩39. CASE:  $\text{TY\_SEQ\_TE\_}\{\text{TVAL}, \text{IF}, \text{BOUND}\}$ .  
 PROOF: By induction.
- ⟨1⟩40. CASE:  $\text{TY\_SEQ\_E\_}\{\text{CCALL}, \text{PROC}, \text{RUN}\}$ .  
 Only focusing on  $\text{CCall}$ , rest are similar.

- ⟨2⟩1.  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{x_i = \sigma(\text{spine\_elem}_i)}^i :: \sigma(\text{arg}) \gg \sigma(\psi); \sigma(\text{ret})$ .  
 PROOF: By induction.
- ⟨2⟩2.  $\text{ident:arg} \equiv \overline{x_i}^i \mapsto \text{texpr} \in \text{Globals}$  is unaffected by the substitution.
- ⟨2⟩3.  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \text{ccall}(\tau, \text{ident}, \overline{\sigma(\text{spine\_elem}_i)}^i) \Rightarrow \sigma, \psi(\text{ret})$  as required.
- ⟨1⟩41. CASE:  $\text{TY\_IS\_}\{\text{MEMOP}, \text{NEG\_ACTION}, \text{ACTION}\}$   
 PROOF: By induction.
- ⟨1⟩42. CASE:  $\text{TY\_SEQ\_TE\_}\{\text{LETP}, \text{LETPT}\}$ .  
 PROOF: See  $\text{TY\_SEQ\_TE\_}\{\text{LET}, \text{LETT}\}$ .
- ⟨1⟩43. CASE:  $\text{TY\_SEQ\_TE\_}\{\text{LET}, \text{LETT}, \text{LETS}\}$ .  
 Only doing LET case, LETT and LETS are similar.  
 $\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}''', \mathcal{R}'' \vdash \text{let } \overline{\text{ret\_pattern}_i}^i = \text{seq\_expr in texpr} \Leftarrow \text{ret}_2$ .
- ⟨2⟩1. By induction,  
 1.  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}' \vdash \sigma(\text{seq\_expr}) \Rightarrow \sigma(\text{ret}_1)$ .  
 2.  $\mathcal{C}, \mathcal{C}_1; \mathcal{L}, \mathcal{L}_1; \Phi, \Phi_1; \mathcal{R}, \mathcal{R}_1 \vdash \sigma(\text{texpr}) \Leftarrow \sigma(\text{ret}_2)$ .
- ⟨2⟩2.  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}', \mathcal{R} \vdash \sigma(\text{let } \overline{\text{ret\_pattern}_i}^i = \text{seq\_expr in texpr}) \Leftarrow \sigma(\text{ret}_2)$  as required.
- ⟨1⟩44. CASE:  $\text{TY\_SEQ\_TE\_CASE}$ .  
 $\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash \text{case pval of } \overline{\text{pattern}_i \Rightarrow \text{texpr}_i}^i \text{ end} \Leftarrow \text{ret}$ .
- ⟨2⟩1. By induction,  
 1.  $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma(\text{pval}) \Rightarrow \beta_1$ .  
 2.  $\overline{\mathcal{C}, \mathcal{C}_i; \mathcal{L}; \Phi, \text{term}_i = \sigma(\text{pval}); \mathcal{R} \vdash \sigma(\text{texpr}_i) \Leftarrow \sigma(\text{ret})}^i$ .
- ⟨2⟩2.  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma(\text{case pval of } \overline{\text{pattern}_i \Rightarrow \text{texpr}_i}^i \text{ end}) \Leftarrow \sigma(\text{ret})$  as required.
- ⟨1⟩45. CASE:  $\text{TY\_TE\_}\{\text{IS}, \text{SEQ}\}$ .  
 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  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}_1, \mathcal{R} \vdash (\sigma, \text{id}):(\mathcal{C}, \mathcal{C}'; \mathcal{L}, \mathcal{L}'; \Phi'; \mathcal{R}_1, \mathcal{R}')$ .

PROOF SKETCH: Induction over the substitution.

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

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

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

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



## 2.7 Let-friendly Substitution Lemma

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

PROOF SKETCH: Apply identity extension then substitution lemma.

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

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

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

PROOF: Apply identity extension to 1.

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

PROOF: Apply substitution lemma (2.5) to  $\langle 1 \rangle 1$ .

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

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

## 3 Progress

### 3.1 Ty\_Spine\_\* and Decons\_Arg\_\* construct same substitution and return type

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

PROOF SKETCH: Induction over  $\text{arg}$ .

### 3.2 Progress Statement and Proof

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

PROOF SKETCH: Induction over the typing rules.

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

2. All patterns in  $e$  are exhaustive.

PROVE: Either  $e$  is a value, or it is unreachable, or  $\forall h : R. \exists e', h'. \langle h; e \rangle \longrightarrow \langle h'; e' \rangle$ .

