Soundness proof of type checking

Dhruv C. Makwana

B1 Commentary

Since Core is a first-order language, and we require that all functions and labels are annotated with the correct type, it suffices to only use purely syntactic techniques to prove soundness. This remains true despite the addition of linear types, systems with which are normally proved using logical relations . There are three main components to this: a joint progress-and-type-preservation proof for resource term reduction, a progress theorem and a type-preservation theorem.

Let a resource be called *normalised* if it is either a *pred*, *qpred* or an under-determined conditional resource. Let a resource context $\underline{\mathcal{R}}$ be called normalised if it contains only normalised resources. Even though the grammar of resources is richer, we can, in all the proofs relating to well-typed closed resource terms, are assume the resource context to be normalised. This is fine because of the following lemma: if a well-typed resource term is closed, then the context in which it is well-typed must normalised.

Operational semantics for resource term happens to be defined using big-step style; this makes its definition concise and modular, at the cost of making the proof of soundness of resource term reduction more complicated since it requires joint progress and type preservation. The configuration for the operational semantics is a pair of a heap and an annotated and let-normalised Core program. Heaps only contain normalised resources. Predicates in heaps are optionally tagged with their "definition" *def* (a resource value of the type of the predicate body) and a sub-heap (of the resources used by the definition). This is to support folding and unfolding predicates in the operational semantics, and to capture the idea that predicate encapsulate their contents until opened.

$$\frac{\mathit{pred} \equiv \mathit{pred\_term}(\,\mathit{oarg}\,) \qquad \langle h + h'; \mathit{res\_pat} = \mathit{def}\rangle \rightsquigarrow \langle h''; \sigma\,\rangle}{\langle h + \{\mathit{pred}\ \&\ \mathit{def}\ \&\ h'\}; \mathtt{fold}\,(\mathit{res\_pat}) = \mathit{pred\_term}\rangle \rightsquigarrow \langle h''; \sigma\,\rangle}$$

$$\frac{\mathit{pred} \equiv \mathit{pred\_term}(\,\mathit{oarg}\,) \quad \langle h; \mathit{res\_term}\rangle \Downarrow \langle h_1; \mathit{def}\rangle \quad \mathtt{footprint\_of}\ \mathit{def}\ \mathtt{in}\ h_1 \rightsquigarrow h_2\ \mathrm{rem}\ h_3}{\langle h; \mathtt{fold}\,\mathit{res\_term}{:}\mathit{pred}\rangle \Downarrow \langle h_3 + \{\mathit{pred}\ \&\ \mathit{def}\ \&\ h_2\}; \mathit{pred\_term}\rangle}$$

The types of heaps are normalised contexts; the rules for these are straightforward, except the fact a heap with a folded predicate requires there exists a context for which the resource value *def* and *heap* is well-typed. This becomes necessary for proving the progress of pattern-matching for the whole of the annotated and let-normalised Core.

Theorem 1 (Progress and type preservation for resource terms) *For all closed resource terms (res_term) which type check or synthesise* $(\cdot; \cdot; \Phi; \underline{\mathcal{R}} \vdash \mathit{res\_term} \Leftarrow \mathit{res})$ *and all well-typed heaps* $(\Phi \vdash h \Leftarrow \underline{\mathcal{R}})$ *there exists a resource value (res_val), context* $(\underline{\mathcal{R}}')$ *and heap* (h')*, such that: the value is well-typed* $(\cdot; \cdot; \Phi; \underline{\mathcal{R}}' \vdash \mathit{res\_val} \Leftarrow \mathit{res})$*; the heap is well-typed* $(\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}')$*; for all frame-heaps* (f)*, the resource term reduces to the resource value without affecting the frame-heap* $(\langle h + f; \mathit{res\_term}\rangle \Downarrow \langle h' + f; \mathit{res\_val}\rangle)$*.*

The interesting case in the proof of this is folding a predicate; proving this case requires a notion of *footprint* of a resource value: the subheap containing the resources referred to by the value.

Theorem 2 (Progress for the annotated and let-normalised Core) *If a top-level expression (texpr) is well-typed* $(\cdot; \cdot; \Phi; \underline{\mathcal{R}} \vdash \mathit{texpr} \Leftarrow \mathit{ret})$ *and all computational patterns in it are exhaustive, then either it is a value (tval), or it is unreachable, or for all heaps* (h)*, if the heap is well-typed* $(\Phi \vdash h \Leftarrow \underline{\mathcal{R}})$ *then there exists another heap* (h') *and expression* (texpr') *which is stepped to* $(\langle h; \mathit{texpr}\rangle \longrightarrow \langle h'; \mathit{texpr}'\rangle)$ *in the operational semantics.*

The assumption that all computational patterns are exhaustive is justified because they are generated by Cerberus. As one might expect, proving progress requires well-typed patterns successfully produce substitutions. However, this complicated by two things, the solution to which requires the introduction of a relation on SMT terms and resource types, $\Phi \vdash \mathit{res} \sim \mathit{res}'$ (to be read "under constraints Φ , res is related to res'").

The first is that the constraint term generated when typing a computational pattern (this is required to record, in the constraint context, which branch the type system is assuming it is in) is not exactly equal to the values it can match in the operational semantics (nor would we want it to be: the pattern $\texttt{Cons}(x_1, x_2)$ should match the value $\texttt{Cons}(\mathit{pval}_1, \texttt{Cons}(\mathit{pval}_{21}, \texttt{Nil}\,\beta(\,))))$. Hence, we must weaken the notion of equality on types to $\sim$ relatedness, which links the two, so that during the proof can substitute the constraint term $x_1 :: x_2$ at the type-level, and maintain a link to the corresponding value. The second is that the conditions of related conditional resource must remain SMT-equivalent (with reference to a constraint context), so that pattern-match typing and resource term typing are consistent.

Theorem 3 (Type preservation for the annotated and let-normalised Core) *For all closed and well-typed top-level expressions ($\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \mathit{texpr} \Leftarrow \mathit{ret}$), well typed heaps ($\Phi \vdash h \Leftarrow \underline{\mathcal{R}}$), frame-heaps ($f$), new heaps ($\mathit{heap}$), and new top-level expressions (texpr'), which are connected by a step in the operational semantics ($\langle h + f; \mathit{texpr}\rangle \longrightarrow \langle \mathit{heap}; \mathit{texpr}'\rangle$), if all top-level functions are annotated correctly, there exists a constraint context (Φ'), sub-heap (h'), and resource context ($\underline{\mathcal{R}}'$), such that the constraint context is Φ extended, the frame is unaffected ($\mathit{heap} = h' + f$), the sub-heap is well-typed ($\Phi' \vdash h' \Leftarrow \underline{\mathcal{R}}'$), and the top-level expression too ($\cdot;\cdot;\Phi';\underline{\mathcal{R}}' \vdash \mathit{texpr}' \Leftarrow \mathit{ret}$).*

A few things are noteworthy about the proof. First is that a frame-heap has to be explicitly passed around. Whilst this is inconvenient, it becomes necessary in the EXPL_TOP_SEQ_LETT case. The next is that proof that well-typed spines produce well-typed substitutions require quantifying over the substitutions done *so far*, so that the inductive case matches up *and* the substitution so far ψ shows up in the conclusion, 'closing' otherwise 'open' substitution and terms. One more is in the proof that well-typed patterns produce well-typed substitutions: unfortunately quantifying over the substitutions done so far is not helpful because *even the substitution itself* can be 'open' (refer to free variables). Hence the peculiar typing of pattern-matching, so that all terms are well-scoped. This allows us to induct usefully, and get the required substitution in the output substitution and its type, making way to apply the substitution lemma afterwards. Lastly, we gather constraints throughout the proof, since these are accumulated by the typing rules, during pattern- matching, case and if. Given the constraint context is always well-formed (w.r.t. to the empty contexts), this means that all the constraints must be trivial (though extra effort would be required to show that they are trivially true, for example, showing that `default bool` cannot occur.

B2 Typing Judgements

In this document, $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}\vdash J$ stands for all *defined* judgements, listed in the remainder of this section after this paragraph. In particular, it does not stand for $\mathcal{C}\vdash \mathit{mem\_val}\Rightarrow \beta$ or $\mathcal{C};\mathcal{L}\vdash \mathit{term}\Rightarrow \beta$. Furthermore, I assume that lemma B6 (Weakening) and lemma B7.3 (Substitution) (proven for the *defined* judgements in the referenced sections) hold for these ($\mathcal{C}\vdash \mathit{mem\_val}\Rightarrow \beta$, and $\mathcal{C};\mathcal{L}\vdash \mathit{term}\Rightarrow \beta$) judgements.

res_judge ::=

| $\Phi\vdash \texttt{cmp\_min}\,(\mathit{iguard},\mathit{iguard}')\rightsquigarrow \mathit{opt\_cmp\_term}$
given constraints Φ , *iguard* is potentially included in *iguard′* (or vice-versa) with ordering and minimum *opt_cmp_term*

| $\Phi\vdash \mathit{qpred\_term}\sqsubseteq ?\ \mathit{qpred\_term}'\rightsquigarrow \mathit{opt\_cmp}$
given constraints Φ , *qpred_term* is potentially included in *qpred_term′* (or vice-versa) with ordering *opt_cmp*

| $\Phi\vdash \mathit{res\_req}\ \equiv\ \mathit{res\_req}'\rightsquigarrow \mathit{bool}$
resource equality: given constraints Φ, *res_req* and *res_req′* are equal according to *bool*

| $\Phi\vdash \mathit{res}\ \equiv\ \mathit{res}'$
resource equality: given constraints Φ, *res* is equal to *res′*

| $\Phi\vdash \texttt{simp\_rec}\,(\mathit{res})\rightsquigarrow \mathit{res}',\mathit{bool}$
partial-simplification of resources: given constraints Φ, *res* partially simplifies (strips ifs) to *res′*

| $\Phi\vdash \texttt{simp}\,(\mathit{res})\rightsquigarrow \mathit{opt\_res}$
partial-simplification of resources: given constraints Φ, *res* attempts a partial simplification (strips ifs) to *opt_res*

ret_judge ::=

| $\Phi\vdash \mathit{ret}\ \equiv\ \mathit{ret}'$
return type equality: given constraints Φ, *ret* is equal to *ret′*

pat_judge ::=

| $\mathit{pat}{:}\beta\rightsquigarrow\ \mathcal{C}\ \texttt{with}\ \mathit{term}$
computational pattern to context: *pat* and type β produces context $\mathcal{C}$ and constraint *term*

| $\mathit{ident\_or\_pat}{:}\beta\rightsquigarrow \mathcal{C}\ \texttt{with}\ \mathit{term}$
identifier-or-pattern to context: *ident_or_pat* and type β produces context $\mathcal{C}$ and constraint *term*

	$\|$	$\mathcal{L};\Phi \vdash \mathit{res\_pat}{:}\mathit{res} \rightsquigarrow \mathcal{L}';\Phi';\mathcal{R}'$ resources pattern to context: given constraints Φ, *res_pat* of type *res* produces contexts $\mathcal{L}';\Phi';\mathcal{R}'$
	$\|$	$\mathcal{C};\mathcal{L};\Phi \vdash \mathit{ret\_pat}{:}\mathit{ret} \rightsquigarrow \mathcal{C}';\mathcal{L}';\Phi';\mathcal{R}'$ return pattern to context: given context $\mathcal{C};\mathcal{L};\Phi$, *ret_pat* and return type *ret* produces contexts $\mathcal{C}';\mathcal{L}';\Phi';\mathcal{R}'$
	$\|$	$\Phi \vdash \mathit{ret\_pat}{:}\mathit{ret} \rightsquigarrow \mathcal{C}';\mathcal{L}';\Phi';\mathcal{R}'$ return pattern to context: given constraints Φ, *ret_pat* and return type *ret* produces contexts $\mathcal{C}';\mathcal{L}';\Phi';\mathcal{R}'$
expl_pure	$::=$	
	$\|$	$\mathcal{C} \vdash \mathit{object\_value} \Rightarrow \beta$ object value synthesises: given $\mathcal{C}$, *object_value* synthesises type β
	$\|$	$\mathcal{C} \vdash \mathit{pval} \Rightarrow \beta$ pure value synthesises: given $\mathcal{C}$, *pval* synthesises type β
	$\|$	$\mathcal{C};\mathcal{L};\Phi \vdash \mathit{pexpr} \Rightarrow \mathit{pure\_ret}$ pure expression synthesises: given $\mathcal{C};\mathcal{L};\Phi$, *pexpr* synthesises a pure (non-resourceful) return type *pure_ret*
	$\|$	$\mathcal{C};\mathcal{L};\Phi \vdash \mathit{tpval} \Leftarrow \mathit{pure\_ret}$ pure top-level value checks: given $\mathcal{C};\mathcal{L};\Phi$, *tpval* checks against *pure_ret*
	$\|$	$\mathcal{C};\mathcal{L};\Phi \vdash \mathit{tpexpr} \Leftarrow \mathit{pure\_ret}$ pure top-level expression checks: given $\mathcal{C};\mathcal{L};\Phi$, *tpexpr* checks against *pure_ret*
expl_res	$::=$	
	$\|$	$\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{pred\_ops} \Rightarrow \mathit{res}$ resource (q)predicate operation term synthesis: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, *pred_ops* synthesises resource *res*
	$\|$	$\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{res\_term} \Rightarrow \mathit{res}$ resource term synthesises: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, *res_term* synthesises resource *res*
	$\|$	$\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{res\_term} \Leftarrow \mathit{res}$ resource term checks: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, *res_term* checks against resource *res*
expl_spine	$::=$	
	$\|$	$\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{spine} :: \mathit{fun} \gg \mathit{ret}$ function call spine checks: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, compatible *spine*, *fun* produces an *ret*

expl_is_expr	::=	
	|	$\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{action} \Rightarrow \mathit{ret}$ memory action synthesises: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, *action* synthesises return type *ret*
	|	$\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{memop} \Rightarrow \mathit{ret}$ memory operation synthesises: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, *memop* synthesises return type *ret*
	|	$\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{is\_expr} \Rightarrow \mathit{ret}$ indet. seq. expression synthesises: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, *is_expr* synthesises return type *ret*
expl_seq_expr	::=	
	|	$\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{seq\_expr} \Rightarrow \mathit{ret}$ seq. expression synthesises: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, *seq_expr* synthesises return type *ret*
expl_top	::=	
	|	$\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{tval} \Leftarrow \mathit{ret}$ top-level value checks: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, *tval* checks against return type *ret*
	|	$\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{seq\_texpr} \Leftarrow \mathit{ret}$ top-level seq. expression checks: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, *seq_texpr* checks against return type *ret*
	|	$\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{is\_texpr} \Leftarrow \mathit{ret}$ top-level indet. seq. expression checks: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, *is_texpr* checks against return type *ret*
	|	$\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{texpr} \Leftarrow \mathit{ret}$ top-level expression checks: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, *texpr* checks against return type *ret*

B3 Operational Semantics Judgements

subs_judge ::=

| $pat = pval \rightsquigarrow \sigma$
computational value deconstruction: *pat* deconstructs *pval* to produce substitution σ

| $ident\_or\_pat = pval \rightsquigarrow \sigma$
computational value deconstruction: ident_or_pat deconstructs *pval* to produce substitution σ

| $\langle h; res\_pat = res\_val\rangle \rightsquigarrow \langle h'; \sigma\rangle$
resource term deconstruction: *res_pat* deconstructs *res_val* to produce substitution σ

| $\langle h; \overline{ret\_pat_i = ret\_term_i}^{\,i}\rangle \rightsquigarrow \langle h'; \sigma\rangle$
return value deconstruction: ret_pat_i deconstructs ret_val_i to produce substitution σ

| $\langle h; \overline{x_i = spine\_elem_i}^{\,i}\rangle :: fun \gg \langle h'; \sigma; ret\rangle$
function call spine: heap h and formal parameters x_i assigned to $spine\_elem_i$ for function of type *fun*, produce new heap h' substitution σ and result type *ret*

pure_opsem_defns ::=

| $\langle pexpr\rangle \longrightarrow \langle tpexpr{:}pure\_ret\rangle$

| $\langle tpexpr\rangle \longrightarrow \langle tpexpr'\rangle$

opsem_defns ::=

| $\langle h; pred\_ops\rangle \Downarrow \langle h'; res\_val\rangle$
big-step resource (q)points-to operation reduction: $\langle h; pred\_ops\rangle$ reduces to $\langle h'; res\_val\rangle$

| $\texttt{footprint\_of}\ res\_val\ \texttt{in}\ h \rightsquigarrow h_1\ \text{rem}\ h_2$
footprint of *res_val* in heap h is h_1 with h_2 remainder/frame

| $\langle h; res\_term\rangle \Downarrow \langle h'; res\_val\rangle$
big-step resource term reduction: $\langle h; res\_term\rangle$ reduces to $\langle h'; res\_val\rangle$

| $\langle h; action\rangle \longrightarrow \langle h'; is\_expr\rangle$

| $\langle h; memop\rangle \longrightarrow \langle h'; is\_expr\rangle$

| $\langle h; is\_expr\rangle \longrightarrow \langle h'; is\_expr'\rangle$

| $\langle h; seq\_expr\rangle \longrightarrow \langle h'; texpr{:}ret\rangle$

| $\langle h; seq\_texpr\rangle \longrightarrow \langle h'; texpr\rangle$

| $\langle h; is\_texpr\rangle \longrightarrow \langle h'; texpr\rangle$

| $\langle h; texpr\rangle \longrightarrow \langle h'; texpr'\rangle$

B4 Proof Judgements

Note that the definition of $term \sim term'$ is omitted/assumed. It simply means that $term$ and $term'$ can be unified. Informally, $term \sim term'$ are defined recursively over the structure of SMT terms, using the standard definition of unification: variables unify with anything (modulo an occurs check), atoms unify if they are identical, compound terms unify if their constructors (except for `Specified`) and arity are identical, and their arguments unify recursively.

To clarify the `Specified` exception: $term \sim \texttt{Specified}(pval)$ (and $\texttt{Specified}(pval) \sim term$) iff $term \sim pval$.

$\sim$ is additionally assumed to be an equivalence relation and preserved by substitution: if $term \sim term'$ and $x \sim y$ in $term_1 \sim term_1'$ then $term/x(term_1) \sim term'/y(term_1')$.

Note: $\sim$ is *only* used in the *proof* of soundness, and *not* in the *explicit CN type system*. There is no unification required in the type system, but the notion of related terms is required to argue for the soundness of pattern-matching (Section B9.4 Well-typed values pattern-match successfully).

misc_extra	::=	extra judgements for proof-related definitions
	|	$\forall\, x.\ \mathit{iguard} \Rightarrow \mathcal{C}; \mathcal{L}; \Phi \vdash h \Leftarrow \underline{\mathcal{R}}$ meta-logical quantification over heap-typing
	|	$\forall\, \mathit{term} \sim \mathit{term}'.\ \Phi \vdash \mathit{fun} \sim \mathit{ret}$ meta-logical quantification over related *fun* and *ret*
	|	$\forall\, \mathit{term} \sim \mathit{term}'.\ \Phi \vdash \mathit{res} \sim \mathit{res}'$ meta-logical quantification over related *res* and res'
	|	$\mathit{term} \sim \mathit{term}'$ omitted/assumed defintion: SMT terms *term* and term' are related
proof_defns	::=	
	|	$\overline{x_i}^{\,i} :: \mathit{fun} \rightsquigarrow \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \mid \mathit{ret}$ matching $\overline{x_i}^{\,i}$ and *fun* produces contexts $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}$ and return type *ret*
	|	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \sqsubseteq \mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'$ context weakening: $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}$ is stronger than $\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'$
	|	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma \Leftarrow (\mathcal{C}; \mathcal{L}; \mathcal{R})$ well-typed substitution: given $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}$, σ checks against type $(\mathcal{C}; \mathcal{L}; \mathcal{R})$. It is complicated by the fact that substitutions are assumed to be sequential/telescoping.
	|	$\mathcal{C}; \mathcal{L}; \Phi \vdash h \Leftarrow \underline{\mathcal{R}}$ heap typing: under context $\mathcal{C}; \mathcal{L}; \Phi$, heap h checks against context/type $\underline{\mathcal{R}}$
	|	$\Phi \vdash h \Leftarrow \underline{\mathcal{R}}$ heap typing: under context Φ, heap h checks against context/type $\underline{\mathcal{R}}$
	|	$\Phi \vdash \mathit{res} \sim \mathit{res}'$ *res* is related to res'
	|	$\Phi \vdash \mathit{fun} \sim \mathit{ret}$ *fun* is related to *ret*

B5 Groups of Rules

B5.1 Typing rules with an smt $(\Phi \Rightarrow qterm)$ premise

IG_Cmp_Eq, IG_Cmp_Lt, IG_Cmp_Gt, Q_Cmp_PtrStep_Neq, Q_Cmp_IG_Neq, Q_Cmp_IArg_Neq, Q_Cmp_Comparable, Req_Eq_PP_IArg_Neq, Req_Eq_PP_Eq, Res_Eq_Phi, Res_Eq_OrdDisj, Res_SimpRec_If_True, Res_SimpRec_If_False, Ret_Eq_Phi, Pat_Res_Match_If_True, Pat_Res_Match_If_False, Pure_Expr_Assert_Undef, Pure_Top_Val_Done, Pure_Top_Val_Undef, Pure_Top_Val_Error, Res_Syn_PredOps_Congeal, Res_Syn_PredOps_Implode, Res_Syn_PredOps_Break, Res_Syn_PredOps_Glue, Res_Syn_PredOps_Inj, Res_Syn_PredOps_Split, Res_Chk_Phi, Res_Chk_If_True, Res_Chk_If_False, Expl_Is_Action_Load, Expl_Is_Action_Store, Expl_Is_Action_Kill_Static, Expl_Is_Memop_PtrValidForDeref, Expl_Top_Val_Undef, Expl_Top_Val_Error.

