Formal definition of the kernel type system

Dhruv C. Makwana

The formalisation is defined over a let-normalised version of the Core language of Cerberus. A proof of soundness of type checking is given in a separate document.

Contents

A1 Commentary

In this document, we formalise "kernel CN", which is essentially ordinary CN with no type and resource inference. In particular, we assume that all universal quantifiers are explicitly instantiated, that all existential quantifiers have explicit witnesses, and all resource manipulations have proof terms with linear/substructural types. However, we do not require proof terms for the logical properties, since by construction all of the entailments fall into the SMT fragment. Since our inference algorithm can be extended to an elaboration algorithm producing a fully-annotated program, kernel CN could serve as an intermediate representation for the CN compiler (which we have formalised elaboration for iterated resource manipulation, though not footprint analysis). Moreover, the lack of inference makes it a simpler language to prove type safety for.

The kernel CN is a calculus in A-normal form, with a bidirectional type system. Since we handle the majority of C, the entire system is very large, and so we only provide commentary on the highlights.

A1.1 Types and Terms

As in the paper, CN programs have both computational and logical terms. Every such term, computational or ghost, has a *base type* β, which are things like unit, booleans, (mathematical) integers, locations, and records of other base types. Each C type τ is mapped to a corresponding base type – for example, $\beta_{\texttt{int*}} = \texttt{pointer}$. Logical terms are variously referred to as *term*, *ptr* (for pointers), *value* (for pointees), *iarg* (for input-arguments), *oarg* (output-arguments, of type record or array of records), and *iguard* (for boolean guards of iterated resources).

$$
\begin{aligned}
\mathit{res} &::= \texttt{emp} \mid \mathit{term} \mid \mathit{pred} \mid \mathit{qpred} \mid \mathit{res}_1 * \mathit{res}_2 \mid \exists\, y{:}\beta.\ \mathit{res}' \mid \texttt{if}\ \mathit{term}\ \texttt{then}\ \mathit{res}_1\ \texttt{else}\ \mathit{res}_2 \\
\mathit{pred} &::= \alpha(\mathit{ptr}, \mathit{iargs})(\mathit{oarg}) \\
\mathit{qpred} &::= (x; \mathit{iguard})\{\alpha(\mathit{ptr} + x \times \mathit{step}, \mathit{iargs})\}(\mathit{oarg})
\end{aligned}
$$

$$
\begin{aligned}
\mathit{res\_term} &::= \texttt{emp} \mid \texttt{term} \mid \mathit{pred\_term} \mid \mathit{qpred\_term} \mid \langle \mathit{res\_term}_1, \mathit{res\_term}_2 \rangle \mid \texttt{pack}\,(\mathit{oarg}, \mathit{res\_term}') \mid \\
&\qquad r \mid \texttt{fold}\ \mathit{res\_term}{:}\mathit{pred} \mid \mathit{pred\_ops} \\
\mathit{pred\_ops} &::= \texttt{explode}\,(\mathit{res\_term}) \mid \texttt{implode}\,(\mathit{res\_term}, \mathit{tag}) \mid \texttt{iterate}\,(\mathit{res\_term}, \mathit{int}) \\
&\quad \texttt{congeal}\,(\mathit{res\_term}, \mathit{int}) \mid \texttt{break}\,(\mathit{res\_term}, \mathit{term}) \mid \texttt{glue}\,(\mathit{res\_term}) \\
&\quad \texttt{inj}\,(\mathit{res\_term}, \mathit{ptr}, \mathit{step}, x.\ \mathit{iargs}) \mid \texttt{split}\,(\mathit{res\_term}, \mathit{iguard})
\end{aligned}
$$

Figure 1: Grammar of Resource Terms

In Figure 1, we give the grammar of resource types (i.e., separation logic predicates) and resource terms (the proof terms used by the kernel Core typechecker). The standard resources *res* can be an empty heap `emp`, a boolean condition *term*, the separating conjunction $\mathit{res}_1 * \mathit{res}_2$, an existential type $\exists\, y{:}\beta.\ \mathit{res}$, and the disjunction $\texttt{if}\ \mathit{term}\ \texttt{then}\ \mathit{res}_1\ \texttt{else}\ \mathit{res}_2$. We use a conditional rather than a traditional disjunction to avoid backtracking during typechecking.

Resource predicates have special syntax to handle the division of their arguments into inputs and outputs. An occurrence of a predicate is written $\alpha(\mathit{ptr}, \mathit{iargs})(\mathit{oarg})$. This is read as the predicate α, applied to a pointer argument *ptr* and a list of other input arguments *iargs*. The output argument *oarg* is highlighted and in a second set of parentheses. Every predicate has exactly one output argument, of type record (with zero or more fields). A *qpred* represents the iterated separating conjunction of predicate instances; it quantifies over integer indices x satisfying a guard *iguard*, and is with input arguments *iargs* and output *oarg*. It represents an instance of α beginning at *ptr*, and repeating every *step* bytes, for as long as the *iguard* is true.

Each resource type has introduction and elimination forms – e.g. $res_1 * res_2$ has pairing and pattern matching proof terms. The standard resource types have the expected rules, and predicate types can be introduced by explicitly folding a predicate definition `fold` *res_term*:*pred*, and unfolded via pattern-matching.

In addition, there are resource operations recording the resource-manipulation steps inference uses to successfully type a program. If we suppress the book-keeping of checking that input arguments match, calculating indices, and updating output arguments, most of these operations have simple intuitions. `explode` (*res_term*) and `implode` (*res_term*, *tag*) are operations on structs and their members; the first takes an `Owned` ⟨`struct` *tag*⟩ and turns it into a `Owned` $\langle\tau_i\rangle$ for each of of its members; the second does the inverse. `iterate` (*res_term*, *int*) and `congeal` (*res_term*, *int*) function similarly, but for C's fixed-size arrays, returning a *quantified* `Owned` $\langle\tau\rangle$ instead.

Morally, `break` has type $qpred \rightarrow qpred * pred$: it extracts a single predicate from a quantified one, and must return the remainder as well because resource terms are linearly typed; `glue` has type $qpred * pred \rightarrow qpred$: it is the inverse to `break`; `split` has type $qpred * iguard \rightarrow qpred * qpred$: given a quantified predicate of index-guard $iguard'$, and an *iguard*, if $iguard \rightarrow iguard'$ then it splits the given quantified predicate into two disjoint parts (one of index-guard *iguard* and the other of $iguard' \land \neg\, iguard$); `inj` has type $pred * ptr * step * iargs \rightarrow qpred$: it turns a predicate $\alpha(ptr', iargs')$ into a quantified predicate, with $iguard = (x = k)$, where $k = (ptr' - ptr)/step$ and $iargs' = k/x(iargs)$. Because our inference algorithm does not support inferring merging arrays, there is no inverse to `split` of type $qpred * qpred \rightarrow qpred$.

A1.2 Judgements and Example Rules

The contexts for the rules consist of four parts: (1) $\mathcal{C}$ containing the computational variables from the Core program; (2) $\mathcal{L}$ containing purely logical variables mentioned in specifications; (3) Φ, the constraint context, containing a list of (non-quantified) SMT constraints; and (4) $\mathcal{R}$ a *linear* context containing the resources available at that point during type-checking. Assuming a constraint context of only non-quantified constraints is an acceptable simplification, because the elaboration pass can annotate terms with fully-instantiated constraints, whose quantifiers were either supplied by lemmas, annotations or default instantiation.

We focus on the judgements for typing resource terms and memory actions. The judgement $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash res\_term \Rightarrow res$ should be read as "under a context of computational variables $\mathcal{C}$, logical variables $\mathcal{L}$, constraints Φ and resources $\mathcal{R}$, the resource term *res_term* synthesises resource type *res*" (the highlighting shows the part of the judgement with an output mode). The judgement $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash res\_term \Leftarrow res$ reads similarly, replacing 'synthesises' with 'checks against'.

We need both judgements because variables, folding, predicate operations are naturally typed as synthesising rules, whereas constraints, packing existentials, and conditional resources require checking. Furthermore, as we shall see soon, memory actions require a synthesising judgement (to obtain and manipulate the output argument of `Owned` $\langle\tau\rangle$), whereas top-level values (such as typing spines) require checking judgements.

RES_CHK_IF_TRUE

$$\frac{\begin{array}{l}1.\ \mathtt{smt}\,(\Phi \Rightarrow \mathit{term})\\ 2.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{res\_term} \Leftarrow \mathit{res}_1\end{array}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{res\_term} \Leftarrow \mathtt{if}\ \mathit{term}\ \mathtt{then}\ \mathit{res}_1\ \mathtt{else}\ \mathit{res}_2}$$

RES_SYN_PRED

$$\frac{\begin{array}{l}1.\ \mathit{pred} \equiv \alpha(\mathit{ptr}, \overline{\mathit{iarg}_i}^{\,i})(\mathit{oarg})\\ 2.\ \alpha \equiv x_p{:}\mathtt{pointer}, \overline{x_i{:}\beta_i}^{\,i}, y{:}\mathtt{record}\,\overline{\mathit{tag}_j{:}\beta'_j}^{\,j} \mapsto \mathit{res} \in \mathtt{Globals}\\ 3.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{res\_term} \Leftarrow [\mathit{oarg}/y, [\overline{\mathit{iarg}_i/x_i}^{\,i}], \mathit{ptr}/x_p](\mathit{res})\end{array}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathtt{fold}\,\mathit{res\_term}{:}\mathit{pred} \Rightarrow \mathit{pred}}$$

EXPL_IS_ACTION_CREATE

$$\frac{\begin{array}{l}1.\ \mathit{ret} \equiv \Sigma\, y_p{:}\mathtt{pointer}.\ \mathit{term} \wedge \exists\, y{:}\mathtt{record}\,\mathit{init}{:}\mathtt{bool}\,\mathit{value}{:}\beta_\tau.\ \mathit{ret}'\\ 2.\ \mathit{ret}' \equiv (y_p \overset{y.\mathit{init}}{\mapsto_\tau} y.\mathit{value}) * y.\mathit{init} = \mathtt{false} \wedge \mathtt{I}\end{array}}{\mathcal{C};\mathcal{L};\Phi;\cdot \vdash \mathtt{create}\,(\mathit{pval}, \tau) \Rightarrow \mathit{ret}}$$

EXPL_IS_ACTION_LOAD

$$\frac{\begin{array}{l}1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{res\_term} \Rightarrow \mathit{term} \overset{\mathit{init}}{\mapsto_\tau} \mathit{pval}_1\\ 2.\ \mathtt{smt}\,(\Phi \Rightarrow (\mathit{term} = \mathit{pval}_0) \wedge (\mathit{init} = \mathtt{true}))\\ 3.\ \mathit{ret} \equiv \Sigma\, y{:}\beta_\tau.\ y = \mathit{pval}_1 \wedge (\mathit{pval}_0 \overset{\mathtt{true}}{\mapsto_\tau} \mathit{pval}_1) * \mathtt{I}\end{array}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathtt{load}\,(\tau, \mathit{pval}_0, \_, \mathit{res\_term}) \Rightarrow \mathit{ret}}$$

EXPL_IS_ACTION_STORE

$$\frac{\begin{array}{l}1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{res\_term} \Rightarrow \mathit{term} \overset{\_}{\mapsto_\tau} \_\\ 2.\ \mathtt{smt}\,(\Phi \Rightarrow \mathit{term} = \mathit{pval}_0)\\ 3.\ \mathit{ret} \equiv \Sigma\, \_{:}\mathtt{unit}.\ (\mathit{pval}_0 \overset{\mathtt{true}}{\mapsto_\tau} \mathit{pval}_1) * \mathtt{I}\end{array}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathtt{store}\,(\_, \tau, \mathit{pval}_0, \mathit{pval}_1, \_, \mathit{res\_term}) \Rightarrow \mathit{ret}}$$

EXPL_IS_ACTION_KILL_STATIC

$$\frac{\begin{array}{l}1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{res\_term} \Rightarrow \mathit{term} \overset{\_}{\mapsto_\tau} \_\\ 2.\ \mathtt{smt}\,(\Phi \Rightarrow \mathit{term} = \mathit{pval})\end{array}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathtt{kill}\,(\mathtt{static}\,\tau, \mathit{pval}, \mathit{res\_term}) \Rightarrow \Sigma\, \_{:}\mathtt{unit}.\ \mathtt{I}}$$

Above is one of two rules for checking a conditional resource. Thanks to the ordered disjunction, the rule is simple: if the SMT solver can statically prove *term*, then check the resource term against the res_1. The converse (omitted) checks against res_2 if the SMT solver can prove the negation of the condition; if neither is provable, the rules try to synthesise an under-determined conditional resource (the only way this is possible is if *res_term* is a variable of an SMT-equivalent type).

The rule for folding predicates shown is simplified for presentation (omitting only the type checking of the all the predicate arguments, and the exclusion of the `Owned` $\langle\tau\rangle$ predicate because it cannot be folded). The first line is a simple lookup based on the predicate name of types of the arguments, and the "body" *res* of the predicate. The second checks *res_term* against the *res* with its arguments (supplied by the fold term) substituted in.

The above rules for typing memory actions are also simplified for presentation. Allocating memory (which takes an alignment *pval* and a C type τ) synthesises a return type *ret* representing: a newly created pointer (referred to in the type by the name y_p), some omitted constraints about alignment and representability (*term*), a logical value (y) representing the output argument of a points-to/`Owned` $\langle\tau\rangle$ resource (which differs slightly from the implementation in that it additionally contains the initialisedness status), the resource itself (`Owned` $\langle\tau\rangle(y_p)(y)$ is pretty-printed in more familiar $\mapsto$ notation), and a constraint that the points-to is not initialised.

Loading from a memory location requires a correctly typed resource term, *and* its output argument's initialisedness status *init* to be true. Because the types are linear, it not only returns the pointed-to value, but also the same permission it consumed.

Storing to a memory location is similar to loading: it requires a points-to permission, but without any constraints on its initialisedness. The permission it returns reflects the fact that the pointee is definitely initialised, and that a new value is pointed to by this location.

De-allocating memory is the converse of allocating memory: a resource term is required, but not returned.

A1.3 Differences from Implementation

There are some minor differences between the implementation and the formalisation. The formalisation has a richer grammar of resources: this means it can support tagged unions more succinctly and can open predicates in more cases. The formalisation assumes that iterated resources output arguments have type array of records, whereas the implementation uses records of arrays.

A2 Types and Patterns

A2.1 Resource Related

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

IG_Cmp_Eq

$$\frac{1.\ \texttt{smt}\,(\Phi \Rightarrow \forall\, x.\ iguard \leftrightarrow iguard')}{\Phi \vdash \texttt{cmp\_min}\,(iguard, iguard') \leadsto \texttt{Eq}, iguard}$$

IG_Cmp_Lt

$$\frac{1.\ \texttt{smt}\,(\Phi \Rightarrow \forall\, x.\ iguard \rightarrow iguard')}{\Phi \vdash \texttt{cmp\_min}\,(iguard, iguard') \leadsto \texttt{Lt}, iguard}$$

IG_Cmp_Gt

$$\frac{1.\ \texttt{smt}\,(\Phi \Rightarrow \forall\, x.\ iguard' \rightarrow iguard)}{\Phi \vdash \texttt{cmp\_min}\,(iguard, iguard') \leadsto \texttt{Gt}, iguard'}$$

IG_Cmp_None

$$\frac{}{\Phi \vdash \texttt{cmp\_min}\,(iguard, iguard') \leadsto \texttt{None}}$$

$\boxed{\Phi \vdash qpred\_term \sqsubseteq ?\ qpred\_term' \leadsto opt\_cmp}$ given constraints Φ , $qpred\_term$ is potentially included in $qpred\_term'$ (or vice-versa) with ordering opt_cmp

Q_Cmp_Name_Neq

$$\frac{1.\ \alpha_1 \neq \alpha_2}{\Phi \vdash (\_; \_)\{\alpha_2(\_ + \_\times\_, \_)\} \sqsubseteq ?\ (\_; \_)\{\alpha_1(\_ + \_\times\_, \_)\} \leadsto \texttt{None}}$$

Q_Cmp_PtrStep_Neq

$$\frac{\begin{array}{l}1.\ term_1\ \equiv\ (ptr = ptr') \wedge (step = step')\\ 2.\ \texttt{smt}\,(\Phi \Rightarrow \neg\, term_1)\end{array}}{\Phi \vdash (x; \_)\{\alpha(ptr + x\times step, \_)\} \sqsubseteq ?\ (x; \_)\{\alpha(ptr' + x\times step', \_)\} \leadsto \texttt{None}}$$

Q_Cmp_IG_Neq

$$\frac{\begin{array}{l} 1.\ term_1 \equiv (ptr = ptr') \land (step = step') \\ 2.\ \texttt{smt}\,(\Phi \Rightarrow term_1) \\ 3.\ \Phi \vdash \texttt{cmp\_min}\,(iguard, iguard') \rightsquigarrow \texttt{None} \end{array}}{\Phi \vdash (x; iguard)\{\alpha(ptr + x \times step, \_)\} \sqsubseteq ?\ (x; iguard')\{\alpha(ptr' + x \times step', \_)\} \rightsquigarrow \texttt{None}}$$

Q_Cmp_IArg_Neq

$$\frac{\begin{array}{l} 1.\ term_1 \equiv (ptr = ptr') \land (step = step') \\ 2.\ \texttt{smt}\,(\Phi \Rightarrow term_1) \\ 3.\ \Phi \vdash \texttt{cmp\_min}\,(iguard, iguard') \rightsquigarrow cmp, iguard'' \\ 4.\ term_2 \equiv iguard'' \to \bigwedge(\overline{iarg_i = iarg'_i}^{\,i}) \\ 5.\ \texttt{smt}\,(\Phi \Rightarrow \exists\, x.\ \neg\, term_2) \end{array}}{\Phi \vdash (x; iguard)\{\alpha(ptr + x \times step, \_)\} \sqsubseteq ?\ (x; iguard')\{\alpha(ptr' + x \times step', \_)\} \rightsquigarrow \texttt{None}}$$

Q_Cmp_Comparable

$$\frac{\begin{array}{l} 1.\ term_1 \equiv (ptr = ptr') \land (step = step') \\ 2.\ \texttt{smt}\,(\Phi \Rightarrow term_1) \\ 3.\ \Phi \vdash \texttt{cmp\_min}\,(iguard, iguard') \rightsquigarrow cmp, iguard'' \\ 4.\ term_2 \equiv iguard'' \to \bigwedge(\overline{iarg_i = iarg'_i}^{\,i}) \\ 5.\ \texttt{smt}\,(\Phi \Rightarrow \forall\, x.\ term_2) \end{array}}{\Phi \vdash (x; iguard)\{\alpha(ptr + x \times step, \_)\} \sqsubseteq ?\ (x; iguard')\{\alpha(ptr' + x \times step', \_)\} \rightsquigarrow cmp}$$

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

Req_Eq_PP_Name_Neq

$$\frac{1.\ \alpha_1 \neq \alpha_2}{\Phi \vdash \alpha_1(\_,\_) \equiv \alpha_2(\_,\_) \rightsquigarrow \texttt{false}}$$

Req_Eq_PP_IArg_Neq

$$\frac{1.\ \texttt{smt}\,(\Phi \Rightarrow \neg\,(ptr_1 = ptr_2 \wedge \bigwedge(\overline{iarg_{1\,i} = iarg_{2\,i}}^{\,i})))}{\Phi \vdash \alpha(ptr_1, \overline{iarg_{1\,i}}^{\,i}) \equiv \alpha(ptr_2, \overline{iarg_{2\,i}}^{\,i}) \rightsquigarrow \texttt{false}}$$

Req_Eq_PP_Eq

$$\frac{1.\ \texttt{smt}\,(\Phi \Rightarrow ptr_1 = ptr_2 \wedge \bigwedge(\overline{iarg_{1\,i} = iarg_{2\,i}}^{\,i}))}{\Phi \vdash \alpha(ptr_1, \overline{iarg_{1\,i}}^{\,i}) \equiv \alpha(ptr_2, \overline{iarg_{2\,i}}^{\,i}) \rightsquigarrow \texttt{true}}$$

Req_Eq_QQ_Eq

$$\frac{1.\ \Phi \vdash qpred\_term \sqsubseteq ?\ qpred\_term' \rightsquigarrow \texttt{Eq}}{\Phi \vdash qpred\_term \equiv qpred\_term' \rightsquigarrow \texttt{true}}$$

Req_Eq_QQ_Neq

$$\frac{1.\ \Phi \vdash qpred\_term \sqsubseteq ?\ qpred\_term' \rightsquigarrow opt\_cmp}{\Phi \vdash qpred\_term \equiv qpred\_term' \rightsquigarrow \texttt{false}}$$

$\boxed{\Phi \vdash res \equiv res'}$ resource equality: given constraints Φ, res is equal to res'

Res_Eq_Emp

$$\frac{}{\Phi \vdash \texttt{emp} \equiv \texttt{emp}}$$

Res_Eq_Phi

$$\frac{1.\ \texttt{smt}\,(\Phi \Rightarrow term \leftrightarrow term')}{\Phi \vdash term \equiv term'}$$

Res_Eq_Pred

$$\frac{1.\ \Phi \vdash pred\_term \equiv pred\_term' \rightsquigarrow \texttt{true}}{\Phi \vdash pred\_term(\_) \equiv pred\_term'(\_)}$$

Res_Eq_QPred

$$\frac{1.\ \Phi \vdash qpred\_term \equiv qpred\_term' \rightsquigarrow \texttt{true}}{\Phi \vdash qpred\_term(\_) \equiv qpred\_term'(\_)}$$

Res_Eq_SepConj

$$\frac{1.\ \Phi \vdash res_1 \equiv res_1' \quad 2.\ \Phi \vdash res_2 \equiv res_2'}{\Phi \vdash res_1 * res_2 \equiv res_1' * res_2'}$$

Res_Eq_Exists

$$\frac{1.\ \Phi \vdash res \equiv res'}{\Phi \vdash \exists\, ident{:}\beta.\ res \equiv \exists\, ident{:}\beta.\ res'}$$

Res_Eq_OrdDisj

1. $\texttt{smt}\,(\Phi \Rightarrow \mathit{term}_1 \leftrightarrow \mathit{term}_2)$
2. $\Phi, \mathit{term}_1 \vdash \mathit{res}_{11} \;\equiv\; \mathit{res}_{21}$
3. $\Phi, \neg\,\mathit{term}_1 \vdash \mathit{res}_{21} \;\equiv\; \mathit{res}_{22}$

$$\overline{\Phi \vdash \texttt{if}\ \mathit{term}_1\ \texttt{then}\ \mathit{res}_{11}\ \texttt{else}\ \mathit{res}_{12} \;\equiv\; \texttt{if}\ \mathit{term}_2\ \texttt{then}\ \mathit{res}_{21}\ \texttt{else}\ \mathit{res}_{22}}$$

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

Res_SimpRec_If_True

1. $\texttt{smt}\,(\Phi \Rightarrow \mathit{term})$
2. $\Phi \vdash \texttt{simp\_rec}\,(\mathit{res}_1) \leadsto \mathit{res}_1', \mathit{bool}$

$$\overline{\Phi \vdash \texttt{simp\_rec}\,(\texttt{if}\ \mathit{term}\ \texttt{then}\ \mathit{res}_1\ \texttt{else}\ \mathit{res}_2) \leadsto \mathit{res}_1', \texttt{true}}$$

Res_SimpRec_If_False

1. $\texttt{smt}\,(\Phi \Rightarrow \neg\,\mathit{term})$
2. $\Phi \vdash \texttt{simp\_rec}\,(\mathit{res}_2) \leadsto \mathit{res}_2', \mathit{bool}$

$$\overline{\Phi \vdash \texttt{simp\_rec}\,(\texttt{if}\ \mathit{term}\ \texttt{then}\ \mathit{res}_1\ \texttt{else}\ \mathit{res}_2) \leadsto \mathit{res}_2', \texttt{true}}$$

Res_SimpRec_SepConj

1. $\Phi \vdash \texttt{simp\_rec}\,(\mathit{res}_1) \leadsto \mathit{res}_1', \mathit{bool}_1$
2. $\Phi \vdash \texttt{simp\_rec}\,(\mathit{res}_2) \leadsto \mathit{res}_2', \mathit{bool}_2$

$$\overline{\Phi \vdash \texttt{simp\_rec}\,(\mathit{res}_1 * \mathit{res}_2) \leadsto \mathit{res}_1' * \mathit{res}_2', \mathit{bool}_1 || \mathit{bool}_2}$$

Res_SimpRec_Exists

1. $\Phi \vdash \texttt{simp\_rec}\,(\mathit{res}) \leadsto \mathit{res}', \mathit{bool}$

$$\overline{\Phi \vdash \texttt{simp\_rec}\,(\exists\, y{:}\beta.\ \mathit{res}) \leadsto \exists\, y{:}\beta.\ \mathit{res}', \mathit{bool}}$$

Res_SimpRec_NoChange

$$\overline{\Phi \vdash \texttt{simp\_rec}\,(\mathit{res}) \leadsto \mathit{res}, \texttt{false}}$$

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

$$\frac{1.\ \Phi \vdash \texttt{simp\_rec}\,(res) \leadsto res, \texttt{false}}{\Phi \vdash \texttt{simp}\,(res) \leadsto \texttt{None}}\ \textsc{Simp\_NoSimp}$$

$$\frac{1.\ \Phi \vdash \texttt{simp\_rec}\,(res) \leadsto res', \texttt{true}}{\Phi \vdash \texttt{simp}\,(res) \leadsto res'}\ \textsc{Simp\_Simp}$$

A2.2 Return Type Equality

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

$$\frac{}{\Phi \vdash \texttt{I} \equiv \texttt{I}}\ \textsc{Ret\_Eq\_End}$$

$$\frac{1.\ \Phi \vdash ret \equiv ret'}{\Phi \vdash \Sigma\, y{:}\beta.\ ret \equiv \Sigma\, y{:}\beta.\ ret'}\ \textsc{Ret\_Eq\_Comp}$$

$$\frac{1.\ \Phi \vdash ret \equiv ret'}{\Phi \vdash \exists\, y{:}\beta.\ ret \equiv \exists\, y{:}\beta.\ ret'}\ \textsc{Ret\_Eq\_Log}$$

$$\frac{1.\ \texttt{smt}\,(\Phi \Rightarrow term \leftrightarrow term')}{\Phi \vdash term \wedge ret \equiv term' \wedge ret'}\ \textsc{Ret\_Eq\_Phi}$$

$$\frac{\begin{array}{l}1.\ \Phi \vdash res \equiv res' \\ 2.\ \Phi \vdash ret \equiv ret'\end{array}}{\Phi \vdash res * ret \equiv res' * ret'}\ \textsc{Ret\_Eq\_Res}$$

A2.3 Patterns

$\boxed{pat{:}\beta \leadsto \mathcal{C}\ \texttt{with}\ term}$ computational pattern to context: pat and type β produces context $\mathcal{C}$ and constraint $term$

Pat_Comp_No_Sym_Annot

$$\frac{}{\_:\beta:\beta \rightsquigarrow \cdot\ \texttt{with}\ \_}$$

Pat_Comp_Sym_Annot

$$\frac{}{x:\beta:\beta \rightsquigarrow x:\beta\ \texttt{with}\ x}$$

Pat_Comp_Nil

$$\frac{}{\texttt{Nil}\ \beta(\,):\texttt{list}\ \beta \rightsquigarrow \cdot\ \texttt{with}\ \texttt{nil}}$$

Pat_Comp_Cons

$$\frac{1.\ pat_1:\beta \rightsquigarrow \mathcal{C}_1\ \texttt{with}\ term_1 \quad 2.\ pat_2:\texttt{list}\ \beta \rightsquigarrow \mathcal{C}_2\ \texttt{with}\ term_2}{\texttt{Cons}(pat_1, pat_2):\texttt{list}\ \beta \rightsquigarrow \mathcal{C}_1, \mathcal{C}_2\ \texttt{with}\ term_1 :: term_2}$$

Pat_Comp_Tuple

$$\frac{1.\ pat_i:\beta_i \rightsquigarrow \mathcal{C}_i\ \texttt{with}\ term_i}{\texttt{Tuple}(\,\overline{pat_i}^{\,i}\,):\overline{\beta_i}^{\,i} \rightsquigarrow \overline{\mathcal{C}_i}^{\,i}\ \texttt{with}\ (\,\overline{term_i}^{\,i}\,)}$$

Pat_Comp_Array

$$\frac{1.\ pat_i:\beta \rightsquigarrow \mathcal{C}_i\ \texttt{with}\ term_i}{\texttt{Array}(\,\overline{pat_i}^{\,i}\,):\texttt{array}\ \beta \rightsquigarrow \overline{\mathcal{C}_i}^{\,i}\ \texttt{with}\ [|\ \overline{term_i}^{\,i}\ |]}$$

Pat_Comp_Specified

$$\frac{1.\ pat:\beta \rightsquigarrow \mathcal{C}\ \texttt{with}\ term}{\texttt{Specified}(pat):\beta \rightsquigarrow \mathcal{C}\ \texttt{with}\ term}$$

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

Pat_Sym_Or_Pat_Sym

$$\frac{}{x:\beta \rightsquigarrow x:\beta\ \texttt{with}\ x}$$

Pat_Sym_Or_Pat_Pat

$$\frac{1.\ pat:\beta \rightsquigarrow \mathcal{C}\ \texttt{with}\ term}{pat:\beta \rightsquigarrow \mathcal{C}\ \texttt{with}\ term}$$

$\boxed{\mathcal{L};\Phi \vdash res\_pat:res \rightsquigarrow \mathcal{L}';\Phi';\mathcal{R}'}$ resources pattern to context: given constraints Φ, *res_pat* of type *res* produces contexts $\mathcal{L}';\Phi';\mathcal{R}'$

Pat_Res_Match_Emp

$$\frac{}{\mathcal{L};\Phi \vdash \texttt{emp}:\texttt{emp} \rightsquigarrow \cdot;\cdot;\cdot}$$

Pat_Res_Match_Phi

$$\frac{}{\mathcal{L};\Phi \vdash \texttt{term}:term \rightsquigarrow \mathcal{L}';\Phi', term;\mathcal{R}'}$$

Pat_Res_Match_If_True

$$\frac{1.\ \texttt{smt}\ (\Phi \Rightarrow term) \quad 2.\ \mathcal{L};\Phi \vdash res\_pat:res_1 \rightsquigarrow \mathcal{L};\Phi;\mathcal{R}}{\mathcal{L};\Phi \vdash res\_pat:\texttt{if}\ term\ \texttt{then}\ res_1\ \texttt{else}\ res_2 \rightsquigarrow \mathcal{L};\Phi;\mathcal{R}}$$

PAT_RES_MATCH_IF_FALSE

$$\frac{\begin{array}{l}1.\ \texttt{smt}\,(\Phi \Rightarrow \neg\, \mathit{term})\\ 2.\ \mathcal{L};\Phi \vdash \mathit{res\_pat}{:}\mathit{res}_2 \rightsquigarrow \mathcal{L};\Phi;\mathcal{R}\end{array}}{\mathcal{L};\Phi \vdash \mathit{res\_pat}{:}\texttt{if}\ \mathit{term}\ \texttt{then}\ \mathit{res}_1\ \texttt{else}\ \mathit{res}_2 \rightsquigarrow \mathcal{L};\Phi;\mathcal{R}}$$

PAT_RES_MATCH_VAR

$$\frac{1.\ \texttt{(replace res with res\_norm for normalised contexts)}}{\mathcal{L};\Phi \vdash r{:}\mathit{res} \rightsquigarrow \cdot;\cdot;r{:}\mathit{res}}$$

PAT_RES_MATCH_SEPCONJ

$$\frac{\begin{array}{l}1.\ \mathcal{L};\Phi \vdash \mathit{res\_pat}_1{:}\mathit{res}_1 \rightsquigarrow \mathcal{L}_1;\Phi_1;\mathcal{R}_1\\ 2.\ \mathcal{L};\Phi \vdash \mathit{res\_pat}_2{:}\mathit{res}_2 \rightsquigarrow \mathcal{L}_2;\Phi_2;\mathcal{R}_2\end{array}}{\mathcal{L};\Phi \vdash \langle \mathit{res\_pat}_1, \mathit{res\_pat}_2\rangle{:}\mathit{res}_1 * \mathit{res}_2 \rightsquigarrow \mathcal{L}_1,\mathcal{L}_2;\Phi_1,\Phi_2;\mathcal{R}_1,\mathcal{R}_2}$$

PAT_RES_MATCH_PACK

$$\frac{1.\ \mathcal{L}, x{:}\beta;\Phi \vdash \mathit{res\_pat}{:}x/y(\mathit{res}) \rightsquigarrow \mathcal{L}';\Phi';\mathcal{R}'}{\mathcal{L};\Phi \vdash \texttt{pack}\,(x, \mathit{res\_pat}){:}\exists\, y{:}\beta.\ \mathit{res} \rightsquigarrow \mathcal{L}', x{:}\beta;\Phi';\mathcal{R}'}$$

PAT_RES_MATCH_FOLD

$$\frac{\begin{array}{l}1.\ \alpha \neq \texttt{Owned}\,\langle\tau\rangle\\ 2.\ \alpha\ \equiv\ x_p{:}\_,\ \overline{x_i{:}\_{}_i}^{\,i},\ y{:}\_ \mapsto \mathit{res}\ \in \texttt{Globals}\\ 3.\ \mathcal{L};\Phi \vdash \mathit{res\_pat}{:}[\mathit{oarg}/y, [\,\overline{\mathit{iarg}_i/x_i}^{\,i}\,], \mathit{ptr}/x_p](\mathit{res}) \rightsquigarrow \mathcal{L}';\Phi';\mathcal{R}'\end{array}}{\mathcal{L};\Phi \vdash \texttt{fold}\,(\mathit{res\_pat}){:}\alpha(\mathit{ptr}, \mathit{iargs})(\,\mathit{oarg}\,) \rightsquigarrow \mathcal{L}';\Phi';\mathcal{R}'}$$

$\boxed{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{ret\_pat}{:}\mathit{ret} \rightsquigarrow \mathcal{C}';\mathcal{L}';\Phi';\mathcal{R}'}$ return pattern to context: given context $\mathcal{C};\mathcal{L};\Phi$, *ret_pat* and return type *ret* produces contexts $\mathcal{C}';\mathcal{L}';\Phi';\mathcal{R}'$

PAT_RET_EMPTY

$$\frac{}{\mathcal{C};\mathcal{L};\Phi \vdash\ {:}\texttt{I} \rightsquigarrow \cdot;\cdot;\cdot;\cdot}$$

PAT_RET_COMP

$$\frac{\begin{array}{l}1.\ \mathit{ident\_or\_pat}{:}\beta \rightsquigarrow \mathcal{C}_1\ \texttt{with}\ \mathit{term}_1\\ 2.\ \mathcal{C},\mathcal{C}_1;\mathcal{L};\Phi \vdash \mathit{ret\_pat}{:}\mathit{term}_1/y(\mathit{ret}) \rightsquigarrow \mathcal{C}_2;\mathcal{L}_2;\Phi_2;\mathcal{R}_2\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{comp}\,\mathit{ident\_or\_pat}, \mathit{ret\_pat}{:}\Sigma\, y{:}\beta.\ \mathit{ret} \rightsquigarrow \mathcal{C}_1,\mathcal{C}_2;\mathcal{L}_2;\Phi_2;\mathcal{R}_2}$$

