Appendix to A Flexible Type System for Fearless Concurrency

MAE MILANO, University of California, Berkeley, USA
JOSHUA TURCOTTI, University of California, Berkeley, USA
ANDREW C. MYERS, Cornell University, USA

This appendix presents all necessary technical details of the type system from the accompanying paper, included to resolve any lingering ambiguities that may have arisen in its consumption.

Contents

Authors' addresses: Mae Milano, University of California, Berkeley, Berkeley, CA, USA, mpmilano@berkeley.edu; Joshua Turcotti, University of California, Berkeley, Berkeley, CA, USA, jturcotti@berkeley.edu; Andrew C. Myers, Cornell University, Ithaca, NY, 14853, USA, andru@cs.cornell.edu.

2022. Manuscript submitted to ACM

1 FUNCTION SYNTAX

In this section we present the syntax by which we expose function types to the programmar. As discussed in the paper, and richly formally explored in the proofs below, our function types contain explicit instances of the context Γ (which would be possible but still exceed the complexity of classical function signatures to write down) and $\mathcal{H}$ (which we would never reasonably expect a programmer to write down). This section provides the alternative.

We begin by providing a grammar of the new function syntax for reference, though we encourage the reader to follow along with our more gradual introduction of it immediately afterwards.

$$\begin{aligned}
\text{FANCY_FUNDEF} &:= \texttt{def}\ fn(\overline{x : \tau}) : \tau\ \text{OPT_BEF OPT_AFT OPT_CONS}\{e\} \\
\text{OPT_BEF} &:= \varepsilon \mid \texttt{before:}\ \overline{\text{BEF}} \\
\text{OPT_AFT} &:= \varepsilon \mid \texttt{after:}\ \overline{\text{AFT}} \\
\text{OPT_CONS} &:= \varepsilon \mid \texttt{consumes}\ \overline{x} \\
\text{BEF} &:= x \mid x.f \mid \text{BEF} \sim \text{BEF} \mid \text{BEF} \sim ? \mid ? \sim \text{BEF} \mid x.? \mid \bot \\
\text{AFT} &:= x \mid x.f \mid \texttt{old}\ x.f \mid \texttt{result} \mid \bot \mid \text{AFT} \sim \text{AFT}
\end{aligned}$$

As discussed in the paper, the first necessary contribution of a function syntax is a robust default case. Given a classical function signature such as:

$$\texttt{def}\ fn(x_1 : \tau_1, x_2 : \tau_2, \ldots, x_n : \tau_n) : \tau_0 \tag{1}$$

We interpret this as a function that assumes all of its arguments come from distinct regions, all of which are preserved, and that the return region is similarly distinct. This gives us again the parsed core function type:

$$(r_1^{\cdot}\langle\rangle, r_2^{\cdot}\langle\rangle, \ldots, r_n^{\cdot}\langle\rangle; x_1 : r_1\ \tau_1, x_2 : r_2\ \tau_2, \ldots, x_n : r_n\ \tau_n) \Rightarrow (r_0^{\cdot}\langle\rangle, r_1^{\cdot}\langle\rangle, r_2^{\cdot}\langle\rangle, \ldots, r_n^{\cdot}\langle\rangle; x_1 : r_1\ \tau_1, x_2 : r_2\ \tau_2, \ldots, x_n : r_n\ \tau_n; r_0; \tau_0) \tag{2}$$

The next expressiveness we introduce is the `consumes` keyword, which negates the assumption that argument regions survive the function. If given the signature:

$$\texttt{def}\ fn(x_1 : \tau_1, t_2 : \tau_2, x_3 : \tau_3) : \tau_0\ \texttt{consumes}\ x_3 \tag{3}$$

We parse similarly to the unannotated case, but lacking the region of x_3 at return, which we note by dropping it from the output $\mathcal{H}$ and converting it to ⊥ in Γ:

$$(r_1^{\cdot}\langle\rangle, r_2^{\cdot}\langle\rangle, r_3^{\cdot}\langle\rangle; x_1 : r_1\ \tau_1, x_2 : r_2\ \tau_2, x_3 : r_3\ \tau_3) \Rightarrow (r_0^{\cdot}\langle\rangle, r_1^{\cdot}\langle\rangle, r_2^{\cdot}\langle\rangle; x_1 : r_1\ \tau_1, x_2 : r_2\ \tau_2, x_3 : \bot\ \tau_3; r_0; \tau_0) \tag{4}$$

The next level of expressiveness that could be required comes with the optional `before` clause, which affects the input context of the function. Most simply, we can expect a field to be explored at the call to the function:

$$\texttt{def}\ fn(x_1 : \tau_1, x_2 : \tau_2) : \tau_0\ \texttt{before:}\ x_1.f \tag{5}$$

$$(r_1^{\cdot}\langle x_1[f \rightarrowtail r_3]\rangle, r_2^{\cdot}\langle\rangle, r_3^{\cdot}\langle\rangle; x_1 : r_1\ \tau_1, x_2 : r_2\ \tau_2) \Rightarrow (r_0^{\cdot}\langle\rangle, r_1^{\cdot}\langle\rangle, r_2^{\cdot}\langle\rangle; x_1 : r_1\ \tau_1, x_2 : r_2\ \tau_2; r_0; \tau_0) \tag{6}$$

On it's own, this is not very useful, as we expect $x_1.f$ to be explored but point to a fresh region r_3. It would be made useful by adding the $\sim$ notation to the `before` as well to indicate that we expect $x_1.f$ to point to an already named region, for example that of x_2 instead of a fresh one:

$$\texttt{def}\ \mathit{fn}(x_1 : \tau_1, x_2 : \tau_2) : \tau_0\ \texttt{before:}\ x_1.f \sim x_2 \tag{7}$$

$$(r_1^{\cdot}\langle x_1[f \rightarrowtail r_2]\rangle, r_2^{\cdot}\langle\rangle; x_1 : r_1\ \tau_1, x_2 : r_2\ \tau_2) \Rightarrow (r_0^{\cdot}\langle\rangle, r_1^{\cdot}\langle\rangle, r_2^{\cdot}\langle\rangle; x_1 : r_1\ \tau_1, x_2 : r_2\ \tau_2; r_0; \tau_0) \tag{8}$$

This function now comes with a strict expressiveness gain, as we can call it on two arguments that have overlapping object graphs, for example:

```
let x = new t; let y = new t;
x.f = y;
f(x, y)
```

Note that the function above expects this relationship between the regions of x_1 and x_2 to be erased by the end of the function body, which could be done for example by repointing $x_1.f$ to a fresh value. If we wanted instead to preserve the relationship we could include it in an `after` clause as well:

$$\texttt{def}\ \mathit{fn}(x_1 : \tau_1, x_2 : \tau_2) : \tau_0\ \texttt{before:}\ x_1.f \sim x_2\ \texttt{after:}\ x_1.f \sim x_2 \tag{9}$$

$$(r_1^{\cdot}\langle x_1[f \rightarrowtail r_2]\rangle, r_2^{\cdot}\langle\rangle; x_1 : r_1\ \tau_1, x_2 : r_2\ \tau_2) \Rightarrow (r_0^{\cdot}\langle\rangle, r_1^{\cdot}\langle x_1[f \rightarrowtail r_2\rangle, r_2^{\cdot}\langle\rangle, r_0\langle\rangle; x_1 : r_1\ \tau_1, x_2 : r_2\ \tau_2; r_0; \tau_0) \tag{10}$$

The $\sim$ relation can be applied between variables, fields, or transitively, as illustrated in the following function, which takes two variables whose objects graphs are disjoint *except* that references $x_1.f$ and $x_2.f$ share a target region, which itself contains x_3, and expects that shared region to be lost by the end.

$$\texttt{def}\ \mathit{fn}(x_1 : \tau, x_2 : \tau, x_3 : \tau') : \tau\ \texttt{before:}\ x_1.f \sim x_2.f \sim x_3\ \texttt{consumes}\ x_3 \tag{11}$$

$$(r_1^{\cdot}\langle x_1[f \rightarrowtail r_3]\rangle, r_2^{\cdot}\langle x_2[f \rightarrowtail r_3]\rangle, r_3^{\cdot}\langle\rangle; x_1 : r_1\ \tau, x_2 : r_2\ \tau, x_3 : r_3\ \tau') \Rightarrow (r_0^{\cdot}\langle\rangle, r_1^{\cdot}\langle\rangle, r_2^{\cdot}\langle\rangle; x_1 : r_1\ \tau_1, x_2 : r_2\ \tau_2, x_3 : \bot\ \tau_3; r_0; \tau') \tag{12}$$

The default behavior for expecting explored fields in an `after` clause always expects them in a fresh region. If instead we wish for them to share the old region of themselves or another field, we use the `old` keyword:

$$\texttt{def}\ \mathit{fn}(x_1 : \tau, x_2 : \tau) : \tau\ \texttt{before:}\ x_1.f \sim x_2.f\ \texttt{after:}\ x_1.f \sim \texttt{old}\ x_1.f \tag{13}$$

$$(r_1^{\cdot}\langle x_1[f \rightarrowtail r_3]\rangle, r_2^{\cdot}\langle x_2[f \rightarrowtail r_3]\rangle, r_3^{\cdot}\langle\rangle; x_1 : r_1\ \tau, x_2 : r_2\ \tau) \Rightarrow (r_0^{\cdot}\langle\rangle, r_1^{\cdot}\langle x_1[f \rightarrowtail r_3]\rangle, r_3^{\cdot}\langle\rangle; x_1 : r_1\ \tau_1, x_2 : r_2\ \tau_2; r_0; \tau') \tag{14}$$

Additionally, we can use the $\sim$ relation to indicate fields as targetting $\bot$, meaning they are in an invalid state until assigned to. This function takes x_1 with field f in an invalid state, expects it re-assigned it to the field $x_2.f$ which was also expected to be explored, and expects $x_2.f$ to point to a fresh region:

$$\texttt{def}\ \mathit{fn}(x_1 : \tau, x_2 : \tau) : \tau\ \texttt{before:}\ x_1.f \sim \bot, x_2.f\ \texttt{after:}\ x_1.f \sim \texttt{old}\ x_2.f, x_2.f \tag{15}$$

$$(r_1^{\cdot}\langle x_1[f \rightarrowtail \bot]\rangle, r_2^{\cdot}\langle x_2[f \rightarrowtail r_3]\rangle, r_3^{\cdot}\langle\rangle; x_1 : r_1\ \tau, x_2 : r_2\ \tau) \Rightarrow (r_0\langle\rangle, r_1^{\cdot}\langle x_1[f \rightarrowtail r_3]\rangle, r_2^{\cdot}\langle f \rightarrowtail r_4\rangle, r_3^{\cdot}\langle\rangle, r_4^{\cdot}\langle\rangle; x_1 : r_1\ \tau_1, x_2 : r_2\ \tau_2; r_0; \tau') \tag{16}$$

If we wish to associate the return region with another input or output region, instead of assuming it to be a fresh output region, we may do so with the `result` keyword, used similarly to $\bot$ but only in an `after` clause.

The following function could return its argument x:

$$\texttt{def}\ fn(x:\tau):\tau\ \texttt{after:result} \sim x \tag{17}$$

$$(r^{\cdot}\langle\rangle; x : r\ \tau) \Rightarrow (r^{\cdot}\langle\rangle; x : r\ \tau; r; \tau) \tag{18}$$

The following function could re-assign $x_1.f$ and return its old value:

$$\texttt{def}\ fn(x_1:\tau):\tau'\ \texttt{before:}\ x_1.f\ \texttt{after:}\ \texttt{result} \sim \texttt{old}\ x_1.f \tag{19}$$

$$(r_1^{\cdot}\langle x_1[f \rightarrowtail r_2]\rangle, r_2^{\cdot}\langle\rangle; x_1 : r_1\ \tau) \Rightarrow (r_1^{\cdot}\langle\rangle, r_2^{\cdot}\langle\rangle; x_1 : r_1\ \tau_1; r_2; \tau') \tag{20}$$

The following function could explore $x_1.f$ and return its new value:

$$\texttt{def}\ fn(x_1:\tau):\tau'\ \texttt{after:}\ \texttt{result} \sim x_1.f \tag{21}$$

$$(r_1^{\cdot}\langle x_1[f \rightarrowtail r_2]\rangle, r_2^{\cdot}\langle\rangle; x_1 : r_1\ \tau) \Rightarrow (r_1^{\cdot}\langle\rangle, r_2^{\cdot}\langle\rangle; x_1 : r_1\ \tau_1; r_2; \tau') \tag{22}$$

The keywords presented so far are sufficiently powerful to express any desired input and output context exacty. Sometimes, we wish to only *partially* describe the input contexts we expect, which motivates the introduction of *pinnedness* to our function types. In particular, one can use the syntax $\sim ?$ or $? \sim$ to specify that a region may be shared with other, unknown focussed variables at the callsite; cueing pinning:

$$\texttt{def}\ fn(x_1:\tau_1, x_2:\tau_2):\tau_0\ \texttt{before:}x_1 \sim ? \tag{23}$$

$$(r_1^{\dagger}\langle\rangle, r_2^{\cdot}\langle\rangle; x_1 : r_1\ \tau_1, x_2 : r_2\ \tau_2) \Rightarrow (r_0^{\cdot}\langle\rangle, r_1^{\dagger}\langle\rangle, r_2^{\cdot}\langle\rangle; x_1 : r_1\ \tau_1, x_2 : r_2\ \tau_2; r_0; \tau_0) \tag{24}$$

The `before:` clause can be freely used to expect structure even in pinned regions:

$$\texttt{def}\ fn(x:\tau, y:\tau'):\texttt{unit}\ \texttt{before:}x\ y\ ?, x.f, x.g, y.h \tag{25}$$

$$(r_1^{\dagger}\langle x^{\cdot}[f \rightarrowtail r_2, g \rightarrowtail r_3], y^{\cdot}[h \rightarrowtail r_4]\rangle, r_2^{\cdot}\langle\rangle, r_3^{\cdot}\langle\rangle, r_4^{\cdot}\langle\rangle; x : r_1\ \tau, y : r_1\ \tau') \Rightarrow (t_0^{\cdot}\langle\rangle, r_1^{\dagger}\langle\rangle; x : r_1\ \tau, y : r_1\ \tau'; r_0; \texttt{unit}) \tag{26}$$

This function would allow us to call it and access the fields of x and y even if other focussed variables were present at the callsite, but if explored fields beyond those mentioned were present they would have to be retracted. If we wish to lift this obligation we can additionally indicate the presence of unknown fields by the $x.?$ notation indicating *pinned variables*:

$$\texttt{def}\ fn(x:\tau, y:\tau'):\texttt{unit}\ \texttt{before:}x\ y\ ?, x.?, y.h \tag{27}$$

$$(r_1^{\dagger}\langle x^{\dagger}[], y^{\cdot}[h \rightarrowtail r_2]\rangle, r_2^{\cdot}\langle\rangle; x : r_1\ \tau, y : r_1\ \tau') \Rightarrow (t_0^{\cdot}\langle\rangle, r_1^{\dagger}\langle x^{\dagger}[]\rangle; x : r_1\ \tau, y : r_1\ \tau'; r_0; \texttt{unit}) \tag{28}$$

This function is callable no matter how many fields of x are explored at the callsite, but cannot access any within the function body. Note also that $x^{\dagger}[]$ occurs at output, as there is no way to unfocus it. This example pins variables in a pinned region, but it is also possible to pin variables in an unpinned region, just as it is possible to pin regions containing unpinned variables.

This concludes the presentation of all elements of the surface-level function syntax, providing a powerful sub-languages for intuitively building function types in our larger type system.

2 FULL SPECIFICATION OF THE SYSTEM

In this section we delve into the presented type system in full formal detail. In particular, we phrase the theorems *Progress* and *Preservation* formally (see 2.2 and 2.3), and prove them, to establish with confidence correctness of the safety properties discussed in the paper.

2.1 Typechecking Metasystem

2.1.1 Functions. We parse the program in the following order: First, we let $\mathcal{F}$ be a single-pass computed list of function names *fn*, and their types τ_f. Then, with this list in hand, we go on to type-check each function body, and after successful application of rule T0, we store the function body as a lambda expression $\lambda x_1, \ldots x_n : e$ in a map Λ from function names *fn* to lambda expressions. $\mathcal{F}$ and Λ are then accessible when typechecking and stepping program bodies.

2.1.2 Types. We parse struct definitions in order to populate the sets $\mathit{fields}(\tau)$ for every type $\tau \in \mathit{Struct}$, which is in particular a set of $(q_{\text{FLD}}\ f\ \tau)$ triples.

2.2 Grammar

$$(\text{function})\ \mathit{fn} \in \mathit{FunctionNames}$$
$$(\text{variable})\ x \in \mathit{VariableNames}$$
$$(\text{class})\ \mathit{Struct} \in \mathit{ClassNames}$$
$$(\text{region})\ r \in \mathit{RegionNames}$$
$$(\text{location})\ l \in \mathit{LocationNames}$$
$$(\text{field})\ f \in \mathit{FieldNames}$$

$$(\text{function type})\ \tau_{fn} ::= (\mathcal{H}; \Gamma) \Rightarrow (\mathcal{H}'; \Gamma'; r, \tau)$$

$$(\text{type})\ \tau ::= \mathit{Struct} \mid \mathit{Struct}?$$
$$(\text{pinnedness metavariable})\ \circ ::= \dagger \mid \cdot$$
$$(\text{heap tracking context})\ \mathcal{H} ::= r^{\circ}\langle X \rangle,\ \mathcal{H} \mid \cdot$$
$$(\text{region tracking contents})\ X ::= x^{\circ}[F],\ X \mid \cdot$$
$$(\text{variable tracking contents})\ F ::= f \rightarrowtail r,\ F \mid \cdot$$
$$(\text{variable bindings context})\ \Gamma ::= x : r\ \tau,\ \Gamma \mid \cdot$$
$$(\text{region names context})\ \Omega ::= r,\ \Omega \mid \cdot$$
$$(\text{location bindings context})\ \mathrm{P} ::= l : r\ \tau,\ \mathrm{P} \mid \cdot$$

$$(\text{variable renaming})\ \Phi_x ::= x \mapsto x, \Phi_x \mid \cdot$$
$$(\text{region renaming})\ \Phi_r ::= r \mapsto r, \Phi_r \mid \cdot$$
$$(\text{frame type})\ A ::= \cdot; \mathcal{H}, \Omega \mid r; \cdot \mid r; X \mid r; x \mid r; x, F \mid \Gamma; \cdot \mid \cdot; r, r$$

$$
\begin{aligned}
\text{(function definition) } \textsc{fdef} &::= \texttt{def } fn : \tau_{fn}\{e\} \\
\text{(program) } p &::= \textsc{fdef};\ p \mid e \\
\text{(expression) } e &::= l \mid x \mid e;e \mid e.f \mid e.f = e \mid x = e \mid fn(x,\ldots,x) \mid e \oplus e \mid \texttt{new } \tau \mid \texttt{declare } x : \tau \texttt{ in } \{e\} \\
&\mid \texttt{if } (e)\ \{e\} \texttt{ else } \{e\} \mid \texttt{while } (e)\ \{e\} \mid \texttt{send-}\tau(e) \mid \texttt{recv-}\tau() \mid \texttt{if disconnected}(x,x)\ \{e\} \texttt{ else } \{e\} \\
&\mid \texttt{none } \tau \mid \texttt{some}(e) \mid \texttt{let some}(x) = (e) \texttt{ in } \{e\} \texttt{ else } \{e\} \\
\text{(evaluation context) } E[] &::= [];e \mid e.f = [] \mid x = [] \mid [] \oplus e \mid l \oplus [] \mid \texttt{if}([])\{e\} \texttt{ else } \{e\} \\
&\mid \texttt{send-}\tau([]) \mid \texttt{some}([]) \mid \texttt{let some}(x) = ([]) \texttt{ in } \{e\} \texttt{ else } \{e\}
\end{aligned}
$$

$$
\begin{aligned}
\text{(dynamic reservation) } d &::= l,\ d \mid \cdot \\
\text{(heap) } h &::= l \mapsto (\tau, v),\ h \mid \cdot \\
\text{(stack) } s &::= x \mapsto l,\ s \mid \cdot \\
\text{(heap value) } v &::= f \mapsto l,\ v \mid \cdot
\end{aligned}
$$

2.3 Invariants

$$
\begin{aligned}
\text{(field sequence) } L &:= l.f, L \mid \cdot \\
\text{(outward path) } p &:= (r \overset{L}{\dashrightarrow} l)
\end{aligned}
$$

$$
\begin{aligned}
\textit{live-roots}(\mathcal{H}, \Gamma, \mathrm{P}, h, s) &:= \mathrm{P}_r \cup \bigcup_{r\langle X\rangle \in \mathcal{H}} \left[\left[s(\Gamma_r^{-1}(r)) \cup \bigcup_{x[F] \in X} \bigcup_{f \rightarrowtail r_f \in F} (h_v(s(x))[f]) \right] \times \{r\} \right] \\
\textit{tracked-refs}(\mathcal{H}, s) &:= \{ l.f \mid \exists r, x, r_f, X, F : \left[(r\langle X\rangle \in \mathcal{H}) \land (x[F] \in X) \land (f \rightarrowtail r_f \in F) \land (s(x) = l) \right] \} \\
\textit{is-iso}(l.f, h) &:= \exists \tau : \texttt{iso } f\ \tau \in \textit{fields}(h_\tau(l)) \\
\textit{iso-subseq}(L, h) &:= [l.f \in L \mid \textit{is-iso}(l.f, h)] \\
\textit{outward-paths}(\mathcal{H}, \Gamma, \mathrm{P}, h, s) &:= \{ (r \overset{L}{\dashrightarrow} l_{n+1}) \mid \exists l_0.f_0, \ldots, l_n.f_n : (\forall i \in [0, n] : (h_v(l_i)[f_i] = l_{i+1})) \\
&\quad \land (l_0, r) \in \textit{live-roots}(\mathcal{H}, \Gamma, \mathrm{P}, h, s) \\
&\quad \land L = \textit{iso-subseq}(\overline{l_i.f_i}, h) \land L \cap \textit{tracked-refs}(\mathcal{H}, s) = \emptyset \} \\
\textit{live-set}(\mathcal{H}, \Gamma, \mathrm{P}, h, s) &:= \{ l \mid (r \overset{L}{\dashrightarrow} l) \in \textit{outward-paths}(\mathcal{H}, \Gamma, \mathrm{P}, h, s) \} \\
\textit{tracked-set}(\mathcal{H}, \Gamma, \mathrm{P}, h, s) &:= \{ l \mid (r \overset{\cdot}{\dashrightarrow} l) \in \textit{outward-paths}(\mathcal{H}, \Gamma, \mathrm{P}, h, s) \}
\end{aligned}
$$