B5.2 Typing rules which change the context

B5.2.1 Rules which add constraints

Expl_Top_Seq_If.

B5.2.2 Rules which add constraints and computational or logical variables

Expl_Top_Seq_LetP, Expl_Top_Seq_LetTP, Expl_Top_Seq_Case.

B5.2.3 Rules which restrict the resource context

No-resource / "pure" rules: IG_Cmp_Eq, IG_Cmp_Lt, IG_Cmp_Gt, IG_Cmp_None, Q_Cmp_Name_Neq, Q_Cmp_PtrStep_Neq, Q_Cmp_IG_Neq, Q_Cmp_IArg_Neq, Q_Cmp_Comparable, Req_Eq_PP_Name_Neq, Req_Eq_PP_IArg_Neq, Req_Eq_PP_Eq, Req_Eq_QQ_Eq, Req_Eq_QQ_Neq, Res_Eq_Emp, Res_Eq_Phi, Res_Eq_Pred, Res_Eq_QPred, Res_Eq_SepConj, Res_Eq_Exists, Res_Eq_OrdDisj, Res_SimpRec_If_True, Res_SimpRec_If_False, Res_SimpRec_SepConj, Res_SimpRec_Exists, Res_SimpRec_NoChange, Simp_NoSimp, Simp_Simp, Ret_Eq_End, Ret_Eq_Comp, Ret_Eq_Log, Ret_Eq_Phi, Ret_Eq_Res, Pat_Comp_No_Sym_Annot, Pat_Comp_Sym_Annot, Pat_Comp_Nil, Pat_Comp_Cons, Pat_Comp_Tuple, Pat_Comp_Array, Pat_Comp_Specified, Pat_Sym_Or_Pat_Sym, Pat_Sym_Or_Pat_Pat, Pat_Res_Match_Emp, Pat_Res_Match_Phi, Pat_Res_Match_If_True, Pat_Res_Match_If_False, Pat_Res_Match_Var, Pat_Res_Match_SepConj, Pat_Res_Match_Pack, Pat_Res_Match_Fold, Pat_Ret_Empty, Pat_Ret_Comp, Pat_Ret_Log, Pat_Ret_Res, Pat_Ret_Phi, Pat_Ret'_Aux, Pure_Val_Obj_Int, Pure_Val_Obj_Ptr, Pure_Val_Obj_Arr, Pure_Val_Obj_Struct, Pure_Val_Var, Pure_Val_Obj, Pure_Val_Loaded, Pure_Val_Unit, Pure_Val_True, Pure_Val_False, Pure_Val_List, Pure_Val_Tuple, Pure_Val_Ctor_Nil, Pure_Val_Ctor_Cons, Pure_Val_Ctor_Tuple, Pure_Val_Ctor_Array, Pure_Val_Ctor_Specified, Pure_Val_Struct, Pure_Expr_Val, Pure_Expr_Array_Shift, Pure_Expr_Member_Shift, Pure_Expr_Not, Pure_Expr_Arith_Binop, Pure_Expr_Rel_Binop, Pure_Expr_Bool_Binop, Pure_Expr_Call, Pure_Expr_Assert_Undef, Pure_Expr_Bool_To_Integer, Pure_Expr_WrapI, Pure_Top_Val_Undef, Pure_Top_Val_Error, Pure_Top_Val_Done.

Resource-mentioning rules: Res_Syn_Emp, Res_Syn_Var, Res_Syn_VarSimp, Res_Syn_Pred, Res_Syn_QPred, Res_Syn_SepConj, Res_Chk_Phi, Res_Chk_SepConj, Expl_Spine_Ret, Expl_Spine_Res, Expl_Is_Action_Create, Expl_Is_Memop_Rel_Binop, Expl_Is_Memop_IntFromPtr, Expl_Is_Memop_PtrFromInt, Expl_Is_Memop_PtrValidForDeref, Expl_Is_Memop_PtrWellAligned, Expl_Is_Memop_IntFromPtr, Expl_Top_Val_Undef, Expl_Top_Val_Error, Expl_Top_Seq_Run, Subs_Chk_Empty, Subs_Chk_Res.

B5.2.4 Rules which add constraints and restrict the resource context

Pure_Top_If.

B5.2.5 Rules which add constraints and variables, and restrict the resource context

Pure_Top_Let, Pure_Top_LetT, Pure_Top_Case, Expl_Top_Seq_Let, Expl_Top_Seq_LetT, Expl_Top_Is_LetS.

B5.3 Value typing rules

Pure_Val_Obj_Int, Pure_Val_Obj_Ptr, Pure_Val_Obj_Arr, Pure_Val_Obj_Struct, Pure_Val_Var, Pure_Val_Obj, Pure_Val_Loaded, Pure_Val_Unit, Pure_Val_True, Pure_Val_False, Pure_Val_Tuple, Pure_Val_Ctor_Nil, Pure_Val_Ctor_Cons, Pure_Val_Ctor_Tuple, Pure_Val_Ctor_Array, Pure_Val_Ctor_Specified, Pure_Val_Struct, Pure_Top_Val_Done, Res_Syn_Emp, Res_Syn_Var, Res_Syn_VarSimp, Res_Syn_Pred, Res_Syn_QPred, Res_Chk_Phi, Expl_Top_Val_Done.

B6 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: $\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}' \vdash J$.

$\langle 1 \rangle 1$. CASE: PURE_VAL_VAR.
PROOF: By WEAK_CONS_COMP, if $x{:}\beta \in \mathcal{C}$ then $x{:}\beta \in \mathcal{C}'$.

$\langle 1 \rangle 2$. CASE: Typing rules with an `smt` $(\Phi \Rightarrow \mathit{qterm})$ premise (see B5.1).

ASSUME: `smt` $(\Phi \Rightarrow \mathit{qterm})$.
PROVE: `smt` $(\Phi' \Rightarrow \mathit{qterm})$.

$\langle 2 \rangle 1$. For all *term*, if $\mathit{term} \in \Phi$ then $\mathit{term} \in \Phi'$.
PROOF: By WEAK_CONS_PHI.

$\langle 2 \rangle 2$. Any extra constraints in Φ' (by WEAK_SKIP_PHI) would either be irrelevant, redundant, or inconsistent.

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

$\langle 1 \rangle 3$. CASE: All remaining rules.

$\langle 2 \rangle 1$. $\mathcal{R} = \mathcal{R}'$.
PROOF: Only WEAK_CONS_RES exists.

$\langle 2 \rangle 2$. All remaining rules are functorial in $\mathcal{C}; \mathcal{L}; \Phi$, so one can proceed by straightforward induction.

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

B7 Substitution

B7.1 Substitutions preserve SMT results

If $\texttt{smt}\,(\Phi \Rightarrow \mathit{qterm})$ and $\mathcal{C};\mathcal{L};\sigma(\Phi);\mathcal{R} \vdash \sigma \Leftarrow (\mathcal{C}';\mathcal{L}';\mathcal{R}')$, then $\texttt{smt}\,(\sigma(\Phi) \Rightarrow \sigma(\mathit{qterm}))$.

Proof: By the first assumption, *qterm* holds for all (well-typed, ensured by the second assumption) instantiations of its free variables.

B7.2 Substitutions can be split up

If $\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \sigma \Leftarrow (\mathcal{C}';\mathcal{L}';\mathcal{R}'_1,\mathcal{R}'_2)$ then
$\exists \mathcal{R}_1,\mathcal{R}_2,\sigma_1,\sigma_2.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R}_1 \vdash \sigma_1 \Leftarrow (\mathcal{C}';\mathcal{L}';\mathcal{R}'_1) \wedge \mathcal{C};\mathcal{L};\Phi;\mathcal{R}_2 \vdash \sigma_2 \Leftarrow (\mathcal{C}';\mathcal{L}';\mathcal{R}'_2)$.

Proof sketch: By induction on the substitution. If $\sigma = [\mathit{res\_term}/r, \sigma']$ where $r{:}\mathit{res}$:

$\langle 1\rangle 1.$ Let $\mathcal{R}'$ be such that $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}' \vdash \mathit{res\_term} \Leftarrow \mathit{res}$.

$\langle 1\rangle 2.$ Recursively split σ' into σ'_1 and $\mathcal{R}''_1$; σ'_2 and $\mathcal{R}''_2$.

$\langle 1\rangle 3.$ If $r \in \mathcal{R}'_1$, let $\sigma_1 = [\mathit{res\_term}/r, \sigma'_1]$ and $\mathcal{R}_1 = \mathcal{R}', \mathcal{R}''_1$.

$\langle 1\rangle 4.$ If $r \in \mathcal{R}'_2$, let $\sigma_2 = [\mathit{res\_term}/r, \sigma'_2]$.

$\langle 1\rangle 5.$ For other cases, both are treated exactly the same.

B7.3 Substitution

If $\mathcal{C}';\mathcal{L}';\Phi;\mathcal{R}' \vdash J$, then $\forall\, \mathcal{C},\mathcal{L},\mathcal{R},\sigma.\ (\mathcal{C};\mathcal{L};\sigma(\Phi);\mathcal{R} \vdash \sigma \Leftarrow (\mathcal{C}';\mathcal{L}';\mathcal{R}')) \Rightarrow \mathcal{C};\mathcal{L};\sigma(\Phi);\mathcal{R} \vdash \sigma(J)$.

For types, substitutions are only defined over resource types *res*, and return types *res*, not base types β. Similarly, for terms, substitutions are only defined over expressions (including SMT terms *term*), but not (computational, resource or return) patterns.

Since Φ is scoped to $\mathcal{C}';\mathcal{L}'$, we must substitute over it as well as all the usual suspects on the right.

Substitution of contexts is defined by substituting over each constraint in Φ. As a result, $\sigma(\Phi_1,\Phi_2) = \sigma(\Phi_1),\sigma(\Phi_2)$, and if $\sigma(\Phi) = \Phi'_1,\Phi'_2$ then $\exists \Phi_1,\Phi_2.\ \sigma(\Phi_1,\Phi_2) = \sigma(\Phi_1),\sigma(\Phi_2)$.

Proof sketch: Induction over the typing judgements.

1. Variable rules: Pure_Val_Var, Res_Syn_VarSimp, Res_Syn_Var.
2. Expl_Top_Val_Done: prove that `to_fun` commutes with substitution.
3. Typing rules which change the context (see B5.2).
4. Remaining rules by straightforward induction.

Assume: 1. $\mathcal{C}';\mathcal{L}';\Phi;\mathcal{R}' \vdash J$.
2. Arbitrary $\mathcal{C},\mathcal{L},\mathcal{R},\sigma$.
3. $\mathcal{C};\mathcal{L};\sigma(\Phi);\mathcal{R} \vdash \sigma \Leftarrow (\mathcal{C}';\mathcal{L}';\mathcal{R}')$.

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

$\langle 1 \rangle 1.$ CASE: PURE_VAL_VAR.
$\mathcal{C}'; \mathcal{L}' \vdash x \Rightarrow \beta$
$\mathcal{C}; \mathcal{L}; \sigma(\Phi); \mathcal{R} \vdash \sigma \Leftarrow (\mathcal{C}'; \mathcal{L}'; \cdot)$.

$\langle 2 \rangle 1.$ $x{:}\beta \in \mathcal{C}'$.
PROOF: By inversion on assumption 1.

$\langle 2 \rangle 2.$ $\mathcal{R}$ is empty.
PROOF: SUBS_CHK_RES is the only rule which could require a non-empty resource context, and it is never used because $\mathcal{R}'$ is empty.

$\langle 2 \rangle 3.$ $\exists \sigma_1, \mathit{pval}, \sigma_2, \beta, \mathcal{C}_1, \mathcal{C}_2, \mathcal{L}_1, \mathcal{L}_2$.
1. $\sigma = [\sigma_1, \mathit{pval}/x, \sigma_2]$
2. $\mathcal{C}; \mathcal{L}; \Phi; \cdot \vdash \sigma_1 \Leftarrow (\mathcal{C}_1; \mathcal{L}_1; \cdot)$
3. $\mathcal{C}; \mathcal{L}; \Phi; \cdot \vdash \sigma_1(\mathit{pval}/x) \Leftarrow (x{:}\beta; \cdot; \cdot)$
4. $\mathcal{C} \vdash \sigma_1(\mathit{pval}) \Rightarrow \beta$
5. $\mathcal{C}; \mathcal{L}; \Phi; \cdot \vdash \sigma_1(\mathit{pval}/x(\sigma_2)) \Leftarrow (\mathcal{C}_2; \mathcal{L}_2; \cdot)$.

PROOF: Repeated inversion on assumption 3 until the SUBS_CHK_COMP responsible for adding x (by $\langle 2 \rangle 1$, there must be at least one).

$\langle 2 \rangle 4.$ Since $\sigma(x) = \sigma_1(\mathit{pval})$, we are done.
PROOF: By $\mathcal{C}; \mathcal{L} \vdash \sigma(x) \Rightarrow \beta$.

$\langle 1 \rangle 2.$ CASE: RES_SYN_VARSIMP.
$\mathcal{C}'; \mathcal{L}'; \Phi; r{:}\mathit{res} \vdash r \Rightarrow \mathit{res}'$
$\mathcal{C}; \mathcal{L}; \sigma(\Phi); \mathcal{R} \vdash \sigma \Leftarrow (\mathcal{C}'; \mathcal{L}'; r{:}\mathit{res}')$.

$\langle 2 \rangle 1.$ $\exists \sigma_1, \mathit{res\_term}, \sigma_2, \mathcal{C}_1, \mathcal{C}_2, \mathcal{L}_1, \mathcal{L}_2$.
1. $\sigma = [\sigma_1, \mathit{res\_term}/r, \sigma_2]$
2. $\mathcal{C}; \mathcal{L}; \sigma(\Phi); \cdot \vdash \sigma_1 \Leftarrow (\mathcal{C}_1; \mathcal{L}_1; \cdot)$
3. $\cdot; \cdot; \sigma(\Phi); \mathcal{R} \vdash \sigma_1(\mathit{res\_term}/r) \Leftarrow (\cdot; \cdot; r{:}\sigma_1(\mathit{res}'))$
4. $\mathcal{C}; \mathcal{L}; \sigma(\Phi); \mathcal{R} \vdash \sigma_1(\mathit{res\_term}) \Leftarrow \sigma_1(\mathit{res}')$
5. $\mathcal{C}; \mathcal{L}; \sigma(\Phi); \cdot \vdash \sigma_1(\mathit{res\_term}/r(\sigma_2)) \Leftarrow (\mathcal{C}_2; \mathcal{L}_2; \cdot)$.

PROOF: Repeated inversion on assumption 3 until the SUBS_CHK_RES responsible for adding r (there must be exactly one).

$\langle 2 \rangle 2.$ SUFFICES: 1. $\sigma(r) = \sigma_1(\mathit{res\_term})$
2. $\sigma(\mathit{res}') = [\sigma_1, \mathit{res\_term}/r, \sigma_2](\mathit{res}') = \sigma_1(\mathit{res}')$.

$\langle 2 \rangle 3.$ $\sigma(r) = \sigma_1(\mathit{res\_term})$.
PROOF: $\sigma_2(r) = r$, because σ_2 does not mention any resource variables.

$\langle 2 \rangle 4.$ $\sigma(\mathit{res}') = [\sigma_1, \mathit{res\_term}/r, \sigma_2](\mathit{res}') = \sigma_1(\mathit{res}')$.

$\langle 3 \rangle 1.$ $[\sigma_1, \mathit{res\_term}/r, \sigma_2](\mathit{res}') = [\sigma_1, \sigma_2](\mathit{res}')$.
PROOF: Resource types do not refer to resource variables.

$\langle 3 \rangle 2.$ $[\sigma_1, \sigma_2](\mathit{res}') = \sigma_1(\mathit{res})$.
PROOF: By $\cdot; \cdot; \sigma(\Phi); \mathcal{R} \vdash \sigma_1(\mathit{res\_term}) \Leftarrow \sigma_1(\mathit{res}')$, we know that res' only refers to variables in $\mathcal{C}, \mathcal{C}_1; \mathcal{L}, \mathcal{L}_1$.

$\langle 1 \rangle 3.$ CASE: RES_SYN_VAR.
$\mathcal{C}'; \mathcal{L}'; \Phi'; r{:}\mathit{res} \vdash r \Rightarrow \mathit{res}$
PROOF: Similar to RES_SYN_VARSIMP, but with $\mathit{res}' = \mathit{res}$.

$\langle 1 \rangle 4.$ CASE: EXPL_TOP_VAL_DONE.
PROOF SKETCH: `to_fun` recursively maps Σ to Π, $\exists$ to $\forall$, $\wedge$ to $\supset$ and $*$ to $-\!\!*$, and otherwise keeps any *term* and *res* the same. Hence, $\sigma(\texttt{to\_fun}\, \mathit{ret}) = \texttt{to\_fun}\, \sigma(\mathit{ret})$, and the case proceeds by induction straightforwardly.

$\langle 1 \rangle 5.$ CASE: Typing rules which change the context (see B5.2), except for PURE_VAL_VAR, RES_SYN_VAR, and RES_SYN_VARSIMP.
For brevity, I shall only go over EXPL_TOP_SEQ_LET, as it is one of the most general rules; one which adds constraints and variables, and restricts the resource context.

PROOF SKETCH: The key idea is to apply lemma B7.2 (Substitutions can be split up) as required by the restrictions on the resource context. If a rule has a $\texttt{smt}\,(\Phi \Rightarrow \mathit{qterm})$ premise, then apply lemma B7.1 (Substitutions preserve SMT results).

$\mathcal{C}'; \mathcal{L}'; \Phi; \mathcal{R}'_1, \mathcal{R}'_2 \vdash \texttt{let}\, \mathit{ret\_pat} = \mathit{seq\_expr}\, \texttt{in}\, \mathit{texpr} \Leftarrow \mathit{ret}_2$
$\mathcal{C}; \mathcal{L}; \sigma(\Phi); \mathcal{R} \vdash \sigma \Leftarrow (\mathcal{C}'; \mathcal{L}'; \mathcal{R}'_1, \mathcal{R}'_2)$.
PROVE: $\mathcal{C}; \mathcal{L}; \sigma(\Phi); \mathcal{R} \vdash \texttt{let}\, \mathit{ret\_pat} = \sigma(\mathit{seq\_expr})\, \texttt{in}\, \sigma(\mathit{texpr}) \Leftarrow \sigma(\mathit{ret}_2)$.

$\langle 2 \rangle 1.$ $\exists \mathit{ret}_1, \mathcal{C}_3, \mathcal{L}_3, \Phi_3, \mathcal{R}_3$.
1. $\mathcal{C}'; \mathcal{L}'; \Phi; \mathcal{R}'_1 \vdash \mathit{seq\_expr} \Rightarrow \mathit{ret}_1$
2. $\Phi \vdash \mathit{ret\_pat}{:}\mathit{ret}_1 \rightsquigarrow \mathcal{C}_3; \mathcal{L}_3; \Phi_3; \mathcal{R}_3$
3. $\mathcal{C}', \mathcal{C}_3; \mathcal{L}', \mathcal{L}_3; \Phi, \Phi_3; \mathcal{R}'_2, \mathcal{R}_3 \vdash \mathit{texpr} \Leftarrow \mathit{ret}_2$.
PROOF: Inversion on assumption 1.

$\langle 2 \rangle 2.$ 1. $\forall \mathcal{C}, \mathcal{L}, \mathcal{R}_1, \sigma_1$.
$(\mathcal{C}; \mathcal{L}; \sigma_1(\Phi); \mathcal{R}_1 \vdash \sigma_1 \Leftarrow (\mathcal{C}'; \mathcal{L}'; \mathcal{R}'_1)) \Rightarrow$
$\mathcal{C}; \mathcal{L}; \sigma_1(\Phi); \mathcal{R}_1 \vdash \sigma_1(\mathit{seq\_expr}) \Rightarrow \sigma_1(\mathit{ret}_1)$.
2. $\forall \mathcal{C}, \mathcal{L}, \mathcal{R}_4, \sigma_2$.
$(\mathcal{C}; \mathcal{L}; \sigma_2(\Phi); \mathcal{R}_4 \vdash \sigma_2 \Leftarrow (\mathcal{C}'; \mathcal{L}'; \cdot)) \Rightarrow$
$\sigma_2(\Phi) \vdash \mathit{ret\_pat}{:}\sigma_2(\mathit{ret}_1) \rightsquigarrow \mathcal{C}_3; \mathcal{L}_3; \sigma_2(\Phi_3); \sigma_2(\mathcal{R}_3)$.
3. $\forall \mathcal{C}, \mathcal{L}, \mathcal{R}_2, \sigma_3$.
$(\mathcal{C}; \mathcal{L}; \sigma_3(\Phi, \Phi_3); \mathcal{R}_2 \vdash \sigma_3 \Leftarrow (\mathcal{C}', \mathcal{C}_3; \mathcal{L}', \mathcal{L}_3; \mathcal{R}'_2, \mathcal{R}_3)) \Rightarrow$
$\mathcal{C}; \mathcal{L}; \sigma(\Phi, \Phi_3); \mathcal{R}_2 \vdash \sigma(\mathit{texpr}) \Leftarrow \sigma(\mathit{ret}_2)$.
PROOF: By induction on $\langle 2 \rangle 1$.