Pat_Ret_Log

$$\frac{1.\ \mathcal{C}; \mathcal{L}, x{:}\beta; \Phi \vdash \mathit{ret\_pat}{:}x/y(\mathit{ret}) \rightsquigarrow \mathcal{C}_2; \mathcal{L}_2; \Phi_2; \mathcal{R}_2}{\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{log}\, x, \mathit{ret\_pat}{:}\exists\, y{:}\beta.\ \mathit{ret} \rightsquigarrow \mathcal{C}_2; y{:}\beta, \mathcal{L}_2; \Phi_2; \mathcal{R}_2}$$

Pat_Ret_Phi

$$\frac{1.\ \mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{ret\_pat}{:}\mathit{ret} \rightsquigarrow \mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'}{\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{ret\_pat}{:}\mathit{term} \wedge \mathit{ret} \rightsquigarrow \mathcal{C}'; \mathcal{L}'; \Phi', \mathit{term}; \mathcal{R}'}$$

Pat_Ret_Res

$$\frac{\begin{array}{l}1.\ \mathcal{L}; \Phi \vdash \mathit{res\_pat}{:}\mathit{res} \rightsquigarrow \mathcal{L}_1; \Phi_1; \mathcal{R}_1\\ 2.\ \mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{ret\_pat}{:}\mathit{ret} \rightsquigarrow \mathcal{C}_2; \mathcal{L}_2; \Phi_2; \mathcal{R}_2\end{array}}{\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{res}\, \mathit{res\_pat}, \mathit{ret\_pat}{:}\mathit{res} * \mathit{ret} \rightsquigarrow \mathcal{C}_2; \mathcal{L}_1, \mathcal{L}_2; \Phi_1, \Phi_2; \mathcal{R}_1, \mathcal{R}_2}$$

$\boxed{\Phi \vdash \mathit{ret\_pat}{:}\mathit{ret} \rightsquigarrow \mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'}$ return pattern to context: given constraints Φ, *ret_pat* and return type *ret* produces contexts $\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'$

Pat_Ret'_Aux

$$\frac{1.\ \cdot; \cdot; \Phi \vdash \mathit{ret\_pat}{:}\mathit{ret} \rightsquigarrow \mathcal{C}'; \mathcal{L}'; \mathit{\Phi}'; \mathcal{R}'}{\Phi \vdash \mathit{ret\_pat}{:}\mathit{ret} \rightsquigarrow \mathcal{C}'; \mathcal{L}'; \mathit{\Phi}'; \mathcal{R}'}$$

A3 Explicit System

A3.1 Pure Expressions

$\boxed{\mathcal{C} \vdash object\_value \Rightarrow \beta}$ object value synthesises: given $\mathcal{C}$, *object_value* synthesises type β

$$\frac{}{\mathcal{C} \vdash mem\_int \Rightarrow \texttt{integer}}\ \textsc{Pure\_Val\_Obj\_Int}$$

$$\frac{}{\mathcal{C} \vdash mem\_ptr \Rightarrow \texttt{pointer}}\ \textsc{Pure\_Val\_Obj\_Ptr}$$

$$\frac{1.\ \overline{\mathcal{C} \vdash object\_value_i \Rightarrow \beta}^{\,i}}{\mathcal{C} \vdash \texttt{array}\,(\,\overline{\texttt{specified}\, object\_value_i}^{\,i}\,) \Rightarrow \texttt{array}\,\beta}\ \textsc{Pure\_Val\_Obj\_Arr}$$

$$\frac{\begin{array}{l}1.\ \texttt{struct}\, tag\ \&\ \overline{member_i : \tau_i}^{\,i} \in \texttt{Globals}\\ 2.\ \overline{\mathcal{C} \vdash mem\_val_i \Rightarrow \beta_{\tau_i}}^{\,i}\end{array}}{\mathcal{C} \vdash (\,\texttt{struct}\, tag)\{\overline{.member_i{:}\tau_i = mem\_val_i}^{\,i}\,\} \Rightarrow \texttt{struct}\, tag}\ \textsc{Pure\_Val\_Obj\_Struct}$$

$\boxed{\mathcal{C} \vdash pval \Rightarrow \beta}$ pure value synthesises: given $\mathcal{C}$, *pval* synthesises type β

$$\frac{1.\ x{:}\beta \in \mathcal{C}}{\mathcal{C} \vdash x \Rightarrow \beta}\ \textsc{Pure\_Val\_Var}$$

$$\frac{1.\ \mathcal{C} \vdash object\_value \Rightarrow \beta}{\mathcal{C} \vdash object\_value \Rightarrow \beta}\ \textsc{Pure\_Val\_Obj}$$

$$\frac{1.\ \mathcal{C} \vdash object\_value \Rightarrow \beta}{\mathcal{C} \vdash \texttt{specified}\, object\_value \Rightarrow \beta}\ \textsc{Pure\_Val\_Loaded}$$

$$\frac{}{\mathcal{C} \vdash \texttt{Unit} \Rightarrow \texttt{unit}}\ \textsc{Pure\_Val\_Unit}$$

$$\frac{}{\mathcal{C} \vdash \texttt{True} \Rightarrow \texttt{bool}}\ \textsc{Pure\_Val\_True}$$

$$\frac{}{\mathcal{C} \vdash \texttt{False} \Rightarrow \texttt{bool}}\ \textsc{Pure\_Val\_False}$$

$$\frac{1.\ \overline{\mathcal{C} \vdash value_i \Rightarrow \beta}^{\,i}}{\mathcal{C} \vdash \beta[\,\overline{value_i}^{\,i}\,] \Rightarrow \texttt{list}\,\beta}\ \textsc{Pure\_Val\_List}$$

$$\frac{1.\ \overline{\mathcal{C} \vdash value_i \Rightarrow \beta_i}^{\,i}}{\mathcal{C} \vdash (\,\overline{value_i}^{\,i}\,) \Rightarrow \overline{\beta_i}^{\,i}}\ \textsc{Pure\_Val\_Tuple}$$

$$\frac{}{\mathcal{C} \vdash \texttt{Nil}\,\beta(\,) \Rightarrow \texttt{list}\,\beta}\ \textsc{Pure\_Val\_Ctor\_Nil}$$

PURE_VAL_CTOR_CONS

$$\frac{1.\ \mathcal{C} \vdash pval_1 \Rightarrow \beta \qquad 2.\ \mathcal{C} \vdash pval_2 \Rightarrow \texttt{list}\ \beta}{\mathcal{C} \vdash \texttt{Cons}(pval_1, pval_2) \Rightarrow \texttt{list}\ \beta}$$

PURE_VAL_CTOR_TUPLE

$$\frac{1.\ \mathcal{C} \vdash pval_i \Rightarrow \beta_i}{\mathcal{C} \vdash \texttt{Tuple}(\ \overline{pval_i}^{\,i}\ ) \Rightarrow \overline{\beta_i}^{\,i}}$$

PURE_VAL_CTOR_ARRAY

$$\frac{1.\ \mathcal{C} \vdash pval_i \Rightarrow \beta}{\mathcal{C} \vdash \texttt{Array}(\ \overline{pval_i}^{\,i}\ ) \Rightarrow \texttt{array}\ \beta}$$

PURE_VAL_CTOR_SPECIFIED

$$\frac{1.\ \mathcal{C} \vdash pval \Rightarrow \beta}{\mathcal{C} \vdash \texttt{Specified}(pval) \Rightarrow \beta}$$

PURE_VAL_STRUCT

$$\frac{1.\ \texttt{struct}\ tag\ \&\ \overline{member_i : \tau_i}^{\,i}\ \in \texttt{Globals} \qquad 2.\ \mathcal{C} \vdash pval_i \Rightarrow \beta_{\tau_i}}{\mathcal{C} \vdash (\ \texttt{struct}\ tag)\{\overline{.member_i = pval_i}^{\,i}\ \} \Rightarrow \texttt{struct}\ tag}$$

$\boxed{\mathcal{C}; \mathcal{L}; \Phi \vdash pexpr \Rightarrow pure\_ret}$ pure expression synthesises: given $\mathcal{C}; \mathcal{L}; \Phi$, $pexpr$ synthesises a pure (non-resourceful) return type $pure\_ret$

PURE_EXPR_VAL

$$\frac{1.\ \mathcal{C} \vdash pval \Rightarrow \beta}{\mathcal{C}; \mathcal{L}; \Phi \vdash pval \Rightarrow \Sigma\ y{:}\beta.\ y = pval \land \mathtt{I}}$$

PURE_EXPR_ARRAY_SHIFT

$$\frac{1.\ \mathcal{C} \vdash pval_1 \Rightarrow \texttt{pointer} \qquad 2.\ \mathcal{C} \vdash pval_2 \Rightarrow \texttt{integer}}{\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{array\_shift}\ (pval_1, \tau, pval_2) \Rightarrow \Sigma\ y{:}\texttt{pointer}.\ y = pval_1 +_{\mathrm{ptr}} (pval_2 \times \mathrm{size\_of}(\tau)) \land \mathtt{I}}$$

PURE_EXPR_MEMBER_SHIFT

$$\frac{1.\ \mathcal{C} \vdash pval \Rightarrow \texttt{pointer} \qquad 2.\ \texttt{struct}\ tag\ \&\ \overline{member_i : \tau_i}^{\,i}\ \in \texttt{Globals}}{\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{member\_shift}\ (pval, tag, member_j) \Rightarrow \Sigma\ y{:}\texttt{pointer}.\ y = pval +_{\mathrm{ptr}} \mathrm{offset\_of}_{tag}(member_j) \land \mathtt{I}}$$

Pure_Expr_Not

$$\frac{1.\ \mathcal{C} \vdash \mathit{pval} \Rightarrow \texttt{bool}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{not}\,(\mathit{pval}) \Rightarrow \Sigma\, y{:}\texttt{bool}.\ y = \neg\,\mathit{pval} \wedge \texttt{I}}$$

Pure_Expr_Arith_Binop

$$\frac{\begin{array}{l}1.\ \mathcal{C} \vdash \mathit{pval}_1 \Rightarrow \texttt{integer}\\ 2.\ \mathcal{C} \vdash \mathit{pval}_2 \Rightarrow \texttt{integer}\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{pval}_1\ \mathit{binop}_{arith}\ \mathit{pval}_2 \Rightarrow \Sigma\, y{:}\texttt{integer}.\ y = (\mathit{pval}_1\ \mathit{binop}_{arith}\ \mathit{pval}_2) \wedge \texttt{I}}$$

Pure_Expr_Rel_Binop

$$\frac{\begin{array}{l}1.\ \mathcal{C} \vdash \mathit{pval}_1 \Rightarrow \texttt{integer}\\ 2.\ \mathcal{C} \vdash \mathit{pval}_2 \Rightarrow \texttt{integer}\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{pval}_1\ \mathit{binop}_{rel}\ \mathit{pval}_2 \Rightarrow \Sigma\, y{:}\texttt{bool}.\ y = (\mathit{pval}_1\ \mathit{binop}_{rel}\ \mathit{pval}_2) \wedge \texttt{I}}$$

Pure_Expr_Bool_Binop

$$\frac{\begin{array}{l}1.\ \mathcal{C} \vdash \mathit{pval}_1 \Rightarrow \texttt{bool}\\ 2.\ \mathcal{C} \vdash \mathit{pval}_2 \Rightarrow \texttt{bool}\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{pval}_1\ \mathit{binop}_{bool}\ \mathit{pval}_2 \Rightarrow \Sigma\, y{:}\texttt{bool}.\ y = (\mathit{pval}_1\ \mathit{binop}_{bool}\ \mathit{pval}_2) \wedge \texttt{I}}$$

Pure_Expr_Call

$$\frac{\begin{array}{l}1.\ \mathit{name}{:}\mathit{pure\_fun} \equiv \overline{x_i}^{\,i} \mapsto \mathit{tpexpr} \in \texttt{Globals}\\ 2.\ \mathcal{C};\mathcal{L};\Phi;\cdot \vdash \overline{\mathit{pval}_i}^{\,i} :: \mathit{pure\_fun} \gg \mathit{pure\_ret}\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{name}(\,\overline{\mathit{pval}_i}^{\,i}\,) \Rightarrow \mathit{pure\_ret}}$$

Pure_Expr_Assert_Undef

$$\frac{\begin{array}{l}1.\ \mathcal{C} \vdash \mathit{pval} \Rightarrow \texttt{bool}\\ 2.\ \texttt{smt}\,(\Phi \Rightarrow \mathit{pval})\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{assert\_undef}\,(\mathit{pval},\ \mathit{UB\_name}) \Rightarrow \Sigma\, y{:}\texttt{unit}.\ y = \texttt{unit} \wedge \texttt{I}}$$

Pure_Expr_Bool_To_Integer

$$\frac{1.\ \mathcal{C} \vdash \mathit{pval} \Rightarrow \texttt{bool}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{bool\_to\_integer}\,(\mathit{pval}) \Rightarrow \Sigma\, y{:}\texttt{integer}.\ y = \texttt{if}\ \mathit{pval}\ \texttt{then}\ \mathit{1}\ \texttt{else}\ \mathit{0} \wedge \texttt{I}}$$

PURE_EXPR_WRAPI

1. $\mathcal{C} \vdash pval \Rightarrow \texttt{integer}$
2. $abbrev_1 \equiv \text{max\_int}_\tau - \text{min\_int}_\tau + 1$
3. $abbrev_2 \equiv pval \text{ rem } abbrev_1$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{wrapI}\,(\tau, pval) \Rightarrow \Sigma\, y{:}\texttt{integer}.\; y = \texttt{if}\; abbrev_2 \leq \text{max\_int}_\tau\; \texttt{then}\; abbrev_2\; \texttt{else}\; abbrev_2 - abbrev_1 \wedge \texttt{I}}$$

$\boxed{\mathcal{C}; \mathcal{L}; \Phi \vdash tpval \Leftarrow pure\_ret}$ pure top-level value checks: given $\mathcal{C}; \mathcal{L}; \Phi$, *tpval* checks against *pure_ret*

PURE_TOP_VAL_DONE

1. $\mathcal{C} \vdash pval \Rightarrow \beta$
2. $\texttt{smt}\,(\Phi \Rightarrow pval/y(term))$

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

PURE_TOP_VAL_UNDEF

1. $\texttt{smt}\,(\Phi \Rightarrow \texttt{false})$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{undef}\;\; UB\_name \Leftarrow \Sigma\, \_{:}\_.\; \_ \wedge \texttt{I}}$$

PURE_TOP_VAL_ERROR

1. $\texttt{smt}\,(\Phi \Rightarrow \texttt{false})$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{error}\,(string, pval) \Leftarrow \Sigma\, \_{:}\_.\; \_ \wedge \texttt{I}}$$

$\boxed{\mathcal{C}; \mathcal{L}; \Phi \vdash tpexpr \Leftarrow pure\_ret}$ pure top-level expression checks: given $\mathcal{C}; \mathcal{L}; \Phi$, *tpexpr* checks against *pure_ret*

PURE_TOP_IF

1. $\mathcal{C} \vdash pval \Rightarrow \texttt{bool}$
2. $\mathcal{C}; \mathcal{L}; \Phi, pval = \texttt{true} \vdash tpexpr_1 \Leftarrow \Sigma\, y{:}\beta.\; term \wedge \texttt{I}$
3. $\mathcal{C}; \mathcal{L}; \Phi, pval = \texttt{false} \vdash tpexpr_2 \Leftarrow \Sigma\, y{:}\beta.\; term \wedge \texttt{I}$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{if}\; pval\; \texttt{then}\; tpexpr_1\; \texttt{else}\; tpexpr_2 \Leftarrow \Sigma\, y{:}\beta.\; term \wedge \texttt{I}}$$

PURE_TOP_LET

1. $\mathcal{C}; \mathcal{L}; \Phi \vdash pexpr \Rightarrow \Sigma\, y_1{:}\beta_1.\; term_1 \wedge \texttt{I}$
2. $ident\_or\_pat{:}\beta_1 \leadsto \mathcal{C}_1\; \texttt{with}\; term$
3. $\mathcal{C}, \mathcal{C}_1; \mathcal{L}; \Phi, term/y_1(term_1) \vdash tpexpr \Leftarrow \Sigma\, y_2{:}\beta_2.\; term_2 \wedge \texttt{I}$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{let}\; ident\_or\_pat = pexpr\; \texttt{in}\; tpexpr \Leftarrow \Sigma\, y_2{:}\beta_2.\; term_2 \wedge \texttt{I}}$$

PURE_TOP_LETT

1. $\mathcal{C}; \mathcal{L}; \Phi \vdash tpexpr_1 \Leftarrow \Sigma\, y_1{:}\beta_1.\; term_1 \wedge \texttt{I}$
2. $ident\_or\_pat{:}\beta_1 \leadsto \mathcal{C}_1\; \texttt{with}\; term$
3. $\mathcal{C}, \mathcal{C}_1; \mathcal{L}; \Phi, term/y_1(term_1) \vdash tpexpr \Leftarrow \Sigma\, y_2{:}\beta_2.\; term_2 \wedge \texttt{I}$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{let}\; ident\_or\_pat{:}pure\_ret = tpexpr_1\; \texttt{in}\; tpexpr_2 \Leftarrow \Sigma\, y_2{:}\beta_2.\; term_2 \wedge \texttt{I}}$$

PURE_TOP_CASE

$$\frac{\begin{array}{l}1.\ \mathcal{C} \vdash \mathit{pval} \Rightarrow \beta_1 \\ 2.\ \overline{\mathit{pat}_i{:}\beta_1 \rightsquigarrow \mathcal{C}_i\ \texttt{with}\ \mathit{term}_i}^{\,i} \\ 3.\ \overline{\mathcal{C}, \mathcal{C}_i; \mathcal{L}; \Phi, \mathit{term}_i = \mathit{pval} \vdash \mathit{tpexpr}_i \Leftarrow \Sigma\, y_2{:}\beta_2.\ \mathit{term}_2 \wedge \mathtt{I}}^{\,i}\end{array}}{\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{case}\ \mathit{pval}\ \texttt{of}\ \overline{\mid \mathit{pat}_i \Rightarrow \mathit{tpexpr}_i}^{\,i}\ \texttt{end} \Leftarrow \Sigma\, y_2{:}\beta_2.\ \mathit{term}_2 \wedge \mathtt{I}}$$

A3.2 Resource Terms

$\boxed{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{pred\_ops} \Rightarrow\ \mathit{res}}$ resource (q)predicate operation term synthesis: given $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}$, *pred_ops* synthesises resource *res*

RES_SYN_PREDOPS_ITERATE

$$\frac{\begin{array}{l}1.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term} \Rightarrow \mathit{ptr} \overset{\mathit{init}}{\mapsto}_{\texttt{array}\ n\ \tau}\ \mathit{value} \\ 2.\ \mathit{oarg}[x].\mathit{init} \equiv \mathit{init}[x] \\ 3.\ \mathit{oarg}[x].\mathit{value} \equiv \mathit{value}[x]\end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \texttt{iterate}\,(\mathit{res\_term}, n) \Rightarrow (\circledast\, x.\ 0 \le x \wedge x \le n - 1 \Rightarrow \mathit{ptr} + x \times \mathrm{size\_of}(\tau) \overset{\mathit{oarg}[x].\mathit{init}}{\mapsto}_{\tau}\ \mathit{oarg}[x].\mathit{value})}$$

RES_SYN_PREDOPS_CONGEAL

$$\frac{\begin{array}{l}1.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term} \Rightarrow (\circledast\, x.\ \mathit{iguard} \Rightarrow \mathit{ptr} + x \times \mathrm{size\_of}(\tau) \overset{\mathit{oarg}[x].\mathit{init}}{\mapsto}_{\tau}\ \mathit{oarg}[x].\mathit{value}) \\ 2.\ \texttt{smt}\,(\Phi \Rightarrow \forall\, x.\ \mathit{iguard} \leftrightarrow (0 \le x \wedge x \le n - 1)) \\ 3.\ \mathit{init}[x] \equiv \mathit{oarg}[x].\mathit{init} \\ 4.\ \mathit{value}[x] \equiv \mathit{oarg}[x].\mathit{value}\end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \texttt{congeal}\,(\mathit{res\_term}, n) \Rightarrow \mathit{ptr} \overset{\mathit{init}}{\mapsto}_{\texttt{array}\ n\ \tau}\ \mathit{value}}$$

Res_Syn_PredOps_Explode

$$\frac{\begin{array}{l} 1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{res\_term} \Rightarrow \mathit{ptr} \mapsto^{\mathit{init}}_{\texttt{struct}\,\mathit{tag}} \mathit{value} \\ 2.\ \texttt{struct}\,\mathit{tag}\ \&\ \overline{\mathit{member}_i : \tau_i}^{\,i} \in \texttt{Globals} \end{array}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \texttt{explode}\,(\mathit{res\_term}) \Rightarrow \circledast\,(\overline{\mathit{ptr} +_{\text{ptr}} \text{offset\_of}_{\mathit{tag}}(\mathit{member}_i) \mapsto^{\mathit{init}.\mathit{member}_i}_{\tau_i} \mathit{value}.\mathit{member}_i}^{\,i})}$$

Res_Syn_PredOps_Implode

$$\frac{\begin{array}{l} 1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{res\_term} \Rightarrow \circledast\,(\overline{\mathit{ptr}_i \mapsto^{\mathit{init}_i}_{\tau_i} \mathit{value}_i}^{\,i}) \\ 2.\ \texttt{struct}\,\mathit{tag}\ \&\ \overline{\mathit{member}_i : \tau_i}^{\,i} \in \texttt{Globals} \\ 3.\ \overline{\mathit{init}.\mathit{member}_i \equiv \mathit{init}_i}^{\,i} \\ 4.\ \overline{\mathit{value}.\mathit{member}_i \equiv \mathit{value}_i}^{\,i} \\ 5.\ \mathit{ptr} \equiv \mathit{ptr}_0 - \text{offset\_of}_{\mathit{tag}}(\mathit{member}_0) \\ 6.\ \texttt{smt}\,(\Phi \Rightarrow \bigwedge(\overline{\mathit{ptr} = \mathit{ptr}_i - \text{offset\_of}_{\mathit{tag}}(\mathit{member}_i)}^{\,i})) \end{array}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \texttt{implode}\,(\mathit{res\_term}, \mathit{tag}) \Rightarrow \mathit{ptr} \mapsto^{\mathit{init}}_{\texttt{struct}\,\mathit{tag}} \mathit{value}}$$

Res_Syn_PredOps_Break

$$\frac{\begin{array}{l} 1.\ \mathcal{C};\mathcal{L} \vdash \mathit{term} \Rightarrow \texttt{integer} \\ 2.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{res\_term} \Rightarrow (x; \mathit{iguard})\{\alpha(\mathit{ptr} + x \times \mathit{step}, \mathit{iargs})\}(\mathit{oarg}) \\ 3.\ \texttt{smt}\,(\Phi \Rightarrow \mathit{term}/x(\mathit{iguard})) \\ 4.\ \mathit{qpred} \equiv (x; \mathit{iguard} \wedge (x \neq \mathit{term}))\{\alpha(\mathit{ptr} + x \times \mathit{step}, \mathit{iargs})\}(\mathit{oarg}) \\ 5.\ \mathit{pred} \equiv \alpha(\mathit{ptr} + (\mathit{term} \times \mathit{step}), \mathit{term}/x(\mathit{iargs}))(\mathit{oarg}[\mathit{term}]) \end{array}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \texttt{break}\,(\mathit{res\_term}, \mathit{term}) \Rightarrow \mathit{qpred} * \mathit{pred}}$$

RES_SYN_PREDOPS_GLUE

$$\begin{array}{l}
1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{res_term} \Rightarrow (x;\mathit{iguard})\{\alpha(\mathit{ptr}_1 + x \times \mathit{step}, \overline{\mathit{iarg}_{1\,i}}^{\,i})\}(\mathit{oarg}_1) * \alpha(\mathit{ptr}_2, \overline{\mathit{iarg}_{2\,i}}^{\,i})(\mathit{oarg}_2) \\
2.\ \mathit{term} \equiv (\mathit{ptr}_2 - \mathit{ptr}_1)/\mathit{step} \\
3.\ \mathtt{smt}\,(\Phi \Rightarrow \bigwedge(\overline{(\mathit{term}/x(\mathit{iarg}_{1\,i})) = \mathit{iarg}_{2\,i}}^{\,i}))
\end{array}$$
$$\overline{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathtt{glue}\,(\mathit{res\_term}) \Rightarrow (x;\mathit{iguard} \vee x = \mathit{term})\{\alpha(\mathit{ptr}_1 + x \times \mathit{step}, \overline{\mathit{iarg}_{1\,i}}^{\,i})\}(\mathit{oarg}_1[\mathit{term}] := \mathit{oarg}_2)}$$

RES_SYN_PREDOPS_INJ

$$\begin{array}{l}
1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{res_term} \Rightarrow \alpha(\mathit{ptr}_2, \overline{\mathit{iarg}_{2\,i}}^{\,i})(\mathit{oarg}) \\
2.\ \mathit{term} \equiv (\mathit{ptr}_2 - \mathit{ptr}_1)/\mathit{step} \\
3.\ \mathtt{smt}\,(\Phi \Rightarrow \bigwedge(\overline{(\mathit{term}/x(\mathit{iarg}_{1\,i})) = \mathit{iarg}_{2\,i}}^{\,i})) \\
4.\ \mathcal{C};\mathcal{L} \vdash \mathit{oarg} \Rightarrow \beta
\end{array}$$
$$\overline{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathtt{inj}\,(\mathit{res\_term}, \mathit{ptr}_1, \mathit{step}, x.\,\overline{\mathit{iarg}_1}^{\,i}) \Rightarrow (x; x = \mathit{term})\{\alpha(\mathit{ptr}_1 + x \times \mathit{step}, \overline{\mathit{iarg}_{1\,i}}^{\,i})\}((\mathtt{default\,array}\,\beta)[\mathit{term}] := \mathit{oarg})}$$

RES_SYN_PREDOPS_SPLIT

$$\begin{array}{l}
1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{res_term} \Rightarrow (x;\mathit{iguard}')\{\alpha(\mathit{ptr} + x \times \mathit{step}, \mathit{iargs})\}(\mathit{oarg}) \\
2.\ \mathtt{smt}\,(\Phi \Rightarrow \forall\, x.\ \mathit{iguard} \rightarrow \mathit{iguard}') \\
3.\ \mathit{iguard}_2 \equiv \mathit{iguard}' \wedge \neg\, \mathit{iguard}
\end{array}$$
$$\overline{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathtt{split}\,(\mathit{res\_term}, \mathit{iguard}) \Rightarrow (x;\mathit{iguard})\{\alpha(\mathit{ptr} + x \times \mathit{step}, \mathit{iargs})\}(\mathit{oarg}) * (x;\mathit{iguard}_2)\{\alpha(\mathit{ptr} + x \times \mathit{step}, \mathit{iargs})\}(\mathit{oarg})}$$

$\boxed{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{res\_term} \Rightarrow \mathit{res}}$ resource term synthesises: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, $\mathit{res\_term}$ synthesises resource res

RES_SYN_EMP

$$\overline{\mathcal{C};\mathcal{L};\Phi;\cdot \vdash \mathtt{emp} \Rightarrow \mathtt{emp}}$$

RES_SYN_VAR

$$\frac{1.\ \Phi \vdash \mathtt{simp}\,(\mathit{res}) \leadsto \mathtt{None}}{\mathcal{C};\mathcal{L};\Phi;r{:}\mathit{res} \vdash r \Rightarrow \mathit{res}}$$

RES_SYN_VARSIMP

$$\frac{1.\ \Phi \vdash \mathtt{simp}\,(\mathit{res}) \leadsto \mathit{res}'}{\mathcal{C};\mathcal{L};\Phi;r{:}\mathit{res} \vdash r \Rightarrow \mathit{res}'}$$

RES_SYN_PRED

$$\frac{\begin{array}{l} 1.\ \mathit{pred\_term}' \equiv \alpha(\mathit{ptr}', \overline{\mathit{iarg}'_i}^{\,i}) \\ 2.\ \alpha \equiv \_{:}\mathtt{pointer}, \overline{\_{:}\beta_i}^{\,i} \mapsto \_ \in \mathtt{Globals} \\ 3.\ \mathcal{C};\mathcal{L} \vdash \mathit{ptr}' \Rightarrow \mathtt{pointer} \\ 4.\ \overline{\mathcal{C};\mathcal{L} \vdash \mathit{iarg}'_i \Rightarrow \beta_i}^{\,i} \\ 5.\ \Phi \vdash \mathit{pred\_term} \equiv \mathit{pred\_term}' \rightsquigarrow \mathtt{true} \end{array}}{\mathcal{C};\mathcal{L};\Phi;\_{:}\mathit{pred\_term}(\mathit{oarg}) \vdash \mathit{pred\_term}' \Rightarrow \mathit{pred\_term}(\mathit{oarg})}$$

RES_SYN_FOLD

$$\frac{\begin{array}{l} 1.\ \mathit{pred\_term} \equiv \alpha(\mathit{ptr}, \overline{\mathit{iarg}_i}^{\,i}) \\ 2.\ \alpha \neq \mathtt{Owned}\langle\tau\rangle \\ 3.\ \alpha \equiv x_p{:}\mathtt{pointer}, \overline{x_i{:}\beta_i}^{\,i}, y{:}\mathtt{record}\ \overline{\mathit{tag}_j{:}\beta'_j}^{\,j} \mapsto \mathit{res} \in \mathtt{Globals} \\ 4.\ \mathcal{C};\mathcal{L} \vdash \mathit{ptr} \Rightarrow \mathtt{pointer} \\ 5.\ \overline{\mathcal{C};\mathcal{L} \vdash \mathit{iarg}_i \Rightarrow \beta_i}^{\,i} \\ 6.\ \mathcal{C};\mathcal{L} \vdash \mathit{oarg} \Rightarrow \mathtt{record}\ \overline{\mathit{tag}_j{:}\beta'_j}^{\,j} \\ 7.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{res\_term} \Leftarrow [\mathit{oarg}/y, [\overline{\mathit{iarg}_i/x_i}^{\,i}], \mathit{ptr}/x_p](\mathit{res}) \end{array}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathtt{fold}\ \mathit{res\_term}{:}\mathit{pred\_term}(\mathit{oarg}) \Rightarrow \mathit{pred\_term}(\mathit{oarg})}$$

RES_SYN_QPRED

$$\frac{\begin{array}{l} 1.\ \mathit{qpred\_term}' \equiv (x;\mathit{iguard}')\{\alpha(\mathit{ptr}' + x \times \mathit{step}, \overline{\mathit{iarg}'_i}^{\,i})\} \\ 2.\ \alpha \equiv \_{:}\mathtt{pointer}, \overline{\_{:}\beta_i}^{\,i} \mapsto \_ \in \mathtt{Globals} \\ 3.\ \mathcal{C};\mathcal{L} \vdash \mathit{ptr}' \Rightarrow \mathtt{pointer} \\ 4.\ \overline{\mathcal{C};\mathcal{L} \vdash \mathit{iarg}'_i \Rightarrow \beta_i}^{\,i} \\ 5.\ \Phi \vdash \mathit{qpred\_term} \equiv \mathit{qpred\_term}' \rightsquigarrow \mathtt{true} \end{array}}{\mathcal{C};\mathcal{L};\Phi;\_{:}\mathit{qpred\_term}(\mathit{oarg}) \vdash \mathit{qpred\_term}' \Rightarrow \mathit{qpred\_term}(\mathit{oarg})}$$

RES_SYN_PREDOPS

$$\frac{1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{pred\_ops} \Rightarrow \mathit{res}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{pred\_ops} \Rightarrow \mathit{res}}$$

RES_SYN_SEPCONJ

$$\frac{\begin{array}{l} 1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R}_1 \vdash \mathit{res\_term}_1 \Rightarrow \mathit{res}_1 \\ 2.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R}_2 \vdash \mathit{res\_term}_2 \Rightarrow \mathit{res}_2 \end{array}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R}_1,\mathcal{R}_2 \vdash \langle \mathit{res\_term}_1, \mathit{res\_term}_2 \rangle \Rightarrow \mathit{res}_1 * \mathit{res}_2}$$

$\boxed{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{res\_term} \Leftarrow \mathit{res}}$ resource term checks: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, *res_term* checks against resource *res*

$$\dfrac{1.\ \texttt{smt}\,(\Phi \Rightarrow \mathit{term})}{\mathcal{C}; \mathcal{L}; \Phi; \cdot \vdash \texttt{term} \Leftarrow \mathit{term}}\ \textsc{Res\_Chk\_Phi}$$

$$\dfrac{\begin{array}{l}1.\ \mathcal{C}; \mathcal{L} \vdash \mathit{oarg} \Rightarrow \beta \\ 2.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term} \Leftarrow \mathit{oarg}/y(\mathit{res})\end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \texttt{pack}\,(\mathit{oarg}, \mathit{res\_term}) \Leftarrow \exists\, y{:}\beta.\ \mathit{res}}\ \textsc{Res\_Chk\_Pack}$$

$$\dfrac{\begin{array}{l}1.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}_1 \vdash \mathit{res\_term}_1 \Leftarrow \mathit{res}_1 \\ 2.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}_2 \vdash \mathit{res\_term}_2 \Leftarrow \mathit{res}_2\end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}_1, \mathcal{R}_2 \vdash \langle \mathit{res\_term}_1, \mathit{res\_term}_2 \rangle \Leftarrow \mathit{res}_1 * \mathit{res}_2}\ \textsc{Res\_Chk\_SepConj}$$

$$\dfrac{\begin{array}{l}1.\ \texttt{smt}\,(\Phi \Rightarrow \mathit{term}) \\ 2.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term} \Leftarrow \mathit{res}_1\end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term} \Leftarrow \texttt{if}\ \mathit{term}\ \texttt{then}\ \mathit{res}_1\ \texttt{else}\ \mathit{res}_2}\ \textsc{Res\_Chk\_If\_True}$$

$$\dfrac{\begin{array}{l}1.\ \texttt{smt}\,(\Phi \Rightarrow \neg\, \mathit{term}) \\ 2.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term} \Leftarrow \mathit{res}_2\end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term} \Leftarrow \texttt{if}\ \mathit{term}\ \texttt{then}\ \mathit{res}_1\ \texttt{else}\ \mathit{res}_2}\ \textsc{Res\_Chk\_If\_False}$$

$$\dfrac{\begin{array}{l}1.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term} \Rightarrow \mathit{res} \\ 2.\ \Phi \vdash \mathit{res} \ \equiv\ \mathit{res}'\end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term} \Leftarrow \mathit{res}'}\ \textsc{Res\_Chk\_Switch}$$

A3.3 Spine Judgement

$\boxed{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{spine} :: \mathit{fun} \gg \mathit{ret}}$ function call spine checks: given $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}$, compatible *spine*, *fun* produces an *ret*

EXPL_SPINE_RET

$$\frac{}{\mathcal{C};\mathcal{L};\Phi;\cdot \vdash \; ::ret \gg ret}$$

EXPL_SPINE_COMP

$$\frac{1.\ \mathcal{C} \vdash pval \Rightarrow \beta \quad 2.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash spine :: pval/x(fun) \gg ret}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash pval, spine :: \Pi\, x{:}\beta.\ fun \gg ret}$$

EXPL_SPINE_LOG

$$\frac{1.\ \mathcal{C};\mathcal{L} \vdash oarg \Rightarrow \beta \quad 2.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash spine :: oarg/x(fun) \gg ret}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash oarg, spine :: \forall\, x{:}\beta.\ fun \gg ret}$$

EXPL_SPINE_PHI

$$\frac{1.\ \texttt{smt}\,(\Phi \Rightarrow term) \quad 2.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash spine :: fun \gg ret}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash spine :: term \supset fun \gg ret}$$

EXPL_SPINE_RES

$$\frac{1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R}_1 \vdash res\_term \Leftarrow res \quad 2.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R}_2 \vdash spine :: fun \gg ret}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R}_1,\mathcal{R}_2 \vdash res\_term, spine :: res \multimap fun \gg ret}$$

