Oxide: The Essence of Rust

AARON WEISS, Northeastern University, USA
OLEK GIERCZAK, Northeastern University, USA
DANIEL PATTERSON, Northeastern University, USA
AMAL AHMED, Northeastern University, USA

Rust claims to advance industrial programming by bridging the gap between *low-level* systems programming and *high-level* application programming, enabling programmers to build more reliable and efficient software. At the heart of this achievement is the *borrow checker* — a novel approach to *ownership* that aims to balance type system expressivity with usability. And yet, to date there is no type system that fully captures Rust's notion of ownership and borrowing, and hence no proper foundation for research on Rust.

We capture the essence of this model of ownership by developing a type systems account of Rust's borrow checker. We present Oxide, a formalized programming language close to *source-level* Rust (but with fully-annotated types). Oxide takes a new view of *lifetimes* as sets of locations called *regions* which approximate the origins of references. Our type system is able to automatically compute this information through a control-flow-based substructural typing judgment. In doing so, we develop a novel type system for *region-based alias management*. Significantly, Oxide is the first type system for core Rust that provides a tested semantics and leverages conventional tools for the formalization and metatheory: it is not built on top of a separation logic and is proved sound using progress and preservation. As such, it offers a self-contained model of borrow checking — including features such as *non-lexical lifetimes* — that provides a basis for future research on Rust.

1 INTRODUCTION

The Rust programming language exists at the intersection of low-level "systems" programming and high-level "applications" programming, providing both fine-grained control over memory and performance *and* high-level abstractions that make software more reliable and quicker to produce. To accomplish this, Rust integrates decades of programming languages research into a production system. Most notably, this includes ideas from linear and ownership types [Clarke et al. 1998; Girard 1987; Lafont 1988; Noble et al. 1998] and region-based memory management [Fluet et al. 2006; Grossman et al. 2002]. Yet, Rust goes beyond prior art in developing a particular typing discipline that aims to balance both *expressivity* and *usability*. As such, Rust has something interesting to teach us about making ownership *practical* for programming.

But without a formal semantics to build upon, it is difficult for researchers to learn, understand, and investigate this new discipline. This is not a new problem; the novelty of new languages has often encouraged their formal study to learn *precisely* what they offer. As Guha et al. [2010] did for JavaScript, we endeavor to do for Rust — capturing the essential pieces of Rust, namely the borrow checker, and providing a foundation for research with our new formally-defined language, Oxide.

While there are existing formalizations of Rust [Benitez 2016; Jung et al. 2019, 2018a; Pearce 2021; Reed 2015], none properly convey the essence of Rust's type system. We will discuss all of them in more detail in §4, but for now, we will focus on RustBelt [Jung et al. 2018a] which represents the most significant effort to date, and the strongest point of comparison to Oxide. RustBelt defines a calculus called λ_{Rust} and takes a semantic approach to type soundness [Ahmed 2004; Ahmed et al. 2010; Milner 1978] to verify that major parts of Rust's standard library APIs (written using `unsafe` code) do not violate its safety guarantees. Yet, λ_{Rust}'s continuation-passing style and low-level nature — closer to Rust's Mid-level Intermediate Representation (MIR) — make it difficult to use for

Authors' addresses: Aaron Weiss, Northeastern University, Boston, MA, 02115, USA, weiss@ccs.neu.edu; Olek Gierczak, Northeastern University, Boston, MA, 02115, USA, gierczak.o@northeastern.edu; Daniel Patterson, Northeastern University, Boston, MA, 02115, USA, dbp@dbpmail.net; Amal Ahmed, Northeastern University, Boston, MA, 02115, USA, amal@ccs.neu.edu.

source-level reasoning. Further, the λ_{Rust} semantics rely on a lifetime logic embedded in Iris [Jung et al. 2018b]. While this logic and embedding is useful for *verifying* the implementation of standard library APIs, the need to understand the lifetime logic and Iris poses a considerable cost to other researchers interested in, for instance, investigating new type features for Rust. Follow-on work by Jung et al. [2019] provides an operational model called *Stacked Borrows* for the comparatively untyped "raw pointers" (usable only in `unsafe`), which is largely orthogonal to our efforts as we focus predominantly on the *static* semantics of Rust.

1.1 Why do another formalism of Rust then?

RustBelt and other prior work formalize a semantics for Rust based on the notion of *lifetimes* as the centerpiece of their borrow checking analysis in some way, and indeed, in context, this was a perfectly sensible decision. After all, Rust's initial versions of borrow checking relied on lifetimes tied to lexical scope (i.e., to a first approximation, an object in memory was considered to live until the end of its lexical scope). However, the work that extended the language to *non-lexical* lifetimes fundamentally complicated reasoning about references tied to lifetimes in this way. Using continuation-passing style [Sussman and Steele 1975] as λ_{Rust} does addresses some of the added complexity of non-lexical lifetimes by providing a natural way for non-lexical lifetimes to be made contiguous. However, we believe it is necessary to model how the *source* program works and how to think about borrow checking with non-lexical lifetimes in that light. To that end, we employ a novel use of *regions* to track *aliasing* in the static semantics of the program in Oxide.

As we will see, Oxide is a higher-level language, with syntax close to that of surface Rust and a semantics that works with an *abstract* notion of memory that does not require us to make concrete memory layout decisions for each type. This is significant because it allows us to focus on the essence of how safe Rust deals with memory and aliasing, while avoiding a need to address details caught up in discussions about memory layout and validity guarantees that are ongoing in the unsafe code guidelines working group [Group 2019]. We also focus our efforts by requiring type annotations on let bindings in Oxide to avoid the orthogonal complexities of type inference, and omitting the trait system which is largely described in the literature on typeclasses. We also do not include operations for concurrency, as we believe borrow checking can be understood clearly *without* it.

1.2 Our Contributions

Our efforts to develop Oxide have led us to five main contributions: (1) We present Oxide as the first formal account close to safe, surface Rust. (2) Most significantly, we note that while Rust's borrow-checking implementation relies on constraint generation and an algorithmic constraint solver, we provide *an inductive definition of borrow checking in terms of conventional inference rules.* This definition builds on a view of lifetimes as sets of locations called *regions* approximating the provenances of references, rather than abstractions of the lines of code where the referenced memory is live. (3) This design represents a novel treatment of regions, leveraging them to manage *aliasing* rather than memory itself, that we call *region-based alias management.* (4) We provide *the first syntactic type safety* [Wright and Felleisen 1992] result for Rust, which is challenging because we must maintain the well-typedness of values on the stack. Ordinarily, this is straightforward, but since our values include suspended computations which can themselves introduce aliasing, we must show that the requirements for safe aliasing in that computation are maintained throughout the program's execution. (5) Oxide features *a tested semantics* which has been validated in its faithfulness to RUSTC borrow checking on the subset of features supported by Oxide using tests from Rust's official borrow checker and non-lexical lifetimes test suites. Thus, we posit that Oxide serves as an explainable *essence of Rust*, and a solid foundation for research on and leveraging Rust.

The rest of the paper is organized as follows: §2 describes the essence of Rust and Oxide at an intuitive level. §3 presents the formal details of Oxide including the syntax (§3.1, §3.2, and §3.4), type system (§3.5), operational semantics (§3.6), and metatheory (§3.7). §3.8 provides evidence that Oxide faithfully models Rust, via discussion of our compiler Reducer from Rust to Oxide and a type checker OxideTC used to validate that Oxide typechecking matches Rust on a subset of Rust's official test suite. We discuss related work in §4 and some higher-level points about Oxide in §5. The technical appendices include complete definitions (§A, §B, §C, §D), typing rules (§B.4), and proofs (§E). Our implementation and test suite for our tested semantics are available on GitHub.

Nothing is yours. It is to use. It is to share.
If you will not share it, you cannot use it.

The Dispossessed
Ursula K. Le Guin

2 DATA THEY CAN CALL THEIR OWN

The essence of Rust lies in its novel approach to *ownership* and *borrowing*, which account for the most interesting parts of the language's static semantics and the justification for its claims to *memory safety* and *data race freedom*. In this section, we gradually introduce Oxide by exploring example programs that illustrate key pieces of how ownership and borrowing function. At the same time, we'll explain the syntax in each example for readers unfamiliar with Rust.

2.1 Ownership as Use-Once Variables

Rust's notion of ownership rests atop a long lineage of work, harkening back to the early days of linear logic [Girard 1987], and especially efforts by Wadler [1991] and Baker [1992] to develop systems for functional programming *without* garbage collection. However, as noted by Wakeling and Runciman [1991], Wadler's effort relied greatly on pervasive copying. This reliance on copying and the associated performance penalty would not suffice for real world systems programming efforts, and thus, Rust's ownership model is best understood as instead building off of Baker's work on Linear Lisp where linearity enabled efficient reuse of objects in memory [Baker 1992, 1994a,b, 1995]. The resemblance is especially strong between Rust *without* borrowing and Baker's 'use-once' variables [Baker 1995]. We illustrate these ideas at work in Oxide with the following example:

```
1  struct Point(u32, u32);
2  let pt = Point(6, 9);
3  let x = pt;
4  let y = pt; // ERROR: pt was already moved
```

In this example, we first declare a type `Point` that consists of a pair of unsigned 32-bit integers (`u32`). Then, on line 2, we create a new `Point` from `(6, 9)` and name it `pt`. We say that this new value is *owned* by the identifier `pt`. Then, on line 3, we transfer this ownership by *moving* the value from `pt` to `x`. Moving the value out of `pt`, invalidates this old name. Subsequently, when we attempt to use it again on line 4, we encounter an error because `pt` was already moved in the previous line. If we instead used `x` instead of `pt` on line 4, we would not error as each variable is used once.

2.2 Borrowing with Loans

Rust's main departure from techniques like 'use-once' variables [Baker 1995] is a softening of a rather stringent requirement: namely, that *everything* must be managed uniquely. Instead, Rust allows the programmer to locally make a decision to use unique references [Minsky 1996] with unguarded mutation *or* to use shared references without such mutation.[1] This flexibility in choosing arises at the point where the programmer creates a new reference, and draws inspiration from work on ownership types and flexible alias protection [Clarke et al. 1998; Noble et al. 1998]. We again illustrate its use in Oxide with an example:

```
1  letrgn<'x, 'y> {
2    let pt = Point(6, 9);
3    let x = &'x shrd pt;
4    let y = &'y shrd pt; // OK: sharing is fine!
5  }
```

In the above example, we replaced the *move* expressions in each binding with *borrow* expressions (written using &) that each create a shared reference to pt. Here, we also see our first syntactic differences from Rust. Namely, in Oxide, borrow expressions include an annotation for their *region* (roughly an analogue of Rust's *lifetimes*) which are bound earlier using letrgn on line 2. As noted in the comment, this program no longer produces an error because the references allow precisely this kind of sharing, but one should note that this sharing would be *disallowed* by a standard linear or affine type system. As a consequence of allowing this sharing, the type systems of both Oxide and Rust prevent mutation through these references. To mutate through a reference, you must instead have a unique reference (i.e. it is the only usable name for the underlying data). Our next example replaces our shared references with unique ones:

```
1  letrgn<'x, 'y> {
2    let pt = Point(6, 9);
3    let x = &'x uniq pt;
4    let y = &'y uniq pt; // ERROR: cannot borrow pt uniquely twice
5    ... // additional code that uses x
6  }
```

We've now chosen to create unique, rather than shared, references to pt. However, since our program attempts to do so twice, we encounter an error similar to the one we had in our first example when we tried to move pt twice. The astute reader might notice that another important change happened — we added some additional code afterward that somehow makes use of x. This is important because of a feature in Rust known as *non-lexical lifetimes* (or NLL for short) [Matsakis 2016; Turon et al. 2017]. With non-lexical lifetimes and no uses of x in the ensuing code, the compiler would figure out that this apparent violation of the uniqueness of unique references would be not be realized since x is never used, and thus may as well not exist. As such, the program would be accepted by the borrow checker. With the additional use of x, the violation *is* realized and so the program is rejected.

Similar to the last example, the borrow checker also prevents us from mixing unique references with shared ones, as in the following example:

```
1  letrgn<'x, 'y> {
2    let pt = Point(6, 9);
3    let x = &'x uniq pt;
4    let y = &'y shrd pt; // ERROR: cannot borrow pt while
5                         // a unique loan is live
6      ... // additional code that uses x
7    }
```

In this case, we've changed the borrow expression on line 4 to create a shared, rather than unique, reference. This again produces an error because Rust forbids the creation of a shared reference

[1]The use of "such" here is intentional as dynamically guarded mutation, e.g. using a Mutex, is still allowed through a shared reference. Indeed, this is precisely what makes such guards *useful* when programming.

while a unique *loan* exists. Here, we use the word loan to refer to the state introduced in the borrow checker (which records the loan's uniqueness and its origin) by the creation of a reference. Regions[2] in Oxide (denoted `'a`, `'b`, etc.) can be understood as collections of these loans which together form a static approximation of the origins of references annotated with that region. In this way, we can think of our regions as a sort of *static* grouping of distinct objects in memory, and by associating a reference with a region, we identify that the objects in memory that reference could point to must be from the collection of loans we've associated with it. The borrow checker leverages this information about the origins of references to determine whether or not a reference is safe to create or use at a given program point.

We use the term *region* here because the term carries a sort of grouping association, and past uses of region in the literature also use the term to represent a grouping of objects in memory. However, it's important to note the difference in how they are used! In the literature, regions are used to manage memory [Fluet et al. 2006; Grossman et al. 2002; Tofte and Talpin 1994, 1997] with each region representing a contiguous chunk of memory in which references are managed. By contrast, Oxide's regions correspond to an abstract and purely static grouping of objects in memory, and doesn't have any influence over where allocation happens. Instead, their use by the borrow checker during typechecking rules out bad aliasing patterns (as we have seen so far), leading us to refer to this approach to regions as *region-based alias management*.

In Oxide, we write these loans as a pair of a place and an ownership qualifier (`uniq` or `shrd`), e.g. $^{\mathtt{uniq}}\mathtt{pt}$ would be the loan corresponding to the borrow on line 4. During typechecking, we associate each of the regions bound with `letrgn` (e.g. `'x` and `'y`) with sets of these loans. Specifically, after line 4, `'x` will map to the loan set $\{\ ^{\mathtt{uniq}}\mathtt{pt}\ \}$ and, after line 5, `'y` will map to the loan set $\{\ ^{\mathtt{shrd}}\mathtt{pt}\ \}$. Although these examples only have single-element loan sets, more complex programs using branching will merge loan sets making them approximate. When typechecking a borrow expression, Oxide looks at these loan sets in the environment to determine whether or not the borrow should be permitted.