$\boxed{\text{I1}}$ – $\textsc{Reservation-Sufficiency}(\mathcal{H}, \Gamma, \mathrm{P}, d, h, s) := \textit{live-set}(\mathcal{H}, \Gamma, \mathrm{P}, h, s) \subseteq d \subseteq \textit{dom}(h)$

$\boxed{\text{I2}}$ – $\textsc{Tree-Of-Untracked-Regions}(\mathcal{H}, \Gamma, \mathrm{P}, h, s) := \forall (r \overset{L}{\dashrightarrow} l), (r' \overset{L'}{\dashrightarrow} l') \in \textit{outward-paths}(\mathcal{H}, \Gamma, \mathrm{P}, h, s) : (l = l') \Rightarrow (r, L) = (r', L')$

$\boxed{\text{I3}}$ – $\textsc{Heap-Closure}(h) := \forall, \tau, v, f, l' : (h(l) = (\tau, v) \wedge (v[f] = l'))$
$\Rightarrow \exists q_{\text{FLD}}, \tau_f, v_f : (q_{\text{FLD}}\ f\ \tau_f \in \textit{fields}(\tau) \wedge h(l') = (\tau_f, v_f))$

$\boxed{\text{I4}}$ – $\textsc{Location-Type-Consistency}(\mathcal{H}, \Gamma, \mathrm{P}, h, s) := (\forall l \in \textit{dom}(\mathrm{P}) : \mathrm{P}_r(l) \in \textit{regs}(\mathcal{H}) \wedge \mathrm{P}_\tau(l) = h_\tau(l))$
$\wedge (\forall x : \Gamma_r(x) \in \textit{regs}(\mathcal{H}) \Rightarrow h_\tau(s(x)) = \Gamma_\tau(x))$

$\boxed{\text{I5}}$ – $\textsc{Variable-Region-Consistency}(\mathcal{H}, \Gamma) := \forall (r\langle X\rangle \in \mathcal{H}) : \forall (x[F] \in X) : (\Gamma_r(x) = r)$

$\boxed{\text{I6}}$ – $\textsc{Focus-Non-Aliasing}(\mathcal{H}, s) := \forall x, x' \in \textit{vars}(\mathcal{H}) : x \neq x' \Rightarrow s(x) \neq s(x')$

$\boxed{\text{I7}}$ – $\textsc{Region-Names-Bounding}(\mathcal{H}, \Gamma, \Omega, \mathrm{P}) := \textit{regs}(\mathcal{H}) \cup \textit{regtgts}(\mathcal{H}) \cup \textit{range}(\Gamma_r) \cup \textit{range}(\mathrm{P}_r) \subseteq \Omega \uplus \{\bot\}$

$\boxed{\text{I0}}$ – $\textsc{Sound-Configuration}(\mathcal{H}, \Gamma, \Omega, \mathrm{P}, d, h, s) := \text{I1}(\mathcal{H}, \Gamma, \mathrm{P}, h, s, d) \wedge \text{I2}(\mathcal{H}, \Gamma, \mathrm{P}, h, s) \wedge \text{I3}(h)$
$\wedge\ \text{I4}(\mathcal{H}, \Gamma, \mathrm{P}, h, s) \wedge \text{I5}(\mathcal{H}, \Gamma) \wedge \text{I6}(\mathcal{H}, s) \wedge \text{I7}(\mathcal{H}, \Gamma, \Omega, \mathrm{P})$

2.4 Typing Rules

$\boxed{\text{T0}}$ - $\textsc{Function-Definition}$

$$\frac{\mathcal{H}; \Gamma; \textit{regs}(\mathcal{H}); \cdot \vdash e : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega \qquad \tau_{fn} = (\mathcal{H}; \Gamma) \Rightarrow (\mathcal{H}'; \Gamma'; r, \tau) \qquad (fn, \tau f) \in \mathcal{F}}{\vdash \texttt{def}\ fn : \tau f\, \{e\}}$$

$\boxed{\text{T1}}$ - $\textsc{Location-Ref}$

$$\frac{(l : r\ \tau) \in \mathrm{P} \qquad r \in \textit{regs}(\mathcal{H})}{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash l : r\ \tau \dashv \mathcal{H}; \Gamma; \Omega}$$

$\boxed{\text{T2}}$ - $\textsc{Variable-Ref}$

$$\frac{x : r\ \tau \in \Gamma \qquad r \in \textit{regs}(\mathcal{H})}{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash x : r\ \tau \dashv \mathcal{H}; \Gamma; \Omega}$$

$\boxed{\text{T3}}$ - $\textsc{Sequence}$

$$\frac{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega' \qquad \mathcal{H}'; \Gamma'; \Omega'; \cdot \vdash e' : r'\ \tau' \dashv \mathcal{H}''; \Gamma''; \Omega''}{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e; e' : r'\ \tau' \dashv \mathcal{H}''; \Gamma''; \Omega''}$$

$\boxed{\text{T4}}$ - $\textsc{Non-Isolated-Field-Reference}$

$$\frac{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega' \qquad \cdot\ f\ \tau_f \in \textit{fields}(\tau)}{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e.f : r\ \tau_f \dashv \mathcal{H}'; \Gamma; \Omega'}$$

T5 - Isolated-Field-Reference

$$\frac{\texttt{iso}\ f\ \tau_f \in \mathit{fields}(\tau) \qquad r^{\circ}\langle x^{\circ'}[f \rightarrowtail r_f, F], X\rangle \in \mathcal{H} \qquad r_f^{\circ''}\langle X'\rangle \in \mathcal{H}}{\mathcal{H}; x : r\ \tau, \Gamma; \Omega; \mathrm{P} \vdash x.f : r_f\ \tau_f \dashv \mathcal{H}; x : r\ \tau, \Gamma; \Omega}$$

T6 - Non-Isolated-Field-Assignment

$$\frac{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e_f : r\ \tau_f \dashv \mathcal{H}'; \Gamma'; \Omega' \qquad \mathcal{H}'; \Gamma'; \Omega'; \mathrm{P}' \vdash e : r\ \tau \dashv \mathcal{H}''; \Gamma''; \Omega'' e_f, \mathrm{P}, \mathrm{P}'}{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e.f = e_f : r\ \tau_f \dashv \mathcal{H}''; \Gamma''; \Omega''}$$

T7 - Isolated-Field-Assignment

$$\frac{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e_f : r_f\ \tau_f \dashv \mathcal{H}', r^{\circ}\langle x^{\circ'}[f \rightarrowtail r_{old}, F], X\rangle; x : r\ \tau, \Gamma'; \Omega' \qquad \texttt{iso}\ f\ \tau_f \in \mathit{fields}(\tau)}{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash x.f = e_f : r_f\ \tau_f \dashv \mathcal{H}', r^{\circ}\langle x^{\circ'}[f \rightarrowtail r_f, F], X\rangle; x : r\ \tau, \Gamma'; \Omega'}$$

T8 - Assign-Var

$$\frac{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e : r\ \tau \dashv \mathcal{H}'; \Gamma', x : r_{old}\ \tau; \Omega' \qquad x \notin \mathit{vars}(\mathcal{H}')}{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash x = e : r\ \tau \dashv \mathcal{H}'; \Gamma', x : r\ \tau; \Omega'}$$

T9 - Function-Application

$$\frac{\begin{array}{c}(\mathit{fn}, (\mathcal{H}; x_1' : r_1'\ \tau_1, \ldots, x_n' : r_n'\ \tau_n) \Rightarrow (\mathcal{H}'; \Gamma'; r_0', \tau_0)) \in \mathcal{F} \\ \overline{\Gamma \vdash x_i : r_i\ \tau_i} \qquad \overline{r_i' \mapsto r_i} \sqsubseteq \Phi_r \in \mathit{bijections}(\mathit{RegionNames}) \\ \Phi_x = \overline{x_i' \mapsto x_i} \in \mathit{bijections}(\mathit{VariableNames}) \qquad \Omega_{new} = \mathit{regs}(\mathcal{H}') - \mathit{regs}(\mathcal{H})\end{array}}{\Phi_x(\Phi_r(\mathcal{H})); \Gamma; \Omega; \mathrm{P} \vdash \mathit{fn}(x_1, \ldots, x_n) : r_0\ \tau_0 \dashv \Phi_x(\Phi_r(\mathcal{H}')); \Phi_x(\Phi_r(\Gamma')); \Omega \uplus \Phi_r(\Omega_{new})}$$

T10 - New-Loc

$$\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash \texttt{new-}\tau : r\ \tau \dashv \mathcal{H}, r^{\cdot}\langle\rangle; \Gamma; \Omega \uplus \{r\}$$

T11 - Declare-Var

$$\frac{\mathcal{H}; \Gamma, x : \bot\ \tau; \Omega; \cdot \vdash e : r\ \tau' \dashv \mathcal{H}'; \Gamma', x : r_{out}\ \tau; \Omega' \qquad x \notin \mathit{vars}(\mathcal{H}')}{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash \texttt{declare}\ x : \tau\ \texttt{in}\ \{e\} : r\ \tau' \dashv \mathcal{H}'; \Gamma'; \Omega'}$$

T12 - OPlus

$$\frac{\begin{array}{c}\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e_1 : r_1\ \tau_1 \dashv \mathcal{H}'; \Gamma'; \Omega' \\ \mathcal{H}'; \Gamma'; \Omega'; \mathrm{P}' \vdash e_2 : r_2\ \tau_2 \dashv \mathcal{H}''; \Gamma''; \Omega'' \qquad r_1 \in \mathit{regs}(\mathcal{H}'') \qquad (e_1, \mathrm{P}, \mathrm{P}')\ \texttt{respects-eval-order} \qquad \vdash \tau_1 \oplus \tau_2 : \tau'\end{array}}{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e_1 \oplus e_2 : r_{out}\ \tau' \dashv \mathcal{H}''', r_{out}^{\cdot}\langle\rangle; \Gamma'''; \Omega''' \uplus \{r_{out}\}}$$

T13 - If-Statement

$$\frac{\begin{array}{c}\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e_b : r_b\ \texttt{bool} \dashv \mathcal{H}'; \Gamma'; \Omega' \\ \mathcal{H}'; \Gamma'; \Omega'; \cdot \vdash e_t : r\ \tau \dashv \mathcal{H}''; \Gamma''; \Omega_t \qquad \mathcal{H}'; \Gamma'; \Omega'; \cdot \vdash e_f : r\ \tau \dashv \mathcal{H}''; \Gamma''; \Omega_f\end{array}}{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash \texttt{if}\ (e_b)\ \{e_t\}\ \texttt{else}\ \{e_f\} : r\ \tau \dashv \mathcal{H}''; \Gamma''; \Omega_t \cup \Omega_f}$$

T14 - While-Loop

$$\frac{\mathcal{H}; \Gamma; \Omega; \cdot \vdash e_b : r_b\ \texttt{bool} \dashv \mathcal{H}; \Gamma; \Omega' \qquad \mathcal{H}; \Gamma; \Omega'; \cdot \vdash e : r\ \tau \dashv \mathcal{H}; \Gamma; \Omega''}{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash \texttt{while}\ (e_b)\ \{e\} : r_u\ \texttt{unit} \dashv r_u^{\cdot}\langle\rangle, \mathcal{H}; \Gamma; \Omega'' \uplus \{r_u\}}$$

T15 - If-Disconnected

$$\frac{\begin{array}{c} r_x^{\cdot}\langle\rangle, r_y^{\cdot}\langle\rangle, \mathcal{H}; x : r_x\ \tau_x, y : r_y\ \tau_y, \Gamma; \Omega \uplus \{r_x, r_y\}; \cdot \vdash e_{succ} : r_{out}\ \tau_{out} \dashv \mathcal{H}'; \Gamma'; \Omega_{succ} \\ r^{\cdot}\langle\rangle, \mathcal{H}; x : r\ \tau_x, y : r\ \tau_y, \Gamma; \Omega; \cdot \vdash e_{fail} : r_{out}\ \tau_{out} \dashv \mathcal{H}'; \Gamma'; \Omega_{fail} \end{array}}{r^{\cdot}\langle\rangle, \mathcal{H}; x : r\ \tau_x, y : r\ \tau_y, \Gamma; \Omega; \mathrm{P} \vdash \texttt{if disconnected}\ (x, y)\ \texttt{in}\ \{e_{succ}\}\ \texttt{else}\ \{e_{fail}\} : r_{out}\ \tau_{out} \dashv \mathcal{H}'; \Gamma'; \Omega_{succ} \cup \Omega_{fail}}$$

T16 - Send

$$\frac{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e : r_e\ \tau \dashv \mathcal{H}', r_e^{\cdot}\langle\rangle; \Gamma'; \Omega'}{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash \mathsf{send}\text{-}\tau(e) : r\ \mathsf{unit} \dashv \mathcal{H}', r^{\cdot}\langle\rangle; \Gamma'; \Omega' \uplus \{r\}}$$

T17 - Receive

$$\frac{}{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash \mathsf{recv}\text{-}\tau() : r\ \tau \dashv \mathcal{H}, r^{\cdot}\langle\rangle; \Gamma; \Omega \uplus \{r\}}$$

T18 - None

$$\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash \mathsf{none}\ \tau : r\ \tau? \dashv \mathcal{H}, r^{\cdot}\langle\rangle; \Gamma; \Omega \uplus \{r\}$$

T19 - Some

$$\frac{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega'}{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash \mathsf{some}(e) : r\ \tau? \dashv \mathcal{H}'; \Gamma'; \Omega'}$$

T20 - Destruct-Option

$$\frac{\begin{array}{c} \mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e : r\ \tau? \dashv \mathcal{H}'; \Gamma'; \Omega' \qquad x \notin dom(\Gamma') \qquad \mathcal{H}'; \Gamma', x : r\ \tau; \Omega'; \cdot \vdash e_s : r_{out}\ \tau_{out} \dashv \mathcal{H}''; \Gamma'', x : r_{out}\ \tau; \Omega_s \\ \mathcal{H}'; \Gamma', x : \bot\ \tau; \Omega'; \cdot \vdash e_n : r_{out}\ \tau_{out} \dashv \mathcal{H}''; \Gamma'', x : r'_{out}\ \tau; \Omega_n \end{array}}{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash \texttt{let some}(x) = (e)\ \texttt{in}\ \{e_s\}\ \texttt{else}\ \{e_n\} : r\ \tau \dashv \mathcal{H}''; \Gamma''; \Omega_s \cup \Omega_n}$$

TS1 - Virtual-Transformation-Structural

$$\frac{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega' \qquad (\mathcal{H}'; \Gamma'; \Omega') \overset{\text{VIR}}{\rightsquigarrow} (\bar{\mathcal{H}}'; \bar{\Gamma}'; \bar{\Omega}') \qquad r \in regs(\bar{\mathcal{H}}')}{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e : r\ \tau \dashv \bar{\mathcal{H}}'; \bar{\Gamma}'; \bar{\Omega}'}$$

TS2 - Framing-Structural

$$\frac{\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega' \qquad (\mathcal{H}; \Gamma; \Omega) \underset{A}{\overset{\text{FRM}(e)}{\rightsquigarrow}} (\bar{\mathcal{H}}; \bar{\Gamma}; \bar{\Omega}) \qquad (\mathcal{H}'; \Gamma'; \Omega') \underset{A}{\overset{\text{FRM}(e)}{\rightsquigarrow}} (\bar{\mathcal{H}}'; \bar{\Gamma}'; \bar{\Omega}') \qquad r \in regs(\bar{\mathcal{H}}')}{\bar{\mathcal{H}}; \bar{\Gamma}; \bar{\Omega}; \mathrm{P} \vdash e : r\ \tau \dashv \bar{\mathcal{H}}'; \bar{\Gamma}'; \bar{\Omega}'}$$

2.5 Virtual Transformation Rules

V1 - Focus

$$(r^{\cdot}\langle\rangle, \mathcal{H}; x : r\ \tau, \Gamma; \Omega) \overset{\text{VIR}}{\rightsquigarrow} (r^{\cdot}\langle x^{\cdot}[]\rangle, \mathcal{H}; x : r\ \tau, \Gamma; \Omega)$$

V2 - Unfocus

$$(r^{\circ}\langle x^{\cdot}[], X\rangle, \mathcal{H}; \Gamma; \Omega) \overset{\text{VIR}}{\rightsquigarrow} (r^{\circ}\langle X\rangle, \mathcal{H}; \Gamma; \Omega)$$

V3 - Explore

$$(r^{\circ}\langle x^{\cdot}[F], X\rangle, \mathcal{H}; \Gamma; \Omega) \overset{\text{VIR}}{\rightsquigarrow} (r^{\circ}\langle x^{\cdot}[f \rightarrowtail r_f, F], X\rangle, r_f^{\cdot}\langle\rangle, \mathcal{H}; \Gamma; \Omega \uplus \{r_f\})$$

V4 - Retract

$$(r^{\circ}\langle x^{\circ'}[f \rightarrowtail r_f, F], X\rangle, r_f^{\cdot}\langle\rangle, \mathcal{H}; \Gamma; \Omega) \overset{\text{VIR}}{\rightsquigarrow} (r^{\circ}\langle x^{\circ'}[F], X\rangle, \mathcal{H}; \Gamma; \Omega)$$

V5 - Attach

$$(r_1^{\cdot}\langle X_1\rangle, r_2^{\circ}\langle X_2\rangle, \mathcal{H}; \Gamma; \Omega) \overset{\text{VIR}}{\rightsquigarrow} (r_2^{\circ}\langle X_1[r_1 \mapsto r_2], X_2[r_1 \mapsto r_2]\rangle, \mathcal{H}[r_1 \mapsto r_2]; \Gamma[r_1 \mapsto r_2]; \Omega)$$

V6 - Drop-Variable

$$\frac{x \notin vars(\mathcal{H})}{(\mathcal{H}; x : r\ \tau, \Gamma; \Omega) \overset{\text{VIR}}{\rightsquigarrow} (\mathcal{H}; \Gamma; \Omega)}$$

V7 - Drop-Region

$$(r^{\circ}\langle X\rangle, \mathcal{H}; \Gamma; \Omega) \overset{\text{VIR}}{\rightsquigarrow} (\mathcal{H}; \Gamma; \Omega)$$

V8 - Invalidate-Variable

$$\frac{x \notin vars(\mathcal{H})}{(\mathcal{H}; x : r\ \tau, \Gamma; \Omega) \overset{\text{VIR}}{\rightsquigarrow} (\mathcal{H}; x : \bot\ \tau, \Gamma; \Omega)}$$

V9 - Invalidate-Field

$$(r^{\circ}\langle x^{\circ'}[f \rightarrowtail r_f, F], X\rangle, \mathcal{H}; \Gamma; \Omega) \overset{\text{VIR}}{\rightsquigarrow} (r^{\circ}\langle x^{\circ'}[f \rightarrowtail \bot, F], X\rangle, \mathcal{H}; \Gamma; \Omega)$$

V10 - Fresh-Region

$$(\mathcal{H}; \Gamma; \Omega) \overset{\text{VIR}}{\rightsquigarrow} (r^{\cdot}\langle\rangle, \mathcal{H}; \Gamma; \Omega \uplus \{r\})$$

2.6 Framing Rules

F1 - Region-Framing

$$\frac{dom(\bar{\mathcal{H}}) \subseteq \bar{\Omega}}{(\mathcal{H}; \Gamma; \Omega) \underset{\cdot;\bar{\mathcal{H}},\bar{\Omega}}{\overset{\text{FRM}(e)}{\rightsquigarrow}} (\mathcal{H} \uplus \bar{\mathcal{H}}; \Gamma; \Omega \uplus \bar{\Omega})}$$

F2 - Region-Pinnedness-Framing

$$(r^{\dagger}\langle X\rangle, \mathcal{H}; \Gamma; \Omega) \underset{r;\cdot}{\overset{\text{FRM}(e)}{\rightsquigarrow}} (r^{\cdot}\langle X\rangle, \mathcal{H}; \Gamma; \Omega)$$

F3 - Tracked-Variable-Framing

$$\frac{dom(\bar{X}) \cap (NV(e) \cup dom(\Gamma)) = \emptyset}{(r^{\dagger}\langle X\rangle, \mathcal{H}; \Gamma; \Omega) \underset{r;\bar{X}}{\overset{\text{FRM}(e)}{\rightsquigarrow}} (r^{\dagger}\langle X \uplus \bar{X}\rangle, \mathcal{H}; \Gamma; \Omega)}$$

F4 - Variable-Pinnedness-Framing

$$(r^{\circ}\langle x^{\dagger}[F], X\rangle, \mathcal{H}; \Gamma; \Omega) \underset{r;x}{\overset{\text{FRM}(e)}{\rightsquigarrow}} (r^{\circ}\langle x^{\cdot}[F], X\rangle, \mathcal{H}; \Gamma; \Omega)$$

F5 - Field-Framing

$$(r^{\circ}\langle x^{\dagger}[F], X\rangle, \mathcal{H}; \Gamma; \Omega) \underset{r;x,\bar{F}}{\overset{\text{FRM}(e)}{\rightsquigarrow}} (r^{\circ}\langle x^{\dagger}[F \uplus \bar{F}], X\rangle, \mathcal{H}; \Gamma; \Omega)$$

F6 - Variable-Framing

$$(\mathcal{H}; \Gamma; \Omega) \underset{\cdot;\bar{\Gamma}}{\overset{\text{FRM}(e)}{\rightsquigarrow}} (\mathcal{H}; \Gamma \uplus \bar{\Gamma}; \Omega)$$

F7 - Consumed-Region-Framing

$$\frac{r \notin regs(\mathcal{H})}{(\mathcal{H}; \Gamma; \Omega) \underset{r;\_}{\overset{\text{FRM}(e)}{\rightsquigarrow}} (\mathcal{H}; \Gamma; \Omega)}$$

F8 - Attach-Pinned-Regions-Framing

$$(r_1^{\dagger}\langle X_1\rangle, r_2^{\dagger}\langle X_2\rangle, \mathcal{H}; \Gamma; \Omega) \underset{\cdot;r_1,r_2}{\overset{\text{FRM}(e)}{\rightsquigarrow}} (r_2^{\dagger}\langle X_1[r_1 \mapsto r_2], X_2[r_1 \mapsto r_2]\rangle, \mathcal{H}; \Gamma[r_1 \mapsto r_2]; \Omega)$$

2.7 Evaluation Rules

E1 - EVALUATION-CONTEXT-STEP

$$\frac{(d,h,s,e)\xrightarrow{\text{EVAL}}(d',h',s',e')}{(d,h,s,E[e])\xrightarrow{\text{EVAL}}(d',h',s',E[e'])}$$