A3.4 Indet. seq. expressions

$\boxed{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash action \Rightarrow ret}$ memory action synthesises: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, *action* synthesises return type ret

EXPL_IS_ACTION_CREATE

$$\frac{1.\ \mathcal{C} \vdash pval \Rightarrow \texttt{integer} \quad 2.\ term \equiv \texttt{representable}\,(\tau*, y_p) \wedge \texttt{alignedI}\,(pval, y_p)}{\mathcal{C};\mathcal{L};\Phi;\cdot \vdash \texttt{create}\,(pval,\tau) \Rightarrow \Sigma\, y_p{:}\texttt{pointer}.\ term \wedge (y_p \overset{\texttt{const}_\tau\texttt{false}}{\mapsto_\tau} \texttt{default}\,\beta_\tau) * \texttt{I}}$$

EXPL_IS_ACTION_LOAD

$$\frac{1.\ \mathcal{C} \vdash pval_0 \Rightarrow \texttt{pointer} \quad 2.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash res\_term \Rightarrow term \overset{init}{\mapsto_\tau} value \quad 3.\ \texttt{smt}\,(\Phi \Rightarrow (term = pval_0) \wedge (init = \texttt{const}_\tau\texttt{true}))}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \texttt{load}\,(\tau, pval_0, \_, res\_term) \Rightarrow \Sigma\, y{:}\beta_\tau.\ y = value \wedge (pval_0 \overset{\texttt{const}_\tau\texttt{true}}{\mapsto_\tau} value) * \texttt{I}}$$

Expl_Is_Action_Store

1. $\mathcal{C} \vdash pval_0 \Rightarrow \texttt{pointer}$
2. $\mathcal{C} \vdash pval_1 \Rightarrow \beta_\tau$
3. $\texttt{smt}\,(\Phi \Rightarrow \texttt{representable}\,(\tau, pval_1))$
4. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash res\_term \Rightarrow term \mapsto_\tau \_$
5. $\texttt{smt}\,(\Phi \Rightarrow term = pval_0)$

$$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \texttt{store}\,(\_, \tau, pval_0, pval_1, \_, res\_term) \Rightarrow \Sigma\ \_{:}\texttt{unit}.\ (pval_0 \overset{\texttt{const}_\tau\texttt{true}}{\mapsto_\tau} pval_1) * \texttt{I}$$

Expl_Is_Action_Kill_Static

1. $\mathcal{C} \vdash pval \Rightarrow \texttt{pointer}$
2. $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash res\_term \Rightarrow term \mapsto_\tau \_$
3. $\texttt{smt}\,(\Phi \Rightarrow term = pval)$

$$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \texttt{kill}\,(\texttt{static}\,\tau, pval, res\_term) \Rightarrow \Sigma\ \_{:}\texttt{unit}.\ \texttt{I}$$

$\boxed{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash memop \Rightarrow ret}$ memory operation synthesises: given $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}$, *memop* synthesises return type *ret*

Expl_Is_Memop_Rel_Binop

1. $\mathcal{C} \vdash pval_1 \Rightarrow \texttt{pointer}$
2. $\mathcal{C} \vdash pval_2 \Rightarrow \texttt{pointer}$

$$\mathcal{C}; \mathcal{L}; \Phi; \cdot \vdash pval_1\ binop_{rel}\ pval_2 \Rightarrow \Sigma\ y{:}\texttt{bool}.\ y = (pval_1\ binop_{rel}\ pval_2) \wedge \texttt{I}$$

Expl_Is_Memop_IntFromPtr

1. $\mathcal{C} \vdash pval \Rightarrow \texttt{pointer}$

$$\mathcal{C}; \mathcal{L}; \Phi; \cdot \vdash \texttt{intFromPtr}\,(\tau_1, \tau_2, pval) \Rightarrow \Sigma\ y{:}\texttt{integer}.\ y = \texttt{cast\_ptr\_to\_int}\,pval \wedge \texttt{I}$$

EXPL_IS_MEMOP_PTRFROMINT

$$\frac{1.\ \mathcal{C} \vdash pval \Rightarrow \texttt{integer}}{\mathcal{C};\mathcal{L};\Phi;\cdot \vdash \texttt{ptrFromInt}\,(\tau_1, \tau_2, pval) \Rightarrow \Sigma\, y{:}\texttt{pointer}.\ y = \texttt{cast\_int\_to\_ptr}\ pval \wedge \texttt{I}}$$

EXPL_IS_MEMOP_PTRVALIDFORDEREF

$$\frac{\begin{array}{l}1.\ \mathcal{C} \vdash pval \Rightarrow \texttt{pointer}\\ 2.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash res\_term \Rightarrow term \overset{init}{\mapsto_\tau} value\\ 3.\ \texttt{smt}\,(\Phi \Rightarrow (term = pval) \wedge (init = \texttt{const}_\tau\texttt{true}))\end{array}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \texttt{ptrValidForDeref}\,(\tau, pval, res\_term) \Rightarrow \Sigma\, y{:}\texttt{bool}.\ y = \texttt{aligned}\,(\tau, pval) \wedge (pval \overset{\texttt{const}_\tau\texttt{true}}{\mapsto_\tau} value) * \texttt{I}}$$

EXPL_IS_MEMOP_PTRWELLALIGNED

$$\frac{1.\ \mathcal{C} \vdash pval \Rightarrow \texttt{pointer}}{\mathcal{C};\mathcal{L};\Phi;\cdot \vdash \texttt{ptrWellAligned}\,(\tau, pval) \Rightarrow \Sigma\, y{:}\texttt{bool}.\ y = \texttt{aligned}\,(\tau, pval) \wedge \texttt{I}}$$

EXPL_IS_MEMOP_PTRARRAYSHIFT

$$\frac{\begin{array}{l}1.\ \mathcal{C} \vdash pval_1 \Rightarrow \texttt{pointer}\\ 2.\ \mathcal{C} \vdash pval_2 \Rightarrow \texttt{integer}\end{array}}{\mathcal{C};\mathcal{L};\Phi;\cdot \vdash \texttt{ptrArrayShift}\,(pval_1, \tau, pval_2) \Rightarrow \Sigma\, y{:}\texttt{pointer}.\ y = pval_1 +_{\text{ptr}} (pval_2 \times \text{size\_of}(\tau)) \wedge \texttt{I}}$$

$\boxed{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash is\_expr \Rightarrow ret}$ indet. seq. expression synthesises: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, is_expr synthesises return type ret

EXPL_IS_TVAL

$$\frac{1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash tval \Leftarrow ret}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash tval{:}ret \Rightarrow ret}$$

EXPL_IS_MEMOP

$$\frac{1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash memop \Rightarrow ret}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \texttt{memop}\,(memop) \Rightarrow ret}$$

EXPL_IS_ACTION

$$\frac{1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash action \Rightarrow ret}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash\ action \Rightarrow ret}$$

EXPL_IS_NEG_ACTION

$$\frac{1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash action \Rightarrow ret}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \texttt{neg}\ action \Rightarrow ret}$$

A3.5 Sequenced expressions

$\boxed{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{seq\_expr} \Rightarrow \mathit{ret}}$ seq. expression synthesises: given $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}$, *seq_expr* synthesises return type *ret*

EXPL_SEQ_CCALL

$$\frac{\begin{array}{l} 1.\ \mathit{ident}{:}\mathit{fun} \equiv \overline{x_i}^{\,i} \mapsto \mathit{texpr} \in \mathtt{Globals} \\ 2.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{\mathit{spine\_elem}_i}^{\,i} :: \mathit{fun} \gg \mathit{ret} \end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathtt{ccall}\,(\tau, \mathit{ident}, \overline{\mathit{spine\_elem}_i}^{\,i}) \Rightarrow \mathit{ret}}$$

EXPL_SEQ_PROC

$$\frac{\begin{array}{l} 1.\ \mathit{name}{:}\mathit{fun} \equiv \overline{x_i}^{\,i} \mapsto \mathit{texpr} \in \mathtt{Globals} \\ 2.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{\mathit{spine\_elem}_i}^{\,i} :: \mathit{fun} \gg \mathit{ret} \end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathtt{pcall}\,(\mathit{name}, \overline{\mathit{spine\_elem}_i}^{\,i}) \Rightarrow \mathit{ret}}$$

A3.6 Top-level Expressions

$\boxed{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{tval} \Leftarrow \mathit{ret}}$ top-level value checks: given $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}$, *tval* checks against return type *ret*

EXPL_TOP_VAL_DONE

$$\frac{1.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{ret\_terms} :: \mathtt{to\_fun}\,\mathit{ret} \gg \mathtt{I}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathtt{done}\,\langle \mathit{ret\_terms} \rangle \Leftarrow \mathit{ret}}$$

EXPL_TOP_VAL_UNDEF

$$\frac{1.\ \mathtt{smt}\,(\Phi \Rightarrow \mathtt{false})}{\mathcal{C}; \mathcal{L}; \Phi; \cdot \vdash \mathtt{undef}\ \ \mathit{UB\_name} \Leftarrow \mathit{ret}}$$

EXPL_TOP_VAL_ERROR

$$\frac{1.\ \mathtt{smt}\,(\Phi \Rightarrow \mathtt{false})}{\mathcal{C}; \mathcal{L}; \Phi; \cdot \vdash \mathtt{error}\,(\mathit{string}, \mathit{pval}) \Leftarrow \mathit{ret}}$$

$\boxed{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{seq\_texpr} \Leftarrow \mathit{ret}}$ top-level seq. expression checks: given $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}$, *seq_texpr* checks against return type *ret*

EXPL_TOP_SEQ_VAL

$$\frac{1.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{tval} \Leftarrow \mathit{ret}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{tval} \Leftarrow \mathit{ret}}$$

EXPL_TOP_SEQ_LETP

$$\frac{\begin{array}{l} 1.\ \mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{pexpr} \Rightarrow \Sigma\, y{:}\beta.\ \mathit{term} \wedge \mathtt{I} \\ 2.\ \mathit{ident\_or\_pat}{:}\beta \rightsquigarrow \mathcal{C}_1\ \mathtt{with}\ \mathit{term}_1 \\ 3.\ \mathcal{C}, \mathcal{C}_1; \mathcal{L}; \Phi, \mathit{term}_1/y(\mathit{term}); \mathcal{R} \vdash \mathit{texpr} \Leftarrow \mathit{ret} \end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathtt{let}\,\mathit{ident\_or\_pat} = \mathit{pexpr}\,\mathtt{in}\,\mathit{texpr} \Leftarrow \mathit{ret}}$$

Expl_Top_Seq_LetTP

$$\frac{\begin{array}{l}1.\ \mathcal{C};\mathcal{L};\Phi \vdash \mathit{tpexpr} \Leftarrow \mathit{pure\_ret}\\ 2.\ \mathit{ident\_or\_pat}{:}\beta \leadsto \mathcal{C}_1\ \texttt{with}\ \mathit{term}_1\\ 3.\ \mathcal{C},\mathcal{C}_1;\mathcal{L};\Phi,\mathit{term}_1/y(\mathit{term});\mathcal{R} \vdash \mathit{texpr} \Leftarrow \mathit{ret}\end{array}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \texttt{let}\ \mathit{ident\_or\_pat}{:}\mathit{pure\_ret} = \mathit{tpexpr}\ \texttt{in}\ \mathit{texpr} \Leftarrow \mathit{ret}}$$

Expl_Top_Seq_Let

$$\frac{\begin{array}{l}1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R}' \vdash \mathit{seq\_expr} \Rightarrow \mathit{ret}_1\\ 2.\ \Phi \vdash \mathit{ret\_pat}{:}\mathit{ret}_1 \leadsto \mathcal{C}_1;\mathcal{L}_1;\Phi_1;\mathcal{R}_1\\ 3.\ \mathcal{C},\mathcal{C}_1;\mathcal{L},\mathcal{L}_1;\Phi,\Phi_1;\mathcal{R},\mathcal{R}_1 \vdash \mathit{texpr} \Leftarrow \mathit{ret}_2\end{array}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R}',\mathcal{R} \vdash \texttt{let}\ \mathit{ret\_pat} = \mathit{seq\_expr}\ \texttt{in}\ \mathit{texpr} \Leftarrow \mathit{ret}_2}$$

Expl_Top_Seq_LetT

$$\frac{\begin{array}{l}1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R}' \vdash \mathit{texpr}_1 \Leftarrow \mathit{ret}_1\\ 2.\ \Phi \vdash \mathit{ret\_pat}{:}\mathit{ret}_1 \leadsto \mathcal{C}_1;\mathcal{L}_1;\Phi_1;\mathcal{R}_1\\ 3.\ \mathcal{C},\mathcal{C}_1;\mathcal{L},\mathcal{L}_1;\Phi,\Phi_1;\mathcal{R},\mathcal{R}_1 \vdash \mathit{texpr}_2 \Leftarrow \mathit{ret}_2\end{array}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R}',\mathcal{R} \vdash \texttt{let}\ \mathit{ret\_pat}{:}\mathit{ret}_1 = \mathit{texpr}_1\ \texttt{in}\ \mathit{texpr}_2 \Leftarrow \mathit{ret}_2}$$

Expl_Top_Seq_Case

$$\frac{\begin{array}{l}1.\ \mathcal{C} \vdash \mathit{pval} \Rightarrow \beta_1\\ 2.\ \overline{\mathit{pat}_i{:}\beta_1 \leadsto \mathcal{C}_i\ \texttt{with}\ \mathit{term}_i}^{\,i}\\ 3.\ \overline{\mathcal{C},\mathcal{C}_i;\mathcal{L};\Phi,\mathit{term}_i = \mathit{pval};\mathcal{R} \vdash \mathit{texpr}_i \Leftarrow \mathit{ret}}^{\,i}\end{array}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \texttt{case}\ \mathit{pval}\ \texttt{of}\ \overline{\mid \mathit{pat}_i \Rightarrow \mathit{texpr}_i}^{\,i}\ \texttt{end} \Leftarrow \mathit{ret}}$$

Expl_Top_Seq_If

$$\frac{\begin{array}{l}1.\ \mathcal{C} \vdash \mathit{pval} \Rightarrow \texttt{bool}\\ 2.\ \mathcal{C};\mathcal{L};\Phi,\mathit{pval} = \texttt{true};\mathcal{R} \vdash \mathit{texpr}_1 \Leftarrow \mathit{ret}\\ 3.\ \mathcal{C};\mathcal{L};\Phi,\mathit{pval} = \texttt{false};\mathcal{R} \vdash \mathit{texpr}_2 \Leftarrow \mathit{ret}\end{array}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \texttt{if}\ \mathit{pval}\ \texttt{then}\ \mathit{texpr}_1\ \texttt{else}\ \mathit{texpr}_2 \Leftarrow \mathit{ret}}$$

Expl_Top_Seq_Run

$$\frac{\begin{array}{l}1.\ \mathit{ident}{:}\mathit{fun} \equiv \overline{x_i}^{\,i} \mapsto \mathit{texpr} \in \texttt{Globals}\\ 2.\ \mathcal{C};\mathcal{L};\Phi;\cdot \vdash \overline{\mathit{pval}_i}^{\,i} :: \mathit{fun} \gg \texttt{false} \wedge \texttt{I}\end{array}}{\mathcal{C};\mathcal{L};\Phi;\cdot \vdash \texttt{run}\ \mathit{ident}\ \overline{\mathit{pval}_i}^{\,i} \Leftarrow \texttt{false} \wedge \texttt{I}}$$

Expl_Top_Seq_Bound

$$\frac{1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{is\_texpr} \Leftarrow \mathit{ret}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \texttt{bound}\,[\mathit{int}](\mathit{is\_texpr}) \Leftarrow \mathit{ret}}$$

$\boxed{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{is\_texpr} \Leftarrow \mathit{ret}}$ top-level indet. seq. expression checks: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, $\mathit{is\_texpr}$ checks against return type ret

Expl_Top_Is_LetS

$$\frac{\begin{array}{l}1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R}' \vdash \mathit{is\_expr} \Rightarrow \mathit{ret}_1 \\ 2.\ \Phi \vdash \mathit{ret\_pat}{:}\mathit{ret}_1 \leadsto \mathcal{C}_1;\mathcal{L}_1;\Phi_1;\mathcal{R}_1 \\ 3.\ \mathcal{C},\mathcal{C}_1;\mathcal{L},\mathcal{L}_1;\Phi,\Phi_1;\mathcal{R},\mathcal{R}_1 \vdash \mathit{texpr} \Leftarrow \mathit{ret}_2\end{array}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R}',\mathcal{R} \vdash \texttt{let}\,\texttt{strong}\,\mathit{ret\_pat} = \mathit{is\_expr}\,\texttt{in}\,\mathit{texpr} \Leftarrow \mathit{ret}_2}$$

$\boxed{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{texpr} \Leftarrow \mathit{ret}}$ top-level expression checks: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, *texpr* checks against return type *ret*

Expl_Top_Is

$$\frac{1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{is\_texpr} \Leftarrow \mathit{ret}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{is\_texpr} \Leftarrow \mathit{ret}}$$

Expl_Top_Seq

$$\frac{1.\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{seq\_texpr} \Leftarrow \mathit{ret}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{seq\_texpr} \Leftarrow \mathit{ret}}$$

A4 Elaboration System

$\boxed{\Phi \vdash \mathit{pred\_term} \in? \mathit{qpred\_term} \rightsquigarrow \mathit{opt\_term}}$ given constraints Φ, *pred_term* is potentially a part of *qpred_term* at index *opt_term*

PinQ_Name_Neq

$$\frac{1.\ \alpha_1 \neq \alpha_2}{\Phi \vdash \alpha_1(\_,\_) \in? (\_;\_)\{\alpha_2(\_+\_\times\_,\_)\} \rightsquigarrow \texttt{None}}$$

PinQ_IG_Or_IArg_Neq

$$\frac{\begin{array}{l} 1.\ \mathit{term} \equiv (\mathit{ptr}_2 - \mathit{ptr}_1)/\mathit{step} \\ 2.\ \mathit{term}_1 \equiv \mathit{term}/x(\mathit{iguard}) \\ 3.\ \mathit{term}_2 \equiv \bigwedge(\overline{\mathit{iarg}_{1\,i} = [\mathit{term}/x](\mathit{iarg}_{2\,i})}^{i}) \\ 4.\ \texttt{smt}\ (\Phi \Rightarrow \neg(\mathit{term}_1 \wedge \mathit{term}_2)) \end{array}}{\Phi \vdash \alpha(\mathit{ptr}_2, \overline{\mathit{iarg}_{1\,i}}^{i}) \in? (x; \mathit{iguard})\{\alpha(\mathit{ptr}_1 + x\times\mathit{step}, \overline{\mathit{iarg}_{2\,i}}^{i})\} \rightsquigarrow \texttt{None}}$$

PinQ_Comp

$$\frac{\begin{array}{l} 1.\ \mathit{term} \equiv (\mathit{ptr}_2 - \mathit{ptr}_1)/\mathit{step} \\ 2.\ \mathit{term}_1 \equiv \mathit{term}/x(\mathit{iguard}) \\ 3.\ \mathit{term}_2 \equiv \bigwedge(\overline{\mathit{iarg}_{1\,i} = [\mathit{term}/x](\mathit{iarg}_{2\,i})}^{i}) \\ 4.\ \texttt{smt}\ (\Phi \Rightarrow \mathit{term}_1 \wedge \mathit{term}_2) \end{array}}{\Phi \vdash \alpha(\mathit{ptr}_2, \overline{\mathit{iarg}_{1\,i}}^{i}) \in? (x; \mathit{iguard})\{\alpha(\mathit{ptr}_1 + x\times\mathit{step}, \overline{\mathit{iarg}_{2\,i}}^{i})\} \rightsquigarrow \mathit{term}}$$

$\boxed{\Phi \vdash \mathit{ident}{:}\underline{\mathit{res}} -? \mathit{res\_req} \rightsquigarrow \mathit{res\_diff}}$ the difference between $\mathit{ident}{:}\underline{\mathit{res}}$ and requested *res_req* is *res_diff*

Res_Diff_If_None

$$\frac{}{\Phi \vdash \_{:}\texttt{if}\ \mathit{term}\ \texttt{then}\ \mathit{res}_1\ \texttt{else}\ \mathit{res}_2 -? \mathit{res\_req} \rightsquigarrow \texttt{None}}$$

Res_Diff_PP_None

$$\frac{1.\ \Phi \vdash \mathit{pred\_term} \equiv \mathit{pred\_term}' \rightsquigarrow \texttt{false}}{\Phi \vdash \_{:}\mathit{pred\_term}'(\_) -? \mathit{pred\_term} \rightsquigarrow \texttt{None}}$$

$$\frac{1.\ \Phi \vdash \mathit{pred\_term} \equiv \mathit{pred\_term}' \rightsquigarrow \texttt{true}}{\Phi \vdash r{:}\mathit{pred\_term}'(\mathit{oarg}) \multimap? \mathit{pred\_term} \rightsquigarrow r \ \texttt{and}\ \mathit{oarg}}\ \textsc{Res\_Diff\_PP\_Exact}$$

$$\frac{1.\ \Phi \vdash \mathit{pred\_term} \in? \mathit{qpred\_term} \rightsquigarrow \texttt{None}}{\Phi \vdash \_{:}\mathit{qpred\_term}(\_) \multimap? \mathit{pred\_term} \rightsquigarrow \texttt{None}}\ \textsc{Res\_Diff\_PQ\_None}$$

$$\frac{\begin{array}{l}1.\ \Phi \vdash \mathit{pred\_term} \in? \mathit{qpred\_term} \rightsquigarrow \mathit{term}\\ 2.\ \mathit{qpred\_term} \equiv (x; \mathit{iguard})\{\alpha(\mathit{ptr}_1 + x \times \mathit{step}, \mathit{iargs})\}\\ 3.\ \mathit{rem} \equiv (x; \mathit{iguard} \wedge (x \neq \mathit{term}))\{\alpha(\mathit{ptr}_1 + x \times \mathit{step}, \mathit{iargs})\}(\mathit{oarg})\end{array}}{\Phi \vdash r{:}\mathit{qpred\_term}(\mathit{oarg}) \multimap? \mathit{pred\_term} \rightsquigarrow \texttt{bind}\ \langle r_1, r_2\rangle{:}\mathit{rem} * \mathit{pred\_term}(\mathit{oarg}[\mathit{term}]) = \texttt{break}\,(r, \mathit{term})\ \texttt{for}\ r_2\ \&\ \mathit{oarg}[\mathit{term}]\ \texttt{and}\ r_1{:}\mathit{rem}}\ \textsc{Res\_Diff\_PQ\_Rem}$$

$$\frac{1.\ \Phi \vdash \mathit{pred\_term} \in? \mathit{qpred\_term} \rightsquigarrow \texttt{None}}{\Phi \vdash \_{:}\mathit{pred\_term}(\_) \multimap? \mathit{qpred\_term} \rightsquigarrow \texttt{None}}\ \textsc{Res\_Diff\_QP\_None}$$

$$\frac{\begin{array}{l}1.\ \Phi \vdash \mathit{pred\_term} \in? \mathit{qpred\_term} \rightsquigarrow \mathit{term}\\ 2.\ \mathit{qpred\_term} \equiv (x; \mathit{iguard})\{\alpha(\mathit{ptr}_1 + x \times \mathit{step}, \mathit{iargs})\}\\ 3.\ \texttt{smt}\,(\Phi \Rightarrow \exists\, x.\ \mathit{iguard} \wedge (x \neq \mathit{term}))\end{array}}{\Phi \vdash r{:}\mathit{pred\_term}(\mathit{oarg}) \multimap? \mathit{qpred\_term} \rightsquigarrow \mathit{oarg}\ \texttt{and}\ (x; \mathit{iguard} \wedge (x \neq \mathit{term}))\{\alpha(\mathit{ptr}_1 + x \times \mathit{step}, \mathit{iargs})\}}\ \textsc{Res\_Diff\_QP\_More}$$

$$\frac{\begin{array}{l}1.\ \Phi \vdash \mathit{pred\_term} \in? \mathit{qpred\_term} \rightsquigarrow \mathit{term}\\ 2.\ \mathit{qpred\_term} \equiv (x; \mathit{iguard})\{\alpha(\mathit{ptr}_1 + x \times \mathit{step}, \mathit{iargs})\}\\ 3.\ \texttt{smt}\,(\Phi \Rightarrow \forall\, x.\ \neg\,(\mathit{iguard} \wedge (x \neq \mathit{term})))\\ 4.\ \mathcal{C}; \mathcal{L} \vdash \mathit{oarg} \Rightarrow \beta\end{array}}{\Phi \vdash r{:}\mathit{pred\_term}(\mathit{oarg}) \multimap? \mathit{qpred\_term} \rightsquigarrow \texttt{inj}\,(r, \mathit{ptr}_1, \mathit{step}, x.\ \mathit{iargs})\ \texttt{and}\ (\texttt{default array}\ \beta)[\mathit{term}] := \mathit{oarg}}\ \textsc{Res\_Diff\_QP\_Last}$$

Res_Diff_QQ_None

$$\frac{1.\ \Phi \vdash \mathit{qpred\_term} \sqsubseteq?\ \mathit{qpred\_term}' \rightsquigarrow \texttt{None}}{\Phi \vdash \_{:}\mathit{qpred\_term}'(\_) \multimap?\ \mathit{qpred\_term} \rightsquigarrow \texttt{None}}$$

Res_Diff_QQ_Eq

$$\frac{1.\ \Phi \vdash \mathit{qpred\_term}' \sqsubseteq?\ \mathit{qpred\_term} \rightsquigarrow \texttt{Eq}}{\Phi \vdash r{:}\mathit{qpred\_term}(\mathit{oarg}) \multimap?\ \mathit{qpred\_term}' \rightsquigarrow r\ \texttt{and}\ \mathit{oarg}}$$

Res_Diff_QQ_Lt

$$\frac{\begin{array}{l}1.\ \Phi \vdash \mathit{qpred\_term}' \sqsubseteq?\ \mathit{qpred\_term} \rightsquigarrow \texttt{Lt}\\ 2.\ \mathit{qpred\_term} \equiv (x; \mathit{iguard})\{\alpha(\mathit{ptr} + x \times \mathit{step}, \mathit{iargs})\}\\ 3.\ \mathit{qpred\_term}' \equiv (x; \mathit{iguard}')\{\alpha(\mathit{ptr} + x \times \mathit{step}, \mathit{iargs})\}\\ 4.\ \mathit{rem} \equiv (x; \mathit{iguard} \wedge \neg\, \mathit{iguard}')\{\alpha(\mathit{ptr} + x \times \mathit{step}, \mathit{iargs})\}(\mathit{oarg})\end{array}}{\Phi \vdash r{:}\mathit{qpred\_term}(\mathit{oarg}) \multimap?\ \mathit{qpred\_term}' \rightsquigarrow \texttt{bind}\ \langle r_1, r_2\rangle{:}\mathit{res} = \texttt{split}\,(r, \mathit{iguard}')\ \texttt{for}\ r_1\ \&\ \mathit{oarg}[k]\ \texttt{and}\ r_2{:}\mathit{rem}}$$

$\boxed{\Phi \vdash \mathit{ident}_1{:}\underline{\mathit{res}} +?\ \mathit{res\_term}_2{:}\mathit{res\_req}\ \&\ \mathit{oarg}_2 \rightsquigarrow \mathit{res\_term}\ \texttt{and}\ \mathit{oarg}_3}$ combining $\mathit{ident}_1{:}\underline{\mathit{res}}$, $\mathit{res\_term}_2{:}\mathit{res\_req}\&\mathit{oarg}_2$, results in $\mathit{res\_term}$ oarg_3

Res_Comb_PQ

$$\frac{\begin{array}{l}1.\ \Phi \vdash \mathit{pred\_term} \in?\ \mathit{qpred\_term} \rightsquigarrow \mathit{term}\\ 2.\ \mathcal{C}; \mathcal{L} \vdash \mathit{oarg}_1 \Rightarrow \texttt{record}\ \overline{\mathit{tag}_i{:}\beta_i}^{\,i}\\ 3.\ \mathcal{C}; \mathcal{L} \vdash \mathit{oarg}_2 \Rightarrow \texttt{array}\,\texttt{record}\ \overline{\mathit{tag}_i{:}\beta_i}^{\,i}\end{array}}{\Phi \vdash r{:}\mathit{pred\_term}(\mathit{oarg}_1) +?\ \mathit{res\_term}{:}\mathit{qpred\_term}\ \&\ \mathit{oarg}_2 \rightsquigarrow \texttt{glue}\,(\langle \mathit{res\_term}, r\rangle)\ \texttt{and}\ \mathit{oarg}_2[\mathit{term}] := \mathit{oarg}_1}$$

$\boxed{\Phi; \underline{\mathcal{R}} \vdash \texttt{wf}\ \mathit{res\_req} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \mathit{res\_term}\ \texttt{and}\ \mathit{oarg} \dashv \underline{\mathcal{R}'}}$ $\Phi; \underline{\mathcal{R}}$ fulfil well-formed request $\mathit{res\_req}$ (via $\mathit{res\_bind}$) for answer $\mathit{res\_term}$ and oarg, with $\underline{\mathcal{R}'}$ leftover

$$\textsc{Req\_Rej}$$

$$\dfrac{\begin{array}{l}1.\ \Phi \vdash r{:}\underline{res} \multimap ?\ res\_req \leadsto \texttt{None} \\ 2.\ \Phi; \underline{\mathcal{R}} \vdash \texttt{wf}\ res\_req \leadsto \texttt{bind}\ res\_bind\ \texttt{for}\ res\_term\ \texttt{and}\ oarg \dashv \underline{\mathcal{R}}\end{array}}{\Phi; \underline{\mathcal{R}}, r{:}\underline{res} \vdash \texttt{wf}\ res\_req \leadsto \texttt{bind}\ res\_bind\ \texttt{for}\ res\_term\ \texttt{and}\ oarg \dashv \underline{\mathcal{R}}, r{:}\underline{res}}$$

$$\textsc{Req\_Acc\_Clean}$$

$$\dfrac{1.\ \Phi \vdash r{:}\underline{res} \multimap ?\ res\_req \leadsto res\_term\ \texttt{and}\ oarg}{\Phi; \underline{\mathcal{R}}, r{:}\underline{res} \vdash \texttt{wf}\ res\_req \leadsto \texttt{bind}\ \cdot\ \texttt{for}\ res\_term\ \texttt{and}\ oarg \dashv \underline{\mathcal{R}}}$$

$$\textsc{Req\_Acc\_Rem}$$

$$\dfrac{1.\ \Phi \vdash r{:}\underline{res} \multimap ?\ res\_req \leadsto \texttt{bind}\ res\_pat_1{:}res_1 = res\_term_1\ \texttt{for}\ r_1\ \&\ oarg\ \texttt{and}\ r_2{:}rem}{\Phi; \underline{\mathcal{R}}, r{:}\underline{res} \vdash \texttt{wf}\ res\_req \leadsto \texttt{bind}\ res\_pat_1{:}res_1 = res\_term_1, \cdot\ \texttt{for}\ r_1\ \texttt{and}\ oarg \dashv \underline{\mathcal{R}}, r_2{:}rem}$$

$$\textsc{Req\_Acc\_More}$$

$$\dfrac{\begin{array}{l}1.\ \Phi \vdash r{:}\underline{res} \multimap ?\ res\_req_1 \leadsto oarg_1\ \texttt{and}\ res\_req_2 \\ 2.\ \Phi; \underline{\mathcal{R}} \vdash \texttt{wf}\ res\_req_2 \leadsto \texttt{bind}\ res\_bind_2\ \texttt{for}\ res\_term_2\ \texttt{and}\ oarg_2 \dashv \underline{\mathcal{R}_2} \\ 3.\ \Phi \vdash r{:}\underline{res} + ?\ res\_term_2{:}res\_req_2\ \&\ oarg_2 \leadsto res\_term_3\ \texttt{and}\ oarg_3\end{array}}{\Phi; \underline{\mathcal{R}}, r{:}\underline{res} \vdash \texttt{wf}\ res\_req_1 \leadsto \texttt{bind}\ res\_bind\ \texttt{for}\ res\_term_3\ \texttt{and}\ oarg_3 \dashv \underline{\mathcal{R}_2}}$$

$\boxed{\Phi; \underline{\mathcal{R}} \vdash res\_req \leadsto \texttt{bind}\ res\_bind\ \texttt{for}\ res\_term\ \texttt{and}\ oarg \dashv \underline{\mathcal{R}'}}$ $\Phi; \underline{\mathcal{R}}$ (check well-formedness of and then) fulfil request res_req (via res_bind) for answer res_term and $oarg$, with $\underline{\mathcal{R}'}$ leftover

REQ_WF_PRED

$$\frac{\begin{array}{l}1.\ \alpha \equiv \_{:}\texttt{pointer}, \overline{\_{i}{:}\beta_i}^{i}, y'{:}\texttt{record}\,\overline{tag_j{:}\beta'_j}^{j} \mapsto res \in \texttt{Globals}\\ 2.\ \mathcal{C};\mathcal{L} \vdash ptr \Rightarrow \texttt{pointer}\\ 3.\ \overline{\mathcal{C};\mathcal{L} \vdash iarg_i \Rightarrow \beta_i}^{i}\\ 4.\ \Phi;\underline{\mathcal{R}} \vdash \texttt{wf}\,\alpha(ptr, \overline{iarg_i}^{i}) \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ res\_term\ \texttt{and}\ oarg \dashv \underline{\mathcal{R}'}\end{array}}{\Phi;\underline{\mathcal{R}} \vdash \alpha(ptr, \overline{iarg_i}^{i}) \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ res\_term\ \texttt{and}\ oarg \dashv \underline{\mathcal{R}'}}$$

REQ_WF_QPRED

$$\frac{\begin{array}{l}1.\ \alpha \equiv \_{:}\texttt{pointer}, \overline{\_{i}{:}\beta_i}^{i}, y'{:}\texttt{record}\,\overline{tag_j{:}\beta'_j}^{j} \mapsto res \in \texttt{Globals}\\ 2.\ \mathcal{C};\mathcal{L} \vdash ptr \Rightarrow \texttt{pointer}\\ 3.\ \overline{\mathcal{C};\mathcal{L} \vdash iarg_i \Rightarrow \beta_i}^{i}\\ 4.\ \Phi;\underline{\mathcal{R}} \vdash \texttt{wf}\,(x; iguard)\{\alpha(ptr + x \times step, \overline{iarg_i}^{i})\} \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ res\_term\ \texttt{and}\ oarg \dashv \underline{\mathcal{R}'}\end{array}}{\Phi;\underline{\mathcal{R}} \vdash (x; iguard)\{\alpha(ptr + x \times step, \overline{iarg_i}^{i})\} \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ res\_term\ \texttt{and}\ oarg \dashv \underline{\mathcal{R}'}}$$

$\boxed{\Phi;\underline{\mathcal{R}} \vdash \texttt{if}\ term\ \texttt{then}\ res_1\ \texttt{else}\ res_2 \rightsquigarrow ident \dashv \underline{\mathcal{R}'}}$ under-determined conditional resource request: $\Phi;\underline{\mathcal{R}}$ fulfil request for `if` $term$ `then` res_1 `else` res_2 with **synthesising** $ident$ and $\underline{\mathcal{R}'}$ leftover

IF_ACC

$$\frac{1.\ \Phi \vdash \underline{res} \equiv \texttt{if}\ term\ \texttt{then}\ res_1\ \texttt{else}\ res_2}{\Phi;\underline{\mathcal{R}}, x{:}\underline{res} \vdash \texttt{if}\ term\ \texttt{then}\ res_1\ \texttt{else}\ res_2 \rightsquigarrow x \dashv \underline{\mathcal{R}}}$$