$\langle 1 \rangle 1.$  CASE:  $\text{TY\_PVAL\_OBJ}^*, \text{TY\_PVAL}^*, \text{TY\_PE\_VAL}, \text{TY\_TPVAL}^*, \text{TY\_TVAL}^*, \text{TY\_SEQ\_TE\_TVAL}$ .

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:  $\text{TY\_PE\_ARRAY\_SHIFT}$ .

PROOF: By inversion on  $\cdot; \cdot; \cdot \vdash pval_1 \Rightarrow \text{loc}, pval_1$  must be a *mem\_ptr* ( $\text{TY\_PVAL\_OBJ\_PTR}$ ). Similarly  $pval_2$  must be a *mem\_int*, so rule  $\text{OP\_PE\_PE\_ARRAYSHIFT}$  applies.

$\langle 1 \rangle 3.$  CASE:  $\text{TY\_PE\_MEMBER\_SHIFT}$ .

PROOF:  $pval$  must be a *mem\_ptr* so  $\text{OP\_PE\_PE\_MEMBERSHIFT}$ .

- ⟨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  $name: pure\_arg \equiv \overline{x_i}^i \mapsto texpr \in \mathbf{Globals}$  and  $\cdot; \cdot; \cdot \vdash \overline{x_i = pval_i}^i :: pure\_arg \gg \sigma; \Sigma y:\beta. term \wedge \mathbf{I}$ , with the latter implying  $\overline{x_i = pval_i}^i :: pure\_arg \gg \sigma; \Sigma y:\beta. term \wedge \mathbf{I}$  (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  $\mathbf{smt}(\Phi \Rightarrow pval)$ . 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\_ptr* 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  $h = \cdot + \{pval_1 \xrightarrow{\check{\tau}} pval_2\}$ , 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  $\cdot + \{mem\_ptr \mapsto_{\tau} \cdot\}$  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  $ident:arg \equiv \bar{x}_i^i \mapsto texpr \in \mathbf{Globals}$  and  $\cdot; \cdot; \cdot \vdash \frac{x_i = spine\_elem_i^i :: arg \gg \sigma; ret}{x_i = spine\_elem_i^i :: arg \gg \sigma; ret}$  (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_IS_E_IS_E_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_IS_E_IS_E_{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`. 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  $\exists h_1, h'_1, h_2. \text{disjoint}(h_1, h_2) \wedge h = h_1 + h_2 \wedge \langle h_1; e \rangle \longrightarrow \langle h'_1; e' \rangle$  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,  $\text{mem\_ptr} \in h_1$  implies  $\text{mem\_ptr} \notin h_2$ .

## 5 Type Preservation

### 5.1 Pointed-to values have type $\beta_\tau$

For  $pt = \_ \check{\mapsto}_\tau pval$ , 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.  $\text{pattern}:\beta \rightsquigarrow \mathcal{C} \text{ with } term$ .  
 2.  $\text{pattern} = pval \rightsquigarrow \sigma$ .

PROVE: The constraint  $term = pval$  holds.

PROOF SKETCH: Induction over *pattern*.

### 5.3 strip\_ifs is idempotent

PROOF SKETCH: Induction over the definition.

### 5.4 Deconstructing a stripped resource produces the same environment

ASSUME: 1.  $\Phi \vdash \text{res\_pattern}:\text{res} \rightsquigarrow \mathcal{L}; \Phi; \mathcal{R}$ .

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

$\langle 1 \rangle 1$ . SUFFICES:  $\Phi \vdash res' = \text{strip\_ifs}(res')$ .  
 PROOF: By `strip_ifs` idempotent and assumption 2.

$\langle 1 \rangle 2$ .  $\Phi \vdash res' \text{ as } res\_pattern \rightsquigarrow \mathcal{L}; \Phi; \mathcal{R}$  by inversion on 1.

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

## 5.5 Deconstructing a pattern leads to a well-typed substitution

First, computational part.

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

PROOF SKETCH: By induction over 2.

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

$\langle 1 \rangle 2$ . CASE: `TY_PAT_NO_SYM_ANNOT` and `TY_PAT_COMP_NIL`.  
 $\sigma$  and  $\mathcal{C}$  are empty.  
 PROOF: By `TY_SUBS_EMPTY`, we are done.

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

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

ASSUME: 1.  $\cdot; \cdot; \cdot; \mathcal{R} \vdash res\_term \Leftarrow res$ .  
 2.  $\Phi \vdash res\_pattern:res \rightsquigarrow \mathcal{L}'; \Phi'; \mathcal{R}'$ .  
 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 1.

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

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

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

⟨1⟩4. CASE: TY\_RES\_SEPCONJ.