E2 - VARIABLE-REF-STEP

$$\frac{s(x)=l \qquad l\in d}{(d,h,s,x)\xrightarrow{\text{EVAL}}(d,h,s,l)}$$

E3 - SEQUENCE-STEP

$$(d,h,s,l;e)\xrightarrow{\text{EVAL}}(d,h,s,e)$$

E4 - CONTEXTUAL-REFERENCE-STEP

$$\frac{(d,h,s,e)\xrightarrow{\text{EVAL}}(d',h',s',e') \qquad e\notin \mathit{VariableNames}}{(d,h,s,e.f)\xrightarrow{\text{EVAL}}(d,h,s,e'.f)}$$

E5A - FINAL-REFERENCE-STEP–VARIABLE

$$\frac{s(x)=l \qquad l,l_f\in d \qquad h_v(l)[f]=l_f}{(d,h,s,x.f)\xrightarrow{\text{EVAL}}(d,h,s,l_f)}$$

E5B - FINAL-REFERENCE-STEP–LOCATION

$$\frac{l,l_f\in d \qquad h_v(l)[f]=l_f}{(d,h,s,l.f)\xrightarrow{\text{EVAL}}(d,h,s,l_f)}$$

E6 - CONTEXTUAL-ASSIGNMENT-STEP

$$\frac{(d,h,s,e)\xrightarrow{\text{EVAL}}(d',h',s',e') \qquad e\notin \mathit{VariableNames}}{(d,h,s,e.f=l)\xrightarrow{\text{EVAL}}(d',h',s',e'.f=l)}$$

E7A - FINAL-ASSIGNMENT-STEP–VARIABLE

$$\frac{s(x)=l \qquad l,l_f\in d}{(d,h\uplus(l\mapsto(\tau,v)),s,x.f=l_f)\xrightarrow{\text{EVAL}}(d,h\uplus(l\mapsto(\tau,v[f\mapsto l_f])),s,l_f)}$$

E7B - FINAL-ASSIGNMENT-STEP–LOCATION

$$\frac{l,l_f\in d}{(d,h\uplus(l\mapsto(\tau,v)),s,l.f=l_f)\xrightarrow{\text{EVAL}}(d,h\uplus(l\mapsto(\tau,v[f\mapsto l_f])),s,l_f)}$$

E8 - ASSIGN-VAR-STEP

$$\frac{l\in d}{(d,h,s\uplus(x\mapsto l_{old}),x=l)\xrightarrow{\text{EVAL}}(d,h,s\uplus(x\mapsto l),l)}$$

E9 - FUNCTION-APPLICATION-STEP

$$\frac{\Lambda(fn)=(\lambda x_1',\ldots,x_n':e) \qquad \overline{x_i'\mapsto x_i}\sqsubseteq\Phi_x\in \mathit{bijections}(\mathit{VariableNames}) \qquad \Phi_x\restriction_{NV(e)-\overline{x_i'}}\ \texttt{is a safe substitution}}{(d,h,s,fn(x_1,\ldots,x_m))\xrightarrow{\text{EVAL}}(d,h,s,\Phi_x(e))}$$

E10 - NEW-LOC-STEP

$$\frac{(h_{new},l)=\mathit{construct\text{-}new}(h,\tau) \qquad d_{new}=\mathit{dom}(h_{new})}{(d,h,s,\mathsf{new}\text{-}\tau)\xrightarrow{\text{EVAL}}(d\uplus d_{new},h\uplus h_{new},s,l)}$$

E11 - Declare-Var-Step

$$\frac{}{(d, h, s, \texttt{declare}\ x : \tau\ \texttt{in}\ \{e\}) \xrightarrow{\text{EVAL}} (d, h, s[x \mapsto \bot], e)}$$

E12 - OPlus-Step

$$\frac{l_1, l_2 \in d \qquad l_3 \notin dom(h) \qquad [[\oplus]](h_v(l_1), h_v(l_2)) = v_3 \qquad \vdash h_\tau(l_1) \oplus h_\tau(l_2) : \tau'}{(d, h, s, l_1 \oplus l_2) \xrightarrow{\text{EVAL}} (d \uplus \{l_3\}, h \uplus (l_3 \mapsto (\tau', v_3)), s, l_3)}$$

E13a - If-True-Step

$$\frac{h_v(l) = \texttt{true} \qquad l \in d}{(d, h, s, \texttt{if}\ (l)\ \{e_t\}\ \texttt{else}\ \{e_f\}) \xrightarrow{\text{EVAL}} (d, h, s, e_t)}$$

E13b - If-False-Step

$$\frac{h_v(l) = \texttt{false} \qquad l \in d}{(d, h, s, \texttt{if}\ (l)\ \{e_t\}\ \texttt{else}\ \{e_f\}) \xrightarrow{\text{EVAL}} (d, h, s, e_f)}$$

E14 - While-Loop-Step

$$\frac{}{(d, h, s, \texttt{while}\ (e_b)\ \{e\}) \xrightarrow{\text{EVAL}} (d, h, s, \texttt{if}\ (e_b)\ \{e; \texttt{while}\ (e_b)\ \{e\}\}\ \texttt{else}\ \{\texttt{new-unit}()\})}$$

E15a - If-Disconnected-Success-Step

$$\frac{\textit{tracked-set}(r^{\cdot}\langle\rangle; x : r\ \tau; \cdot; h, s) \cap \textit{tracked-set}(r^{\cdot}\langle\rangle; y : r\ \tau; \cdot; h, s) = \emptyset}{(d, h, s, \texttt{if disconnected}(x, y)\ \{e_{succ}\}\ \texttt{else}\ \{e_{fail}\}) \xrightarrow{\text{EVAL}} (d, h, s, e_{succ})}$$

E15b - If-Disconnected-Failure-Step

$$\frac{\textit{tracked-set}(r^{\cdot}\langle\rangle; x : r\ \tau; \cdot; h; s) \cap \textit{tracked-set}(r^{\cdot}\langle\rangle; y : r\ \tau; \cdot; h; s) \neq \emptyset}{(d, h, s, \texttt{if disconnected}(x, y)\ \{e_{succ}\}\ \texttt{else}\ \{e_{fail}\}) \xrightarrow{\text{EVAL}} (d, h, s, e_{fail})}$$

E18 - None-Step

$$(d, h, s, \texttt{none}\ \tau) \xrightarrow{\text{EVAL}} (d \uplus \{l\}, h \uplus \{l \mapsto (\tau?, \cdot)\}, s, l)$$

E19 - Some-Step

$$\frac{l \in d \qquad h(l) = (\tau, v)}{(d, h, s, \texttt{some}(l)) \xrightarrow{\text{EVAL}} (d \uplus \{l'\}, h \uplus \{l' \mapsto (\tau?, v)\}, s, l')}$$

E20a - Let-Some-Step

$$\frac{h(l) = (\tau?, v) \qquad v \neq \cdot}{(d, h, s, \texttt{let some}(x) = (l)\ \texttt{in}\ \{e_s\}\ \texttt{else}\ \{e_n\}) \xrightarrow{\text{EVAL}} (d \uplus \{l'\}, h \uplus \{l' \mapsto (\tau, v)\}, s[x \mapsto l'], e_s)}$$

E20b - Let-None-Step

$$\frac{h(l) = (\tau?, \cdot)}{(d, h, s, \texttt{let some}(x) = (l)\ \texttt{in}\ \{e_s\}\ \texttt{else}\ \{e_n\}) \xrightarrow{\text{EVAL}} (d, h, s[x \mapsto \bot], e_n)}$$

2.8 Progress and Preservation

2.8.1 Preliminary Definitions.

Definition 2.1 (Non-Blocking Expression). We say that an expression e is *non-blocking* if its redex (through evaluation contexts $E^*[]$) is not a `send` (T15 - If-Disconnected) or a `recv` (T16 - Send).

2.8.2 Main Theorems. This subsection gives statements of Progress and Preservation, which are largely standard and unsurprising except that 1) Progress applies only to *non-blocking* expressions, and 2) Preservation provides static output contexts after the step that possibly differ from those used before the step by a renaming of invalid region names used in $\mathcal{H}$.

Theorem 2.2 (Progress). *Let $e \notin$ LocationNames be a well-typed non-blocking expression with a sound dynamic configuration, i.e. $\mathcal{H}; \Gamma; \Omega; P \vdash e : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega'$ and $(\mathcal{H}, \Gamma, \Omega, P, d, h, s)$ satisfies I0 - Sound-Configuration. Then there exists an expression e' with dynamic configuration (d', h', s') such that $(d, h, s, e) \xrightarrow{\text{EVAL}} (d', h', s', e')$.*

Theorem 2.3 (Preservation). *Let e be a well-typed expression with a sound dynamic configuration and a valid step, i.e. $\mathcal{H}; \Gamma; \Omega; P \vdash e : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega'$, $(\mathcal{H}, \Gamma, \Omega, P, d, h, s)$ satisfies I0, and $(d, h, s, e) \xrightarrow{\text{EVAL}} (d', h', s', e')$. Then there exist contexts $(\bar{\mathcal{H}}, \bar{\Gamma}, \bar{\Omega}, \bar{P}, \bar{\mathcal{H}}', \bar{\Gamma}', \Omega_{new})$ such that $\bar{\mathcal{H}}; \bar{\Gamma}; \bar{\Omega}; \bar{P} \vdash e' : r\ \tau \dashv \bar{\mathcal{H}}'; \bar{\Gamma}'; \Omega' \uplus \Omega_{new}$ and $(\bar{\mathcal{H}}, \bar{\Gamma}, \bar{\Omega}, \bar{P}, d', h', s')$ satisfies I0. Further, $\bar{P}$ agrees with P on all output-tracked regions, i.e. $P \restriction_{P_r^{-1}(regs(\mathcal{H}'))} \sqsubseteq \bar{P}$, and $\bar{\mathcal{H}}', \bar{\Gamma}'$ differ from $\mathcal{H}', \Gamma'$, only by a region renaming that preserves all names tracked in $\mathcal{H}'$, i.e. $\bar{\mathcal{H}}', \bar{\Gamma}' = \Phi_r(\mathcal{H}'), \Phi_r(\Gamma')$, where $\Phi_r \restriction_{regs(\mathcal{H}')} = Id$*

2.8.3 Key Lemmas.

Definition 2.4 (Typing Deriviation Normal Form). We say that a typing derivation is in *normal form* if all instances of framing structural rules occur around function application. In other words, all instances of TS2 - Framing-Structural derive their premises only from other instance of TS2 or from T9 - Function-Application.

Lemma 2.5 (Normalization of Typing Derivations). *Given a typing derivation for the judgement $\mathcal{H}; \Gamma; \Omega; P \vdash e : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega'$, there exists a typing derivation in normal form that derives the same judgement up to a renaming of invalid regions in the output contexts $\mathcal{H}', \Gamma', \Omega'$ and uses at most as many T rule instances as the original.*

Proof. We will prove this by induction on the depth of the typing derivation in T (including TS) rules. If this depth is 1, then that single rule cannot be a TS rule, so the tree has no TS2 instances and we are done. Now assume the lemma proven for trees of depth at most n, and consider a typing derivation of depth $n + 1$, applying the lemma inductively to the (max depth n) typing derivations of all premises of the root to ensure they satisfy its post-conditions. Now assume the root of our tree is an instance of TS2 (as otherwise the inductive step is trivial) and relies on a premise derived from some rule $\boxed{X}$. If $\boxed{X}$ = TS2 or T9 then we are also done, as the lemma allows for TS2 instances whose premises are such $\boxed{X}$, leaving two cases: $\boxed{X}$ = TS1 - Virtual-Transformation-Structural and $\boxed{X}$ = T<n> for $n \neq 9$. We proceed to perform this casework:

- TS1: This case, in which a TS2 framing rule is applied to the output of a TS1 virtual transformation rule, is the most interesting, as most of the context transformations take place in these two categories of rules. We will first split into two cases on whether or not F7 - Consumed-Region-Framing derived the $\overset{\text{FRM}(e)}{\underset{A}{\rightsquigarrow}}$ premise of the root TS2 rule:
  - F7: Let $\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e : r_e\ \tau_e \dashv \mathcal{H}'; \Gamma'; \Omega'$ be the premise to the TS1 instance, and $(\bar{\mathcal{H}}'; \bar{\Gamma}'; \bar{\Omega}')$ be the output contexts from the conclusion of the TS1 instance and from both the premise and conclusion of the TS2 instance (all three triples of contexts we know to be equal from our assumptions of the structure of this case and of the output of F7). From the premise of F7, we know that our framing transformation is of the form $\overset{\text{FRM}(e)}{\underset{r;\_}{\rightsquigarrow}}$ for some region $r \notin regs(\bar{\mathcal{H}}')$. If $r \notin regs(\mathcal{H}')$ then we are done, as the framing rule will be a no-op on the outputs when applied to $\mathcal{H}', \Gamma', \Omega'$ directly as well, and the swap may be performed to yield the same conclusion

from the pair of rules. If $r \in regs(\mathcal{H}')$, then one of V4 - Retract, V5 - Attach, or V7 - Drop-Region derived our the premise of TS1. To reason that even after the swap, the same $(\bar{\mathcal{H}}', \bar{\Gamma}, \bar{\Omega}')$ will be output by the $\overset{\text{VIR}}{\rightsquigarrow}$ transformation, it suffices to show that the LHS matching of the $\overset{\text{VIR}}{\rightsquigarrow}$ transformation still applies, as any changes to the contexts by our framing rule necessarily happen within region r (this is the commonality of all F rules that derive transformations of the form $\underset{r;\_}{\overset{\text{FRM}(e)}{\rightsquigarrow}}$), and these changes will be nullified by the $\overset{\text{VIR}}{\rightsquigarrow}$ transformation dropping region r. V7 performs no restrictive matching whatsoever, so it is always safe to swap. V4 and V5 require that the region be unpinned, and V4 further requires that it be empty. From the applicability to $\mathcal{H}', \Gamma', \Omega'$, we know that these conditions hold there, and we observe that no F rule derives a transformation that pins an existing region, or that introduces variables to an existing unpinned region. Thus the V rule is always safe to apply after the F rule's $\underset{r;\_}{\overset{\text{FRM}(e)}{\rightsquigarrow}}$ transformation. Our last obligation is to show that the F rule can still always be applied to $\mathcal{H}', \Gamma', \Omega'$. Because of V7, we cannot assume that the region r is unpinned or empty in $\mathcal{H}'$. Luckily, for each rule that derives a $\underset{r;\_}{\overset{\text{FRM}(e)}{\rightsquigarrow}}$ transformation, namely F2 - Region-Pinnedness-Framing, F3 - Tracked-Variable-Framing, F4 - Variable-Pinnedness-Framing and F5 - Field-Framing, the knowledge that it has been succesfully applied to the context $(\mathcal{H}; \Gamma; \Omega)$, combined with the assumed typing relation of $\mathcal{H}, \Gamma, \Omega$ to $\mathcal{H}', \Gamma', \Omega'$, is sufficient to conclude that it can be succesfully applied to $(\mathcal{H}'; \Gamma'; \Omega')$. This logic is captured in lemmas 2.11 and 2.12. Lemma 2.11 argues that pinned regions from the LHS of typing judgements can only appear pinned on the RHS, and regions with pinned variables in the LHS can only appear with the same pinned variables on the RHS. This covers the majority of the proof obligation as it gives the soundness of the necessary LHS pattern matching for the presence of pinned regions and variables performed by rules F2-F5. Lemma 2.12 covers the remaining obligation by demonstrating preservation of the $dom(\Gamma)$-disjointness condition of F3, noting that the $NV(e)$-disjointness condition is independent of the static contexts. This concludes our argument that TS2 instances derived from F7 can be swapped with arbitrary TS1 instances. In effect, this case captures the entirety of the logic required for function application that partially matches on regions that are dropped within its body.

NOT F7 - Consumed-Region-Framing: Here, we perform further casework on the V rule that could have derived the premise of our TS1 - Virtual-Transformation-Structural application. In particular, we show that if $(\mathcal{H}_0, \Gamma_0, \Omega_0) \overset{\text{VIR}}{\rightsquigarrow} (\mathcal{H}_1, \Gamma_1, \Omega_1) \underset{A}{\overset{\text{FRM}(e)}{\rightsquigarrow}} (\mathcal{H}_2, \Gamma_2, \Omega_2)$ is derivable, then so is $(\mathcal{H}_0, \Gamma_0, \Omega_0) \underset{A}{\overset{\text{FRM}(e)}{\rightsquigarrow}} (\mathcal{H}_1', \Gamma_1', \Omega_1') \overset{\text{VIR}}{\rightsquigarrow} (\mathcal{H}_2, \Gamma_2, \Omega_2)$ for some $(\mathcal{H}_1', \Gamma_1', \Omega_1')$. In other words, all framing transformations $\underset{A}{\overset{\text{FRM}(e)}{\rightsquigarrow}}$ applied to contexts that are the output of a $\overset{\text{VIR}}{\rightsquigarrow}$ transformation may be freely exchanged with that $\overset{\text{VIR}}{\rightsquigarrow}$ transformation to yield the same output. The only premise of TS1 and TS2 - Framing-Structural besides matching on context transformations $(r \in regs(\bar{\mathcal{H}}'))$ is shared between the two, so proving exchangeability of the context transformations suffices to show we can swap the instances of TS1 and TS2 in the typing derivation, concluding this case by inductive application to the depth $< n$ tree now rooted at TS2 after the swap.

V1 - Focus: No F rule becomes inapplicable if a variable of the form $x^{\cdot}[]$ is removed from $\mathcal{H}$, so we can apply any $\underset{A}{\overset{\text{FRM}(e)}{\rightsquigarrow}}$ transformation to the pre V1 configuration as easily as to the result of its $\overset{\text{VIR}}{\rightsquigarrow}$ transformation. Whatever the F rule chosen to derive the $\underset{A}{\overset{\text{FRM}(e)}{\rightsquigarrow}}$ transformation, it cannot add a variable to an unpinned region, so if r was empty before the $\underset{A}{\overset{\text{FRM}(e)}{\rightsquigarrow}}$ application it is also empty afterwards, and thus applying the $\overset{\text{VIR}}{\rightsquigarrow}$

transformation after the $\overset{\text{FRM}(e)}{\underset{A}{\rightsquigarrow}}$ transformation will have the same effect as applying it before: the region r will now be of the form $r^{\cdot}\langle x^{\cdot}[]\rangle$, and the virtual transformation will have no other effects.

- V2 - Unfocus: The only F rule that could become inapplicable if its transformation applied to the pre V2 configuration is F3. Notably, this rule requires r to be pinned, so, considering the well-typedness judgment V2 is originally applied to, if it were not focused in the input contexts it would not be focused in the output contexts, which it must be for V2 to be applicable. Conversely, if it were focused in the input contexts then the framing rule F3 could not symmetrically expand by it, so we can dismiss the only potentially unsafe case.
- V3 - Explore and V4 - Retract: Largely the same idea as V1 - Focus and V2, noting additionally that these rules and F8 - Attach-Pinned-Regions-Framing can be freely exchanged due to their introduction/elimination only of unpinned regions, and F8's action only on pinned regions.
- V5 - Attach: This is the most interesting case, as the transformations here do *not* exactly exchangeable. Rather we must invoke the "untracked region renaming" condition of the lemma. In particular, the region being attached, r_1, can be introduced into the output $\mathcal{H}$ or Γ by F1 - Region-Framing, F3, or F5 - Field-Framing as a field target, or by F3 into Γ. When the V5 instance is applied before the F instance, this will function only to introduce an invalid variable or field to the output contexts. However, if the F rule is applied first than any such fields or variables will be valid with regionality r_2. This transformation is exactly described as a "region renaming that preserves all names tracked", as r_1 is not tracked in the output contexts. The final case for exchangability is V5 with F8, in which we note that (in the hard case) the 3 regions being acted upon by these 2 transformations can be attached in any order to yield the same result, provided both attachs do not change direction, which they do not.
- V6 - Drop-Variable: Exchangeability could fail only if the F rule introduces x to $\mathcal{H}$ or Γ, which it cannot do because x is necessarily in the domain of Γ at the input contexts V6 is originally applied to, making F3 inapplicable.
- V7 - Drop-Region: Exchangeability could fail only if the F rule introduces r to $\mathcal{H}$, which it cannot do because r is already in Ω and F1 is the only candidate, which symmetrically expands $\mathcal{H}$ and Ω.
- V8 - Invalidate-Variable: Exchangeability could fail only if the F rule alters the binding of x in Γ, or introduces it to $\mathcal{H}$. The former cannot happen because no F rule removes a binding from Γ or alters one, and the latter cannot happen because F3 is the only rule that can add variables to $\mathcal{H}$, and it is only applicable when those variables are not present in Γ, as x is.
- V9 - Invalidate-Field: Exchangeability could fail only if the F rule alters or removes the tracking of the field f in $\mathcal{H}$, which no F rule does.

We can conclude that no matter which V rule derived the $\overset{\text{VIR}}{\rightsquigarrow}$ transformation, it can be exchanged with any $\overset{\text{FRM}(e)}{\underset{A}{\rightsquigarrow}}$ transformation, concluding the case for TS1 - Virtual-Transformation-Structural and TS2 - Framing-Structural swapping.

- T1 - Location-Ref: No framing rules remove regions from $\mathcal{H}$, so T1 can be directly applied to yield the conclusion of TS2.
- T2 - Variable-Ref: Similar to the above, no framing rule removes regions from $\mathcal{H}$ or variables from Γ, so T2 can be applied directly to yield the conclusion of TS2.
- T3 - Sequence: We can easily push the framing rule above this rule to the level of the subexpressions, and apply the lemma inductively to obtain desirable typing derivations for the subexpressions, which possibly differ by the

untracked region renaming - but such a change to the input context of the left subexpression will not invalidate its typing judgment, as no rule matches for invalidity, only against it.

T4 - Non-Isolated-Field-Reference: Trivially swappable - no framing rule affects the static knowledge of types' fields

T5 - Isolated-Field-Reference: Framing rules only expand $\mathcal{H}$, so moving any application to before the check for appropriate field tracking here will not invalidate that check.

T6 - Non-Isolated-Field-Assignment: Logic from T3 - Sequence and T4 suffices.

T7 - Isolated-Field-Assignment: As noted in T5, the check for appropriate tracking cannot be invalidated by a framing transformation.

T8 - Assign-Var: The only way that this rule could be invalidate by swapping up the framing application is if x were introduced to $\mathcal{H}'$, which cannot happen because it is already present in the output Γ necessarily, and F3 - Tracked-Variable-Framing is the only candidate bad actor, which requires symmetric expansion.

T9 - Function-Application: Case is skipped - we accept framing rules on function applications.