IF_REJ

$$\frac{1.\ \Phi;\underline{\mathcal{R}} \vdash \texttt{if}\ term\ \texttt{then}\ res_1\ \texttt{else}\ res_2 \rightsquigarrow x \dashv \underline{\mathcal{R}}}{\Phi;\underline{\mathcal{R}}, x{:}\underline{res} \vdash \texttt{if}\ term\ \texttt{then}\ res_1\ \texttt{else}\ res_2 \rightsquigarrow x \dashv \underline{\mathcal{R}}, x{:}\underline{res}}$$

$\boxed{\Phi;\underline{\mathcal{R}} \vdash \texttt{calc}\ y\ \texttt{using}\ res \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ res\_term\ \texttt{and}\ oarg \dashv \underline{\mathcal{R}'}}$ arbirtrary resource and output-arg request: $\Phi;\underline{\mathcal{R}}$ fulfil request for resource res and output-arg y (via res_bind) with **checking** res_term and $oarg$, leaving resources $\underline{\mathcal{R}'}$

Oarg_Empty

$$\frac{}{\Phi; \underline{\mathcal{R}} \vdash \texttt{calc}\ \_\ \texttt{using}\ \texttt{emp} \rightsquigarrow \texttt{bind}\ \cdot\ \texttt{for}\ \texttt{emp}\ \texttt{and}\ \texttt{unit} \dashv \underline{\mathcal{R}}}$$

Oarg_Return

$$\frac{}{\Phi; \underline{\mathcal{R}} \vdash \texttt{calc}\ y\ \texttt{using}\ \bigwedge(\overline{y.x_i = \mathit{term}_i}^{\,i}) \rightsquigarrow \texttt{bind}\ \cdot\ \texttt{for}\ \texttt{term}\ \texttt{and}\ \{\overline{x_i = \mathit{term}_i}^{\,i}\} \dashv \underline{\mathcal{R}}}$$

Oarg_EndIf_True

$$\frac{\begin{array}{l}1.\ \texttt{smt}\,(\Phi \Rightarrow \mathit{term})\\ 2.\ \Phi; \underline{\mathcal{R}} \vdash \texttt{calc}\ y\ \texttt{using}\ \mathit{res}_1 \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \mathit{res\_term}\ \texttt{and}\ \mathit{oarg} \dashv \underline{\mathcal{R}'}\end{array}}{\Phi; \underline{\mathcal{R}} \vdash \texttt{calc}\ y\ \texttt{using}\ \texttt{if}\ \mathit{term}\ \texttt{then}\ \mathit{res}_1\ \texttt{else}\ \mathit{res}_2 \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \mathit{res\_term}\ \texttt{and}\ \mathit{oarg} \dashv \underline{\mathcal{R}'}}$$

Oarg_EndIf_False

$$\frac{\begin{array}{l}1.\ \texttt{smt}\,(\Phi \Rightarrow \neg\,\mathit{term})\\ 2.\ \Phi; \underline{\mathcal{R}} \vdash \texttt{calc}\ y\ \texttt{using}\ \mathit{res}_2 \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \mathit{res\_term}\ \texttt{and}\ \mathit{oarg} \dashv \underline{\mathcal{R}'}\end{array}}{\Phi; \underline{\mathcal{R}} \vdash \texttt{calc}\ y\ \texttt{using}\ \texttt{if}\ \mathit{term}\ \texttt{then}\ \mathit{res}_1\ \texttt{else}\ \mathit{res}_2 \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \mathit{res\_term}\ \texttt{and}\ \mathit{oarg} \dashv \underline{\mathcal{R}'}}$$

Oarg_EndIf_UnderDet

$$\frac{1.\ \Phi; \underline{\mathcal{R}} \vdash \texttt{if}\ \mathit{term}\ \texttt{then}\ \mathit{res}_1\ \texttt{else}\ \mathit{res}_2 \rightsquigarrow x \dashv \underline{\mathcal{R}'}}{\Phi; \underline{\mathcal{R}} \vdash \texttt{calc}\ \_\ \texttt{using}\ \texttt{if}\ \mathit{term}\ \texttt{then}\ \mathit{res}_1\ \texttt{else}\ \mathit{res}_2 \rightsquigarrow \texttt{bind}\ \cdot\ \texttt{for}\ x\ \texttt{and}\ \texttt{unit} \dashv \underline{\mathcal{R}'}}$$

OARG_MIDDLEIF

1. $\Phi; \underline{\mathcal{R}} \vdash$ `calc_using if` term `then` res_1 `else` $\mathit{res}_2 \rightsquigarrow$ `bind` $\mathit{res\_bind}$ `for` $\mathit{res\_term}$ `and unit` $\dashv \underline{\mathcal{R}'}$
2. $\Phi; \underline{\mathcal{R}'} \vdash$ `calc` y `using` $\mathit{res}_3 \rightsquigarrow$ `bind` $\mathit{res\_bind}_3$ `for` $\mathit{res\_term}_3$ `and` $\mathit{oarg} \dashv \underline{\mathcal{R}''}$

$\Phi; \underline{\mathcal{R}} \vdash$ `calc` y `using (if` term `then` res_1 `else` $\mathit{res}_2) * \mathit{res}_3 \rightsquigarrow$ `bind` $\mathit{res\_bind}, \mathit{res\_bind}_3$ `for` $\langle \mathit{res\_term}, \mathit{res\_term}_3 \rangle$ `and` $\mathit{oarg} \dashv \underline{\mathcal{R}''}$

OARG_ASSERT

1. `smt` $(\Phi \Rightarrow \mathit{term})$
2. $\Phi; \underline{\mathcal{R}} \vdash$ `calc` y `using` $\mathit{res} \rightsquigarrow$ `bind` $\mathit{res\_bind}$ `for` $\mathit{res\_term}$ `and` $\mathit{oarg} \dashv \underline{\mathcal{R}'}$

$\Phi; \underline{\mathcal{R}} \vdash$ `calc` y `using` $\mathit{term} * \mathit{res} \rightsquigarrow$ `bind` $\mathit{res\_bind}$ `for` $\langle \mathtt{term}, \mathit{res\_term} \rangle$ `and` $\mathit{oarg} \dashv \underline{\mathcal{R}'}$

OARG_LETPRED

1. $\Phi; \underline{\mathcal{R}} \vdash \mathit{pred\_term} \rightsquigarrow$ `bind` $\mathit{res\_bind}_1$ `for` $\mathit{res\_term}_1$ `and` $\mathit{oarg}' \dashv \underline{\mathcal{R}'}$
2. $\Phi; \underline{\mathcal{R}'} \vdash$ `calc` y `using` $\mathit{oarg}'/y'(\mathit{res}) \rightsquigarrow$ `bind` $\mathit{res\_bind}_2$ `for` $\mathit{res\_term}_2$ `and` $\mathit{oarg} \dashv \underline{\mathcal{R}''}$
3. $\mathit{res\_term} \equiv$ `pack` $(\mathit{oarg}', \langle \mathit{res\_term}_1, \mathit{res\_term}_2 \rangle)$

$\Phi; \underline{\mathcal{R}} \vdash$ `calc` y `using` $\exists\, y'$:`record` $\overline{\mathit{tag}_j{:}\beta'_j}^{\,j}.\ \mathit{pred\_term}(\,y'\,) * \mathit{res} \rightsquigarrow$ `bind` $\mathit{res\_bind}_1, \mathit{res\_bind}_2$ `for` $\mathit{res\_term}$ `and` $\mathit{oarg} \dashv \underline{\mathcal{R}''}$

OARG_LETQPRED

1. $\Phi; \underline{\mathcal{R}} \vdash \mathit{qpred\_term} \rightsquigarrow$ `bind` $\mathit{res\_bind}_1$ `for` $\mathit{res\_term}_1$ `and` $\mathit{oarg}' \dashv \underline{\mathcal{R}'}$
2. $\Phi; \underline{\mathcal{R}'} \vdash$ `calc` y `using` $\mathit{oarg}'/y'(\mathit{res}) \rightsquigarrow$ `bind` $\mathit{res\_bind}_2$ `for` $\mathit{res\_term}_2$ `and` $\mathit{oarg} \dashv \underline{\mathcal{R}''}$
3. $\mathit{res\_term} \equiv$ `pack` $(\mathit{oarg}', \langle \mathit{res\_term}_1, \mathit{res\_term}_2 \rangle)$

$\Phi; \underline{\mathcal{R}} \vdash$ `calc` y `using` $\exists\, y'$:`array record` $\overline{\mathit{tag}_j{:}\beta'_j}^{\,j}.\ \mathit{qpred\_term}(\,y'\,) * \mathit{res} \rightsquigarrow$ `bind` $\mathit{res\_bind}_1, \mathit{res\_bind}_2$ `for` $\mathit{res\_term}$ `and` $\mathit{oarg} \dashv \underline{\mathcal{R}''}$

$\boxed{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \mathit{action} \rightsquigarrow \mathtt{bind}\ \mathit{res\_bind}\ \mathtt{for}\ \mathit{action}'{:}\mathit{norm\_ret} \dashv \underline{\mathcal{R}'}}$ memory action elaboration: given $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}}$, action elaborates (via $\mathit{res\_bind}$) to $\mathit{action}'{:}\mathit{norm\_ret}$, with $\underline{\mathcal{R}'}$ leftover

Elab_Is_Action_Create

1. $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \texttt{create}\,(pval, \tau) \Rightarrow \underline{ret}$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \texttt{create}\,(pval, \tau) \rightsquigarrow \texttt{bind}\ \cdot\ \texttt{for}\ \texttt{create}\,(pval, \tau) : \underline{ret} \dashv \underline{\mathcal{R}}}$$

Elab_Is_Action_Load

1. $\mathcal{C} \vdash pval_0 \Rightarrow \texttt{pointer}$
2. $\Phi; \underline{\mathcal{R}} \vdash \texttt{Owned}\,\langle\tau\rangle(pval_0) \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ res\_term\ \texttt{and}\ oarg \dashv \underline{\mathcal{R}'}$
3. $\texttt{smt}\,(\Phi \Rightarrow oarg.init = \texttt{const}_\tau\texttt{true})$
4. $\underline{ret} \equiv \Sigma\, y{:}\beta_\tau.\ y = oarg.value \wedge pt * \texttt{I}$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \texttt{load}\,(\tau, pval_0, \_, \_) \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ \texttt{load}\,(\tau, pval_0, \_, res\_term) : \underline{ret} \dashv \underline{\mathcal{R}'}}$$

Elab_Is_Action_Store

1. $\mathcal{C} \vdash pval_0 \Rightarrow \texttt{pointer}$
2. $\mathcal{C} \vdash pval_1 \Rightarrow \beta_\tau$
3. $\texttt{smt}\,(\Phi \Rightarrow \texttt{representable}\,(\tau, pval_1))$
4. $\Phi; \underline{\mathcal{R}} \vdash \texttt{Owned}\,\langle\tau\rangle(pval_0) \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ res\_term\ \texttt{and}\ \_ \dashv \underline{\mathcal{R}'}$
5. $\underline{ret} \equiv \Sigma\, \_{:}\texttt{unit}.\ (pval_0 \overset{\texttt{const}_\tau\texttt{true}}{\mapsto_\tau} pval_1) * \texttt{I}$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \texttt{store}\,(\_, \tau, pval_0, pval_1, \_, \_) \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ \texttt{store}\,(\_, \tau, pval_0, pval_1, \_, res\_term) : \underline{ret} \dashv \underline{\mathcal{R}'}}$$

Elab_Is_Action_Kill_Static

1. $\mathcal{C} \vdash pval_0 \Rightarrow \texttt{pointer}$
2. $\Phi; \underline{\mathcal{R}} \vdash \texttt{Owned}\,\langle\tau\rangle(pval) \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ res\_term\ \texttt{and}\ \_ \dashv \underline{\mathcal{R}'}$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \texttt{kill}\,(\texttt{static}\,\tau, pval, \_) \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ \texttt{kill}\,(\texttt{static}\,\tau, pval, res\_term) : \Sigma\, \_{:}\texttt{unit}.\ \texttt{I} \dashv \underline{\mathcal{R}'}}$$

$\boxed{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash memop \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ memop' : norm\_ret \dashv \underline{\mathcal{R}'}}$ memory operation elaboration: given $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}}$, $memop$ elaborates to (via res_bind) to $memop' : norm\_ret$, with $\underline{\mathcal{R}'}$ leftover

Elab_Is_Memop_PtrValidForDeref

1. $\mathcal{C} \vdash pval_0 \Rightarrow \texttt{pointer}$
2. $\Phi; \underline{\mathcal{R}} \vdash \texttt{Owned}\,\langle\tau\rangle(pval_0) \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ res\_term\ \texttt{and}\ oarg \dashv \underline{\mathcal{R}'}$
3. $\texttt{smt}\,(\Phi \Rightarrow oarg.init = \texttt{const}_\tau\texttt{true})$
4. $\underline{ret} \equiv \Sigma\, y{:}\texttt{bool}.\ y = \texttt{aligned}\,(\tau, pval_0) \wedge pt' * \texttt{I}$

$$\overline{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \texttt{ptrValidForDeref}\,(\tau, pval_0, \_) \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ \texttt{ptrValidForDeref}\,(\tau, pval_0, res\_term){:}\underline{ret} \dashv \underline{\mathcal{R}'}}$$

Elab_Is_Memop_Rest

1. $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash memop \Rightarrow \underline{ret}$

$$\overline{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash memop \rightsquigarrow \texttt{bind}\ \cdot\ \texttt{for}\ memop{:}\underline{ret} \dashv \underline{\mathcal{R}}}$$

$\boxed{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash is\_expr \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ (is\_expr'){:}ret \dashv \underline{\mathcal{R}'}}$ indet. seq. expression elaboration: given $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}}$, is_expr elaborates (via res_bind) to $is\_expr'{:}ret$, with $\underline{\mathcal{R}'}$ leftover

Elab_Is_Memop

1. $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash memop \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ memop'{:}\underline{ret} \dashv \underline{\mathcal{R}'}$

$$\overline{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \texttt{memop}\,(memop) \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ (\texttt{memop}\,(memop')){:}\underline{ret} \dashv \underline{\mathcal{R}'}}$$

Elab_Is_Action

1. $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash action \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ action'{:}\underline{ret} \dashv \underline{\mathcal{R}'}$

$$\overline{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash\ action \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ (\ action'){:}\underline{ret} \dashv \underline{\mathcal{R}'}}$$

Elab_Is_Neg_Action

1. $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash action \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ action'{:}\underline{ret} \dashv \underline{\mathcal{R}'}$

$$\overline{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \texttt{neg}\ action \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ (\texttt{neg}\ action'){:}\underline{ret} \dashv \underline{\mathcal{R}'}}$$

ELAB_IS_PACK

$$\frac{\begin{array}{l}1.\ \alpha \equiv x_p{:}\texttt{pointer}, \overline{x_i{:}\beta_i}^{\,i}, y{:}\texttt{record}\,\overline{tag_j{:}\beta'_j}^{\,j} \mapsto res \in \texttt{Globals}\\ 2.\ \mathcal{C} \vdash pval \Rightarrow \texttt{pointer}\\ 3.\ \overline{\mathcal{C} \vdash pval_i \Rightarrow \beta_i}^{\,i}\\ 4.\ \Phi;\underline{\mathcal{R}} \vdash \texttt{calc}\,y\,\texttt{using}\,[[\overline{pval_i/x_i}^{\,i}], pval/x_p](res) \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ res\_term\ \texttt{and}\ oarg \dashv \underline{\mathcal{R}'}\\ 5.\ ret \equiv \Sigma\,\_{:}\texttt{unit}.\ \alpha(pval, \overline{pval_i}^{\,i})(oarg) * \texttt{I}\end{array}}{\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash \texttt{pack}\,\alpha(pval, \overline{pval_i}^{\,i}) \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\,(\texttt{done}\,\langle\texttt{Unit}, \texttt{fold}\,res\_term{:}\alpha(pval, \overline{pval_i}^{\,i})(oarg)\rangle{:}ret){:}\,ret \dashv \underline{\mathcal{R}'}}$$

ELAB_IS_UNPACK

$$\frac{\begin{array}{l}1.\ \alpha \equiv x_p{:}\texttt{pointer}, \overline{x_i{:}\beta_i}^{\,i}, y{:}\texttt{record}\,\overline{tag_j{:}\beta'_j}^{\,j} \mapsto res \in \texttt{Globals}\\ 2.\ \mathcal{C} \vdash pval \Rightarrow \texttt{pointer}\\ 3.\ \overline{\mathcal{C} \vdash pval_i \Rightarrow \beta_i}^{\,i}\\ 4.\ \Phi;\underline{\mathcal{R}} \vdash \alpha(pval, \overline{pval_i}^{\,i}) \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ res\_term\ \texttt{and}\ oarg \dashv \underline{\mathcal{R}'}\\ 5.\ res' \equiv [oarg/y, [\overline{iarg_i/x_i}^{\,i}], ptr/x_p](res)\\ 6.\ ret \equiv \Sigma\,\_{:}\texttt{unit}.\ res' * \texttt{I}\end{array}}{\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash \texttt{unpack}\,\alpha(pval, \overline{pval_i}^{\,i}) \rightsquigarrow \texttt{bind}\ \texttt{fold}\,(x){:}\alpha(pval, \overline{pval_i}^{\,i})(oarg) = res\_term\ \texttt{for}\,(\texttt{done}\,\langle\texttt{Unit}, x\rangle{:}ret){:}\,ret \dashv \underline{\mathcal{R}'}}$$

$\boxed{\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash spine :: \underline{fun} \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ spine'\ \texttt{and}\ norm\_ret \dashv \underline{\mathcal{R}'}}$ spine elaboration: given $\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}}$, arguments *spine* and function type $\underline{fun}$ elaborate (via res_bind) to $spine'$ and result type $norm\_ret$, with $\underline{\mathcal{R}'}$ leftover

ELAB_SPINE_EMPTY

$$\frac{}{\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash\ ::\underline{ret} \rightsquigarrow \texttt{bind}\ \cdot\ \texttt{for}\ \ \texttt{and}\ \underline{ret} \dashv \underline{\mathcal{R}}}$$

ELAB_SPINE_COMP

$$\frac{\begin{array}{l}1.\ \mathcal{C} \vdash pval \Rightarrow \beta\\ 2.\ \mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash spine :: pval/x(\underline{fun}) \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ spine'\ \texttt{and}\ \underline{ret} \dashv \underline{\mathcal{R}'}\end{array}}{\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash pval, spine :: \Pi\,x{:}\beta.\ \underline{fun} \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ pval, spine'\ \texttt{and}\ \underline{ret} \dashv \underline{\mathcal{R}'}}$$

Elab_Spine_LetPred

$$\frac{\begin{array}{l}1.\ \Phi; \underline{\mathcal{R}} \vdash \mathit{pred\_term} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \mathit{res\_term}\ \texttt{and}\ \mathit{oarg} \dashv \underline{\mathcal{R}'}\\ 2.\ \mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}'} \vdash \mathit{spine} :: \underline{\mathit{fun}} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}'\ \texttt{for}\ \mathit{spine}'\ \texttt{and}\ \underline{\mathit{ret}} \dashv \underline{\mathcal{R}''}\end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \mathit{spine} :: \forall\, y{:}\texttt{record}\,\overline{\mathit{tag}_j{:}\beta_j}^{\,j}.\ \mathit{pred\_term}(\,y\,) \twoheadrightarrow \underline{\mathit{fun}} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}, \mathit{res\_bind}'\ \texttt{for}\ \mathit{oarg}, \mathit{res\_term}, \mathit{spine}'\ \texttt{and}\ \underline{\mathit{ret}} \dashv \underline{\mathcal{R}''}}$$

Elab_Spine_LetQPred

$$\frac{\begin{array}{l}1.\ \Phi; \underline{\mathcal{R}} \vdash \mathit{qpred\_term} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \mathit{res\_term}\ \texttt{and}\ \mathit{oarg} \dashv \underline{\mathcal{R}'}\\ 2.\ \mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}'} \vdash \mathit{spine} :: \mathit{oarg}/y(\underline{\mathit{fun}}) \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}'\ \texttt{for}\ \mathit{spine}'\ \texttt{and}\ \underline{\mathit{ret}'} \dashv \underline{\mathcal{R}''}\end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \mathit{spine} :: \forall\, y{:}\texttt{array}\,\texttt{record}\,\overline{\mathit{tag}_j{:}\beta_j}^{\,j}.\ \mathit{qpred\_term}(\,y\,) \twoheadrightarrow \underline{\mathit{fun}} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}, \mathit{res\_bind}'\ \texttt{for}\ \mathit{oarg}, \mathit{res\_term}, \mathit{spine}'\ \texttt{and}\ \underline{\mathit{ret}} \dashv \underline{\mathcal{R}''}}$$

Elab_Spine_MiddleIf

$$\frac{\begin{array}{l}1.\ \Phi; \underline{\mathcal{R}} \vdash \texttt{calc\_using}\,\texttt{if}\ \mathit{term}\ \texttt{then}\ \mathit{res}_1\ \texttt{else}\ \mathit{res}_2 \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \mathit{res\_term}\ \texttt{and}\ \texttt{unit} \dashv \underline{\mathcal{R}'}\\ 2.\ \mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}'} \vdash \mathit{spine} :: \underline{\mathit{fun}} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}'\ \texttt{for}\ \mathit{spine}'\ \texttt{and}\ \underline{\mathit{ret}} \dashv \underline{\mathcal{R}''}\end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \mathit{spine} :: \texttt{if}\ \mathit{term}\ \texttt{then}\ \mathit{res}_1\ \texttt{else}\ \mathit{res}_2 \twoheadrightarrow \underline{\mathit{fun}} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}, \mathit{res\_bind}'\ \texttt{for}\ \mathit{res\_term}, \mathit{spine}'\ \texttt{and}\ \underline{\mathit{ret}} \dashv \underline{\mathcal{R}''}}$$

Elab_Spine_Phi

$$\frac{\begin{array}{l}1.\ \texttt{smt}\,(\Phi \Rightarrow \mathit{term})\\ 2.\ \mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \mathit{spine} :: \underline{\mathit{fun}} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \mathit{spine}'\ \texttt{and}\ \underline{\mathit{ret}} \dashv \underline{\mathcal{R}'}\end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \mathit{spine} :: \mathit{term} \supset \underline{\mathit{fun}} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \mathit{spine}'\ \texttt{and}\ \underline{\mathit{ret}} \dashv \underline{\mathcal{R}'}}$$

$\boxed{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \mathit{seq\_expr} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \mathit{seq\_expr}'{:}\mathit{norm\_ret} \dashv \underline{\mathcal{R}'}}$ seq. expression elaboration: given $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}}$, $\mathit{seq\_expr}$ elaborates (via $\mathit{res\_bind}$) to $\mathit{seq\_expr}'{:}\mathit{norm\_ret}$, with $\underline{\mathcal{R}'}$ leftover

ELAB_SEQ_CCALL

$$\frac{\begin{array}{l}1.\ ident{:}\underline{fun}\ \equiv\ \overline{x_i}^{\,i} \mapsto texpr\ \in \texttt{Globals}\\ 2.\ \mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash spine :: \underline{fun} \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ \overline{spine\_elem_i}^{\,i}\ \texttt{and}\ \underline{ret} \dashv \underline{\mathcal{R}'}\end{array}}{\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash \texttt{ccall}\,(\tau, ident, spine) \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ \texttt{ccall}\,(\tau, ident, \overline{spine\_elem_i}^{\,i}){:}\underline{ret} \dashv \underline{\mathcal{R}'}}$$

ELAB_SEQ_PROC

$$\frac{\begin{array}{l}1.\ name{:}\underline{fun}\ \equiv\ \overline{x_i}^{\,i} \mapsto texpr\ \in \texttt{Globals}\\ 2.\ \mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash spine :: \underline{fun} \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ \overline{spine\_elem_i}^{\,i}\ \texttt{and}\ \underline{ret} \dashv \underline{\mathcal{R}'}\end{array}}{\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash \texttt{pcall}\,(name, spine) \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ \texttt{pcall}\,(name, \overline{spine\_elem_i}^{\,i}){:}\underline{ret} \dashv \underline{\mathcal{R}'}}$$

$\boxed{\Phi \vdash res \rightsquigarrow res\_pat}$ resource normalisation by pat-matching: under constraints Φ , res will produces a normalised resourced context if it matches against res_pat

ELAB_RES_PAT_EMPTY

$$\frac{}{\Phi \vdash \texttt{emp} \rightsquigarrow \texttt{emp}}$$

ELAB_RES_PAT_PHI

$$\frac{}{\Phi \vdash term \rightsquigarrow \texttt{term}}$$

ELAB_RES_PAT_IF_TRUE

$$\frac{\begin{array}{l}1.\ \texttt{smt}\,(\Phi \Rightarrow term)\\ 2.\ \Phi \vdash res_1 \rightsquigarrow res\_pat\end{array}}{\Phi \vdash \texttt{if}\ term\ \texttt{then}\ res_1\ \texttt{else}\ res_2 \rightsquigarrow res\_pat}$$

ELAB_RES_PAT_IF_FALSE

$$\frac{\begin{array}{l}1.\ \texttt{smt}\,(\Phi \Rightarrow \neg\, term)\\ 2.\ \Phi \vdash res_2 \rightsquigarrow res\_pat\end{array}}{\Phi \vdash \texttt{if}\ term\ \texttt{then}\ res_1\ \texttt{else}\ res_2 \rightsquigarrow res\_pat}$$

ELAB_RES_PAT_VAR

$$\frac{}{\Phi \vdash \underline{res} \rightsquigarrow r}$$

ELAB_RES_PAT_SEPCONJ

$$\frac{\begin{array}{l}1.\ \Phi \vdash res_1 \rightsquigarrow res\_pat_1\\ 2.\ \Phi \vdash res_2 \rightsquigarrow res\_pat_2\end{array}}{\Phi \vdash res_1 * res_2 \rightsquigarrow \langle res\_pat_1, res\_pat_2\rangle}$$

Elab_Res_Pat_Pack

$$\frac{1.\ \Phi \vdash x/y(\mathit{res}) \leadsto \mathit{res\_pat}}{\Phi \vdash \exists\, y{:}\beta.\ \mathit{res} \leadsto \mathtt{pack}\,(x, \mathit{res\_pat})}$$

$\boxed{\Phi \vdash \mathit{ret} \leadsto \mathit{ret\_pat}}$ return-value normalisation by pattern-matching: under constraints Φ , ret will produce a normalised resourced context if it matches against $\mathit{ret\_pat}$

Elab_Ret_Pat_I

$$\frac{}{\Phi \vdash \mathtt{I} \leadsto }$$

Elab_Ret_Pat_Res

$$\frac{\begin{array}{l}1.\ \Phi \vdash \mathit{res} \leadsto \mathit{res\_pat}\\ 2.\ \Phi \vdash \mathit{ret} \leadsto \mathit{ret\_pat}\end{array}}{\Phi \vdash \mathit{res} * \mathit{ret} \leadsto \mathtt{res}\,\mathit{res\_pat}, \mathit{ret\_pat}}$$

Elab_Ret_Pat_Log

$$\frac{1.\ \Phi \vdash \mathit{ret} \leadsto \mathit{ret\_pat}}{\Phi \vdash \exists\, x{:}\beta.\ \mathit{ret} \leadsto \mathtt{log}\,x, \mathit{ret\_pat}}$$

$\boxed{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \mathit{is\_texpr} \Leftarrow \underline{\mathit{ret}} \leadsto \mathit{texpr}}$ top-level indet. seq. expression elaboration: given $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}}$, $\mathit{is\_texpr}$ elaborates to texpr

Elab_Top_Is_LetS

$$\frac{\begin{array}{l}1.\ \mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \mathit{is\_expr} \leadsto \mathtt{bind}\ \mathit{res\_bind}\ \mathtt{for}\,(\mathit{is\_expr}'){:}\Sigma\, y{:}\beta.\ \mathit{ret} \dashv \underline{\mathcal{R}}'\\ 2.\ \Phi \vdash \mathit{ret} \leadsto \mathit{ret\_pat}\\ 3.\ \Phi \vdash \mathtt{comp}\,\mathit{ident\_or\_pat}, \mathit{ret\_pat}{:}\Sigma\, y{:}\beta.\ \mathit{ret} \leadsto \mathcal{C}_1; \mathcal{L}_1; \Phi_1; \underline{\mathcal{R}}_1\\ 4.\ \mathcal{C}, \mathcal{C}_1; \mathcal{L}, \mathcal{L}', \mathcal{L}_1; \Phi, \Phi', \Phi_1; \underline{\mathcal{R}}', \underline{\mathcal{R}}_1 \vdash \mathit{texpr} \Leftarrow \underline{\mathit{ret}_2} \leadsto \mathit{texpr}'\\ 5.\ \mathit{texpr}'' \equiv \mathtt{insert\_lets}\,(\mathit{res\_bind}, \mathtt{let\ strong\ comp}\,\mathit{ident\_or\_pat}, \mathit{ret\_pat} = \mathit{is\_expr}'\ \mathtt{in}\ \mathit{texpr}')\end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \mathtt{let\ strong\ comp}\,\mathit{ident\_or\_pat} = \mathit{is\_expr}\ \mathtt{in}\ \mathit{texpr} \Leftarrow \underline{\mathit{ret}_2} \leadsto \mathit{texpr}''}$$

$\boxed{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \mathit{tval} \Leftarrow \underline{\mathit{ret}} \leadsto \mathtt{bind}\ \mathit{res\_bind}\ \mathtt{for}\ \mathit{tval}' \dashv \underline{\mathcal{R}}'}$ top-level value elaboration: given $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}}$, tval elaborates (via $\mathit{res\_bind}$) to tval' with $\underline{\mathcal{R}}'$ leftover

Elab_Top_Val_Done

$$\frac{1.\ \mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash \mathit{ret\_terms} :: \texttt{to\_fun}\,\underline{\mathit{ret}} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \mathit{ret\_terms}'\ \texttt{and}\ \texttt{I} \dashv \underline{\mathcal{R}}'}{\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash \texttt{done}\,\langle \mathit{ret\_terms}\rangle \Leftarrow \underline{\mathit{ret}} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \texttt{done}\,\langle \mathit{ret\_terms}'\rangle \dashv \underline{\mathcal{R}}'}$$

Elab_Top_Val_Undef

$$\frac{1.\ \texttt{smt}\,(\Phi \Rightarrow \texttt{false})}{\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash \texttt{undef}\ \mathit{UB\_name} \Leftarrow \underline{\mathit{ret}} \rightsquigarrow \texttt{bind}\ \cdot\ \texttt{for}\ \texttt{undef}\ \mathit{UB\_name} \dashv \underline{\mathcal{R}}}$$

$\boxed{\Phi;\underline{\mathcal{R}} \rightsquigarrow \mathit{res\_bind}}$ partial-simplification of resource context: given $\Phi;\underline{\mathcal{R}}$ can partially simplify the resources using $\mathit{res\_bind}$

Elab_Simp_Ctx_Empty

$$\frac{}{\Phi;\cdot \rightsquigarrow \cdot}$$

Elab_Simp_Ctx_Simp

$$\frac{\begin{array}{l}1.\ \Phi \vdash \texttt{simp}\,(\underline{\mathit{res}}) \rightsquigarrow \mathit{res}'\\ 2.\ \Phi \vdash \mathit{res}' \rightsquigarrow \mathit{res\_pat}\\ 3.\ \Phi;\underline{\mathcal{R}} \rightsquigarrow \mathit{res\_bind}\end{array}}{\Phi;\underline{\mathcal{R}}, x{:}\underline{\mathit{res}} \rightsquigarrow \mathit{res\_bind}, \mathit{res\_pat}{:}\mathit{res}' = x}$$

Elab_Simp_Ctx_Skip

$$\frac{\begin{array}{l}1.\ \Phi \vdash \texttt{simp}\,(\mathit{res}) \rightsquigarrow \texttt{None}\\ 2.\ \Phi;\underline{\mathcal{R}} \rightsquigarrow \mathit{res\_bind}\end{array}}{\Phi;\underline{\mathcal{R}}, \_{:}\underline{\mathit{res}} \rightsquigarrow \mathit{res\_bind}}$$

$\boxed{\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash \mathit{seq\_texpr} \Leftarrow \underline{\mathit{ret}} \rightsquigarrow \mathit{seq\_texpr}'}$ top-level seq. expression elaboration: given $\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}}$, $\mathit{seq\_texpr}$ checks against $\underline{\mathit{ret}}$ and elaborates to $\mathit{seq\_texpr}'$

Elab_Top_Seq_TVal

$$\frac{1.\ \mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash \mathit{tval} \Leftarrow \underline{\mathit{ret}} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \mathit{tval}' \dashv \cdot}{\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash \mathit{tval} \Leftarrow \underline{\mathit{ret}} \rightsquigarrow \texttt{insert\_lets}\,(\mathit{res\_bind}, \mathit{tval})}$$

Elab_Top_Seq_LetP

$$\frac{\begin{array}{l} 1.\ \mathcal{C};\mathcal{L};\Phi \vdash \mathit{pexpr} \Rightarrow \Sigma\, y{:}\beta.\ \mathit{term} \wedge \mathtt{I} \\ 2.\ \mathit{ident\_or\_pat}{:}\beta \rightsquigarrow \mathcal{C}_1\ \mathtt{with}\ \mathit{term}_1 \\ 3.\ \mathcal{C},\mathcal{C}_1;\mathcal{L};\Phi, \mathit{term}_1/y(\mathit{term});\underline{\mathcal{R}} \vdash \mathit{texpr} \Leftarrow \underline{\mathit{ret}} \rightsquigarrow \mathit{texpr}' \end{array}}{\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash \mathtt{let}\ \mathit{ident\_or\_pat} = \mathit{pexpr}\ \mathtt{in}\ \mathit{texpr} \Leftarrow \underline{\mathit{ret}} \rightsquigarrow \mathtt{let}\ \mathit{ident\_or\_pat} = \mathit{pexpr}\ \mathtt{in}\ \mathit{texpr}'}$$

Elab_Top_Seq_LetTP

$$\frac{\begin{array}{l} 1.\ \mathcal{C};\mathcal{L};\Phi \vdash \mathit{tpexpr} \Leftarrow \mathit{pure\_ret} \\ 2.\ \mathit{ident\_or\_pat}{:}\beta \rightsquigarrow \mathcal{C}_1\ \mathtt{with}\ \mathit{term}_1 \\ 3.\ \mathcal{C},\mathcal{C}_1;\mathcal{L};\Phi, \mathit{term}_1/y(\mathit{term});\underline{\mathcal{R}} \vdash \mathit{texpr} \Leftarrow \underline{\mathit{ret}} \rightsquigarrow \mathit{texpr}' \end{array}}{\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash \mathtt{let}\ \mathit{ident\_or\_pat}{:}\mathit{pure\_ret} = \mathit{tpexpr}\ \mathtt{in}\ \mathit{texpr} \Leftarrow \underline{\mathit{ret}} \rightsquigarrow \mathtt{let}\ \mathit{ident\_or\_pat}{:}\mathit{pure\_ret} = \mathit{tpexpr}\ \mathtt{in}\ \mathit{texpr}'}$$