While we were unable to create a second reference to the same place as an existing unique reference in our past examples, Oxide and Rust both allow the programmer to create two unique references to disjoint paths within the same object, as in the following example:

```
letrgn<'x, 'y> {
  let mut pt = Point(6, 9);
  let x = &'x uniq pt.0;
  let y = &'y uniq pt.1;
  // no error, our loans don't overlap!
  ...
}
```

In this example, we're borrowing from specific paths within `pt` (namely, the first and second projections respectively). Since these paths give a name to the places being referenced, we refer to them as *places*. Here, we see that our notion of ownership is fine-grained, allowing unique loans against non-overlapping places within *aggregate structures* (like structs, enumerations, and tuples). Intuitively, this is safe because the parts of memory referred to by each individual place (in this case, `pt.0` and `pt.1`) do not overlap, and thus they represent portions that can each be uniquely owned and borrowed.

Rust supports an additional pattern that weakens conventional notions of flexible alias protection. In particular, it allows the programmer to create a unique reference by borrowing from one they

[2]Historically, Rust has used the term *lifetime*, rather than region. Recent efforts on a borrow checker rewrite called Polonius have transitioned to using the term origin [Matsakis 2018; Matsakis and Contributors 2020] for a similar concept to our regions. We discuss Polonius further in §5.

Variables	x	Functions	f	Type Vars.	α
Frame Vars.	φ	Concrete Regions	r	Abstract Regions	ϱ

$$\begin{array}{llcl}
\text{Path} & q & ::= & \epsilon \mid n.q \\
\text{Places} & \pi & ::= & x.q \\
\text{Place Exprs.} & p & ::= & x \mid *p \mid p.n \\
\text{Place Expr. Contexts} & p^{\square} & ::= & \square \mid *p^{\square} \mid p^{\square}.n \\
 & & & \\
\text{Regions} & \rho & ::= & \varrho \mid r \\
\text{Ownership Qualifiers} & \omega & ::= & \texttt{shrd} \mid \texttt{uniq}
\end{array}$$

$$\begin{array}{llcl}
\text{Constants} & c & ::= & (\,) \mid n \mid \texttt{true} \mid \texttt{false} \\
\text{Expressions} & e & ::= & c \mid p \mid \&r\ \omega\ p \mid \&r\ \omega\ p[e] \mid \&r\ \omega\ p[e_1..e_2] \\
 & & \mid & e_1;\ e_2 \mid p \coloneqq e \mid (e_1\,,\ \ldots\,,\ e_n) \mid [e_1\,,\ \ldots\,,\ e_n] \\
 & & \mid & \texttt{letrgn}\,{<}r{>}\ \{\ e\ \} \mid \texttt{let}\ x : \tau^{\text{SI}} = e_1;\ e_2 \\
 & & \mid & |x_1 : \tau_1^{\text{SI}}\,,\ \ldots\,,\ x_n : \tau_n^{\text{SI}}| \ \longrightarrow\ \tau_r^{\text{SI}}\ \{\ e\ \} \\
 & & \mid & e_f{::}{<}\overline{\Phi},\ \overline{\rho},\ \overline{\tau^{\text{SI}}}{>}(e_1\,,\ \ldots\,,\ e_n) \\
 & & \mid & \texttt{if}\ e_1\ \{\ e_2\ \}\ \texttt{else}\ \{\ e_3\ \} \mid p[e] \mid \texttt{abort!(str)} \\
 & & \mid & \texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \} \mid \texttt{while}\ e_1\ \{\ e_2\ \} \\
 & & \mid & \texttt{Left::}{<}\tau_1^{\text{SI}}\,,\ \tau_2^{\text{SI}}{>}(e) \mid \texttt{Right::}{<}\tau_1^{\text{SI}}\,,\ \tau_2^{\text{SI}}{>}(e) \\
 & & \mid & \texttt{match}\ e\ \ \{\ \texttt{Left}(x_1) \Rightarrow e_1\,,\ \texttt{Right}(x_2) \Rightarrow e_2\ \}
\end{array}$$

Fig. 1. Term Syntax of Oxide

already have. However, the program is unable to use the old reference until the *reborrowed* one is destroyed. We produce the same behavior in Oxide, and we can see it at work in the following example:

```
letrgn<'x, 'y> {
  let mut pt = Point(6, 9);
  let x = &'x uniq pt.0;
  let y = &'y uniq *x;
  // we can use y, cannot use x until we drop y
  ...
}
```

In this example, we borrow the first projection of `pt` (`pt.0`) and then reborrow it by creating a borrow to `*x`. We then can use `y` in the continuation, but won't be able to use `x` until `y` is dropped. This particular pattern of *reborrowing* is perhaps one of the most unique things about Rust's design.

The combination of features discussed above, namely moves, shared and unique borrows, the ability to create unique references to disjoint paths, reborrowing, and non-lexical lifetimes, makes borrow checking extremely subtle, even when we focus on just the safe subset of Rust. In the rest of the paper, we present Oxide and discuss how our formalism deals with this mix of features.

3 OXIDE, FORMALLY

In this section, we present Oxide's formal semantics. We first discuss the terms in the language (§3.1), then the types (§3.2) and *regions* (§3.4), and our environments and the mechanics of typechecking (§3.5). Finally, we move on to discussion of our metatheory (§3.7) and tested semantics (§3.8).

3.1 The Syntax of Oxide

Figure 1 presents the syntax of Oxide terms, in four broad groupings: (1) metavariables for the various kinds of names that exist, (2) places, which act as names for abstract memory locations, (3) annotations for references, and (4) the actual terms of the language. The first group is fairly conventional, but we'll discuss special names like frame variables as they come up.

Places and Place Expressions. As we saw in §2.2, places π and place expressions p are names for paths from a particular variable to a particular part of the object stored there, whether that be a projection of a tuple, or a field of a struct (where a struct is really precisely just a named tuple or record type). One might think of place expressions as a sort of syntactic generalization of variables. They are analogous to what are called `lvalues` in C. Places π are a subset of place expressions that do not include dereferences. They can intuitively be thought of as an abstract name of a memory location since when bound, they will always correspond to one particular value on the stack. Place expression contexts $p^\square$ are used in various parts of the formalism to decompose place expressions p into an innermost dereferenced place, $*\pi$, and an outer context $p^\square$.

Annotations for References. In Oxide, we have two annotations that we provide for every borrowing expression. First, we annotate references with ownership qualifiers ω, indicating whether the reference is shared (`shrd`) or unique (`uniq`). We use these rather than their equivalents in Rust (no annotation and **`mut`** respectively) because the terms more accurately reflect the semantic focus on *aliasing*, rather than *mutation*. Indeed, in Rust, a value of the type `&&mut u32` *cannot* be mutated (because we have a shared reference to a unique reference), and a value of the type `&Cell<u32>`[3] *can* be mutated through the method `Cell::set`.

Second, we annotate references with *regions*. Regions ρ have two forms: abstract regions ϱ (pronounced *var-rho*) and concrete regions r. Abstract regions correspond to lifetime variables `'a`, `'b`, etc. in Rust, and are used polymorphically in function types to indicate that the function is agnostic to the particular regions of reference-type parameters. Concrete regions, by contrast, carry concrete information in the environment where they correspond to a set of loans. A loan $^{\omega}p$ indicates a possible origin (p), qualified by whether the loan is unique or shared (ω). Intuitively, each loan tells us a single possible origin for a reference, while a concrete region maps to *all* possible origins of a reference. As we will see in §3.4, regions are essential to enabling our type system to guarantee the correct use of unique and shared references.

Expressions. Expressions e in Oxide are numerous, but largely standard. For example, constants c consist of the unit value $()$, unsigned 32-bit integers n, and boolean values `true` and `false`. The most interesting expressions in Oxide are the ones we've already seen by example: place expression usage (written simply p) and borrowing (with several forms that we explain shortly). The former may be thought of as variables that behave linearly for non-copyable data (removing the place from the environment after use), and traditionally for copyable data. (As a first approximation, one can think of all data that is not a unique pointer as safely copyable.)

There are three borrowing forms, and all work in fundamentally the same way: they are each used as introduction forms for references. The simplest case, written $\&r\ \omega\ p$, introduces an ω-reference (with region r) to the location that the place expression p evaluates to. The next form borrows from $p[e]$ instead of simply p, and is used to borrow an element out of an array or slice p at the index given by e. The final form borrows from $p[e_1..e_2]$, and is used to borrow a slice of p using the range given by e_1 and e_2. A *slice* is Rust terminology for a dynamically-sized subsection of an array.

In these last two cases, one might wonder "why are indexing and slicing not places themselves?" The answer comes in two parts: (1) indexing and slicing take arbitrary expressions, while places are entirely static, and (2) unlike tuple projections which have a fine-grained notion of ownership, indexing and slicing affect the ownership of the array or slice overall. This second part means that while you can create two unique references to different projections of the same tuple, you cannot create two unique references to different indices of an array.

The remainder of our expressions are standard or discussed already. These include sequencing, assignment, and creation of tuples and arrays. Our closure syntax follows the syntax of Rust, and thus uses vertical bars to denote the closure's parameters. As in Rust, closures are not polymorphic; only global functions (shown in Figure 3) may be polymorphic and specify where-bounds on regions.[4] We use function application when applying closures as well as global functions. Hence,

[3]`Cell<T>` is a Rust standard library type that provides a "mutable memory location" that allows mutation in its API.

[4]In Rust, where-bounds in functions are used to constrain one lifetime to outlive another, meaning that a reference with the larger lifetime must be valid at least as long as a reference with the shorter lifetime.

$$\begin{array}{llll}
\text{Kinds} & \kappa & ::= & \star \mid \mathtt{RGN} \mid \mathtt{FRM} \\
\text{Base Types} & \tau^{\mathrm{B}} & ::= & \mathtt{bool} \mid \mathtt{u32} \mid \mathtt{unit}
\end{array}$$

$$\begin{array}{llll}
\text{Sized Types} & \tau^{\mathrm{SI}} & ::= & \tau^{\mathrm{B}} \mid \alpha \mid \&\rho\ \omega\ \tau^{\mathrm{XI}} \\
 & & \mid & (\tau_1^{\mathrm{SI}}, \ldots, \tau_n^{\mathrm{SI}}) \mid [\tau^{\mathrm{SI}}; n] \mid \mathtt{Either}{<}\tau_1^{\mathrm{SI}}, \tau_2^{\mathrm{SI}}{>} \\
 & & \mid & \forall{<}\overline{\varphi}, \overline{\varrho}, \overline{\alpha}{>}(\tau_1^{\mathrm{SI}}, \ldots, \tau_n^{\mathrm{SI}}) \overset{\Phi}{\to} \tau_r^{\mathrm{SI}}\ \mathtt{where}\ \overline{\varrho_1 : \varrho_2}
\end{array}$$

$$\begin{array}{llll}
\text{Maybe Unsized Types} & \tau^{\mathrm{XI}} & ::= & \tau^{\mathrm{SI}} \mid [\tau^{\mathrm{SI}}] \\
\text{Dead Types} & \tau^{\mathrm{SD}} & ::= & {\tau^{\mathrm{SI}}}^{\dagger} \mid (\tau_1^{\mathrm{SD}}, \ldots, \tau_n^{\mathrm{SD}}) \\
\text{Maybe Dead Types} & \tau^{\mathrm{SX}} & ::= & \tau^{\mathrm{SI}} \mid \tau^{\mathrm{SD}} \mid (\tau_1^{\mathrm{SX}}, \ldots, \tau_n^{\mathrm{SX}}) \\
\text{Types} & \tau & ::= & \tau^{\mathrm{XI}} \mid \tau^{\mathrm{SX}}
\end{array}$$

Fig. 2. Type Syntax of Oxide

function application additionally includes polymorphic instantiation written using Rust's turbofish syntax (`::<>`). An `abort!(str)` indicates irrecoverable failure; it terminates the program with the given string as a diagnostic message. Finally, Oxide includes tagged sums, which are introduced using the `Left` and `Right` forms and eliminated using `match`.[5]

3.2 Types in Oxide

In Oxide, we have five distinct categories of types (based on two features we need to distinguish: sized vs. unsized, and initialized vs. dead), and a kind system to track the three kinds of polymorphism in the language. While these distinctions may seem complex, they greatly simplify the well-formedness conditions required on types during typechecking. The grammars are all present in Figure 2, and are explained in detail in the rest of the section.

Sized and Unsized Types. We need to distinguish between types based on sizedness, which is a direct consequence of Rust itself. All bindings in Rust (and in Oxide) must be able to fit on the stack which requires that they have a statically-known size. In Rust, this is dealt with using a special automatically-derived *marker trait* called `Sized` which serves as a tag during typechecking to indicate that a type has a statically-knowable size. For pragmatic reasons (since one typically works with sized types), Rust decided on using `Sized?` to indicate that a type is "possibly unsized" (and thus could only be part of a type for a let binding if it is behind a reference). In Oxide, we have a comparable syntactic distinction between *sized* types τ^{SI} and *maybe unsized* types τ^{XI}. Sized types characterize all the types with statically-known sizes and maybe unsized types τ^{XI} include all such types *and* the slice type $[\tau^{\mathrm{SI}}]$ which corresponds to a dynamically-sized portion of an array.

Initialized and Dead Types. We also need to distinguish between types based on initialization, which we use to model the 'use-once' linearity of variables referring to non-copyable data. To that end, we introduce two categories. First, *dead* types τ^{SD} which is either a sized and initialized type with a dagger on it (indicating that it is dead) or a product of dead types. These correspond to totally moved types. Second, we have *maybe dead* types τ^{SX} which can be either initialized, dead, or a product of maybe dead types, corresponding to types where some of their components have been moved. Though not supported directly in our formalism, these dead and maybe dead types also can be used directly to support uninitialized and partially-initialized variable bindings.