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

⟨1⟩5. CASE: TY\_RES\_CONJ.

PROOF: By  $\text{smt}(\cdot \Rightarrow \text{term})$  (from 1) and induction with TY\_SUB\_CONS\_PHI.

⟨1⟩6. CASE: TY\_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.

⟨1⟩7. CASE: TY\_RES\_FOLD.

$\text{res\_pattern} = \text{fold}(\text{res\_pattern}')$ ,  $\text{res\_term} = \text{fold}(\text{res\_term}')$ ,  $\text{res} = \alpha(\overline{\text{pval}_i}^i)$ .

⟨2⟩1. 1.  $\alpha \equiv \overline{x_i:\beta_i}^i \mapsto \text{res}' \in \text{Globals}$ .

2.  $\Phi \vdash \text{res}'' = \text{strip\_ifs}(\text{res}')$ .

3.  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \text{res\_term}' \Leftarrow \text{res}''$ .

PROOF: Inversion on 1.

⟨2⟩2.  $\Phi \vdash \text{res\_pattern}': \overline{\text{pval}_i/x_i, \cdot}^i(\text{res}') \rightsquigarrow \mathcal{L}'; \Phi'; \mathcal{R}'$ .

PROOF: Inversion on 2.

⟨2⟩3.  $\Phi \vdash \text{res\_pattern}': \text{res}'' \rightsquigarrow \mathcal{L}'; \Phi'; \mathcal{R}'$ .

PROOF: By ⟨2⟩1.2, ⟨2⟩2 and deconstructing a stripped resource produces the same environment (lemma 5.4).

⟨2⟩4.  $\text{res\_pattern}' = \text{res\_term}' \rightsquigarrow \sigma$ .

PROOF: By inversion on 3.

⟨2⟩5.  $\cdot; \cdot; \cdot; \mathcal{R} \vdash (\sigma):(\cdot; \mathcal{L}; \Phi; \mathcal{R}')$ .

PROOF: By induction on ⟨2⟩1.3, ⟨2⟩3 and ⟨2⟩4.

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

ASSUME: 1.  $\overline{\text{ret\_pattern}_i} = \overline{\text{spine\_elem}_i}^i \rightsquigarrow \sigma$ .

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

3.  $\Phi \vdash \overline{\text{ret\_pattern}_i}^i : \text{ret} \rightsquigarrow \mathcal{C}; \mathcal{L}'; \Phi'; \mathcal{R}'$ .

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  $\text{smt}(\cdot \Rightarrow \text{term})$   
(required by  $\text{TY\_SUBS\_CONS\_PHI}$ ).

## 5.6 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:  $\text{TY\_PE\_ARRAY\_SHIFT}$ .

LET:  $\text{term} = \text{mem\_ptr} +_{\text{ptr}} (\text{mem\_int} \times \text{size\_of}(\tau))$ .

ASSUME: 1.  $\cdot; \cdot; \cdot \vdash \text{array\_shift}(\text{mem\_ptr}, \tau, \text{mem\_int}) \Rightarrow y:\text{loc}. y = \text{term}$ .

2.  $\langle \text{array\_shift}(\text{mem\_ptr}, \tau, \text{mem\_int}) \rangle \longrightarrow \langle \text{mem\_ptr}' \rangle$ .

PROVE:  $\cdot; \cdot; \cdot \vdash \text{mem\_ptr}' \Rightarrow y:\text{loc}. y = \text{term}$ .

PROOF: By  $\text{TY\_PVAL\_OBJ\_INT}$ ,  $\text{TY\_PVAL\_OBJ}$ ,  $\text{TY\_PE\_VAL}$  and construction of  $\text{mem\_ptr}'$   
(inversion on 2).

<1>2. CASE:  $\text{TY\_PE\_MEMBER\_SHIFT}$ .

PROOF SKETCH: Similar to  $\text{TY\_ARRAY\_SHIFT}$ .

<1>3. CASE:  $\text{TY\_PE\_NOT}$ .

ASSUME: 1.  $\cdot; \cdot; \cdot \vdash \text{not}(\text{bool\_value}) \Rightarrow y:\text{bool}. y = \neg \text{bool\_value}$ .

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

PROVE:  $\cdot; \cdot; \cdot \vdash \text{bool\_value}' \Rightarrow y:\text{bool}. y = \neg \text{bool\_value}$ .

PROOF: By  $\text{TY\_PVAL\_}\{\text{TRUE}, \text{FALSE}\}$ ,  $\text{TY\_PE\_VAL}$  and 2.

<1>4. CASE:  $\text{TY\_PE\_ARITH\_BINOP}$ .

LET:  $\text{term} = \text{mem\_int}_1 \text{binop}_{\text{arith}} \text{mem\_int}_2$ .