$\langle 2 \rangle 3.$ σ and $\mathcal{R}$ can be split up into σ_1 and $\mathcal{R}_1$; σ_2; and σ_3 and $\mathcal{R}_2$ such that:
1. $\mathcal{R} = \mathcal{R}_1, \mathcal{R}_2$
2. $\mathcal{C}; \mathcal{L}; \sigma(\Phi); \mathcal{R}_1 \vdash \sigma_1 \Leftarrow (\mathcal{C}'; \mathcal{L}'; \mathcal{R}'_1)$
3. $\mathcal{C}; \mathcal{L}; \sigma(\Phi); \cdot \vdash \sigma_2 \Leftarrow (\mathcal{C}'; \mathcal{L}'; \cdot)$
4. $\mathcal{C}; \mathcal{L}; \sigma(\Phi); \mathcal{R}_2 \vdash \sigma_3 \Leftarrow (\mathcal{C}'; \mathcal{L}'; \mathcal{R}'_2)$.

PROOF: By lemma B7.2 (Substitutions can be split up).

$\langle 2\rangle 4.$ 1. $\sigma(\Phi) = \sigma_1(\Phi) = \sigma_2(\Phi) = \sigma_3(\Phi)$
2. $\sigma(\Phi_3) = \sigma_2(\Phi_3) = \sigma_3(\Phi_3)$
3. $\sigma(\mathcal{R}_3) = \sigma_2(\mathcal{R}_3) = \sigma_3(\mathcal{R}_3)$.
PROOF: All the substitutions differ only the resource-variable substitutions, but *term* and *res* (and so *ret* and Φ) do not mention resource variables.

$\langle 2\rangle 5.$ SUFFICES: $\exists \mathcal{R}_1, \mathcal{R}_2, ret_1, \mathcal{C}_3, \mathcal{L}_3, \Phi_3, \mathcal{R}_3.$
1. $\mathcal{R} = \mathcal{R}_1, \mathcal{R}_2$
2. $\mathcal{C}; \mathcal{L}; \sigma(\Phi); \mathcal{R}_1 \vdash \sigma(seq\_expr) \Rightarrow ret_1$
3. $\sigma(\Phi) \vdash ret\_pat{:}\sigma(ret_1) \rightsquigarrow \mathcal{C}_3; \mathcal{L}_3; \Phi_3; \mathcal{R}_3$
4. $\mathcal{C}, \mathcal{C}_3; \mathcal{L}, \mathcal{L}_3; \sigma(\Phi), \Phi_3; \mathcal{R}_2, \mathcal{R}_3 \vdash \sigma(texpr) \Leftarrow \sigma(ret_2)$.
PROOF: By EXPL_TOP_SEQ_LET.

$\langle 2\rangle 6.$ LET: $\mathcal{R}_1; \mathcal{R}_2; \sigma(ret_1); \mathcal{C}_3; \mathcal{L}_3; \sigma(\Phi_3); \sigma(\mathcal{R}_3)$ be the witnesses for $\langle 2\rangle 5$.
SUFFICES: 1. $\mathcal{R} = \mathcal{R}_1, \mathcal{R}_2$
2. $\mathcal{C}; \mathcal{L}; \sigma(\Phi); \mathcal{R}_1 \vdash \sigma(seq\_expr) \Rightarrow \sigma(ret_1)$
3. $\sigma(\Phi) \vdash ret\_pat{:}\sigma(ret_1) \rightsquigarrow \mathcal{C}_3; \mathcal{L}_3; \sigma(\Phi_3); \sigma(\mathcal{R}_3)$
4. $\mathcal{C}, \mathcal{C}_3; \mathcal{L}, \mathcal{L}_3; \sigma(\Phi), \sigma(\Phi_3); \mathcal{R}_2, \sigma(\mathcal{R}_3) \vdash \sigma(texpr) \Leftarrow \sigma(ret_2)$.

$\langle 2\rangle 7.$ We are done.
PROOF: Apply $\langle 2\rangle 2$ with $\langle 2\rangle 3$ and $\langle 2\rangle 4$.

$\langle 1\rangle 6.$ CASE: All remaining rules.
PROOF SKETCH: By straightforward induction. If the rule has a `smt` $(\Phi \Rightarrow qterm)$ premise, apply lemma B7.1 (Substitutions preserve SMT results).

B8 Resource Term Lemmas

B8.1 Definition: Normalised contexts

A resource context is *normalised* is it contains only predicates, quantified predicates and under-determined conditional resources.

B8.2 Resource contexts typing closed terms must be normalised

ASSUME: 1. Arbitrary *res*
2. Closed (no free-variables) *res_term*
3. $\cdot; \cdot; \Phi; \mathcal{R} \vdash \textit{res\_term} \Leftarrow \textit{res}$ (or synthesising)

PROVE: $\exists \underline{\mathcal{R}}.\, \mathcal{R} = \underline{\mathcal{R}}$.

PROOF SKETCH: By induction on the typing judgement.

⟨1⟩1. CASE: RES_SYN_EMP, RES_SYN_PRED, RES_SYN_QPRED, RES_CHK_PHI.
PROOF: $\underline{\mathcal{R}} = \mathcal{R}$ (the context is already normalised).

⟨1⟩2. CASE: RES_SYN_VAR, RES_SYN_VARSIMP
PROOF: Impossible cases (*res_terms* are not closed).

⟨1⟩3. CASE: All remaining cases (RES_SYN_PREDOPS_ITERATE, RES_SYN_PREDOPS_CONGEAL, RES_SYN_PREDOPS_EXPLODE, RES_SYN_PREDOPS_IMPLODE, RES_SYN_PREDOPS_BREAK, RES_SYN_PREDOPS_GLUE, RES_SYN_PREDOPS_INJ, RES_SYN_PREDOPS_SPLIT, RES_SYN_PREDOPS, RES_SYN_FOLD, RES_SYN_SEPCONJ, RES_CHK_PACK, RES_CHK_SEPCONJ, RES_CHK_IF_TRUE, RES_CHK_IF_FALSE, RES_CHK_SWITCH).
PROOF: By induction.

B8.3 Non-conditional resources determine context and values

This is a simple inversion lemma.

ASSUME: 1. Arbitrary *res_val*
2. $\textit{res} \neq \texttt{if}\ \textit{term}\ \texttt{then}\ \textit{res}_1\ \texttt{else}\ \textit{res}_2$.
3. $\cdot; \cdot; \Phi; \underline{\mathcal{R}} \vdash \textit{res\_val} \Leftarrow \textit{res}$ (or synthesising)

⟨1⟩1. The typing assumption cannot be any of: RES_SYN_VAR, RES_SYN_VARSIMP, RES_SYN_FOLD, RES_SYN_PREDOPS_ITERATE, RES_SYN_PREDOPS_CONGEAL, RES_SYN_PREDOPS_EXPLODE, RES_SYN_PREDOPS_IMPLODE, RES_SYN_PREDOPS_BREAK, RES_SYN_PREDOPS_GLUE, RES_SYN_PREDOPS_INJ, RES_SYN_PREDOPS_SPLIT, RES_SYN_PREDOPS.
PROOF: *res_terms* in these rules are not values.

⟨1⟩2. If $\textit{res} = \texttt{emp}$, then $\underline{\mathcal{R}} = \cdot$ and $\textit{res\_val} = \texttt{emp}$.
PROOF: By inversion, the assumption must be RES_SYN_EMP (and optionally RES_CHK_

SWITCH).

$\langle 1\rangle 3.$ If $res = term$, then $\underline{\mathcal{R}} = \cdot$ and $res\_val = \mathtt{term}$.
PROOF: By inversion, the assumption must be RES_CHK_PHI.

$\langle 1\rangle 4.$ If $res = pred\_term(oarg)$, then $\underline{\mathcal{R}} = \_{:}pred\_term(oarg)$ and $res\_val = pred\_term$.
PROOF: By inversion, the assumption must be RES_SYN_PRED (and optionally RES_CHK_SWITCH).

$\langle 1\rangle 5.$ If $res = qpred\_term(oarg)$, then $\underline{\mathcal{R}} = \_{:}qpred\_term(oarg)$ and $res\_val = qpred\_term$.
PROOF: By inversion, the assumption must be RES_SYN_QPRED (and optionally RES_CHK_SWITCH).

$\langle 1\rangle 6.$ If $res = res_1 * res_2$, then $\underline{\mathcal{R}} = \underline{\mathcal{R}}_1, \underline{\mathcal{R}}_2$ and $res\_val = \langle res\_val_1, res\_val_2\rangle$.
PROOF: By inversion, the assumption must be RES_SYN_SEPCONJ (and optionally RES_CHK_SWITCH), or RES_CHK_SEPCONJ.

$\langle 1\rangle 7.$ If $res = \exists\, y{:}\beta.\ res$, then $res\_val = \mathtt{pack}\,(oarg, res\_val')$.
PROOF: By inversion, the assumption must be RES_CHK_PACK.

B8.4 Normalised resource context determines structure of heap

This is as simple inversion lemma.

ASSUME: $\Phi \vdash h \Leftarrow \underline{\mathcal{R}}$.
$\langle 1\rangle 1.$ If $\underline{\mathcal{R}} = \cdot$, then $h = \cdot$.
PROOF: By inversion, the assumption must be HEAP_EMPTY.

$\langle 1\rangle 2.$ If $pred = ptr \overset{init}{\mapsto}_\tau value$, $\underline{\mathcal{R}} = \_{:}pred$, then $h = \{pred' \,\&\, \mathtt{None}\}$ for $\Phi \vdash pred \equiv pred'$.
PROOF: By inversion, the assumption must be HEAP_PRED_OWNED.

$\langle 1\rangle 3.$ If $\underline{\mathcal{R}} = \_{:}pred$, then $h = \{pred' \,\&\, def \,\&\, h'\}$.
for $\Phi \vdash pred \equiv pred'$.
PROOF: By inversion, the assumption must be HEAP_PRED_OTHER.

$\langle 1\rangle 4.$ If $qpred = \circledast\ x.\ iguard \Rightarrow ptr + x{\times}\mathrm{size\_of}(\tau) \overset{oarg[x].init}{\mapsto}_\tau oarg[x].value$, $\underline{\mathcal{R}} = \_{:}qpred$, then $h = \{qpred' \,\&\, \cdot\}$ for $\Phi \vdash qpred \equiv qpred'$.
PROOF: By inversion, the assumption must be HEAP_QPRED_OWNED.

$\langle 1\rangle 5.$ If $\underline{\mathcal{R}} = \_{:}qpred$, then $h = \{qpred' \,\&\, arr\_def\_heap\}$ for $\Phi \vdash qpred \equiv qpred'$.
PROOF: By inversion, the assumption must be HEAP_QPRED_OTHER.

$\langle 1\rangle 6.$ If $\underline{\mathcal{R}} = \underline{\mathcal{R}}_1, \underline{\mathcal{R}}_2$, then $h = h_1 + h_2$, where $\Phi \vdash h_1 \Leftarrow \underline{\mathcal{R}}_1$ and $\Phi \vdash h_2 \Leftarrow \underline{\mathcal{R}}_2$.
PROOF: By inversion, the assumption must be HEAP_CONCAT.

B8.5 Well-typed resource value determines its footprint

ASSUME: $\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \mathit{res\_val} \Leftarrow \mathit{res}$ (or synthesising)
$\Phi \vdash h \Leftarrow \underline{\mathcal{R}}$.
PROVE: $\forall f.\, \texttt{footprint\_of}\, \mathit{res\_val}\, \texttt{in}\, h + f \rightsquigarrow h \text{ rem } f$.

PROOF SKETCH: By induction on the typing judgement.

⟨1⟩1. CASE: RES_SYN_EMP or RES_CHK_PHI
$\underline{\mathcal{R}} = \cdot$ and so $h = \cdot$ by lemma B8.4.
PROOF: FOOTPRINT_EMP or FOOTPRINT_TERM respectively.

⟨1⟩2. CASE: RES_SYN_PRED or RES_SYN_QPRED
$\underline{\mathcal{R}} = \_{:}\mathit{pred\_term}(\mathit{oarg})$ or $\_{:}\mathit{qpred\_term}(\mathit{oarg})$, and so
$h = \{\mathit{pred\_term}(\mathit{oarg})\ \&\ \mathit{opt\_def\_heap}\}$ or $\{\mathit{qpred\_term}(\mathit{oarg})\ \&\ \mathit{arr\_def\_heap}\}$ by lemma B8.4.
PROOF: FOOTPRINT_PRED or FOOTPRINT_QPRED respectively.

⟨1⟩3. CASE: $\texttt{pack}\,(\mathit{oarg}, \mathit{res\_val}')$.
PROOF: By induction.

⟨1⟩4. CASE: FOOTPRINT_SEPPAIR.
$\mathit{res\_val} = \langle \mathit{res\_val}_1, \mathit{res\_val}_2 \rangle$,
$\underline{\mathcal{R}} = \underline{\mathcal{R}}_1, \underline{\mathcal{R}}_2$, and so
$h = h_1 + h_2$ where $\Phi \vdash h_1 \Leftarrow \underline{\mathcal{R}}_1$ and $\Phi \vdash h_2 \Leftarrow \underline{\mathcal{R}}_2$ by lemma B8.4.

⟨2⟩1. $\texttt{footprint\_of}\, \mathit{res\_val}_1\, \texttt{in}\, h_1 + h_2 + f \rightsquigarrow h_1 \text{ rem } h_2 + f$.
PROOF: Instantiate inductive hypothesis with $h_2 + f$.

⟨2⟩2. $\texttt{footprint\_of}\, \mathit{res\_val}_1\, \texttt{in}\, h_2 + f \rightsquigarrow h_2 \text{ rem } f$.
PROOF: Instantiate inductive hypothesis with f.

B8.6 Progress and type preservation for resource terms

ASSUME: 1. Closed (no free-variables) $\mathit{res\_term}$
2. $\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \mathit{res\_term} \Leftarrow \mathit{res}$ (or synthesising)
3. $\Phi \vdash h \Leftarrow \underline{\mathcal{R}}$

PROVE: $\exists \mathit{res\_val}, \underline{\mathcal{R}}', h'$.
1. $\cdot;\cdot;\Phi;\underline{\mathcal{R}}' \vdash \mathit{res\_val} \Leftarrow \mathit{res}$ (or synthesising respectively)
2. $\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}'$
3. $\forall f.\, \langle h + f; \mathit{res\_term} \rangle \Downarrow \langle h' + f; \mathit{res\_val} \rangle$.

PROOF SKETCH: Induction on the resource term typing assumption. The type dictates the value and context, the latter of which dictates the shape of the heap.

Because of this direction of information, you cannot prove that

$\forall \underline{\mathcal{R}'}.\ (\Phi \vdash h' \Leftarrow \underline{\mathcal{R}'}) \Rightarrow (\cdot;\cdot;\cdot;\mathcal{R}'' \vdash \mathit{res\_val}' \Leftarrow \mathit{res})$. The converse is already true by the composition of lemmas B8.3 and B8.4. You need the existential, so that you can provide it as a witness when proving heap typing for folded predicates, which you need to use in proving unfolding predicates in pattern-matching.

⟨1⟩1. CASE: RES_SYN_PREDOPS_ITERATE

LET: $\mathit{res\_term} = \texttt{iterate}\,(\mathit{res\_term}', n)$
$\mathit{qpred\_term} = (x; 0 \le x \wedge x \le n-1)\{\texttt{Owned}\,\langle\tau\rangle(\mathit{ptr} + x \times \mathrm{size\_of}(\tau))\}$
$\mathit{res} = \mathit{qpred\_term}(\mathit{oarg})$
$\mathit{pred\_term} = \texttt{Owned}\,\langle\texttt{array}\,n\,\tau\rangle(\mathit{ptr})$
$\mathit{res}' = \mathit{pred\_term}(\mathit{oarg}')$.

⟨2⟩1. $\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \mathit{res\_term}' \Rightarrow \mathit{res}'$.
PROOF: By inversion on the typing assumption.

⟨2⟩2. $\exists h'', \underline{\mathcal{R}''}, \mathit{res\_val}'$.
1. $\cdot;\cdot;\cdot;\mathcal{R}'' \vdash \mathit{res\_val}' \Rightarrow \mathit{res}'$
2. $\Phi \vdash h'' \Leftarrow \underline{\mathcal{R}''}$
3. $\forall f.\ \langle h + f; \mathit{res\_term}'\rangle \Downarrow \langle h'' + f; \mathit{res\_val}'\rangle$.
PROOF: By the induction hypothesis.

⟨2⟩3. $\mathit{res\_val}' = \mathit{pred\_term}$ and $\underline{\mathcal{R}''} = \_{:}\mathit{res}'$.
PROOF: By ⟨2⟩2 and lemma B8.3 (Non-conditional resources determine context and values).

⟨2⟩4. $h'' = \{\mathit{pred\_term}(\mathit{oarg}')\ \&\ \texttt{None}\}$.
PROOF: By ⟨2⟩3 and lemma B8.4 (Normalised resource context determines structure of heap).

⟨2⟩5. LET: $\mathit{res\_val} = (x; 0 \le x \wedge x \le n-1)\{\texttt{Owned}\,\langle\tau\rangle(\mathit{ptr} + x \times \mathrm{size\_of}(\tau))\}$
$\underline{\mathcal{R}'} = \_{:}\mathit{qpred\_term}(\mathit{oarg})$ and $h' = \{\mathit{qpred\_term}(\mathit{oarg})\ \&\ \cdot\}$.
PROOF: Prove value typing using RES_SYN_QPRED; heaping typing using HEAP_QPRED_OWNED; reduction using PREDOPS_RESV_ITERATE.

⟨1⟩2. CASE: RES_SYN_PREDOPS_CONGEAL

PROOF: Like RES_SYN_PREDOPS_ITERATE, but with:
$\mathit{res\_term} = \texttt{congeal}\,(\mathit{res\_term}', n)$
$\mathit{res} = \mathit{pred\_term}(\mathit{oarg})$ where $\mathit{pred\_term} = \texttt{Owned}\,\langle\texttt{array}\,n\,\tau\rangle(\mathit{ptr})$

$\mathit{res}' = \mathit{qpred\_term}(\mathit{oarg}')$ where $\mathit{qpred\_term} = (x; \mathit{iguard})\{\texttt{Owned}\,\langle\tau\rangle(\mathit{ptr} + x \times \mathrm{size\_of}(\tau))\}$
$\mathit{res\_val}' = \mathit{qpred\_term}$ and $\underline{\mathcal{R}''} = \_{:}\mathit{res}'$, by lemma B8.3

Let $\mathit{res\_val} = \mathit{pred\_term}$, $\underline{\mathcal{R}'} = \_{:}\mathit{pred\_term}(\mathit{oarg})$ and $h' = \{\mathit{pred\_term}(\mathit{oarg})\ \&\ \texttt{None}\}$ to prove: value typing using RES_SYN_PRED; heap typing using HEAP_PRED_OWNED; reduction using PREDOPS_RESV_CONGEAL.

⟨1⟩3. CASE: RES_SYN_PREDOPS_EXPLODE

PROOF: Like RES_SYN_PREDOPS_ITERATE, but with:
$\mathit{res\_term} = \texttt{explode}\,(\mathit{res\_term}')$

$res = \circledast\ (\overline{pred\_term_i(\,oarg_i\,)}^{\,i}\ )$ where $pred\_term_i = \texttt{Owned}\,\langle\tau_i\rangle(ptr +_{\mathrm{ptr}} \mathrm{offset\_of}_{tag}(member_i))$

$res' = pred\_term(\,oarg\,)$ where $pred\_term = \texttt{Owned}\,\langle\texttt{struct}\,tag\rangle(ptr)$
$res\_val' = pred\_term$ and $\underline{\mathcal{R}}'' = \_{:}pred\_term(\,oarg\,)$, by lemma B8.3

Let $res\_val = \langle\,\overline{pred\_term_i}^{\,i}\,\rangle$, $\underline{\mathcal{R}}' = \overline{\_{:}pred\_term_i(\,oarg_i\,)}^{\,i}$ and
$h' = \overline{\{pred\_term_i(\,oarg_i\,)\ \&\ \texttt{None}\}}^{\,i}$, to prove: value typing using Res_Syn_Pred and Res_Syn_SepConj; heap typing using Heap_Concat and Heap_Pred_Owned; reduction using PredOps_ResV_Explode.