Kinds and Polymorphism. Oxide has three kinds κ: the kind of ordinary types $\star$, the kind of regions `RGN`, and the kind of *frame typings* `FRM`. (Frame typings are relevant for closures, as we'll see below.) We abstract over variables of each kind in Oxide and, to aid the reader, we have separate syntax for each: α, ϱ, and φ, respectively. For simplicity, Oxide restricts type variables α to being

[5]Rust, of course, supports more general n-ary tagged sums with user-definable tags (calling the whole system enumerations), but binary sums suffice to get at the essence of Rust without requiring a complicated formalization for pattern matching

$$\begin{array}{llcl}
\text{Global Environment} & \Sigma & ::= & \bullet \mid \Sigma,\ \varepsilon \\
\text{Global Entries} & \varepsilon & ::= & \texttt{fn}\, f\texttt{<}\overline{\varphi},\ \overline{\varrho},\ \overline{\alpha}\texttt{>}(x_1 : \tau_1^{\text{SI}},\ \ldots,\ x_n : \tau_n^{\text{SI}}) \rightarrow \tau_r^{\text{SI}}\ \texttt{where}\ \overline{\varrho_1 : \varrho_2}\ \{\ e\ \}
\end{array}$$

$$\begin{array}{llcl}
\text{Type Environment} & \Delta & ::= & \bullet \mid \Delta,\ \alpha : \star\ \mid \Delta,\ \varrho : \mathsf{RGN} \mid \Delta,\ \varphi : \mathsf{FRM} \mid \Delta,\ \varrho :> \varrho' \\
\text{Continuation Typing} & \Theta & ::= & \bullet \mid \Theta,\ \tau^{\text{SI}} \\
\text{Stack Typing} & \Gamma & ::= & \bullet \mid \Gamma \natural \mathcal{F} \\
\text{Frame Typing} & \mathcal{F} & ::= & \bullet \mid \mathcal{F},\ x\ :\ \tau^{\text{SX}} \mid \mathcal{F},\ r \mapsto \{\ \overline{\ell}\ \} \\
\text{Frame Expressions} & \Phi & ::= & \varphi \mid \mathcal{F}
\end{array}$$

Fig. 3. Environments in Oxide

instantiated only with sized and initialized types, but this limitation could be addressed by enriching kinds further with a unique kind for each sort of type.

The Types Themselves. The majority of types in Oxide are sized & initialized types, including base types τ^{B}, type variables α, tuples $(\tau_1^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}})$, arrays of length n $[\tau^{\text{SI}};\ n]$, binary sums `Either<`τ_1^{SI}`, `τ_2^{SI}`>`, references $\&\rho\ \omega\ \tau^{\text{XI}}$, and function types. With the exception of references, any types that occur within these types are themselves required to be both sized and initialized. For reference types $\&\rho\ \omega\ \tau^{\text{XI}}$, we include both the region ρ and ownership qualifier ω in the type which allow us to understand statically both a reference's origin as well as its aliasing requirements. We allow potentially unsized types under references since the reference itself will always have a fixed size regardless of what it points to (e.g. 64-bit on a 64-bit machine).

Function types have three notable features. First, each function type can possibly include a frame expression Φ (syntax in Figure 3) over the arrow indicating what bindings, if any, were caught up in the closure environment (when nothing is captured, we put nothing over the arrow). Next, functions are polymorphic in type and region variables, as well as in frame variables φ to enable the use of higher-order functions. Finally, functions can relate types with abstract regions using outlives bounds: `where` $\varrho_1 : \varrho_2$ means ϱ_1 outlives ϱ_2. These `where` bounds come directly from Rust, and are useful in making functions that, e.g., reborrow from one of several reference-typed parameters.

3.3 Environments for Typechecking

With the syntax of terms and types in hand, we can look more closely at some example Oxide programs to understand the environments we'll be using for typechecking. We'll start with a simple example using reborrowing, much like our last example in §2.

```
1  struct Obj(u32);
2  letrgn<'y, 'z> { // 'y -> {}, 'z -> {}
3    let mut x = Obj(5); // x : Obj
4    let y = &'y uniq x /* 'y -> { uniq x } */;
5    let z = &'z uniq *y /* 'z -> { uniq x , uniq *y } */;
6  }
```

Here, we create an object named `x` and in the comment, we see how our stack typing (written Γ) will record the new binding and its type. Then, on line 4, we produce a unique reference to `x`. In the comment, we see the metadata produced by this borrow associating the region `'y` with the set of loans $\{\ {}^{\texttt{uniq}}x\ \}$. This metadata means that a reference with the region `'y` must necessarily point to `x`. Note that this metadata is produced immediately after the borrow expression, and doesn't depend on the binding `y` being introduced for it. Then, on line 5, we reborrow from `y` as `z`, and again, we can see the corresponding metadata produced associating the region `'z` with the set of loans $\{\ {}^{\texttt{uniq}}x\ ,\ {}^{\texttt{uniq}}{*y}\ \}$. Note that borrowing $*y$ here means reborrowing from `y`. This tells us two things: (1) that a reference with the region `'z` points to `x`, and (2) that a reference with the region `'z` was created by reborrowing from `y`. That latter means that `y` ought to be rendered unusable

$\boxed{\Delta;\ \Gamma;\ \Theta \vdash^{\overline{\pi}}_{\omega} p \Rightarrow \{\ \overline{{}^{\omega}p'}\ \}}$ where $\Delta;\ \Gamma;\ \Theta \vdash_{\omega} p \Rightarrow \{\ \overline{{}^{\omega}p'}\ \}$ means $\Delta;\ \Gamma;\ \Theta \vdash^{\bullet}_{\omega} p \Rightarrow \{\ \overline{{}^{\omega}p'}\ \}$.

O-SafePlace

$$\frac{\begin{array}{c}\forall r' \mapsto \{\ \bar{\ell}\ \} \in \text{regions}(\Gamma,\ \Theta).\ (\forall\, {}^{\omega'}p^{\square}[\pi']\ \in \{\ \bar{\ell}\ \}.(\omega = \mathsf{uniq} \vee \omega' = \mathsf{uniq}) \implies \pi' \# \pi) \\ \vee\ (\exists \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \wedge \nexists \&r'\ \omega'\ \tau' \in \Theta\ \wedge (\forall \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\ \overline{\pi_e}\ \}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash^{\overline{\pi_e}}_{\omega} \pi \Rightarrow \{\ {}^{\omega}\pi\ \}}$$

O-Deref

$$\frac{\begin{array}{c}\Gamma(\pi) = \&r\ \omega_{\pi}\ \tau_{\pi} \qquad \Gamma(r) = \{\ \overline{{}^{\omega'}p}^{\,n}\ \} \qquad \text{excl} = \{\ \pi_j \text{ where } j \in \{\ 1,\ \ldots\ n\ \} \mid p_j = p^{\square}_j[*\pi_j]\ \} \qquad \omega \lesssim \omega_{\pi} \\ \forall i \in \{\ 1\ \ldots\ n\ \}.\ \Delta;\ \Gamma;\ \Theta \vdash^{\overline{\pi_e},\,\text{excl},\,\pi}_{\omega} p^{\square}[p_i] \Rightarrow \{\ \overline{{}^{\omega}p'_i}\ \} \\ \forall r' \mapsto \{\ \bar{\ell}\ \} \in \text{regions}(\Gamma,\ \Theta).\ (\forall\, {}^{\omega'}p''\ \in \{\ \bar{\ell}\ \}.(\omega = \mathsf{uniq} \vee \omega' = \mathsf{uniq}) \implies p'' \# p^{\square}[*\pi]) \\ \vee\ (\exists \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \wedge \nexists \&r'\ \omega'\ \tau' \in \Theta\ \wedge (\forall \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\ \overline{\pi_e},\ \text{excl},\ \pi\ \}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash^{\overline{\pi_e}}_{\omega} p^{\square}[*\pi] \Rightarrow \{\ \overline{{}^{\omega}p'_1},\ \ldots\ \overline{{}^{\omega}p'_n},\ {}^{\omega}p^{\square}[*\pi]\ \}}$$

O-DerefAbs

$$\frac{\begin{array}{c}\Gamma(\pi) = \&\varrho\ \omega_{\pi}\ \tau_{\pi} \qquad \Delta;\ \Gamma \vdash_{\omega} p^{\square}[*\pi] : \tau \qquad \omega \lesssim \omega_{\pi} \\ \forall r' \mapsto \{\ \bar{\ell}\ \} \in \text{regions}(\Gamma,\ \Theta).\ (\forall\, {}^{\omega'}p\ \in \{\ \bar{\ell}\ \}.(\omega = \mathsf{uniq} \vee \omega' = \mathsf{uniq}) \implies p \# p^{\square}[*\pi]) \\ \vee\ (\exists \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \wedge \nexists \&r'\ \omega'\ \tau' \in \Theta\ \wedge (\forall \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\ \overline{\pi_e},\ \pi\ \}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash^{\overline{\pi_e}}_{\omega} p^{\square}[*\pi] \Rightarrow \{\ {}^{\omega}p^{\square}[*\pi]\ \}}$$

Fig. 4. Ownership Safety in Oxide

as long as our reference z (with region 'z) exists. The two pieces of information we've seen here, in-scope bindings with their types and borrowing metadata, are precisely what's necessary for us to track in our stack typing Γ, and each entry follows the syntax seen here. In this next example, we'll see a bit more complexity by defining and using a closure.

```
1  let x = Obj(5); // x : Obj
2  let y = Obj(9); // y : Obj
3  letrgn<'z> { // z -> {}
4    let f = |obj: &'z uniq Obj| -> Obj { Obj((*obj).0 + y.0) }; // y : Obj†
5    let z = &'z uniq x /* 'z -> { uniqx } */;
6    f(z) // z : (&'z uniq Obj)†
7  }
```

We create two objects named x and y respectively. Then, on lines 4-6, we define a closure named f that moves y from the context. This movement is described by the annotation on line 6 that shows the new entry for y in our stack typing Γ. This new entry differs in that the type associated with y is marked with a dagger indicating that it is now dead. On line 7, we create a reference z that we then pass on line 8 as an argument for f. On line 7, we also see the annotation for the region indicating that a reference with region 'z must point to x, much like we saw in the previous example. In the comment on line 8, we see that the use of z *moved* it into the function call as well, thus we mark its whole type with the dagger.

One question that might arise here is "given we marked it dead, how are we still allowed to use y *inside* the closure?" This is an important point that leads to a key aspect of the structure of our stack typing Γ. The low-level nature of Oxide means we need to actually statically record the frame-based structure of the stack at runtime. So, when we construct this closure, the moved objects (in this case, solely y) are recorded in a new frame $\mathcal{F}$ that is tracked in the type of the closure, $\forall$<>(&'z uniq Obj) $\xrightarrow{\mathcal{F}}$ Obj. This frame provides the type information necessary to typecheck the body in the future, and the closure at runtime has a corresponding stack frame as part of its value form. When we typecheck the body of the closure, they are appended to the current stack typing with ♮. So, our stack typing is really a collection of frames $\mathcal{F}$ separated by ♮

and our frames $\mathcal{F}$ contain both in-scope bindings with their types and in-scope regions with their associated loan sets. As the language of stacks and frames may imply, both stack typings Γ and frames $\mathcal{F}$ are *ordered* which makes it easier to state invariants like outlives (which we will see in §3.5) and resolves many of the typical issues that arise formally with variable binding.

This is all formally-defined in Figure 3 which features a grammar for all of the environments used in the static semantics. As suggested by the grammars, there are two other environments that are non-conventional. First, we have a global environment Σ which consists of a series of top-level function definitions. We define the well-formedness of these function definitions by saying that their bodies must be well-typed assuming that all other functions (including itself) are well-typed at their annotated type, which enables a simple treatment of mutually recursive function definitions. Second, we have a temporary typing Θ which consists of a sequence of types for parts of objects that will be in the continuation of the term being typechecked. The need for this is subtle, but we will explain it using the following example:

```
letrgn<'a, 'b> {
  let x = Obj(5);
  let y = Obj(9);
  let tup = (&'a uniq x, (); &'b uniq x);
}
```

In this example, we create two objects `x` and `y` and then attempt to create a pair named `tup` that consists of two unique references. Exactly as written, we first uniquely borrow `x` with region `'a` and then in the second component, we sequence a unit value with a unique borrow of `x` with region `'b`. Of course, the very idea of having a product of unique references to the same data sounds like a contradiction and so we would hope to reject this program! However, to capture the expressivity of Rust's borrow-checking in Oxide, we also have to clear loan sets associated with unused regions at sequencing points in programs. This leads to a dilemma: by reading the program, we know that the first reference with region `'a` is still used when we go to define the second one, but the naive definition of use would correspond to "is there currently a reference with that region in the stack typing?" The continuation typing comes in to resolve this. We don't have a name (yet) for the already-typechecked portions of a product, but we can record their types in the continuation typing to record that they are around since they may then be let-bound and used further in the program. We'll see formally how this interaction happens during typechecking in the T-Tuple rule in §3.5.

Now, we can look at the shape of our typing judgment, which we will return to define in §3.5. The shape of our typing judgement is $\Sigma; \Delta; \Gamma; \Theta \vdash e : \tau \Rightarrow \Gamma'$. This is read: with global environment Σ, type environment Δ, temporary typing Θ, and stack typing Γ, e is a well-typed expression of type τ with an updated stack typing Γ' for use in typechecking the continuation of e.

3.4 Region-Based Alias Management

As discussed in §3.1, borrow expressions in Oxide all have *region* annotations which are used to associate references statically with information about their possible referents. This information is essential to our formulation of borrow checking since we leverage it to determine if new references would be safe to create. This is done formally with a judgement called *ownership safety* which has the form $\Delta; \Gamma; \Theta \vdash_\omega p \Rightarrow \{ \overline{{}^{\omega}p'} \}$. We can read it as saying "in the environments Δ and Γ, it is safe to use the place expression p ω-ly and producing all of the loans in $\{ \overline{{}^{\omega}p'} \}$ (called the borrow chain)." where ω is either `uniq` or `shrd`. That is, if we have a derivation where ω is `uniq`, we know that we can use the place expression p uniquely because we have a proof that there are no live loans against the section(s) of memory that p represents. Further, when we have a derivation where ω is `shrd`, we know that we can use the place expression p sharedly because we have a proof that there are no live *unique* loans against the section(s) of memory that p represents. The produced borrow chain is used to create the borrowing metadata to store in the environment when the ownership safety check is guarding a borrow expression, and in the simplest case, is just precisely p.