ASSUME: 1.  $\cdot; \cdot; \cdot \vdash \text{mem\_int}_1 \text{binop}_{\text{arith}} \text{mem\_int}_2 \Rightarrow y:\text{integer}. y = \text{term}$ .

2.  $\langle \text{mem\_int}_1 \text{binop}_{\text{arith}} \text{mem\_int}_2 \rangle \longrightarrow \langle \text{mem\_int} \rangle$ .

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

PROOF: By  $\text{TY\_PVAL\_OBJ\_INT}$ ,  $\text{TY\_PVAL\_OBJ}$ ,  $\text{TY\_PE\_VAL}$  and construction of  $\text{mem\_int}$   
(inversion on 2).

<1>5. CASE:  $\text{TY\_PE\_}\{\text{REL}, \text{BOOL}\}\_\text{BINOP}$ .

PROOF SKETCH: Similar to  $\text{TY\_PE\_ARITH\_BINOP}$ .

<1>6. CASE:  $\text{TY\_PE\_CALL}$ .

PROOF: See  $\text{TY\_SEQ\_E\_CALL}$  for a more general case and proof.

<1>7. CASE:  $\text{TY\_PE\_ASSERT\_UNDEF}$ .

ASSUME: 1.  $\cdot; \cdot; \cdot \vdash \text{assert\_undef}(\text{True}, \text{UB\_name}) \Rightarrow y:\text{unit}. y = \text{unit}$ .

2.  $\langle \text{assert\_undef}(\text{True}, \text{UB\_name}) \rangle \longrightarrow \langle \text{Unit} \rangle$ .

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

PROOF: By  $\text{TY\_PVAL\_UNIT}$  and  $\text{TY\_PE\_VAL}$ .