T10 - New-Loc: Framing cannot add r to Ω, so we can replace it with direct application of T10.

T11 - Declare-Var: The only candidate framing rule which could invalidate the premises here is F3, which is explicitly banned from expanding by a variable x when it syntactically occurs in the expression being typechecked, as it does here.

T12 - OPlus-T14 - While-Loop: The same subexpression threading logic from the T3 case suffices.

T15 - If-Disconnected: Unpinnedness of the source region of the variables being split prevents F3 from acting, all other framing rules may be applied to the subexpressions with the same effect.

T16 - Send: Unpinnedness and presence of relevant region names in Ω prevent any possible premise invalidation by an F rule.

T17 - Receive: Same as T10.

T18 - None: Same as T17.

T19 - Some: Same as T16.

T20 - Destruct-Option: Arguments from T11 and T14 suffice.

□

This proof provides local rewrite rules to convert any typing derivation into *normal form* , and will be of key importance in the proofs of Progress and Preservation, as it allows us to reason about typing derivations without worrying about arbitrarily pervasive framing.

Lemma 2.6 (Virtual Transformation Dynamic Soundness). *Soundness of dynamic configurations is preserved under virtual transformations, i.e. given* I0$(\mathcal{H}, \Gamma, \Omega, P, d, h, s)$ *and* $(\mathcal{H}; \Gamma; \Omega) \overset{\text{VIR}}{\rightsquigarrow} (\mathcal{H}'; \Gamma'; \Omega')$*, we can conclude* I0$(\mathcal{H}', \Gamma', \Omega', P, d, h, s)$.

Proof. We consider each invariant comprising the definition of I0, and prove that it is preserved by the $\overset{\text{VIR}}{\rightsquigarrow}$ transformation, where necessary performing casework on the V rule that derived the transformation.

I1 - Reservation-Sufficiency and I2 - Tree-Of-Untracked-Regions: To show preservation of these two invariants, we argue that *outward-paths*$(\mathcal{H}', \Gamma', \mathrm{P}, h, s)$ has an image in *outward-paths*$(\mathcal{H}, \Gamma, \mathrm{P}, h, s)$ under a map Φ that preserves target locations l, and maps distinct (r, L) pairs to distinct (r, L) pairs. We will then argue that the existence of such an embedding suffices to show preservation of I1 and I2. By casework:

V1 - Focus and V2 - Unfocus: For these transformations, the *outward-paths* are unchanged, as the set of untracked fields and the *live-set* are both unchanged by the addition or removal of a focused variable.

V3 - Explore: Let $O = (r_o \overset{L_o}{\dashrightarrow} l_o) \in$ *outward-paths*$(\mathcal{H}', \Gamma', \mathrm{P}, h, s)$ If r_o is not the target r_f of the explore then $O \in$ *outward-paths*$(\mathcal{H}, \Gamma, \mathrm{P}, h, s)$. Otherwise, let $\Phi(O) = (r \overset{l.f, L_o}{\dashrightarrow} l_o)$, where r is the source region of the explore and $l.f$ is the field being explored. Since $l.f$ is tracked in $\mathcal{H}'$, it can be contained in no other members of *outward-paths*$(\mathcal{H}', \Gamma', \mathrm{P}, h, s)$, so non-collision as specified above is guaranteed.

V4 - Retract: Let $O = (r_o \overset{L_o}{\dashrightarrow} l_o) \in$ *outward-paths*$(\mathcal{H}', \Gamma', \mathrm{P}, h, s)$ If r_o is not the source r of the retract, or if L_0 does not begin with $l.f$, the field being retracted, then $O \in$ *outward-paths*$(\mathcal{H}, \Gamma, \mathrm{P}, h, s)$. Otherwise, let $L_0 = l.f, L_0'$, and let $\Phi(O) = (r_f \overset{L_o'}{\dashrightarrow} l_o)$, where r_f is the target region of the retract. Since r_f is untracked in $\mathcal{H}'$, it can be the start of no other members of *outward-paths*$(\mathcal{H}', \Gamma', \mathrm{P}, h, s)$, and non-collision as specified above is guaranteed.

V5 - Attach: Here is the map is fairly straightforwards. *outward-paths*$(\mathcal{H}', \Gamma', \mathrm{P}, h, s)$ is just *outward-paths*$(\mathcal{H}, \Gamma, \mathrm{P}, h, s)$ with all occurrences of region r_1 (source of the attach) as a start replaced with r_2 (target of the attach). This mapping is injective, as two *outward-paths* that differed only in start region would violate I2$(\mathcal{H}, \Gamma, \mathrm{P}, h, s)$, so it is invertible, and its inverse is exactly the map that we seek to construct for this case.

V6 - Drop-Variable, V7 - Drop-Region, V8 - Invalidate-Variable, V9 - Invalidate-Field: These transformations do not change the set of *outward-paths* except by possibly shrinking the set of *live-roots*, which would yield *outward-paths*$(\mathcal{H}', \Gamma', \mathrm{P}, h, s) \subseteq$ *outward-paths*$(\mathcal{H}, \Gamma, \mathrm{P}, h, s)$, admitting a clear embedding.

We now know that, regardless of the V rule that derived our $\overset{\text{VIR}}{\rightsquigarrow}$ transformation, *outward-paths*$(\mathcal{H}', \Gamma', \mathrm{P}, h, s)$ can be embedded into *outward-paths*$(\mathcal{H}, \Gamma, \mathrm{P}, h, s)$ by a map that preserves target locations and preserves distinctness of (r, L) pairs. The former condition on this map suffices to guarantee that *live-set*$(\mathcal{H}', \Gamma', \mathrm{P}, h, s) \subseteq$ *live-set*$(\mathcal{H}, \Gamma, \mathrm{P}, h, s)$, and thus gaurantee preservation of I1 - Reservation-Sufficiency. Considering also the latter condition of the map, we see that any negative examples to I2 present in the *outward-paths*$(\mathcal{H}', \Gamma', \mathrm{P}, h, s)$ would also have been present in *outward-paths*$(\mathcal{H}, \Gamma, \mathrm{P}, h, s)$. We can conclude that both I1 and I2 are preserved by $\overset{\text{VIR}}{\rightsquigarrow}$ transformations.

I3 - Heap-Closure: This invariant relies only on h, which is not acted upon by $\overset{\text{VIR}}{\rightsquigarrow}$ transformations, so no work need be done in this case.

I4 - Location-Type-Consistency: This invariant is preserved under changes to $\mathcal{H}$ and Γ as long as the types in Γ of variables does not change, and no variables exist in Γ' whose region is tracked in $\mathcal{H}'$ that did not exist in Γ with region tracked in $\mathcal{H}$. No $\overset{\text{VIR}}{\rightsquigarrow}$ transformation produces such a variable, so we can conclude this invariant is preserved.

I5 - Variable-Region-Consistency: This invariant is preserved under changes to $\mathcal{H}$ and Γ as long as any variable that appears tracked in $\mathcal{H}'$ appears in Γ' with the same region. This can only be violated if a new variable is introduced as tracked in $\mathcal{H}'$ that was not tracked in $\mathcal{H}$, or if an existing variable tracked in $\mathcal{H}$ has its region changed in Γ'. The former is only a concern for V1 - Focus, as it is the only $\overset{\text{VIR}}{\rightsquigarrow}$ to introduce a tracked variable, and it explicitly ensures the necessary condition on Γ. The latter is only a concern for V5, which alter $\mathcal{H}$ and Γ in parallel so is no cause for concern, and V6 and V8, which explicitly check that the variable being altered in Γ does not appear in $\mathcal{H}$.

I6 - Focus-Non-Aliasing: The only $\overset{\text{VIR}}{\rightsquigarrow}$ transformation under which this invariant could possibly be invalidated is V1, which introduces a new tracked variable to a region which previously had no tracked variables. This variable's location l with region r is in the *live-roots*, so $(r \overset{\cdot}{\dashrightarrow} l) \in$ *outward-paths*$(\mathcal{H}, \Gamma, \mathrm{P}, h, s)$. If some other tracked variable x' appearing in another region r' of $\mathcal{H}$ were mapped to l by s, then by I5$(\mathcal{H}, \Gamma)$, $\Gamma_r(x') = r'$ so (l, r') would also be a live root and $(r' \overset{\cdot}{\dashrightarrow} l) \in$ *outward-paths*$(\mathcal{H}, \Gamma, \mathrm{P}, h, s)$, which contradicts I2$(\mathcal{H}, \Gamma, \mathrm{P}, h, s)$. Thus no such x' can exist, and we can conclude this invariant is preserved even under V1.

I7 - Region-Names-Bounding: The only $\overset{\text{VIR}}{\rightsquigarrow}$ transformation to introduce a fresh region name is V3 - Explore, and it also ensures that this name shows up in Ω', so this invariant is preserved.

Having shown that all invariants I1 - Reservation-Sufficiency-I7 are preserved under $\overset{\text{VIR}}{\rightsquigarrow}$ transformations, we can conclude our proof of this lemma. □

2.8.4 Side Lemmas.

Lemma 2.7 (Virtual Transformation Renaming Invariance). *For any transformation* $(\mathcal{H}; \Gamma; \Omega) \overset{\text{VIR}}{\rightsquigarrow} (\mathcal{H}'; \Gamma'; \Omega')$ *and contexts* $\bar{\mathcal{H}}, \bar{\Gamma}, \bar{\Omega}$ *that differ from* $\mathcal{H}, \Gamma, \Omega$ *only by a renaming of untracked regions to tracked regions, then there exist* $\bar{\mathcal{H}}', \bar{\Gamma}', \bar{\Omega}'$ *that differ from* $\mathcal{H}', \Gamma', \Omega'$ *only by such a renaming, and for which* $(\bar{\mathcal{H}}; \bar{\Gamma}; \bar{\Omega}) \overset{\text{VIR}}{\rightsquigarrow} (\bar{\mathcal{H}}'; \bar{\Gamma}'; \bar{\Omega}')$.

Proof. Any V rule that names a region forces that region to be tracked, and no V rule relies on a region being untracked. In all cases, it is clear to see that the same partial function that was applied to $(\mathcal{H}; \Gamma; \Omega)$ by the original $\overset{\text{VIR}}{\rightsquigarrow}$ transformation to $(\mathcal{H}'; \Gamma'; \Omega')$ may also be applied to $\bar{\mathcal{H}}, \bar{\Gamma}, \bar{\Omega}$ and yield $\bar{\mathcal{H}}', \bar{\Gamma}', \bar{\Omega}'$ that differ from $(\mathcal{H}', \Gamma', \Omega')$ only through the same renaming of untracked regions. □

Lemma 2.8 (Well-Typedness Renaming Invariance). *Given a well-typed expression* $\mathcal{H}; \Gamma; \Omega; P \vdash e : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega'$*, contexts* $\bar{\mathcal{H}}, \bar{\Gamma}, \bar{\Omega}$ *that differ from* $\mathcal{H}, \Gamma, \Omega$ *only by a renaming of untracked regions to tracked regions, and* $\bar{\mathrm{P}}$ *that exactly preserves all locations of egionality tracked in* $\mathcal{H}$*, then there exist* $\bar{\mathcal{H}}', \bar{\Gamma}', \bar{\Omega}'$ *that differ from* $(\mathcal{H}', \Gamma', \Omega')$ *only by such a renaming, and for which* $\bar{\mathcal{H}}; \bar{\Gamma}; \bar{\Omega}; \bar{P} \vdash e : r\ \tau \dashv \bar{\mathcal{H}}'; \bar{\Gamma}'; \bar{\Omega}'$.

Proof. The proof is by induction, and considering the root rule T rule application, most cases follow from simple inspection. T1 - Location-Ref is notable, as it relies on the preservation of locations in P when their region is tracked, and rules such as T10 - New-Loc that introduce a fresh region name are notable because we must argue that the same fresh region name may still be chosen after the renaming in input contexts. We can note this because the renaming is to a name that is already tracked, and thus already present in Ω, which prevents it from being chosen by rules such as T10 as fresh. The case for TS1 - Virtual-Transformation-Structural reduces to lemma 2.7, and the case for TS2 - Framing-Structural requires verifying that the renaming does not affect the matching of any $\underset{A}{\overset{\text{FRM}(e)}{\rightsquigarrow}}$ transformations, which it does not. There are no remaining notabl cases, and we can conclude that the renaming of untracked regions is safe and does not invalidate well-typedness. □

Lemma 2.9 (Regionality Tracking). *Given a well-typed expression* $\mathcal{H}; \Gamma; \Omega; P \vdash e : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega'$*,* $\mathcal{H}'$ *tracks* r.

Proof. Trivial induction on the typing derivation. □

Lemma 2.10 (Safe P Expansion). *If* $(r \overset{\cdot}{\dashrightarrow} l) \in$ *outward-paths*$(\mathcal{H}, \Gamma, P, h, s)$*, and* $h_\tau(l) = \tau$*, then* I0$(\mathcal{H}, \Gamma, \Omega, P, d, h, s) \Rightarrow$ I0$(\mathcal{H}, \Gamma, \Omega, \{l : r\ \tau\} \cup P, d, h, s)$.

Proof. The invariants I1-I7 depend on P in three ways, each of which we will argue is preserved. First, I7 ensures that the region names in *range*(P_r) are contained in Ω. Since $(r \overset{\cdot}{\dashrightarrow} l)$ is an outward path, there is no way that region r does not already appear in $\mathcal{H}$, P, or Γ, so this dependence is preserved. Second, I4 - Location-Type-Consistency ensures that the types given to locations by P agree with the types given by h. Preservation of this dependence is explicitly ensured as a premise. Finally, I1 and I2 - Tree-Of-Untracked-Regions depend on P through the set of *outward-paths*, which we will argue does not change. First we observe that $\textit{live-roots}(\mathcal{H}, \Gamma, \{l : r\ \tau\} \cup \mathrm{P}, h, s) = \{(l, r)\} \cup \textit{live-roots}(\mathcal{H}, \Gamma, \mathrm{P}, h, s)$. Now, we observe that there must have been a fully non-isolated sequence in h from some other $(l', r) \in \textit{live-roots}(\mathcal{H}, \Gamma, \mathrm{P}, h, s)$ in order to have $(r \overset{\cdot}{\dashrightarrow} l) \in \textit{outward-paths}(\mathcal{H}, \Gamma, \mathrm{P}, h, s)$, so any sequence originating at l could be extended to originate at l'. Since sequences beginning at the new live root l were the only ones that could have yielded elements of $\textit{outward-paths}(\mathcal{H}, \Gamma, \{l : r\ \tau\} \cup \mathrm{P}, h, s,) - \textit{outward-paths}(\mathcal{H}, \Gamma, \mathrm{P}, h, s)$, but any such sequences yielded some $(r \overset{L_s}{\dashrightarrow} l_s)$ that was already present before the addition of (l, r) as a live root, we can conclude that the set of *outward-paths* does not grow, and so the invariants I1 and I2 are trivially preserved. This concludes our argument that I0 is preserved under addition of locations to P as qualified in the hypotheses. ◻

Lemma 2.11 (Pinnedness Permanence). *Let* $\mathcal{H}; \Gamma; \Omega; P \vdash e : r_e\ \tau_e \dashv \mathcal{H}'; \Gamma'; \Omega'$. *If* $r^{\dagger}\langle X \rangle \in \mathcal{H}$ *and* $r^{\circ}\langle X' \rangle \in \mathcal{H}'$, *then* $\circ = \dagger$. *Further, if* $x^{\dagger}[F] \in X$, *then* $x^{\dagger}[F'] \in X'$ *for some* F'.

Proof. Clear from inspection of the conclusions of all typing rules, including inspection of the transformations of V and F rules, and including structural induction. ◻

Lemma 2.12 (Γ Non-Growth). *Let* $\mathcal{H}; \Gamma; \Omega; P \vdash e : r_e\ \tau_e \dashv \mathcal{H}'; \Gamma'; \Omega'$. *Then* $dom(\Gamma) \supseteq dom(\Gamma')$.

Proof. Clear from inspection of the conclusions of all typing rules. ◻

2.8.5 *Proof of Progress.*

Theorem 2.2 (Progress (Restated)). *Let* $e \notin \textit{LocationNames}$ *be a well-typed non-blocking expression with a sound dynamic configuration, i.e.* $\mathcal{H}; \Gamma; \Omega; P \vdash e : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega'$ *and* $(\mathcal{H}, \Gamma, \Omega, P, d, h, s)$ *satisfies* I0 - Sound-Configuration. *Then there exists an expression* e' *with dynamic configuration* (d', h', s') *such that* $(d, h, s, e) \xrightarrow{\text{EVAL}} (d', h', s', e')$.

We prove theorem 2.2, Progress, inductively over the derivation of well-typedness of our expression. We can proceed by casework on the typing rule that derived the well-typedness judgment $\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega'$, noting we are also given I0$(\mathcal{H}, \Gamma, \Omega, \mathrm{P}, d, h, s)$ as a hypothesis. Without loss of generality, by lemma 2.5, we assume that the typing derivation is in *normal form*.

- T1 - Location-Ref: Not possible - we assumed in the hypothesis that $e \notin \textit{LocationNames}$
- T2 - Variable-Ref: Invariant I4 - Location-Type-Consistency tells us that $s(x) = l$ for some l, which will be in the *live-roots*, and thus contained in the *live-set* which, by I0, is contained in d. This is sufficient to derive a step by E2 - Variable-Ref-Step.
- T3 - Sequence: If $e = l; e'$, then we can directly apply E3 - Sequence-Step. Otherwise $e = e_l; e_r$ for $e_l \notin \textit{LocationNames}$. In this case by T3, e_l typechecks with the same input contexts as e, and so we can resort to induction to obtain a step $(d, h, s, e') \xrightarrow{\text{EVAL}} (d', h', s', e''')$, to which we can then apply E1 - Evaluation-Context-Step and conclude that e itself steps.
- T4 - Non-Isolated-Field-Reference: If $e = l.f$, then we note from inversion of T1 that $l \in \textit{live-roots} \subseteq \textit{live-set} \subseteq d$, and letting $l_f = h_v(l)[f]$ we note that $(r \overset{\cdot}{\dashrightarrow} l_f) \in \textit{outward-paths}$, so $l_f \in \textit{live-set} \subseteq d$, and we have sufficient information to apply E5b - Final-Reference-Step–Location and step. If $e = x.f$, then I4 and I1 - Reservation-Sufficiency give us $s(x) = l \in \textit{live-roots} \subseteq d$, and for $l_f = h_v(l)[f]$, the *outward-paths* reasoning above tells us

$l_f \in d$ as well, so we can apply E5A - Final-Reference-Step–Variable and step in this case as well. If $e = e'.f$ for $e' \notin LocationNames \cup VariableNames$, then we note that T4 gives us a typing for e' with the same static input contexts as e, so we can resort to induction to obtain a step $(d, h, s, e') \xrightarrow{\text{EVAL}} (d', h', s', e'')$ followed by an application of E4 - Contextual-Reference-Step to conclude $(d, h, s, e'.f) \xrightarrow{\text{EVAL}} (d', h', s', e''.f)$.

T5 - Isolated-Field-Reference: Invariant I4 - Location-Type-Consistency tells us $s(x) = l \in$ *live-roots*. Since $x.f$ is tracked, $l_f = h_v(s(x))[f] \in$ *live-roots* as well, so I1 - Reservation-Sufficiency tells us that $l, l_f \in d$ and we can apply E5A - Final-Reference-Step–Variable to derive a step.

T6 - Non-Isolated-Field-Assignment: If e is of the form $l.f = l_f$, then we make the same observations as in the case T4 - Non-Isolated-Field-Reference through inversion of T1 - Location-Ref to determine that $l, l_f \in d$ and apply E7B - Final-Assignment-Step–Location to step. Also similarly to T4, we can step if e is of the form $x.f = l_f$ via E7A - Final-Assignment-Step–Variable. If e is of the form $e'.f = l_f$ for $e' \notin LocationNames \cup VariableNames$, then inversion of T6 tells us that some static input contexts suffice to type-check e'. Further, these contexts can be chosen to differ from those used to type-check e only by the application of a $\overset{\text{VIR}}{\rightsquigarrow}$ transformation, as the typing derivations for l (location) typings can consists only of applications of T1 and TS1 - Virtual-Transformation-Structural. Note that we excluded the possibility of TS2 - Framing-Structural occurring in our lemma 2.5 choice of typing derivation above. Lemma 2.6 now tells us that any dynamic configuration (d, h, s) that was sound with respect to e's input contexts is also sound with respect to e''s, which suffices to inductively derive a step $(d, h, s, e') \xrightarrow{\text{EVAL}} (d', h', s', e'')$ that lifts by application of E6 - Contextual-Assignment-Step to a step $(d, h, s, e'.f = l_f) \xrightarrow{\text{EVAL}} (d', h', s', e''.f = l_f)$. The final case we must consider is when e takes the form $e'.f = e_f$. We can conclude by inversion of T6 that e_f typechecks with the same static input contexts as e, and inductively derive a step for e_f which then lifts by E1 - Evaluation-Context-Step to a step for $e'.f = e_f$.

T7 - Isolated-Field-Assignment: If e takes the form $x.f = l$, then observations similar to those above in T5 allow us to conclude that $s(x), l \in d$ and then apply E7A to obtain a step. Otherwise, if e takes the form $x.f = e_f$ for $e_f \notin LocationNames$, then we resort to induction exactly as in the last case of T6.

T8 - Assign-Var: If e takes the form $x = l$, then inversion of T1 gives us $l \in d$, which suffices to apply E8 - Assign-Var-Step and obtain a step. If e takes the form $x = e_x$ for $e_x \notin LocationNames$, then we note e_x's typing with the same static input contexts as e, and apply induction followed by E1 to obtain a step for e.

T9 - Function-Application: Program well-typedness tells us that *fn* has a lookup in Λ, and inversion of T9 tells us that the needed bijection Φ_x exists. All that remains is choice of fresh variable names to guarantee a safe substitution. Thus we can apply E9 - Function-Application-Step.

T10 - New-Loc: Application of E10 - New-Loc-Step actually requires no inversion of T10, the premises require only that we come up with a fresh heap h_{new} of type τ, and then append it to our existing dynamic configuration, which can always be done through recursive instantiation, generating a valid (though possibility infinite) fresh heap.

T11 - Declare-Var: Application of E11 - Declare-Var-Step is trivial.

T12 - OPlus: Subsumed by the T6 case, which has a strictly larger set of premises and analogous cases for stepping.

T13 - If-Statement: If e_b in our `if` statement has not been reduced to a location, we apply E1 as before. If $e_b = l_b$ for some l_b, then either $h_v(l) =$ `true` or $h_v(l) =$ `false`, and we can step with either E13A - If-True-Step or E13B - If-False-Step respectively.