Since it is precisely this ownership safety judgment that captures the essence of Rust's ownership semantics, we understand Rust's borrow checking system as ultimately being a system for statically building a proof that data in memory is either *uniquely owned* (and thus able to allow unguarded mutation) or *collectively shared*, but not both. To do so, intuitively, ownership safety looks at all of the concrete regions in Γ, and ensures that all the loans they map to are not in conflict with the place expression p we are attempting to use. For a `uniq` borrow, a conflict occurs if *any* loan maps to an overlapping place, but for a `shrd` borrow, a conflict occurs only when a `uniq` loan maps to an overlapping place. Since places are abstract memory locations, we can consider two places as overlapping if they are equal or one is a prefix of the other (meaning that the the longer place corresponds to a piece of the larger object in memory that the shorter place refers to).

Unfortunately, reborrowing complicates matters. To support reborrowing, ownership safety uses an expanded inner form written $\Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi}} p \Rightarrow \{\ \overline{{}^{\omega}p'}\ \}$, which says that p is ω-safe under Δ and Γ, with *reborrow exclusion list* $\overline{\pi}$, and produces the loans $\{\ \overline{{}^{\omega}p'}\ \}$. Intuitively, we use this reborrow exclusion list $\overline{\pi}$ to rule out precisely the loan conflicts that arise from reborrowing as a programming pattern — namely, a reference conficting with loans from a reference it was reborrowed from. Reborrowing is also what causes the borrow chain to contain more than just p for two reasons: (1) we may be reborrowing from a reference whose region has already lost precision, i.e. contains multiple loans, and thus we cannot have perfect information about the reborrowed reference either, and (2) the reborrowed borrow chain will include an additional loan that records that it was reborrowed (e.g. a unique loan for a reborrow from `x` would appear as ${}^{\texttt{uniq}}\ {*x}$). This second point is precisely what produces the *chain* aspect of the borrow chain since we can use this information in each loan set to follow a series of consecutive reborrows.

Formally, the first rule, O-SafePlace, checks if a place π is ω-safe by looking at each loan in every region r' in Γ and either (1) making sure that if either that loan or ω is `uniq` then π does not overlap with the loan; or (2) checking that all references in Γ with region r' are in the reborrow exclusion list (meaning we need not check if there is overlap with π).

The next two rules check if a place expression p is ω-safe, decomposing the place expression into a place expression context $p^{\square}$ (see Figure 1) with $*\pi$ in the hole. The last two lines of premises for both essentially ensure that either (1) or (2) holds, but each one adds to the incoming reborrow exclusion list when checking (2) by also including the place itself π along with all of the places π_j that π was borrowed from according to the loan set r associated with its type. Both rules also check $\omega \lesssim \omega_{\pi}$ (defined as the reflexive closure of `shrd` $\lesssim$ `uniq`) in order to ensure that the reference has sufficient permission to be used, preventing a dereference of a `uniq` reference in a `shrd` context.

Unlike O-DerefAbs, O-Deref is dereferencing a reference π with a concrete region r. As such, we can look at the loans present for r in the stack typing. These loans consist of both direct loans to places π_j which correspond to a possible origin for the reference, and indirect loans to place expressions p_j which capture how this reference was reborrowed from other references. As such, when we recursively check for the safety of these regions, we append the reborrow origins (the π_j prefixes of these p_j) to the reborrow exclusion list. This means that they will not be considered as possible conflicts in the rest of ownership safety. At the end, we union together the borrow chains from all the possible origins to determine our final borrow chain. We also include an additional loan ${}^{\omega}p^{\square}[*\pi]$ to indicate that this use was reborrowed from $*\pi$.

$$\boxed{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash e : \tau \Rightarrow \Gamma'}$$

T-u32

$$\frac{}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{n} : \texttt{u32} \Rightarrow \Gamma}$$

T-Move

$$\frac{\begin{gathered}\Delta;\ \Gamma;\ \Theta \vdash_{\texttt{uniq}} \pi \Rightarrow \{\ {}^{\texttt{uniq}}\pi\ \}\\ \Gamma(\pi) = \tau^{\text{SI}} \qquad \texttt{noncopyable}_{\Sigma}\ \tau^{\text{SI}}\end{gathered}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\pi} : \tau^{\text{SI}} \Rightarrow \Gamma[\pi \mapsto \tau^{\text{SI}^{\dagger}}]}$$

T-Copy

$$\frac{\begin{gathered}\Delta;\ \Gamma;\ \Theta \vdash_{\texttt{shrd}} p \Rightarrow \{\ \overline{\ell}\ \}\\ \Delta;\ \Gamma \vdash_{\texttt{shrd}} p : \tau^{\text{SI}} \qquad \texttt{copyable}_{\Sigma}\ \tau^{\text{SI}}\end{gathered}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{p} : \tau^{\text{SI}} \Rightarrow \Gamma}$$

T-Borrow

$$\frac{\begin{gathered}\Gamma(r) = \emptyset \qquad \Gamma;\ \Theta \vdash r\ \texttt{rnic}\\ \Delta;\ \Gamma;\ \Theta \vdash_{\omega} p \Rightarrow \{\ \overline{\ell}\ \} \qquad \Delta;\ \Gamma \vdash_{\omega} p : \tau^{\text{XI}}\end{gathered}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\&r\ \omega\ p} : \&r\ \omega\ \tau^{\text{XI}} \Rightarrow \Gamma[r \mapsto \{\ \overline{\ell}\ \}]}$$

T-LetRegion

$$\frac{\Sigma;\ \Delta;\ \Gamma,\ r \mapsto \{\};\ \Theta \vdash \boxed{e} : \tau^{\text{SI}} \Rightarrow \Gamma',\ r \mapsto \{\overline{\ell}\}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{letrgn}\ \texttt{<}r\texttt{>}\ \{\ e\ \}} : \tau^{\text{SI}} \Rightarrow \Gamma'}$$

T-Branch

$$\frac{\begin{gathered}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1} : \texttt{bool} \Rightarrow \Gamma_1 \qquad \Sigma;\ \Delta;\ \Gamma_1;\ \Theta \vdash \boxed{e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2\\ \Sigma;\ \Delta;\ \Gamma_1;\ \Theta \vdash \boxed{e_3} : \tau_3^{\text{SI}} \Rightarrow \Gamma_3 \qquad \tau^{\text{SI}} = \tau_2^{\text{SI}} \vee \tau^{\text{SI}} = \tau_3^{\text{SI}}\\ \Delta;\ \Gamma_2;\ \Theta \vdash^{+} \tau_2^{\text{SI}} \rightsquigarrow \tau^{\text{SI}} \dashv \Gamma_2' \qquad \Delta;\ \Gamma_3;\ \Theta \vdash^{+} \tau_3^{\text{SI}} \rightsquigarrow \tau^{\text{SI}} \dashv \Gamma_3' \qquad \Gamma_2' \Cup \Gamma_3' = \Gamma'\end{gathered}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{if}\ e_1\ \{\ e_2\ \}\ \texttt{else}\ \{\ e_3\ \}} : \tau^{\text{SI}} \Rightarrow \Gamma'}$$

T-Seq

$$\frac{\begin{gathered}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1} : \tau_1^{\text{SI}} \Rightarrow \Gamma_1\\ \Sigma;\ \Delta;\ \text{gc-loans}_{\Theta}(\Gamma_1);\ \Theta \vdash \boxed{e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2\end{gathered}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1;\ e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2}$$

T-Let

$$\frac{\begin{gathered}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1} : \tau_1^{\text{SI}} \Rightarrow \Gamma_1 \qquad \Delta;\ \Gamma_1;\ \Theta \vdash^{+} \tau_1^{\text{SI}} \rightsquigarrow \tau_a^{\text{SI}} \dashv \Gamma_1'\\ \forall r \in \text{free-regions}(\tau_a^{\text{SI}}).\ \Gamma_1' \vdash r\ \texttt{rnrb} \qquad \Sigma;\ \Delta;\ \text{gc-loans}_{\Theta}(\Gamma_1',\ x\ :\ \tau_a^{\text{SI}});\ \Theta \vdash \boxed{e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2,\ x\ :\ \tau^{\text{SD}}\end{gathered}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{let}\ x : \tau_a^{\text{SI}} = e_1;\ e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2}$$

T-Drop

$$\frac{\Gamma(\pi) = \tau_{\pi}^{\text{SI}} \qquad \Sigma;\ \Delta;\ \Gamma[\pi \mapsto \tau_{\pi}^{\text{SI}^{\dagger}}];\ \Theta \vdash \boxed{e} : \tau^{\text{SX}} \Rightarrow \Gamma_f}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau^{\text{SX}} \Rightarrow \Gamma_f}$$

T-AssignDeref

$$\frac{\begin{gathered}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau_n^{\text{SI}} \Rightarrow \Gamma_1 \qquad \Delta;\ \Gamma_1 \vdash_{\texttt{uniq}} p : \tau_o^{\text{SI}}\\ \Delta;\ \Gamma_1;\ \Theta \vdash^{+} \tau_n^{\text{SI}} \rightsquigarrow \tau_o^{\text{SI}} \dashv \Gamma' \qquad \Delta;\ \Gamma';\ \Theta \vdash_{\texttt{uniq}} p \Rightarrow \{\ \overline{\ell}\ \}\end{gathered}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{p := e} : \texttt{unit} \Rightarrow \Gamma'}$$

T-Assign

$$\frac{\begin{gathered}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau^{\text{SI}} \Rightarrow \Gamma_1 \qquad \Gamma_1(\pi) = \tau^{\text{SX}} \qquad \tau^{\text{SX}} = \&r\ \omega\ \tau^{\text{XI}} \implies r \text{ is unique to } \pi \text{ in } \Gamma_1\\ \Delta;\ \Gamma_1 \trianglerighteq *\pi;\ \Theta \vdash^{=} \tau^{\text{SI}} \rightsquigarrow \tau^{\text{SX}} \dashv \Gamma' \qquad (\tau^{\text{SX}} = \tau^{\text{SD}} \vee \Delta;\ \Gamma';\ \Theta \vdash_{\texttt{uniq}} \pi \Rightarrow \{\ {}^{\texttt{uniq}}\pi\ \})\end{gathered}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\pi := e} : \texttt{unit} \Rightarrow \Gamma'[\pi \mapsto \tau^{\text{SI}}]}$$

T-Closure

$$\frac{\begin{gathered}\text{free-vars}(e) \setminus \overline{x} = \overline{x_f} \qquad \text{free-nc-vars}_{\Gamma}(e) \setminus \overline{x} = \overline{x_{nc}}\\ \overline{r} = \overline{\text{free-regions}(\Gamma(x_f))},\ (\text{free-regions}(e) \setminus (\overline{\text{free-regions}(\tau^{\text{SI}})}, \text{free-regions}(\tau_r^{\text{SI}})))\\ \mathcal{F}_c = \overline{r \mapsto \Gamma(r)},\ \overline{x_f\ :\ \Gamma(x_f)} \qquad \forall r_p \in \bigcup_{i=1}^{n} \text{free-regions}(\tau_i^{\text{SI}}) \cup \text{free-regions}(\tau_r^{\text{SI}}).\ \Gamma(r_p) = \emptyset\\ \Sigma;\ \Delta;\ \Gamma[\overline{x_{nc} \mapsto \Gamma(x_{nc})^{\dagger}}] \natural \mathcal{F}_c,\ x_1\ :\ \tau_1^{\text{SI}},\ \ldots,\ x_n\ :\ \tau_n^{\text{SI}};\ \Theta \vdash \boxed{e} : \tau_r^{\text{SI}} \Rightarrow \Gamma' \natural \mathcal{F}\end{gathered}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{|x_1 : \tau_1^{\text{SI}},\ \ldots,\ x_n : \tau_n^{\text{SI}}|\ \rightarrow\ \tau_r^{\text{SI}}\ \{\ e\ \}} : (\tau_1^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}}) \overset{\mathcal{F}_c}{\rightarrow} \tau_r^{\text{SI}} \Rightarrow \Gamma'}$$

T-Tuple

$$\frac{\forall i \in \{\ 1\ \ldots\ n\ \}.\ \Sigma;\ \Delta;\ \Gamma_{i-1};\ \Theta,\ \tau_1^{\text{SI}},\ \ldots,\ \tau_{i-1}^{\text{SI}} \vdash \boxed{e_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_i}{\Sigma;\ \Delta;\ \Gamma_0;\ \Theta \vdash \boxed{(e_1,\ \ldots,\ e_n)} : (\tau_1^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}}) \Rightarrow \Gamma_n}$$

T-While

$$\frac{\begin{gathered}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1} : \texttt{bool} \Rightarrow \Gamma_1 \qquad \Sigma;\ \Delta;\ \Gamma_1;\ \Theta \vdash \boxed{e_2} : \texttt{unit} \Rightarrow \Gamma_2\\ \Sigma;\ \Delta;\ \Gamma_2;\ \Theta \vdash \boxed{e_1} : \texttt{bool} \Rightarrow \Gamma_2 \qquad \Sigma;\ \Delta;\ \Gamma_2;\ \Theta \vdash \boxed{e_2} : \texttt{unit} \Rightarrow \Gamma_2\end{gathered}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{while}\ e_1\ \{\ e_2\ \}} : \texttt{unit} \Rightarrow \Gamma_2}$$

T-Abort

$$\frac{}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{abort!(str)}} : \tau^{\text{SX}} \Rightarrow \Gamma}$$

Fig. 5. Selected Oxide Typing Rules

3.5 Typechecking Oxide Programs

Figure 5 presents a selection of Oxide typing rules. In every rule, we highlight the expression being typechecked with a $\boxed{\text{framebox}}$. The shape of our typing judgement is $\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash e : \tau \Rightarrow \Gamma'$: we typecheck e in a global environment Σ, type environment Δ, temporary typing Θ, and stack typing Γ, producing an updated stack typing Γ' for typing the continuation of e. These rules rely on the region rewriting and outlives judgments (Figure 6), which we'll discuss below, and the ownership safety judgment (Figure 4), which we discussed in §3.4. We elide the various well-formedness judgments (for types, stack typings, etc.); see the appendix (§B.1).

To best understand our typing judgment as a whole, it is useful to first know a bit about what lies ahead in the metatheory (§3.7). In our type preservation proof, we need to maintain the well-typedness of values stored on the stack in Oxide. Since our values include closures which themselves may introduce more aliasing, we then need to maintain our ownership safety judgment. To make this possible, there are a number of restrictions that arise throughout the type system on how regions annotate the program (and discussed further in §5). We urge readers to keep this in mind.