⟨1⟩4. Case: Res_Syn_PredOps_Implode
Proof: Like Res_Syn_PredOps_Iterate, but with:
$res\_term = \texttt{implode}\,(res\_term', tag)$
$res = pred\_term(\,oarg\,)$ where $pred\_term = \texttt{Owned}\,\langle\texttt{struct}\,tag\rangle(ptr)$

$res' = \circledast\ (\overline{pred\_term_i(\,oarg_i\,)}^{\,i}\ )$ where $pred\_term_i = \texttt{Owned}\,\langle\tau_i\rangle(ptr +_{\mathrm{ptr}} \mathrm{offset\_of}_{tag}(member_i))$
$res\_val' = \overline{pred\_term_i}^{\,i}$ and $\underline{\mathcal{R}}'' = \overline{\_{:}pred\_term_i(\,oarg_i\,)}^{\,i}$, by lemma B8.3

Let $res\_val = \texttt{Owned}\,\langle\texttt{struct}\,tag\rangle(ptr)$, $\underline{\mathcal{R}}' = \_{:}pred\_term(\,oarg\,)$, and
$h' = \{pred\_term(\,oarg\,)\ \&\ \texttt{None}\}$, to prove: value typing using Res_Syn_Pred; heap typing using Heap_Pred_Owned; reduction using PredOps_ResV_Implode.

⟨1⟩5. Case: Res_Syn_PredOps_Break
Proof: Like Res_Syn_PredOps_Iterate, but with:
$res\_term = \texttt{break}\,(res\_term', term)$
$res = qpred\_term(\,oarg\,) * pred\_term(\,oarg[term]\,)$ where
$qpred\_term = (x; iguard \wedge (x \neq term))\{\alpha(ptr + x{\times}step, iargs)\}$ and
$pred\_term = \alpha(ptr + (term \times step), term/x(iargs))$

$res' = qpred\_term'(\,oarg\,)$ where $qpred\_term' = (x; iguard)\{\alpha(ptr + x{\times}step, iargs)\}$
$res\_val' = qpred\_term'$, and $\underline{\mathcal{R}}'' = \_{:}qpred\_term'(\,oarg\,)$, by lemma B8.3.

If predicate is $\texttt{Owned}\,\langle\tau\rangle$, $h'' = \{qpred\_term'(\,oarg\,)\ \&\ \cdot\}$ (by lemma B8.4), so let
$h' = \{qpred\_term(\,oarg\,)\ \&\ \cdot\} + \{pred\_term(\,oarg[term]\,)\ \&\ \texttt{None}\}$ (by $\cdot[term] = \texttt{None}$).
Otherwise, $h'' = \{qpred\_term'(\,oarg\,)\ \&\ arr\_def\_heap\}$, (again by lemma B8.4), so let
$h' = \{qpred\_term(\,oarg\,)\ \&\ arr\_def\_heap\} + \{pred\_term(\,oarg[term]\,)\ \&\ arr\_def\_heap[term]\}$.

Let $res\_val = \langle qpred\_term, pred\_term\rangle$ and
$\underline{\mathcal{R}}' = \_{:}qpred\_term(\,oarg\,), \_{:}pred\_term(\,oarg[term]\,)$ to prove: value typing using Res_Syn_QPred, Res_Syn_Pred, Res_Syn_SepConj; heap typing using Heap_Concat, Heap_QPred_Owned / Heap_QPred_Other, and Heap_Pred_Owned / Heap_Pred_Other (with witness $\_{:}pred\_term(\,oarg[term]\,)$); reduction using PredOps_ResV_Break.

⟨1⟩6. Case: Res_Syn_PredOps_Glue
Proof: Like Res_Syn_PredOps_Iterate, but with:
$res\_term = \texttt{glue}\,(res\_term')$

$res = qpred\_term(oarg_1[term] := oarg_2)$ where
$qpred\_term = (x; iguard \vee x = term)\{\alpha(ptr_1 + x \times step, \overline{iarg_{1\,i}}^{\,i})\}$

$res' = qpred\_term_1(oarg_1) * pred\_term(oarg_2)$ where
$qpred\_term_1 = (x; iguard)\{\alpha(ptr_1 + x \times step, \overline{iarg_{1\,i}}^{\,i})\}$ and $pred\_term = \alpha(ptr_2, \overline{iarg_{2\,i}}^{\,i})$.
$res\_val' = \langle qpred\_term_1, pred\_term\rangle$, and $\underline{\mathcal{R}}'' = \_{:}qpred\_term_1(oarg_1), \_{:}pred\_term(oarg_2)$, by lemma B8.3.

If predicate is $\texttt{Owned}\,\langle\tau\rangle$, $h'' = \{qpred\_term_1(oarg_1)\ \&\ \cdot\} + \{pred\_term(oarg_2)\ \&\ \texttt{None}\}$ (by lemma B8.4), so let $h' = \{qpred\_term(oarg)\ \&\ \cdot\}$ (by $\cdot[term] := \texttt{None} = \cdot$). Otherwise, $h'' = \{qpred\_term_1(oarg_1)\ \&\ arr\_def\_heap\} + \{pred\_term(oarg_2)\ \&\ def\ \&\ heap\}$ (again by lemma B8.4), so let $h' = \{qpred\_term(oarg)\ \&\ arr\_def\_heap[term] := def\ \&\ heap\}$.

Let $res\_val = qpred\_term$ and $\underline{\mathcal{R}} = \_{:}qpred\_term(oarg_1[term] := oarg_2)$, to prove: value typing using RES_SYN_QPRED; heap typing using HEAP_QPRED_OWNED / HEAP_QPRED_OTHER; reduction using PREDOPS_RESV_GLUE.

⟨1⟩7. CASE: RES_SYN_PREDOPS_INJ
PROOF: Like RES_SYN_PREDOPS_ITERATE, but with:
$res\_term = \texttt{inj}\,(res\_term', ptr_1, step, x.\,\overline{iarg_1}^{\,i})$
$res = qpred\_term((\texttt{default array}\,\beta)[term] := oarg)$ where
$qpred\_term = (x; x = term)\{\alpha(ptr_1 + x \times step, \overline{iarg_{1\,i}}^{\,i})\}$

$res' = pred\_term(oarg)$ where $pred\_term = \alpha(ptr_2, \overline{iarg_{2\,i}}^{\,i})$
$res\_val' = pred\_term$, and $\underline{\mathcal{R}}'' = \_{:}pred\_term(oarg)$, by lemma B8.3.

If predicate is $\texttt{Owned}\,\langle\tau\rangle$, $h'' = \{pred\_term(oarg)\ \&\ \texttt{None}\}$ (by lemma B8.4), so let $h' = \{qpred\_term((\texttt{default array}\,\beta)[term] := oarg)\ \&\ \cdot\}$ (by $\cdot[term] := \texttt{None} = \cdot$). Otherwise, $h'' = \{pred\_term(oarg)\ \&\ def\ \&\ heap\}$ (again by lemma B8.4), so let $h' = \{qpred\_term((\texttt{default array}\,\beta)[term] := oarg)\ \&\ \cdot[term] := def\ \&\ heap\}$.

Let $res\_val = qpred\_term$, and $\underline{\mathcal{R}} = \_{:}qpred\_term((\texttt{default array}\,\beta)[term] := oarg)$, to prove typing using HEAP_QPRED_OWNED / HEAP_QPRED_OTHER, and reduction using PREDOPS_RESV_INJ.

⟨1⟩8. CASE: RES_SYN_PREDOPS_SPLIT
PROOF: Like RES_SYN_PREDOPS_ITERATE, but with:
$res\_term = \texttt{split}\,(res\_term', iguard)$
$res = qpred\_term_1(oarg) * qpred\_term_2(oarg)$ where
$qpred\_term_1 = (x; iguard)\{\alpha(ptr + x \times step, iargs)\}$ and
$qpred\_term_2 = (x; iguard_2)\{\alpha(ptr + x \times step, iargs)\}$

$res' = qpred\_term(oarg)$ where $qpred\_term = (x; iguard')\{\alpha(ptr + x \times step, iargs)\}$
$res\_val' = qpred\_term$, and $\underline{\mathcal{R}}'' = \_{:}qpred\_term(oarg)$, by lemma B8.3.

If predicate is $\texttt{Owned}\,\langle\tau\rangle$, $h'' = \{qpred\_term(oarg)\ \&\ \cdot\}$ (by lemma B8.4), so let $h' = \{qpred\_term_1(oarg)\ \&\ \cdot\} + \{qpred\_term_2(oarg)\ \&\ \cdot\}$. Otherwise,

$h'' = \{qpred\_term(oarg)\ \&\ arr\_def\_heap\}$ (again by lemma B8.4), so let
$h' = \{qpred\_term_1(oarg)\ \&\ arr\_def\_heap\} + \{qpred\_term_2(oarg)\ \&\ arr\_def\_heap\}$.

Let $res\_val = \langle qpred\_term_1, qpred\_term_2 \rangle$, and
$\underline{\mathcal{R}} = \_{:}qpred\_term_1(oarg), \_{:}qpred\_term_2(oarg)$, to prove: value typing using RES_SYN_QPRED and RES_SYN_SEPCONJ; heap typing using HEAP_CONCAT and HEAP_QPRED_OWNED / HEAP_QPRED_OTHER; reduction using PREDOPS_RESV_SPLIT.

$\langle 1 \rangle 9$. CASE: RES_SYN_EMP, RES_SYN_PRED, RES_SYN_QPRED, RES_CHK_PHI.
PROOF: In these cases, $h = h'$, $\underline{\mathcal{R}} = \underline{\mathcal{R}}'$ and $res\_term = res\_val$.
Typing holds by assumption; prove reduction using REST_RESV_VAL.

$\langle 1 \rangle 10$. CASE: RES_SYN_PREDOPS
PROOF: Both typing and reduction (using REST_RESV_PREDOPS) hold by induction.

$\langle 1 \rangle 11$. CASE: RES_SYN_SEPCONJ, RES_CHK_SEPCONJ.
$res = res_1 * res_2$,
$res\_term = \langle res\_term_1, res\_term_2 \rangle$,
$h = h_1 + h_2$, so $\underline{\mathcal{R}} = \underline{\mathcal{R}}_1, \underline{\mathcal{R}}_2$,
$\Phi \vdash h_1 \Leftarrow \underline{\mathcal{R}}_1$ and $\Phi \vdash h_2 \Leftarrow \underline{\mathcal{R}}_2$.

$\langle 2 \rangle 1$. $\exists h'_1, \underline{\mathcal{R}}'_1, res\_val_1 \ldots \wedge (\forall f_1 \ldots)$
$\exists h'_2, \underline{\mathcal{R}}'_2, res\_val_2 \ldots \wedge (\forall f_2 \ldots)$
PROOF: By induction.

$\langle 2 \rangle 2$. $\langle h_1 + h_2 + f; res\_term_1 \rangle \Downarrow \langle h'_1 + h_2 + f; res\_val_1 \rangle$.
$\langle h'_1 + h_2 + f; res\_term \rangle \Downarrow \langle h'_1 + h'_2 + f; res\_val_2 \rangle$.
PROOF: Instantiate f_1 with $h_2 + f$, and f_2 with, $h'_1 + f$.

$\langle 2 \rangle 3$. LET: $res\_val = \langle res\_val_1, res\_val_2 \rangle$, $\underline{\mathcal{R}}' = \underline{\mathcal{R}}'_1, \underline{\mathcal{R}}'_2$, and $h' = h'_1 + h'_2$.
Prove value typing using RES_SYN_SEPCONJ / RES_CHK_SEPCONJ; heap typing using HEAP_CONCAT; reduction using $\langle 2 \rangle 2$ and REST_RESV_SEPPAIR.

$\langle 1 \rangle 12$. CASE: RES_CHK_PACK
PROOF: Like RES_SYN_PREDOPS_ITERATE, but with:
$res\_term = \texttt{pack}\,(oarg, res\_term')$, $res = \exists\, y{:}\beta.\ res''$, $res' = oarg/y(res'')$
$res\_val = \texttt{pack}\,(oarg, res\_val')$. Value and heap typing hold by induction; prove reduction using REST_RESV_PACK.

$\langle 1 \rangle 13$. CASE: RES_SYN_FOLD
PROOF: Like RES_SYN_PREDOPS_ITERATE, but with:
$\alpha \ \equiv\ x_p{:}\_,\ \overline{x_i{:}\_{i}}^{\,i}, y{:}\_ \mapsto res'' \ \in \texttt{Globals}$
$res\_term = \texttt{fold}\, res\_term'{:}\alpha(ptr, \overline{iarg_i}^{\,i})(oarg)$
$res = \alpha(ptr, \overline{iarg_i}^{\,i})(oarg)$
$res' = [oarg/y, [\,\overline{iarg_i/x_i}^{\,i}\,], ptr/x_p](res'')$.

$\exists h_1, \underline{\mathcal{R}}', res\_val'$.
1. $\cdot; \cdot; \Phi; \underline{\mathcal{R}}' \vdash res\_val' \Leftarrow res'$

2. $\Phi \vdash h_1 \Leftarrow \underline{\mathcal{R}}'$
3. $\forall f.\, \langle h + f; \mathit{res\_term}\rangle \Downarrow \langle h_1 + f; \mathit{res\_val}'\rangle$

(by induction).

Let $\mathit{res\_val} = \alpha(\mathit{ptr}, \overline{\mathit{iarg}_i}^{\,i})$, $\underline{\mathcal{R}}' = \_{:}\alpha(\mathit{ptr}, \overline{\mathit{iarg}_i}^{\,i})(\mathit{oarg})$ and $h' = \{\alpha(\mathit{ptr}, \overline{\mathit{iarg}_i}^{\,i})(\mathit{oarg})\ \&\ \mathit{res\_val}'\ \&\ h_1\}$, to prove: value typing using Res_Syn_Pred; heap typing using Heap_Pred_Other.
Since $\texttt{footprint\_of}\ \mathit{res\_val}'\ \texttt{in}\ h_1 + f \rightsquigarrow h_1$ rem f by lemma B8.5 (Well-typed resource value determines its footprint), prove reduction using ResT_ResV_Fold.

⟨1⟩14. Case: Res_Chk_If_True, Res_Chk_If_False
Proof: By induction with res' as res_1 or res_2 respectively. This is exhaustive because only variables can synthesise under-determined conditional resources and those are excluded by assumption of $\mathit{res\_term}$ being closed.

⟨1⟩15. Case: Res_Chk_Switch
Proof: By induction on the synthesising judgement.

B8.7 Resource term reduction is deterministic

Proof sketch: Induction over the definition: it is syntax directed.

B8.8 Resource term reduction is isolated

If $\mathit{res\_term}$ is closed, $\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \mathit{res\_term} \Leftarrow \mathit{res}$ $\Phi \vdash h \Leftarrow \underline{\mathcal{R}}$ and $\langle h + f; \mathit{res\_term}\rangle \Downarrow \langle \mathit{heap}; \mathit{res\_val}\rangle$ then $\exists h', \underline{\mathcal{R}}'.\, \mathit{heap} = h' + f \wedge (\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}') \wedge (\cdot;\cdot;\Phi;\underline{\mathcal{R}}' \vdash \mathit{res\_val} \Leftarrow \mathit{res})$.

Proof: This simply the composition of lemma B8.7 (Resource term reduction is deterministic) and lemma B8.6 (Progress and type preservation for resource terms).

B9 Progress

B9.1 $\Phi \vdash \mathit{res} \sim \mathit{res}'$ is an equivalence relation

Proof sketch: By induction and $\mathit{term} \sim \mathit{term}'$ assumed to be an equivalence relation (see section B4 Proof Judgements).

B9.2 $\Phi \vdash \mathit{res} \sim \mathit{res}'$ is preserved by substitution

If $x \sim y$ in $\Phi \vdash \mathit{res} \sim \mathit{res}'$ and $\mathit{term} \sim \mathit{term}'$ then $\Phi \vdash \mathit{term}/x(\mathit{res}) \sim \mathit{term}/y(\mathit{res}')$.

Proof sketch: By induction and $\mathit{term} \sim \mathit{term}'$ assumed to be preserved by substitution (see section B4 Proof Judgements).

B9.3 Well-typed spines produce substitutions and the same return type

Assume: $\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \overline{\mathit{spine\_elem}_i}^{\,i} :: \psi_1(\mathit{fun}_1) \gg \mathit{ret}_1$
$\Phi \vdash h \Leftarrow \underline{\mathcal{R}}$ and $\psi_1(\mathit{fun}_1) = \psi_2(\mathit{fun}_2) = \psi_2(\mathit{fun}_3)$
$\langle h + f;\ \overline{x_i = \mathit{spine\_elem}_i}^{\,i} \rangle :: \psi_2(\mathit{fun}_2) \gg \langle \mathit{heap}; \sigma_2; \mathit{ret}_2 \rangle$
$\overline{x_i}^{\,i} :: \mathit{fun}_3 \rightsquigarrow \mathcal{C};\mathcal{L};\Phi';\mathcal{R}' \mid \mathit{ret}_3$.

Prove: $\psi_1(\mathit{ret}_1) = \psi_2(\mathit{ret}_2) = [\psi_2, \sigma_2](\mathit{ret}_3)$
$\exists h', \underline{\mathcal{R}}'.\ \mathit{heap} = h' + f, \Phi \vdash h' \Leftarrow \underline{\mathcal{R}}'$ and
$\cdot;\cdot;\Phi;\underline{\mathcal{R}}' \vdash \psi_2(\sigma_2) \Leftarrow (\mathcal{C};\mathcal{L};\psi_2(\mathcal{R}'))$.

⟨1⟩1. Case: Expl_Spine_Ret
$\cdot;\cdot;\Phi;\cdot \vdash\ ::\psi_1(\mathit{ret}_1) \gg \psi_1(\mathit{ret})$
$\Phi \vdash \cdot \Leftarrow \cdot$ (by inversion, Heap_Empty).
$\langle f;\ \rangle :: \psi_2(\mathit{ret}) \gg \langle f; \cdot; \psi_2(\mathit{ret}) \rangle$ (by inversion, Subs_Spine_Empty).
$::\mathit{ret} \rightsquigarrow \cdot;\cdot;\cdot;\cdot \mid \mathit{ret}$ where $\mathit{ret}'' = \mathit{ret}$ (by inversion, Fun_Env_Ret).

$\psi_1(\mathit{ret}_1) = \psi_2(\mathit{ret}_2) = [\psi_2, \cdot](\mathit{ret}_3)$ (by assumption)
Let: $h' = \cdot$, $\underline{\mathcal{R}}' = \cdot$.
$f = h' + f$ trivally.
$\Phi \vdash \cdot \Leftarrow \cdot$ by Heap_Empty.
$\cdot;\cdot;\Phi;\cdot \vdash \cdot \Leftarrow (\cdot;\cdot;\cdot)$ by Subs_Chk_Empty.

⟨1⟩2. Case: Expl_Spine_Comp
$\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \overline{\mathit{spine\_elem}_i}^{\,i} :: [\mathit{pval}/x, \psi_1](\mathit{fun}_1') \gg \mathit{ret}_1$
$\cdot \vdash \mathit{pval} \Rightarrow \beta$
$\langle h + f;\ \overline{x_i = \mathit{spine\_elem}_i}^{\,i} \rangle :: [\mathit{pval}/x, \psi_2](\mathit{fun}_2') \gg \langle \mathit{heap}; \sigma_2'; \mathit{ret}_2 \rangle$
(by inversion, Subs_Spine_Comp) x, $\overline{x_i}^{\,i} :: \Pi\, x{:}\beta.\ \mathit{fun}_3' \rightsquigarrow x{:}\beta, \mathcal{C};\mathcal{L};\Phi';\mathcal{R}' \mid \mathit{ret}_3$ (by inversion, Fun_Env_Comp).

$[\mathit{pval}/x, \psi_1](\mathit{ret}_1) = [\mathit{pval}/x, \psi_2](\mathit{ret}_2) = [\mathit{pval}/x, \psi_2, \sigma_2'](\mathit{ret}_3)$
$\exists h', \underline{\mathcal{R}}'.\ \mathit{heap} = h' + f, \Phi \vdash h' \Leftarrow \underline{\mathcal{R}}'$ and
$\cdot;\cdot;\Phi;\underline{\mathcal{R}}' \vdash [\mathit{pval}/x, \psi_2](\sigma_2') \Leftarrow (\mathcal{C}';\mathcal{L};[\mathit{pval}/x, \psi_2](\mathcal{R}'))$ (by induction).

$[\psi_2, pval/x, \sigma'_2](ret_3) = [pval/x, \psi_2, \sigma'_2]$ and
$\cdot;\cdot;\Phi;\underline{\mathcal{R}}' \vdash \psi_2([pval/x, \sigma'_2]) \Leftarrow (\mathcal{C}', x{:}\beta; \mathcal{L}; \psi_2(\mathcal{R}'))$,
by SUBS_CHK_COMP and SUBS_CHK_CONCAT (because *pval* is closed, we have $[pval/x, \psi_2(\sigma'_2)] = \psi_2([pval/x, \sigma'_2])$).