- ⟨1⟩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`.
- ⟨1⟩9. CASE: `TY_PE_WRAP1`.  
 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`.
- ⟨1⟩10. CASE: `TY_TPE_IF`.  
 PROOF: See `TY_SEQ_TE_IF` for a more general case and proof.
- ⟨1⟩11. CASE: `TY_TPE_LET`.  
 PROOF: See `TY_SEQ_TE_LET` for a more general case and proof.
- ⟨1⟩12. CASE: `TY_TPE_LETT`.  
 PROOF: See `TY_SEQ_TE_LETT` for a more general case and proof.
- ⟨1⟩13. CASE: `TY_TPE_CASE`.  
 PROOF: See `TY_SEQ_TE_CASE` for a more general case and proof.
- ⟨1⟩14. CASE: `TY_ACTION_CREATE`.  
 LET:  $pt = mem\_ptr \xrightarrow{\times}_\tau pval$ .  
 $term = \text{representable}(\tau*, y_p) \wedge \text{alignedI}(mem\_int, y_p)$ .  
 $ret = \Sigma y_p:\beta_\tau. term \wedge \exists y:\beta_\tau. y_p \xrightarrow{\times}_\tau y \otimes \mathbf{I}$ .  
 ASSUME: 1.  $\cdot; \cdot; \cdot \vdash \text{create}(mem\_int, \tau) \Rightarrow ret$ .  
 2.  $\langle \cdot; \text{create}(mem\_int, \tau) \rangle \longrightarrow \langle \cdot + \{pt\}; \text{done } mem\_ptr, pval, pt \rangle$ .  
 PROVE:  $\cdot; \cdot; \cdot, \cdot \vdash pt \vdash \text{done } mem\_ptr, pval, pt \Leftarrow ret$ .  
 ⟨2⟩1.  $\cdot; \cdot; \cdot \vdash mem\_ptr \Rightarrow \text{loc}$  by `TY_PVAL_OBJ_INT` and `TY_PVAL_OBJ`.  
 ⟨2⟩2.  $\text{smt}(\cdot \Rightarrow term)$  by construction of  $mem\_ptr$ .  
 ⟨2⟩3.  $\cdot; \cdot; \cdot \vdash pval \Rightarrow \beta_\tau$  by construction of  $pval$ .  
 ⟨2⟩4.  $\cdot; \cdot; \cdot, \cdot \vdash pt \vdash pt \Leftarrow pt$  by `TY_RES_POINTS_TO`.  
 ⟨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:  $pt = mem\_ptr \xrightarrow{\check{\times}}_\tau pval$ .  
 $ret = \Sigma y:\beta_\tau. y = pval \wedge pt \otimes \mathbf{I}$ .  
 ASSUME: 1.  $\cdot; \cdot; \cdot, \mathcal{R} \vdash \text{load}(\tau, mem\_ptr, \cdot, pt) \Rightarrow ret$ .  
 2.  $\langle \cdot + \{pt\}; \text{load}(\tau, mem\_ptr, \cdot, pt) \rangle \longrightarrow \langle \cdot + \{pt\}; \text{done } pval, pt \rangle$ .  
 PROVE:  $\cdot; \cdot; \cdot, \mathcal{R} \vdash \text{done } pval, pt \Leftarrow ret$   
 ⟨2⟩1.  $\mathcal{R} = \cdot, \cdot \vdash pt' \text{ where } \cdot \vdash pt' \equiv pt$  by inversion on 1.  
 ⟨2⟩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 pointed-values have the right type (lemma 5.1).
- $\langle 2 \rangle 4.$  By  $\text{TY\_TVAL\_I}$  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:  $\text{TY\_ACTION\_STORE}$ .  
 LET:  $pt = mem\_ptr \xrightarrow{\check{\tau}} \cdot$ .  
 $pt' = mem\_ptr \xrightarrow{\check{\tau}} pval$ .  
 $ret = \Sigma \cdot; \text{unit}. pt' \otimes I$ .  
 ASSUME: 1.  $\cdot; \cdot; \cdot; \mathcal{R} \vdash \text{store}(\cdot, \tau, pval_0, pval_1, \cdot, pt) \Rightarrow ret$ .  
 2.  $\langle \cdot + \{pt\}; \text{store}(\cdot, \tau, mem\_ptr, pval, \cdot, pt) \rangle \longrightarrow \langle \cdot + \{pt'\}; \text{done Unit}, pt' \rangle$ .  
 PROVE:  $\cdot; \cdot; \cdot; \cdot, \cdot; pt' \vdash \text{done Unit}, pt' \Leftarrow ret$ .
- $\langle 2 \rangle 1.$   $\mathcal{R} = \cdot, \cdot; pt_2$  where  $\cdot \vdash pt'' \equiv pt$ , by inversion on the typing assumption.
- $\langle 2 \rangle 2.$   $\cdot; \cdot; \cdot \vdash \text{Unit} \Rightarrow \text{unit}$  by  $\text{TY\_PVAL\_UNIT}$ .
- $\langle 2 \rangle 3.$   $\cdot; \cdot; \cdot; \cdot, \cdot; pt' \vdash pt' \Leftarrow pt'$  by  $\text{TY\_RES\_POINTSTO}$ .
- $\langle 2 \rangle 4.$  By  $\text{TY\_TVAL\_I}$  and  $\langle 2 \rangle 2$  and  $\langle 2 \rangle 3$  with  $\text{TY\_TVAL\_}\{\text{RES}, \text{COMP}\}$  respectively, we are done.
- $\langle 1 \rangle 17.$  CASE:  $\text{TY\_ACTION\_KILL\_STATIC}$ .  
 LET:  $pt = mem\_ptr \mapsto_\tau \cdot$ .  
 ASSUME: 1.  $\cdot; \cdot; \cdot; \mathcal{R} \vdash \text{kill}(\text{static } \tau, pval_0, pt) \Rightarrow \Sigma \cdot; \text{unit}. I$ .  
 2.  $\langle \cdot + \{pt\}; \text{kill}(\text{static } \tau, mem\_ptr, pt) \rangle \longrightarrow \langle h; \text{done Unit} \rangle$ .  
 PROVE:  $\cdot; \cdot; \cdot; \cdot \vdash \text{done Unit} \Leftarrow \Sigma \cdot; \text{unit}. I$   
 PROOF: By  $\text{TY\_TVAL\_I}$ ,  $\text{TY\_PVAL\_UNIT}$  and then  $\text{TY\_TVAL\_COMP}$ . Note:  $\mathcal{R} = \cdot, \cdot; pt'$  where  $\cdot \vdash pt' \equiv pt$ .
- $\langle 1 \rangle 18.$  CASE:  $\text{TY\_MEMOP\_REL\_BINOP}$ .  
 PROOF: Similar  $\text{TY\_PE\_REL\_BINOP}$ , except with  $\text{TY\_TVAL\_}\{\text{I}, \text{PHI}, \text{COMP}\}$  at the end.
- $\langle 1 \rangle 19.$  CASE:  $\text{TY\_MEMOP\_INTFROMPTR}$ .  
 LET:  $ret = \Sigma y; \text{integer}. y = \text{cast\_ptr\_to\_int } mem\_ptr \wedge I$ .  
 ASSUME: 1.  $\cdot; \cdot; \cdot; \cdot \vdash \text{intFromPtr}(\tau_1, \tau_2, mem\_ptr) \Rightarrow ret$ .  
 2.  $\langle \cdot; \text{intFromPtr}(\tau_1, \tau_2, mem\_ptr) \rangle \longrightarrow \langle \cdot; \text{done } mem\_int \rangle$ .  
 PROVE:  $\cdot; \cdot; \cdot; \cdot \vdash \text{done } mem\_int \Leftarrow ret$
- $\langle 2 \rangle 1.$   $\text{smt}(\cdot \Rightarrow mem\_int = \text{cast\_ptr\_to\_int } mem\_ptr)$  by construction of  $mem\_int$  (inversion on 2).
- $\langle 2 \rangle 2.$   $\cdot; \cdot; \cdot \vdash mem\_int \Rightarrow \text{integer}$  by  $\text{TY\_PVAL\_OBJ\_INT}$  and  $\text{TY\_PVAL\_OBJ}$ .
- $\langle 2 \rangle 3.$  By  $\text{TY\_TVAL\_I}$  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  $\text{integer}$  and  $\text{loc}$ .
- $\langle 1 \rangle 21.$  CASE:  $\text{TY\_MEMOP\_PTRVALIDFORDEREF}$ .  
 LET:  $pt = mem\_ptr \xrightarrow{\check{\tau}} \cdot$ .  
 $ret = \Sigma y; \text{bool}. y = \text{aligned}(\tau, mem\_ptr) \wedge pt \otimes I$ .