Moving. In Oxide, as in Rust, owned values are moved or copied out of a place π when used, just as we saw in our first example in §2.1 where `pt` was moved to `x` and thus could not be bound again to `y`. In order to move π, three conditions must hold: (1) π must be able to be used `uniq`-ly (checked using the ownership safety judgment in Figure 4 from §3.4); (2) π must have a sized and initialized (not dead) type τ^{SI} in Γ; and (3) this type τ^{SI} must be `noncopyable`.[6] If these requirements hold, then we use a dagger to mark the place π dead in the continuation stack typing, preventing further expressions from reusing it. Requirement (1) is needed to ensure that we do not invalidate any existing references to π by moving it and requirement (2) ensures that there is currently data owned by π. If requirement (3) does not hold, we'll copy rather than move which permits more programs to typecheck since T-Copy does not mark the copied place dead in the continuation stack typing as T-Move does. Further, unlike moves (which are disallowed through dereferences, leading to the restriction to places π rather than place expressions p in T-Move), the copying variant T-Copy can happen through a dereference and thus handles place expressions generally.

Borrowing. As with moving, borrowing $\&r\,\omega\,p$ relies on our ownership safety judgment (Figure 4) with the `uniq` and `shrd` modalities corresponding precisely to the invariants of unique and shared pointers. Namely, when ω is `uniq`, we require that the place expression p have no extant loans in Γ and when ω is `shrd`, we require no extant *unique* loans. To actually know the type of the reference as a whole, we also have to know the type of the place expression itself and we rely on an auxillary judgment $\Delta;\ \Gamma \vdash_{\omega} p : \tau^{\text{XI}}$ (defined in the appendix) to compute the type τ^{XI} by starting with the type of its root identifier and following the sequence of projections and dereferences from p through the type. For example, if we had the place expression `*(x.0)` where `x` is a product of references with the type `&'a uniq u32`, this judgment would produce the type `u32`. Much like how T-Move updates the continuation stack typing to prevent further uses of the moved place, T-Borrow updates the continuation stack typing to associate the region r used for the borrow with the loans that represent where the new reference may point (i.e. its provenance). In many simple cases (such as borrowing a binding `x` with type `u32`), there will be only one loan (corresponding to precise knowledge of where it points), but as we will see with branching, these loan sets may grow larger to account for information loss inherent to static analysis of reference provenance.

[6]We've elided definitions of `copyable` and `noncopyable`, but they're straightforward. Intuitively, a type is safe to copy if none of its constituent parts are unique. Thus, all types that don't contain a unique reference are copyable. Generic types are always non-copyable. In Rust, `copyable` is actually the `Copy` trait, but `copyable` can be thought of as special casing it.

Region Rewriting and Outlives. We examine region rewriting next (Figure 6) since some of the typing rules discussed below require it. The region rewriting judgment $\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma'$ says under Δ, Θ, and Γ, we can rewrite the regions from τ_1 into the corresponding regions in τ_2 which will then be interpreted under Γ'. We produce an output Γ' with updated regions to be used when typing the continuation after an appeal to region rewriting. The need for this judgment arises from the need for values to be given the same type, in spite of the fact that they can be annotated differently originally. Consider, for example, a branching term such as `if cond { &'a uniq x } else { &'b uniq y }`. In this case, the type of each side of the branch will be something like `&'a uniq u32` and `&'b uniq u32` respectively, but we need to give an overall type for the term. To deal with this, we pick one of the two types and use rewriting to write the other one into the chosen type. For safety reasons, we pick the region with the *shortest* scope.

The rewriting judgment itself is fairly straightforward: it is reflexive and transitive. Each rule for larger types recursively rewrites in any smaller types, threading the output environment much like our typing judgment. The actual rewriting portion arises, as one might expect, in the rule for references (RR-Reference) which appeals to a judgment on regions which says that the region from τ_1 *outlives* the region from τ_2 while performing the work required to enable the rewriting.

The outlives judgment (Figure 6) $\Delta;\ \rho_1;\ \Gamma \vdash^{\mu} \rho_2 :> \Gamma' \dashv$ says under Δ and Γ, ρ_1 outlives ρ_2 rewriting the latter in Γ' according to the mode μ. Every region outlives itself (reflexivity). An abstract region outlives another if there's a corresponding outlives relation in Δ (OL-BothAbstract) or if we can transitively put together outlives relations from Δ (OL-Trans).[7] OL-CombineConcrete (in + mode) says that r_1 outlives r_2 if it occurs earlier than r_2 in Γ. It also requires that there not exist any references with either region which have been reborrowed ($\Gamma \vdash r_1$ `rnrb` and $\Gamma \vdash r_2$ `rnrb`, where `rnrb` is an abbreviation for region-not-reborrowed). This invariant ensures that value typing is preserved under region rewriting. When this is the case, the output typing Γ' is updated to associate r_2 with the union of the loans from both r_1 and r_2. The variant OL-CheckConcrete (in = mode) has the same obligations, but instead leaves the output unchanged. The behavioral difference between these rules is the reason for the rewriting modes.

The last two rules say when a concrete region outlives an abstract one and vice versa. In essence, a concrete region r can only outlive an abstract region ϱ (OL-ConcreteAbstract) if r was *reborrowed*. The first two premises check for reborrowing: r's loan set must be non-empty (otherwise there is no reborrow), and must consist solely of place expressions $\overline{p}$ (since place expressions, unlike places, contain dereferences, which identifies this as a reborrow instead of a borrow). The third premise collects all the regions $\overline{\rho_i}$ that annotate any references dereferenced in each place expression p_i (see the type-computation judgment $\Delta;\ \Gamma \vdash_{\omega} p : \tau,\ \{\ \overline{\rho}\ \}$ in the appendix (§B.5)), while the last premise ensures that all of these outlive ϱ. The final rule, OL-AbstractConcrete, says that an abstract region *always* outlives a concrete region. This is subtle but makes sense because any abstract region ϱ is bound in a top-level function (recall that closures don't abstract over regions), while a concrete region r must be bound by `letrgns` *inside* the function body. Ultimately, any concrete region r' that gets substituted for ϱ upon application will already exist before r (even for recursive calls), meaning it outlives r.

Branching and Sequencing. The next two rules illustrate how stack typings are threaded through larger programs since the form of our typing judgment requires each rule to specify its continuation's stack typing. T-Branch uses the stack typing Γ_1 that we get from typing the conditional e_1 when typing each of the two branches. The type τ^{SI} ascribed to the overall expression must be a supertype of the types τ_2^{SI} and τ_3^{SI} of each branch and equal to one of them. Additionally, branching uses a union operation $\Cup$ to combine the output stack typings from each branch to produce the final

[7] We do not need transitivity for concrete regions beyond what we can already conclude from the remaining OL rules.

$$\text{Region Rewriting Modes} \quad \mu \quad ::= \quad + \mid \boxplus \mid =$$

$$\boxed{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma'}$$

$$\textsc{RR-Refl}\ \frac{}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_1 \dashv \Gamma}$$

$$\textsc{RR-Trans}\ \frac{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma' \qquad \Delta;\ \Gamma';\ \Theta \vdash^{\mu} \tau_2 \rightsquigarrow \tau_3 \dashv \Gamma''}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_3 \dashv \Gamma''}$$

$$\textsc{RR-Dead}\ \frac{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1^{\mathrm{SI}} \rightsquigarrow \tau_2^{\mathrm{SI}} \dashv \Gamma'}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1^{\mathrm{SI}} \rightsquigarrow \tau_2^{\mathrm{SI}^{\dagger}} \dashv \Gamma}$$

$$\textsc{RR-Tuple}\ \frac{\forall i \in \{\,1 \ldots n\,\}.\ \Delta;\ \Gamma_{n-1};\ \Theta \vdash^{\mu} \tau_i \rightsquigarrow \tau_i' \dashv \Gamma_i}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} (\tau_1 \ldots \tau_n) \rightsquigarrow (\tau_1' \ldots \tau_n') \dashv \Gamma_n}$$

$$\textsc{RR-Reference}\ \frac{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \rho_1 :> \rho_2 \dashv \Gamma' \qquad \Delta;\ \Gamma';\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma''}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \&\rho_1\ \omega\ \tau_1 \rightsquigarrow \&\rho_2\ \omega\ \tau_2 \dashv \Gamma''}$$

$$\boxed{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \rho_1 :> \rho_2 \dashv \Gamma'}$$

$$\textsc{OL-Refl}\ \frac{}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \rho :> \rho \dashv \Gamma}$$

$$\textsc{OL-Trans}\ \frac{\begin{array}{c}\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \varrho_1 :> \varrho_2 \dashv \Gamma' \\ \Delta;\ \Gamma';\ \Theta \vdash^{\mu} \varrho_2 :> \varrho_3 \dashv \Gamma''\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \varrho_1 :> \varrho_3 \dashv \Gamma''}$$

$$\textsc{OL-BothAbstract}\ \frac{\varrho_1 : \mathsf{RGN} \in \Delta \qquad \varrho_2 : \mathsf{RGN} \in \Delta \qquad \varrho_1 :> \varrho_2 \in \Delta}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \varrho_1 :> \varrho_2 \dashv \Gamma}$$

$$\textsc{OL-CombineConcrete}\ \frac{\begin{array}{c}\Gamma \vdash r_1\ \mathtt{rnrb} \qquad \Gamma \vdash r_2\ \mathtt{rnrb} \qquad \Gamma;\ \Theta \vdash \{\,r_1,\ r_2\,\}\ \mathtt{clrs} \\ r_1\ \mathtt{occurs\ before}\ r_2\ \mathtt{in}\ \Gamma \qquad \{\,\overline{\ell}\,\} = \Gamma(r_1) \cup \Gamma(r_2)\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash^{+} r_1 :> r_2 \dashv \Gamma[r_2 \mapsto \{\,\overline{\ell}\,\}]}$$

$$\textsc{OL-CombineConcreteUnrestricted}\ \frac{\begin{array}{c}\Gamma \vdash r_1\ \mathtt{rnrb} \qquad \Gamma \vdash r_2\ \mathtt{rnrb} \\ r_1\ \mathtt{occurs\ before}\ r_2\ \mathtt{in}\ \Gamma \qquad \{\,\overline{\ell}\,\} = \Gamma(r_1) \cup \Gamma(r_2)\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash^{\boxplus} r_1 :> r_2 \dashv \Gamma[r_2 \mapsto \{\,\overline{\ell}\,\}]}$$

$$\textsc{OL-CheckConcrete}\ \frac{\begin{array}{c}\Gamma \vdash r_1\ \mathtt{rnrb} \qquad \Gamma \vdash r_2\ \mathtt{rnrb} \\ r_1\ \mathtt{occurs\ before}\ r_2\ \mathtt{in}\ \Gamma\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash^{=} r_1 :> r_2 \dashv \Gamma}$$

$$\textsc{OL-AbstractConcrete}\ \frac{\varrho : \mathsf{RGN} \in \Delta \qquad r \in \mathrm{dom}(\Gamma)}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \varrho :> r \dashv \Gamma}$$

$$\textsc{OL-ConcreteAbstract}\ \frac{\begin{array}{c}\Gamma_{1,0}(r) = \{\,\overline{{}^{\omega}p}^{\,n}\,\} \neq \emptyset \qquad \forall i \in \{\,1 \ldots n\,\}.\ \nexists \pi.\ p_i = \pi \qquad \forall i \in \{\,1 \ldots n\,\}.\ \Delta;\ \Gamma_0 \vdash_{\mathsf{shrd}} p_i : \_,\ \overline{\rho_i}^{\,m_i} \\ \varrho : \mathsf{RGN} \in \Delta \qquad \forall i \in \{\,1 \ldots n\,\}. \forall j \in \{\,1 \ldots m_i\,\}.\ \Delta;\ \Gamma_{i,j-1};\ \Theta \vdash^{\mu} \rho_{i,j} :> \varrho \dashv \Gamma_{i,j}\end{array}}{\Delta;\ \Gamma_{1,0};\ \Theta \vdash^{\mu} r :> \varrho \dashv \Gamma_{n,m_n}}$$

Fig. 6. Region Rewriting and Outlives Relations in Oxide

stack typing Γ' for the overall expression. $\Cup$ requires that types of bound variables in the two stack typings be equal (which potentially demands use of T-Drop when typing the branches), and unions the loan sets for each region r from both stack typings (full definition in technical appendix). Note that we only need to union stack typings with identical domains — we typecheck both branches under Γ_1 so they produce output stack typings with the same domains (since `let` and `letrgn` are the only means for introducing variables and regions, but both are lexically-scoped), and region rewriting does not change the domain of stack typings between its input and its output.

When typing $e_1;\ e_2$, we typecheck e_2 under the stack typing Γ_1 we got from typechecking e_1. But, importantly, we apply a metafunction gc-loans$_{\Theta}(\cdot)$ to Γ_1 to empty out the loan sets of regions not used in Γ_1 before typing e_2 because e_1 may have been a unique reference that is thrown away at runtime before moving on to e_2. Without *garbage collecting* loans, Oxide would reject programs that are safe and accepted by Rust. Namely, this clearing allows us to handle the sort of "early dropping of references" inherent to non-lexical lifetimes. Specifically, gc-loans$_{\Theta}(\Gamma)$ empties out the loan set of each r that does not appear in any of the types in Γ or Θ. The full formal definition is present in the technical appendix.

Binding. In Oxide, T-Let is interesting in three ways. First, we allow rewriting in T-Let to a specified annotated type. This rewriting allows us to change the regions in the computed type to match the annotated type by conservatively combining the loans associated with each region into the output, as described earlier in the section on region rewriting. Then, similar to sequencing, T-Let uses the metafunction $\text{gc-loans}_\Theta(\cdot)$ to eliminate any loans that might be unnecessary as a result of e_1 potentially being promoted to the annotated type τ_a^{SI}. Additionally, in the output stack typing from e_2, we see that our binding for x must have a dead type τ^{SD} with the whole binding being dropped in the overall stack typing Γ_2 output from T-Let (since the scope of x ends at that point). The requirement that the type be dead means we must have either used T-Move to move out of that binding or we must have explicitly used T-Drop on x in the derivation for e_2, and can be thought of as a formalization of the "resource acquisition is initialization" pattern [Stroustrup 1994] since we are explicitly requiring a first-in-last-out allocation/deallocation pattern and require everything to have been used in either a move or a drop before we can end its scope.