T14 - While-Loop: Application of E14 - While-Loop-Step is trivial.

- T15 - If-Disconnected: The premise of E15a - If-Disconnected-Success-Step is the negation of the premise of E15b - If-Disconnected-Failure-Step, so by excluding the middle we conclude that one rule will always be applicable.
- T16 - Send and T17 - Receive: Neither of these could have derived e, as then it would not be *non-blocking*.
- T18 - None: Application of E18 - None-Step is trivial.
- T19 - Some: Logic from prior cases allows us to conclude that $l \in d$ and $h(l) = (\tau, v)$, yielding application of E19 - Some-Step.
- T20 - Destruct-Option: We know that $h(l) = (\tau, v)$ for some $v = \cdot$ or $v \neq \cdot$. In the former case we step with E20a - Let-Some-Step, and in the latter with E20b - Let-None-Step.
- TS1 - Virtual-Transformation-Structural: Here we resort to simple induction obtaining a step for e from its well-typedness as expressed in the premise of TS1, which satisfies all needed conditions for this case.
- TS2 - Framing-Structural: By our application of lemma 2.5, we have well-typedness for e derived by a sequence of TS2 instances ending in T9 - Function-Application. Given *normal form*, this is the only case in which a TS2 instance can occur as the root of the typing derivation. Thus our expression e is of the form $fn(x_1, \ldots, x_n)$, and, as seen in the case for T9 directly, we need no examination of the input contexts to conclude that an image of fn exists in Λ, that a bijection Φ_x exists on the x_i', and that a safe substitution exists for the remaining variables of the function body looked up in Λ. One can note that this case highlights the need for *normal form*; if an arbitrary rule could derive the premises of TS2 then we would need an analgous of lemma 2.6 for F rules, which does not hold.

This casework allows to conclude that whatever T rule derived well-typedness for e, we have sufficient information to derive a step for e with the provided dynamic contexts (d, h, s), concluding our proof of Progress in this system.

2.8.6 *Proof of Preservation.*

Theorem 2.3 (Preservation (Restated)). *Let e be a well-typed expression with a sound dynamic configuration and a valid step, i.e.* $\mathcal{H}; \Gamma; \Omega; P \vdash e : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega'$, $(\mathcal{H}, \Gamma, \Omega, P, d, h, s)$ *satisfies* I0 - Sound-Configuration, *and* $(d, h, s, e) \xrightarrow{\text{EVAL}} (d', h', s', e')$. *Then there exist contexts* $(\bar{\mathcal{H}}, \bar{\Gamma}, \bar{\Omega}, \bar{P}, \bar{\mathcal{H}}', \bar{\Gamma}', \Omega_{new})$ *such that* $\bar{\mathcal{H}}; \bar{\Gamma}; \bar{\Omega}; \bar{P} \vdash e' : r\ \tau \dashv \bar{\mathcal{H}}'; \bar{\Gamma}'; \Omega' \uplus \Omega_{new}$ *and* $(\bar{\mathcal{H}}, \bar{\Gamma}, \bar{\Omega}, \bar{P}, d', h', s')$ *satisfies* I0. *Further,* $\bar{P}$ *agrees with* P *on all output-tracked regions, i.e.* $P \restriction_{P_r^{-1}(regs(\mathcal{H}'))} \sqsubseteq \bar{P}$, *and* $\bar{\mathcal{H}}', \bar{\Gamma}'$ *differ from* $\mathcal{H}', \Gamma'$, *only by a region renaming that preserves all names tracked in* $\mathcal{H}'$, *i.e.* $\bar{\mathcal{H}}', \bar{\Gamma}' = \Phi_r(\mathcal{H}'), \Phi_r(\Gamma')$, *where* $\Phi_r \restriction_{regs(\mathcal{H}')} = Id$

We prove theorem 2.3, Preservation, inductively over the joint structures of the typing derivation for $\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega'$, and the step $(d, h, s, e) \xrightarrow{\text{EVAL}} (d', h', s', e')$. As for progress, we assume, WLOG by lemma 2.5, that the typing derivation applies TS2 only to instances of itself (TS2, and to T9. This may cause our output contexts to differ by a renaming of untracked regions, but this is allowed for in the phrasing of theorem 2.3.

First, we consider the case in which the root rule deriving well-typedness is TS1. This case implies well-typedness, dynamic soundness, and a step of the same expression but with respect to output contexts $\mathcal{H}_v, \Gamma_v, \Omega_v$ that could differ from our given contexts $\mathcal{H}', \Gamma', \Omega'$ by a V transformation $(\mathcal{H}_v, \Gamma_v, \Omega_v) \overset{\text{VIR}}{\rightsquigarrow} (\mathcal{H}', \Gamma', \Omega')$. Lemma 2.6 is sufficient to conclude that after the step, we can re-apply the same $\overset{\text{VIR}}{\rightsquigarrow}$ transformation and obtain a well-typedness and sound dynamic configuration that satisfy the conditions of 2.3.

We now proceed with casework on the E rule that derived our given step. Recall that in each case, the rule deriving our well-typedness is not TS1, and is not TS2 unless our expression is of the form $fn(x_1, \ldots, x_n)$.

E1 - Evaluation-Context-Step: We are given $\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash E[e] : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega'$, I0$(\mathcal{H}, \Gamma, \Omega, \mathrm{P}, d, h, s)$, and $(d, h, s, E[e]) \xrightarrow{\text{EVAL}} (d', h', s', E[e'])$ as premises in this case, and wish to conclude $\bar{\mathcal{H}}; \bar{\Gamma}; \bar{\Omega}; \bar{\mathrm{P}} \vdash E[e'] : r\ \tau \dashv \bar{\mathcal{H}}'; \bar{\Gamma}'; \bar{\Omega}'$ and I0$(\bar{\mathcal{H}}, \bar{\Gamma}, \bar{\Omega}, \bar{\mathrm{P}}, d', h', s')$. We can invert E1 to obtain the step $(d, h, s, e) \xrightarrow{\text{EVAL}} (d', h', s', e')$. We proceed b casework on the evaluation context $E[]$.

- $\boxed{\bar{e}.f = []}$: Here either T6 - Non-Isolated-Field-Assignment or T7 - Isolated-Field-Assignment derived the well-typedness of $\bar{e}.f = e$. In either case we can conclude well-typedness of e with the same input contexts as $E[e]$, so we can apply Preservation inductively and to obtain well typedness of e' with new static input contexts $\bar{\mathcal{H}}, \bar{\Gamma}, \bar{\Omega}, \bar{\mathrm{P}}$ that, together with d', h', s', satisfy I0 - Sound-Configuration. All that remains to be seen is that the new static output contexts checking e' satisfy the rest of the conditions of our T rule so that it can be re-applied to obtain well-typedness for $E[e']$. For T6 this reduces to lemma 2.8, and for T7, it is easy to see that the matching on the output is not invalidated by renaming of untracked regions, and in either case we are done.
- $\boxed{x = []}$: Here T8 - Assign-Var derived the well-typedness of $x = e$. Similarly to the previous case, we obtain well-typedness for the subexpression e with the same static input contexts, and then apply inductive Preservation to conclude well-typedness of a new configuration for e', to which T8 can then be applied to obtain well-typedness of $x = e'$ with static input contexts that, together with d', h', s', satisfy I0.
- $\boxed{[] \oplus \bar{e}}$: Here T12 - OPlus derived the well-typedness of $e \oplus \bar{e}$. There is no additional logic needed to justify preservation beyond that present in the case for $\bar{e}.f = []$, as the relationship of the typing of the e to the typing of $E[e]$ is the same in both cases.
- $\boxed{l \oplus []}$: Here too, T12 derived the well-typedness. This is similar to the above cases, except that we must argue that the $\bar{\mathrm{P}}$ obtained by inductive preservation to type-check e' preserves the mapping of l by the original P, which is ensured by the inductive hypothesis.
- $\boxed{\texttt{if}([])\{e_t\}\ \texttt{else}\ \{e_f\}}$: Here T13 - If-Statement derived our well-typedness, and the inductive application of Preservation does not differ from the above cases, for example for $[] \oplus \bar{e}$.
- $\boxed{\texttt{send-}\tau([])}$: Here T16 - Send derived our well-typedness, and induction is clearly sufficient to conclude Preservation.

We can conclude that whatever form $E[]$ takes, the induction is sound, and preservation for stepping e implies preservation for stepping $E[e]$.

E2 - Variable-Ref-Step: We are given $\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash x : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega'$, I0$(\mathcal{H}, \Gamma, \Omega, \mathrm{P}, d, h, s)$, and $(d, h, s, x) \xrightarrow{\text{EVAL}} (d', h', s', l)$ as premises in this case, and wish to conclude $\mathcal{H}; \Gamma; \Omega; \mathrm{P}' \vdash l : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega'$ and I0$(\mathcal{H}, \Gamma, \Omega, \mathrm{P}', d, h, s)$, where $\mathrm{P}' = l : r\ \tau, \mathrm{P}$. The well-typedness is easily derivable by T1 - Location-Ref, and as the mapping $l : r\ \tau$ in P' agrees with the mapping $x : r\ \tau$ in Γ, we can conclude I0 is preserved as well.

E3 - Sequence-Step: We are given $\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash l; e : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega'$, I0$(\mathcal{H}, \Gamma, \Omega, \mathrm{P}, d, h, s)$, and $(d, h, s, x) \xrightarrow{\text{EVAL}} (d', h', s', l)$ as premises in this case, and wish to conclude $\bar{\mathcal{H}}; \bar{\Gamma}; \bar{\Omega}; \cdot \vdash e : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega'$ and I0$(\bar{\mathcal{H}}, \bar{\Gamma}, \bar{\Omega}, \cdot, d, h, s)$ for some $\bar{\mathcal{H}}, \bar{\Gamma}, \bar{\Omega}$. In particular, we note from inversion of the T3 - Sequence rule that must have typechecked $l; e$ that exactly such $\bar{\mathcal{H}}, \bar{\Gamma}, \bar{\Omega}$ exist, and can differ from $\mathcal{H}, \Gamma, \Omega$ only in the case that TS1 - Virtual-Transformation-Structural derived the typing for l, in which they differ exactly by some virtual transformation $\overset{\text{VIR}}{\rightsquigarrow}$. We can then apply lemma 2.6 to conclude that I0 holds as desired. We must also weaken P down to $\cdot$ in the configuration, which only has the effect of weakening I0. This concludes preservation in this case.

E4 - Contextual-Reference-Step and E6 - Contextual-Assignment-Step: These cases utilize the inductive hypothesis just as the case for E1 - Evaluation-Context-Step did. The rules T4 - Non-Isolated-Field-Reference and T6 rely on their subexpressions only as in the $l \oplus []$ subcase above.

E5a - Final-Reference-Step–Variable: We are given $\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash x.f : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega'$, I0$(\mathcal{H}, \Gamma, \Omega, \mathrm{P}, d, h, s)$, and $(d, h, s, x.f) \xrightarrow{\text{EVAL}} (d, h, s, l_f)$ as premises in this case, and wish to conclude $\mathcal{H}; \Gamma; \Omega; \bar{\mathrm{P}} \vdash l_f : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega'$ and I0$(\mathcal{H}, \Gamma, \Omega, \bar{\mathrm{P}}, d, h, s)$ for $\bar{\mathrm{P}} = l_f : r\ \tau, \mathrm{P}$. Our desired well-typedness follows trivially from choice of $\bar{\mathrm{P}}$, so it suffices to show that I0 is preserved under the possible expansion to P. Inversion of E5a, and I3 - Heap-Closure are sufficient to tell us that $h_\tau(l_f) = \tau$, so by lemma 2.10 it suffices to show that $(r \overset{\cdot}{\dashrightarrow} l_f) \in$ *outward-paths*$(\mathcal{H}, \Gamma, \mathrm{P}, h, s)$. Inversion of T5 - Isolated-Field-Reference tells us that the reference $x.f$ is fully tracked, and we know from E5a that $s(x) = l_f$, so it is clear that $(l_f, r) \in$ *live-roots*$(\mathcal{H}, \Gamma, \mathrm{P}, h, s)$, and we are done.

E5b - Final-Reference-Step–Location: We are given $\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash l.f : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega'$, I0$(\mathcal{H}, \Gamma, \Omega, \mathrm{P}, d, h, s)$, and $(d, h, s, l.f) \xrightarrow{\text{EVAL}} (d, h, s, l_f)$ as premises in this case, and wish to conclude $\bar{\mathcal{H}}; \bar{\Gamma}; \bar{\Omega}; \bar{\mathrm{P}} \vdash l_f : r\ \tau \dashv \mathcal{H}'; \Gamma'; \Omega'$ and I0$(\bar{\mathcal{H}}, \bar{\Gamma}, \bar{\Omega}, \bar{\mathrm{P}}, d, h, s)$ for some $\bar{\mathcal{H}}, \bar{\Gamma}, \bar{\Omega}, \bar{\mathrm{P}}$. As in the case for E3 - Sequence-Step we reason through inversion of T1 - Location-Ref that we can choose the new contexts such that $(\mathcal{H}; \Gamma; \Omega) \overset{\text{VIR}}{\rightsquigarrow} (\bar{\mathcal{H}}; \bar{\Gamma}; \bar{\Omega})$, and so I0 on the new configuration is given by lemma 2.6. We additionally choose $\bar{\mathrm{P}} = l_f : r\ \tau, \mathrm{P}$, and now must show that I0$(\bar{\mathcal{H}}, \bar{\Gamma}, \bar{\Omega}, \mathrm{P}, d, h, s) \Rightarrow$ I0$(\bar{\mathcal{H}}, \bar{\Gamma}, \bar{\Omega}, \bar{\mathrm{P}}, d, h, s)$. We note that $(r \overset{\cdot}{\dashrightarrow} l_f) \in$ *outward-paths*$(\bar{\mathcal{H}}, \bar{\Gamma}, \mathrm{P}, h, s)$, and by I3$(h)$, $h_\tau(l_f) = \tau$, so lemma 2.10 allows us to conclude exactly this. We finally note that our choice of $\bar{\mathrm{P}}$ makes the desired well-typedness easy to conclude through T1, and we conclude this case.

E7a - Final-Assignment-Step–Variable: We are given $\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash x.f = l_f : r_f\ \tau_f \dashv \mathcal{H}'; \Gamma'; \Omega'$, I0$(\mathcal{H}, \Gamma, \Omega, \mathrm{P}, d, h, s)$ and $(d, h \uplus (l \mapsto (\tau, v)), s, l.f = l_f) \xrightarrow{\text{EVAL}} (d, h \uplus (l \mapsto (\tau, v[f \mapsto l_f])), s, l_f)$ as premises in this case, where $\mathcal{H}' = \mathcal{H}'', r^\circ\langle x^{\circ'}[f \rightarrowtail r_f, F], X\rangle, r_f^{\circ_f}\langle X_f\rangle$ and we wish to conclude that $\mathcal{H}'; \Gamma'; \Omega'; \mathrm{P} \vdash l_f : r_f\ \tau_f \dashv \mathcal{H}'; \Gamma'; \Omega'$ and I0$(\mathcal{H}', \Gamma', \Omega', \mathrm{P}, d, h \uplus (l \mapsto (\tau, v[f \mapsto l_f])), s)$. As in the T6 - Non-Isolated-Field-Assignment case, the well-typedness proof goal is easily obtained from inversion of the original, yielding $(l_f : r_f\ \tau_f) \in \mathrm{P}$, and combining with our definitional knowledge that $r_f \in$ *regs*$(\mathcal{H}')$ to directly apply T1. Continuing to examine the results of inverting T7 - Isolated-Field-Assignment, we note that I0$(\mathcal{H}'', r^\circ\langle x^{\circ'}[f \rightarrowtail r_{old}, F], X\rangle, r_f^{\circ_f}\langle X_f\rangle; \Gamma'; \Omega'; \mathrm{P}; d; h \uplus (l \mapsto (\tau, v)); s)$ follows from lemma 2.6, so we must only reason about preservation of I0 under the assignment updates to $\mathcal{H}$ and h. Verifying I3-I7 - Region-Names-Bounding is routine, following from simple observations such as $r_f \in \Omega$ and $h_\tau(l_f) = \tau_f$ that are directly evident from inversion of rules on the hypotheses of this case. Verifying I1 - Reservation-Sufficiency and I2 - Tree-Of-Untracked-Regions requires us to reason that the set of *outward-paths* after the step is contained within the set from before the update. To illustrate this embedding, let $(r_o \overset{L_o}{\dashrightarrow} l_o) \in$ *outward-paths*$(\mathcal{H}', \Gamma', \mathrm{P}, d, h \uplus (l \mapsto (\tau, v[f \mapsto l_f])), s)$. We wish to show that $(r_o \overset{L_o}{\dashrightarrow} l_o) \in$ *outward-paths*$(\mathcal{H}'', r^\circ\langle x^{\circ'}[f \rightarrowtail r_{old}, F], X\rangle, r_f^{\circ_f}\langle X_f\rangle; \Gamma'; \mathrm{P}; d; h \uplus (l \mapsto (\tau, v)); s)$. Since $l.f$ is tracked in $\mathcal{H}'$, clearly $l.f \notin L_o$. Since there are no other fields that differ between the pre and post-step h, we can see that L_o is the isolated subsequence of a valid sequence of fields and locations in the pre-step configuration as well. We note that the only possible location that could be an element of the post-step *live-roots* but not the prestep is l_f itself, because now it is the target of a tracked field, but in fact it was already a live root because it necessarily appeared in P from inversion of T1 on the original well-typedness. Thus the live root origin of the sequence that generated L_o is a live root in the pre-step configuration. Finally, we note that all no fields besides $l.f$ are tracked in the pre-step but not post-step configurations, so it is impossible that some $l.f \in L_o$ is tracked pre-step. This covers all the conditions to establish that $(r_o \overset{L_o}{\dashrightarrow} l_o)$ is in the *outward-paths* of the pre-step configuration as well, so we have shown that the set can only shrink, which allows us to conclude preservation of I1 and I2, completing our argument in this case.

E7b - Final-Assignment-Step–Location: We are given $\mathcal{H};\Gamma;\Omega;\mathrm{P} \vdash l.f = l_f : r\ \tau_f \dashv \mathcal{H}';\Gamma';\Omega'$, I0$(\mathcal{H},\Gamma,\Omega,\mathrm{P},d,h,s)$ and $(d, h \uplus (l \mapsto (\tau,v)), s, l.f = l_f) \xrightarrow{\text{EVAL}} (d, h \uplus (l \mapsto (\tau, v[f \mapsto l_f])), s, l_f)$ as premises in this case, and wish to conclude $\mathcal{H};\Gamma;\Omega;\mathrm{P} \vdash l_f : r\ \tau_f \dashv \mathcal{H}';\Gamma';\Omega'$ and I0$(\mathcal{H},\Gamma,\Omega,\mathrm{P},d,h \uplus (l \mapsto (\tau, v[f \mapsto l_f])), s)$. This is already nearly exactly a premise yielded from inversion of T6 - Non-Isolated-Field-Assignment on the given well-typedness, but we may need to add one layer of TS1 - Virtual-Transformation-Structural application that was previously found on the typing for l. All that remains is to show I0 holds after updating h as specified by the step. I4 - Location-Type-Consistency is a function of h, but only of h_τ, which is not affected by this step and thus the invariant is preserved. To show preservation of I3 - Heap-Closure, we only need to check that $h_\tau(l_f) = \tau_f$, which follows from the necessity of $\mathrm{P}_\tau(l_f) = \tau_f$ to derive well-typedness, and I4 on the old configuration. To show that I1 - Reservation-Sufficiency and I2 - Tree-Of-Untracked-Regions are preserved, we argue that *outward-paths*$(\mathcal{H},\Gamma,\mathrm{P},h \uplus (l \mapsto (\tau, v[f \mapsto l_f])), s) \subseteq$ *outward-paths*$(\mathcal{H},\Gamma,\mathrm{P},h \uplus (l \mapsto (\tau,v)), s)$. The key observation is that $(l_f, r) \in$ *live-roots*$(\mathcal{H},\Gamma,\mathrm{P},h \uplus (l \mapsto (\tau,v)), s)$, so any outward path in *outward-paths*$(\mathcal{H},\Gamma,\mathrm{P},h \uplus (l \mapsto (\tau, v[f \mapsto l_f])), s)$ which was generated including the fact that $l.f = l_f$ is identically present in *outward-paths*$(\mathcal{H},\Gamma,\mathrm{P},h \uplus (l \mapsto (\tau,v)), s)$ with l_f as its root, and any sequence generated otherwise is present for the exact same reason. From this containment, preservation of I1 and I2 easily follow, concluding this case.

E8 - Assign-Var-Step: We are given $\mathcal{H};\Gamma;\Omega;\mathrm{P} \vdash x = l : r\ \tau \dashv \mathcal{H}';x : r\ \tau,\Gamma';\Omega'$, I0$(\mathcal{H},\Gamma,\Omega,\mathrm{P},d,h,s \uplus (x \mapsto l_{old}))$, and $(d,h,s \uplus (x \mapsto l_{old}), x = l) \xrightarrow{\text{EVAL}} (d,h,s \uplus (x \mapsto l), l)$ as premises in this case, and wish to conclude $\mathcal{H}';x : r\ \tau,\Gamma';\Omega';\mathrm{P} \vdash l : r\ \tau \dashv \mathcal{H}';x : r\ \tau,\Gamma';\Omega'$ and I0$(\mathcal{H}';x : r\ \tau,\Gamma',\Omega',\mathrm{P},d,h,s \uplus (x \mapsto l))$. As in prior cases, the well-typedness is easy, as it only requires inverting T8 - Assign-Var on our hypothesis to obtain a typing for l, which inverts under T1 - Location-Ref to yield $(l : r\ \tau) \in \mathrm{P}$, and tracking of r in the static output contexts. There could still be an application of TS1 layered on top of that T1 to obtain the typing for l, but that is guaranteed not to drop r from $\mathcal{H}'$ so the judgments we care about remain intact, and we can conclude that l types under P and $\mathcal{H}'$, concluding the well-typedness portion of our argument in this case. We can reason via lemma 2.6 that I0$(\mathcal{H}';x : r_{old}\ \tau,\Gamma';\Omega';\mathrm{P};d;h;s \uplus (x \mapsto l_{old}))$, but must resort to further logic in order to obtain our final dynamic soundness result on the updated Γ and s. I1 and I2 depend on Γ and s only to determine the *live-roots* and the images of tracked variables, and since (l,r) was already a live root pre-step, the update to x can only decrease the set of live roots and thus outward path, weakening I1 and I2 as seen several times above, and since x is untracked the updates influence these invariants no further. Thus the first two are preserved under the update. I3 only concerns h, I4 checks for agreement between Γ and s, which is guaranteed by their mutual update, I5 - Variable-Region-Consistency and I6 - Focus-Non-Aliasing again only concern tracked variables, and I7 - Region-Names-Bounding is preserved trivially because no new region names are introduced. Thus we can conclude that I0 holds on the post-step configuration, concluding our argument in this case.