Elab_Top_Seq_Let

$$\frac{\begin{array}{l} 1.\ \mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}_1} \vdash \mathit{seq\_expr} \rightsquigarrow \mathtt{bind}\ \mathit{res\_bind}\ \mathtt{for}\ \mathit{seq\_expr}'{:}\Sigma\, y{:}\beta.\ \underline{\mathit{ret}_1} \dashv \underline{\mathcal{R}'_1} \\ 2.\ \Phi \vdash \underline{\mathit{ret}_1} \rightsquigarrow \mathit{ret\_pat} \\ 3.\ \Phi \vdash \mathtt{comp}\ \mathit{ident\_or\_pat}, \mathit{ret\_pat}{:}\Sigma\, y{:}\beta.\ \underline{\mathit{ret}_1} \rightsquigarrow \mathcal{C}_1;\mathcal{L}_1;\Phi_1;\underline{\mathcal{R}''_1} \\ 4.\ \mathcal{C},\mathcal{C}_1;\mathcal{L},\mathcal{L}_1;\Phi,\Phi_1;\underline{\mathcal{R}'_1},\underline{\mathcal{R}''_1},\underline{\mathcal{R}_2} \vdash \mathit{texpr} \Leftarrow \underline{\mathit{ret}_2} \rightsquigarrow \mathit{texpr}' \\ 5.\ \mathit{seq\_texpr}'' \equiv \mathtt{insert\_lets}\,(\mathit{res\_bind}, \mathtt{let\ comp}\ \mathit{ident\_or\_pat}, \mathit{ret\_pat} = \mathit{seq\_expr}'\ \mathtt{in}\ \mathit{texpr}') \end{array}}{\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}_1},\underline{\mathcal{R}_2} \vdash \mathtt{let\ comp}\ \mathit{ident\_or\_pat} = \mathit{seq\_expr}\ \mathtt{in}\ \mathit{texpr} \Leftarrow \underline{\mathit{ret}_2} \rightsquigarrow \mathit{seq\_texpr}''}$$

Elab_Top_Seq_LetT

$$\frac{\begin{array}{l} 1.\ \mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}'} \vdash \mathit{texpr}_1 \Leftarrow \Sigma\, y{:}\beta.\ \underline{\mathit{ret}_1} \rightsquigarrow \mathit{texpr}'_1 \\ 2.\ \Phi \vdash \underline{\mathit{ret}_1} \rightsquigarrow \mathit{ret\_pat} \\ 3.\ \Phi \vdash \mathtt{comp}\ \mathit{ident\_or\_pat}, \mathit{ret\_pat}{:}\Sigma\, y{:}\beta.\ \underline{\mathit{ret}_1} \rightsquigarrow \mathcal{C}_1;\mathcal{L}_1;\Phi_1;\underline{\mathcal{R}_1} \\ 4.\ \mathcal{C},\mathcal{C}_1;\mathcal{L},\mathcal{L}_1;\Phi,\Phi_1;\underline{\mathcal{R}},\underline{\mathcal{R}_1} \vdash \mathit{texpr}_2 \Leftarrow \underline{\mathit{ret}_2} \rightsquigarrow \mathit{texpr}'_2 \\ 5.\ \mathit{seq\_texpr}'' \equiv \mathtt{let\ comp}\ \mathit{ident\_or\_pat}, \mathit{ret\_pat}{:}\Sigma\, y{:}\beta.\ \underline{\mathit{ret}_1} = \mathit{texpr}'_1\ \mathtt{in}\ \mathit{texpr}'_2 \end{array}}{\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}'},\underline{\mathcal{R}} \vdash \mathtt{let\ comp}\ \mathit{ident\_or\_pat}{:}\Sigma\, y{:}\beta.\ \underline{\mathit{ret}_1} = \mathit{texpr}_1\ \mathtt{in}\ \mathit{texpr}_2 \Leftarrow \underline{\mathit{ret}_2} \rightsquigarrow \mathit{seq\_texpr}''}$$

Elab_Top_Seq_Case

1. $\mathcal{C} \vdash pval \Rightarrow \beta_1$
2. $pat_i{:}\beta_1 \rightsquigarrow \mathcal{C}_i$ with $term_i$
3. $\overline{\Phi, term_i = pval; \underline{\mathcal{R}} \rightsquigarrow res\_bind_i}^{i}$
4. $\overline{\mathcal{C}, \mathcal{C}_i; \mathcal{L}; \Phi, term_i = pval; \underline{\mathcal{R}} \vdash \texttt{insert\_lets}\,(res\_bind_i, texpr_i) \Leftarrow \underline{ret} \rightsquigarrow texpr'_i}^{i}$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \texttt{case}\; pval\; \texttt{of}\; \overline{\mid pat_i \Rightarrow texpr_i}^{i}\; \texttt{end} \Leftarrow \underline{ret} \rightsquigarrow \texttt{case}\; pval\; \texttt{of}\; \overline{\mid pat_i \Rightarrow texpr'_i}^{i}\; \texttt{end}}$$

Elab_Top_Seq_If

1. $\mathcal{C} \vdash pval \Rightarrow \texttt{bool}$
2. $\Phi, pval = \texttt{true}; \underline{\mathcal{R}} \rightsquigarrow res\_bind_1$
3. $\mathcal{C}; \mathcal{L}; \Phi, pval = \texttt{true}; \underline{\mathcal{R}}_1 \vdash \texttt{insert\_lets}\,(res\_bind_1, texpr_1) \Leftarrow \underline{ret} \rightsquigarrow texpr'_1$
4. $\Phi, pval = \texttt{false}; \underline{\mathcal{R}}_2 \rightsquigarrow res\_bind_2$
5. $\mathcal{C}; \mathcal{L}; \Phi, pval = \texttt{false}; \underline{\mathcal{R}}_2 \vdash \texttt{insert\_lets}\,(res\_bind_2, texpr_2) \Leftarrow \underline{ret} \rightsquigarrow texpr'_2$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \texttt{if}\; pval\; \texttt{then}\; texpr_1\; \texttt{else}\; texpr_2 \Leftarrow \underline{ret} \rightsquigarrow \texttt{if}\; pval\; \texttt{then}\; \texttt{insert\_lets}\,(res\_bind_1, texpr'_1)\; \texttt{else}\; \texttt{insert\_lets}\,(res\_bind_2, texpr'_2)}$$

Elab_Top_Seq_Run

1. $ident{:}\underline{fun} \equiv \overline{x_i}^{i} \mapsto texpr \in \texttt{Globals}$
2. $\mathcal{C}; \mathcal{L}; \Phi; \cdot \vdash \overline{pval_i}^{i} :: \underline{fun} \gg \texttt{false} \wedge \texttt{I}$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi; \cdot \vdash \texttt{run}\; ident\; \overline{pval_i}^{i} \Leftarrow \texttt{false} \wedge \texttt{I} \rightsquigarrow \texttt{run}\; ident\; \overline{pval_i}^{i}}$$

Elab_Top_Seq_Bound

1. $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash is\_texpr \Leftarrow \underline{ret} \rightsquigarrow \texttt{insert\_lets}\,(res\_bind, is\_texpr')$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \texttt{bound}\,[int](is\_texpr) \Leftarrow \underline{ret} \rightsquigarrow \texttt{insert\_lets}\,(res\_bind, \texttt{bound}\,[int](is\_texpr'))}$$

$\boxed{\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash texpr \Leftarrow \underline{ret} \rightsquigarrow texpr'}$ top-level expression elaboration: given $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}}$, $texpr$ checks against $\underline{ret}$ and elaborates to $texpr'$

$$\textsc{Elab\_Top\_Seq}\quad \frac{1.\ \mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash \mathit{seq\_texpr} \Leftarrow \underline{\mathit{ret}} \rightsquigarrow \mathit{seq\_texpr}'}{\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash \mathit{seq\_texpr} \Leftarrow \underline{\mathit{ret}} \rightsquigarrow \mathit{seq\_texpr}'}$$

$$\textsc{Elab\_Top\_Is}\quad \frac{1.\ \mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash \mathit{is\_texpr} \Leftarrow \underline{\mathit{ret}} \rightsquigarrow \mathit{texpr}}{\mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash \mathit{is\_texpr} \Leftarrow \underline{\mathit{ret}} \rightsquigarrow \mathit{texpr}}$$

A5 Operational Semantics

$\boxed{pat = pval \leadsto \sigma}$ computational value deconstruction: *pat* deconstructs *pval* to produce substitution σ

SUBS_PAT_VALUE_NO_SYM_ANNOT

$$\frac{}{\_:\_ = pval \leadsto \cdot}$$

SUBS_PAT_VALUE_SYM_ANNOT

$$\frac{}{x:\_ = pval \leadsto pval/x}$$

SUBS_PAT_VALUE_NIL

$$\frac{}{\texttt{Nil}\,\beta() = \texttt{Nil}\,\beta() \leadsto \cdot}$$

SUBS_PAT_VALUE_CONS

$$\frac{\begin{array}{l}1.\ pat_1 = pval_1 \leadsto \sigma_1\\ 2.\ pat_2 = pval_2 \leadsto \sigma_2\end{array}}{\texttt{Cons}(pat_1, pat_2) = \texttt{Cons}(pval_1, pval_2) \leadsto [\sigma_1, \sigma_2]}$$

SUBS_PAT_VALUE_TUPLE

$$\frac{1.\ \overline{pat_i = pval_i \leadsto \sigma_i}^{\,i}}{\texttt{Tuple}(\,\overline{pat_i}^{\,i}\,) = \texttt{Tuple}(\,\overline{pval_i}^{\,i}\,) \leadsto [\,\overline{\sigma_i}^{\,i}\,]}$$

SUBS_PAT_VALUE_ARRAY

$$\frac{1.\ \overline{pat_i = pval_i \leadsto \sigma_i}^{\,i}}{\texttt{Array}(\,\overline{pat_i}^{\,i}\,) = \texttt{Array}(\,\overline{pval_i}^{\,i}\,) \leadsto [\,\overline{\sigma_i}^{\,i}\,]}$$

SUBS_PAT_VALUE_SPECIFIED

$$\frac{1.\ pat = pval \leadsto \sigma}{\texttt{Specified}(pat) = \texttt{Specified}(pval) \leadsto \sigma}$$

$\boxed{ident\_or\_pat = pval \leadsto \sigma}$ computational value deconstruction: ident_or_pat deconstructs *pval* to produce substitution σ

SUBS_PAT_VALUE'_SYM

$$\frac{}{x = pval \leadsto pval/x}$$

SUBS_PAT_VALUE'_PAT

$$\frac{1.\ pat = pval \leadsto \sigma}{pat = pval \leadsto \sigma}$$

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

Subs_Pat_Res_Emp

$$\frac{}{\langle h; \texttt{emp} = \texttt{emp}\rangle \rightsquigarrow \langle h; \cdot \rangle}$$

Subs_Pat_Res_Phi

$$\frac{}{\langle h; \texttt{term} = \texttt{term}\rangle \rightsquigarrow \langle h; \cdot \rangle}$$

Subs_Pat_Res_Var

$$\frac{}{\langle h; \mathit{ident} = \mathit{res\_val}\rangle \rightsquigarrow \langle h; \mathit{res\_val}/\mathit{ident} \rangle}$$

Subs_Pat_Res_Pair

$$\frac{\begin{array}{l} 1.\ \langle h; \mathit{res\_pat}_1 = \mathit{res\_val}_1\rangle \rightsquigarrow \langle h_1; \sigma_1 \rangle \\ 2.\ \langle h_1; \mathit{res\_pat}_2 = \mathit{res\_val}_2\rangle \rightsquigarrow \langle h_2; \sigma_2 \rangle \end{array}}{\langle h; \langle \mathit{res\_pat}_1, \mathit{res\_pat}_2\rangle = \langle \mathit{res\_val}_1, \mathit{res\_val}_2\rangle\rangle \rightsquigarrow \langle h_2; [\sigma_1, \sigma_2] \rangle}$$

Subs_Pat_Res_Pack

$$\frac{1.\ \langle h; \mathit{res\_pat} = \mathit{res\_val}\rangle \rightsquigarrow \langle h'; \sigma \rangle}{\langle h; \texttt{pack}\,(\mathit{ident}, \mathit{res\_pat}) = \texttt{pack}\,(\mathit{oarg}, \mathit{res\_val})\rangle \rightsquigarrow \langle h'; [\mathit{oarg}/\mathit{ident}, \sigma] \rangle}$$

Subs_Pat_Res_Fold

$$\frac{1.\ \langle h + h'; \mathit{res\_pat} = \mathit{def}\rangle \rightsquigarrow \langle h''; \sigma \rangle}{\langle h + \{\mathit{pred\_term}(\mathit{oarg})\ \&\ \mathit{def}\ \&\ h'\}; \texttt{fold}\,(\mathit{res\_pat}) = \mathit{pred\_term}\rangle \rightsquigarrow \langle h''; \sigma \rangle}$$

$\boxed{\langle h; \overline{\mathit{ret\_pat}_i = \mathit{ret\_term}_i}^{\,i}\ \rangle \rightsquigarrow \langle h'; \sigma \rangle}$ return value deconstruction: $\mathit{ret\_pat}_i$ deconstructs $\mathit{ret\_val}_i$ to produce substitution σ

Subs_Pat_Ret_Empty

$$\frac{}{\langle h;\ \rangle \rightsquigarrow \langle h; \cdot \rangle}$$

Subs_Pat_Ret_Comp

$$\frac{\begin{array}{l} 1.\ \mathit{ident\_or\_pat} = \mathit{pval} \rightsquigarrow \sigma \\ 2.\ \langle h; \overline{\mathit{ret\_pat}_i = \mathit{ret\_term}_i}^{\,i}\ \rangle \rightsquigarrow \langle h'; \psi \rangle \end{array}}{\langle h; \texttt{comp}\,\mathit{ident\_or\_pat} = \mathit{pval}, \overline{\mathit{ret\_pat}_i = \mathit{ret\_term}_i}^{\,i}\ \rangle \rightsquigarrow \langle \sigma(h'); [\sigma, \psi] \rangle}$$

Subs_Pat_Ret_Log

$$\frac{1.\ \langle h; \overline{\mathit{ret\_pat}_i = \mathit{ret\_term}_i}^{\,i}\ \rangle \rightsquigarrow \langle h'; \sigma \rangle}{\langle h; \texttt{log}\,y = \mathit{oarg}, \overline{\mathit{ret\_pat}_i = \mathit{ret\_term}_i}^{\,i}\ \rangle \rightsquigarrow \langle \mathit{oarg}/y(h'); [\mathit{oarg}/y, \sigma] \rangle}$$

SUBS_PAT_RET_RES

$$\frac{\begin{array}{l}1.\ \langle h; \mathit{res\_term}\rangle \Downarrow \langle h_1; \mathit{res\_val}\rangle\\ 2.\ \langle h; \mathit{res\_pat} = \mathit{res\_val}\rangle \rightsquigarrow \langle h_2; \sigma\rangle\\ 3.\ \langle h_2; \overline{\mathit{ret\_pat}_i = \mathit{ret\_term}_i}^{\,i}\rangle \rightsquigarrow \langle h_3; \psi\rangle\end{array}}{\langle h; \mathtt{res}\ \mathit{res\_pat} = \mathit{res\_term},\ \overline{\mathit{ret\_pat}_i = \mathit{ret\_term}_i}^{\,i}\rangle \rightsquigarrow \langle h_3; [\sigma, \psi]\rangle}$$

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

SUBS_SPINE_EMPTY

$$\frac{}{\langle h;\ \rangle :: \mathit{ret} \gg \langle h; \cdot; \mathit{ret}\rangle}$$

SUBS_SPINE_COMP

$$\frac{1.\ \langle h;\ \overline{x_i = \mathit{spine\_elem}_i}^{\,i}\rangle :: \mathit{pval}/x(\mathit{fun}) \gg \langle h'; \sigma; \mathit{ret}\rangle}{\langle h; x = \mathit{pval},\ \overline{x_i = \mathit{spine\_elem}_i}^{\,i}\rangle :: \Pi\, x{:}\beta.\ \mathit{fun} \gg \langle h'; [\mathit{pval}/x, \sigma]; \mathit{ret}\rangle}$$

SUBS_SPINE_LOG

$$\frac{1.\ \langle h;\ \overline{x_i = \mathit{spine\_elem}_i}^{\,i}\rangle :: \mathit{oarg}/x(\mathit{fun}) \gg \langle h'; \sigma; \mathit{ret}\rangle}{\langle h; x = \mathit{oarg},\ \overline{x_i = \mathit{spine\_elem}_i}^{\,i}\rangle :: \forall\, x{:}\beta.\ \mathit{fun} \gg \langle h'; [\mathit{pval}/x, \sigma]; \mathit{ret}\rangle}$$

SUBS_SPINE_RES

$$\frac{\begin{array}{l}1.\ \langle h; \mathit{res\_term}\rangle \Downarrow \langle h'; \mathit{res\_val}\rangle\\ 2.\ \langle h';\ \overline{x_i = \mathit{spine\_elem}_i}^{\,i}\rangle :: \mathit{fun} \gg \langle h''; \sigma; \mathit{ret}\rangle\end{array}}{\langle h; x = \mathit{res\_term},\ \overline{x_i = \mathit{spine\_elem}_i}^{\,i}\rangle :: \mathit{res} \twoheadrightarrow \mathit{fun} \gg \langle h''; [\mathit{res\_val}/x, \sigma]; \mathit{ret}\rangle}$$

SUBS_SPINE_PHI

$$\frac{1.\ \langle h;\ \overline{x_i = \mathit{spine\_elem}_i}^{\,i}\rangle :: \mathit{fun} \gg \langle h'; \sigma; \mathit{ret}\rangle}{\langle h;\ \overline{x_i = \mathit{spine\_elem}_i}^{\,i}\rangle :: \mathit{term} \supset \mathit{fun} \gg \langle h'; \sigma; \mathit{ret}\rangle}$$

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

PE_TP_ARRAY_SHIFT

$$\frac{\begin{array}{l}1.\ \mathit{mem\_ptr}' \equiv \mathit{mem\_ptr} +_{\mathrm{ptr}} (\mathit{mem\_int} \times \mathrm{size\_of}(\tau)) \\ 2.\ \mathit{pure\_ret} \equiv \Sigma\, y{:}\mathtt{pointer}.\ y = \mathit{mem\_ptr} +_{\mathrm{ptr}} (\mathit{mem\_int} \times \mathrm{size\_of}(\tau)) \wedge \mathtt{I}\end{array}}{\langle \mathtt{array\_shift}\,(\mathit{mem\_ptr}, \tau, \mathit{mem\_int})\rangle \longrightarrow \langle \mathtt{done}\, \mathit{mem\_ptr}'{:}\mathit{pure\_ret}\rangle}$$

PE_TP_MEMBER_SHIFT

$$\frac{\begin{array}{l}1.\ \mathit{mem\_ptr}' \equiv \mathit{mem\_ptr} +_{\mathrm{ptr}} \mathrm{offset\_of}_{\mathit{tag}}(\mathit{member}) \\ 2.\ \mathit{pure\_ret} \equiv \Sigma\, y{:}\mathtt{pointer}.\ y = \mathit{mem\_ptr} +_{\mathrm{ptr}} \mathrm{offset\_of}_{\mathit{tag}}(\mathit{member}) \wedge \mathtt{I}\end{array}}{\langle \mathtt{member\_shift}\,(\mathit{mem\_ptr}, \mathit{tag}, \mathit{member})\rangle \longrightarrow \langle \mathtt{done}\, \mathit{mem\_ptr}'{:}\mathit{pure\_ret}\rangle}$$

PE_TP_NOT_TRUE

$$\frac{}{\langle \mathtt{not}\,(\mathtt{True})\rangle \longrightarrow \langle \mathtt{done}\, \mathtt{False}{:}\Sigma\, y{:}\mathtt{bool}.\ y = \neg\, \mathtt{True} \wedge \mathtt{I}\rangle}$$

PE_TP_NOT_FALSE

$$\frac{}{\langle \mathtt{not}\,(\mathtt{False})\rangle \longrightarrow \langle \mathtt{done}\, \mathtt{True}{:}\Sigma\, y{:}\mathtt{bool}.\ y = \neg\, \mathtt{False} \wedge \mathtt{I}\rangle}$$

PE_TP_ARITH_BINOP

$$\frac{\begin{array}{l}1.\ \mathit{mem\_int} \equiv \mathit{mem\_int}_1\ \mathit{binop}_{\mathit{arith}}\ \mathit{mem\_int}_2 \\ 2.\ \mathit{pure\_ret} \equiv \Sigma\, y{:}\mathtt{integer}.\ y = \mathit{mem\_int}_1\ \mathit{binop}_{\mathit{arith}}\ \mathit{mem\_int}_2 \wedge \mathtt{I}\end{array}}{\langle \mathit{mem\_int}_1\ \ \mathit{binop}_{\mathit{arith}}\ \ \mathit{mem\_int}_2\rangle \longrightarrow \langle \mathtt{done}\, \mathit{mem\_int}{:}\mathit{pure\_ret}\rangle}$$

PE_TP_REL_BINOP

$$\frac{\begin{array}{l}1.\ \mathit{bool\_value} \equiv \mathit{mem\_int}_1\ \mathit{binop}_{\mathit{rel}}\ \mathit{mem\_int}_2 \\ 2.\ \mathit{pure\_ret} \equiv \Sigma\, y{:}\mathtt{bool}.\ y = \mathit{mem\_int}_1\ \mathit{binop}_{\mathit{rel}}\ \mathit{mem\_int}_2 \wedge \mathtt{I}\end{array}}{\langle \mathit{mem\_int}_1\ \ \mathit{binop}_{\mathit{rel}}\ \ \mathit{mem\_int}_2\rangle \longrightarrow \langle \mathtt{done}\, \mathit{bool\_value}{:}\mathit{pure\_ret}\rangle}$$

PE_TP_BOOL_BINOP

$$\frac{\begin{array}{l}1.\ \mathit{bool\_value} \equiv \mathit{bool\_value}_1\ \mathit{binop}_{\mathit{bool}}\ \mathit{bool\_value}_2 \\ 2.\ \mathit{pure\_ret} \equiv \Sigma\, y{:}\mathtt{bool}.\ y = \mathit{bool\_value}_1\ \mathit{binop}_{\mathit{bool}}\ \mathit{bool\_value}_2 \wedge \mathtt{I}\end{array}}{\langle \mathit{bool\_value}_1\ \ \mathit{binop}_{\mathit{bool}}\ \ \mathit{bool\_value}_2\rangle \longrightarrow \langle \mathtt{done}\, \mathit{bool\_value}{:}\mathit{pure\_ret}\rangle}$$

PE_TP_ASSERT_UNDEF

$$\frac{}{\langle \mathtt{assert\_undef}\,(\mathtt{True},\ \mathit{UB\_name})\rangle \longrightarrow \langle \mathtt{done}\, \mathtt{Unit}{:}\Sigma\, \_{:}\mathtt{unit}.\ \mathtt{I}\rangle}$$

PE_TP_BOOL_TO_INTEGER_TRUE

$$\frac{}{\langle \mathtt{bool\_to\_integer}\,(\mathtt{True})\rangle \longrightarrow \langle \mathtt{done}\, 1{:}\Sigma\, y{:}\mathtt{integer}.\ y = 1 \wedge \mathtt{I}\rangle}$$

PE_TP_BOOL_TO_INTEGER_FALSE

$$\frac{}{\langle \texttt{bool\_to\_integer}\,(\texttt{False})\rangle \longrightarrow \langle \texttt{done}\, 0{:}\Sigma\, y{:}\texttt{integer}.\ y = 0 \wedge \texttt{I}\rangle}$$

PE_TP_WRAPI

1. $abbrev_1 \equiv \max\_int_\tau - \min\_int_\tau + 1$
2. $abbrev_2 \equiv pval \text{ rem } abbrev_1$
3. $mem\_int' \equiv \texttt{if}\ abbrev_2 \leq \max\_int_\tau\ \texttt{then}\ abbrev_2\ \texttt{else}\ abbrev_2 - abbrev_1$
4. $pure\_ret \equiv \Sigma\, y{:}\texttt{integer}.\ y = mem\_int' \wedge \texttt{I}$

$$\langle \texttt{wrapI}\,(\tau, mem\_int)\rangle \longrightarrow \langle \texttt{done}\, mem\_int'{:}pure\_ret\rangle$$

PE_TP_CALL

1. $name{:}pure\_fun \equiv \overline{x_i}^{\,i} \mapsto tpexpr \in \texttt{Globals}$
2. $\langle \cdot;\ \overline{x_i = pval_i}^{\,i}\rangle :: pure\_fun \gg \langle \cdot;\ \sigma;\ pure\_ret\rangle$

$$\langle name(\,\overline{pval_i}^{\,i}\,)\rangle \longrightarrow \langle \sigma(tpexpr){:}pure\_ret\rangle$$

$\boxed{\langle tpexpr\rangle \longrightarrow \langle tpexpr'\rangle}$

TP_TP_CASE

1. $pat_j = pval \rightsquigarrow \sigma_j$
2. $\forall\, i < j.\ \texttt{not}\,(pat_i = pval \rightsquigarrow \sigma_i)$

$$\langle \texttt{case}\ pval\ \texttt{of}\ \overline{\mid\ pat_i \Rightarrow tpexpr_i}^{\,i}\ \texttt{end}\rangle \longrightarrow \langle \sigma_j(tpexpr_j)\rangle$$

TP_TP_LET_SUB

1. $ident\_or\_pat = pval \rightsquigarrow \sigma$

$$\langle \texttt{let}\ ident\_or\_pat = pval\ \texttt{in}\ tpexpr\rangle \longrightarrow \langle \sigma(tpexpr)\rangle$$

TP_TP_LET_LET

1. $\langle pexpr\rangle \longrightarrow \langle tpval{:}pure\_ret\rangle$

$$\langle \texttt{let}\ ident\_or\_pat = pexpr\ \texttt{in}\ tpexpr\rangle \longrightarrow \langle \texttt{let}\ ident\_or\_pat{:}pure\_ret = tpval\ \texttt{in}\ tpexpr\rangle$$

TP_TP_LET_LETT

1. $\langle pexpr\rangle \longrightarrow \langle tpexpr_1{:}pure\_ret\rangle$

$$\langle \texttt{let}\ ident\_or\_pat = pexpr\ \texttt{in}\ tpexpr_2\rangle \longrightarrow \langle \texttt{let}\ ident\_or\_pat{:}pure\_ret = tpexpr_1\ \texttt{in}\ tpexpr_2\rangle$$

TP_TP_LetT_Sub

$$\frac{1.\ ident\_or\_pat = pval \rightsquigarrow \sigma}{\langle \texttt{let}\ ident\_or\_pat{:}pure\_ret = \texttt{done}\ pval\ \texttt{in}\ tpexpr \rangle \longrightarrow \langle \sigma(tpexpr) \rangle}$$

TP_TP_LetT_LetT

$$\frac{1.\ \langle tpexpr_1 \rangle \longrightarrow \langle tpexpr'_1 \rangle}{\langle \texttt{let}\ ident\_or\_pat{:}pure\_ret = tpexpr_1\ \texttt{in}\ tpexpr_2 \rangle \longrightarrow \langle \texttt{let}\ ident\_or\_pat{:}pure\_ret = tpexpr'_1\ \texttt{in}\ tpexpr_2 \rangle}$$

TP_TP_If_True

$$\frac{}{\langle \texttt{if True then}\ tpexpr_1\ \texttt{else}\ tpexpr_2 \rangle \longrightarrow \langle tpexpr_1 \rangle}$$

TP_TP_If_False

$$\frac{}{\langle \texttt{if False then}\ tpexpr_1\ \texttt{else}\ tpexpr_2 \rangle \longrightarrow \langle tpexpr_2 \rangle}$$

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

PredOps_ResV_Iterate

1. $\langle h; res\_term \rangle \Downarrow \langle h' + \{pred\_term(oarg)\ \&\ \texttt{None}\}; pred\_term \rangle$
2. $pred\_term \equiv \texttt{Owned}\,\langle \texttt{array}\ n\ \tau \rangle(ptr)$
3. $qpred\_term \equiv (x; 0 \leq x \wedge x \leq n-1)\{\texttt{Owned}\,\langle \tau \rangle(ptr + x \times \text{size\_of}(\tau))\}$
4. $oarg'[x].init \equiv oarg.init[x]$
5. $oarg'[x].value \equiv oarg.value[x]$

$$\overline{\langle h; \texttt{iterate}\,(res\_term, n) \rangle \Downarrow \langle h' + \{qpred\_term(oarg')\ \&\ \cdot\}; qpred\_term \rangle}$$

$$\textsc{PredOps\_ResV\_Congeal}$$

$$\frac{\begin{array}{l} 1.\ \langle h; res\_term\rangle \Downarrow \langle h' + \{qpred\_term(oarg)\ \&\ \cdot\}; qpred\_term\rangle \\ 2.\ qpred\_term \equiv (x; iguard)\{\texttt{Owned}\,\langle\tau\rangle(ptr + x \times \mathrm{size\_of}(\tau))\} \\ 3.\ \texttt{smt}\ (\cdot \Rightarrow \forall\, x.\ iguard \leftrightarrow (0 \leq x \wedge x \leq n-1)) \\ 4.\ pred\_term \equiv \texttt{Owned}\,\langle \texttt{array}\ n\ \tau\rangle(ptr) \\ 5.\ oarg'.init[x] \equiv oarg[x].init \\ 6.\ oarg'.value[x] \equiv oarg[x].value \end{array}}{\langle h; \texttt{congeal}\,(res\_term, n)\rangle \Downarrow \langle h' + \{pred\_term(oarg')\ \&\ \texttt{None}\}; pred\_term\rangle}$$

$$\textsc{PredOps\_ResV\_Explode}$$

$$\frac{\begin{array}{l} 1.\ \langle h; res\_term\rangle \Downarrow \langle h' + \{pred\ \&\ \texttt{None}\}; pred\_term\rangle \\ 2.\ pred \equiv \texttt{Owned}\,\langle \texttt{struct}\ tag\rangle(ptr)(oarg) \\ 3.\ \texttt{struct}\ tag\ \&\ \overline{member_i{:}\tau_i}^{\,i} \in \texttt{Globals} \\ 4.\ ptr_i \equiv ptr + \mathrm{offset\_of}_{tag}(member_i) \\ 5.\ pred_i \equiv \texttt{Owned}\,\langle\tau_i\rangle(ptr_i)(oarg_i) \\ 6.\ oarg_i.init \equiv oarg.init.member_i \\ 7.\ oarg_i.value \equiv oarg.value.member_i \end{array}}{\langle h; \texttt{explode}\,(res\_term)\rangle \Downarrow \langle h' + \overline{\{pred_i\ \&\ \texttt{None}\}}^{\,i}; \langle\,\overline{pred\_term_i}^{\,i}\,\rangle\rangle}$$

$$\textsc{PredOps\_ResV\_Implode}$$

$$\dfrac{\begin{array}{l} 1.\ \langle h; \mathit{res\_term}\rangle \Downarrow \langle h' + \overline{\{\mathit{pred\_term}_i(\mathit{oarg}_i)\ \&\ \texttt{None}\}}^{i}; \langle \overline{\mathit{pred\_term}_i}^{i} \rangle\rangle \\ 2.\ \texttt{struct}\ \mathit{tag}\ \&\ \overline{\mathit{member}_i{:}\tau_i}^{i} \in \texttt{Globals} \\ 3.\ \mathit{pred\_term}_i \equiv \texttt{Owned}\langle\tau_i\rangle(\mathit{ptr}_i) \\ 4.\ \mathit{ptr} \equiv \mathit{ptr}_0 - \mathrm{offset\_of}_{\mathit{tag}}(\mathit{member}_0) \\ 5.\ \texttt{smt}\ (\cdot \Rightarrow \bigwedge(\overline{\mathit{ptr} = \mathit{ptr}_i - \mathrm{offset\_of}_{\mathit{tag}}(\mathit{member}_i)}^{i})) \\ 6.\ \mathit{pred\_term} \equiv \texttt{Owned}\langle\texttt{struct}\ \mathit{tag}\rangle(\mathit{ptr}) \\ 7.\ \mathit{oarg}.\mathit{init}.\mathit{member}_i \equiv \mathit{oarg}_i.\mathit{init} \\ 8.\ \mathit{oarg}.\mathit{value}.\mathit{member}_i \equiv \mathit{oarg}_i.\mathit{value} \end{array}}{\langle h; \texttt{implode}\,(\mathit{res\_term}, \mathit{tag})\rangle \Downarrow \langle h' + \{\mathit{pred\_term}(\mathit{oarg})\ \&\ \texttt{None}\}; \mathit{pred\_term}\rangle}$$

$$\textsc{PredOps\_ResV\_Break}$$

$$\dfrac{\begin{array}{l} 1.\ \langle h; \mathit{res\_term}\rangle \Downarrow \langle h' + \{\mathit{qpred\_term}(\mathit{oarg})\ \&\ \mathit{arr\_def\_heap}\}; \mathit{qpred\_term}\rangle \\ 2.\ \mathit{qpred\_term} \equiv (x; \mathit{iguard})\{\alpha(\mathit{ptr} + x \times \mathit{step}, \mathit{iargs})\} \\ 3.\ \texttt{smt}\ (\cdot \Rightarrow \mathit{term}/x(\mathit{iguard})) \\ 4.\ \mathit{ptr}' \equiv \mathit{ptr} +_{\mathrm{ptr}} (\mathit{term} \times \mathit{step}) \\ 5.\ \mathit{qpred\_term}' \equiv (x; \mathit{iguard} \wedge (x \neq \mathit{term}))\{\alpha(\mathit{ptr} + x \times \mathit{step}, \mathit{iargs})\} \\ 6.\ \mathit{pred\_term} \equiv \alpha(\mathit{ptr}', \mathit{term}/x(\mathit{iargs})) \end{array}}{\langle h; \texttt{break}\,(\mathit{res\_term}, \mathit{term})\rangle \Downarrow \langle h' + \{\mathit{qpred\_term}'(\mathit{oarg})\ \&\ \mathit{arr\_def\_heap}\} + \{\mathit{pred\_term}(\mathit{oarg}[\mathit{term}])\ \&\ \mathit{arr\_def\_heap}[\mathit{term}]\}; \langle \mathit{qpred\_term}', \mathit{pred\_term}\rangle\rangle}$$

$$\textsc{PredOps\_ResV\_Glue}$$

$$\dfrac{\begin{array}{l} 1.\ \langle h; \mathit{res\_term}\rangle \Downarrow \langle h' + \{\mathit{qpred\_term}(\mathit{oarg})\ \&\ \mathit{arr\_def\_heap}\} + \{\mathit{pred\_term}(\mathit{oarg}')\ \&\ \mathit{opt\_def\_heap}\}; \langle \mathit{qpred\_term}, \mathit{pred\_term}\rangle\rangle \\ 2.\ \mathit{qpred\_term} \equiv (x; \mathit{iguard})\{\alpha(\mathit{ptr} + x \times \mathit{step}, \overline{\mathit{iarg}_i}^{i})\} \\ 3.\ \mathit{pred\_term} \equiv \alpha(\mathit{ptr}', \overline{\mathit{iarg}'_i}^{i}) \\ 4.\ \mathit{term} \equiv (\mathit{ptr}' - \mathit{ptr})/\mathrm{size\_of}(\tau) \\ 5.\ \texttt{smt}\ (\cdot \Rightarrow \bigwedge(\overline{\mathit{term}/x(\mathit{iarg}_i) = \mathit{iarg}'_i}^{i})) \\ 6.\ \mathit{qpred\_term} \equiv (x; \mathit{iguard} \vee x = \mathit{term})\{\alpha(\mathit{ptr} + x \times \mathit{step}, \mathit{iargs})\} \end{array}}{\langle h; \texttt{glue}\,(\mathit{res\_term})\rangle \Downarrow \langle h' + \{\mathit{qpred\_term}(\mathit{oarg}[\mathit{term}] := \mathit{oarg}')\ \&\ \mathit{arr\_def\_heap}[\mathit{term}] := \mathit{opt\_def\_heap}\}; \mathit{qpred\_term}\rangle}$$