ASSUME: 1.  $\cdot; \cdot; \cdot; \mathcal{R} \vdash \text{ptrValidForDeref}(\tau, \text{mem\_ptr}, pt) \Rightarrow \text{ret}$ .  
 2.  $\langle \cdot + \{pt\}; \text{ptrValidForDeref}(\tau, \text{mem\_ptr}, pt) \rangle \longrightarrow \langle \cdot + \{pt\}; \text{done } \text{bool\_value}, pt \rangle$ .  
 PROVE:  $\cdot; \cdot; \cdot; \cdot, \cdot, pt \vdash \text{done } \text{bool\_value}, pt \Leftarrow \text{ret}$ .

$\langle 2 \rangle 1$ .  $\cdot; \cdot; \cdot; \cdot, \cdot, pt \vdash pt \Leftarrow pt$ , by inversion on 1.

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

$\langle 2 \rangle 3$ .  $\cdot; \cdot; \cdot \vdash \text{bool\_value} \Rightarrow \text{bool}$  by  $\text{TY\_PVAL\_}\{\text{TRUE}, \text{FALSE}\}$ .

$\langle 2 \rangle 4$ . By  $\text{TY\_TVAL\_I}$ , and then  $\langle 2 \rangle 1 - \langle 2 \rangle 3$  with  $\text{TY\_TVAL\_}\{\text{RES}, \text{PHI}, \text{COMP}\}$  respectively, we are done.

$\langle 1 \rangle 22$ . CASE:  $\text{TY\_MEMOP\_PTRWELLALIGNED}$ .

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

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

2.  $\langle \cdot; \text{ptrWellAligned}(\tau, \text{mem\_ptr}) \rangle \longrightarrow \langle \cdot; \text{done } \text{bool\_value} \rangle$ .

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

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

$\langle 2 \rangle 2$ .  $\cdot; \cdot; \cdot \vdash \text{bool\_value} \Rightarrow \text{bool}$  by  $\text{TY\_PVAL\_}\{\text{TRUE}, \text{FALSE}\}$ .

$\langle 2 \rangle 3$ . By  $\text{TY\_TVAL\_I}$  and  $\langle 2 \rangle 1$  and  $\langle 2 \rangle 2$  with  $\text{TY\_TVAL\_}\{\text{PHI}, \text{COMP}\}$  respectively, we are done.

$\langle 1 \rangle 23$ . CASE:  $\text{TY\_MEMOP\_PTRARRAYSHIFT}$ .

PROOF: Similiar to  $\text{TY\_PE\_ARRAY\_SHIFT}$ , except with  $\text{TY\_TVAL\_}\{\text{I}, \text{PHI}, \text{COMP}\}$  at the end.

$\langle 1 \rangle 24$ . CASE:  $\text{TY\_SEQ\_E\_CCALL}$ .

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

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

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

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

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

$\langle 2 \rangle 3$ .  $\sigma = \sigma'$  and  $\text{ret} = \text{ret}'$  by induction on  $\text{arg}$ .

PROOF:  $\text{TY\_SPINE\_}^*$  and  $\text{DECONS\_ARG\_}^*$  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  $\sigma: \cdot; \cdot; \cdot; \mathcal{R} \vdash (\sigma):(\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}')$ .

PROOF: From  $\langle 2 \rangle 2$  we may deduce

1.  $\mathcal{C}; \mathcal{L}; \Phi \vdash \text{pval}_i \Rightarrow \beta_i$  for each  $x_i:\beta_i \in \mathcal{C}$  or  $x_i:\beta_i \in \mathcal{L}$ .