E9 - Function-Application-Step: We are given $\mathcal{H};\Gamma;\Omega;\mathrm{P} \vdash fn(x_1,\ldots,x_n) : r_0\ \tau_0 \dashv \mathcal{H}';\Gamma';\Omega'$, I0$(\mathcal{H},\Gamma,\Omega,\mathrm{P},d,h,s)$, and $(d,h,s,fn(x_1,\ldots,x_n)) \xrightarrow{\text{EVAL}} (d,h,s,e)$ as premises in this case, and which to conclude $\mathcal{H};\Gamma;\Omega;\mathrm{P} \vdash e : r_0\ \tau_0 \dashv \mathcal{H}';\Gamma';\Omega''$. We do not have to derive a new configuration soundness result, because all relevant contexts (i.e. $\mathcal{H},\Gamma,\Omega,\mathrm{P},d,h,s$) take the same form pre and post-step. Unlike all of the other cases here, in which we are able to state with certainty which T rule derived our well-typedness, in this case it is possible that either a

TS2 - Framing-Structural or T9 - Function-Application instance derived our well-typedness, and in the former case it as actually possible that several TS2 instances are nested before reaching a T9 instance. To derive well-typedness for the stepped expression e, we will use the exact same tower of TS2 instances. We know that $\Lambda(fn) = (\lambda x'_1, \ldots, x'_n : e')$, where $e = \Phi_x(e')$ and Φ_x is a bijection on region names that chooses names that will not occur anywhere else in the program (we omit formalization of this "safe substitution logic" as it is heavyweight but unenlightening) for variables declared within e, and chooses the appropriate, syntactically supplied variable names for $x'_1, \ldots, x'_n$. Where $\bar{\mathcal{H}}; \bar{\Gamma}; \bar{\Omega}; \bar{\mathrm{P}} \vdash fn(x_1, \ldots, x_n) : r_0\ \tau_0 \dashv \bar{\mathcal{H}}'; \bar{\Gamma}'; \bar{\Omega}'$ is the judgment concluded by the relevant instance of T9 in our typing derivation, we know that $e = \Phi_x(e')$ also admits derivation of the typing judgment $\bar{\mathcal{H}}; \bar{\Gamma}; \bar{\Omega}; \bar{\mathrm{P}} \vdash e : r_0\ \tau_0 \dashv \bar{\mathcal{H}}'; \bar{\Gamma}'; \bar{\Omega}' \uplus \Omega_{new}$. This involved possibly choosing a bijective region renaming Φ_r, as specified in the typing rule T9, but choice of region names is arbitrary and so this is easily derivable as an alternative to contexts with the original region names from the function type. We also note that we had to introduce the possibility of additional region names, Ω_{new}, being introduced, as the function type alone did not capture that possibility. To this typing judgment for e, we can then apply the stack of TS2 instances that were present in the original, noting that the expansion by unique names in Ω_{new} does not interfere with this, to conclude exactly that $\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash e : r_0\ \tau_0 \dashv \mathcal{H}'; \Gamma'; \Omega' \uplus \Omega_{new}$, which fits our desired conclusion and thus establishes preservation in this case.

E10 - New-Loc-Step and E12 - OPlus-Step The argument for preservation in these cases is easy because the existing contexts are not changed in any way. A totally new heap h_{new} is constructed and appended to the existing heap, all of its locations d_{new} are appended to the existing dynamic reservation, s is not updated, and $\mathcal{H}$ and Ω just have the fresh region of the root appended to them. The only invariant that it is not immediately obvious holds on the new configuration is I2 - Tree-Of-Untracked-Regions, and this may be seen to hold by examining the algorithm for constructing our new heap, and noticing that every location is uniquely defined by its path in isolated references from the root location l, which is exactly what we need to conclude I2 holds on that new heap. We can conclude that a P including the binding $l : r\ \tau$ is a sound update to our configuration, and that all the invariants are preserved by that and accompanying changes, so preservation holds here.

E11 - Declare-Var-Step: We are given $\mathcal{H}; \Gamma; \Omega; \mathrm{P} \vdash \texttt{declare}\ x : \tau\ \texttt{in}\ \{e\} : r_e\ \tau_e \dashv \mathcal{H}'; \Gamma'; \Omega'$, I0$(\mathcal{H}, \Gamma, \Omega, \mathrm{P}, d, h, s)$, and $(d, h, s, \texttt{declare}\ x : \tau\ \texttt{in}\ \{e\}) \xrightarrow{\text{EVAL}} (d, h, s[x \mapsto \bot], e)$ as premises in this case, and we wish to conclude $\mathcal{H}; \Gamma, x : \bot\ \tau; \Omega; \mathrm{P} \vdash e : r_e\ \tau_e \dashv \mathcal{H}'; \Gamma'; \Omega'$ and I0$(\mathcal{H}; \Gamma, x : \bot\ \tau; \Omega; \mathrm{P}; d; h; s[x \mapsto \bot])$. The goal well-typedness judgment is very similar to one obtained exactly through inversion of T11 - Declare-Var on the assumed well-typedness, but differs exactly in that the mapping of x in Γ is present in the judgment we have, but not the one we want. We note that this can be remedied exactly by an application of TS1 - Virtual-Transformation-Structural with V6 - Drop-Variable its $\overset{\text{VIR}}{\rightsquigarrow}$ transformation, and thus the desired typing judgment can be obtained. We must now argue that I0 is preserved under the step's updates to Γ and s. I1 - Reservation-Sufficiency-I6 - Focus-Non-Aliasing depend on Γ only on the portion of its domain mapped to tracked regions, and on s only through its image of the same domain of Γ. Since Γ's restriction to that domain is unaltered by the update $\Gamma \mapsto \Gamma, x : \bot\ \tau$, I1-I6 are easily seen to be preserved. We can also note that I7 - Region-Names-Bounding explicitly allows for the inclusion of the bottom region in Γ_r's range, making its preservation evident as well. This allows us to conclude I0 holds on the post-step configuration, concluding preservation in this case.

E13a - If-True-Step and E13b - If-False-Step: There are no interesting effects in these cases: well-typedness up to virtual transformations $\overset{\text{VIR}}{\rightsquigarrow}$ is guaranteed by inversion of T13 - If-Statement, and so lemma 2.6 is sufficient to conclude soundness of the new dynamic configuration, concluding our proof obligations in this case.

E14 - While-Loop-Step: We are given $\mathcal{H};\Gamma;\Omega;\mathrm{P} \vdash \texttt{while}\ (e_b)\ \{e\} : r_u\ \texttt{unit} \dashv \mathcal{H};\Gamma;\Omega'' \uplus \{r_u\}$, I0$(\mathcal{H},\Gamma,\Omega,\mathrm{P},d,h,s)$, and $(d,h,s,\texttt{while}\ (e_b)\ \{e\}) \xrightarrow{\text{EVAL}} (d,h,s,\texttt{if}\ (e_b)\ \{e;\texttt{while}\ (e_b)\ \{e\}\}\ \texttt{else}\ \{\texttt{new-unit()}\})$ as premises in this case, and we wish to conclude $\mathcal{H};\Gamma;\Omega;\mathrm{P} \vdash \texttt{if}\ (e_b)\ \{e;\texttt{while}\ (e_b)\ \{e\}\}\ \texttt{else}\ \{\texttt{new-unit()}\} : r_u\ \texttt{unit} \dashv \mathcal{H};\Gamma;\Omega'' \uplus \Omega_{new}$. We do not have to derive a new configuration soundness, because all relevant contexts take the same form pre and post-step. To obtain the needed well-typedness judgment, we observe that inversion of T9 - Function-Application gives us typings for both e_b and e that use the exact same $(\mathcal{H},\Gamma)$ pairs on the input and output, as well as empty P, and a possibly larger Ω on the output than input. Let n be difference in cardinality of output and input Ω in the typing for e, and m be that difference for e_b. Since all region names chosen by a typing derivation are arbitrary, for any disjoint Ω_l and Ω_r, where Ω_r has cardinality n (resp. m) we are free to obtain a typing for e (resp. e_b) of the form $\mathcal{H};\Gamma;\Omega \uplus \Omega_l;\cdot \vdash e : r\ \tau$ (resp. $e_b : r_b\ \tau_b$) $\dashv \mathcal{H};\Gamma;\Omega \uplus \Omega_l \uplus \Omega_r$. This family of typings for e and e_b thus obtained is sufficient to choose 4 pairwise disjoint sets $\Omega_1,\Omega_2,\Omega_3,\Omega_4$ of cardinalities n,m,n,m respectively, and type-check the entire expression $\texttt{if}\ (e_b)\ \{e;\texttt{while}\ (e_b)\ \{e\}\}\ \texttt{else}\ \{\texttt{new-unit()}\})$ with input Ω, and output $\Omega \uplus \Omega_1 \uplus \Omega_2 \uplus \Omega_3 \uplus \Omega_4 \uplus \{r_u\}$, which fits the desired form and thus concludes our proof in this case.

E15a - If-Disconnected-Success-Step: We are given $r^{\cdot}\langle\rangle,\mathcal{H};x : r\ \tau_x, y : r\ \tau_y,\Gamma;\Omega;\mathrm{P} \vdash \texttt{if disconnected}(x,y)\ \{e_{succ}\}\ \texttt{else}\ \{e_{fail}\} : r_{out}\ \tau_{out} \dashv \mathcal{H}';\Gamma';\Omega_{succ} \cup \Omega_{fail}$, I0$(r^{\cdot}\langle\rangle,\mathcal{H};x : r\ \tau_x, y : r\ \tau_y,\Gamma;\Omega;\mathrm{P};d;h;s)$, and $(d,h,s,\texttt{if disconnected}(x,y)\ \{e_{succ}\}\ \texttt{else}\ \{e_{fail}\}) \xrightarrow{\text{EVAL}} (d$ as premises in this case, and we wish to conclude the well-typedness $r_x^{\cdot}\langle\rangle, r_y^{\cdot}\langle\rangle,\mathcal{H};x : r_x\ \tau_x, y : r_y\ \tau_y,\Gamma;\Omega \uplus \{r_x,r_y\} \uplus (\Omega_{fail} - \Omega_{succ});\cdot \vdash e_{succ} : r_{out}\ \tau_{out} \dashv \mathcal{H}';\Gamma';\Omega'' \uplus (\Omega_{fail} - \Omega_{succ})$, and the soundness I0$(r_x^{\cdot}\langle\rangle, r_y^{\cdot}\langle\rangle,\mathcal{H};x : r_x\ \tau_x, y : r_y\ \tau_y,\Gamma;\Omega \uplus \{r_x,r_y\} \uplus (\Omega_{fail} - \Omega_{succ});d,h,s)$. The well-typedness is not difficult, as it is given (up to Ω expansion) exactly by inversion of T15 - If-Disconnected. We proceed to verify that each of the invariants I1 - Reservation-Sufficiency-I7 - Region-Names-Bounding is preserved under the step. To verify I1, we will show that the *live-set* can only decrease under this step. Let l_o be in the post-step *live-set*, and let $(r_o \overset{L_o}{\dashrightarrow} l_o)$ be the post-step outward path guaranteeing it. Let Φ map all regions except r_x, r_y to themselves, and map r_x and r_y to r. As no untracked fields in the post-step configuration are tracked in the pre-step configuration, and as any location in the *live-roots* with region r_o of the post-step configuration is in the *live-roots* with region $\Phi(r_o)$ in the pre-step configuration, we can conclude $(\Phi(r_o) \overset{L_o}{\dashrightarrow} l_o)$ is a pre-step outward path. This establishes that l_o is also in the pre-step *live-set*, so the *live-set* can only shrink, and I1 is preserved. To verify I2 - Tree-Of-Untracked-Regions, it suffices to show that if two *outward-paths* exist in the post-step configuration that share a target location but differ in (r,L) pairs, then I2 does not hold for the pre-step configuration. Let $(r_1 \overset{L_1}{\dashrightarrow} l)$, $(r_2 \overset{L_2}{\dashrightarrow} l)$ be two such *outward-paths*, with $(r_1,L_1) \neq (r_2,L_2)$. We know that $(\Phi(r_1) \overset{L_1}{\dashrightarrow} l)$ and $(\Phi(r_2) \overset{L_2}{\dashrightarrow} l)$ are pre-step outward paths. So if we can show that $(\Phi(r_1),L_1) \neq (\Phi(r_2),L_2)$ then we are done. If $L_1 \neq L_2$ then we are done, so assume $L_1 = L_2$. Now if $L_1 = \cdot$, then $l \in$ *tracked-set*$(r_1^{\cdot}\langle\rangle;x : r_1\ \tau;h;s) \cap$ *tracked-set*$(r_2^{\cdot}\langle\rangle;y : r_2\ \tau;h;s)$, which contradicts the premise we can derive from inversion of E15a on the given step. Noting this contradiction involved noting that region names are interchangeable, as they do not appear in h or s. Thus we can assume L_1 has some first element $l'.f$. But then l' is reachable via *outward-paths* $(r_1 \overset{\cdot}{\dashrightarrow} l')$ and $(r_2 \overset{\cdot}{\dashrightarrow} l')$, and we have again violated the premise of E15a. Thus there is no case in which assumption of a violation of I2 in the post-step configuration does not imply a violation of I1 in the pre-step configuration, so we can conclude that I2 is preserved by this step. I3 - Heap-Closure is trivially preserved as h does not differ pre and post-step. I4 - Location-Type-Consistency is strictly weakened, as the set of variables and locations that lie in a tracked region is decreased. I5 - Variable-Region-Consistency and I6 - Focus-Non-Aliasing only concern tracked variables, and we enforce in T15 that the only region affected by this step contains no tracked variables, so they are preserved. Finally, the new region names r_x and r_y are introduced to Ω, so I7 is preserved. We have shown that all invariants I1-I7 are preserved by this step, which concludes our argument for preservation in this case.

E15b - If-Disconnected-Failure-Step: This case is trivial - as `if disconnected` in the negative case is effectively a no-op, and inversion of T15 - If-Disconnected tells us that we are able to type the post-step expression e_{fail} with the exact same (up to Ω expansion) contexts as the entire pre-step `if disconnected` expression.

E18 - None-Step: Subsumed by argument from E10 - New-Loc-Step case.

E19 - Some-Step: Well-typedness of the location l' after the step is trivial, as we simply add the binding $l' : r\ \tau?$ to P, where $l : r\ \tau$ was bound pre-step. We do note that (l', r) is a new elements of the *live-roots*, but the set of paths from l' is exactly the set of paths from l with the initial location replaced in each. This means that the only addition to the *live-set* is l' itself, which we add to d explicitly in E19 to preserve I1 - Reservation-Sufficiency. I2 - Tree-Of-Untracked-Regions is seen to be preserved by noting that the alteration of a path by replacing the initial location with one in the same region does not changes the *outward-paths*. I3 - Heap-Closure and I4 - Location-Type-Consistency are trivially preserved, noting for the former that all targets of fields from l' were targets from l, and for the latter that we added exactly $l' : r\ \tau?$ to P. The remaining invariants I4-I7 - Region-Names-Bounding do not rely on h, so we can conclude that I0 - Sound-Configuration is preserved by this step.

E20a - Let-Some-Step: Subsumed by composition of arguments from E11 - Declare-Var-Step and E19 cases.

E20b - Let-None-Step: Subsumed by argument from E11 case.

This casework allows to conclude that whatever E rule derived our step, we have sufficient information to derive well-typedness for the post-step expression, and soundness for for the post-step configuration, concluding our proof of Preservation in this system.

2.9 Concurrency

2.9.1 Definitions. We first state the definitions that allow us to lift our configurations to a concurrent setting.

Definition 2.4 (Concurrent Configuration). We let a Concurrent Configuration be a tuple consisting of a heap h, and an arbitrarily sized vector of triples (d_i, s_i, e_i) representing the reservation, stack, and active expression of each thread.

Definition 2.5 (Sound Concurrent Configuration). We call a concurrent configuration $(h, \overline{\langle d_n, s_n, e_n \rangle})$ *sound* if the following two conditions hold:

Well-Typedness: Each thread is individually well-typed with the common heap h: for each $i \leq n$, there exist $\mathcal{H}_i, \Gamma_i, \Omega_i, \mathrm{P}_i, r_i, \tau_i, \mathcal{H}'_i, \Gamma'_i, \Omega'_i$ such that $\mathcal{H}_i; \Gamma_i; \Omega_i; \mathrm{P}_i \vdash e_i : r_i\ \tau_i \dashv \mathcal{H}'_i; \Gamma'_i; \Omega'_i$ and I0 holds for the configuration $(\mathcal{H}_i, \Gamma_i, \Omega_i, \mathrm{P}_i, d_i, h, s_i)$.

Separation: The threads' reservations are pairwise disjoint: for any $i, j \leq n$, $i \neq j \Rightarrow d_i \cap d_j = \emptyset$.

Definition 2.6 (Non-blocked Concurrent Configuration). We call a concurrent configuration $(h, \overline{\langle d_n, s_n, e_n \rangle})$ *non-blocked* if at least one of the following two conditions holds:

Single-Steppable: For some $i \leq n$, the expression e_i is *non-blocking*.

Pair-Steppable: For some type τ, and for some $i, j \leq n$, i's redex is of the form `send`-$\tau(l)$ and j's redex is of the form `recv`-$\tau()$.

2.9.2 Small Step Relation. Now we give the relation for stepping concurrency configurations, which takes the form of two rules, one of which calls out to a subordinate judgement:

EC1 - Concurrent-Single-Step

$$\frac{i \in \{1..n\} \qquad (d_i, h, s_i, e_i) \xrightarrow{\text{EVAL}} (d_i', h', s_i', e_i') \qquad \forall j \in \{1..n\} - \{i\} : (d_j', s_j', e_j') = (d_j, s_j, e_j)}{(h, \overline{\langle d_n, s_n, e_n \rangle}) \xrightarrow{\text{concur-eval}} (h', \overline{\langle d_n', s_n', e_n' \rangle})}$$

EC2 - Concurrent-Paired-Step

$$\frac{\begin{array}{c} a, b \in \{1..n\} \\ s_a', s_b' = s_a, s_b \qquad h \vdash (d_a, e_a; d_b, e_b) \xrightarrow{\text{comm-eval}} (d_a', e_a'; d_b', e_b') \qquad \forall i \in \{1..n\} - \{a, b\} : (d_i', s_i', e_i') = (d_i, s_i, e_i) \end{array}}{(h, \overline{\langle d_n, s_n, e_n \rangle}) \xrightarrow{\text{concur-eval}} (h, \overline{\langle d_n', s_n', e_n' \rangle})}$$

EC3 - Communication-Paired-Step

$$\frac{d_{sep} = \textit{live-set}(r^{\cdot}\langle\rangle; \cdot; l : r\ \tau; h; \cdot)}{h \vdash (d_a \uplus d_{sep}, E_a^*[\texttt{send-}\tau(l_{root})]; d_b, E_b^*[\texttt{recv-}\tau()]) \xrightarrow{\text{comm-eval}} (d_a, E_a^*[\texttt{new-unit}]; d_b \uplus d_{sep}, E_b^*[l_{root}])}$$

2.9.3 Statements of Progress and Preservation.

Theorem 2.7 (Concurrent Progress). *Given a sound, non-blocked concurrent configuration* $(h, \overline{\langle d_n, s_n, e_n \rangle})$*, there exists a step* $(h, \overline{\langle d_n, s_n, e_n \rangle}) \xrightarrow{\textit{concur-eval}} (h', \overline{\langle d_n', s_n', e_n' \rangle})$

This theorem will be proven in subsection 2.9.4.

Theorem 2.8 (Concurrent Preservation). *Given a sound concurrent configuration* $(h, \overline{\langle d_n, s_n, e_n \rangle})$ *and a step* $(h, \overline{\langle d_n, s_n, e_n \rangle}) \xrightarrow{\textit{concur-eval}} (h', \overline{\langle d_n', s_n', e_n' \rangle})$*, the concurrent configuration* $(h', \overline{\langle d_n', s_n', e_n' \rangle})$ *is sound.*

This theorem will be proven in subsection 2.9.5.

2.9.4 Proof of Concurrent Progress. Proving theorem 2.7 is not hard, as the conditions in the definition of *non-blocked* are parallel to the possible rules EC1 - Concurrent-Single-Step and EC2 - Concurrent-Paired-Step for deriving a $\xrightarrow{\text{concur-eval}}$ step. Let $(h, \overline{\langle d_n, s_n, e_n \rangle})$ be a *sound*, *non-blocked* concurrent configuration. We show that whether the *single-steppable* or the *pair-steppable* condition was used to derive the *non-blocked* state of the configuration, there exists a step $(h, \overline{\langle d_n, s_n, e_n \rangle}) \xrightarrow{\text{concur-eval}} (h', \overline{\langle d_n', s_n', e_n' \rangle})$.

Single-Steppable: In this case, we are given some i such that e_i is a *non-blocking* expression. We note that the definition of a *sound* concurrent configuration gives us $\mathcal{H}_i; \Gamma_i; \Omega_i; \mathrm{P}_i \vdash e_i : r_i\ \tau_i \dashv \mathcal{H}_i'; \Gamma_i'; \Omega_i'$ and I0$(\mathcal{H}_i, \Gamma_i, \Omega_i, \mathrm{P}_i, d_i, h, s_i)$. Theorem 2.2, single-threaded Progress, then gives us a step $(d_i, h, s_i, e_i) \xrightarrow{\text{EVAL}} (d_i', h', s_i', e_i')$. For all $j \neq i$, we let $(d_j', s_j', e_j') = (d_j, s_j, e_j)$. and we can apply EC1 to obtain a step $(h, \overline{\langle d_n, s_n, e_n \rangle}) \xrightarrow{\text{concur-eval}} (h', \overline{\langle d_n', s_n', e_n' \rangle})$, concluding our argument in this case.