PredOps_ResV_Inj

1. $\langle h; \mathit{res\_term}\rangle \Downarrow \langle h' + \{\mathit{pred\_term}(\mathit{oarg}) \,\&\, \mathit{opt\_def\_heap}\}; \mathit{pred\_term}\rangle$
2. $\mathit{pred\_term} \equiv \alpha(\mathit{ptr}_2, \overline{\mathit{iarg}_{2\,i}}^{\,i})$
3. $\mathit{term} \equiv (\mathit{ptr}_2 - \mathit{ptr}_1)/\mathit{step}$
4. $\texttt{smt}\,(\cdot \Rightarrow \bigwedge(\overline{\mathit{term}/x(\mathit{iarg}_{1\,i}) = \mathit{iarg}_{2\,i}}^{\,i}))$
5. $\mathit{qpred\_term} \equiv (x; x = \mathit{term})\{\alpha(\mathit{ptr}_1 + x\times\mathit{step}, \overline{\mathit{iarg}_{1\,i}}^{\,i})\}$
6. $\cdot;\cdot \vdash \mathit{oarg} \Rightarrow \beta$
7. $\mathit{oarg}' \equiv (\texttt{default}\,\beta)[\mathit{term}] := \mathit{oarg}$

$$\overline{\langle h; \texttt{inj}\,(\mathit{res\_term}, \mathit{ptr}_1, \mathit{step}, x.\,\overline{\mathit{iarg}_1}^{\,i})\rangle \Downarrow \langle h' + \{\mathit{qpred\_term}(\mathit{oarg}') \,\&\, \cdot[\mathit{term}] := \mathit{opt\_def\_heap}\}; \mathit{qpred\_term}\rangle}$$

PredOps_ResV_Split

1. $\langle h; \mathit{res\_term}\rangle \Downarrow \langle h' + \{\mathit{qpred\_term}(\mathit{oarg}) \,\&\, \mathit{arr\_def\_heap}\}; \mathit{qpred\_term}\rangle$
2. $\mathit{qpred\_term} \equiv (x; \mathit{iguard}')\{\alpha(\mathit{ptr} + x\times\mathit{step}, \mathit{iargs})\}$
3. $\texttt{smt}\,(\cdot \Rightarrow \forall\, x.\ \mathit{iguard} \rightarrow \mathit{iguard}')$
4. $\mathit{qpred\_term}_1 \equiv (x; \mathit{iguard})\{\alpha(\mathit{ptr} + x\times\mathit{step}, \mathit{iargs})\}$
5. $\mathit{qpred\_term}_2 \equiv (x; \mathit{iguard}' \wedge \neg\, \mathit{iguard})\{\alpha(\mathit{ptr} + x\times\mathit{step}, \mathit{iargs})\}$

$$\overline{\langle h; \texttt{split}\,(\mathit{res\_term}, \mathit{iguard})\rangle \Downarrow \langle h' + \{\mathit{qpred\_term}_1(\mathit{oarg}) \,\&\, \mathit{arr\_def\_heap}\} + \{\mathit{qpred\_term}_2(\mathit{oarg}) \,\&\, \mathit{arr\_def\_heap}\}; \langle \mathit{qpred\_term}_1, \mathit{qpred\_term}_2\rangle\rangle}$$

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

Footprint_Emp

$$\overline{\texttt{footprint\_of emp in}\ h \rightsquigarrow \cdot\ \text{rem}\ h}$$

Footprint_Term

$$\overline{\texttt{footprint\_of term in}\ h \rightsquigarrow \cdot\ \text{rem}\ h}$$

Footprint_Pred

$$\overline{\texttt{footprint\_of}\ \mathit{pred\_term}\ \texttt{in}\ h + \{\mathit{pred\_term}(\mathit{oarg}) \,\&\, \mathit{opt\_def\_heap}\} \rightsquigarrow \{\mathit{pred\_term}(\mathit{oarg}) \,\&\, \mathit{opt\_def\_heap}\}\ \text{rem}\ h}$$

FOOTPRINT_QPRED

$$\frac{}{\texttt{footprint\_of}\ \mathit{qpred\_term}\ \texttt{in}\ h + \{\mathit{qpred\_term}(\mathit{oarg})\ \&\ \mathit{arr\_def\_heap}\} \rightsquigarrow \{\mathit{qpred\_term}(\mathit{oarg})\ \&\ \mathit{arr\_def\_heap}\}\ \text{rem}\ h}$$

FOOTPRINT_SEPPAIR

$$\frac{\begin{array}{l}1.\ \texttt{footprint\_of}\ \mathit{res\_val}_1\ \texttt{in}\ h \rightsquigarrow h_1\ \text{rem}\ h' \\ 2.\ \texttt{footprint\_of}\ \mathit{res\_val}_2\ \texttt{in}\ h' \rightsquigarrow h_2\ \text{rem}\ h''\end{array}}{\texttt{footprint\_of}\ \langle \mathit{res\_val}_1, \mathit{res\_val}_2\rangle\ \texttt{in}\ h \rightsquigarrow h_1 + h_2\ \text{rem}\ h''}$$

FOOTPRINT_PACK

$$\frac{1.\ \texttt{footprint\_of}\ \mathit{res\_val}\ \texttt{in}\ h \rightsquigarrow h'\ \text{rem}\ h''}{\texttt{footprint\_of}\ \texttt{pack}\,(\mathit{oarg}, \mathit{res\_val})\ \texttt{in}\ h \rightsquigarrow h'\ \text{rem}\ h''}$$

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

REST_RESV_VAL

$$\frac{}{\langle h; \mathit{res\_val}\rangle \Downarrow \langle h; \mathit{res\_val}\rangle}$$

REST_RESV_SEPPAIR

$$\frac{\begin{array}{l}1.\ \langle h; \mathit{res\_term}_1\rangle \Downarrow \langle h_1; \mathit{res\_val}_1\rangle \\ 2.\ \langle h_1; \mathit{res\_term}_2\rangle \Downarrow \langle h_2; \mathit{res\_val}_2\rangle\end{array}}{\langle h; \langle \mathit{res\_term}_1, \mathit{res\_term}_2\rangle\rangle \Downarrow \langle h_2; \langle \mathit{res\_val}_1, \mathit{res\_val}_2\rangle\rangle}$$

REST_RESV_PREDOPS

$$\frac{1.\ \langle h; \mathit{pred\_ops}\rangle \Downarrow \langle h'; \mathit{res\_val}\rangle}{\langle h; \mathit{pred\_ops}\rangle \Downarrow \langle h'; \mathit{res\_val}\rangle}$$

REST_RESV_FOLD

$$\frac{\begin{array}{l}1.\ \langle h; \mathit{res\_term}\rangle \Downarrow \langle h_1; \mathit{def}\rangle \\ 2.\ \texttt{footprint\_of}\ \mathit{def}\ \texttt{in}\ h_1 \rightsquigarrow h_2\ \text{rem}\ h_3\end{array}}{\langle h; \texttt{fold}\ \mathit{res\_term}{:}\mathit{pred\_term}(\mathit{oarg})\rangle \Downarrow \langle h_3 + \{\mathit{pred\_term}(\mathit{oarg})\ \&\ \mathit{def}\ \&\ h_2\}; \mathit{pred\_term}\rangle}$$

REST_RESV_PACK

$$\frac{1.\ \langle h; \mathit{res\_term}\rangle \Downarrow \langle h'; \mathit{res\_val}\rangle}{\langle h; \texttt{pack}\,(\mathit{oarg}, \mathit{res\_term})\rangle \Downarrow \langle h'; \texttt{pack}\,(\mathit{oarg}, \mathit{res\_val})\rangle}$$

$\boxed{\langle h; \mathit{action}\rangle \longrightarrow \langle h'; \mathit{is\_expr}\rangle}$

Action_Is_Create

1. $\texttt{fresh}\,(\mathit{mem\_ptr})$
2. $\texttt{representable}\,(\tau*, \mathit{mem\_ptr}) \wedge \texttt{alignedI}\,(\mathit{mem\_int}, \mathit{mem\_ptr})$
3. $\mathit{pt} \;\equiv\; \mathit{mem\_ptr} \overset{\texttt{const}_\tau\texttt{false}}{\mapsto_\tau} \texttt{default}\;\beta_\tau$
4. $\mathit{ret} \;\equiv\; \Sigma\, y_p{:}\texttt{pointer}.\; \mathit{term} \wedge (y_p \overset{\texttt{const}_\tau\texttt{false}}{\mapsto_\tau} \texttt{default}\;\beta_\tau) * \texttt{I}$
5. $\mathit{term} \;\equiv\; \texttt{representable}\,(\tau*, y_p) \wedge \texttt{alignedI}\,(\mathit{mem\_int}, y_p)$

$$\langle h; \texttt{create}\,(\mathit{mem\_int}, \tau)\rangle \longrightarrow \langle h + \{\mathit{pt}\;\&\;\texttt{None}\}; \texttt{done}\,\langle \mathit{mem\_ptr}, \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr})\rangle{:}\mathit{ret}\rangle$$

Action_Is_Load

1. $\langle h; \mathit{res\_term}\rangle \Downarrow \langle h' + \{\mathit{pt}\;\&\;\texttt{None}\}; \texttt{Owned}\,\langle\tau\rangle(\mathit{term})\rangle$
2. $\mathit{pt} \;\equiv\; \mathit{term} \overset{\mathit{init}}{\mapsto_\tau} \mathit{pval}$
3. $\texttt{smt}\,(\cdot \Rightarrow \mathit{term} = \mathit{mem\_ptr} \wedge \mathit{init} = \texttt{const}_\tau\texttt{true})$
4. $\mathit{ret} \;\equiv\; \Sigma\, y{:}\beta_\tau.\; y = \mathit{pval} \wedge (\mathit{mem\_ptr} \overset{\texttt{const}_\tau\texttt{true}}{\mapsto_\tau} \mathit{pval}) * \texttt{I}$

$$\langle h; \texttt{load}\,(\tau, \mathit{mem\_ptr}, \_, \mathit{res\_term})\rangle \longrightarrow \langle h' + \{\mathit{pt}\;\&\;\texttt{None}\}; \texttt{done}\,\langle \mathit{pval}, \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr})\rangle{:}\mathit{ret}\rangle$$

Action_Is_Store

1. $\langle h; \mathit{res\_term}\rangle \Downarrow \langle h' + \{\mathit{pt}\;\&\;\texttt{None}\}; \texttt{Owned}\,\langle\tau\rangle(\mathit{term})\rangle$
2. $\mathit{pt} \;\equiv\; \mathit{term} \overset{\_}{\mapsto_\tau} \_$
3. $\texttt{smt}\,(\cdot \Rightarrow \mathit{term} = \mathit{mem\_ptr})$
4. $\texttt{smt}\,(\cdot \Rightarrow \texttt{representable}\,(\tau, \mathit{pval}))$
5. $\mathit{pt}' \;\equiv\; \mathit{mem\_ptr} \overset{\texttt{const}_\tau\texttt{true}}{\mapsto_\tau} \mathit{pval}$
6. $\mathit{ret} \;\equiv\; \Sigma\, \_{:}\texttt{unit}.\; (\mathit{mem\_ptr} \overset{\texttt{const}_\tau\texttt{true}}{\mapsto_\tau} \mathit{pval}) * \texttt{I}$

$$\langle h; \texttt{store}\,(\_, \tau, \mathit{mem\_ptr}, \mathit{pval}, \_, \mathit{res\_term})\rangle \longrightarrow \langle h' + \{\mathit{pt}'\;\&\;\texttt{None}\}; \texttt{done}\,\langle \texttt{Unit}, \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr})\rangle{:}\mathit{ret}\rangle$$

Action_Is_Kill_Static

$$\begin{array}{l}
1.\ \langle h; \mathit{res_term}\rangle \Downarrow \langle h' + \{\mathit{pt}\ \&\ \texttt{None}\}; \texttt{Owned}\,\langle\tau\rangle(\mathit{term})\rangle \\
2.\ \mathit{pt}\ \equiv\ \mathit{term} \overset{_}{\mapsto}_{\tau}\ _ \\
3.\ \texttt{smt}\,(\cdot \Rightarrow \mathit{term} = \mathit{mem_ptr}) \\
4.\ \mathit{ret}\ \equiv\ \Sigma\,_{:}\texttt{unit}.\ \texttt{I}
\end{array}$$
$$\overline{\langle h; \texttt{kill}\,(\texttt{static}\,\tau, \mathit{mem\_ptr}, \mathit{res\_term})\rangle \longrightarrow \langle h'; \texttt{done}\,\langle\texttt{Unit}\rangle{:}\mathit{ret}\rangle}$$

$$\boxed{\langle h; \mathit{memop}\rangle \longrightarrow \langle h'; \mathit{is\_expr}\rangle}$$

Memop_Is_Rel_Binop

$$\begin{array}{l}
1.\ \mathit{bool_value}\ \equiv\ \mathit{mem_int}_1\ \mathit{binop}_{rel}\ \mathit{mem_int}_2 \\
2.\ \mathit{ret}\ \equiv\ \Sigma\, y{:}\texttt{bool}.\ y = \mathit{bool_value} \wedge \texttt{I}
\end{array}$$
$$\overline{\langle h; \mathit{mem\_int}_1\ \mathit{binop}_{rel}\ \mathit{mem\_int}_2\rangle \longrightarrow \langle h; \texttt{done}\,\langle \mathit{bool\_value}\rangle{:}\mathit{ret}\rangle}$$

Memop_Is_IntFromPtr

$$\begin{array}{l}
1.\ \mathit{mem_int}\ \equiv\ \texttt{cast_ptr_to_int}\,\mathit{mem_ptr} \\
2.\ \mathit{ret}\ \equiv\ \Sigma\, y{:}\texttt{integer}.\ y = \mathit{mem_int} \wedge \texttt{I}
\end{array}$$
$$\overline{\langle h; \texttt{intFromPtr}\,(\tau_1, \tau_2, \mathit{mem\_ptr})\rangle \longrightarrow \langle h; \texttt{done}\,\langle \mathit{mem\_int}\rangle{:}\mathit{ret}\rangle}$$

Memop_Is_PtrFromInt

$$\begin{array}{l}
1.\ \mathit{mem_ptr}\ \equiv\ \texttt{cast_ptr_to_int}\,\mathit{mem_int} \\
2.\ \mathit{ret}\ \equiv\ \Sigma\, y{:}\texttt{pointer}.\ y = \mathit{mem_ptr} \wedge \texttt{I}
\end{array}$$
$$\overline{\langle h; \texttt{ptrFromInt}\,(\tau_1, \tau_2, \mathit{mem\_int})\rangle \longrightarrow \langle h; \texttt{done}\,\langle \mathit{mem\_ptr}\rangle{:}\mathit{ret}\rangle}$$

Memop_Is_PtrValidForDeref

$$\begin{array}{l}
1.\ \langle h; \mathit{res_term}\rangle \Downarrow \langle h' + \{\mathit{pt}\ \&\ \texttt{None}\}; \texttt{Owned}\,\langle\tau\rangle(\mathit{term})\rangle \\
2.\ \mathit{pt}\ \equiv\ \mathit{term} \overset{\mathit{init}}{\mapsto}_{\tau}\ \mathit{value} \\
3.\ \mathit{bool_value}\ \equiv\ \texttt{aligned}\,(\tau, \mathit{term}) \\
4.\ \texttt{smt}\,(\cdot \Rightarrow (\mathit{term} = \mathit{mem_ptr}) \wedge (\mathit{init} = \texttt{const}_{\tau}\texttt{true})) \\
5.\ \mathit{ret}\ \equiv\ \Sigma\, y{:}\texttt{bool}.\ y = \texttt{aligned}\,(\tau, \mathit{pval}) \wedge (\mathit{mem_ptr} \overset{\texttt{const}_{\tau}\texttt{true}}{\mapsto}_{\tau}\ \mathit{value}) * \texttt{I}
\end{array}$$
$$\overline{\langle h; \texttt{ptrValidForDeref}\,(\tau, \mathit{mem\_ptr}, \mathit{res\_term})\rangle \longrightarrow \langle h' + \{\mathit{pt}\ \&\ \texttt{None}\}; \texttt{done}\,\langle \mathit{bool\_value}, \texttt{Owned}\,\langle\tau\rangle(\mathit{mem\_ptr})\rangle{:}\mathit{ret}\rangle}$$

Memop_Is_PtrWellAligned

$$\frac{\begin{array}{l}1.\ bool\_value \equiv \texttt{aligned}\,(\tau, mem\_ptr)\\ 2.\ ret \equiv \Sigma\, y{:}\texttt{bool}.\ y = bool\_value \wedge \texttt{I}\end{array}}{\langle h; \texttt{ptrWellAligned}\,(\tau, mem\_ptr)\rangle \longrightarrow \langle h; \texttt{done}\,\langle bool\_value\rangle{:}ret\rangle}$$

Memop_Is_PtrArrayShift

$$\frac{\begin{array}{l}1.\ mem\_ptr' \equiv mem\_ptr +_{\mathrm{ptr}} (mem\_int \times \mathrm{size\_of}(\tau))\\ 2.\ ret \equiv \Sigma\, y{:}\texttt{pointer}.\ y = mem\_ptr' \wedge \texttt{I}\end{array}}{\langle h; \texttt{ptrArrayShift}\,(mem\_ptr, \tau, mem\_int)\rangle \longrightarrow \langle h; \texttt{done}\,\langle mem\_ptr'\rangle{:}ret\rangle}$$

$\boxed{\langle h; is\_expr\rangle \longrightarrow \langle h'; is\_expr'\rangle}$

Is_Is_Memop

$$\frac{1.\ \langle h; memop\rangle \longrightarrow \langle h; tval{:}ret\rangle}{\langle h; \texttt{memop}\,(memop)\rangle \longrightarrow \langle h; tval{:}ret\rangle}$$

Is_Is_Action

$$\frac{1.\ \langle h; action\rangle \longrightarrow \langle h'; tval{:}ret\rangle}{\langle h;\ action\rangle \longrightarrow \langle h'; tval{:}ret\rangle}$$

Is_Is_Neg_Action

$$\frac{1.\ \langle h; action\rangle \longrightarrow \langle h'; tval{:}ret\rangle}{\langle h; \texttt{neg}\ action\rangle \longrightarrow \langle h'; tval{:}ret\rangle}$$

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

Seq_T_CCall

$$\frac{\begin{array}{l}1.\ ident{:}fun \equiv \overline{x_i}^{\,i} \mapsto texpr \in \texttt{Globals}\\ 2.\ \langle h; \overline{x_i = spine\_elem_i}^{\,i}\rangle :: fun \gg \langle h'; \sigma; ret\rangle\end{array}}{\langle h; \texttt{ccall}\,(\tau, ident, \overline{spine\_elem_i}^{\,i})\rangle \longrightarrow \langle h'; \sigma(texpr){:}ret\rangle}$$

Seq_T_Proc

$$\frac{\begin{array}{l}1.\ name{:}fun \equiv \overline{x_i}^{\,i} \mapsto texpr \in \texttt{Globals}\\ 2.\ \langle h; \overline{x_i = spine\_elem_i}^{\,i}\rangle :: fun \gg \langle h'; \sigma; ret\rangle\end{array}}{\langle h; \texttt{pcall}\,(name, \overline{spine\_elem_i}^{\,i})\rangle \longrightarrow \langle h'; \sigma(texpr){:}ret\rangle}$$

$\boxed{\langle h; seq\_texpr\rangle \longrightarrow \langle h'; texpr\rangle}$

TSeq_T_Run

$$\frac{\begin{array}{l}1.\ \mathit{ident}{:}\mathit{fun}\ \equiv\ \overline{x_i}^{\,i}\ \mapsto\ \mathit{texpr}\ \in\ \texttt{Globals}\\ 2.\ \langle h;\ \overline{x_i = \mathit{pval}_i}^{\,i}\ \rangle :: \mathit{fun} \gg \langle h';\sigma;\texttt{false}\wedge\texttt{I}\rangle\end{array}}{\langle h;\texttt{run}\,\mathit{ident}\,\overline{\mathit{pval}_i}^{\,i}\rangle \longrightarrow \langle h';\sigma(\mathit{texpr})\rangle}$$

TSeq_T_Case

$$\frac{\begin{array}{l}1.\ \mathit{pat}_j = \mathit{pval} \rightsquigarrow \sigma_j\\ 2.\ \forall\, i\ <\ j.\ \texttt{not}\,(\mathit{pat}_i = \mathit{pval} \rightsquigarrow \sigma_i)\end{array}}{\langle h;\texttt{case}\,\mathit{pval}\,\texttt{of}\,\overline{\mid\,\mathit{pat}_i \Rightarrow\ \mathit{texpr}_i}^{\,i}\ \texttt{end}\rangle \longrightarrow \langle h;\sigma_j(\mathit{texpr}_j)\rangle}$$

TSeq_T_LetP_Sub

$$\frac{1.\ \mathit{ident\_or\_pat} = \mathit{pval} \rightsquigarrow \sigma}{\langle h;\texttt{let}\,\mathit{ident\_or\_pat} = \mathit{pval}\,\texttt{in}\,\mathit{texpr}\rangle \longrightarrow \langle h;\sigma(\mathit{texpr})\rangle}$$

TSeq_T_LetP_LetP

$$\frac{1.\ \langle \mathit{pexpr}\rangle \longrightarrow \langle \mathit{tpval}{:}\mathit{pure\_ret}\rangle}{\langle h;\texttt{let}\,\mathit{ident\_or\_pat} = \mathit{pexpr}\,\texttt{in}\,\mathit{texpr}\rangle \longrightarrow \langle h;\texttt{let}\,\mathit{ident\_or\_pat}{:}\mathit{pure\_ret} = \mathit{tpval}\,\texttt{in}\,\mathit{texpr}\rangle}$$

TSeq_T_LetP_LetTP

$$\frac{1.\ \langle \mathit{pexpr}\rangle \longrightarrow \langle \mathit{tpexpr}{:}\mathit{pure\_ret}\rangle}{\langle h;\texttt{let}\,\mathit{ident\_or\_pat} = \mathit{pexpr}\,\texttt{in}\,\mathit{texpr}\rangle \longrightarrow \langle h;\texttt{let}\,\mathit{ident\_or\_pat}{:}\mathit{pure\_ret} = \mathit{tpexpr}\,\texttt{in}\,\mathit{texpr}\rangle}$$

TSeq_T_LetTP_Sub

$$\frac{1.\ \mathit{ident\_or\_pat} = \mathit{pval} \rightsquigarrow \sigma}{\langle h;\texttt{let}\,\mathit{ident\_or\_pat}{:}\mathit{pure\_ret} = \texttt{done}\,\mathit{pval}\,\texttt{in}\,\mathit{texpr}\rangle \longrightarrow \langle h;\sigma(\mathit{texpr})\rangle}$$

TSeq_T_LetTP_LetTP

$$\frac{1.\ \langle \mathit{tpexpr}\rangle \longrightarrow \langle \mathit{tpexpr}'\rangle}{\langle h;\texttt{let}\,\mathit{ident\_or\_pat}{:}\mathit{pure\_ret} = \mathit{tpexpr}\,\texttt{in}\,\mathit{texpr}\rangle \longrightarrow \langle h;\texttt{let}\,\mathit{ident\_or\_pat}{:}\mathit{pure\_ret} = \mathit{tpexpr}'\,\texttt{in}\,\mathit{texpr}\rangle}$$

TSeq_T_LetT_Sub

$$\frac{1.\ \langle h;\ \overline{ret\_pat_i = ret\_term_i}^{\,i}\ \rangle \rightsquigarrow \langle h'; \sigma\rangle}{\langle h; \texttt{let}\ \overline{ret\_pat_i}^{\,i}{:}ret = \texttt{done}\ \langle\overline{ret\_term_i}^{\,i}\rangle\ \texttt{in}\ texpr\rangle \longrightarrow \langle h'; \sigma(texpr)\rangle}$$

TSeq_T_Let_LetT

$$\frac{1.\ \langle h; seq\_expr\rangle \longrightarrow \langle h'; texpr_1{:}ret\rangle}{\langle h; \texttt{let}\ ret\_pat = seq\_expr\ \texttt{in}\ texpr_2\rangle \longrightarrow \langle h'; \texttt{let}\ ret\_pat{:}ret = texpr_1\ \texttt{in}\ texpr_2\rangle}$$

TSeq_T_LetT_LetT

$$\frac{1.\ \langle h; texpr_1\rangle \longrightarrow \langle h'; texpr'_1\rangle}{\langle h; \texttt{let}\ ret\_pat{:}ret = texpr_1\ \texttt{in}\ texpr_2\rangle \longrightarrow \langle h'; \texttt{let}\ ret\_pat{:}ret = texpr'_1\ \texttt{in}\ texpr_2\rangle}$$

TSeq_T_If_True

$$\frac{}{\langle h; \texttt{if True then}\ texpr_1\ \texttt{else}\ texpr_2\rangle \longrightarrow \langle h; texpr_1\rangle}$$

TSeq_T_If_False

$$\frac{}{\langle h; \texttt{if False then}\ texpr_1\ \texttt{else}\ texpr_2\rangle \longrightarrow \langle h; texpr_2\rangle}$$

TSeq_T_Bound

$$\frac{}{\langle h; \texttt{bound}\ [int](is\_texpr)\rangle \longrightarrow \langle h; is\_texpr\rangle}$$

$\boxed{\langle h; is\_texpr\rangle \longrightarrow \langle h'; texpr\rangle}$

TIs_T_LetS_Sub

$$\frac{1.\ \langle h;\ \overline{ret\_pat_i = ret\_term_i}^{\,i}\ \rangle \rightsquigarrow \langle h'; \sigma\rangle}{\langle h; \texttt{let strong}\ \overline{ret\_pat_i}^{\,i} = \texttt{done}\ \langle\overline{ret\_term_i}^{\,i}\rangle{:}ret\ \texttt{in}\ texpr\rangle \longrightarrow \langle h'; \sigma(texpr)\rangle}$$

TIs_T_LetS_LetS

$$\frac{1.\ \langle h; is\_expr\rangle \longrightarrow \langle h'; is\_expr'\rangle}{\langle h; \texttt{let strong}\ ret\_pat = is\_expr\ \texttt{in}\ texpr\rangle \longrightarrow \langle h'; \texttt{let strong}\ ret\_pat = is\_expr'\ \texttt{in}\ texpr\rangle}$$

$$\boxed{\langle h; texpr\rangle \longrightarrow \langle h'; texpr'\rangle}$$

T_T_TSeq_T

$$\frac{1.\ \langle h; seq\_texpr\rangle \longrightarrow \langle h; texpr\rangle}{\langle h; seq\_texpr\rangle \longrightarrow \langle h; texpr\rangle}$$

T_T_TIs_T

$$\frac{1.\ \langle h; is\_texpr\rangle \longrightarrow \langle h'; texpr\rangle}{\langle h; is\_texpr\rangle \longrightarrow \langle h'; texpr\rangle}$$

A6 Miscellaneous

$\boxed{\overline{x_i}^{\,i} :: \mathit{fun} \leadsto \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \mid \mathit{ret}}$ matching $\overline{x_i}^{\,i}$ and *fun* produces contexts $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}$ and return type *ret*

FUN_ENV_RET

$$\frac{}{::\mathit{ret} \leadsto \cdot; \cdot; \cdot; \cdot \mid \mathit{ret}}$$

FUN_ENV_COMP

$$\frac{1.\ \overline{x_i}^{\,i} :: \mathit{fun} \leadsto \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \mid \mathit{ret}}{x,\ \overline{x_i}^{\,i} :: \Pi\, x{:}\beta.\ \mathit{fun} \leadsto x{:}\beta, \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \mid \mathit{ret}}$$

FUN_ENV_LOG

$$\frac{1.\ \overline{x_i}^{\,i} :: \mathit{fun} \leadsto \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \mid \mathit{ret}}{x,\ \overline{x_i}^{\,i} :: \forall\, x{:}\beta.\ \mathit{fun} \leadsto \mathcal{C}; x{:}\beta, \mathcal{L}; \Phi; \mathcal{R} \mid \mathit{ret}}$$

FUN_ENV_PHI

$$\frac{1.\ \overline{x_i}^{\,i} :: \mathit{fun} \leadsto \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \mid \mathit{ret}}{\overline{x_i}^{\,i} :: \mathit{term} \supset \mathit{fun} \leadsto \mathcal{C}; \mathcal{L}; \mathit{term}, \Phi; \mathcal{R} \mid \mathit{ret}}$$

FUN_ENV_RES

$$\frac{1.\ \overline{x_i}^{\,i} :: \mathit{fun} \leadsto \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \mid \mathit{ret}}{x,\ \overline{x_i}^{\,i} :: \mathit{res} \twoheadrightarrow \mathit{fun} \leadsto \mathcal{C}; \mathcal{L}; \Phi; x{:}\mathit{res}, \mathcal{R} \mid \mathit{ret}}$$

$\boxed{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \sqsubseteq \mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'}$ context weakening: $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}$ is stronger than $\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'$

WEAK_EMPTY

$$\frac{}{\cdot; \cdot; \cdot; \cdot \sqsubseteq \cdot; \cdot; \cdot; \cdot}$$

WEAK_CONS_COMP

$$\frac{1.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \sqsubseteq \mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'}{\mathcal{C}, x{:}\beta; \mathcal{L}; \Phi; \mathcal{R} \sqsubseteq \mathcal{C}', x{:}\beta; \mathcal{L}'; \Phi'; \mathcal{R}'}$$

WEAK_CONS_LOG

$$\frac{1.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \sqsubseteq \mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'}{\mathcal{C}; \mathcal{L}, x{:}\beta; \Phi; \mathcal{R} \sqsubseteq \mathcal{C}'; \mathcal{L}', x{:}\beta; \Phi'; \mathcal{R}'}$$

WEAK_CONS_PHI

$$\frac{1.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \sqsubseteq \mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'}{\mathcal{C}; \mathcal{L}; \Phi, \mathit{term}; \mathcal{R} \sqsubseteq \mathcal{C}'; \mathcal{L}'; \Phi', \mathit{term}; \mathcal{R}'}$$

WEAK_CONS_RES

$$\frac{1.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \sqsubseteq \mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}, x{:}\mathit{res} \sqsubseteq \mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}', x{:}\mathit{res}}$$

WEAK_SKIP_COMP

$$\frac{1.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \sqsubseteq \mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \sqsubseteq \mathcal{C}', x{:}\beta; \mathcal{L}'; \Phi'; \mathcal{R}'}$$

WEAK_SKIP_LOG

$$\frac{1.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \sqsubseteq \mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \sqsubseteq \mathcal{C}'; \mathcal{L}', x{:}\beta; \Phi'; \mathcal{R}'}$$

WEAK_SKIP_PHI

$$\frac{1.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \sqsubseteq \mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \sqsubseteq \mathcal{C}'; \mathcal{L}'; \Phi', \mathit{term}; \mathcal{R}'}$$

$\boxed{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \sigma \Leftarrow (\mathcal{C}; \mathcal{L}; \mathcal{R})}$ 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.

$$\textsc{Subs\_Chk\_Empty}\quad \frac{}{\mathcal{C}; \mathcal{L}; \Phi; \cdot \vdash [\,] \Leftarrow (\cdot; \cdot; \cdot)}$$

$$\textsc{Subs\_Chk\_Comp}\quad \frac{1.\ \mathcal{C} \vdash \mathit{pval} \Rightarrow \beta}{\mathcal{C}; \mathcal{L}; \Phi; \cdot \vdash \mathit{pval}/x \Leftarrow (x{:}\beta; \cdot; \cdot)}$$

$$\textsc{Subs\_Chk\_Log}\quad \frac{1.\ \mathcal{C}; \mathcal{L} \vdash \mathit{term} \Rightarrow \beta}{\mathcal{C}; \mathcal{L}; \Phi; \cdot \vdash \mathit{term}/x \Leftarrow (\cdot; x{:}\beta; \cdot)}$$

$$\textsc{Subs\_Chk\_Res}\quad \frac{1.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term} \Leftarrow \mathit{res}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{res\_term}/x \Leftarrow (\cdot; \cdot; x{:}\mathit{res})}$$

$$\textsc{Subs\_Chk\_Concat}\quad \frac{\begin{array}{l}1.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}_2 \vdash \psi(\sigma) \Leftarrow (\mathcal{C}_2; \mathcal{L}_2; \psi(\mathcal{R}'_2))\\ 2.\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}_1 \vdash \psi \Leftarrow (\mathcal{C}_1; \mathcal{L}_1; \mathcal{R}'_1)\end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}_1, \mathcal{R}_2 \vdash [\psi, \sigma] \Leftarrow (\mathcal{C}_1, \mathcal{C}_2; \mathcal{L}_1, \mathcal{L}_2; \mathcal{R}'_1, \mathcal{R}'_2)}$$

$\boxed{\mathcal{C}; \mathcal{L}; \Phi \vdash h \Leftarrow \underline{\mathcal{R}}}$ heap typing: under context $\mathcal{C}; \mathcal{L}; \Phi$, heap h checks against context/type $\underline{\mathcal{R}}$

$$\textsc{Heap\_Empty}\quad \frac{}{\mathcal{C}; \mathcal{L}; \Phi \vdash \cdot \Leftarrow \cdot}$$

$$\textsc{Heap\_If}\quad \frac{1.\ \Phi \vdash \texttt{if}\ \mathit{term}\ \texttt{then}\ \mathit{res}_1\ \texttt{else}\ \mathit{res}_2 \ \equiv\ \texttt{if}\ \mathit{term}'\ \texttt{then}\ \mathit{res}'_1\ \texttt{else}\ \mathit{res}'_2}{\mathcal{C}; \mathcal{L}; \Phi \vdash \{\ \texttt{if}\ \mathit{term}\ \texttt{then}\ \mathit{res}_1\ \texttt{else}\ \mathit{res}_2\} \Leftarrow \_{:}\texttt{if}\ \mathit{term}'\ \texttt{then}\ \mathit{res}'_1\ \texttt{else}\ \mathit{res}'_2}$$

$$\textsc{Heap\_Pred\_Owned}\quad \frac{\begin{array}{l}1.\ \Phi \vdash \mathit{pred} \ \equiv\ \mathit{pred}'\\ 2.\ \mathit{pred} \ \equiv\ \mathit{ptr} \overset{\mathit{init}}{\mapsto}_{\tau} \mathit{value}\\ 3.\ \mathcal{C}; \mathcal{L} \vdash \mathit{init} \Rightarrow \mathit{bool}_{\tau}\\ 4.\ \mathcal{C}; \mathcal{L} \vdash \mathit{value} \Rightarrow \beta_{\tau}\end{array}}{\mathcal{C}; \mathcal{L}; \Phi \vdash \{\mathit{pred}\ \&\ \texttt{None}\} \Leftarrow \_{:}\mathit{pred}'}$$

Heap_Pred_Other

$$\frac{\begin{array}{l} 1.\ \Phi \vdash \mathit{pred\_term}(\mathit{oarg}) \equiv \mathit{pred\_term}'(\mathit{oarg}') \\ 2.\ \mathit{pred\_term} \equiv \alpha(\mathit{ptr}, \overline{\mathit{iarg}_i}^{\,i}) \\ 3.\ \alpha \neq \mathtt{Owned}\,\langle\tau\rangle \\ 4.\ \alpha \equiv x_p{:}\mathtt{pointer},\ \overline{x_i{:}\beta_i}^{\,i},\ y{:}\mathtt{record}\,\overline{\mathit{tag}_j{:}\beta'_j}^{\,j} \mapsto \mathit{res} \in \mathtt{Globals} \\ 5.\ \mathcal{C};\mathcal{L};\Phi;\underline{\mathcal{R}} \vdash \mathit{def} \Leftarrow [\mathit{oarg}/y, [\,\overline{\mathit{iarg}_i/x_i}^{\,i}\,], \mathit{ptr}/x_p](\mathit{res}) \\ 6.\ \mathcal{C};\mathcal{L};\Phi \vdash \mathit{heap} \Leftarrow \underline{\mathcal{R}} \end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \{\mathit{pred\_term}(\mathit{oarg})\ \&\ \mathit{def}\ \&\ \mathit{heap}\} \Leftarrow \_{:}\mathit{pred\_term}'(\mathit{oarg}')}$$