⟨1⟩3. CASE: EXPL_SPINE_COMP
Similar to EXPL_SPINE_COMPbut with SUBS_CHK_LOG.

⟨1⟩4. CASE: EXPL_SPINE_PHI
By induction (does not affect substitution).

⟨1⟩5. CASE: EXPL_SPINE_RES
$\cdot;\cdot;\Phi;\underline{\mathcal{R}}_2 \vdash \overline{spine\_elem_i}^{\,i} :: \psi_1(fun_1) \gg ret_1$
$\cdot;\cdot;\Phi;\underline{\mathcal{R}}_1 \vdash res\_term \Leftarrow \psi_1(res)$
$\exists h_1, h_2.\, h = h_1 + h_2 \wedge \Phi \vdash h_1 \Leftarrow \underline{\mathcal{R}}_1 \wedge \Phi \vdash h_2 \Leftarrow \underline{\mathcal{R}}_2$ (by B8.4).
$\langle h_1 + h_2 + f; res\_term \rangle \Downarrow \langle heap_1; res\_val \rangle$
$\langle heap_1; \overline{x_i = spine\_elem_i}^{\,i} \rangle :: [res\_val/x, \psi'_2](fun_2) \gg \langle heap_2; \sigma'_2; ret_2 \rangle$ (by inversion, SUBS_SPINE_RES).

$\exists h'_1, \underline{\mathcal{R}}'_1, res\_val'.\ \cdot;\cdot;\Phi;\underline{\mathcal{R}}'_1 \vdash res\_val' \Leftarrow \psi_1(res),\ \Phi \vdash h'_1 \Leftarrow \underline{\mathcal{R}}'_1$
and $\langle h_1 + h_2 + f; res\_term \rangle \Downarrow \langle h'_1 + h_2 + f; res\_val' \rangle$
(by lemma B8.6 (Progress and type preservation for resource terms)).
$heap_1 = h'_1 + h_2 + f$ and $res\_val = res\_val'$
(by lemma B8.8 (Resource term reduction is isolated))

$\psi_1(ret_1) = [res\_val/x, \psi_2](ret_2) = [res\_val/x, \psi_2, \sigma'_2](ret_3)$
(because resources variables not in types)
$\exists h'_2, \underline{\mathcal{R}}'_2.\, heap_2 = h'_2 + h'_1 + f \wedge \Phi \vdash h'_2 \Leftarrow \underline{\mathcal{R}}'_2$
$\cdot;\cdot;\Phi;\underline{\mathcal{R}}'_2 \vdash [res\_val/x, \psi_2](\sigma'_2) \Leftarrow (\mathcal{C}; \mathcal{L}; [res\_val/x, \psi_2](\mathcal{R}'))$ (by induction).

LET: $h' = h'_1 + h'_2$ and $\underline{\mathcal{R}}' = \underline{\mathcal{R}}'_1, \underline{\mathcal{R}}'_2$.
Hence $\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}'_1, \underline{\mathcal{R}}'_2$ (by HEAP_CONCAT),
$[res\_val/x, \psi_2, \sigma'_2](ret_3) = [\psi_2, res\_val/x, \sigma'_2](ret_3)$ and
$\cdot;\cdot;\Phi;\underline{\mathcal{R}}'_1, \underline{\mathcal{R}}'_2 \vdash \psi_2([res\_val/x, \sigma'_2]) \Leftarrow (\mathcal{C}; \mathcal{L}; \psi_2(x{:}res, \mathcal{R}'))$, by SUBS_CHK_RES and SUBS_CHK_CONCAT (because *res_val* is closed, we have $[res\_val/x, \psi_2(\sigma'_2)] = \psi_2([res\_val/x, \sigma'_2])$).

B9.4 Well-typed values pattern-match successfully

Note that the definition of $term \sim term'$ is not explicitly stated; see section B4 (Proof Judgements) for more details.

ASSUME: 1. $\mathcal{C}; \mathcal{L}; \Phi \vdash \overline{ret\_pat_i}^{\,i}{:}ret \rightsquigarrow \mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'$
2. $\overline{ret\_pat_i}^{\,i}$ is exhaustive
3. $\Phi \vdash fun \sim ret$
4. $\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \overline{ret\_term_i}^{\,i} :: fun \gg \mathtt{I}$
5. $\Phi \vdash h \Leftarrow \underline{\mathcal{R}}$

Prove: $\exists h', \sigma.$
$\forall f.\, \langle h + f;\, \overline{\mathit{ret\_pat}_i = \mathit{ret\_term}_i}^{\,i} \,\rangle \rightsquigarrow \langle h' + f;\, \sigma \rangle$
$\exists \underline{\mathcal{R}}'.$
$\mathcal{C}; \mathcal{L}; \Phi \vdash h' \Leftarrow \underline{\mathcal{R}}' \wedge \mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}}' \vdash \sigma \Leftarrow (\mathcal{C}'; \mathcal{L}'; \mathcal{R}').$

Proof sketch: Induction over the pattern-matching judgement.

⟨1⟩1. Case: Pat_Ret_Empty
$\mathcal{C}; \mathcal{L}; \Phi \vdash \ \mathtt{:I} \rightsquigarrow \cdot; \cdot; \cdot; \cdot$
which means $\mathit{fun} = \mathtt{I}$ (by inversion, Rel_Ret_I)
and so $\mathcal{C}; \mathcal{L}; \Phi; \cdot \vdash \ \mathtt{::I} \gg \mathtt{I}$ (by inversion, Expl_Spine_Ret),
and $h = \cdot$ (by lemma B8.4).
Let $h' = \cdot$, to step with Subs_Pat_Ret_Empty.
Let $\underline{\mathcal{R}}' = \cdot$, to type h' with Heap_Empty and σ with Subs_Chk_Empty.

⟨1⟩2. Case: Pat_Ret_Comp
$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathtt{comp}\, \mathit{ident\_or\_pat},\, \overline{\mathit{ret\_pat}_j}^{\,j} {:} \Sigma\, y{:}\beta.\, \mathit{ret} \rightsquigarrow \mathcal{C}_1, \mathcal{C}_2; \mathcal{L}_2; \Phi_2; \mathcal{R}_2$
which means $\mathit{fun} = \Pi\, x{:}\beta.\, \mathit{fun}'$ (by inversion, Rel_Ret_Comp),
and so $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \mathit{pval},\, \overline{\mathit{ret\_term}_j}^{\,j} :: \Pi\, x{:}\beta.\, \mathit{fun}' \gg \mathtt{I}$ (by inversion, Expl_Spine_Comp).

$\mathit{ident\_or\_pat}{:}\beta \rightsquigarrow \mathcal{C}_1\, \mathtt{with}\, \mathit{term}_1$ (from the pattern-matching assumption),
$\mathit{ident\_or\_pat}$ is exhaustive (from the exhaustive asumption),
and $\cdot \vdash \mathit{pval} \Rightarrow \beta$ (from the spine typing assumption),
imply $\mathit{term}_1 \sim \mathit{pval}$, $\mathit{ident\_or\_pat} = \mathit{pval} \rightsquigarrow \sigma_1$
and $\cdot; \cdot; \cdot; \cdot \vdash \sigma_1 \Leftarrow (\mathcal{C}_1; \cdot; \cdot)$ (by the nested proof below).

$\mathcal{C}, \mathcal{C}_1; \mathcal{L}; \Phi \vdash \overline{\mathit{ret\_pat}_j}^{\,j} {:} \mathit{term}_1/y(\mathit{ret}') \rightsquigarrow \mathcal{C}_2; \mathcal{L}_2; \Phi_2; \mathcal{R}_2$ (from the pattern-matching assumption),
$\forall\, \mathit{term}_1 \sim \mathit{pval}.\ \Phi \vdash \mathit{pval}/x(\mathit{fun}') \sim \mathit{term}_1/y(\mathit{ret}')$ (from the related assumption),
$\cdot; \cdot; \cdot; \underline{\mathcal{R}} \vdash \overline{\mathit{ret\_term}_j}^{\,j} :: \mathit{pval}/x(\mathit{fun}') \gg \mathtt{I}$ (from the spine typing assumption),
and $\Phi \vdash h \Leftarrow \underline{\mathcal{R}}$, imply $\langle h + f;\, \overline{\mathit{ret\_pat}_j = \mathit{ret\_term}_j}^{\,j} \,\rangle \rightsquigarrow \langle h'' + f;\, \sigma_2 \rangle$
and that $\exists \underline{\mathcal{R}}''$ such that $\mathcal{C}, \mathcal{C}_1; \mathcal{L}; \Phi \vdash h'' \Leftarrow \underline{\mathcal{R}}''$ and $\mathcal{C}, \mathcal{C}_1; \mathcal{L}; \Phi; \underline{\mathcal{R}}'' \vdash \sigma_2 \Leftarrow (\mathcal{C}_2; \mathcal{L}_2; \mathcal{R}_2)$ (by induction).

Since $\mathcal{C}; \mathcal{L}; \sigma_1(\Phi); \sigma_1(\underline{\mathcal{R}}') \vdash [\mathrm{id}, \sigma_1] \Leftarrow (\mathcal{C}, \mathcal{C}_1; \mathcal{L}; \underline{\mathcal{R}}')$, and $\sigma_1(\Phi) = \Phi$ (because Φ is well-scoped / does not contain any variables from $\mathcal{C}_1$) we have $\mathcal{C}; \mathcal{L}; \Phi \vdash \sigma_1(h'') \Leftarrow \sigma_1(\underline{\mathcal{R}}'')$ and $\mathcal{C}; \mathcal{L}; \Phi; \sigma_1(\underline{\mathcal{R}}') \vdash \sigma_1(\sigma_2) \Leftarrow (\mathcal{C}_2; \mathcal{L}_2; \sigma_1(\mathcal{R}_2))$ (by lemma B7.3 (Substitution)).

Let: $h' = \sigma_1(h'')$, $\sigma = [\sigma_1, \sigma_2]$ to step with Subs_Pat_Ret_Comp.
$\underline{\mathcal{R}}' = \sigma_1(\underline{\mathcal{R}}'')$.
So $\mathcal{C}; \mathcal{L}; \Phi \vdash h' \Leftarrow \underline{\mathcal{R}}'$
and $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}}' \vdash \sigma \Leftarrow (\mathcal{C}_1, \mathcal{C}_2; \mathcal{L}_2; \mathcal{R}_2)$ hold by lemma B6 (Weakening) and Subs_Chk_Concat.

Assume: 1. $\mathit{ident\_or\_pat}{:}\beta \rightsquigarrow \mathcal{C}_1\, \mathtt{with}\, \mathit{term}_1$
2. $\mathit{ident\_or\_pat}$ is exhaustive

3. $\cdot \vdash \mathit{pval} \Rightarrow \beta$

PROVE: 1. $\mathit{term}_1 \sim \mathit{pval}$
2. $\exists \sigma.\ \mathit{ident\_or\_pat} = \mathit{pval} \rightsquigarrow \sigma$ and $\cdot; \cdot; \cdot; \cdot \vdash \sigma \Leftarrow (\mathcal{C}_1; \cdot; \cdot)$.

⟨2⟩1. CASE: PAT_COMP_NO_SYM_ANNOT
PROOF: term_1 is a wildcard (fresh variable) which would unfiy with pval; let $\sigma = \cdot$ for SUBS_PAT_VALUE_NO_SYM_ANNOT / SUBS_CHK_EMPTY.

⟨2⟩2. CASE: PAT_COMP_SYM_ANNOT, PAT_SYM_OR_PAT_SYM
PROOF: $\mathit{term}_1 = x$, a fresh pattern variable, so would unify with pval;
let $\sigma = \mathit{pval} \ \ \texttt{/} \ \ x$ for SUBS_PAT_VALUE_SYM_ANNOT / SUBS_CHK_COMP (using $\cdot \vdash \mathit{pval} \Rightarrow \beta$).

⟨2⟩3. CASE: PAT_COMP_NIL
PROOF: $\mathit{term}_1 = \texttt{nil}$, and by inversion on the typing assumption, and then by exhaustiveness, $\mathit{pval} = \texttt{Nil}\,\beta(\,)$, so would unify; let $\sigma = \cdot$ for SUBS_PAT_VALUE_NIL / SUBS_CHK_EMPTY.

⟨2⟩4. CASE: PAT_COMP_CONS
PROOF: $\mathit{term}_1 = \mathit{term}_{11} :: \mathit{term}_{12}$, and by inversion on the typing assumption, and then by exhaustiveness, $\mathit{pval} = \texttt{Cons}(\mathit{pval}_1, \mathit{pval}_2)$. By induction (1) they would unify and (2) let $\sigma = [\sigma_1, \sigma_2]$ for SUBS_PAT_VALUE_CONS / SUBS_CHK_COMP and SUBS_CHK_CONCAT (both are independent).

⟨2⟩5. CASE: PAT_COMP_TUPLE
PROOF: $\mathit{term}_1 = (\,\overline{\mathit{term}_i}^{\,i}\,)$, and by inversion on the typing assumption,
$\mathit{pval} = \texttt{Tuple}(\,\overline{\mathit{pval}_i}^{\,i}\,)$. By induction (1) they would unify
(2) let $\sigma = [\,\overline{\sigma_i}^{\,i}\,]$ for SUBS_PAT_VALUE_TUPLE /
SubsChkComp and SUBS_CHK_CONCAT.

⟨2⟩6. CASE: PAT_COMP_ARRAY
PROOF: Similar to PAT_COMP_TUPLE, but with SUBS_PAT_VALUE_ARRAY.

⟨2⟩7. CASE: PAT_COMP_SPECIFIED
PROOF: By induction we have (1) $\mathit{term}_1 \sim \mathit{pval}$, and by the `Specified` exception (see Section B4, Proof Judgements) $\mathit{term}_1 \sim \texttt{Specified}(\mathit{pval})$;
σ for SUBS_PAT_VALUE_SPECIFIED, typing by induction.

⟨2⟩8. CASE: PAT_SYM_OR_PAT_PAT
PROOF: By induction.

⟨1⟩3. CASE: PAT_RET_LOG
$\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{log}\, y', \overline{\mathit{ret\_pat}_j}^{\,j} {:} \exists\, y{:}\beta.\ \mathit{ret} \rightsquigarrow \mathcal{C}_2; y'{:}\beta, \mathcal{L}_2; \Phi_2; \mathcal{R}_2$
which means $\mathit{fun} = \forall\, x{:}\beta.\ \mathit{fun}'$ (by inversion, REL_RET_LOG)
and so $\cdot; \cdot; \Phi; \underline{\mathcal{R}} \vdash \mathit{oarg}, \overline{\mathit{ret\_term}_j}^{\,j} :: \forall\, x{:}\beta.\ \mathit{fun}' \gg \texttt{I}$ (by inversion, EXPL_SPINE_LOG).

$\mathcal{C}; \mathcal{L}, y{:}\beta; \Phi \vdash \overline{\mathit{ret\_pat}_j}^{\,j} {:} \mathit{ret}' \rightsquigarrow \mathcal{C}_2; \mathcal{L}_2; \Phi_2; \mathcal{R}_2$ (from the pattern-matching assumption),

$\forall\, oarg \sim oarg'.\ \Phi \vdash oarg/x(fun') \sim oarg'/y'(ret')$ (from the related assumption),
$\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \overline{ret\_term_j}^{\,j} :: oarg/x(fun') \gg \mathtt{I}$ (from the spine typing assumption)
and $\Phi \vdash h \Leftarrow \underline{\mathcal{R}}$, imply $\langle h + f;\ \overline{ret\_pat_j = ret\_term_j}^{\,j}\ \rangle \rightsquigarrow \langle h'' + f;\ \sigma\rangle$
and $\exists \underline{\mathcal{R}}'$ such that $\mathcal{C};\mathcal{L}, y'{:}\beta;\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}''$ and $\mathcal{C};\mathcal{L}, y'{:}\beta;\Phi;\underline{\mathcal{R}}'' \vdash \sigma_2 \Leftarrow (\mathcal{C}_2;\mathcal{L}_2;\mathcal{R}_2)$ (by induction).

Since $\cdot;\cdot \vdash oarg \Rightarrow \beta$, and $oarg/y'(\Phi) = \Phi$ (because it is well-scoped / doesn't refer to y')
and $\mathcal{C};\mathcal{L}; oarg/y'(\Phi); oarg/y'(\underline{\mathcal{R}}') \vdash [\mathrm{id}, oarg/y'] \Leftarrow (\mathcal{C};\mathcal{L}, y'{:}\beta;\underline{\mathcal{R}}')$,
we have $\mathcal{C};\mathcal{L};\Phi \vdash oarg/y'(h'') \Leftarrow oarg/y'(\underline{\mathcal{R}}'')$, and
$\mathcal{C};\mathcal{L};\Phi; oarg/y'(\underline{\mathcal{R}}'') \vdash oarg/y'(\sigma_2) \Leftarrow (\mathcal{C}_2;\mathcal{L}_2; oarg/y'(\mathcal{R}_2))$ (by lemma B7.3 (Substitution)).

LET: $h' = oarg/y'(h'')$, $\sigma = [oarg/y', \sigma_2]$ to step with SUBS_PAT_RET_LOG.
$\underline{\mathcal{R}}' = oarg/y'(\underline{\mathcal{R}}'')$.
So $\mathcal{C};\mathcal{L};\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}'$
and $\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}}' \vdash \sigma \Leftarrow (\mathcal{C}_2; y'{:}\beta, \mathcal{L}_2;\mathcal{R}_2)$ by SUBS_CHK_CONCAT.

⟨1⟩4. CASE: PAT_RET_PHI
$\mathcal{C};\mathcal{L};\Phi \vdash \overline{ret\_pat_i}^{\,i}{:}term' \wedge ret' \rightsquigarrow \mathcal{C}';\mathcal{L}';\Phi', term';\mathcal{R}'$
which means $fun = term \supset fun'$ (by inversion, REL_RET_PHI),
and so $\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \overline{ret\_term_j}^{\,j} :: term \twoheadrightarrow fun' \gg \mathtt{I}$ (by inversion, EXPL_SPINE_RES).

$\mathcal{C};\mathcal{L};\Phi \vdash \overline{ret\_pat_i}^{\,i}{:}ret' \rightsquigarrow \mathcal{C}';\mathcal{L}';\Phi', term';\mathcal{R}'$ (from the pattern-matching assumption)
$\Phi \vdash fun' \sim ret'$ (from the related assumption),
$\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \overline{ret\_term_j}^{\,j} :: fun' \gg \mathtt{I}$ (from the spine typing assumption)
imply $\langle h + f;\ \overline{ret\_pat_i = ret\_term_i}^{\,i}\ \rangle \rightsquigarrow \langle h' + f;\ \sigma\rangle$ and the heap and substitution typings (by induction).

⟨1⟩5. CASE: PAT_RET_RES
$\mathcal{C};\mathcal{L};\Phi \vdash \mathtt{res}\ res\_pat, ret\_pat{:}res' * ret' \rightsquigarrow \mathcal{C}_4;\mathcal{L}_3,\mathcal{L}_4;\Phi_3,\Phi_4;\mathcal{R}_3,\mathcal{R}_4$
which means $fun = res \twoheadrightarrow fun'$ (by inversion, REL_RET_RES),
and so $\cdot;\cdot;\Phi;\underline{\mathcal{R}}_1,\underline{\mathcal{R}}_2 \vdash res\_term, spine :: res \twoheadrightarrow fun \gg \mathtt{I}$ (by inversion, EXPL_SPINE_RES),
and $h = h_1 + h_2$ where $\Phi \vdash h_1 \Leftarrow \underline{\mathcal{R}}_1$ and $\Phi \vdash h_2 \Leftarrow \underline{\mathcal{R}}_2$ (by lemma B8.4).

$\cdot;\cdot;\cdot;\underline{\mathcal{R}}_1 \vdash res\_term \Leftarrow res$ (from the spine typing assumption)
and $\Phi \vdash h_1 \Leftarrow \underline{\mathcal{R}}_1$, imply $\exists res\_val, \underline{\mathcal{R}}_1', h_1'$ such that $\Phi \vdash h_1' \Leftarrow \underline{\mathcal{R}}_1'$, $\cdot;\cdot;\Phi;\underline{\mathcal{R}}_1' \vdash res\_val \Leftarrow res'$
and $\langle h_1 + h_2 + f; res\_term\rangle \Downarrow \langle h_1' + h_2 + f; res\_val\rangle$ (by lemma B8.6 (Progress and type preservation for resource terms)).

$\mathcal{L};\Phi \vdash res\_pat{:}res' \rightsquigarrow \mathcal{L}_3;\Phi_3;\mathcal{R}_3$ (from the pattern matching assumption),
$\Phi \vdash res \sim res'$ (from the related assumption),
$\cdot;\cdot;\Phi;\underline{\mathcal{R}}_1' \vdash res\_val \Leftarrow res$ and $\Phi \vdash h_1' \Leftarrow \underline{\mathcal{R}}_1'$, imply
$\exists h_1'', \underline{\mathcal{R}}_1''$ such that $\langle h_1' + h_2 + f; res\_pat = res\_val\rangle \rightsquigarrow \langle h_1'' + h_2 + f; \sigma_1\rangle$ $\mathcal{C};\mathcal{L};\Phi \vdash h_1'' \Leftarrow \underline{\mathcal{R}}_1''$,
and $\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}}_1'' \vdash \sigma_1 \Leftarrow (\cdot;\mathcal{L}_3;\mathcal{R}_3)$ (by the nested proof below).