Drop. As alluded to in the previous two paragraphs, Oxide has a rule called T-Drop which is used non-deterministically during typechecking to mark a particular place π as being dead. This rule corresponds roughly to a conventional weakening rule where π "doesn't exist" (in this case, is dead) in the premise, but exists in the conclusion. The main difference is that while the data is thought to be deallocated, the name is still in-scope to be dropped when its scope ends in T-Let.

Assignment. Assignment is interesting in a few ways. First, assignment is broken up into two rules T-Assign and T-AssignDeref where the former is able to assign to a place π that is dead, and the latter is able to assign to a place through a reference (i.e. by using dereferencing). The basic structure of each rule is the same. For both rules, we typecheck the new expression to be assigned, look up the type of the place we're assigning to (a lookup in T-Assign and a type computation in T-AssignDeref), check compatibility of the new expression's type with the type of the place we're assigning to, and then finally check that it's safe to use the place we're assigning to uniquely according to ownership safety.

The differences between the two play a fundamental role in allowing us to appropriately model how assignment works in Rust. Notably, the region rewriting judgment in T-Assign uses the checking mode (denoted =) to limit how conservative the borrow checker need be after an assignment. As discussed earlier, this mode does not change its output environment (thus, $\Gamma' = \Gamma_1 \rhd *\pi$ in T-Assign). This is okay in context because after the typing rule is done, the type of π is updated to τ^{SI} (the type of its new value) in the continuation. A similar update in T-AssignDeref would entail updating the types of arbitrarily many bindings, and so instead the more conservative combine mode (denoted +) is used. Further, in T-Assign, we employ the operation $\Gamma_1 \rhd *\pi$ for the input environment to the region rewriting judgment. This operation is defined to remove any loans prefixed by $*\pi$ from every loan set in Γ_1. The $\rhd$ operation (informally called the "kill rules" in Rust) amounts to erasing reborrowing relationships that no longer hold as a result of this assignment.

Concretely, consider an environment with two references, one named `x` with type `&'x uniq u32` and another named `y` reborrowed from `x` with type `&'y uniq u32`. This means in our environment we would have loan sets that look something like `'x` $\mapsto \{ \overline{\ell} \}$ and `'y` $\mapsto \{ \overline{\ell}, {}^{\text{uniq}} *x \}$. If we were to then assign to `x` some unrelated reference, the kill rules would delete the loan $*x$ in the loan set of `'y` since after the assignment ran, the two would represent distinct and disjoint references.

The last difference between the two assignment rules is presence of an additional obligation in T-Assign: τ^{SX} is unique to π in Γ_1. This obligation means that there are not any places in Γ_1 that share the outermost region of τ^{SX} (i.e. if $\tau^{\text{SX}} = \&r\ \omega\ \tau^{\text{XI}}$, then r would be its outermost region). This allows us to guarantee that the garbage collection discussed for the sequencing rule will always clear out this outermost region r before the subsequent expression is typechecked, helping us

T-AppClosure

$$\frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_f} : \forall\text{<>}(\tau_1^{\text{SI}},\ \dots,\ \tau_n^{\text{SI}}) \overset{\Phi_c}{\rightarrow} \tau_f^{\text{SI}} \Rightarrow \Gamma_0 \\ \forall i \in \{\ 1\ \dots\ n\ \}.\ \Sigma;\ \Delta;\ \Gamma_{i-1};\ \Theta,\ \tau_1^{\text{SI}}\ \dots\ \tau_{i-1}^{\text{SI}} \vdash \boxed{e_i} : \tau_{i'}^{\text{SI}} \Rightarrow \Gamma_i \qquad \Delta;\ \Gamma_i;\ \Theta \vdash^{\boxplus} \tau_{i'}^{\text{SI}} \rightsquigarrow \tau_i^{\text{SI}} \dashv \Gamma_i' \\ \forall i \in \{\ 1\ \dots\ n\ \}.\ \forall r \in \text{free-regions}(\tau_i^{\text{SI}}).\ \Gamma_n' \vdash r\ \mathsf{rnrb}\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_f(e_1,\ \dots,\ e_n)} : \tau_f^{\text{SI}} \Rightarrow \Gamma_n'}$$

Fig. 7. Oxide Typing Rule for Application

greatly in our proofs. It's important to note that this obligation pertains to annotations added to the Oxide program compared to Rust, and so it only limits the patterns of region annotation that can be applied, rather than the space of Rust programs that can be typechecked in Oxide.

Closures and Application. Closures in Oxide correspond to *move closures* in Rust which move or copy their free variables from the outer environment into the closure.[8] As such, T-Closure must compute the captured frame by looking at the free variables (and free regions) of the closure's body, and it must mark dead (add daggers to the types of) any variables in the stack typing with *non-copyable* types. The captured frame is suspended over the arrow in the function type to keep track of the fact that the data caught up in the closure is still alive (and thus must be considered in ownership safety). We elide the rule for top-level function definitions, which gives a function f the type that f is annotated with in Σ, relying on well-formedness of Σ to know that this is okay.

The rule for application (T-AppClosure in Figure 7) is roughly as one would expect: we typecheck the function, and then the arguments, threading through the environments. However, at each step, we do a region rewriting using the unrestricted combine mode (denoted $\boxplus$) for the computed argument types to the annotated parameter types. This unrestricted mode allows us to push loans from the surrounding context into the regions used in the closure's signature, a behavior ruled out by the conventional combine mode (+) because they would enable degenerate annotation patterns that make it difficult to prove soundness. For top-level functions, we have an additional rule T-AppFunction in the technical appendix that does not use a rewriting for the types, but (1) substitutes all frame, type, and region variables in the types and (2) checks that any outlives bounds specified on the function signature hold (via outlives in Figure 6). This does not appear in T-AppClosure since closures cannot be polymorphic, nor can they possess where bounds.

Values and Aggregates. The typing rules for base types (T-u32, T-True, T-False, etc.) are standard, and leave the type environment unchanged in their output. Aggregate structures like tuples check the types of their components while threading through the environments in left-to-right order. This left-to-right ordering for typechecking corresponds to the ordering implemented by Rust's typechecker and borrow checker. One subtlety to note (discussed already with the last example in §3.3) is that when typechecking a component e_i of the tuple, we add the types of all earlier tuple components to the temporary typing Θ. This is needed because e_i might be a let-binding or sequencing expression that invokes $\text{gc-loans}_{\Theta}(\cdot)$ on its environment during typechecking. If the types of the earlier tuple components we've just typechecked aren't present in the environment, to serve as roots for the regions mentioned in those types, we may end up incorrectly garbage collecting the loans that these regions map to. This would make programs with subsequent borrows typecheck even when the borrow should not be allowed. The elided typing rule for arrays is similar.

The formalism of Oxide omits a specific treatment of structs, but we note that they are essentially the same as tuples, only featuring a tag that must also be checked. Our implementation which we discuss in §3.8 relies on exactly this approach to support structs.

[8] Rust's standard closures implicitly introduce borrowed temporaries for all the free variables. We can recover this behavior via a simple, local transformation to move closures.

$$
\begin{array}{llcl}
\text{Referent} & \mathcal{R} & ::= & x \mid \mathcal{R}.n \mid \mathcal{R}[n] \mid \mathcal{R}[n_1..n_2] \\
\text{Expressions} & e & ::= & \ldots \mid \texttt{framed}\ e \mid \texttt{shift}\ e \\
 & & \mid & \llbracket v_1, \ldots, v_n \rrbracket \mid \texttt{dead} \mid \texttt{ptr}\ \mathcal{R} \\
 & & \mid & \langle \varsigma,\ |x_1 : \tau_1^{\text{SI}}, \ldots, x_n : \tau_n^{\text{SI}}| \rightarrow \tau_r^{\text{SI}}\ \{\ e\ \}\ \rangle \\
\text{Eval. Contexts} & C & ::= & \square \mid \&\rho\ \omega\ p[C] \mid \&\rho\ \omega\ p[C..\hat{e}] \mid \&\rho\ \omega\ p[v..C] \\
 & & \mid & \texttt{let}\ x : \tau^{\text{SI}} = C;\ e \mid \texttt{letrgn}\ \texttt{<}r\texttt{>}\ \{\ C\ \} \\
 & & \mid & p \coloneqq C \mid C;\ e \mid \texttt{framed}\ C \mid \texttt{shift}\ C \mid \texttt{shiftprov}\ C \\
 & & \mid & C\texttt{::<}\overline{\Phi},\ \overline{\rho},\ \overline{\tau^{\text{SI}}}\texttt{>}(\hat{e}_1, \ldots, \hat{e}_n) \\
 & & \mid & v\texttt{::<}\overline{\Phi},\ \overline{\rho},\ \overline{\tau^{\text{SI}}}\texttt{>}(v_1, \ldots, v_m, C, \hat{e}_1, \ldots, \hat{e}_n) \\
 & & \mid & p[C] \mid \texttt{if}\ C\ \{\ e_1\ \}\ \texttt{else}\ \{\ e_2\ \} \mid \texttt{while}\ e_1\ \{\ e_2\ \} \\
 & & \mid & (v_1, \ldots, v_m, C, \hat{e}_1, \ldots, \hat{e}_n) \\
 & & \mid & \texttt{Left::<}\tau_1^{\text{SI}}, \tau_2^{\text{SI}}\texttt{>}(C) \mid \texttt{Right::<}\tau_1^{\text{SI}}, \tau_2^{\text{SI}}\texttt{>}(C) \\
 & & \mid & \texttt{match}\ C\ \{\ \texttt{Left}(x_1) \Rightarrow e_1,\ \texttt{Right}(x_2) \Rightarrow e_2\ \} \\
 & & & \\
\text{Values} & v & ::= & c \mid f \mid \texttt{dead} \mid (v_1, \ldots, v_n) \mid [v_1, \ldots, v_n] \mid \llbracket v_1, \ldots, v_n \rrbracket \mid \texttt{ptr}\ \mathcal{R} \\
 & & \mid & \langle \varsigma,\ |x_1 : \tau_1^{\text{SI}}, \ldots, x_n : \tau_n^{\text{SI}}| \rightarrow \tau_r^{\text{SI}}\ \{\ e\ \}\ \rangle \\
\text{Value Contexts} & \mathcal{V} & ::= & \square \mid (v_1, \ldots, \mathcal{V}, \ldots, v_n) \mid [v_1, \ldots, \mathcal{V}_1, \ldots, \mathcal{V}_m, \ldots, v_n] \\
\text{Stacks} & \sigma & ::= & \bullet \mid \sigma \natural \varsigma \\
\text{Stack Frame} & \varsigma & ::= & \bullet \mid \varsigma,\ x \mapsto v
\end{array}
$$

Fig. 8. Oxide Syntax Extensions for Dynamics

Remaining Rules. The remaining rules in Figure 5 are straightforward or covered earlier. Elided typing rules all concern arrays and are given in the technical appendix (§B.4).

3.6 Operational Semantics

For our operational semantics, we extend the syntax of Oxide in Figure 8 with terms that only arise at runtime. First, to be able to specify what "address" a pointer points to, we introduce an abstract form of memory addresses called *referents*. Referents $\mathcal{R}$ essentially record what the offsets are from a variable on the stack in order to specify a precise "memory address," (e.g., a particular element of an array or tuple, or a particular slice of an array). We also include some administrative forms: (1) `framed` e and `shift` e which are used when evaluating application and let bindings discussed below, (2) `dead` (the dead value), and (3) $\llbracket v_1, \ldots, v_n \rrbracket$ which is a dynamically-sized slice of an array. Then, we introduce value forms including pointers to referents, and closures packaged with their environment ς. Figure 8 also includes stacks σ as a sequence of stack frames ς, and value contexts $\mathcal{V}$ which allow array values to be decomposed with multiple holes when dealing with slices.

In Figure 9, we present a selection of our small-step operational semantics which is defined using Felleisen and Hieb-style left-to-right evaluation contexts [Felleisen and Hieb 1992] over configurations of the form $(\sigma;\ \boxed{e})$. Since our semantics uses referents $\mathcal{R}$ as an abstract version of memory addresses, some of our rules rely on a notion of place-expression evaluation, $\sigma \vdash p \Downarrow \mathcal{R} \mapsto \mathcal{V}[v]$ (Figure 9, top), which should be read as: p evaluates to $\mathcal{R}$, which maps to v with a surrounding context $\mathcal{V}$ in σ.

The evaluation rules are straightforward: E-Move returns a value by moving it off of the stack σ, replacing it with `dead`. E-Copy copies the value from the stack. E-Borrow creates a pointer value to the referent $\mathcal{R}$. Branching is completely standard, hence elided. Assignment, similar to E-Copy and E-Borrow, uses the place-expression evaluation rules, but instead cares specifically about the value context $\mathcal{V}$, rather than the value v. Assignment also decomposes the computed referent $\mathcal{R}$ to get its root identifier x. Then, it updates the stack by maintaining this value context when it updates x (mapping it to $\mathcal{V}[v]$).

Binding and the Stack. Bindings are interesting in that they introduce our two administrative forms, `framed` e and `shift` e. For instance, in E-Let, we step to `shift` e rather than e alone in

$\boxed{\sigma \vdash p \Downarrow \mathcal{R} \mapsto \mathcal{V}[v]}$ $\quad \sigma \vdash p^{\square}[x] \Downarrow \mathcal{R} \mapsto \mathcal{V}[v] \overset{\text{def}}{=} \sigma \vdash p^{\square} \times x \Downarrow (\mathcal{R}, \mathcal{V}, v).$

$\boxed{\sigma \vdash p^{\square} \times \mathcal{R} \Downarrow (\mathcal{R}', \mathcal{V}, v)}$ $\quad$ read: "$\mathcal{R}$ in a context $p^{\square}$ computes to $\mathcal{R}'$ which maps to v in σ."