Heap_QPred_Owned

$$\frac{\begin{array}{l} 1.\ \Phi \vdash \mathit{qpred} \equiv \mathit{qpred}' \\ 2.\ \mathit{qpred} \equiv \circledast\, x.\ \mathit{iguard} \Rightarrow \mathit{ptr} + x \times \mathrm{size\_of}(\tau) \overset{\mathit{oarg}[x].\mathit{init}}{\mapsto_\tau} \mathit{oarg}[x].\mathit{value} \end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \{\mathit{qpred}\ \&\ \cdot\} \Leftarrow \_{:}\mathit{qpred}'}$$

Heap_QPred_Other

$$\frac{\begin{array}{l} 1.\ \Phi \vdash \mathit{qpred\_term}(\mathit{oarg}) \equiv \mathit{qpred\_term}'(\mathit{oarg}') \\ 2.\ \mathit{qpred\_term}' \equiv (x; \mathit{iguard})\{\alpha(\mathit{ptr} + x \times \mathit{step}, \mathit{iargs})\} \\ 3.\ \alpha \neq \mathtt{Owned}\,\langle\tau\rangle \\ 4.\ \forall\, x.\ \mathit{iguard} \Rightarrow \mathcal{C};\mathcal{L};\Phi \vdash \{\alpha(\mathit{ptr}, \mathit{iargs})(\mathit{oarg}[x])\ \&\ \mathit{arr\_def\_heap}[x]\} \Leftarrow \_{:}\alpha(\mathit{ptr}, \mathit{iargs})(\mathit{oarg}[x]) \end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \{\mathit{qpred\_term}(\mathit{oarg})\ \&\ \mathit{arr\_def\_heap}\} \Leftarrow \_{:}\mathit{qpred\_term}'(\mathit{oarg}')}$$

Heap_Concat

$$\frac{\begin{array}{l} 1.\ \mathcal{C};\mathcal{L};\Phi \vdash h \Leftarrow \underline{\mathcal{R}} \\ 2.\ \mathcal{C};\mathcal{L};\Phi \vdash h' \Leftarrow \underline{\mathcal{R}'} \end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash h + h' \Leftarrow \underline{\mathcal{R}}, \underline{\mathcal{R}'}}$$

$\boxed{\Phi \vdash h \Leftarrow \underline{\mathcal{R}}}$ heap typing: under context Φ, heap h checks against context/type $\underline{\mathcal{R}}$

Heap'_Aux

$$\frac{1.\ \cdot;\cdot;\Phi \vdash h \Leftarrow \underline{\mathcal{R}}}{\Phi \vdash h \Leftarrow \underline{\mathcal{R}}}$$

$\boxed{\Phi \vdash res \sim res'}$ *res* is related to *res'*

REL_RES_EMP

$$\frac{}{\Phi \vdash \texttt{emp} \sim \texttt{emp}}$$

REL_RES_PHI

$$\frac{1.\ term \sim term'}{\Phi \vdash term \sim term'}$$

REL_RES_IF

$$\frac{\begin{array}{l}1.\ term \sim term' \\ 2.\ \texttt{smt}\,(\Phi \Rightarrow term \leftrightarrow term') \\ 3.\ \Phi \vdash res_1 \sim res'_1 \\ 4.\ \Phi \vdash res_2 \sim res'_2\end{array}}{\Phi \vdash \texttt{if}\ term\ \texttt{then}\ res_1\ \texttt{else}\ res_2 \sim \texttt{if}\ term'\ \texttt{then}\ res'_1\ \texttt{else}\ res'_2}$$

REL_RES_EXISTS

$$\frac{1.\ \forall\, term \sim term'.\ \Phi \vdash term/y(res_1) \sim term'/y'(res'_1)}{\Phi \vdash \exists\, y{:}\beta.\ res_1 \sim \exists\, y'{:}\beta.\ res'_1}$$

REL_RES_SEPCONJ

$$\frac{\begin{array}{l}1.\ \Phi \vdash res_1 \sim res'_1 \\ 2.\ \Phi \vdash res_2 \sim res'_2\end{array}}{\Phi \vdash res_1 * res_2 \sim res'_1 * res'_2}$$

REL_RES_PRED

$$\frac{\begin{array}{l}1.\ ptr \sim ptr' \\ 2.\ \overline{iarg_i \sim iarg'_i}^{\,i} \\ 3.\ oarg \sim oarg'\end{array}}{\Phi \vdash \alpha(ptr, \overline{iarg_i}^{\,i})(oarg) \sim \alpha(ptr', \overline{iarg'_i}^{\,i})(oarg')}$$

REL_RES_QPRED

$$\frac{\begin{array}{l}1.\ iguard \sim iguard' \\ 2.\ ptr \sim ptr' \\ 3.\ \overline{iarg_i \sim iarg'_i}^{\,i} \\ 4.\ oarg \sim oarg'\end{array}}{\Phi \vdash (x; iguard)\{\alpha(ptr + x \times step, \overline{iarg_i}^{\,i})\}(oarg) \sim (x'; iguard')\{\alpha(ptr' + x' \times step, \overline{iarg'_i}^{\,i})\}(oarg')}$$

$\boxed{\Phi \vdash \mathit{fun} \sim \mathit{ret}}$ *fun* is related to *ret*

REL_RET_I

$$\frac{}{\Phi \vdash \mathtt{I} \sim \mathtt{I}}$$

REL_RET_COMP

$$\frac{1.\ \forall\, \mathit{term} \sim \mathit{pval}.\ \Phi \vdash \mathit{term}/y(\mathit{fun}) \sim \mathit{pval}/y'(\mathit{ret})}{\Phi \vdash \Pi\, y{:}\beta.\ \mathit{fun} \sim \Sigma\, y'{:}\beta.\ \mathit{ret}}$$

REL_RET_LOG

$$\frac{1.\ \forall\, \mathit{oarg} \sim \mathit{oarg}'.\ \Phi \vdash \mathit{oarg}/y(\mathit{fun}) \sim \mathit{oarg}'/y'(\mathit{ret})}{\Phi \vdash \forall\, y{:}\beta.\ \mathit{fun} \sim \exists\, y'{:}\beta.\ \mathit{ret}}$$

REL_RET_PHI

$$\frac{1.\ \mathit{term} \sim \mathit{term}' \quad 2.\ \Phi \vdash \mathit{fun} \sim \mathit{ret}}{\Phi \vdash \mathit{term} \supset \mathit{fun} \sim \mathit{term}' \wedge \mathit{ret}}$$

REL_RET_RES

$$\frac{1.\ \Phi \vdash \mathit{res} \sim \mathit{res}' \quad 2.\ \Phi \vdash \mathit{fun} \sim \mathit{ret}}{\Phi \vdash \mathit{res} \multimap \mathit{fun} \sim \mathit{res}' * \mathit{ret}}$$

$\boxed{[\![\mathit{spec}]\!](\mathit{opt\_ident}) = \mathit{res}}$ specification *spec* (with optional record *opt_ident*) represents resource *res* (*opt_ident* is present return when a return is expected, absent when it is not)

SPEC_RES_NONE

$$\frac{}{[\![\cdot]\!](\mathtt{None}) = \mathtt{emp}}$$

SPEC_RES_RETURN

$$\frac{}{[\![\mathtt{return}\,\{\,\overline{x_i = \mathit{term}_i}^{\,i}\,\};]\!](y) = (\bigwedge(\,\overline{y.x_i = \mathit{term}_i}^{\,i}\,))}$$

SPEC_RES_LETTERM

$$\frac{1.\ [\![\mathit{spec}]\!](\mathit{opt\_ident}) = \mathit{res}}{[\![\mathtt{let}\ y = \mathit{term};\mathit{spec}]\!](\mathit{opt\_ident}) = \mathit{term}/y(\mathit{res})}$$

SPEC_RES_ASSERT

$$\frac{1.\ [\![\mathit{spec}]\!](\mathit{opt\_ident}) = \mathit{res}}{[\![\mathtt{assert}\ (\mathit{term});\mathit{spec}]\!](\mathit{opt\_ident}) = \mathit{term} * \mathit{res}}$$

SPEC_RES_ENDIF

$$\frac{1.\ [\![\mathit{spec}_1]\!](\mathit{opt\_ident}) = \mathit{res}_1 \quad 2.\ [\![\mathit{spec}_2]\!](\mathit{opt\_ident}) = \mathit{res}_2}{[\![\mathtt{if}\ (\mathit{term})\{\mathit{spec}_1\}\ \mathtt{else}\ \{\mathit{spec}_2\}\cdot]\!](\mathit{opt\_ident}) = \mathtt{if}\ \mathit{term}\ \mathtt{then}\ \mathit{res}_1\ \mathtt{else}\ \mathit{res}_2}$$

Spec_Res_MiddleIf

1. $[\![spec_1]\!](\texttt{None}) = res_1$
2. $[\![spec_2]\!](\texttt{None}) = res_2$
3. $[\![spec_3]\!](opt\_ident) = res_3$

$$\frac{}{[\![\texttt{if}\ (term)\{spec_1\}\ \texttt{else}\ \{spec_2\} spec_3]\!](opt\_ident) = (\texttt{if}\ term\ \texttt{then}\ res_1\ \texttt{else}\ res_2) * res_3}$$

Spec_Res_LetPred

1. $\alpha \equiv \_{:}\texttt{pointer}, \overline{\_{:}\_{i}}^{\,i}, \_{:}\texttt{record}\ \overline{tag_j{:}\beta_j}^{\,j} \mapsto \_ \in \texttt{Globals}$
2. $[\![spec]\!](opt\_ident) = res$

$$\frac{}{[\![\texttt{let}\ y = \alpha(ptr, iargs); spec]\!](opt\_ident) = \exists\, y{:}\texttt{record}\ \overline{tag_j{:}\beta_j}^{\,j}.\ \alpha(ptr, iargs)(\,y\,) * res}$$

Spec_Res_LetQPred

1. $qpred\_term \equiv (x; iguard)\{\alpha(ptr + x{\times}step, iargs)\}$
2. $\alpha \equiv \_{:}\texttt{pointer}, \overline{\_{:}\_{i}}^{\,i}, \_{:}\texttt{record}\ \overline{tag_j{:}\beta_j}^{\,j} \mapsto \_ \in \texttt{Globals}$
3. $[\![spec]\!](opt\_ident) = res$

$$\frac{}{[\![\texttt{let}\ y = qpred\_term; spec]\!](opt\_ident) = \exists\, y{:}\texttt{array record}\ \overline{tag_j{:}\beta_j}^{\,j}.\ (x; iguard)\{\alpha(ptr + x{\times}step, iargs)\}(\,y\,) * res}$$

$\boxed{[\![spec]\!] = norm\_ret}$ specification *spec* represents normalised return type $norm\_ret$

Spec_NormRet_None

$$\frac{}{[\![\cdot]\!] = \texttt{I}}$$

Spec_NormRet_LetTerm

1. $[\![spec]\!] = \underline{ret}$

$$\frac{}{[\![\texttt{let}\ y = term; spec]\!] = term/y(\underline{ret})}$$

Spec_NormRet_Assert

1. $[\![spec]\!] = \underline{ret}$

$$\frac{}{[\![\texttt{assert}\ (term); spec]\!] = term \wedge \underline{ret}}$$

SPEC_NORMRET_IF

1. $[\![\texttt{if}\ (term)\{spec_1\}\ \texttt{else}\ \{spec_2\}\cdot]\!](\texttt{None}) = \underline{res}$
2. $[\![spec_3]\!] = \underline{ret}$

$$\frac{}{[\![\texttt{if}\ (term)\{spec_1\}\ \texttt{else}\ \{spec_2\}spec_3]\!] = \underline{res} * \underline{ret}}$$

SPEC_NORMRET_LETPRED

1. $\alpha \equiv \_{:}\texttt{pointer}, \overline{\_{:}\_i}^{\,i}, \_{:}\texttt{record}\ \overline{tag_j{:}\beta_j}^{\,j} \mapsto \_ \in \texttt{Globals}$
2. $[\![spec]\!] = \underline{ret}$

$$\frac{}{[\![\texttt{let}\ y = \alpha(ptr, iargs); spec]\!] = \exists\, y{:}\texttt{record}\ \overline{tag_j{:}\beta_j}^{\,j}.\ \alpha(ptr, iargs)(\,y\,) * \underline{ret}}$$

SPEC_NORMRET_LETQPRED

1. $qpred\_term \equiv (x; iguard)\{\alpha(ptr + x \times step, iargs)\}$
2. $\alpha \equiv \_{:}\texttt{pointer}, \overline{\_{:}\_i}^{\,i}, \_{:}\texttt{record}\ \overline{tag_j{:}\beta_j}^{\,j} \mapsto \_ \in \texttt{Globals}$
3. $[\![spec]\!] = \underline{ret}$

$$\frac{}{[\![\texttt{let}\ y = qpred\_term; spec]\!] = \exists\, y{:}\texttt{array record}\ \overline{tag_j{:}\beta_j}^{\,j}.\ (x; iguard)\{\alpha(ptr + x \times step, iargs)\}(\,y\,) * \underline{ret}}$$

$\boxed{[\![spec \mid \underline{ret}]\!] = norm\_fun}$ specification *spec* represents normalised argument type $norm\_fun$

SPEC_NORMARG_NONE

$$\frac{}{[\![\cdot \mid \underline{ret}]\!] = \underline{ret}}$$

SPEC_NORMARG_LETTERM

1. $[\![spec \mid \underline{ret}]\!] = \underline{fun}$

$$\frac{}{[\![\texttt{let}\ y = term; spec \mid \underline{ret}]\!] = term/y(\underline{fun})}$$

SPEC_NORMARG_ASSERT

1. $[\![spec \mid \underline{ret}]\!] = \underline{fun}$

$$\frac{}{[\![\texttt{assert}\ (term); spec \mid \underline{ret}]\!] = term \supset \underline{fun}}$$

SPEC_NORMARG_IF

1. $[\![\texttt{if}\ (term)\{spec_1\}\ \texttt{else}\ \{spec_2\}\cdot]\!](\texttt{None}) = \underline{res}$
2. $[\![spec_3 \mid \underline{ret}]\!] = \underline{fun}$

$$\frac{}{[\![\texttt{if}\ (term)\{spec_1\}\ \texttt{else}\ \{spec_2\}spec_3 \mid \underline{ret}]\!] = \underline{res} \mathrel{-\!\!*} \underline{fun}}$$

Spec_NormArg_LetPred

1. $\alpha \equiv \_\!:\!\mathtt{pointer},\ \overline{\_\!:\!\__i}^{\,i},\ \_\!:\!\mathtt{record}\ \overline{tag_j\!:\!\beta_j}^{\,j} \mapsto \_ \in \mathtt{Globals}$
2. $[\![spec \mid \underline{ret}]\!] = \underline{fun}$

$$[\![\mathtt{let}\ y = \alpha(ptr, iargs);\, spec \mid \underline{ret}]\!] = \forall\, y\!:\!\mathtt{record}\ \overline{tag_j\!:\!\beta_j}^{\,j}.\ \alpha(ptr, iargs)(\,y\,) \twoheadrightarrow \underline{fun}$$

Spec_NormArg_LetQPred

1. $qpred\_term \equiv (x; iguard)\{\alpha(ptr + x \times step, iargs)\}$
2. $\alpha \equiv \_\!:\!\mathtt{pointer},\ \overline{\_\!:\!\__i}^{\,i},\ \_\!:\!\mathtt{record}\ \overline{tag_j\!:\!\beta_j}^{\,j} \mapsto \_ \in \mathtt{Globals}$
3. $[\![spec \mid \underline{ret}]\!] = \underline{fun}$

$$[\![\mathtt{let}\ y = qpred\_term;\, spec \mid \underline{ret}]\!] = \forall\, y\!:\!\mathtt{array\ record}\ \overline{tag_j\!:\!\beta_j}^{\,j}.\ (x; iguard)\{\alpha(ptr + x \times step, iargs)\}(\,y\,) \twoheadrightarrow \underline{fun}$$

$\boxed{[\![\tau name(\,\overline{\tau_i\ x_i}^{\,i}\,)\ \mathtt{requires}\ spec_1\ \mathtt{ensures}\ spec_2]\!] = norm\_fun}$ user-defined C function specification represents normalised argument type $norm\_fun$

Spec_UserDef_CFunc_Base

1. $[\![spec_2]\!] = \underline{ret}$
2. $[\![spec_1 \mid \Sigma\, y\!:\!\beta_\tau.\ \underline{ret}]\!] = \underline{fun}$

$$[\![\tau name(\,)\ \mathtt{requires}\ spec_1\ \mathtt{ensures}\ spec_2]\!] = \underline{fun}$$

Spec_UserDef_CFunc_Arg

1. $[\![\tau name(\,\overline{\tau_i\ x_i}^{\,i}\,)\ \mathtt{requires}\ spec_1\ \mathtt{ensures}\ spec_2]\!] = \underline{fun}$

$$[\![\tau name(\tau_1\ x_1,\ \overline{\tau_i\ x_i}^{\,i}\,)\ \mathtt{requires}\ spec_1\ \mathtt{ensures}\ spec_2]\!] = \Pi\ x_1\!:\!\beta_{\tau_1}.\ \underline{fun}$$

$\boxed{[\![\,\mathtt{predicate}\,\{\,\overline{\beta'_j\ y_j}^{\,j}\,\}\alpha(\,\overline{\beta_i\ x_i}^{\,i}\,)\{spec\}]\!] = \alpha' \equiv \overline{r_k\!:\!\beta''_k}^{\,k} \mapsto res}$ user-defined resource predicate definition represents predicate

Spec_PredDef_Pred

1. $[\![spec]\!](y) = res$

$$[\![\,\mathtt{predicate}\,\{\,\overline{\beta'_j\ y_j}^{\,j}\,\}\alpha(\mathtt{pointer}\ x_p,\ \overline{\beta_i\ x_i}^{\,i}\,)\{spec\}]\!] = \alpha \equiv x_p\!:\!\mathtt{pointer},\ \overline{x_i\!:\!\beta_i}^{\,i},\ y\!:\!\mathtt{record}\ \overline{y_j\!:\!\beta'_j}^{\,j} \mapsto res$$

A7 Metvars and Grammar

ident, x, x_p, y, y_p, y_f, _, *abbrev*, r	subscripts: p for pointers, f for functions
n, i, j, k	index variables
impl_const	implementation-defined constant
member	C struct/union member name
	Ott-hack, ignore (annotations)
nat	OCaml arbitrary-width natural number
mem_ptr	abstract pointer value
mem_val	abstract memory value
	Ott-hack, ignore (locations)
mem_iv_c	OCaml type for memory constraints on integer values
UB_name	undefined behaviour
string, List	OCaml string
	Ott-hack, ignore (OCaml type variable TY)
$\mathbb{Q}$	OCaml type for rational numbers
	Ott-hack, ignore (OCaml Symbol.prefix)
mem_order, _	OCaml type for memory order
linux_mem_order	OCaml type for Linux memory order
	Ott-hack, ignore (OCaml type variable bt)

int, _, *step*	::=	OCaml fixed-width integer	
	|	i	
		literal integer	
	|	size_of(τ)	M
		size of a C type	

Sctypes_t, τ	::=	partial/relevant grammar of C types
	|	`array` *int* τ
		fixed-length array of element type τ
	|	`int`
		C (signed) integer
	|	$\tau *$
		pointer to type τ

	$\vert$	`struct` *tag*	
		C struct type	
$tag,\ init,\ value$	::=	OCaml type for struct/union tag	
	$\vert$	*ident*	
$\beta,\ \_$	::=	base types	
	$\vert$	`unit`	
		unit	
	$\vert$	`bool`	
		boolean	
	$\vert$	`integer`	
		integer	
	$\vert$	`real`	
		rational numbers?	
	$\vert$	`pointer`	
		location	
	$\vert$	`struct` *tag*	
		C structs	
	$\vert$	`record` $\overline{tag_i{:}\beta_i}^{\,i}$	
		res. pred. output arguments	
	$\vert$	`map` $\beta\ \beta'$	
		map	
	$\vert$	`array` β	M
		array (integer-indexed map)	
	$\vert$	`list` β	
		list	
	$\vert$	$\overline{\beta_i}^{\,i}$	
		tuple	
	$\vert$	`set` β	
		set	
	$\vert$	$bool_\tau$	M
		boolean from C type	

| | β_τ M
of a C type

binop ::= binary operators
| `+`
addition
| `-`
subtraction
| `*`
multiplication
| `/`
division
| `mod`
modulus
| `rem`
remainder
| `^`
exponentiation
| `=`
equality, defined both for integer and C types
| `!=`
inequality, similiarly defined
| `>`
greater than, similarly defined
| `<`
less than, similarly defined
| `>=`
greater than or equal to, similarly defined
| `<=`
less than or equal to, similarly defined
| `/\`
conjucntion
| `\/`

disjunction

$binop_{arith}$ ::= arithmentic binary operators
| `+`
| `-`
| `*`
| `/`
| `mod`
| `rem`
| `^`

$binop_{rel}$ ::= relational binary operators
| `=`
| `!=`
| `>`
| `<`
| `>=`
| `<=`

$binop_{bool}$ ::= boolean binary operators
| `/\`
| `\/`

mem_int ::= memory integer value
| 1 M
| 0 M

object_value ::= C object values (inhabitants of object types), which can be read/stored
| *mem_int*
integer value
| *mem_ptr*
pointer value

	|	$\texttt{array}\,(\,\overline{\mathit{loaded\_value}_i}^{\,i}\,)$
		C array value
	|	$(\,\texttt{struct}\,\mathit{ident})\{\overline{.\mathit{member}_i{:}\tau_i = \mathit{mem\_val}_i}^{\,i}\,\}$
		C struct value
	|	$(\,\texttt{union}\,\mathit{ident})\{.\mathit{member} = \mathit{mem\_val}\}$
		C union value
loaded_value	::=	potentially unspecified C object values
	|	$\texttt{specified}\,\mathit{object\_value}$
		specified loaded value
value	::=	Core values
	|	*object_value*
		C object value
	|	*loaded_value*
		loaded C object value
	|	`Unit`
		unit
	|	`True`
		boolean true
	|	`False`
		boolean false
	|	$\beta[\,\overline{\mathit{value}_i}^{\,i}\,]$
		list
	|	$(\,\overline{\mathit{value}_i}^{\,i}\,)$
		tuple
bool_value	::=	Core booleans
	|	`True`
		boolean true
	|	`False`
		boolean false

ctor_val ::= data constructors (values, do not reduce)
| `Nil` β
empty list
| `Cons`
list cons
| `Tuple`
tuple
| `Array`
C fixed-size array (guaranteed to be non-empty)
| `Specified`
non-unspecified loaded value

ctor_expr ::= data constructors (expressions, do reduce)
| `IvCOMPL`
bitwise complement
| `IvAND`
bitwise AND
| `IvOR`
bitwise OR
| `IvXOR`
bitwise XOR
| `Fvfromint`
cast integer to floating value
| `Ivfromfloat`
cast floating to integer value

name ::=
| *ident*
Core identifier
| *impl_const*
implementation-defined constant

pval ::= pure values

	|	$ident$	
		Core identifier	
	|	$impl\_const$	
		implementation-defined constant	
	|	$value$	
		Core values	
	|	$\texttt{constrained}\,(\overline{mem\_iv\_c_i, pval_i}^{\,i})$	
		constrained value	
	|	$ctor\_val(\overline{pval_i}^{\,i})$	
		data constructor application	
	|	$(\texttt{struct}\,ident)\{\overline{.member_i = pval_i}^{\,i}\}$	
		C struct expression	
	|	$(\texttt{union}\,ident)\{.member = pval\}$	
		C union expression	
	|	$\sigma(pval)$	M
		substitution for pure values	
$pvals$	::=	list of pure values	
	|	$\overline{pval_i}^{\,i}$	
	|	$\sigma(pvals)$	M
$tpval$	::=	top-level pure values	
	|	$\texttt{done}\,pval$	
		pure done	
	|	$\texttt{undef}\ UB\_name$	
		undefined behaviour	
	|	$\texttt{error}\,(string, pval)$	
		impl-defined static error	
$ident\_opt\_\beta$	::=	type annotated optional identifier	
	|	$\_{:}\beta$	binders $= \{\}$
	|	$ident{:}\beta$	binders $= ident$

pat	::=	computational patterns	
	|	$ident\_opt\_\beta$	binders = binders($ident\_opt\_\beta$)
	|	$ctor\_val(\overline{pat_i}^{i})$	binders = binders($\overline{pat_i}^{i}$)
ident_or_pat	::=	identifier or pattern	
	|	*ident*	binders = *ident*
	|	*pat*	binders = binders(*pat*)
z	::=	OCaml arbitrary-width integer	
	|	i	M
		literal integer	
	|	*int*	M
	|	*mem_int*	M
		convert *mem_int* to an integer	
	|	*mem_ptr*	M
		convert *mem_ptr* to an ptreger	
	|	$\text{offset\_of}_{tag}(member)$	M
		offset of a struct member	
	|	`ptr_size`	M
		size of a pointer	
	|	max_int_{τ}	M
		maximum value of int of type τ	
	|	min_int_{τ}	M
		minimum value of int of type τ	
bool, _	::=	OCaml booleans	
	|	`true`	
	|	`false`	
	|	$bool \|\| bool'$	M
lit	::=		
	|	*ident*	

	|	z	
	|	z	
	|	$\mathbb{Q}$	
	|	$bool$	
	|	`unit`	
	|	`default` β	
	|	`null`	
$arith\_op$	::=	SMT term arithmetic operations	
	|	$term_1 + term_2$	
	|	$term_1 - term_2$	
	|	$term_1 \times term_2$	
	|	$term_1 / term_2$	
	|	$term_1 \ \mathrm{mod}\ term_2$	
	|	$term_1 \ \mathrm{rem}\ term_2$	
	|	$term_1 \uparrow term_2$	
	|	$term_1\ binop_{arith}\ term_2$	M
$bool\_op$	::=	SMT term boolean operations	
	|	$\bigwedge(\overline{term_i}^{\,i})$	
	|	$\bigvee(\overline{term_i}^{\,i})$	
	|	$term_1 \rightarrow term_2$	
	|	$term_1 \leftrightarrow term_2$	M
	|	$\neg\, term$	
	|	`if` $term_1$ `then` $term_2$ `else` $term_3$	
	|	$term_1 = term_2$	
	|	$term_1 \neq term_2$	M
	|	$term_1\ binop_{bool}\ term_2$	M
$tuple\_op$	::=	SMT term tuple constructor and projections	
	|	$(\overline{term_i}^{\,i})$	
	|	$term^{(int)}$	

struct_op	::=	SMT term for struct field-projection	
	|	$\mathit{term}.\mathit{member}$	
record_op	::=	SMT term for record operations	
	|	$\{\,\overline{\mathit{ident}_i = \mathit{term}_i}^{\,i}\,\}$	
	|	$\mathit{term}.\mathit{ident}$	
pointer_op	::=	SMT term pointer operations	
	|	$\mathit{term}_1 +_{\mathrm{ptr}} \mathit{term}_2$	
	|	`cast_int_to_ptr` *term*	
	|	`cast_ptr_to_int` *term*	
list_op	::=	SMT term list constructors and operations	
	|	`nil`	
	|	$\mathit{term}_1 :: \mathit{term}_2$	
	|	`tl` *term*	
	|	$\mathit{term}^{(\mathit{int})}$	
ct_pred	::=	SMT predicates for C-types	
	|	`representable` (τ, term)	
	|	`aligned` (τ, term)	
	|	`alignedI` $(\mathit{term}_1, \mathit{term}_2)$	
cmp_op	::=	SMT term relational operations	
	|	$\mathit{term}_1 < \mathit{term}_2$	
	|	$\mathit{term}_1 \leq \mathit{term}_2$	
	|	$\mathit{term}_1\ \mathit{binop}_{\mathit{rel}}\ \mathit{term}_2$	M
map_op	::=	SMT term map operations	
	|	$[\|\ \overline{\mathit{term}_i}^{\,i}\ \|]$	M
		array literal	
	|	$\mathit{term}_1[\mathit{term}_2]$	

	|	$\texttt{const}\ term$	
	|	$term_1[term_2] := term_3$	
	|	$ident{:}\beta.\ term$	
$term,\ iguard,\ ptr,\ init,\ \_,\ value,\ iarg,\ oarg$	::=	SMT term grammar	
	|	lit	
	|	$arith\_op$	
	|	$bool\_op$	
	|	cmp_op	
	|	$tuple\_op$	
	|	$struct\_op$	
	|	$record\_op$	
	|	$pointer\_op$	
	|	$list\_op$	
	|	ct_pred	
	|	map_op	
	|	$string(term_1, .., term_n)$	
	|	$(term)$	S
		parentheses	
	|	$\sigma(term)$	M
		substitute σ in *term*	
	|	$pvals$	M
		tranlate pure values *pvals* into corresponding SMT term	
	|	$\texttt{const}_\tau bool$	M
		term with structure corresponding to τ	
$iargs,\ \_$	::=	list of terms (predicate input-arguments)	
	|	$\overline{iarg_i}^{\,i}$	
	|	$\sigma(iargs)$	M
$qterm$	::=	quantified SMT terms	
	|	$term$	
		unquantified SMT term	

	$\mid$	$\forall\, ident.\, term$	
		universally quantified SMT term	
	$\mid$	$\exists\, ident.\, term$	
		existentially quantified SMT term	
	$\mid$	$\sigma(qterm)$	M
		substitute σ into *qterm*	
$pred\_term$	::=	predicate term/request	
	$\mid$	$\alpha(ptr, iargs)$	
		first parameter must be a pointer	
$qpred\_term$	::=	quantifed predicate term/request	
	$\mid$	$(x; iguard)\{\alpha(ptr + x \times step, iargs)\}$	bind x in $iargs$
		iguard , *ptr* and *step* must be specified	
	$\mid$	`each` $qpred\_term$	S
res_req	::=	resource request	
	$\mid$	$pred\_term$	
		request a resource predicate	
	$\mid$	$qpred\_term$	
		request a quantified resource predicate	
$pred\_name,\ \alpha$	::=	names of predicates	
	$\mid$	`Owned` $\langle\tau\rangle$	
		sep. logic points-to indexed by C type τ	
	$\mid$	$string$	
		user-defined name	
$pred,\ points\_to,\ pt$	::=	*precise* seperation-logic predicates	
	$\mid$	$pred\_term(oarg)$	
		a predicate-type is simply the term with an output argument	
	$\mid$	$ptr \overset{init}{\mapsto}_{\tau}\ value$	S

pretty-printing for points-to predicate $\texttt{Owned}\langle\tau\rangle(ptr,)\&\{init, value\}$

$qpred,\ qpoints\_to,\ qpt$	::=	quantified (integer-indexed) seperation logic predicate	
	|	$qpred\_term(oarg)$ a qpredicate-type is simply the term with an array output argument	
	|	$\circledast\, x.\ iguard \Rightarrow ptr + x{\times}\text{size\_of}(\tau) \overset{oarg[x].init}{\mapsto_\tau} oarg[x].value$ pretty-printing for quantified points-to predicate $\circledast\ x.\ iguard \Rightarrow \texttt{Owned}\langle\tau\rangle(ptr + x{\times}\text{size\_of}(\tau), oarg[x])$	S
$res,\ \_$	::=	resources	
	|	$\texttt{emp}$ empty heap	
	|	$term$ logical assertion, implicitly with emp	
	|	$pred$ heap predicate	
	|	$qpred$ quantified (integer-indexed) heap predicate	
	|	$res_1 * res_2$ seperating conjunction	
	|	$\circledast\,(\overline{res_i}^{\,i})$ notation for nested sep. conj.	M
	|	$\exists\, ident{:}\beta.\ res$ existential	
	|	$\texttt{if}\ term\ \texttt{then}\ res_1\ \texttt{else}\ res_2$ conditional resource / ordered disjuction	
	|	$\sigma(res)$ substitute σ in res	M
$\underline{res},\ rem$	::=	normalised resources	

	$\mid$	`if` term `then` res_1 `else` res_2
		conditional resource / ordered disjuction
	$\mid$	pred
		heap predicate
	$\mid$	qpred
		quantified (integer-indexed) heap predicate
$\mathit{opt\_res}$	::=	optional resource
	$\mid$	`None`
	$\mid$	res
ret, _	::=	return types
	$\mid$	$\Sigma\, \mathit{ident}{:}\beta.\ \mathit{ret}$
		return a computational value
	$\mid$	$\exists\, \mathit{ident}{:}\beta.\ \mathit{ret}$
		return a logical (output) value
	$\mid$	$\mathit{res} * \mathit{ret}$
		return a resource
	$\mid$	$\mathit{term} \wedge \mathit{ret}$
		guarantee a constraint (post-condition)
	$\mid$	`I`
		end return list
	$\mid$	$\sigma(\mathit{ret})$ M
		substitute σ in ret
$\mathit{pure\_ret}$	::=	pure return types
	$\mid$	$\Sigma\, \mathit{ident}{:}\beta.\ \mathit{pure\_ret}$
	$\mid$	$\mathit{term} \wedge \mathit{pure\_ret}$
	$\mid$	`I`
	$\mid$	$\sigma(\mathit{pure\_ret})$ M
		substitute σ in $\mathit{pure\_ret}$

$\underline{ret}$	::=	normalised return types	
	|	$\Sigma\, ident{:}\beta.\ \underline{ret}$	
	|	$\exists\, ident{:}\beta.\ \underline{ret}$	
	|	$\underline{res} * \underline{ret}$	
	|	$term \wedge \underline{ret}$	
	|	$\mathtt{I}$	
	|	$\sigma(\underline{ret})$	M

$pexpr$	::=	pure expressions
	|	$pval$ pure values
	|	$ctor\_expr(\,\overline{pval_i}^{\,i}\,)$ data constructor application
	|	$\mathtt{array\_shift}\,(pval_1, \tau, pval_2)$ pointer array shift
	|	$\mathtt{member\_shift}\,(pval, ident, member)$ pointer struct/union member shift
	|	$\mathtt{not}\,(pval)$ boolean not
	|	$pval_1\ binop\ pval_2$ binary operations
	|	$\mathtt{memberof}\,(ident, member, pval)$ C struct/union member access
	|	$name(\,\overline{pval_i}^{\,i}\,)$ pure function call
	|	$\mathtt{assert\_undef}\,(pval,\ UB\_name)$ if $pval$ then UB for reason UB_name
	|	$\mathtt{bool\_to\_integer}\,(pval)$ convert boolean $pval$ to integer
	|	$\mathtt{conv\_int}\,(\tau, pval)$ convert between different integer types
	|	$\mathtt{wrapI}\,(\tau, pval)$ wrap integer