$\mathcal{C};\mathcal{L};\Phi \vdash \overline{ret\_pat_j}^{\,j}{:}ret' \rightsquigarrow \mathcal{C}_4;\mathcal{L}_4;\Phi_4;\mathcal{R}_4$ (from the pattern matching assumption),
$\Phi \vdash fun' \sim ret'$ (from the related assumption),

$\cdot;\cdot;\Phi;\underline{\mathcal{R}}_2 \vdash \overline{ret\_term_j}^{\,j} :: fun' \gg \mathtt{I}$ (from the spine typing assumption), and $\Phi \vdash h_2 \Leftarrow \underline{\mathcal{R}}_2$, imply $\langle h_2 + h_1'' + f;\ \overline{ret\_pat_j = ret\_term_j}^{\,j}\ \rangle \rightsquigarrow \langle h_2' + h_1'' + f; \sigma_2\rangle$ and $\exists \underline{\mathcal{R}}_2'$ such that $\mathcal{C};\mathcal{L};\Phi \vdash h_2' \Leftarrow \underline{\mathcal{R}}_2'$, and $\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}}_2' \vdash \sigma_2 \Leftarrow (\mathcal{C}_4;\mathcal{L}_4;\mathcal{R}_4)$ (by induction).

LET: $h' = h_1'' + h_2'$
$\sigma = [\sigma_1, \sigma_2]$ to step with SUBS_PAT_RET_RES.
$\underline{\mathcal{R}}' = \underline{\mathcal{R}}_1', \underline{\mathcal{R}}_2'$.
So $\mathcal{C};\mathcal{L};\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}'$ by HEAP_CONCAT
and $\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}}' \vdash \sigma \Leftarrow (\mathcal{C}_4;\mathcal{L}_3,\mathcal{L}_4;\mathcal{R}_3,\mathcal{R}_4)$ by SUBS_CHK_CONCAT
(because $\underline{\mathcal{R}}_2'$ is well-formed w.r.t. $\mathcal{C};\mathcal{L}$, it does not contain any variables from $\mathcal{L}_3;\mathcal{R}_3$ so $\sigma_1(\underline{\mathcal{R}}_2') = \underline{\mathcal{R}}_2'$).

ASSUME: 1. $\mathcal{L};\Phi \vdash res\_pat{:}res' \rightsquigarrow \mathcal{L}';\Phi';\mathcal{R}'$
2. $\Phi \vdash res \sim res'$
3. $\exists \underline{\mathcal{R}}.(\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash res\_val \Leftarrow res) \land (\mathcal{C};\mathcal{L};\Phi \vdash h \Leftarrow \underline{\mathcal{R}})$

PROVE: $\exists h', \sigma.$
$\forall f.\ \langle h + f; res\_pat = res\_val\rangle \rightsquigarrow \langle h' + f; \sigma\rangle$
$\exists \underline{\mathcal{R}}'.\ \mathcal{C};\mathcal{L};\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}' \land \mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}}' \vdash \sigma \Leftarrow (\cdot;\mathcal{L}';\mathcal{R}')$.

⟨2⟩1. CASE: PAT_RES_MATCH_FOLD
$\mathcal{L};\Phi \vdash \mathtt{fold}\,(res\_pat'){:}\alpha(ptr', \overline{iarg_i'}^{\,i})(oarg') \rightsquigarrow \mathcal{L}';\Phi';\mathcal{R}'$
which means $res = \alpha(ptr, \overline{iarg_i}^{\,i})(oarg)$ (by inversion, REL_RES_PRED)
and so $\underline{\mathcal{R}} = \_{:}\alpha(ptr, \overline{iarg_i}^{\,i})(oarg)$ and $res\_val = \alpha(ptr_2, \overline{iarg_{2\,i}}^{\,i})$ (by lemma B8.3).

⟨3⟩1. $h = \{\alpha(ptr, \overline{iarg_i}^{\,i})(oarg)\ \&\ def\ \&\ heap\}$
PROOF: $\alpha \neq \mathtt{Owned}\langle\tau\rangle$ (from the pattern-matching assumption), and lemma B8.4 (Normalised resource context determines structure of heap).

⟨3⟩2. $\exists \underline{\mathcal{R}}_1'$.
1. $\alpha \equiv x_p{:}\mathtt{pointer}, \overline{x_i{:}\beta_i}^{\,i}, y{:}\mathtt{record}\,\overline{tag_j{:}\beta_j'}^{\,j} \mapsto res'' \in \mathtt{Globals}$
2. $\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}}_1' \vdash def \Leftarrow [oarg/y, [\,\overline{iarg/x_i}^{\,i}\,], ptr/x_p](res'')$
3. $\mathcal{C};\mathcal{L};\Phi \vdash heap \Leftarrow \underline{\mathcal{R}}_1'$
PROOF: By inversion, $\mathcal{C};\mathcal{L};\Phi \vdash h \Leftarrow \underline{\mathcal{R}}$ is HEAP_PRED_OTHER.

⟨3⟩3. $\mathcal{L};\Phi \vdash res\_pat'{:}[oarg'/y, [\,\overline{iarg_i'/x_i}^{\,i}\,], ptr'/x_p](res'') \rightsquigarrow \mathcal{L};\Phi;\mathcal{R}$
PROOF: By inversion on the pattern-matching assumption.

⟨3⟩4. $\Phi \vdash [oarg/y, [\,\overline{iarg/x_i}^{\,i}\,], ptr/x_p](res'') \sim [oarg'/y, [\,\overline{iarg_i'/x_i}^{\,i}\,], ptr'/x_p](res'')$
PROOF: By lemma B9.2, using $\Phi \vdash res'' \sim res''$ (by lemma B9.1) and $ptr \sim ptr'$, $\overline{iarg_i \sim iarg_i'}^{\,i}$ and $oarg \sim oarg'$.

⟨3⟩5. $\langle heap + f; res\_pat = res\_val\rangle \rightsquigarrow \langle h' + f; \sigma\rangle$
PROOF: By induction, using ⟨3⟩2, ⟨3⟩3 and ⟨3⟩4.

⟨3⟩6. Step with SUBS_PAT_RES_FOLD.

⟨3⟩7. $h', \underline{\mathcal{R}}'$ as given by induction.

⟨2⟩2. CASE: PAT_RES_MATCH_EMP / PAT_RES_MATCH_PHI
$res = \texttt{emp}$ or $term$ (by inversion, REL_RES_EMP / REL_RES_PHI) and so
$res\_val = \texttt{emp}$ or $\texttt{term}$ and $\underline{\mathcal{R}} = \cdot$ (by lemma B8.3), meaning $h = \cdot$ (by lemma B8.4).
PROOF: Let $h' = \cdot$ to step with SUBS_PAT_RES_EMP / SUBS_PAT_RES_PHI.
$\underline{\mathcal{R}}' = \cdot$, so HEAP_EMPTY and SUBS_CHK_EMPTYsuffice.

⟨2⟩3. CASE: PAT_RES_MATCH_IF_TRUE / PAT_RES_MATCH_IF_FALSE
Only showing true case, false case is symmetric.

$res' = \texttt{if}\ term'\ \texttt{then}\ res'_1\ \texttt{else}\ res'_2$ so
$res = \texttt{if}\ term\ \texttt{then}\ res_1\ \texttt{else}\ res_2$ (by inversion, REL_RES_IF).

Since $\texttt{smt}\,(\Phi \Rightarrow term')$ (from the pattern-matching assumption) and
$\texttt{smt}\,(\Phi \Rightarrow term \leftrightarrow term')$, we can conclude the typing assumption must be RES_CHK_IF_TRUE.

From there, we proceed by induction.

⟨2⟩4. CASE: PAT_RES_MATCH_VAR
PROOF: Let $h' = h$ to step with SUBS_PAT_RES_VAR.
$\underline{\mathcal{R}}' = \underline{\mathcal{R}}$ so SUBS_CHK_RES.

⟨2⟩5. CASE: PAT_RES_MATCH_SEPCONJ
$\mathcal{L}; \Phi \vdash \langle res\_pat_1, res\_pat_2\rangle{:}res'_1 * res'_2 \rightsquigarrow \mathcal{L}_1, \mathcal{L}_2; \Phi_1, \Phi_2; \mathcal{R}_1, \mathcal{R}_2$

$res = res_1 * res_2$ (by inversion, REL_RES_SEPCONJ) and
$\cdot;\cdot;\Phi;\underline{\mathcal{R}}_1,\underline{\mathcal{R}}_1 \vdash \langle res\_val_1, res\_val_2\rangle \Leftarrow res_1 * res_2$ (by lemma B8.3),
so $h = h_1 + h_2$ where $\Phi \vdash h \Leftarrow \underline{\mathcal{R}}_1$ and $\Phi \vdash h \Leftarrow \underline{\mathcal{R}}_2$.

By induction, obtain h'_1 and h'_2, and then let $h' = h'_1 + h'_2$. Instantiate the frame, from the inductive hypothesis with $h_2 + f$ and then $h'_1 + f$ to conclude
$\langle h_1 + h_2 + f; res\_pat_1 = res\_val_1\rangle \rightsquigarrow \langle h'_1 + h_2 + f; \sigma_1\rangle$ and
$\langle h_2 + h'_1 + f; res\_pat_2 = res\_val_2\rangle \rightsquigarrow \langle h'_2 + h'_1 + f; \sigma_2\rangle$ to step with SUBS_PAT_RES_PAIR.
LET: $\underline{\mathcal{R}}' = \underline{\mathcal{R}}'_1, \underline{\mathcal{R}}'_2$ (obtained from induction).
We then have and $\mathcal{C};\mathcal{L};\Phi \vdash h'_1 + h'_2 \Leftarrow \underline{\mathcal{R}}'$ and (since $\sigma_1(\mathcal{R}_2) = \mathcal{R}_2$ because it can not refer to $\mathcal{L}_1$) $\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}}' \vdash [\sigma_1, \sigma_2] \Leftarrow (\cdot;\mathcal{L}_1,\mathcal{L}_2;\mathcal{R}_1,\mathcal{R}_2)$.

⟨2⟩6. CASE: PAT_RES_MATCH_PACK
$\mathcal{L}; \Phi \vdash \texttt{pack}\,(x, res\_pat'){:}\exists\, y'{:}\beta.\ res'_1 \rightsquigarrow x{:}\beta, \mathcal{L}'; \Phi'; \mathcal{R}'$

$res = \exists\, y{:}\beta.\ res_1$ (by inversion, REL_RES_EXISTS) and
$\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \texttt{pack}\,(oarg, res\_val') \Leftarrow \exists\, y{:}\beta.\ res_1$ (by lemma B8.3).
$\mathcal{L}, x{:}\beta; \Phi \vdash res\_pat'{:}x/y'(res'_1) \rightsquigarrow \mathcal{L}'; \Phi'; \mathcal{R}'$ (from the pattern-matching assumption),
$\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash res\_val' \Leftarrow oarg/y(res_1)$ (from the typing assumption),
$\forall\, term \sim term'.\ \Phi \vdash term/y(res_1) \sim term'/y'(res'_1)$ (from the related assumption),
$oarg \sim x$ imply $\exists h'', \sigma', .\ \forall f \ldots$
and $\exists \underline{\mathcal{R}}''.\ \mathcal{C}; \mathcal{L}, x{:}\beta; \Phi \vdash h' \Leftarrow \underline{\mathcal{R}}' \wedge \mathcal{C}; x{:}\beta, \mathcal{L}; \Phi; \underline{\mathcal{R}}'' \vdash \sigma' \Leftarrow (\cdot; \mathcal{L}'; \mathcal{R}')$

Since $\cdot;\cdot \vdash oarg \Rightarrow \beta$, and $oarg/x(\Phi) = \Phi$ (because it is well-scoped / doesn't refer to

x) and $\mathcal{C}; \mathcal{L}; oarg/x(\Phi); oarg/x(\underline{\mathcal{R}}') \vdash [\mathrm{id}, oarg/x] \Leftarrow (\mathcal{C}; x{:}\beta, \mathcal{L}'; \underline{\mathcal{R}}')$,
we have $\mathcal{C}; \mathcal{L}; \Phi \vdash oarg/x(h'') \Leftarrow oarg/x(\underline{\mathcal{R}}'')$, and
$\mathcal{C}; \mathcal{L}; \Phi; oarg/x(\underline{\mathcal{R}}'') \vdash oarg/x(\sigma') \Leftarrow (\cdot; \mathcal{L}'; oarg/x(\mathcal{R}_2))$ (by lemma B7.3 (Substitution)).

LET: $h' = oarg/x(h'')$
$\sigma = [oarg/x, \sigma']$ to step with SUBS_PAT_RES_PACK.
$\underline{\mathcal{R}}' = oarg/x(\underline{\mathcal{R}}'')$ so $\mathcal{C}; \mathcal{L}; \Phi; oarg/x(\underline{\mathcal{R}}'') \vdash [oarg/x, \sigma'] \Leftarrow (\cdot; \mathcal{L}'; \mathcal{R}_2)$ by SUBS_CHK_CONCAT.

B9.5 $\Phi \vdash \mathtt{to\_fun}\, ret \sim ret$

PROOF SKETCH: Induction over *ret*.

B9.6 Statement and proof

ASSUME: 1. Closed (no free-variables) expression *texpr*.
2. $\cdot; \cdot; \Phi; \underline{\mathcal{R}} \vdash texpr \Leftarrow ret$
3. All patterns in *texpr* are exhaustive.

PROVE: Either *texpr* is a value *tval*, or it is unreachable, or
$\forall h, \underline{\mathcal{R}}.\ (\Phi \vdash h \Leftarrow \underline{\mathcal{R}}) \Rightarrow \exists h', texpr'.\ \langle h; texpr\rangle \longrightarrow \langle h'; texpr'\rangle$.

PROOF SKETCH: Induction over the typing rules.

⟨1⟩1. CASE: Value typing rules (see B5.3).
PROOF: All these judgements/rules give types to syntactic values; and there are no operational rules corresponding to them (see Section B3).

⟨1⟩2. CASE: PURE_TOP_VAL_UNDEF, PURE_TOP_VAL_ERROR,EXPL_TOP_VAL_UNDEF, EXPL_TOP_VAL_ERROR.
PROOF: All these rules require inconsistent constraint context, and so would be unreachable.

⟨1⟩3. CASE: PURE_EXPR_ARRAY_SHIFT.
PROOF: By inversion on $\cdot \vdash pval_1 \Rightarrow \mathtt{pointer}$, $pval_1$ must be a *mem_ptr* (PURE_VAL_OBJ_PTR). Similarly $pval_2$ must be a *mem_int*, so step with PE_TP_ARRAY_SHIFT.

⟨1⟩4. CASE: PURE_EXPR_MEMBER_SHIFT.
PROOF: *pval* must be a *mem_ptr* so step with PE_TP_MEMBER_SHIFT.

⟨1⟩5. CASE: PURE_EXPR_NOT.
PROOF: *pval* must be a *bool_value* so step with PE_TP_NOT_TRUE or PE_TP_NOT_FALSE.

⟨1⟩6. CASE: PURE_EXPR_ARITH_BINOP, PURE_EXPR_REL_BINOP.
PROOF: $pval_1$ and $pval_2$ must be *mem_ints*, so step with PE_TP_ARITH_BINOP or PE_TP_REL_BINOP respectively.

⟨1⟩7. CASE: PURE_EXPR_BOOL_BINOP.
PROOF: $pval_1$ and $pval_2$ must be *bool_values*, so step with PE_TP_BOOL_BINOP.

⟨1⟩8. CASE: PURE_EXPR_CALL.

⟨2⟩1. 1. $name{:}pure\_fun \equiv \overline{x_i}^{\,i} \mapsto tpexpr \in \texttt{Globals}$.
2. $\cdot;\cdot;\Phi;\cdot \vdash \overline{pval_i}^{\,i} :: pure\_fun \gg \Sigma\, y{:}\beta.\ term \wedge \mathtt{I}$.
PROOF: By inversion on the assumption.

⟨2⟩2. $\langle \cdot;\ \overline{x_i = pval_i}^{\,i}\ \rangle :: pure\_fun \gg \langle \cdot;\ \sigma;\ \Sigma\, y{:}\beta.\ term \wedge \mathtt{I} \rangle$.
PROOF: By lemma B9.3.

⟨2⟩3. Thus it can step with PE_TP_CALL.

⟨1⟩9. CASE: PURE_EXPR_ASSERT_UNDEF.

⟨2⟩1. *pval* must be a *bool_value* PROOF: By PURE_VAL_TRUE, PURE_VAL_FALSE.

⟨2⟩2. $\texttt{smt}\,(\Phi \Rightarrow pval)$. PROOF: By inversion on the assumption.

⟨2⟩3. If it is `False`, then by the latter, we have an inconsistent constraints context, meaning the code is unreachable.

⟨2⟩4. If it is `True`, we may step with PE_TP_ASSERT_UNDEF.

⟨1⟩10. CASE: PURE_EXPR_BOOL_TO_INTEGER.
PROOF: *pval* must be a *bool_value* (PURE_VAL_TRUE, PURE_VAL_FALSE) and so step with PE_TP_BOOL_TO_INTEGER_TRUE, PE_TP_BOOL_TO_INTEGER_FALSE respectively.

⟨1⟩11. CASE: PURE_EXPR_WRAPI.
PROOF: *pval* must be a *mem_int* (PURE_VAL_OBJ_PTR) and so step with PE_TP_WRAPI.

⟨1⟩12. CASE: PURE_TOP_IF, PURE_TOP_CASE, PURE_TOP_LET, PURE_TOP_LETT.
PROOF: See EXPL_TOP_SEQ_IF, EXPL_TOP_SEQ_CASE, EXPL_TOP_SEQ_LET, EXPL_TOP_SEQ_LETT, case for more general proofs.

⟨1⟩13. CASE: EXPL_IS_ACTION_CREATE.

⟨2⟩1. *pval* must be a *mem_int*.
PROOF: By PURE_VAL_OBJ_PTR.

⟨2⟩2. h must be $\cdot$ (empty).
PROOF: By HEAP_EMPTY.

⟨2⟩3. Step with ACTION_IS_CREATE.
PROOF: *mem_ptr* is free in the premises and so can be constructed to satisfy the requirements.

⟨1⟩14. CASE: EXPL_IS_ACTION_LOAD.

$\langle 2\rangle 1.$ $pval_0$ must be a mem_ptr.
PROOF: By PURE_VAL_OBJ_PTR.

$\langle 2\rangle 2.$ $\cdot;\cdot;\Phi;\underline{\mathcal{R}}' \vdash res\_term \Rightarrow term \overset{init}{\mapsto}_\tau pval_1$.
$\mathtt{smt}\,(\Phi \Rightarrow (term = mem\_ptr) \wedge (init = \mathtt{const}_\tau\mathtt{true}))$.
PROOF: By inversion on the typing assumption and $\langle 2\rangle 1$.

$\langle 2\rangle 3.$ $\exists h', \underline{\mathcal{R}}', res\_val$.
1. $\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}'$
2. $\langle h; res\_term\rangle \Downarrow \langle h'; res\_val\rangle$
3. $\cdot;\cdot;\Phi;\underline{\mathcal{R}}' \vdash res\_val \Rightarrow term \overset{init}{\mapsto}_\tau pval_1$
PROOF: By $\langle 2\rangle 2$ and lemma B8.6 (Progress and type preservation for resource terms).

$\langle 2\rangle 4.$ $res\_val = \mathtt{Owned}\,\langle\tau\rangle(term)$.
PROOF: By lemma B8.3 (Non-conditional resources determine context and values).

$\langle 2\rangle 5.$ $h' = \{term \overset{init}{\mapsto}_\tau pval_1 \,\&\, \mathtt{None}\}$.
PROOF: By inversion on the term typing assumption in $\langle 2\rangle 3$ using $\langle 2\rangle 4$, $\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}'$ and lemma B8.4 (Normalised resource context determines structure of heap).

$\langle 2\rangle 6.$ Step with ACTION_IS_LOAD.

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

$\langle 2\rangle 1.$ $pval_0$ must both be a mem_ptr.
PROOF: By PURE_VAL_OBJ_PTR.

$\langle 2\rangle 2.$ $\mathtt{smt}\,(\Phi \Rightarrow \mathtt{representable}\,(\tau, pval_1))$
$\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash res\_term \Rightarrow term \overset{-}{\mapsto}_\tau \_$
$\mathtt{smt}\,(\Phi \Rightarrow term = mem\_ptr)$.
PROOF: By inversion on the typing assumption and $\langle 2\rangle 1$.

$\langle 2\rangle 3.$ $\exists h', \underline{\mathcal{R}}', res\_val$.
1. $\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}'$
2. $\langle h; res\_term\rangle \Downarrow \langle h'; res\_val\rangle$
3. $\cdot;\cdot;\cdot;\underline{\mathcal{R}}' \vdash res\_val \Rightarrow term \overset{-}{\mapsto}_\tau \_$.
PROOF: By $\langle 2\rangle 2$ and lemma B8.6 (Progress and type preservation for resource terms).