2.  $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}' \vdash \text{res\_term}_i \Leftarrow \text{res}_i$  for each  $\text{res}_i \in \mathcal{R}'$ .

3.  $\text{smt}(\cdot \Rightarrow \text{term})$  for each  $\text{term} \in \Phi$ .

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

PROOF: By induction on  $\text{arg}$ .

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

⟨1⟩25. CASE:  $\text{TY\_SEQ\_E\_PROC}$ .

PROOF: Similar to  $\text{TY\_SEQ\_E\_CCALL}$ .

⟨1⟩26. CASE:  $\text{TY\_IS\_E\_MEMOP}$ .

PROOF: By induction on  $\text{TY\_MEMOP}^*$  cases.

⟨1⟩27. CASE:  $\text{TY\_IS\_E\_}\{\text{NEG\_}\}\text{ACTION}$ .

PROOF: By induction on  $\text{TY\_ACTION}^*$  cases.

⟨1⟩28. CASE:  $\text{TY\_SEQ\_TE\_LETP}$ .

PROOF SKETCH: Only covering case  $\langle pexpr \rangle \longrightarrow \langle pexpr' \rangle$  here.

See  $\text{TY\_SEQ\_TE\_LET}$  for a more general version and proof for the remaining  $\langle pexpr \rangle \longrightarrow \langle texpr:(y:\beta. term) \rangle$  case.

ASSUME: 1.  $\cdot; \cdot; \cdot \vdash \text{let ident\_or\_pattern} = pexpr \text{ in } texpr \Leftarrow y_2:\beta_2. term_2$ .

2.  $\langle \text{let ident\_or\_pattern} = pexpr \text{ in } texpr \rangle \longrightarrow \langle \text{let ident\_or\_pattern} = pexpr' \text{ in } texpr \rangle$ .

PROVE:  $\cdot; \cdot; \cdot \vdash \text{let ident\_or\_pattern} = pexpr' \text{ in } texpr \Leftarrow y_2:\beta_2. term_2$ .

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

2.  $\text{ident\_or\_pattern}:\beta \rightsquigarrow \mathcal{C}_1 \text{ with } term_1$ .

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

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.  $\cdot; \cdot; \cdot \vdash \text{let ident\_or\_pattern} = pexpr' \text{ in } texpr \Leftarrow y_2:\beta_2. term_2$ .

PROOF: By  $\text{TY\_SEQ\_TE\_LETP}$  using ⟨2⟩1.2,3 and ⟨2⟩3.

⟨1⟩29. CASE:  $\text{TY\_SEQ\_TE\_LETPT}$ .

PROOF: See  $\text{TY\_SEQ\_TE\_LETT}$  for a more general case and proof.

⟨1⟩30. CASE:  $\text{TY\_SEQ\_TE\_LET}$ .

ASSUME: 1.  $\cdot; \cdot; \cdot; \mathcal{R}' \vdash \text{let } \overline{\text{ret\_pattern}_i}^i = seq\_expr \text{ in } texpr_2 \Leftarrow ret_2$ .

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

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

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

2.  $\Phi \vdash \overline{\text{ret\_pattern}_i}^i : ret_1 \rightsquigarrow \mathcal{C}_1; \mathcal{L}_1; \Phi_1; \mathcal{R}_1$ .

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

PROOF: By inversion on 1.

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

PROOF: By inversion on 2.

⟨2⟩3.  $\cdot; \cdot; \cdot; \mathcal{R}' \vdash texpr_1 \Leftarrow ret_1$ .

PROOF: By induction on ⟨2⟩1.1 and ⟨2⟩2.

⟨2⟩4.  $ret_1 = ret'_1$ .  
 PROOF: By cases  $TY\_SEQ\_E\_ \{CCALL, PCALL\}$ .

⟨2⟩5. By  $TY\_SEQ\_TE\_LET$  with ⟨2⟩1.2,3 and ⟨2⟩3, we are done.

⟨1⟩31. CASE:  $TY\_SEQ\_TE\_LETT$ .  
 NOTE:  $h : \mathcal{R}', \mathcal{R}$  and  $h : \mathcal{R}_1, \mathcal{R}$ .

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

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

⟨2⟩2.  $\overline{ret\_pattern_i}^i = \overline{spine\_elem_i}^i \rightsquigarrow \sigma$ .  
 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⟩2 using lemma 5.5 (deconstructing a pattern produces a well-typed substitution).