Pair-Steppable: In this case, we are given some type τ, and some a, b, where e_a's redex is of the form $\texttt{send-}\tau(l)$ and e_b's redex is of the form $\texttt{recv-}\tau()$. Unfolding the definition of a *sound* concurrent configuration gives us static contexts for typechecking each of e_a and e_b that satisfy the I0 invariant with h their respective dynamic configurations (d_a, s_a) and (d_b, s_b). We recall the proof of Theorem 2.3, in which we argued that given a static typing for an expression of the form $E[e]$ dynamically sound w.r.t. some d, h, s, we can generate static contexts that type-check e and are sound w.r.t. the same dynamic contexts d, h, s. We apply that logic here to obtain static typings $\mathcal{H}_a; \Gamma_a; \Omega_a; \mathrm{P}_a \vdash \texttt{send-}\tau(l) : r_u\ \texttt{unit} \dashv \mathcal{H}'_a; \Gamma'_a; \Omega'_a$ and $\mathcal{H}_b; \Gamma_b; \Omega_b; \mathrm{P}_b \vdash \texttt{recv-}\tau() : r\ \tau \dashv \mathcal{H}'_b; \Gamma'_b; \Omega'_b$ with I0($\mathcal{H}_a, \Gamma_a, \Omega_a, \mathrm{P}_a, d_a, h, s_a$) and I0($\mathcal{H}_b, \Gamma_b, \Omega_b, \mathrm{P}_b, d_b, h, s_b$). Letting $d_{sep} = \textit{live-set}(r\, \dot{}\, \langle\rangle; \cdot; l : r\ \tau; h; \cdot)$ as chosen in the premise of EC3 - Communication-Paired-Step, we wish to show that $d_{sep} \subseteq d_a$ and $d_{sep} \cap d_b = \emptyset$. This suffices to conclude preservation in this case, as it allows us to apply EC3 to obtain $h \vdash (d_a, e_a; d_b; e_b) \xrightarrow{\text{comm-eval}} (d'_a, e'_a; d'_b, e'_b)$, which is then sufficient to apply EC2 and obtain $(h, \overline{\langle d_n, s_n, e_n \rangle}) \xrightarrow{\text{concur-eval}} (h, \overline{\langle d'_n, s'_n, e'_n \rangle})$. We now proceed to show $d_{sep} \subseteq d_a$ and $d_{sep} \cap d_b = \emptyset$. We note that the latter follows from the former, as disjointness of d_a and d_b is guaranteed by their presence in the *sound* concurrent configuration $(h, \overline{\langle d_n, s_n, e_n \rangle})$. Also from the definition of *sound*, we know that invariant I1 - Reservation-Sufficiency holds for $(\mathcal{H}_a, \Gamma_a, \mathrm{P}_a, d_a, h, s_a)$, which tells us $\textit{live-set}(\mathcal{H}_a, \Gamma_a, \mathrm{P}_a, h, s_a) \subseteq d_a$. It thus suffices to show $d_{sep} = \textit{live-set}(r\, \dot{}\, \langle\rangle; \cdot; l : r\ \tau; h; \cdot) \subseteq \textit{live-set}(\mathcal{H}_a, \Gamma_a, \mathrm{P}_a, h, s_a)$. By inversion of T16 - Send and T1 - Location-Ref, we know that $l : r'\ \tau \in \mathrm{P}_a$ for some region r', and since the r chosen in d_{sep}'s definition is an arbitrary placeholder, let $r = r'$. Now let $l_a \in \textit{live-set}(r\, \dot{}\, \langle\rangle; \cdot; l : r\ \tau; h; \cdot)$ be arbitrary. To conclude our proof, it suffices to show $l_a \in \textit{live-set}(\mathcal{H}_a, \Gamma_a, \mathrm{P}_a, h, s_a)$. From the definition of *live-set* we know that there exists some $(r_o \overset{L_o}{\dashrightarrow} l_a) \in \textit{outward-paths}(r\, \dot{}\, \langle\rangle; \cdot; l : r\ \tau; h; \cdot)$, and from the definitions of *outward-paths* and *live-roots* we can see that necessarily $r_o = r$, and L_o is the *iso-subseq*of some continuous sequence of locations and fields $l, f_0, l_1, f_1, \ldots, l_k, f_k$, noting that (l, r) is the only element of $\textit{live-roots}(r\, \dot{}\, \langle\rangle; \cdot; l : r\ \tau; h; \cdot)$. It suffices to show that $(r \overset{L_o}{\dashrightarrow} l_a)$ is in the full configuration's set $\textit{outward-paths}(\mathcal{H}_a, \Gamma_a, \mathrm{P}_a, h, s_a)$. From our observation that $l : r\ \tau \in \mathrm{P}_a$, we can see that (l, r) is still in the *live-roots* for the full configuration, and the same heap h is used, so the only way that $(r \overset{L_o}{\dashrightarrow} l_a)$ could fail to be in the full configurations *outward-paths* is if some field in L_o is in $\textit{tracked-refs}(\mathcal{H}_a, s_a)$. For the sake of contradiction, let $l_{bad}, f_{bad} \in L_o \cap \textit{tracked-refs}(\mathcal{H}_a, s_a)$. Then there exists x_{bad} such that $s_a(x_{bad}) = l_{bad}$, and x_{bad} is tracked under some region r_{bad} in $\mathcal{H}_a$. Since r itself has no focused variables by inversion of T16, we know that $r_{bad} \neq r$. But now (l_{bad}, r_{bad}) is a live root, so we now have two distinct *outward-paths* terminating at l_{bad}, one generated from a prefix of L_o and beginning at r, and one of the form $(r_{bad} \overset{\cdot}{\dashrightarrow} l_{bad})$. This contradicts I2 - Tree-Of-Untracked-Regions for the configuration, so we can conclude no such l_{bad} exists, and thus l_a is in the full configuration's *live-set*, concluding our argument that $d_{sep} \subseteq d_a$ and giving us applications of EC3 and EC2 that conclude progress in this case.

We have considered both cases under which our given configuration could be *non-blocked* and shown Progress in each, allowing us to conclude Progress holds in general.

2.9.5 Proof of Concurrent Preservation. We are given a *sound* concurrent configuration $(h, \overline{\langle d_n, s_n, e_n \rangle})$, and a step $(h, \overline{\langle d_n, s_n, e_n \rangle}) \xrightarrow{\text{concur-eval}} (h', \overline{\langle d'_n, s'_n, e'_n \rangle})$. We wish to show that the concurrent configuration $(h', \overline{\langle d'_n, s'_n, e'_n \rangle})$ is *sound* as well. We split into two cases based on whether EC1 - Concurrent-Single-Step or EC2 - Concurrent-Paired-Step derived our step:

EC1: For some i, the configuration (d_i, h, s_i, e_i) individually steps as $(d_i, h, s_i, e_i) \xrightarrow{\text{EVAL}} (d'_i, h', s'_i, e'_i)$. From single-threaded Preservation (theorem 2.3) applied to the know properties of thread i in a *sound* concurrent configuration, we can obtain static contexts such that $\mathcal{H}_i; \Gamma_i; \Omega_i; \mathrm{P}_i \vdash e'_i : r_i\ \tau_i \dashv \mathcal{H}'_i; \Gamma'_i; \Omega'_i$ and I0($\mathcal{H}_i, \Gamma_i, \Omega_i, \mathrm{P}_i, d'_i, h', s'_i$). To conclude that the full post-step concurrent configuration $(h', \overline{\langle d'_n, s'_n, e'_n \rangle})$ is *sound*, it suffices to show that $(d'_i - d_i) \subseteq (h' - h)$, and thus pairwise disjointness of all d_j for $j \leq n$ implies pairwise disjointess of d'_j for all $j \leq n$ by the upper bound in invariant I1 - Reservation-Sufficiency, and to show that the update $h \mapsto h'$ does not invalidate I0 for any other thread $j \neq i$. The former property is easily verified by inspection of the E rules; E1 - Evaluation-Context-Step, E4 - Contextual-Reference-Step and E6 - Contextual-Assignment-Step are the inductive cases, and E10 - New-Loc-Step and E12 - OPlus-Step are the only nontrivial base cases in which d grows, and h can be clearly seen to grow in parallel. The latter property requires us to perform the following reasoning. First, that single-threaded steps exactly preserve the heap outside their reservation, i.e. for any step $(d, h, s, e) \xrightarrow{\text{EVAL}} (d', h', s', e')$, any location $l \in dom(h) - d$ satisfies $h'(l) = h(l)$, and second, that dynamic soundness of a configuration $(\mathcal{H}, \Gamma, \Omega, \mathrm{P}, d, h, s)$ is preserved under replacement of h with h' as long as $h' \restriction_d = h \restriction_d$ and I3(h') holds. Together, these two properties are sufficient to conclude that all threads are dynamically sound with the updated h', because the former tells us that the updates occur only in thread i's reservation d_i, pre-step soundness tells us that all other threads' reservations are disjoint from d_i, and the latter property tells us that updates to the heap strictly outside of a thread's reservation do not invalidate that thread's I0 invariant, noting that single-threaded preservation gave us I3 on the new heap h'. Of the two properties that we now must verify to conclude our proof, the former is verified through simple inspection of the E rules, noting the cases E7a - Final-Assignment-Step–Variable, E7b - Final-Assignment-Step–Location, E10, and E12 as the only ones that update h, and always do so in a manner that preserves h outside their own reservation. To show the latter property, we reason that any dependence on h in the invariants I1-I7 - Region-Names-Bounding depends only at locations within the *live-set*, which by I1 is a subset of d. I1 and I2 - Tree-Of-Untracked-Regions are easily seen to be preserved, as they depend on h only through the set of *outward-paths*, and any location that is part of an outward path is already in the *live-set*, so changes to h outside the *live-set* will not change the *outward-paths*. I3 need not be verified individually, as its preservation is a premise of the property we are verifying. I4 - Location-Type-Consistency depends on h only at locations that are also *live-roots*, and thus in the *live-set*. I5 - Variable-Region-Consistency-I7 do not depend on h. We have now shown sufficient properties of our E steps and I invariants to conclude that single-threaded Preservation lifts to concurrent preservation, and we can conclude our proof in this case.

EC2 - Concurrent-Paired-Step: For some a, b, we are given the communication step $h \vdash (d_a \uplus d_{sep}, E^*_a[\texttt{send-}\tau(l_{root})]; d_b, E^*_b[\texttt{recv-}\tau()]) \xrightarrow{\text{comm-eval}} (d_a, E^*_a[\texttt{new-unit}]; d_b \uplus d_{sep}, E^*_b[l_{root}])$, and we wish to show that the concurrent configuration resulting from performing this step is *sound*. Since h, and d_i, s_i, e_i for all $i \notin \{a, b\}$ are preserved exactly, we needn't worry about reasoning about preservation for any other thread; our proof obligation is to demonstrate that there exist static typing contexts that admit well-typedness for the resulting expressions in threads a, b and that are dynamically sound w.r.t. the post-step reservations. We omit some details from this proof that are purely structural, and already covered in the proof of single-threaded preservation (theorem 2.3) such as lifting well-typedness and dynamic soundness through evaluation contexts (see the E1 - Evaluation-Context-Step, E4 - Contextual-Reference-Step and E6 - Contextual-Assignment-Step cases), and the occurance in the typing derivation of TS1 - Virtual-Transformation-Structural (entire case written up at beginning of proof) or TS2 - Framing-Structural (*normal form* guarantees this does not occur when the redex is a `send` or `recv`). With this structural reasoning in hand, we phrase our proof goal as:

$$\text{given: } \mathcal{H}_a, r^{\cdot}\langle\rangle; \Gamma_a; \Omega_a; \mathrm{P}_a \vdash \texttt{send-}\tau(l) : r_u \; \texttt{unit} \dashv \mathcal{H}_a, r_u^{\cdot}\langle\rangle; \Gamma_a; \Omega_a \uplus \{r_u\}$$

$$\mathcal{H}_b, \Gamma_b; \Omega_b; \mathrm{P}_b \vdash \texttt{recv-}\tau() : r\;\tau \dashv \mathcal{H}_b, r^{\cdot}\langle\rangle; \Gamma_b; \Omega_b \uplus \{r\}$$

$$\mathrm{I0}(\mathcal{H}_a, r^{\cdot}\langle\rangle; \Gamma_a; \Omega_a; \mathrm{P}_a; d_a \uplus d_{sep}; h; s_a)$$

$$\mathrm{I0}(\mathcal{H}_b; \Gamma_b; \Omega_b; \mathrm{P}_b; d_b; h; s_b)$$

$$\text{prove: } \mathcal{H}_a; \Gamma_a, \Omega_a, \mathrm{P}_a \vdash \texttt{new-unit} : r_u \; \texttt{unit} \dashv \mathcal{H}_a, r_u^{\cdot}\langle\rangle; \Gamma_a; \Omega_a \uplus \{r_u\}$$

$$\mathcal{H}_b, r^{\cdot}\langle\rangle; \Gamma_b; \Omega_b \uplus \{r\}; \mathrm{P}_b, l : r\;\tau \vdash l : r\;\tau \dashv \mathcal{H}_b, r^{\cdot}\langle\rangle; \Gamma_b; \Omega_b \uplus \{r\}$$

$$\mathrm{I0}(\mathcal{H}_a; \Gamma_a; \Omega_a; \mathrm{P}_a; d_a; h; s_a)$$

$$\mathrm{I0}(\mathcal{H}_b, r^{\cdot}\langle\rangle; \Gamma_b; \Omega_b \uplus \{r\}; \mathrm{P}_b, l : r\;\tau; d_b \uplus d_{sep}; h; s_b)$$

The two well-typedness goals are easily obtained as application of T10 - New-Loc and T1 - Location-Ref respectively, so we proceed to prove the two dynamic soundness results:

$\boxed{\mathrm{I0}(\mathcal{H}_a; \Gamma_a; \Omega_a; \mathrm{P}_a; d_a; h; s_a)}$ This is very similar to the given soundness result $\mathrm{I0}(\mathcal{H}_a, r^{\cdot}\langle\rangle; \Gamma_a; \Omega_a; \mathrm{P}_a; d_a \uplus d_{sep}; h; s_a)$. For each invariant I1 - Reservation-Sufficiency-I7 - Region-Names-Bounding, we reason that it is preserved by the update from the latter to the former.

- I1 and I2 - Tree-Of-Untracked-Regions: We claim that *outward-paths*$(\mathcal{H}_a; \Gamma_a; \mathrm{P}_a; h; s_a) \subseteq \{(r_o \overset{L_o}{\dashrightarrow} l_o) \in$ *outward-paths*$(\mathcal{H}_a, r^{\cdot}\langle\rangle; \Gamma_a; \mathrm{P}_a; h; s_a) \mid l_o \notin d_{sep}\}$. This claim gives us preservation of I1 because then the post-step *live-set* does not exceed the pre-step *live-set* minus d_{sep}, and it gives us preservation of I2 because any pair of *outward-paths* that violated the post-step condition would necessarily also violate the pre-step condition. To show contaiment, let $O = (r_o \overset{L_o}{\dashrightarrow} l_o) \in$ *outward-paths*$(\mathcal{H}_a; \Gamma_a; \mathrm{P}_a; h; s_a)$. We wish to show $O \in$ *outward-paths*$(\mathcal{H}_a, r^{\cdot}\langle\rangle; \Gamma_a; \mathrm{P}_a; h; s_a)$ and $l_o \notin d_{sep}$. The former is easy to see, as the addition of an empty tracked region does not affect any of the conditions that establlish O's membership in the *outward-paths* set. For the latter, we recall that d_{sep} is exactly the set of locations reachable from l, which is a live root in region r, so if $l_o \in d_{sep}$, then I2 would be violated because *outward-paths*$(\mathcal{H}_a, r^{\cdot}\langle\rangle; \Gamma_a; \mathrm{P}_a; h; s_a)$ would contains two outward paths terminating at l_o, one starting at r and the other at r_o, and since $(r_o \overset{L_o}{\dashrightarrow} l_o) \in$ *outward-paths*$(\mathcal{H}_a; \Gamma_a; \mathrm{P}_a; h; s_a)$ where $r \notin$ *regs*$(\mathcal{H}_a)$, $r_o \neq r$. This establishes containment, so we can conclude that I1 and I2 are preserved by this step.
- I3 - Heap-Closure: This invariant does not depend on $\mathcal{H}$, so is trivially preserved.
- I4 - Location-Type-Consistency: This invariant depends on $\mathcal{H}$ only through *regs*$(\mathcal{H})$, and a shrinking of *regs*$(\mathcal{H})$ as occurs in this step only weakens the invariant.
- I5 - Variable-Region-Consistency and I6 - Focus-Non-Aliasing: These invariants depend on $\mathcal{H}$ only through its focused variables, which are not affected by this step, so the invaraints are easily seen to be preserved.
- I7: No new region names are introduced, so this invariant is preserved

We have argued that each invariant I1-I7 is preserved by the step, and thus we can conclude that the desired dynamic soundness result for thread a's post-step configuration holds.

$\boxed{\mathrm{I0}(\mathcal{H}_b, r^{\cdot}\langle\rangle; \Gamma_b; \Omega_b \uplus \{r\}; \mathrm{P}_b, l : r\;\tau; d_b \uplus d_{sep}; h; s_b)}$ This is very similar to the given soundness result $\mathcal{H}_b; \Gamma_b; \Omega_b; \mathrm{P}_b \vdash \texttt{recv-}\tau() : r\;\tau \dashv \mathcal{H}_b, r^{\cdot}\langle\rangle; \Gamma_b; \Omega_b \uplus \{r\}$. For each invariant I1-I7, we reason that it is preserved by the update from the latter to the former.

I1: Let $l_o \in$ *live-set*$(\mathcal{H}_b, r^{\cdot}\langle\rangle; \Gamma_b; \mathrm{P}_b, l : r\ \tau; h; s_b)$. Then $(r_o \overset{L_o}{\dashrightarrow} l_o) \in$ *outward-paths*$(\mathcal{H}_b, r^{\cdot}\langle\rangle; \Gamma_b; \mathrm{P}_b, l : r\ \tau; h; s_b)$ for some r_o, L_o. If $r_o \neq r$, then whatever $(l_0, r_o) \in$ *live-roots*$(\mathcal{H}_b, r^{\cdot}\langle\rangle; \Gamma_b; \mathrm{P}_b, l : r\ \tau; h; s_b)$ generated $(r_o \overset{L_o}{\dashrightarrow} l_o)$ is also in *live-roots*$(\mathcal{H}_b; \Gamma_b; \mathrm{P}_b; h; s_b)$, so by I1 on the old configuration is contained in d_b. If $r_o = r$, then (l, r) is the live root that generated $(r_o \overset{L_o}{\dashrightarrow} l_o)$, so $l_o \in d_{sep}$ by the definition of d_{sep} as the reachable locations from l. In either case $l_o \in d_b \uplus d_{sep}$, so I1 holds on the new configuration.

I2 - Tree-Of-Untracked-Regions: For the sake of contradiction, let $(r_o \overset{L_o}{\dashrightarrow} l_o), (r'_o \overset{L'_o}{\dashrightarrow} l_o) \in$ *outward-paths*$(\mathcal{H}_b, r^{\cdot}\langle\rangle; \Gamma_b; \mathrm{P}_b, l : r\ \tau; h; s_b)$, where $(r_o, L_o) \neq (r'_o, L'_o)$. The three cases we split into are whether both, exactly one, or neither of r_o, r'_o are equal to r. In the first case, $r_o = r'_o = r$, and necessarily (l, r) is the live root that generated both outward paths. But we note that in our original configuration, inversion of T16 - Send and T1 - Location-Ref tells us $(l : r\ \tau) \in \mathrm{P}_a$, so $(r_o \overset{L_o}{\dashrightarrow} l_o), (r'_o \overset{L'_o}{\dashrightarrow} l_o) \in$ *outward-paths*$(\mathcal{H}_a, r^{\cdot}\langle\rangle; \Gamma_a; \mathrm{P}_a; h; s_a)$, which contradicts the given dynamic soundness for thread a: I0$(\mathcal{H}_a, r^{\cdot}\langle\rangle; \Gamma_a; \Omega_a; \mathrm{P}_a; d_a \uplus d_{sep}; h; s_a)$. Thus we cannot have $r_o = r'_o = r$. Next we consider the case in which neither r_o nor r'_o are equal to r. But then $(r_o \overset{L_o}{\dashrightarrow} l_o), (r'_o \overset{L'_o}{\dashrightarrow} l_o) \in$ *outward-paths*$(\mathcal{H}_b; \Gamma_b; \mathrm{P}_b; h; s_b)$, which violates I2 for thread b's pre-step configuration, and we observe we cannot have this case either. Finally, we consider the case in which exactly one of r_o, r'_o is equal to r. But then, for the respective reasons of each of the above cases, we can conclude that l_o must be in both thread a and thread b's pre-step *live-set*, which contradicts the disjointness component of a *sound* concurrent configuration. We can conclude that no pair of outward paths may exist that vioalte I2 in thread b's post-step configuration, and thus I2 is preserved.

I3 - Heap-Closure, I5 - Variable-Region-Consistency, I6 - Focus-Non-Aliasing, I7 - Region-Names-Bounding: Trivially preserved for the same reasons as for thread a.

I4 - Location-Type-Consistency: By I7 on thread b's pre-step configuration, neither P_b nor Γ_b contains any mention of region r because Ω_b does not contain r, so the only strengthening of I4 that occurs by adding r to *regs*$(\mathcal{H}_b)$ occurs through the replacement of P_b with of $\mathrm{P}_b, l : r\ \tau$, which requires us to check that $h_\tau(l) = \tau$. This is provided by I4 on thread a's pre-step configuration, so we are done.

We have argued that each invariant I1 - Reservation-Sufficiency-I7 is preserved by the step, and thus we can conclude that the desired dynamic soundness result for thread b's post-step configuration holds.

We have considered both cases under which our given step could be derived, and shown Preservation in each, allowing us to conclude Preservation holds in general.

3 VIRTUAL COMMAND INFERENCE

Implementation of an efficient type-checker for this system relies on an efficient decision procedure for the application of the structural rules TS1 - Virtual-Transformation-Structural and TS2 - Framing-Structural, which are the only rules whose application is not syntax-directed. As argued in our proofs of progress and preservation, typing derivations in which TS2 is applied only to T9 - Function-Application and other instances of TS2 are fully general, so an efficient decision procedure for application of the structural rules at function application sites suffices, which we provide in section 3.3. Application of TS1 will be much more pervasive, being required to succesfully type-check many common expressions such as isolated field reference and assignment. This is easily accomplished through a greedy approach, detailed in section 3.1. TS1 is also needed in the case that we are explicitly given two $\mathcal{H}, \Gamma$ pairs, and want to transform the former into the latter. This case occurs at the conclusion of function bodies, to coerce the derived output contexts of the body into the output contexts expected by the function signature, and in while loops, where the output of typechecking the body must be coerced to match the input. This coercion can be accomplished through targetted re-use of the greedy algorithms used elsewhere, and is detailed in section 3.2. The only remaining case requiring virtual command inference is conditionals, including `if`, `let some` and `if disconnected`, in which, given two $\mathcal{H}, \Gamma$ pairs representing the outputs at the ends of respective branches, we seek to find a third $\mathcal{H}, \Gamma$ pair with coercions from both outputs. Unfortunately, our approach here is only efficiently decidable in the case of an oracle, or by replacing that oracle with a heuristic that marginally limits expressiveness. We provide details of our approach and its limitations in section 3.4.