	$\mid$	$\sigma(pexpr)$ substitution for pure expressions	M
tpexpr	::=	top-level pure expressions	
	$\mid$	*tpval* top-level pure values	
	$\mid$	$\texttt{case}\ pval\ \texttt{of}\ \overline{\mid\ tpexpr\_case\_branch_i}^{\,i}\ \texttt{end}$ pat matching	
	$\mid$	$\texttt{let}\ ident\_or\_pat = pexpr\ \texttt{in}\ tpexpr$ pure let	bind binders(*ident_or_pat*) in *tpexpr*
	$\mid$	$\texttt{let}\ ident\_or\_pat{:}pure\_ret = tpexpr_1\ \texttt{in}\ tpexpr_2$ annoted pure let	bind binders(*ident_or_pat*) in $tpexpr_2$
	$\mid$	$\texttt{if}\ pval\ \texttt{then}\ tpexpr_1\ \texttt{else}\ tpexpr_2$ pure if	
	$\mid$	$\sigma(tpexpr)$ substitute σ in *tpexpr*	M
tpexpr_case_branch	::=	pure top-level case expression branch	
	$\mid$	$pat \Rightarrow tpexpr$ top-level case expression branch	bind binders(*pat*) in *tpexpr*
m_kill_kind	::=		
	$\mid$	`dynamic`	
	$\mid$	$\texttt{static}\ \tau$	
pred_ops	::=	(q)points-to operation terms	
	$\mid$	$\texttt{iterate}\ (res\_term, int)$ transform points-to-array into quantified points-to	
	$\mid$	$\texttt{congeal}\ (res\_term, int)$ transform quantified points-to into points-to-array	
	$\mid$	$\texttt{explode}\ (res\_term)$ transform points-to-struct into member points-tos	

	|	`implode` (res_term, tag)
		transform member points-tos into points-to-struct
	|	`break` $(res\_term, term)$
		break a qpred into a qpred and a pred
	|	`glue` (res_term)
		glue a qpred and a pred (back) into a qpred
	|	`inj` $(res\_term, ptr, step, x.\ iargs)$
		transform a pred into a singleton qpred
	|	`split` $(res\_term, iguard)$
		split a qpred into two qpreds along *iguard*
res_term, $\_$	::=	resource terms
	|	`emp`
		empty heap
	|	`term`
		term for assertion
	|	$pred\_term$
		heap predicate
	|	$qpred\_term$
		quanitfied (integer-indexed) heap predicate
	|	$ident$
		variable
	|	$\langle\, \overline{res\_term_i}^{\,i}\, \rangle$
		(nested) seperating-conjunction pair
	|	`pack` $(oarg, res\_term)$
		packing for existentials
	|	`fold` $res\_term{:}pred$
		fold into recursive res. pred.
	|	$pred\_ops$
		(q)predicate operation terms
	|	(res_term) S
		parentheses
	|	$\sigma(res\_term)$ M

substitution for resource terms

res_val, *def*	::=	resource terms values	
	|	`emp`	
		empty heap	
	|	`term`	
		term for assertion	
	|	*pred_term*	
		heap predicate	
	|	*qpred_term*	
		quanitfied (integer-indexed) heap predicate	
	|	$\langle \overline{res\_val_i}^{\,i} \rangle$	
		(nested) seperating-conjunction pair	
	|	`pack` $(oarg, res\_val)$	
		packing for existentials	
	|	(res_val)	S
		parentheses	
	|	$\sigma(res\_val)$	M
		substitution for resource terms	
action	::=	memory actions	
	|	`create` $(pval, \tau)$	
	|	`create_readonly` $(pval_1, \tau, pval_2)$	
	|	`alloc` $(pval_1, pval_2)$	
	|	`kill` $(m\_kill\_kind, pval, res\_term)$	
	|	`store` $(bool, \tau, pval_1, pval_2, mem\_order, res\_term)$	
		true means store is locking	
	|	`load` $(\tau, pval, mem\_order, res\_term)$	
	|	`rmw` $(\tau, pval_1, pval_2, pval_3, mem\_order_1, mem\_order_2)$	
	|	`fence` (mem_order)	
	|	`cmp_exch_strong` $(\tau, pval_1, pval_2, pval_3, mem\_order_1, mem\_order_2)$	
	|	`cmp_exch_weak` $(\tau, pval_1, pval_2, pval_3, mem\_order_1, mem\_order_2)$	
	|	`linux_fence` $(linux\_mem\_order)$	

	|	`linux_load` $(\tau, pval, linux\_mem\_order)$
	|	`linux_store` $(\tau, pval_1, pval_2, linux\_mem\_order)$
	|	`linux_rmw` $(\tau, pval_1, pval_2, linux\_mem\_order)$
polarity	::=	polarities for memory actions
	|	
		(pos) sequenced by `let weak` and `let strong`
	|	`neg`
		only sequenced by `let strong`
pol_mem_action	::=	memory actions with polarity
	|	*polarity action*
memop	::=	operations involving the memory state
	|	$pval_1\ binop_{rel}\ pval_2$
		pointer relational binary operations
	|	$pval_1 -_{\tau} pval_2$
		pointer subtraction
	|	`intFromPtr` $(\tau_1, \tau_2, pval)$
		cast pointer value to integer value
	|	`ptrFromInt` $(\tau_1, \tau_2, pval)$
		cast integer value to pointer value
	|	`ptrValidForDeref` $(\tau, pval, res\_term)$
		dereferencing validity predicate
	|	`ptrWellAligned` $(\tau, pval)$
	|	`ptrArrayShift` $(pval_1, \tau, pval_2)$
	|	`memcpy` $(pval_1, pval_2, pval_3)$
	|	`memcmp` $(pval_1, pval_2, pval_3)$
	|	`realloc` $(pval_1, pval_2, pval_3)$
	|	`va_start` $(pval_1, pval_2)$
	|	`va_copy` $(pval)$
	|	`va_arg` $(pval, \tau)$
	|	`va_end` $(pval)$

$ret\_term,\ spine\_elem$	::=	return values / spine element	
	|	$pval$	
		pure computational value	
	|	$oarg$	
		logical value	
	|	res_term	
		resource term	
	|	$\sigma(ret\_term)$	M
		substitution for return values / spine elements	
$ret\_terms,\ spine$	::=	return values / spine	
	|	ret_term, ret_terms	M
	|	$\overline{ret\_term_i}^{\,i}$	
$tval$	::=	(effectful) top-level values	
	|	`done` $\langle ret\_terms\rangle$	
		end of top-level expression	
	|	`undef` UB_name	
		undefined behaviour	
	|	`error` $(string, pval)$	
		impl-defined static error	
	|	$\sigma(tval)$	M
		substitution for top-level values	
res_pat	::=	resource patterns	
	|	`emp`	binders = {}
		empty heap	
	|	`term`	binders = {}
		logical assertion token	
	|	$ident$	binders = $ident$
		variable	
	|	`fold` (res_pat)	binders = {}
		unfold (recursive) predicate	

	$\mid$	$\langle res\_pat_1, res\_pat_2 \rangle$	binders = binders(res_pat_1) $\cup$ binders(res_pat_2)
		seperating-conjunction pair	
	$\mid$	$\texttt{pack}\,(ident, res\_pat)$	binders = $ident \cup$ binders(res_pat)
		packing for existentials	
ret_pat	::=	return pat	
	$\mid$	$\texttt{comp}\, ident\_or\_pat$	binders = binders($ident\_or\_pat$)
		computational pattern	
	$\mid$	$\texttt{log}\, ident$	binders = $ident$
		logical variable	
	$\mid$	$\texttt{res}\, res\_pat$	binders = binders(res_pat)
		resource pattern	
	$\mid$	$\overline{ret\_pat_i}^{\,i}$	binders = binders($\overline{ret\_pat_i}^{\,i}$)
		sequence of return patterns	
seq_expr	::=	sequential (effectful) expressions	
	$\mid$	$\texttt{ccall}\,(\tau, ident, spine)$	
		C function call	
	$\mid$	$\texttt{pcall}\,(name, spine)$	
		procedure call	
	$\mid$	$\sigma(seq\_expr)$	M
seq_texpr	::=	sequential top-level (effectful) expressions	
	$\mid$	$tval$	
		(effectful) top-level values	
	$\mid$	$\texttt{run}\, ident\, \overline{pval_i}^{\,i}$	
		run from label	
	$\mid$	$\texttt{let}\, ident\_or\_pat = pexpr\ \texttt{in}\ texpr$	bind binders($ident\_or\_pat$) in $texpr$
		pure let	
	$\mid$	$\texttt{let}\, ident\_or\_pat{:}pure\_ret = tpexpr\ \texttt{in}\ texpr$	bind binders($ident\_or\_pat$) in $texpr$
		annotated pure let	
	$\mid$	$\texttt{let}\, ret\_pat = seq\_expr\ \texttt{in}\ texpr$	bind binders(ret_pat) in $texpr$
		bind return pats	

	|	let ret_pat:ret = $texpr_1$ in $texpr_2$	bind binders(ret_pat) in $texpr_2$
		annotated bind return pats	
	|	case $pval$ of $\overline{\mid texpr\_case\_branch_i}^{i}$ end	
		pat matching	
	|	if $pval$ then $texpr_1$ else $texpr_2$	
		conditional	
	|	bound [int](is_texpr)	
		limit scope of indet seq behaviour, absent at runtime	
	|	insert_lets (res_bind, seq_texpr)	M
		insert let expressions for binding resources	
$texpr\_case\_branch$	::=	top-level case expression branch	
	|	$pat \Rightarrow texpr$	bind binders(pat) in $texpr$
		top-level case expression branch	
is_expr	::=	indet seq (effectful) expressions	
	|	$tval$:ret	
		(effectful) top-level values	
	|	memop ($memop$)	
		pointer op involving memory	
	|	pol_mem_action	
		memory action	
	|	pack α($pval$, $pvals$)	
		fold a predicate	
	|	unpack α($pval$, $pvals$)	
		unfold a predicate	
is_texpr	::=	indet seq top-level (effectful) expressions	
	|	let weak ret_pat = is_expr in $texpr$	bind binders(ret_pat) in $texpr$
		weak sequencing	
	|	let strong ret_pat = is_expr in $texpr$	bind binders(ret_pat) in $texpr$
		strong sequencing	

texpr	::=	top-level (effectful) expressions	
	|	*seq_texpr*	
		sequential (effectful) expressions	
	|	*is_texpr*	
		indet seq (effectful) expressions	
	|	`insert_lets` (*res_bind*, *texpr*)	M
		insert let expressions for binding resources	
	|	$\sigma(texpr)$	M
		substitute σ in *texpr*	
fun	::=	function types	
	|	$\Pi\, ident{:}\beta.\, fun$	
		assume a computational value	
	|	$\forall\, ident{:}\beta.\, fun$	
		assume a logical value	
	|	$res \twoheadrightarrow fun$	
		assume a resource	
	|	$term \supset fun$	
		assume a constraint (pre-condition)	
	|	*ret*	
		return a value of type *ret*	
	|	`to_fun` *ret*	M
		change a return to an arugment type	
	|	$\sigma(fun)$	M
		substitute σ in *fun*	
pure_fun	::=	pure function types	
	|	$\Pi\, ident{:}\beta.\, pure\_fun$	
	|	$term \supset pure\_fun$	
	|	*pure_ret*	
$\underline{fun}$	::=	normalised function types	
	|	$\Pi\, ident{:}\beta.\, \underline{fun}$	

		assume a computational value	
	|	$\forall\, \mathit{ident}{:}\beta.\ \underline{\mathit{fun}}$	
		assume a logical value	
	|	$\underline{\mathit{res}} \multimap \underline{\mathit{fun}}$	
		assume a resource	
	|	$\mathit{term} \supset \underline{\mathit{fun}}$	
		assume a constraint (pre-condition)	
	|	$\underline{\mathit{ret}}$	
		return a value of type $\underline{\mathit{ret}}$	
	|	`to_fun` $\underline{\mathit{ret}}$	M
		change a return to an arugment type	
	|	$\sigma(\underline{\mathit{fun}})$	M
		substitute σ in $\underline{\mathit{fun}}$	
$\sigma,\ \psi$	::=	substitutions	
	|	$\mathit{ret\_term}/\mathit{ident}$	
		sub $\mathit{ret\_term}$ for ident	
	|	$[\,\overline{\sigma_i}^{\,i}\,]$	
		sequential substitutions	
	|	$\cdot$	M
		empty substitution	
	|	$\sigma(\psi)$	M
		apply σ to all elements in ψ	
$\mathit{opt\_ident}$	::=	optional identifier	
	|	`None`	
	|	ident	
$\mathit{spec\_expr}$	::=	expressions for specifications	
	|	term	
	|	$\mathit{pred\_term}$	
	|	$\mathit{qpred\_term}$	

spec	::=	alternative, C-programmer friendly syntax for defining predicates and writing specifications
	|	$\cdot$ empty specification
	|	$\texttt{return}\,\{\,\overline{ident_i = term_i}^{\,i}\,\};$ specify output arguments
	|	$\texttt{let}\,ident = spec\_expr; spec$ bind either terms, or output arguments of resource (q)predicates
	|	$\texttt{assert}\,(term); spec$ assert specification
	|	$\texttt{if}\,(term)\{spec_1\}\,\texttt{else}\,\{spec_2\}spec_3$ conditional specification
user_def	::=	syntax for user-defined predicates and function specifications (pre- and post-conditions)
	|	$\texttt{predicate}\,\{\,\overline{\beta'_j\,tag_j}^{\,j}\,\}\alpha(\,\overline{\beta_i\,x_i}^{\,i}\,)\{spec\}$
	|	$\tau name(\,\overline{\tau_i\,x_i}^{\,i}\,)\,\texttt{requires}\,spec_1\,\texttt{ensures}\,spec_2$
$\mathcal{C}$	::=	computational variable context
	|	$ident{:}\beta$ add to context
	|	$\overline{\mathcal{C}_i}^{\,i}$ concatenate contexts
	|	$\cdot$ M empty context
$\mathcal{L}$	::=	logical variable context
	|	$ident{:}\beta$ add to context
	|	$\overline{\mathcal{L}_i}^{\,i}$ concatenate contexts
	|	$\cdot$ M empty context

Φ	::=	constraints environment	
	|	$term$	
		add to context	
	|	$\overline{\Phi_i}^{\,i}$	
		concatenate contexts	
	|	$\cdot$	M
		empty context	
	|	$\sigma(\Phi)$	M
		substitute σ over all constraints in Φ	
$\mathcal{R}$	::=	resource environment	
	|	$ident{:}res$	
		add to context	
	|	$\overline{\mathcal{R}_i}^{\,i}$	
		concatenate contexts	
	|	$\cdot$	M
		empty context	
	|	$\sigma(\mathcal{R})$	M
		substitute σ over all SMT terms in all resource types in $\mathcal{R}$	
$\underline{\mathcal{R}}$, Rem, Fr	::=	normalised resource env	
	|	$ident{:}\underline{res}$	
		add to context	
	|	$\overline{\underline{\mathcal{R}}_i}^{\,i}$	
		concatenate contexts	
	|	$\cdot$	M
		empty context	
	|	$\sigma(\mathcal{R})$	M
		substitute σ over all SMT terms in all resource types in $\mathcal{R}$	
ty_extra	::=	extra judgements for explicit and inference typing systems	
	|	$\texttt{smt}\,(\Phi \Rightarrow qterm)$	
		check if $qterm$ is SMT-provable in constraint context Φ	

| $ident{:}\beta \in \mathcal{C}$
lookup type of *ident* in context $\mathcal{C}$
| $\texttt{struct}\, tag \,\&\, \overline{member_i{:}\tau_i}^{i} \in \texttt{Globals}$
lookup types of struct *tag* fields in `Globals`
| $\alpha \equiv \overline{x_i{:}\beta_i}^{i} \mapsto res \in \texttt{Globals}$
lookup body of resource predicate α in `Globals`
| $\mathcal{C} \vdash mem\_val \Rightarrow \beta$
dependent on memory object model
| $\mathcal{C}; \mathcal{L} \vdash term \Rightarrow \beta$
omitted/assumed definition: *term* is (a) well-formed (b) annotated with β
| $pred\_name_1 \neq pred\_name_2$
check if $pred\_name_1$ and $pred\_name_2$ are unequal

formula ::=
| *judgement*
| *ty_extra*
| *opsem_extra*
| *misc_extra*
| $res \equiv res'$
resource type abbreviation
| $res\_term \equiv res\_term'$
resource term abbreviation
| $ret \equiv ret'$
return type abbreviation
| $term \equiv term'$
SMT term / constraint abbrevation
| $texpr \equiv texpr'$
top-level expression abbrevation
| $name{:}pure\_fun \equiv \overline{x_i}^{i} \mapsto tpexpr \in \texttt{Globals}$
lookup type and body of pure function *name* in `Globals`
| $name{:}fun \equiv \overline{x_i}^{i} \mapsto texpr \in \texttt{Globals}$

lookup type and body of function *name* in `Globals`

res_diff	::=	resource difference
	|	`None`
		not possible to take a difference
	|	*res_term* `and` *oarg*
		request is satisfied exactly by *res_term* and the output argument is *oarg*
	|	*oarg* `and` *res_req*
		request is satisfied partially with output argument *oarg* with remaining *res_req*
	|	`bind` res_pat_1:res_1 = res_term_1 `for` $ident_1$ & *oarg* `and` $ident_2$:*rem*
		deconstruct res_term_1:res_1 using res_pat_1 to satisfy request exactly (using $ident_1$ and *oarg*) with remainder $ident_2$:*rem*
res_bind	::=	resource bindings
	|	·
		empty resource binding
	|	*res_pat*:*res* = *res_term*
		match *res_term*:*res* against *res_pat*
	|	$\overline{res\_bind_i}^{\,i}$
		concatenate resource bindings
opt_term	::=	optional SMT term
	|	`None`
	|	*term*
cmp	::=	result of binary comparison
	|	`Lt`
		less-than
	|	`Eq`
		equals

	$\mid$	`Gt`	
		greater-than	
opt_cmp	$::=$	optional result of binary comparison	
	$\mid$	`None`	
	$\mid$	cmp	
opt_cmp_term	$::=$	optional result of binary comparison and SMT term	
	$\mid$	`None`	
	$\mid$	$cmp, term$	
$heap,\ h,\ f$	$::=$	heaps	
	$\mid$	$\{\,\texttt{if}\ term\ \texttt{then}\ res_1\ \texttt{else}\ res_2\}$	
	$\mid$	$\{pred\ \&\ opt\_def\_heap\}$	
	$\mid$	$\{qpred\ \&\ arr\_def\_heap\}$	
	$\mid$	$\overline{heap_i}^{\,i}$	M
	$\mid$	$\cdot$	M
	$\mid$	$\sigma(heap)$	M
$opt\_res\_val\_heap,\ opt\_def\_heap$	$::=$	optional resource term value	
	$\mid$	`None`	
	$\mid$	$def\ \&\ heap$	
	$\mid$	$arr\_def\_heap[term]$	
$arr\_opt\_res\_val\_heap,\ arr\_def\_heap$	$::=$	array of optional resource term value	
	$\mid$	$\cdot$	
	$\mid$	$arr\_def\_heap_1 + arr\_def\_heap_2$	
	$\mid$	$arr\_def\_heap[term] := opt\_def\_heap$	
$opsem\_extra$	$::=$	extra judgements for operational semantics	
	$\mid$	$\forall\, i\ <\ j.\ \texttt{not}\,(pat_i = pval \rightsquigarrow \sigma_i)$	
		all patterns prior to j failed to match/deconstruct	

	$\mid$	$\texttt{fresh}\,(\mathit{mem\_ptr})$ create a fresh address $\mathit{mem\_ptr}$
	$\mid$	term arbitrary logical constraint
$\mathit{misc\_extra}$	::=	extra judgements for proof-related definitions
	$\mid$	$\forall\, x.\ \mathit{iguard} \Rightarrow \mathcal{C}; \mathcal{L}; \Phi \vdash h \Leftarrow \underline{\mathcal{R}}$ meta-logical quantification over heap-typing
	$\mid$	$\forall\, \mathit{term} \sim \mathit{term}'.\ \Phi \vdash \mathit{fun} \sim \mathit{ret}$ meta-logical quantification over related *fun* and *ret*
	$\mid$	$\forall\, \mathit{term} \sim \mathit{term}'.\ \Phi \vdash \mathit{res} \sim \mathit{res}'$ meta-logical quantification over related *res* and res'
	$\mid$	$\mathit{term} \sim \mathit{term}'$ omitted/assumed defintion: SMT terms *term* and term' are related
$\mathit{res\_judge}$	::=	
	$\mid$	$\Phi \vdash \texttt{cmp\_min}\,(\mathit{iguard}, \mathit{iguard}') \leadsto \mathit{opt\_cmp\_term}$ given constraints Φ , *iguard* is potentially included in iguard' (or vice-versa) with ordering and minimum $\mathit{opt\_cmp\_term}$
	$\mid$	$\Phi \vdash \mathit{qpred\_term} \sqsubseteq ?\ \mathit{qpred\_term}' \leadsto \mathit{opt\_cmp}$ given constraints Φ , $\mathit{qpred\_term}$ is potentially included in $\mathit{qpred\_term}'$ (or vice-versa) with ordering $\mathit{opt\_cmp}$
	$\mid$	$\Phi \vdash \mathit{res\_req} \equiv \mathit{res\_req}' \leadsto \mathit{bool}$ resource equality: given constraints Φ, $\mathit{res\_req}$ and $\mathit{res\_req}'$ are equal according to *bool*
	$\mid$	$\Phi \vdash \mathit{res} \equiv \mathit{res}'$ resource equality: given constraints Φ, *res* is equal to res'
	$\mid$	$\Phi \vdash \texttt{simp\_rec}\,(\mathit{res}) \leadsto \mathit{res}', \mathit{bool}$ partial-simplification of resources: given constraints Φ, *res* partially simplifies (strips ifs) to res'
	$\mid$	$\Phi \vdash \texttt{simp}\,(\mathit{res}) \leadsto \mathit{opt\_res}$ partial-simplification of resources: given constraints Φ, *res* attempts a partial simplification (strips ifs) to $\mathit{opt\_res}$

ret_judge ::=

| $\Phi \vdash ret \equiv ret'$
  return type equality: given constraints Φ, ret is equal to ret'

pat_judge ::=

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

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

| $\mathcal{L}; \Phi \vdash res\_pat{:}res \rightsquigarrow \mathcal{L}'; \Phi'; \mathcal{R}'$
  resources pattern to context: given constraints Φ, res_pat of type res produces contexts $\mathcal{L}'; \Phi'; \mathcal{R}'$

| $\mathcal{C}; \mathcal{L}; \Phi \vdash ret\_pat{:}ret \rightsquigarrow \mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'$
  return pattern to context: given context $\mathcal{C}; \mathcal{L}; \Phi$, ret_pat and return type ret produces contexts $\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'$

| $\Phi \vdash ret\_pat{:}ret \rightsquigarrow \mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'$
  return pattern to context: given constraints Φ, ret_pat and return type ret produces contexts $\mathcal{C}'; \mathcal{L}'; \Phi'; \mathcal{R}'$

$expl\_pure$ ::=

| $\mathcal{C} \vdash object\_value \Rightarrow \beta$
  object value synthesises: given $\mathcal{C}$, $object\_value$ synthesises type β

| $\mathcal{C} \vdash pval \Rightarrow \beta$
  pure value synthesises: given $\mathcal{C}$, $pval$ synthesises type β

| $\mathcal{C}; \mathcal{L}; \Phi \vdash pexpr \Rightarrow pure\_ret$
  pure expression synthesises: given $\mathcal{C}; \mathcal{L}; \Phi$, $pexpr$ synthesises a pure (non-resourceful) return type $pure\_ret$

| $\mathcal{C}; \mathcal{L}; \Phi \vdash tpval \Leftarrow pure\_ret$
  pure top-level value checks: given $\mathcal{C}; \mathcal{L}; \Phi$, $tpval$ checks against $pure\_ret$

	$\mid$	$\mathcal{C};\mathcal{L};\Phi \vdash \mathit{tpexpr} \Leftarrow \mathit{pure\_ret}$ pure top-level expression checks: given $\mathcal{C};\mathcal{L};\Phi$, *tpexpr* checks against *pure_ret*
expl_res	::=	
	$\mid$	$\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{pred\_ops} \Rightarrow \mathit{res}$ resource (q)predicate operation term synthesis: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, *pred_ops* synthesises resource *res*
	$\mid$	$\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{res\_term} \Rightarrow \mathit{res}$ resource term synthesises: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, *res_term* synthesises resource *res*
	$\mid$	$\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{res\_term} \Leftarrow \mathit{res}$ resource term checks: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, *res_term* checks against resource *res*
expl_spine	::=	
	$\mid$	$\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{spine} :: \mathit{fun} \gg \mathit{ret}$ function call spine checks: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, compatible *spine*, *fun* produces an *ret*
expl_is_expr	::=	
	$\mid$	$\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{action} \Rightarrow \mathit{ret}$ memory action synthesises: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, *action* synthesises return type *ret*
	$\mid$	$\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{memop} \Rightarrow \mathit{ret}$ memory operation synthesises: given $\mathcal{C};\mathcal{L};\Phi;\mathcal{R}$, *memop* synthesises return type *ret*
	$\mid$	$\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{is\_expr} \Rightarrow \mathit{ret}$ 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*

inf_res ::=

| $\Phi \vdash \mathit{pred\_term} \in ?\ \mathit{qpred\_term} \leadsto \mathit{opt\_term}$

given constraints Φ, *pred_term* is potentially a part of *qpred_term* at index *opt_term*

| $\Phi \vdash \mathit{ident}{:}\underline{\mathit{res}} - ?\ \mathit{res\_req} \leadsto \mathit{res\_diff}$

the difference between $\mathit{ident}{:}\underline{\mathit{res}}$ and requested *res_req* is *res_diff*

| $\Phi \vdash \mathit{ident}_1{:}\underline{\mathit{res}} + ?\ \mathit{res\_term}_2{:}\mathit{res\_req}\ \&\ \mathit{oarg}_2 \leadsto \mathit{res\_term}\ \texttt{and}\ \mathit{oarg}_3$

combining $\mathit{ident}_1{:}\underline{\mathit{res}}$, $\mathit{res\_term}_2{:}\mathit{res\_req}\&\mathit{oarg}_2$, results in *res_term* oarg_3

| $\Phi;\underline{\mathcal{R}} \vdash \texttt{wf}\ \mathit{res\_req} \leadsto \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \mathit{res\_term}\ \texttt{and}\ \mathit{oarg} \dashv \underline{\mathcal{R}'}$

$\Phi;\underline{\mathcal{R}}$ fulfil well-formed request *res_req* (via *res_bind*) for answer *res_term* and *oarg*, with $\underline{\mathcal{R}'}$ leftover

	|	$\Phi; \underline{\mathcal{R}} \vdash \mathit{res\_req} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \mathit{res\_term}\ \texttt{and}\ \mathit{oarg} \dashv \underline{\mathcal{R}'}$ $\Phi; \underline{\mathcal{R}}$ (check well-formedness of and then) fulfil request *res_req* (via *res_bind*) for answer *res_term* and *oarg*, with $\underline{\mathcal{R}'}$ leftover
	|	$\Phi; \underline{\mathcal{R}} \vdash \texttt{if}\ \mathit{term}\ \texttt{then}\ \mathit{res}_1\ \texttt{else}\ \mathit{res}_2 \rightsquigarrow \mathit{ident} \dashv \underline{\mathcal{R}'}$ under-determined conditional resource request: $\Phi; \underline{\mathcal{R}}$ fulfil request for `if` *term* `then` res_1 `else` res_2 with **synthesising** *ident* and $\underline{\mathcal{R}'}$ leftover
	|	$\Phi; \underline{\mathcal{R}} \vdash \texttt{calc}\ y\ \texttt{using}\ \mathit{res} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \mathit{res\_term}\ \texttt{and}\ \mathit{oarg} \dashv \underline{\mathcal{R}'}$ arbirtrary resource and output-arg request: $\Phi; \underline{\mathcal{R}}$ fulfil request for resource *res* and output-arg y (via *res_bind*) with **checking** *res_term* and *oarg*, leaving resources $\underline{\mathcal{R}'}$
elab_is_expr	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \mathit{action} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \mathit{action}' {:} \mathit{norm\_ret} \dashv \underline{\mathcal{R}'}$ memory action elaboration: given $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}}$, *action* elaborates (via *res_bind*) to $\mathit{action}'{:}\mathit{norm\_ret}$, with $\underline{\mathcal{R}'}$ leftover
	|	$\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \mathit{memop} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \mathit{memop}' {:} \mathit{norm\_ret} \dashv \underline{\mathcal{R}'}$ memory operation elaboration: given $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}}$, *memop* elaborates to (via *res_bind*) to $\mathit{memop}'{:}\mathit{norm\_ret}$, with $\underline{\mathcal{R}'}$ leftover
	|	$\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \mathit{is\_expr} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ (\mathit{is\_expr}') {:} \mathit{ret} \dashv \underline{\mathcal{R}'}$ indet. seq. expression elaboration: given $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}}$, *is_expr* elaborates (via *res_bind*) to $\mathit{is\_expr}'{:}\mathit{ret}$, with $\underline{\mathcal{R}'}$ leftover
elab_spine	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \mathit{spine} :: \underline{\mathit{fun}} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \mathit{spine}'\ \texttt{and}\ \mathit{norm\_ret} \dashv \underline{\mathcal{R}'}$ spine elaboration: given $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}}$, arguments *spine* and function type $\underline{\mathit{fun}}$ elaborate (via *res_bind*) to spine' and result type *norm_ret* , with $\underline{\mathcal{R}'}$ leftover
elab_seq_expr	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash \mathit{seq\_expr} \rightsquigarrow \texttt{bind}\ \mathit{res\_bind}\ \texttt{for}\ \mathit{seq\_expr}' {:} \mathit{norm\_ret} \dashv \underline{\mathcal{R}'}$ seq. expression elaboration: given $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}}$, *seq_expr* elaborates (via *res_bind*) to $\mathit{seq\_expr}'{:}\mathit{norm\_ret}$, with $\underline{\mathcal{R}'}$ leftover

elab_top ::=

| $\Phi \vdash res \rightsquigarrow res\_pat$
resource normalisation by pat-matching: under constraints Φ , *res* will produces a normalised resourced context if it matches against *res_pat*

| $\Phi \vdash ret \rightsquigarrow ret\_pat$
return-value normalisation by pattern-matching: under constraints Φ , *ret* will produce a normalised resourced context if it matches against *ret_pat*

| $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash is\_texpr \Leftarrow \underline{ret} \rightsquigarrow texpr$
top-level indet. seq. expression elaboration: given $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}}$, *is_texpr* elaborates to *texpr*

| $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash tval \Leftarrow \underline{ret} \rightsquigarrow \texttt{bind}\ res\_bind\ \texttt{for}\ tval' \dashv \underline{\mathcal{R}'}$
top-level value elaboration: given $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}}$, *tval* elaborates (via *res_bind*) to $tval'$ with $\underline{\mathcal{R}'}$ leftover

| $\Phi; \underline{\mathcal{R}} \rightsquigarrow res\_bind$
partial-simplification of resource context: given $\Phi; \underline{\mathcal{R}}$ can partially simplify the resources using *res_bind*

| $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash seq\_texpr \Leftarrow \underline{ret} \rightsquigarrow seq\_texpr'$
top-level seq. expression elaboration: given $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}}$, *seq_texpr* checks against $\underline{ret}$ and elaborates to seq_texpr'

| $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}} \vdash texpr \Leftarrow \underline{ret} \rightsquigarrow texpr'$
top-level expression elaboration: given $\mathcal{C}; \mathcal{L}; \Phi; \underline{\mathcal{R}}$, *texpr* checks against $\underline{ret}$ and elaborates to $texpr'$

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

	$\mid$	$\langle h; \mathit{res\_pat} = \mathit{res\_val} \rangle \rightsquigarrow \langle h'; \sigma \rangle$ resource term deconstruction: $\mathit{res\_pat}$ deconstructs $\mathit{res\_val}$ to produce substitution σ
	$\mid$	$\langle h; \overline{\mathit{ret\_pat}_i = \mathit{ret\_term}_i}^{\,i} \rangle \rightsquigarrow \langle h'; \sigma \rangle$ return value deconstruction: $\mathit{ret\_pat}_i$ deconstructs $\mathit{ret\_val}_i$ to produce substitution σ
	$\mid$	$\langle h; \overline{x_i = \mathit{spine\_elem}_i}^{\,i} \rangle :: \mathit{fun} \gg \langle h'; \sigma; \mathit{ret} \rangle$ function call spine: heap h and formal parameters x_i assigned to $\mathit{spine\_elem}_i$ for function of type fun, produce new heap h' substitution σ and result type ret
$\mathit{pure\_opsem\_defns}$	::=	
	$\mid$	$\langle \mathit{pexpr} \rangle \longrightarrow \langle \mathit{tpexpr}{:}\mathit{pure\_ret} \rangle$
	$\mid$	$\langle \mathit{tpexpr} \rangle \longrightarrow \langle \mathit{tpexpr}' \rangle$
$\mathit{opsem\_defns}$	::=	
	$\mid$	$\langle h; \mathit{pred\_ops} \rangle \Downarrow \langle h'; \mathit{res\_val} \rangle$ big-step resource (q)points-to operation reduction: $\langle h; \mathit{pred\_ops} \rangle$ reduces to $\langle h'; \mathit{res\_val} \rangle$
	$\mid$	$\texttt{footprint\_of}\ \mathit{res\_val}\ \texttt{in}\ h \rightsquigarrow h_1\ \mathrm{rem}\ h_2$ footprint of $\mathit{res\_val}$ in heap h is h_1 with h_2 remainder/frame
	$\mid$	$\langle h; \mathit{res\_term} \rangle \Downarrow \langle h'; \mathit{res\_val} \rangle$ big-step resource term reduction: $\langle h; \mathit{res\_term} \rangle$ reduces to $\langle h'; \mathit{res\_val} \rangle$
	$\mid$	$\langle h; \mathit{action} \rangle \longrightarrow \langle h'; \mathit{is\_expr} \rangle$
	$\mid$	$\langle h; \mathit{memop} \rangle \longrightarrow \langle h'; \mathit{is\_expr} \rangle$
	$\mid$	$\langle h; \mathit{is\_expr} \rangle \longrightarrow \langle h'; \mathit{is\_expr}' \rangle$
	$\mid$	$\langle h; \mathit{seq\_expr} \rangle \longrightarrow \langle h'; \mathit{texpr}{:}\mathit{ret} \rangle$
	$\mid$	$\langle h; \mathit{seq\_texpr} \rangle \longrightarrow \langle h'; \mathit{texpr} \rangle$
	$\mid$	$\langle h; \mathit{is\_texpr} \rangle \longrightarrow \langle h'; \mathit{texpr} \rangle$
	$\mid$	$\langle h; \mathit{texpr} \rangle \longrightarrow \langle h'; \mathit{texpr}' \rangle$

$proof\_defns$ ::=

| $\overline{x_i}^{\,i} :: fun \rightsquigarrow \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \mid ret$
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 $\underline{\mathcal{R}}$

| $\Phi \vdash res \sim res'$
res is related to res'

| $\Phi \vdash fun \sim ret$
fun is related to ret

$spec\_defns$ ::=

| $[\![spec]\!](opt\_ident) = res$
specification $spec$ (with optional record opt_ident) represents resource res (opt_ident is present return when a return is expected, absent when it is not)

| $[\![spec]\!] = norm\_ret$
specification $spec$ represents normalised return type $norm\_ret$

| $[\![spec \mid \underline{ret}]\!] = norm\_fun$
specification $spec$ represents normalised argument type $norm\_fun$

| $[\![\tau name(\overline{\tau_i\, x_i}^{\,i})\, \texttt{requires}\; spec_1\; \texttt{ensures}\; spec_2]\!] = norm\_fun$
user-defined C function specification represents normalised argument type $norm\_fun$

| $[\![\texttt{predicate}\,\{\,\overline{\beta'_j\;y_j}^{\,j}\,\}\alpha(\,\overline{\beta_i\;x_i}^{\,i}\,)\{\mathit{spec}\}]\!] = \alpha' \;\equiv\; \overline{r_k{:}\beta''_k}^{\,k} \;\mapsto\; \mathit{res}$
user-defined resource predicate definition represents predicate