P-Referent
$$\frac{\sigma \vdash \mathcal{R} \Downarrow \mathcal{V} \times v}{\sigma \vdash \square \times \mathcal{R} \Downarrow (\mathcal{R}, \mathcal{V}, v)}$$

P-Proj
$$\frac{\sigma \vdash p^{\square} \times \mathcal{R}_1 \Downarrow (\mathcal{R}_2, \mathcal{V}, (v_0, \ldots, v_i, \ldots, v_n))}{\sigma \vdash p^{\square}[\square.i] \times \mathcal{R}_1 \Downarrow (\mathcal{R}_2.i, \mathcal{V}[(v_0, \ldots, \square, \ldots, v_n)], v_i)}$$

P-DerefPtr
$$\frac{\sigma \vdash \square \times \mathcal{R}_1 \Downarrow (\_, \texttt{ptr}\ \pi,) \qquad \sigma \vdash p^{\square} \times \pi \Downarrow (\mathcal{R}_2, \mathcal{V}, v)}{\sigma \vdash p^{\square}[*\square] \times \mathcal{R}_1 \Downarrow (\mathcal{R}_2, \mathcal{V}, v)}$$

$\boxed{\Sigma \vdash (\sigma; \boxed{e}) \rightarrow (\sigma'; \boxed{e'})}$

E-Move
$$\frac{\sigma \vdash \pi \Downarrow \pi \mapsto \_[v]}{\Sigma \vdash (\sigma; \boxed{\pi}) \rightarrow (\sigma[\pi \mapsto \texttt{dead}]; \boxed{v})}$$

E-Copy
$$\frac{\sigma \vdash p \Downarrow \_ \mapsto \_[v]}{\Sigma \vdash (\sigma; \boxed{p}) \rightarrow (\sigma; \boxed{v})}$$

E-Borrow
$$\frac{\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[\_]}{\Sigma \vdash (\sigma; \boxed{\&r\ \omega\ p}) \rightarrow (\sigma; \boxed{\texttt{ptr}\ \mathcal{R}})}$$

E-Seq
$$\frac{}{\Sigma \vdash (\sigma; \boxed{v; e}) \rightarrow (\sigma; \boxed{e})}$$

E-LetRegion
$$\frac{}{\Sigma \vdash (\sigma; \boxed{\texttt{letrgn <r> \{ } v \texttt{ \}}}) \rightarrow (\sigma; \boxed{v})}$$

E-Assign
$$\frac{\sigma \vdash p \Downarrow \mathcal{R} \mapsto \mathcal{V}[\_] \qquad \mathcal{R} = \mathcal{R}^{\square}[x]}{\Sigma \vdash (\sigma; \boxed{p := v}) \rightarrow (\sigma[x \mapsto \mathcal{V}[v]]; \boxed{()})}$$

E-Let
$$\frac{}{\Sigma \vdash (\sigma; \boxed{\texttt{let}\ x : \tau^{\text{SI}}_a = v;\ e}) \rightarrow (\sigma,\ x \mapsto v; \boxed{\texttt{shift}\ e})}$$

E-Shift
$$\frac{}{\Sigma \vdash (\sigma,\ x \mapsto v'; \boxed{\texttt{shift}\ v}) \rightarrow (\sigma; \boxed{v})}$$

E-Closure
$$\frac{\overline{x_f} = \text{free-vars}(e) \qquad \overline{x_{nc}} = \text{free-nc-vars}_{\sigma}(e) \qquad \varsigma_c = \sigma\,|_{\overline{x_f}}}{\Sigma \vdash (\sigma; \boxed{|x_1 : \tau^{\text{S}}_1, \ldots, x_n : \tau^{\text{S}}_n| \rightarrow \tau^{\text{S}}_r \{ e \}}) \rightarrow (\sigma[\overline{x_{nc} \mapsto \texttt{dead}}]; \boxed{\langle \varsigma_c,\ |x_1 : \tau^{\text{S}}_1, \ldots, x_n : \tau^{\text{S}}_n| \rightarrow \tau^{\text{S}}_r \{ e \} \rangle})}$$

E-AppClosure
$$\frac{v_f = \langle \varsigma_c,\ |x_1 : \tau^{\text{S}}_1, \ldots, x_n : \tau^{\text{S}}_n| \rightarrow \tau^{\text{S}}_r \{ e \} \rangle}{\Sigma \vdash (\sigma; \boxed{v_f(v_1, \ldots, v_n)}) \rightarrow (\sigma \natural \varsigma_c,\ x_1 \mapsto v_1, \ldots, x_n \mapsto v_n; \boxed{\texttt{framed}\ e})}$$

E-Framed
$$\frac{}{\Sigma \vdash (\sigma \natural \varsigma; \boxed{\texttt{framed}\ v}) \rightarrow (\sigma; \boxed{v})}$$

E-While
$$\frac{}{\Sigma \vdash (\sigma; \boxed{\texttt{while}\ e_1 \{ e_2 \}}) \rightarrow (\sigma; \boxed{\texttt{if}\ e_1 \{ e_2;\ \texttt{while}\ e_1 \{ e_2 \} \}\ \texttt{else} \{ () \}})}$$

Fig. 9. Selected Place Expression Evaluation Rules (top) and Reduction Rules (bottom)

order to ensure that the binding for x is well-scoped and ends when it should (seen in E-Shift). In E-AppClosure, we similarly step to `framed` e to ensure that after evaluating the body of the closure we drop the stack frame from that function call (seen in E-Framed). Both E-Shift and E-Framed rely crucially on the fact that our stack σ is ordered — they must match the most recent entry.

3.7 Well-typed Oxide programs won't go wrong!

We prove syntactic type safety for Oxide using progress and preservation [Wright and Felleisen 1992].

Lemma 3.1 (Progress). *If* $\Sigma; \bullet; \Gamma; \Theta \vdash \boxed{e} : \tau^{SI} \Rightarrow \Gamma'$ *and* $\Sigma \vdash \sigma : \Gamma$*, then either* e *is a value,* e *is an* `abort!` $(\ldots)$*, or* $\exists\, \sigma', e'.\ \Sigma \vdash (\sigma; \boxed{e}) \rightarrow (\sigma'; \boxed{e'})$.

The Progress lemma says that if we can typecheck e under a valid global environment Σ, temporary typing Θ, and stack typing Γ, *and* we have a stack σ that satisfies this stack typing Γ, then either e is a value, an `abort!` expression, or we can take a step. Our stack typing judgment $\Sigma \vdash \sigma : \Gamma$ says that each value in the stack σ has the corresponding type attributed to it in the typing Γ. The proof proceeds by induction on the typing derivation for e, and relies on a Canonical Forms lemma and Lemma 3.2 which says that place expressions can be reduced at runtime to values with their computed types. This lets us apply the rules for moves, copies, borrowing, and assignment.

Lemma 3.2 (Place Expressions Reduce). *If* $\Delta;\ \Gamma \vdash_{\omega} p : \tau^{XI}$ *and* $\Sigma \vdash \sigma : \Gamma$*, then* $\sigma \vdash p \Downarrow \mathcal{R} \mapsto \mathcal{V}[v]$ *and* $\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau^{XI} \Rightarrow \Gamma$.

Our proof of Lemma 3.2 (ifanonymousLemma 5.3Lemma E.4 in appendix) relies on the shared inductive structure of type computation $\Delta;\ \Gamma \vdash_{\omega} p : \tau^{XI}$ and place expression evaluation $\sigma \vdash p \Downarrow \mathcal{R} \mapsto \mathcal{V}[v]$.

Lemma 3.3 (Preservation). *If* $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau_1^{SI} \Rightarrow \Gamma_f$ *and* $\Sigma \vdash \sigma : \Gamma$ *and* $\Sigma;\ \Gamma \vdash \overline{v} : \Theta$ *and* $\Sigma \vdash (\sigma;\ \boxed{e}) \longrightarrow (\sigma';\ \boxed{e'})$*, then there exists* Γ_i *such that* $\Sigma \vdash \sigma' : \Gamma_i$ *and* $\Sigma;\ \Gamma_i \vdash \overline{v} : \Theta$ *and* $\Sigma;\ \bullet;\ \Gamma_i;\ \Theta \vdash \boxed{e'} : \tau_2^{SI} \Rightarrow \Gamma_f'$ *and* $\bullet;\ \tau_2^{SI};\ \Gamma_f' \vdash^{+} \tau_1^{SI} \rightsquigarrow \Gamma_s \dashv$ *and there exists* Γ_o *such that* $\Gamma_f = \Gamma_s \Cup \Gamma_o$.

The Preservation lemma says that if e has type τ_1^{SI} under a valid global environment Σ, temporary typing Θ, and stack typing Γ, *and* we have a stack σ that satisfies Γ and a sequence of values $\overline{v}$ that satisfies the temporary typing Θ, *and* we know that e can take a step under σ to the new configuration $(\sigma';\ \boxed{e'})$, then the following conditions hold. Our updated stack σ' satisfies Γ_i, our sequence of temporary values $\overline{v}$ continue to satisfy Θ, and our new expression e' typechecks with the type τ_2^{SI}. In each of these judgments, we use an intermediate stack typing Γ_i that corresponds to the changes the evaluated portion of code made to the environment. Rather than constrain the type to be the same as the type of our original e, our Preservation lemma allows τ_2^{SI} to differ in its regions by the region rewriting judgment, since evaluation potentially can lead to a type having more precise regions. Further, the output stack typing from typechecking e' is threaded through the rewriting and then ultimately said to union with some other stack typing Γ_o in order to capture the relationship between the old output environment Γ_f and the new one Γ_f' when we have taken a step into one side or the other of a branch.

As discussed in §3.5, the most challenging part of proving preservation is in showing that the various changes to the environment preserve the well-typedness of values, with closures in particular posing the greatest issue. Indeed, we believe it's clear that a formalization of Rust without closures misses a significant piece of the language's essence since closures interact consistently with all parts of the formalism. To that end, the proof of preservation uses several families of lemmas that follow the same pattern: since our preservation theorem has to maintain the well-typedness of the stack σ and temporary values $\overline{v}$, we must show that various judgments in our system preserve the well-typedness of values. We will highlight these lemmas here, noting that each require sublemmas for expressions in closure bodies remaining well-typed which subsequently requires that region rewriting and ownership safety judgments are preserved by these judgments.

Lemma 3.4 (Values are Well-Typed after Region Rewriting). *(Lemma E.13 in appendix)* *If* $\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau \Rightarrow \Gamma$ *and* $\Delta;\ \tau_2;\ \Gamma \vdash^{\mu} \tau_1 \rightsquigarrow \Gamma' \dashv$ *then* $\Sigma;\ \Delta;\ \Gamma';\ \Theta \vdash \boxed{v} : \tau \Rightarrow \Gamma'$.

Lemma 3.5 (Values are Well-Typed after Drop/GC-Loans). *(Lemma E.25)* *If* $\Sigma;\ \bullet \vdash \Gamma \triangleright \Gamma'$ *and* $\Sigma;\ \bullet;\ \Gamma;\ \bullet \vdash \boxed{v} : \Gamma(x) \Rightarrow \Gamma$*, then* $\Sigma;\ \bullet;\ \Gamma';\ \bullet \vdash \boxed{v} : \Gamma'(x) \Rightarrow \Gamma'$.

Lemma 3.6 (Values are Well-Typed under Well-Typed Extensions). *(Lemma E.34)* *If* $\Sigma;\ \bullet;\ \Gamma \vdash \tau_x^{SI}$ *and* $\forall r \in$ *free-regions*(τ_x^{SI}). $\Gamma \vdash r\ \texttt{rnrb}$ *and* $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau^{SI} \Rightarrow \Gamma$*, then* $\Sigma;\ \bullet;\ \Gamma\ ,\ x : \tau_x^{SI};\ \Theta \vdash \boxed{v} : \tau^{SI} \Rightarrow \Gamma\ ,\ x : \tau_x^{SI}$.

Lemma 3.7 (Values are Well-Typed after Assignment). *(Lemma E.40)* *If* $\Gamma(\pi_a) = \tau^{SX} \land \Delta;\ \Gamma \trianglerighteq *\pi_a;\ \Theta \vdash^{=} \tau^{SI} \rightsquigarrow \tau^{SX} \dashv \Gamma' \land \bullet;\ \Gamma';\ \Theta \vdash_{\texttt{uniq}} \pi_a \Rightarrow \{\ {}^{\texttt{uniq}}\pi_a\ \} \land \Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau \Rightarrow \Gamma$*, then* $\Sigma;\ \bullet;\ \textit{gc-loans}_{\Theta}(\Gamma'[\pi_a \mapsto \tau^{SI}]);\ \Theta \vdash \boxed{v} : \tau \Rightarrow \textit{gc-loans}_{\Theta}(\Gamma'[\pi_a \mapsto \tau^{SI}])$.

passing		disqualified						
borrowck	nll	heap	out-of-scope library	enums	statics & consts	traits	uninitialized variables	misc.
89	119	63	40	50	40	93	40	81

Fig. 10. Tested Semantics Results

Lemma 3.8 (Values are Well-Typed under Safe Loan Updates). *(Lemma E.62)* *If* $\bullet; \Gamma; \Theta \vdash_\omega p \Rightarrow \{\, \overline{\ell} \,\}$ *and* $\Gamma; \Theta \vdash r$ `rnic` *and* $\Gamma(r) = \emptyset$ *and* $\Sigma; \bullet; \Gamma; \Theta \vdash \boxed{v} : \tau^{SI} \Rightarrow \Gamma$*, then* $\Sigma; \bullet; \Gamma[r \mapsto \{\, \overline{\ell} \,\}]; \Theta \vdash \boxed{v} : \tau^{SI} \Rightarrow \Gamma[r \mapsto \{\, \overline{\ell} \,\}]$.

This pattern of a value typing lemma demanding a whole family of related lemmas is consistent throughout the supporting lemmas, and they eventually rely on an ownership safety lemma where we need to actually consider how the various judgments each change the environment and argue that the separation provided by closures — including for regions — is sufficient to prevent those changes from breaking ownership safety derivations in the closure bodies themselves. In each case, the hardest part of proving each family is ensuring that the induction hypothesis for the ownership safety lemma was rich enough to address the changes effects on the loans in the environment and the reborrow exclusion list discussed in §3.4.

With Lemma 3.1 and Lemma 3.3 in hand, we also prove a conventional type safety theorem as a corollary. However, we note that Preservation itself represents a more interesting metatheoretic result because it requires us to show that all of the type system invariants (namely, the aliasing requirements) are maintained throughout the execution of well-typed programs.