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

⟨1⟩32. CASE:  $TY\_SEQ\_TE\_LETT$ .  
 ASSUME: 1.  $\cdot; \cdot; \cdot; \mathcal{R}', \mathcal{R} \vdash \text{let } \overline{ret\_pattern_i}^i : ret_1 = texpr_1 \text{ in } texpr_2 \Leftarrow ret_2$ .  
 2.  $\langle h; \text{let } \overline{ret\_pattern_i}^i : ret = texpr_1 \text{ in } texpr_2 \rangle \longrightarrow \langle h'; \text{let } \overline{ret\_pattern_i}^i : ret = texpr'_1 \text{ in } texpr_2 \rangle$ .  
 PROVE:  $\cdot; \cdot; \cdot; \mathcal{R}'', \mathcal{R} \vdash \text{let } \overline{ret\_pattern_i}^i : ret_1 = texpr'_1 \text{ in } texpr_2 \Leftarrow ret_2$ .

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

⟨2⟩2.  $\langle h; texpr_1 \rangle \longrightarrow \langle h'; texpr'_1 \rangle$ .  
 PROOF: By inversion on 2.

⟨2⟩3.  $\cdot; \cdot; \cdot; \mathcal{R}'' \vdash texpr'_1 \Leftarrow ret_1$ .  
 PROOF: By induction on ⟨2⟩1.1 and ⟨2⟩2.

⟨2⟩4. By ⟨2⟩3, ⟨2⟩2.2,3 using  $TY\_SEQ\_TE\_LETT$ , we are done.

⟨1⟩33. CASE:  $TY\_SEQ\_TE\_CASE$ .  
 ASSUME: 1.  $\cdot; \cdot; \cdot; \mathcal{R} \vdash \text{case pval of } \overline{pattern_i \Rightarrow texpr_i}^i \text{ end} \Leftarrow ret$ .  
 2.  $\langle h; \text{case pval of } \overline{pattern_i \Rightarrow texpr_i}^i \text{ end} \rangle \longrightarrow \langle h; \sigma_j(texpr_j) \rangle$ .  
 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.  $\overline{pattern_i; \beta_1}^i \rightsquigarrow \mathcal{C}_i \text{ with } term_i^i$ .  
 3.  $\mathcal{C}_i; \cdot; \cdot, term_i = pval; \mathcal{R} \vdash \overline{texpr_i}^i \Leftarrow ret$ .

PROOF: By inversion on 1.

- ⟨2⟩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.

- ⟨2⟩3.  $term_j = pval$ .

PROOF: By ⟨2⟩1.2 and terms derived from patterns are “equal to” matching values (lemma 5.2).

- ⟨2⟩4.  $\cdot; \cdot; \cdot; \cdot \vdash (\sigma_j)(C_j; \cdot; \cdot, term_j = pval; \cdot)$ .

PROOF: By ⟨2⟩3 and lemma 5.5 (deconstructing a pattern produces a well-typed substitution).

- ⟨2⟩5. By ⟨2⟩4, ⟨2⟩1.3 and substitution lemma 2.5, we are done.

- ⟨1⟩34. CASE: TY\_SEQ\_TE\_IF.

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

ASSUME: 1.  $\cdot; \cdot; \cdot; \mathcal{R} \vdash \text{if True then } texpr_1 \text{ else } texpr_2 \Leftarrow ret$ .

2.  $\langle h; \text{if True then } texpr_1 \text{ else } texpr_2 \rangle \longrightarrow \langle h; texpr_1 \rangle$ .

PROVE:  $\cdot; \cdot; \cdot; \mathcal{R} \vdash texpr_1 \Leftarrow ret$ .

PROOF: Invert 1, note  $\cdot; \cdot; \cdot; \mathcal{R} \vdash (id)(\cdot; \cdot; \cdot, \text{true} = \text{true}; \mathcal{R})$  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

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

## 7 Opsem Judgements

$pure\_opsem\_jtype \quad ::=$   
 $\quad | \quad \langle pexpr \rangle \longrightarrow \langle pexpr' \rangle$   
 $\quad | \quad \langle pexpr \rangle \longrightarrow \langle tpepr:(y:\beta. term) \rangle$   
 $\quad | \quad \langle tpepr \rangle \longrightarrow \langle tpepr' \rangle$

$opsem\_jtype \quad ::=$   
 $\quad | \quad \langle h; seq\_expr \rangle \longrightarrow \langle h'; texpr:ret \rangle$   
 $\quad | \quad \langle h; seq\_texpr \rangle \longrightarrow \langle h'; texpr \rangle$   
 $\quad | \quad \langle h; mem\_op \rangle \longrightarrow \langle h'; tval \rangle$   
 $\quad | \quad \langle h; mem\_action \rangle \longrightarrow \langle h'; tval \rangle$   
 $\quad | \quad \langle h; is\_expr \rangle \longrightarrow \langle h'; is\_expr' \rangle$   
 $\quad | \quad \langle h; is\_texpr \rangle \longrightarrow \langle h'; texpr \rangle$   
 $\quad | \quad \langle h; texpr \rangle \longrightarrow \langle h'; texpr' \rangle$