$\langle 2\rangle 4.$ $res\_val = \mathtt{Owned}\,\langle\tau\rangle(term)$.
PROOF: By lemma B8.3 (Non-conditional resources determine context and values).

$\langle 2\rangle 5.$ $h' = \{term \overset{-}{\mapsto}_\tau \_ \,\&\, \mathtt{None}\}$.
PROOF: By inversion on the term typing assumption in $\langle 2\rangle 3$, $\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}'$ and lemma B8.4 (Normalised resource context determines structure of heap).

$\langle 2\rangle 6.$ Step with ACTION_IS_STORE.

$\langle 1\rangle 16.$ CASE: EXPL_IS_ACTION_KILL_STATIC

⟨2⟩1. *pval* must be a *mem_ptr*.
PROOF: By PURE_VAL_OBJ_PTR.

⟨2⟩2. $\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \mathit{res\_term} \Rightarrow \mathit{term} \overset{\bar{\ }}{\mapsto}_\tau \_$
$\mathtt{smt}\,(\Phi \Rightarrow \mathit{term} = \mathit{mem\_ptr})$.
PROOF: By inversion on the typing assumption and ⟨2⟩1.

⟨2⟩3. $\exists h', \underline{\mathcal{R}}', \mathit{res\_val}$.
1. $\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}'$
2. $\langle h; \mathit{res\_term}\rangle \Downarrow \langle h'; \mathit{res\_val}\rangle$
3. $\cdot;\cdot;\Phi;\underline{\mathcal{R}}' \vdash \mathit{res\_val} \Rightarrow \mathit{term} \overset{\bar{\ }}{\mapsto}_\tau \_$.
PROOF: By ⟨2⟩2 and lemma B8.6 (Progress and type preservation for resource terms).

⟨2⟩4. $\mathit{res\_val} = \mathtt{Owned}\,\langle\tau\rangle(\mathit{term})$.
PROOF: By lemma B8.3 (Non-conditional resources determine context and values).

⟨2⟩5. $h' = \{\mathit{term} \overset{\bar{\ }}{\mapsto}_\tau \_ \,\&\, \mathtt{None}\}$.
PROOF: By inversion on the typing assumption in ⟨2⟩3, $\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}'$ and lemma B8.4 (Normalised resource context determines structure of heap).

⟨2⟩6. Step with ACTION_IS_KILL_STATIC.

⟨1⟩17. CASE: EXPL_IS_MEMOP_REL_BINOP.
PROOF: Similar to PURE_EXPR_REL_BINOP, but step with MEMOP_IS_REL_BINOP.

⟨1⟩18. CASE: EXPL_IS_MEMOP_INTFROMPTR.
PROOF: *pval* must be a *mem_ptr*, so step with MEMOP_IS_INTFROMPTR.

⟨1⟩19. CASE: EXPL_IS_MEMOP_PTRFROMINT.
PROOF: *pval* must be a *mem_int*, so step with MEMOP_IS_PTRFROMINT.

⟨1⟩20. CASE: EXPL_IS_MEMOP_PTRVALIDFORDEREF.

⟨2⟩1. *pval* must be a *mem_ptr*.
PROOF: By PURE_VAL_OBJ_PTR.

⟨2⟩2. $\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \mathit{res\_term} \Rightarrow \mathit{term} \overset{\mathit{init}}{\mapsto}_\tau \_$.
$\mathtt{smt}\,(\Phi \Rightarrow (\mathit{term} = \mathit{mem\_ptr}) \wedge (\mathit{init} = \mathtt{const}_\tau\mathtt{true}))$.
PROOF: By inversion on the typing assumption and ⟨2⟩1.

⟨2⟩3. $\exists h'\underline{\mathcal{R}}', \mathit{res\_val}$.
1. $\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}'$
2. $\langle h; \mathit{res\_term}\rangle \Downarrow \langle h'; \mathit{res\_val}\rangle$
3. $\cdot;\cdot;\Phi;\underline{\mathcal{R}}' \vdash \mathit{res\_val} \Rightarrow \mathit{term} \overset{\mathit{init}}{\mapsto}_\tau \_$.
PROOF: By ⟨2⟩2 and lemma B8.6 (Progress and type preservation for resource terms).

⟨2⟩4. $\mathit{res\_val} = \mathtt{Owned}\,\langle\tau\rangle(\mathit{term})$.

PROOF: By lemma B8.3 (Non-conditional resources determine context and values).

⟨2⟩5. $h' = \{term \overset{init}{\mapsto}_{\tau}\ \_\ \&\ \texttt{None}\}$.
PROOF: By inversion on the typing assumption in ⟨2⟩3 using ⟨2⟩4, $\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}'$ and lemma B8.4 (Normalised resource context determines structure of heap).

⟨2⟩6. Step with MEMOP_IS_PTRVALIDFORDEREF.

⟨1⟩21. CASE: EXPL_IS_MEMOP_PTRWELLALIGNED.
PROOF: *pval* must be a *mem_ptr*, so step with MEMOP_IS_PTRWELLALIGNED.

⟨1⟩22. CASE: EXPL_IS_MEMOP_PTRARRAYSHIFT.
PROOF: $pval_1$ must be a *mem_ptr* and $pval_2$ must be a *mem_int*, so step with MEMOP_IS_PTRARRAYSHIFT.

⟨1⟩23. CASE: EXPL_SEQ_CCALL.

⟨2⟩1. $ident{:}fun\ \equiv\ \overline{x_i}^{\,i}\ \mapsto\ texpr\ \in\ \texttt{Globals}$
$\cdot; \cdot; \Phi; \underline{\mathcal{R}} \vdash \overline{spine\_elem_i}^{\,i} :: fun \gg ret$.
PROOF: By inversion.

⟨2⟩2. $\langle h;\ \overline{x_i = spine\_elem_i}^{\,i}\ \rangle :: fun \gg \langle h'; \sigma_2; ret\rangle$.
PROOF: By ⟨2⟩1 and lemma B9.3 (Well-typed spines produce substitutions and the same return type).

⟨2⟩3. Step with SEQ_T_CCALL.

⟨1⟩24. CASE: EXPL_SEQ_PROC, EXPL_TOP_SEQ_RUN.
PROOF: Similar to EXPL_SEQ_CCALL, but step with SEQ_T_PROC / TSEQ_T_RUN.

⟨1⟩25. CASE: EXPL_IS_MEMOP.
PROOF: By induction, if *memop* is unreachable, then the whole expression is so. *memops* are not values. Only stepping cases applies, so step with IS_IS_MEMOP.

⟨1⟩26. CASE: EXPL_IS_ACTION, EXPL_IS_NEG_ACTION.
PROOF: By induction, if *action* is unreachable, then the whole expression is so. *actions* are not values. Only stepping case applies, so step with IS_IS_ACTION (or IS_IS_NEG_ACTION respectively).

⟨1⟩27. CASE: EXPL_TOP_SEQ_LETP, EXPL_TOP_SEQ_LETTP.
PROOF: See EXPL_TOP_SEQ_LET / EXPL_TOP_SEQ_LETT for more general cases and proofs.

⟨1⟩28. CASE: EXPL_TOP_SEQ_LET.
PROOF: By induction, since *seq_expr* is not value, if it is unreachable, the whole expression is so. If *seq_expr* takes a step, the whole expression steps with TSEQ_T_LET_LETT.

⟨1⟩29. CASE: EXPL_TOP_SEQ_LETT.

PROOF: By induction, if *texpr* is unreachable, so is the whole expression.

If if it a *tval*, use lemma B9.4 (Well-typed values pattern-match successfully), with lemma B9.5 ($\Phi \vdash \texttt{to\_fun}\, ret \sim ret$) and the assumption that all patterns are exhaustive, so the whole expression steps with TSEQ_T_LETT_SUB.

If *texpr* takes a step, the whole expression steps with TSEQ_T_LETT_LETT.

⟨1⟩30. CASE: EXPL_TOP_SEQ_CASE.
PROOF: By assumption that all patterns are exhaustive, and lemma B9.4 (Well-typed values pattern-match successfully), there is at least one pattern against which *pval* will match, so TSEQ_T_CASE.

⟨1⟩31. CASE: EXPL_TOP_SEQ_IF.
PROOF: *pval* must be a *bool_value* and so TSEQ_T_IF_TRUE/ TSEQ_T_IF_FALSE.

⟨1⟩32. CASE: EXPL_TOP_SEQ_BOUND.
PROOF: Step with TSEQ_T_BOUND.

⟨1⟩33. CASE: EXPL_TOP_IS_LETS.
PROOF: Similar to EXPL_TOP_SEQ_LETT, but step with TIS_T_LETS_SUB / TIS_T_LETS_LETSinstead.

⟨1⟩34. CASE: EXPL_TOP_SEQ, EXPL_TOP_IS.
PROOF: Step with T_T_TSEQ_T / T_T_TIS_T respectively.

B10 Type Preservation

B10.1 $\texttt{Owned}\,\langle\tau\rangle$ resource output values have type β_τ

If $\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \texttt{Owned}\,\langle\tau\rangle(\mathit{ptr}) \Leftarrow \mathit{ptr}' \overset{\mathit{init}}{\mapsto_\tau} \mathit{pval}$ then $\mathcal{C} \vdash \mathit{pval} \Rightarrow \beta_\tau$ and $\mathcal{C};\mathcal{L} \vdash \mathit{init} \Rightarrow \mathit{bool}_\tau$.

PROOF SKETCH: Induction over the typing judgements. Only EXPL_IS_ACTION_STORE constrain $\_.\mathit{value}$ of $\texttt{Owned}\,\langle\tau\rangle$ resources, and its premises ensure it has type β_τ; EXPL_IS_ACTION_LOAD and EXPL_IS_MEMOP_PTRVALIDFORDEREF simply propagate the value. EXPL_IS_ACTION_CREATE, EXPL_IS_ACTION_LOAD and EXPL_IS_ACTION_STORE and ensure $\_.\mathit{init}$ has type bool_τ.

B10.2 Type Preservation Statement and Proof

If $\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \mathit{texpr} \Leftarrow \mathit{ret}$ and $\Phi \vdash h \Leftarrow \underline{\mathcal{R}}$, and all top-level functions are well-typed[1] then $\forall f.\, \langle h + f; \mathit{texpr}\rangle \longrightarrow \langle \mathit{heap}; \mathit{texpr}'\rangle \Rightarrow \exists \Phi', h', \underline{\mathcal{R}}'.\, (\cdot;\cdot;\Phi;\cdot \sqsubseteq \cdot;\cdot;\Phi';\cdot) \wedge \mathit{heap} = h' + f \wedge (\Phi' \vdash h' \Leftarrow \underline{\mathcal{R}}') \wedge (\cdot;\cdot;\Phi';\underline{\mathcal{R}}' \vdash \mathit{texpr}' \Leftarrow \mathit{ret})$.

You can equally well prove $\forall \underline{\mathcal{R}}'.\, \Phi \vdash h' \Leftarrow \underline{\mathcal{R}}' \Rightarrow \cdot;\cdot;\Phi;\underline{\mathcal{R}}' \vdash \mathit{texpr}' \Leftarrow \mathit{ret}$ instead. Instead of supplying $\underline{\mathcal{R}}'$ and proving heap typing, you instead invert heap typing to deduce that $\underline{\mathcal{R}}'$ can only be what you would have supplied anyways.

It's worth noting that the constraint context will always only contain trivially true constraints (since $\mathcal{C};\mathcal{L}$ are both empty, all the *terms* in $\Phi;\mathcal{R}$ will be closed). This does not, by itself, guarantee that all conditional resources will be determined (e.g. $\texttt{if}\,\texttt{default}\,\texttt{bool}\,\texttt{then}\,\mathit{res}_1\,\texttt{else}\,\mathit{res}_2$), but there are other ways of excluding this (not allowing under-determined in heaps).

PROOF SKETCH: Induction over the typing rules, which don't refer to values or unreachable program points.

ASSUME: 1. $\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \mathit{texpr} \Leftarrow \mathit{ret}$,
2. $\Phi \vdash h \Leftarrow \underline{\mathcal{R}}$
3. all top-level functions are well-typed
4. $\forall f.\, \langle h + f; \mathit{texpr}\rangle \longrightarrow \langle \mathit{heap}; \mathit{texpr}'\rangle$

PROVE: $\exists \Phi', h', \underline{\mathcal{R}}'$.
1. $\cdot;\cdot;\Phi;\cdot \sqsubseteq \cdot;\cdot;\Phi';\cdot$
2. $\mathit{heap} = h' + f$
3. $\Phi' \vdash h' \Leftarrow \underline{\mathcal{R}}'$
4. $\cdot;\cdot;\Phi';\underline{\mathcal{R}}' \vdash \mathit{texpr}' \Leftarrow \mathit{ret}$.

$\langle 1 \rangle$1. CASE: PURE_EXPR_ARRAY_SHIFT.
For all pure expressions, $\Phi \vdash h \Leftarrow \cdot$, $h = \cdot$, $\mathit{heap} = f$.
LET: $h' = \cdot$ and $\underline{\mathcal{R}}' = \cdot$, so $\mathit{heap} = h' + f$ trivially and $\Phi \vdash \cdot \Leftarrow \cdot$ (by HEAP_EMPTY).
$\mathit{ret} = \Sigma\, y{:}\texttt{pointer}.\ y = \mathit{mem\_ptr} +_{\mathrm{ptr}} (\mathit{mem\_int} \times \mathrm{size\_of}(\tau)) \wedge \texttt{I}$

PROOF: By PURE_TOP_VAL_DONE, suffices to show $\cdot \vdash \mathit{mem\_ptr}' \Rightarrow \texttt{pointer}$ (true by

[1] More precisely, if $\mathit{ident}{:}\mathit{fun} \equiv \overline{x_i}^{\,i} \mapsto \mathit{texpr} \in \texttt{Globals}$ and $\overline{x_i}^{\,i} :: \mathit{fun} \rightsquigarrow \mathcal{C}'';\mathcal{L}'';\Phi'';\mathcal{R}'' \mid \mathit{ret}''$ then $\mathcal{C}'';\mathcal{L}'';\Phi'';\mathcal{R}'' \vdash \mathit{texpr} \Leftarrow \mathit{ret}''$.

PURE_VAL_OBJ_PTR) and $\texttt{smt}\,(\Phi \Rightarrow \mathit{mem\_ptr}' = \mathit{mem\_ptr} +_{\mathrm{ptr}} (\mathit{mem\_int} \times \mathrm{size\_of}(\tau)))$ (true by definition of PE_TP_ARRAY_SHIFT).

⟨1⟩2. CASE: PURE_EXPR_MEMBER_SHIFT, PURE_EXPR_NOT, PURE_EXPR_ARITH_BINOP, PURE_EXPR_BOOL_BINOP, PURE_EXPR_REL_BINOP, PURE_EXPR_ASSERT_UNDEF, PURE_EXPR_BOOL_TO_INTEGER, PURE_EXPR_WRAPI.
PROOF: Similar to PURE_EXPR_ARRAY_SHIFT.

⟨1⟩3. CASE: PURE_EXPR_CALL
PROOF: See EXPL_SEQ_CCALL for a more general case and proof.

⟨1⟩4. CASE: PURE_TOP_IF.
PROOF: See EXPL_TOP_SEQ_IF for a more general case and proof.

⟨1⟩5. CASE: PURE_TOP_LET.
PROOF: See EXPL_TOP_SEQ_LET for a more general case and proof.

⟨1⟩6. CASE: PURE_TOP_LETT.
PROOF: See EXPL_TOP_SEQ_LETT for a more general case and proof.

⟨1⟩7. CASE: PURE_TOP_CASE.
PROOF: See EXPL_TOP_SEQ_CASE for a more general case and proof.

⟨1⟩8. CASE: EXPL_IS_ACTION_CREATE.
LET: $\mathit{ret} = \Sigma\, y_p{:}\texttt{pointer}.\ \mathit{term} \wedge (y_p \overset{\texttt{const}_\tau\texttt{false}}{\mapsto_\tau} \texttt{default}\,\beta_\tau) * \texttt{I}$
where $\mathit{term} = \texttt{representable}\,(\tau*, y_p) \wedge \texttt{alignedI}\,(\mathit{mem\_int}, y_p)$.
$\mathit{pt} = \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr})(\mathit{oarg})$ where
$\mathit{oarg} = \{\mathit{init} = \texttt{const}_\tau\texttt{false}, \mathit{value} = \texttt{default}\,\beta_\tau\}$.

ASSUME: $\cdot;\cdot;\Phi;\cdot \vdash \texttt{create}\,(\mathit{mem\_int}, \tau) \Rightarrow \mathit{ret}$ and so $h = \cdot$ (by inversion, HEAP_EMPTY) and $\mathit{heap} = f + \{\mathit{pt}\ \&\ \texttt{None}\}$.
LET: $h' = \{\mathit{pt}\ \&\ \texttt{None}\}$, $\underline{\mathcal{R}}' = \_{:}\mathit{pt}$.
This means $\mathit{heap} = h' + f$ (trivially) and $\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}'$ (by HEAP_PRED_OWNED).

PROVE: $\cdot;\cdot;\Phi;\underline{\mathcal{R}}' \vdash \texttt{done}\,\langle \mathit{mem\_ptr}, \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr})\rangle{:}\mathit{ret} \Rightarrow \mathit{ret}$.

⟨2⟩1. $\cdot \vdash \mathit{mem\_ptr} \Rightarrow \texttt{pointer}$ by PURE_VAL_OBJ_PTR and PURE_VAL_OBJ.

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

⟨2⟩3. $\cdot;\cdot;\cdot;\underline{\mathcal{R}}' \vdash \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr}) \Leftarrow \mathit{pt}$ by RES_SYN_PRED.

⟨2⟩4. Prove typing with EXPL_SPINE_RET; ⟨2⟩3 – ⟨2⟩1 with EXPL_SPINE_RES, EXPL_SPINE_PHI, EXPL_SPINE_COMP respectively; EXPL_IS_TVAL.

⟨1⟩9. CASE: EXPL_IS_ACTION_LOAD.
LET: $\mathit{ret} = \Sigma\, y{:}\beta_\tau.\ y = \mathit{pval} \wedge (\mathit{mem\_ptr} \overset{\texttt{const}_\tau\texttt{true}}{\mapsto_\tau} \mathit{pval}) * \texttt{I}$

$pt = \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr})(\mathit{oarg})$ where $\mathit{oarg} = \{\mathit{init} = \texttt{const}_\tau\texttt{true}, \mathit{value} = \mathit{pval}\}$.

ASSUME: $\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \texttt{load}\,(\tau, \mathit{mem\_ptr}, \_, \mathit{res\_term}) \Rightarrow \mathit{ret}$
and $\mathit{heap} = \mathit{heap}' + \{pt \,\&\, \texttt{None}\}$ so
$\langle h + f; \mathit{res\_term}\rangle \Downarrow \langle \mathit{heap}' + \{pt \,\&\, \texttt{None}\}; \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr})\rangle$.

LET: h' and $\underline{\mathcal{R}}'$ be as per lemma B8.8 (Resource term reduction is isolated).
$\underline{\mathcal{R}}' = \_{:}pt$ by lemma B8.3 and $h' = \{pt \,\&\, \texttt{None}\}$ by lemma B8.4, hence $\mathit{heap}' = f$.
This means $\mathit{heap} = h' + f$, $\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}'$ and $\cdot;\cdot;\Phi;\underline{\mathcal{R}}' \vdash \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr}) \Rightarrow pt$.

PROVE: $\cdot;\cdot;\Phi;\underline{\mathcal{R}}' \vdash \texttt{done}\,\langle \mathit{pval}, \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr})\rangle{:}\mathit{ret} \Rightarrow \mathit{ret}$.

⟨2⟩1. $\cdot \vdash \mathit{pval} \Rightarrow \beta_\tau$ by lemma B10.1 (Owned $\langle\tau\rangle$ resource output values have type β_τ).

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

⟨2⟩3. $\cdot;\cdot;\Phi;\underline{\mathcal{R}}' \vdash \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr}) \Rightarrow pt$, already established.

⟨2⟩4. Prove typing with EXPL_SPINE_RET; ⟨2⟩3 – ⟨2⟩1 with EXPL_SPINE_RES, EXPL_SPINE_LOG, EXPL_SPINE_COMP respectively; EXPL_IS_TVAL.

⟨1⟩10. CASE: EXPL_IS_ACTION_STORE.
LET: $\mathit{ret} = \Sigma\,\_{:}\texttt{unit}.\,(\mathit{mem\_ptr} \overset{\texttt{const}_\tau\texttt{true}}{\mapsto_\tau} \mathit{pval}) * \texttt{I}$.
$pt = \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr})(\_)$, $pt' = \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr})(\mathit{oarg})$, where
$\mathit{oarg} = \{\mathit{init} = \texttt{const}_\tau\texttt{true}, \mathit{value} = \mathit{pval}\}$.