3.1 Common Expressions

Syntax-direction is sufficient to guide the application of all T rules in the construction of typing derivations for a given expression. However, many T rules place restrictions on contexts $\mathcal{H}, \Gamma$ that will not hold without prior V rule transformations. In this section we detail the common expressions for which V rules can be used to construct a typing derivation. Many times, an application of a rule V1 - Focus, V2 - Unfocus, V3 - Explore or V4 - Retract will be blocked by conflicting structure in $\mathcal{H}$. Greedily, we can apply subordinate V rules to coerce $\mathcal{H}$ to match the desired LHS if possible. For V1, we must apply V2 to unfocus any other focused variables in the same region as our target variable. For V2, we must apply V4 to retract any fields that are currently explored in our target variable. For V3, we must ensure that our target variable is itself focused, so we defer to application of V1. For V4 we must ensure that the region currently pointed to by out target field has no focused variables, so we apply V2 as necessary. In each of these cases, failure indicates that the original requested virtual command cannot be applied.

- $\boxed{l}$ Given a location l, if it is not bound in P then no virtual commands will alter the inapplicability of T1 - Location-Ref. Similarly, if it is bound but to a region not tracked by $\mathcal{H}$, no virtual commands will be of assistance, as they introduce only fresh regions to $regs(\mathcal{H})$.
- $\boxed{x}$ Similar to the case for locations, no virtual commands are of assistance in application of T2 - Variable-Ref.
- $\boxed{e; e}$ No inference required at this level
- $\boxed{e.f}$ If this is a non-isolated field reference, then no inference is required. If it is an isolated field reference, then we must ensure that it is explored, which we do by greedy application of V3 as detailed above.
- $\boxed{e.f = e}$ If this is a non-isolated field assignment, then we must ensure that the source and target region are the same, which we do through a single application of V5 - Attach. If this is an isoalted field assignment, then we must ensure that the source field is explored, which is done by greedy application of V3.
- $\boxed{x = e}$ Succesfully application of T8 - Assign-Var relies only on unfocussing x, done greedily by V2.
- $\boxed{fn(x, \ldots, x)}$ This case is nontrivial, and deferred to section 3.3
- $\boxed{e \oplus e}$ Similar to the case for sequence, no virtual commands are of assistance at this level.
- $\boxed{\texttt{new}\ \tau}$ No virtual commands are of assistance
- $\boxed{\texttt{declare}\ x : \tau\ \texttt{in}\ \{e\}}$ No virtual commands are of assistance prior to the the `declare`, but after typechecking the body greedy application of V2 could effect successful typechecking.
- $\boxed{\texttt{if}\ (e)\ \{e\}\ \texttt{else}\ \{e\}}$ This case is nontrivial, and deferred to section 3.4
- $\boxed{\texttt{while}\ (e)\ \{e\}}$ This case is nontrivial, and deferred to section 3.2
- $\boxed{\texttt{send-}\tau(e)}$ Greedy application of V2 is useful to ensure the region of the expression being sent is empty.

`recv-`τ`()`	As for `new`, virtual commands are of no assistance here.
`if-disconnected`(x, x) $\{e\}$ `else` $\{e\}$	This case is nontrivial, and deferred to section 3.4
`none` τ	Same as `recv`
`some`(e)	Same as `send`
`let some`(x) = (e) `in` $\{e\}$ `else` $\{e\}$	This case is nontrivial, and deferred to section 3.4

By noting the cases in which greedy application of V1 - Focus-V4 - Retract suffices to effect any possible successful typechecking, we have deferred all of the nontrivial inference problems to the following sections.

3.2 Coercion

This section deals with the most straightforward nontrivial inference problem, that is directly dispatched to by inference for T14 - While-Loop and used as a subroutine by the framing procedure below for function application. Namely, given $\mathcal{H}, \Gamma, \mathcal{H}', \Gamma'$, determine whether a sequence of virtual command transformations (as given by V rules) suffices to transform $(\mathcal{H}, \Gamma)$ into $(\mathcal{H}', \Gamma')$. There are 3 steps, and at each we declare failure only if we know that no such sequence exists.

(1) Examine the set of variables focused in either $\mathcal{H}$ or $\mathcal{H}'$. Any variable focused in $\mathcal{H}$ but not $\mathcal{H}'$ needs to be unfocussed, so we greedily attempt to apply V2 - Unfocus, and any focused in $\mathcal{H}'$ but not $\mathcal{H}$ needs to be focused, so we greedily attempt to apply V1. As discussed above, these greedy calls will fail only if the desired focus or unfocus is impossible, so this stage will terminate with either $\mathcal{H}$ and $\mathcal{H}'$ with the same set of tracked variables, or with necessary failure.
(2) Examine the set of fields explored in either $\mathcal{H}$ or $\mathcal{H}'$. Similarly to the stage for variables we greedily explore (V3 - Explore) or retract (V4) to obtain agreement between the explored fields of $\mathcal{H}$ and $\mathcal{H}'$ or fail. If any fields are encountered that cannot be retracted in $\mathcal{H}$ because their target region is untracked, and additionally if they are tracked but with target bottom in $\mathcal{H}'$, we also have the option to apply V9 - Invalidate-Field to retarget them to bottom. If the target was a non-bottom region in $\mathcal{H}'$, then coercion of $\mathcal{H}$ to $\mathcal{H}'$ is not possible, but in some cases, such as while loops, the same V9 transformation can also be applied before the body so that our coercion target is indeed achievable.
(3) Finally, we are guaranteed that $\mathcal{H}$ and $\mathcal{H}'$ match in every way but region names, so we apply V5 - Attach as necessary to attach the regions of variables and of field targets in $\mathcal{H}$ to the corresponding regions in $\mathcal{H}'$. This attaching should also unify Γ to Γ', but could fail if region names exist in Γ that are not tracked in $\mathcal{H}$. If the corresponding variables are not in the bottom region in Γ', we need to request as above that the caller apply V8 - Invalidate-Variable to invalidate them in the target before requesting coercion or no coercion is possible. If the corresponding variables are in the bottom region in Γ', then we conclude by application of V8.

3.3 Framing

In this section we will deal with the obligation from above of inference at function call sites. In particular, we will be given $\mathcal{H}', \Gamma', \Omega'$ as output contexts from the prior typechecked expression, and $\mathcal{H}, \Gamma, \Omega$ as desired input contexts to the function. We assume that variable and region names have already been chosen appropriately through the freedom granted by the injections Φ_r, Φ_x in T9 - Function-Application. Framing consists of a sequence of F rule applications detailed below, followed by a final call to coercion (3.2) to ensure completeness.

(1) First, for each pinned variable present in $\mathcal{H}$ that is also present (possibly unpinned) in $\mathcal{H}'$, we apply F5 - Field-Framing to add its contents from $\mathcal{H}'$ that are not already present in $\mathcal{H}$. If this variable is pinned in $\mathcal{H}$ but not $\mathcal{H}'$, we apply F4 - Variable-Pinnedness-Framing as well.
(2) Next, for each pinned region present in $\mathcal{H}$ that is also present (possibly unpinned) in $\mathcal{H}'$, we apply F3 - Tracked-Variable-Framing to add its contents from $\mathcal{H}'$ that are not already present in $\mathcal{H}$, noting that this could fail by necessity.
(3) Next, we examine the partition of regions, and if $\mathcal{H}$ admits distinction between any two pinned regions not distinct in $\mathcal{H}'$, we apply F8 - Attach-Pinned-Regions-Framing. After all such attaches have been made, we apply F2 - Region-Pinnedness-Framing to any regions pinned in $\mathcal{H}$ but not $\mathcal{H}'$.
(4) Finally, we apply F1 - Region-Framing and F6 - Variable-Framing to expand $\mathcal{H}$, Γ, and Ω on the top level to include any structure present in $\mathcal{H}', \Gamma', \Omega'$.

This sequence of steps utilizes F1-F8 as fully as possible to attempt to frame a T9 - Function-Application-derived judgment to match $\mathcal{H}', \Gamma', \Omega'$, and to check if any remaining V rules may be applied to effect a match not already made, we dispatch to coercion and conclude.

3.4 Unification

In all prior cases, we were able to provide an efficient (namely, linear in the size of the current $\mathcal{H}, \Gamma, \Omega$) decision procedure for the application of TS1 - Virtual-Transformation-Structural and TS2 - Framing-Structural to construct typing derivations. Unfortunately, no such procedure exists for conditionals (T13 - If-Statement) without an oracle. Formally, we are given $\mathcal{H}_t, \Gamma_t, \mathcal{H}_f, \Gamma_f$ and wish to compute $\mathcal{H}', \Gamma'$ with virtual command transformations from each of $\mathcal{H}_t, \Gamma_t$ and $\mathcal{H}_f, \Gamma_f$. To illustrate why local (i.e. without backtracking) algorithms fail to find such $\mathcal{H}', \Gamma'$ consider the following example:

$$\begin{aligned}
\mathcal{H}_t &= \dot{r_x}\langle x[f \rightarrowtail r]\rangle, \dot{r_y}\langle y[f \rightarrowtail r]\rangle, \dot{r}\langle\rangle \\
\mathcal{H}_f &= \dot{r_x}\langle\rangle, \dot{r_y}\langle\rangle \\
\Gamma_t = \Gamma_f &= x : r_x\ \tau, y : r_y\ \tau \\
\mathcal{H}'_x &= \dot{r_x}\langle\rangle, \dot{r_y}\langle y[f \rightarrowtail \bot]\rangle \\
\mathcal{H}'_y &= \dot{r_x}\langle x[f \rightarrowtail \bot]\rangle, \dot{r_y}\langle\rangle
\end{aligned}$$

Both $\mathcal{H}'_x$ and $\mathcal{H}'_y$ are possible targets for unification of the two branches $\mathcal{H}_t$ and $\mathcal{H}_f$. If later in the program, $\mathsf{send} - \tau(z)$ is called for some z, it is possible that this could fail because z is in region r_x at that time and the unification algorithm chose $\mathcal{H}'_y$ as its target. Since invalid fields are un-retractable, there is no way that the call to send will type-check in this case. Conversely, that call could fail if z is in region r_y and unification chose $\mathcal{H}'_x$. Since there is no way to determine what region variables will be in before running the type system, there is no way to determine which of $\mathcal{H}'_x$ and $\mathcal{H}'_y$ should be chosen at the point of T13 application. A similar problem arises with the decision to, or not to, invalidate fields:

$$\mathcal{H}_t = \dot{r_x}\langle x[f \rightarrowtail r_1]\rangle, r_1\langle\rangle, r_2\langle\rangle$$
$$\mathcal{H}_f = \dot{r_x}\langle x[f \rightarrowtail r_2]\rangle, r_1\langle\rangle, r_2\langle\rangle$$
$$\Gamma_t = \Gamma_f = x : r_x\ \tau, y_1 : r_1\ \tau, y_2 : r_2\ \tau$$
$$\mathcal{H}_0; \Gamma_0 = \dot{r_x}\langle x[f \rightarrowtail \bot]\rangle, r_1\langle\rangle, r_2\langle\rangle; \Gamma_t$$
$$\mathcal{H}_1; \Gamma_1 = \dot{r_x}\langle x[f \rightarrowtail r_1]\rangle, r_1\langle\rangle; x : r_x\ \tau, y_1 : r_1\ \tau, y_2 : r_2\ \tau$$

As above, we have identified two possible targets for unification $\mathcal{H}_0, \Gamma_0$ and $\mathcal{H}_1, \Gamma_1$. The former will allow us to `send` region r_1 away in the future while still permitting us to access r_2 afterwards (for example, `send`$(z_1); z_2$ for z_1, z_2 in regions r_1, r_2 respectively), and the latter will allow us to send region r_x. Since it is impossible to determine which of these actions may be attempted after the unification without running the type system forwards, this highlights another case in which local inference fails.

4 IF DISCONNECTED ALGORITHM

In this section we provide a verbal description of the `if disconnected` algorithm, an argument for its efficiency, and a prototype C++ implementation.

4.1 System Revisions to Support the Check

Change the heap context: h now maps locations to a 4-tuple (type,value,refcount,traversal-state)

Traversal-state itself is an enum with five possible values:

- (untraversed)
- (left-owned,partially-traversed)
- (right-owned,partially-traversed)
- (left-owned,fully-traversed)
- (right-owned,fully-traversed)

Change the assignment rule, splitting it into an isolated and non-isolated case (isolated case needs to actually check if isolated, otherwise dispatch to location-case)

Location case:

```
l, l' ∈ d
-----------------------------------------------
(d,h ⊎ (l ↦ (τ,v[f ↦ lₒ],rc,ts), l' ↦ (τ',v',rc',ts'), lₒ ↦ (τₒ,vₒ,rcₒ,tsₒ)),s,l.f = l') -->
(d,h ⊎ (l ↦ (τ,v[f ↦ l'],rc,ts), l' ↦ (τ',v',rc'+1,ts'), lₒ ↦ (τₒ,vₒ,rcₒ-1,tsₒ)),s, l')
```

We now track refcounts on assignment to locations. All refcounts are initialized to zero and all traversal-state is initialized to untraversed.

Traversal algorithm: alternate between the left and right. Let `role` be `left` for the left traversal and `right` for the right traversal. The `yield` keyword indicates a switch between the traversals. WLOG we start with left.

4.2 Prototype Implementation

```
#include <list>

using namespace std;

using type = int; using value = int;
using refcount_t = unsigned long long;

enum class owner_status{unknown, left, right};


#define N 4

struct object{
    value v{};
    type t{};
    const refcount_t existing_refcount{0};
    refcount_t discovered_refcount{0};
    traversal_status status{traversal_status::untraversed};
    owner_status owner{owner_status::unknown};
    list<object> next_hops;
    constexpr object() = default;
    constexpr object(const &object) = delete;
    constexpr ~object() = default;
};

constexpr bool visit(owner_status role, object &o, list<object>& discovered_objects){
    assert(role != owner_status::unknown);
    switch(o.owner){
    case owner_status::unknown:
    {
  o.owner = role;
  discovered_objects.push_back(o);
  yield;
  for (object &next_object : o.next_hops){
      next_object.discovered_refcount++;
      visit(role,next_object,discovered_objects);
  }
  return true;
    }
    default:
  return o.owner == role;
    }
}


constexpr bool determine_split(owner_status role, object &root){
    list<object> discovered_objects;
    if (visit(role,root,discovered_objects)){
  for (auto object : discovered_objects){
      if (object.discovered_refcount < object.existing_refcount)
    return false;
```

```
  }
  return true;
    }
    else return false;

}
```

5 CODE EXAMPLES

5.1 Singly Linked List

```
struct sll {
    iso head : sll_node?;
}

struct sll_node {
    iso payload : data;
    iso next : sll_node?;
}

def new_sll() : sll {
    let l = new sll;
    l.head = none sll_node;
    l
}


def new_sll_node(d : data) : sll_node
    consumes d {

    let n = new sll_node;
    n.next = none sll_node;
    n.payload = d;
    n
}

def is_none(opt_n : sll_node?) : bool {
    let some(n) = opt_n in { false } else { true }
}

def remove_tail(n: sll_node) : data? {
    let some(next) = n.next in {
  if (is_none(next.next)) {
      n.next = none sll_node;
      some(next.payload)
  } else {
      remove_tail(next)
  }
    } else { none data }
}

def swap_tails(left, right : sll_node) : unit {
```

```
    let some(left_tail) = remove_tail(left) in {
        let some(right_tail) = remove_tail(right) in {
      push_tail(left, right_tail);
      push_tail(right, left_tail);
      skip
  }
    }
}

def pop(l : sll) : data? {
    let some(n) = l.head in {
  l.head = n.next;
  some (n.payload)
    } else {
  none data
    }
}

def remove_from(l : sll, pos : int) : data? {

    let some(n) = l.head in {
  if (pos == 0) {
      l.head = n.next;
      some (n.payload)
  } else {
      remove_from_nonhead(n, pos)
  }
    } else {
  none data
    }
}


def remove_from_nonhead(n: sll_node, pos : int) : data? {

    let some(next) = n.next in {
  if (pos == 1) {
      n.next = next.next;
      some(next.payload)
  } else {
      remove_from_nonhead(next, pos - 1)
  }
    } else {
  none data
    }
}

def insert_at(l : sll, d : data, pos : int) : unit
    consumes d {

    if (pos == 0) {
```

```
  let n = new_sll_node(d);
  n.next = l.head;
  l.head = some(n);
  skip
    } else {
  let some(n) = l.head in {
      insert_at_nonhead(n, d, pos);
      skip
  } else {
      l.head = some(new_sll_node(d));
      skip
  }
    }
}



def insert_at_nonhead(n : sll_node, d : data, pos : int) : unit
    consumes d {

    if (pos == 1) {
  let next = new_sll_node(d);
  next.next = n.next;
  n.next = some(next);
  skip
    } else {
  let some(next) = n.next in {
      insert_at_nonhead(next, d, pos - 1);
      skip
  } else {
      n.next = some(new_sll_node(d));
      skip
  }
    }
}

def push(l : sll, d : data) : unit
    consumes d {

    let n = new_sll_node(d);
    n.next = l.head;
    l.head = some(n);
    skip
}

def push_tail(l : sll_node, d : data) : unit
    consumes d {

    let some(next) = l.next in {
  push_tail(next, d)
    } else {
```

```
  l.next = some(new_sll_node(d));
  skip
    }
}

def concat(l1, l2 : sll_node) : unit
    consumes l2 {

    let some(l1_next) = l1.next in {
  concat(l1_next, l2); skip
    } else {
  l1.next = some l2; skip
    }
}

let l = new_sll();

insert_at(l, new data, 0);
insert_at(l, new data, 1);

let some(d) = remove_from(l, 0) in {
    insert_at(l, d, 2)
} else {
    insert_at(l, new data, 2)
}
```

5.2 Doubly Linked List

```
struct node {
    iso payload : data;
    next : node;
    prev : node;
}

struct dll {
    iso head : node;
}

def length (l : dll) : int {

    let n = l.head;
    let len = 0;
    while (n != l.head.prev) {
  len = (len + 1);
  n = n.next
    };
    len
}

def get_nth_node(l : dll, pos : int) : node
    after: l.head ~ result {
```

```
  let n = l.head;
  while (pos > 0) {
      n = n.next;
      pos = pos - 1
  }; n
}

def insert(l : dll, d : data, pos : int) : unit
    consumes d {

    let n = get_nth_node(l, pos);
    let n'␣=␣new␣node;
␣␣␣␣n'.payload = d;
    n'.next␣=␣n;
␣␣␣␣n'.prev = n.prev;
    n.prev.next = n';
␣␣␣␣n.prev␣=␣n';
    skip
}

def swap_out(l : dll, d : data, pos : int) : data
    consumes d {

    let n = get_nth_node(l, pos);
    let out = n.payload;
    n.payload = d;
    out
}

def EXCEPTION() : data {new data}

def split_out(l : dll, pos : int) : data {

    let n = get_nth_node(l, pos);
    let prev = n.prev;
    prev.next = n.next;
    n.next.prev = n.prev;
    n.next = n;
    n.prev = n;

    let head = l.head;
    if disconnected(n, head) {
  l.head = head;
  n.payload
    } else {
  EXCEPTION()
    }
}

let l = new dll;
```

```
insert(l, new data, 0);

let d = swap_out(l, new data, 0);

d = split_out(l, 2);

insert(l, d, 2)
```

5.3 Red-Black Tree

```
struct rbtree {
    iso root : rbtree_node?;
}

struct rbtree_node {
    iso payload : data;
    is_red : bool;
    iso left : rbtree_node?;
    iso right: rbtree_node?;
}

def new_rbtree() : rbtree {
    let r = new rbtree;
    r.root = none rbtree_node;
    r
}

def new_rbtree_node(d : data) : rbtree_node
    consumes d {

    let r = new rbtree_node;
    r.is_red = true;
    r.payload = d;
    r.left = none rbtree_node;
    r.right = none rbtree_node;
    r
}

def contains(t : rbtree, d : data) : bool {
    let some(r) = t.root in {
  contains_node(r, d)
    } else {
  false
    }
}

def contains_node(n : rbtree_node, d : data) : bool {
    if (n.payload == d) {
  true
    } else if (n.payload > d) {
  let some(left) = n.left in {
```

```
        contains_node(left, d)
    } else {
        false
    }
      } else {
    let some(right) = n.right in {
        contains_node(right, d)
    } else {
        false
    }
      }
}

def shuffle(x, y, z: rbtree_node, a, b, c, d: rbtree_node?) : rbtree_node
    before: x.left ~ x.right ~ y.left ~ y.right ~ z.left ~ z.right ~ write-only
    after: result ~ y, y.left ~ x, y.right ~ z, x.left ~ a,x.right ~ b,
          z.left ~ c, z.right ~ d {

    y.left = some x;
    y.right = some z;
    x.left = a;
    x.right = b;
    z.left = c;
    z.right = d;
    y
}

def balance(root : rbtree_node) : rbtree_node consumes root {

    let some(l) = root.left, some(ll) = l.left where (l.is_red && ll.is_red) in {
  shuffle(ll, l, root, ll.left, ll.right, l.right, root.right)
    } else let some(l) = root.left, some(lr) = l.right where (l.is_red && lr.is_red) in {
      shuffle(l, lr, root, l.left, lr.left, lr.right, root.right)
    } else let some(r) = root.right, some(rl) = r.left where (r.is_red && rl.is_red) in {
      shuffle(root, rl, r, root.left, rl.left, rl.right, r.right)
    } else let some(r) = root.right, some(rr) = r.right where (r.is_red && rr.is_red) in {
      shuffle(root, r, rr, root.left, r.left, rr.left, rr.right)
    } else {
  root
    }
}

def insert_node(t_opt : rbtree_node?, d : data) : rbtree_node
    consumes t_opt, d {

    let out : rbtree_node;

    let some(t) = t_opt in {
      if (d < t.payload) {
     t.left = some(insert_node(t.left, d));
     out = balance(t)
```

```
  } else if (d > t.payload) {
   t.right = some(insert_node(t.right, d));
   out = balance(t)
  } else {
   out = t
  }
    } else {
     out = new_rbtree_node(d)
    };
    out
}

def insert(t : rbtree, d : data) : unit
    consumes d {

    let new_root = insert_node(t.root, d);
    new_root.is_red = false;
    t.root = some(new_root);
    skip
}

let tree = new_rbtree();
insert(tree, new data);
insert(tree, new data);
contains(tree, new data)
```