ASSUME: $\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \texttt{store}\,(\_, \tau, \mathit{mem\_ptr}, \mathit{pval}, \_, \mathit{res\_term}) \Rightarrow \mathit{ret}$ and
$\mathit{heap} = \mathit{heap}' + \{pt' \,\&\, \texttt{None}\}$ so
$\langle h + f; \mathit{res\_term}\rangle \Downarrow \langle \mathit{heap}' + \{pt \,\&\, \texttt{None}\}; \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr})\rangle$.

$\exists h'', \underline{\mathcal{R}}''$ such that $\mathit{heap}' + \{pt \,\&\, \texttt{None}\} = h'' + f$, $\Phi \vdash h'' \Leftarrow \underline{\mathcal{R}}''$ and
$\cdot;\cdot;\Phi;\underline{\mathcal{R}}'' \vdash \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr}) \Rightarrow pt$, by lemma B8.8 (Resource term reduction is isolated).

$\underline{\mathcal{R}}'' = \_{:}pt$ by lemma B8.3 and $h'' = \{pt \,\&\, \texttt{None}\}$ by lemma B8.4, hence $\mathit{heap}' = f$.
LET: $h' = \{pt' \,\&\, \texttt{None}\}$ and $\underline{\mathcal{R}}' = \_{:}pt'$.
This means $\mathit{heap} = h' + f$ and $\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}'$ (by HEAP_PRED_OWNED).

PROVE: $\cdot;\cdot;\Phi;\underline{\mathcal{R}}' \vdash \texttt{done}\,\langle \texttt{Unit}, \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr})\rangle{:}\mathit{ret} \Rightarrow \mathit{ret}$.

⟨2⟩1. $\cdot \vdash \texttt{Unit} \Rightarrow \texttt{unit}$ by PURE_VAL_UNIT.

⟨2⟩2. $\cdot;\cdot;\Phi;\_{:}pt' \vdash \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr}) \Leftarrow pt'$ by RES_SYN_PRED.

⟨2⟩3. Prove typing with EXPL_SPINE_RET; ⟨2⟩2 – ⟨2⟩1 with EXPL_SPINE_RES, EXPL_SPINE_COMP respectively; EXPL_IS_TVAL.

⟨1⟩11. CASE: EXPL_IS_ACTION_KILL_STATIC.
ASSUME: $\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \texttt{kill}\,(\texttt{static}\,\tau, \mathit{mem\_ptr}, \mathit{res\_term}) \Rightarrow \Sigma\,\_{:}\texttt{unit}.\,\texttt{I}$ and

$\langle h + f; \mathit{res\_term}\rangle \Downarrow \langle \mathit{heap} + \{\mathit{pt}\,\&\,\texttt{None}\}; \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr})\rangle$.

$\exists h'', \underline{\mathcal{R}}''$ such that $\mathit{heap} + \{\mathit{pt}\,\&\,\texttt{None}\} = h'' + f$, $\Phi \vdash h'' \Leftarrow \underline{\mathcal{R}}''$ and $\cdot;\cdot;\Phi;\underline{\mathcal{R}}'' \vdash \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr}) \Rightarrow \mathit{pt}$, by lemma B8.8 (Resource term reduction is isolated).

$\underline{\mathcal{R}}'' = \_{:}\mathit{pt}$ by lemma B8.3 and $h'' = \{\mathit{pt}\,\&\,\texttt{None}\}$ by lemma B8.4, hence $\mathit{heap} = f$.
LET: $h' = \cdot$ and $\underline{\mathcal{R}}' = \cdot$.
This means $\mathit{heap} = h' + f$ and $\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}'$ (by HEAP_EMPTY).

PROVE: $\cdot;\cdot;\Phi;\cdot \vdash \texttt{done}\,\langle\texttt{Unit}\rangle{:}\Sigma\,\_{:}\texttt{unit}.\,\texttt{I} \Rightarrow \Sigma\,\_{:}\texttt{unit}.\,\texttt{I}$
PROOF: By EXPL_SPINE_RET, PURE_VAL_UNIT, EXPL_SPINE_COMP, EXPL_IS_TVAL.

⟨1⟩12. CASE: EXPL_IS_MEMOP_REL_BINOP.
PROOF: Similar to PURE_EXPR_REL_BINOP.

⟨1⟩13. CASE: EXPL_IS_MEMOP_INTFROMPTR.
LET: $\mathit{ret} = \Sigma\, y{:}\texttt{integer}.\; y = \texttt{cast\_ptr\_to\_int}\,\mathit{mem\_ptr} \wedge \texttt{I}$. Since $\Phi \vdash h \Leftarrow \cdot$, $h = \cdot$, $\mathit{heap} = f$.
ASSUME: $\cdot;\cdot;\Phi;\cdot \vdash \texttt{intFromPtr}\,(\tau_1, \tau_2, \mathit{mem\_ptr}) \Rightarrow \mathit{ret}$.
LET: $h' = \cdot$ and $\underline{\mathcal{R}}' = \cdot$, so $\mathit{heap} = h' + f$ trivially and $\Phi \vdash \cdot \Leftarrow \cdot$ (by HEAP_EMPTY).
PROVE: $\cdot;\cdot;\cdot;\cdot \vdash \texttt{done}\,\langle\mathit{mem\_int}\rangle{:}\mathit{ret} \Rightarrow \mathit{ret}$
PROOF: Prove typing with EXPL_SPINE_RET, EXPL_SPINE_PHI, EXPL_SPINE_COMP and EXPL_TOP_VAL_DONE instead.

⟨1⟩14. CASE: EXPL_IS_MEMOP_PTRFROMINT.
PROOF: Similar to EXPL_IS_MEMOP_INTFROMPTR, swapping base types `integer` and `pointer`.

⟨1⟩15. CASE: EXPL_IS_MEMOP_PTRVALIDFORDEREF.
LET: $\mathit{pt} = \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr})(\mathit{oarg})$ where $\mathit{oarg} = \{\mathit{init} = \texttt{const}_\tau\texttt{true}, \mathit{value} = \mathit{value}\}$
$\mathit{ret} = \Sigma\, y{:}\texttt{bool}.\; y = \texttt{aligned}\,(\tau, \mathit{pval}) \wedge \mathit{pt} * \texttt{I}$

ASSUME: $\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \texttt{ptrValidForDeref}\,(\tau, \mathit{mem\_ptr}, \mathit{res\_term}) \Rightarrow \mathit{ret}$
and $\mathit{heap} = \mathit{heap}' + \{\mathit{pt}\,\&\,\texttt{None}\}$ so
$\langle h + f; \mathit{res\_term}\rangle \Downarrow \langle \mathit{heap}' + \{\mathit{pt}\,\&\,\texttt{None}\}; \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr})\rangle$.

LET: h' and $\underline{\mathcal{R}}'$ be as per lemma B8.8 (Resource term reduction is isolated).
$\underline{\mathcal{R}}' = \_{:}\mathit{pt}$ by lemma B8.3 and $h' = \{\mathit{pt}\,\&\,\texttt{None}\}$ by lemma B8.4, hence $\mathit{heap}' = f$.
This means $\mathit{heap} = h' + f$, $\Phi \vdash h' \Leftarrow \underline{\mathcal{R}}'$ and $\cdot;\cdot;\Phi;\underline{\mathcal{R}}' \vdash \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr}) \Rightarrow \mathit{pt}$.

PROVE: $\cdot;\cdot;\Phi;\underline{\mathcal{R}}' \vdash \texttt{done}\,\langle\mathit{bool\_value}, \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr})\rangle{:}\mathit{ret} \Rightarrow \mathit{ret}$.

⟨2⟩1. $\cdot \vdash \mathit{bool\_value} \Rightarrow \texttt{bool}$ by PURE_VAL_TRUE/ PURE_VAL_FALSE.

⟨2⟩2. $\texttt{smt}\,(\cdot \Rightarrow \mathit{bool\_value} = \texttt{aligned}\,(\tau, \mathit{mem\_ptr}))$.
PROOF: By construction of *bool_value* (inversion on the transition).

$\langle 2\rangle 3.$ $\cdot;\cdot;\Phi;\_{:}pt \vdash \texttt{Owned}\,\langle\tau\rangle(mem\_ptr) \Leftarrow pt$, already established.

$\langle 2\rangle 4.$ Prove typing with EXPL_SPINE_RET; $\langle 2\rangle 3 - \langle 2\rangle 1$ with EXPL_SPINE_RES, EXPL_SPINE_PHI, EXPL_SPINE_COMP respectively; EXPL_IS_TVAL.

$\langle 1\rangle 16.$ CASE: EXPL_IS_MEMOP_PTRWELLALIGNED.
LET: $ret = \Sigma\, y{:}\texttt{bool}.\ y = \texttt{aligned}\,(\tau, mem\_ptr) \land \texttt{I}$.
ASSUME: $\cdot;\cdot;\Phi;\cdot \vdash \texttt{ptrWellAligned}\,(\tau, mem\_ptr) \Rightarrow ret$.
Since $\Phi \vdash h \Leftarrow \cdot$, $h = \cdot$, $heap = f$.
LET: $h' = \cdot$ and $\underline{\mathcal{R}'} = \cdot$, so $heap = h' + f$ trivially and $\Phi \vdash \cdot \Leftarrow \cdot$ (by HEAP_EMPTY).
PROVE: $\cdot;\cdot;\Phi;\cdot \vdash \texttt{done}\,\langle bool\_value\rangle{:}ret \Rightarrow ret$.

$\langle 2\rangle 1.$ $\cdot \vdash bool\_value \Rightarrow \texttt{bool}$ by PURE_VAL_TRUE/ PURE_VAL_FALSE.

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

$\langle 2\rangle 3.$ Prove typing with EXPL_SPINE_RET, EXPL_SPINE_PHI, EXPL_SPINE_COMP, EXPL_IS_TVAL.

$\langle 1\rangle 17.$ CASE: EXPL_IS_MEMOP_PTRARRAYSHIFT.
PROOF: Similar to PURE_EXPR_ARRAY_SHIFT, but with EXPL_IS_TVAL.

$\langle 1\rangle 18.$ CASE: EXPL_SEQ_CCALL.
ASSUME: $ident{:}fun \equiv \overline{x_i}^{\,i} \mapsto texpr \in \texttt{Globals}$
$\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \overline{spine\_elem_i}^{\,i} :: fun \gg ret$.
$\Phi \vdash h \Leftarrow \underline{\mathcal{R}}$
$\langle h + f; \texttt{ccall}\,(\tau, ident, \overline{spine\_elem_i}^{\,i})\rangle \longrightarrow \langle heap; \sigma_2(texpr){:}ret'\rangle$
$\mathcal{C};\mathcal{L};\Phi'';\mathcal{R}'' \vdash texpr \Leftarrow ret''$ where $\overline{x_i}^{\,i} :: fun \rightsquigarrow \mathcal{C};\mathcal{L};\Phi'';\mathcal{R}'' \mid ret''$.

PROVE: $\exists h', \Phi', \underline{\mathcal{R}'}$ such that
$\cdot;\cdot;\Phi;\cdot \sqsubseteq \cdot;\cdot;\Phi';\cdot$
$heap = h' + f$
$\Phi' \vdash h' \Leftarrow \underline{\mathcal{R}'}$
and $\cdot;\cdot;\Phi';\underline{\mathcal{R}'} \vdash \sigma_2(texpr) \Leftarrow ret$.

$\langle 2\rangle 1.$ $\mathcal{C};\mathcal{L};\Phi,\Phi'';\mathcal{R}'' \vdash texpr \Leftarrow ret''$
$\Phi, \sigma_2(\Phi'') \vdash h \Leftarrow \underline{\mathcal{R}'}$
$\cdot;\cdot;\Phi,\sigma_2(\Phi'');\underline{\mathcal{R}} \vdash \overline{spine\_elem_i}^{\,i} :: fun \gg ret$.
PROOF: By lemma B6 (Weakening).

$\langle 2\rangle 2.$ $ret = ret' = \sigma_2(ret'') \land \exists h'_1, \underline{\mathcal{R}'_1}$.
$heap = h'_1 + f$, and $(\Phi, \sigma_2(\Phi'') \vdash h'_1 \Leftarrow \underline{\mathcal{R}'_1})$
$\cdot;\cdot;\Phi,\sigma_2(\Phi'');\underline{\mathcal{R}'_1} \vdash \sigma_2 \Leftarrow (\mathcal{C};\mathcal{L};\mathcal{R}'')$.
PROOF: By lemma B9.3 (Well-typed spines produce substitutions and the same return type).

$\langle 2\rangle 3.$ $\cdot;\cdot;\Phi,\sigma_2(\Phi'');\underline{\mathcal{R}'_1} \vdash \sigma(texpr) \Leftarrow \sigma(ret'')$.
PROOF: By lemma B7.3 (Substitution), because $\sigma_2(\Phi) = \Phi$ since it contains only closed terms / is well-formed w.r.t $\cdot;\cdot$.

⟨2⟩4. LET: $h' = h'_1$; $\Phi' = \Phi, \sigma_2(\Phi'')$; $\underline{\mathcal{R}}' = \underline{\mathcal{R}}'_1$.

⟨2⟩5. $\cdot;\cdot;\Phi;\cdot \sqsubseteq \cdot;\cdot;\Phi,\sigma_2(\Phi'');\cdot$ trivially.

⟨1⟩19. CASE: EXPL_SEQ_PROC.
PROOF: Similar to EXPL_SEQ_PROC.

⟨1⟩20. CASE: EXPL_IS_MEMOP, EXPL_IS_ACTION, EXPL_IS_NEG_ACTION.
PROOF: By induction.

⟨1⟩21. CASE: EXPL_TOP_SEQ_LETP, EXPL_TOP_SEQ_LETTP, EXPL_TOP_SEQ_LET.
PROOF: See EXPL_TOP_SEQ_LETTfor a more general case and proof.

⟨1⟩22. CASE: EXPL_TOP_SEQ_LETT.
ASSUME: $\cdot;\cdot;\cdot;\underline{\mathcal{R}}_1,\underline{\mathcal{R}}_2 \vdash \texttt{let}\,\overline{\mathit{ret\_pat}_i}^{\,i}{:}\mathit{ret}_1 = \texttt{done}\,\langle\overline{\mathit{ret\_term}_i}^{\,i}\rangle\,\texttt{in}\,\mathit{texpr}_2 \Leftarrow \mathit{ret}_2$
so $\cdot;\cdot;\Phi;\underline{\mathcal{R}}_1 \vdash \texttt{done}\,\langle\overline{\mathit{ret\_term}_i}^{\,i}\rangle \Leftarrow \mathit{ret}_1$
and $\Phi \vdash \mathit{ret\_pat}{:}\mathit{ret}_1 \rightsquigarrow \mathcal{C}_3;\mathcal{L}_3;\Phi_3;\mathcal{R}_3$
and $\mathcal{C}_3;\mathcal{L}_3;\Phi,\Phi_3;\underline{\mathcal{R}}_2,\mathcal{R}_3 \vdash \mathit{texpr} \Leftarrow \mathit{ret}_2$ (by inversion).

$\Phi \vdash h \Leftarrow \underline{\mathcal{R}}_1,\underline{\mathcal{R}}_2$ so $h = h_1 + h_2$ where $\Phi \vdash h_1 \Leftarrow \underline{\mathcal{R}}_1$ and $\Phi \vdash h_2 \Leftarrow \underline{\mathcal{R}}_2$ by lemma B8.4 (Normalised resource context determines structure of heap).
$\langle h + f; \texttt{let}\,\overline{\mathit{ret\_pat}_i}^{\,i}{:}\mathit{ret}_1 = \texttt{done}\,\langle\overline{\mathit{ret\_term}_i}^{\,i}\rangle\,\texttt{in}\,\mathit{texpr}\rangle \longrightarrow \langle \mathit{heap}; \sigma(\mathit{texpr})\rangle$.
where $\langle h; \overline{\mathit{ret\_pat}_i = \mathit{ret\_term}_i}^{\,i}\rangle \rightsquigarrow \langle \mathit{heap}; \sigma\rangle$.

PROVE: $\exists \Phi', h', \underline{\mathcal{R}}'$.
$\cdot;\cdot;\Phi;\cdot \sqsubseteq \cdot;\cdot;\Phi';\cdot$
$\mathit{heap} = h' + f$ and $\Phi' \vdash h' \Leftarrow \underline{\mathcal{R}}'$
$\cdot;\cdot;\Phi';\underline{\mathcal{R}}' \vdash \sigma(\mathit{texpr}_2) \Leftarrow \sigma(\mathit{ret}_2)$.

$\exists \underline{\mathcal{R}}'_1.\ \mathit{heap} = h'_1 + h_2 + f$
$\Phi \vdash h'_1 \Leftarrow \underline{\mathcal{R}}'_1$ and $\cdot;\cdot;\Phi;\underline{\mathcal{R}}'_1 \vdash \sigma \Leftarrow (\mathcal{C}';\mathcal{L}';\mathcal{R}')$
by lemma B9.4 (Well-typed values pattern-match successfully).

This means $\cdot;\cdot;[\mathrm{id},\sigma](\Phi,\Phi_3);\underline{\mathcal{R}}'_1,\underline{\mathcal{R}}_2 \vdash [\mathrm{id},\sigma] \Leftarrow (\mathcal{C}';\mathcal{L}';\underline{\mathcal{R}}_2,\mathcal{R}')$ by lemma B6 (Weakening).

LET: $\Phi' = \Phi, \sigma(\Phi_3)$, $h' = h'_1 + h_2$ and $\underline{\mathcal{R}}' = \underline{\mathcal{R}}'_1,\underline{\mathcal{R}}_2$.
By lemma B7.3 (Substitution), because $\sigma(\Phi) = \Phi$ since it contains only closed terms / is well-formed w.r.t $\cdot;\cdot$.

⟨1⟩23. CASE: EXPL_TOP_SEQ_LETT.
ASSUME: $\cdot;\cdot;\Phi;\underline{\mathcal{R}}_1,\underline{\mathcal{R}}_2 \vdash \texttt{let}\,\overline{\mathit{ret\_pat}_i}^{\,i}{:}\mathit{ret}_1 = \mathit{texpr}_1\,\texttt{in}\,\mathit{texpr}_2 \Leftarrow \mathit{ret}_2$
so $\cdot;\cdot;\Phi;\underline{\mathcal{R}}_1 \vdash \mathit{texpr}_1 \Leftarrow \mathit{ret}_1$
and $h = h_1 + h_2$ where $\Phi \vdash h_1 \Leftarrow \underline{\mathcal{R}}_1$ and $\Phi \vdash h_2 \Leftarrow \underline{\mathcal{R}}_2$ by lemma B8.4 (Normalised resource context determines structure of heap).
$\langle h; \mathit{texpr}_1\rangle \longrightarrow \langle \mathit{heap}; \mathit{texpr}'_1\rangle$.

Proceed by induction, instantiating the frame from the inductive hypothesis with $h_2 + f$.

⟨1⟩24. Case: Expl_Top_Seq_Case.

Assume: $\cdot;\cdot;\Phi;\underline{\mathcal{R}} \vdash \texttt{case}\, \mathit{pval}\, \texttt{of}\, \overline{\mid \mathit{pat}_i \Rightarrow \mathit{texpr}_i}^{\,i}\, \texttt{end} \Leftarrow \mathit{ret}$

$\overline{\mathit{pat}_i{:}\beta_1 \rightsquigarrow \mathcal{C}_i \,\texttt{with}\, \mathit{term}_i}^{\,i}$

$\overline{\mathcal{C}, \mathcal{C}_i; \mathcal{L}; \Phi, \mathit{term}_i = \mathit{pval}; \mathcal{R} \vdash \mathit{texpr}_i \Leftarrow \mathit{ret}}^{\,i}$.

$\mathit{pat}_j = \mathit{pval} \rightsquigarrow \sigma_j$ and $\forall\, i \,<\, j.\ \texttt{not}\,(\mathit{pat}_i = \mathit{pval} \rightsquigarrow \sigma_i)$.

Let: $\Phi' = \Phi, \sigma_j(\mathit{term}_j = \mathit{pval})$, $h' = h$ and $\underline{\mathcal{R}}' = \underline{\mathcal{R}}$.
$\cdot;\cdot;\Phi';\underline{\mathcal{R}} \vdash [\mathrm{id}, \sigma_j] \Leftarrow (\mathcal{C}_j;\cdot;\underline{\mathcal{R}})$ by lemma B9.4 (Well-typed values pattern-match successfully) and lemma B6 (Weakening).
Hence $\cdot;\cdot;\Phi;\cdot \sqsubseteq \cdot;\cdot;\Phi';\cdot$ and $\cdot;\cdot;\Phi';\underline{\mathcal{R}} \vdash \sigma_j(\mathit{texpr}_j) \Leftarrow \sigma_j(\mathit{ret})$ by lemma B7.3 (Substitution).

⟨1⟩25. Case: Expl_Top_Seq_If.
See Expl_Top_Seq_Case for more general case and proof.

⟨1⟩26. Case: Expl_Top_Seq_Run.
Proof: Similar to Expl_Seq_CCall.

⟨1⟩27. Case: Expl_Top_Seq_Bound.
Proof: By induction.

⟨1⟩28. Case: Expl_Top_Is_LetS.
Proof: Similar to Expl_Top_Seq_LetT.