3.8 Tested Semantics

We set out at the onset to solve a particular problem — there is no high-level specification of the Rust programming language and its borrowchecker. If there were, this would be the point where we might present a proof that every expression that typechecks in Oxide also typechecks in Rust and vice versa. Since doing that is not possible, we follow Guha et al. [2010] in developing a *tested semantics* for Oxide. We built an implementation of our Oxide typechecking algorithm, OxideTC, alongside a compiler, Reducer, from a subset of Rust (with a small number of additional annotations) to Oxide. In addition to the features described in §3, our implementation supports **`struct`**s by treating them as tagged tuples or records. Together, Reducer and OxideTC allowed us to use tests from the official borrow checker (`borrowck`) and non-lexical lifetime (`nll`) test suites to validate Oxide against Rust's implementation, rustc. The results of this testing are summarized in Figure 10.

For the 208 passing tests, we can compile the test case into Oxide with Reducer and then use OxideTC to either successfully typecheck the program or to produce a type error. We compare this typechecking result to the expected behavior according to the rustc test suite. All 208 tests either type check when rustc does so, or produce an error corresponding to the error produced by rustc.

The remaining 407 tests were taken out of consideration on the basis of being out-of-scope for this work. There were 20 categories for exclusion, the majority of which had fewer than 10 applicable tests. Figure 10 includes the 6 largest categories: (1) heap allocation, (2) out-of-scope libraries, (3) enumerations, (4) statics and constants, (5) traits, and (6) uninitialized variables. One specialized category (multithreading) was folded into out-of-scope libraries in this table, with miscellaneous aggregating the remaining smaller categories: control flow, casting, first-class constructors, compiler internals dumping, function mutability, inline assembly, macros, slice patterns, two-phase borrows, uninitialized variables, universal function-call syntax, unsafe, and variable mutability.

Combined, heap allocation and out-of-scope libraries (of which the former is a specialization of the latter) make up for the largest excluded category with 103 tests, and can be extended in future work using the strategy outlined by Weiss et al. [2019]. The next largest category, traits, accounts for 93 tests. Though the trait system is in some ways novel, the bulk of its design is rooted in the work on Haskell typeclasses and their extensions. As such, we feel that they are not an *essential* part of Rust, though exploring the particularities of their design may be a fruitful avenue for future work on typeclasses. We are working on extending our implementation with sums to support enumerations, and they are already present in the formalism. Many of the other categories describe features (e.g., macros, control flow, casting, statics, and constants) that are well-studied in the literature, and in which we believe Rust has made relatively standard design choices.

The last issue to discuss involving the tested semantics is the aforementioned annotation burden. This burden comes directly out of the syntactic differences between Oxide and Rust, and so are overall rather minor. The most immediately apparent need is to provide a region annotation on borrow expressions, which we handle using Rust's compiler annotation support. In our tests, a borrow expression like &'a uniq x appears as `#[lft="a"] &mut x`. However, we reduce the need for this by automatically generating a fresh local region for borrow expressions without an annotation. This suffices for the majority of expressions without change. Relatedly, one might also expect to see the introduction of `letrgn` throughout. To alleviate the need for this, our implementation automatically binds free region at the beginning of each function body.

The other main change we had to make relates to the use of explicit environment polymorphism in Oxide. In Rust, every closure has a unique type without a syntax for writing it down. To work with higher-order functions, these closures implement one of three language-defined traits (`Fn`, `FnMut`, and `FnOnce`) which can be used as bounds in higher-order functions. We compile the use of these trait bounds to environment polymorphism in a straight-forward manner (turning instances of the same `Fn`-bound polymorphic type into uses of function types with the same environment variable), but need to introduce a way of writing down which environment to use at instantiation. We use a compiler annotation (`#[envs(c1, ..., cn)]`) on applications which says to instantiate the environment variables with the captured environments of the types of these bindings. If the bindings are unbound or not at a function type, we produce an error indicating as much.

Aside from these two changes, there are a handful of smaller changes that we made by hand to simplify implementation of Reducer and OxideTC, though the need for these could be obviated with more work. Our implementation does not support method call syntax, and so we translate method definitions (which take `self`, `&self`, or `&mut self` as their first argument) into function definitions with a named first argument at the method receiver's type. Relatedly, some of the tests used traits in a trivial way to define methods polymorphic in their receiver type. Like other methods, we translated these into function definitions, but used a polymorphic type for the receiver. rustc also allows for a number of convenient programming patterns (like borrowing from a constant, e.g. `&0`) which are not supported by our implementation. To deal with these cases, we manually introduced temporaries (a process that rustc does automatically). As a simplification for the typechecker, OxideTC only reports the first error that occurs in the program. To ensure that we find a correspondence between all errors, we split up test files with multiple errors into one file per test.

Finally, an earlier version of our implementation required type annotations on all let bindings, and so currently many tests include fully-annotated types. We later realized our typing judgment is very-nearly a type *synthesis* judgment as in bidirectional typechecking, and changed the implementation to support unannotated bindings by using the type synthesized for the expression being bound. This works for all expressions except `abort!` which can produce any type and so requires annotation.

4 RELATED WORK

4.1 Semantics for Rust

Early Work. Reed [2015] developed *Patina*, a formal semantics for an early version of Rust (pre-1.0) focused on proving memory safety for a language with a syntactic version of borrow checking and unique pointers. Unfortunately, the design of the language was not yet stable, and the language overall has drifted from their model. Also, unlike Oxide, Patina made concrete decisions about memory layout and validity which is problematic as Rust itself has not yet made such commitments.

Benitez [2016] developed *Metal*, a formal calculus that, by their characterization, has a Rust-like type system using an *algorithmic* borrow-checking formulation. Their model relies on capabilities as in the Capability Calculus of Crary et al. [1999], but manages them indirectly (compared to the first-class capabilities of Crary et al. [1999] or Morrisett et al. [2007]). Compared to Rust and our work on Oxide, Metal is unable to deal with the proper LIFO ordering for object destruction and their algorithmic formulation is less expressive than our declarative formulation.

RustBelt. In the RustBelt project, Jung et al. [2018a] developed a formal semantics called λ_{Rust} for a continuation-passing style intermediate language in the Rust compiler known as MIR. They mechanized this formal semantics in Iris [Jung et al. 2018b] and used it to verify the extrinsic safety of important Rust standard library abstractions that make extensive use of `unsafe` code. Their goal was distinct from ours in that we instead wish to reason about how programs work at the source-level, and our goals are fortunately complementary. As argued by Weiss et al. [2018], we can incorporate `unsafe` code in the standard library by adding primitives to Oxide, and the verified specifications from RustBelt provide further justification for their safety.

Featherweight Rust. Recent work by Pearce [2021] developed a calculus called FR that, like us, takes inspiration from the Featherweight Java of Igarashi et al. [2001]. Indeed, they take this inspiration so seriously that FR is limited solely to let bindings, assignment, moves, and borrows. Such a simplification misses much of the interesting parts of borrow checking. Without branching, it is possible to statically maintain total knowledge of pointer provenance for every reference, trivializing checking for conflicting borrows. Without aggregate data types like tuples and enumerations, there's no notions of partial ownership and no need for the infrastructure of *places* and *place expressions*. Further, without closures, there is no ability for computation to be suspended with ownership effects caught up in it. Dealing with closures correctly was an immense part of the effort in designing Oxide, and ruled out many simpler borrow checking schemes we developed along the way. Pearce attempts to address this in their work by describing extensions for branching, tuples, and top-level functions with very brief arguments as to why the extension would not break their proofs. However, the answers there are unsatisfying: the argument for branching, for instance, is roughly that one could individually consider each straightline execution path through the program as its own program that then has a precise environment in their calculus. Perhaps most importantly, they limit their attention to modeling Rust with "lexical lifetimes," a language that has not actually existed in *five years* at the time of writing. Like with closures, Oxide required a great deal of careful design work to appropriately handle the behavior of Rust's *non-lexical lifetimes*.

Polonius. Polonius [Matsakis 2018] is a new alias-based implementation of Rust's borrow checker that uses information from the Rust compiler as input facts for a logic program that checks the safety of borrows in a program. Much as we have done with Oxide, Polonius shifts the view of *lifetimes* to a model of *origins* as sets of loans which approximate the possible provenances of a reference. As described by Matsakis [2018], a reference is no longer valid when any of the constituent loans of an origin are invalidated. In Oxide, we take an analogous view: a reference type is valid only when its constituent loans are bound in the stack typing Γ. Though we have not formally explored the connection, based on the commonality between both new views on lifetimes, we feel that Oxide corresponds to a sort of type-systems analogue of Polonius' constraint solving approach.

4.2 Practical Substructural Programming

As a practical programming language with substructural typing, Rust does not exist in a vacuum. There have been numerous efforts in the programming languages community to produce languages that rely on substructurality. Though different in their design from Rust, these languages sit in the same broader design space, finding a balance between usability and expressivity.

Pottier and Protzenko [2013] developed Mezzo, an ML-family language with a static discipline of duplicable and affine permissions to control aliasing and ownership. Similar to Rust, Mezzo is able to have types refer directly to values, rather than always requiring indirection as in work on ownership types [Clarke et al. 1998; Noble et al. 1998]. However, unlike Rust, Mezzo uses a permissions system that works as a sort of type-system formulation of separation logic [Reynolds 2002]. By contrast, Rust relies on a borrow checking analysis to ensure that its guarantees about aliasing and ownership are maintained. In Oxide, we formalized this analysis as the ownership safety judgment which determines if it is safe to use a place uniquely or sharedly in a given context.

Munch-Maccagnoni [2018] has recently proposed a backwards-compatible model of resource management for OCaml. Though not yet a part of OCaml, the proposal is promising and aims to integrate ideas from Rust and C++ (like ownership and so-called "resource acquisition is initialization" [Stroustrup 1994]) with a garbage-collected runtime system for a functional language. Munch-Maccagnoni [2018] argues that these efforts can learn from Rust, and we hope that Oxide provides a strong footing to do so.

Grossman et al. [2002] developed Cyclone as a safe C alternative. To do so, they rely on techniques from region-based memory management [Tofte and Talpin 1994, 1997]. For Cyclone, regions indicate where an object is located in memory (e.g. on the stack or heap), while in Oxide regions are used for managing *aliasing* by abstracting over a reference's possible origins, regardless of the memory mode at runtime. Like Oxide, Fluet et al. [2006] developed a formal semantics to demonstrate the essence of Cyclone.

5 DISCUSSION

Region Reuse in Oxide. Overall, in Oxide, we've seen a number of restrictions related to the concrete region annotations that are added to the source program relative to Rust. This includes the region-not-reborrowed and region-not-in-closure judgments in rules such as T-Borrow and T-Let, as well as in the outlives judgment (Figure 6). Overall, these restrictions may seem to risk limiting our support for Rust's diverse borrowing patterns, but we've found with our implementation that this is not the case. In general, we are able to employ a strategy of always preferring a new region except when required (to pass multiple distinct references to a polymorphic function such as `fn choose_ref<'a>(&'a uniq u32, &'a uniq u32) -> &'a uniq u32`) and indeed, our Oxide implementation can do virtually all of this work automatically. Polonius [Matsakis 2018; Matsakis and Contributors 2020], a new borrow-checker for rustc discussed in §5, relies on a similar scheme of generating new origins and constraining them to be equal only when strictly necessary.

Substructurality in Oxide. Since Rust's release, the folklore has said that, of course, "Rust is an affine language." As such, one might have expected to see the explicit removal of the structural rule of contraction in a formal calculus. However, with behavior like copyable types and implicit drops, the substructurality story for Oxide is a bit more complicated. Like an ordered type system, Oxide does not allow exchange to maintain the ordered end of scopes for bindings, but its rules for variable use (moving, copying, and borrowing) all employ judgments that enable out-of-order *use* of variables. Like an affine type system, Oxide has a rule T-Drop which resembles a weakening rule by allowing a program to typecheck with a binding whenever it is possible to typecheck with that binding *dead*. Unlike conventional weakening, however, the binding itself must still be present (with a dead type) because of the ordering requirement! Finally, Oxide even has something resembling contraction in the form of T-Copy which allows many types to be used multiple times, lowering the friction of the duplicable of-course types common in the substructural typing literature.

6 CONCLUSION AND FUTURE WORK

In this paper, we have presented Oxide as a formal model of *the essence of Rust* with a novel approach for reasoning about the behavior of source-level Rust programs with *region-based alias management*. We leveraged syntactic techniques to prove type safety for Oxide (§3.7), and implemented a prototype typechecker in OCaml along side a compiler from Rust to Oxide which we used to validate our semantics against a suite of over two-hundred tests from the official rustc test suite.

With Oxide in hand, we believe there is a host of new possibilities for research involving Rust. For instance, while there are some early efforts to bring formal verification to Rust [Astrauskas et al. 2018; Baranowski et al. 2018; Toman et al. 2015; Ullrich 2016], the possibilities are limited without an appropriate semantics to work from. As one particular example, the work by Astrauskas et al. [2018] builds verification support for Rust into Viper [Müller et al. 2016], but uses an ad-hoc subset without support for shared references. Further, Rust's memory safety guarantees lend themselves well to security-critical applications. However, the existing compiler toolchain (leveraging LLVM [Lattner and Adve 2004]) does not lend itself well to preserving these kinds of guarantees. As such, another avenue for future work using Oxide would be to build an alternative verified compiler toolchain, perhaps by compilation to Vellvm [Zhao et al. 2012] or CompCert's Clight [Blazy and Leroy 2009]. Overall, we hope that Oxide can serve as a rich platform for research with Rust even beyond our own imaginations.

REFERENCES

Amal Ahmed. 2004. *Semantics of Types for Mutable State*. Ph.D. Dissertation. Princeton University.

Amal Ahmed, Andrew W. Appel, Christopher D. Richards, Kedar N. Swadi, Gang Tan, and Daniel C. Wang. 2010. Semantic Foundations for Typed Assembly Languages. *ACM Transactions on Programming Languages and Systems* 32, 3 (March 2010), 1–67.

Vytautas Astrauskas, Peter Müller, Federico Poli, and Alexander J. Summers. 2018. *Leveraging Rust Types for Modular Specification and Verification*. Technical Report. Eidgenössische Technische Hochschule Zürich.

Henry G. Baker. 1992. Lively Linear Lisp — 'Look Ma, No Garbage!'. *SIGPLAN Notices* (1992).

Henry G. Baker. 1994a. Linear Logic and Permutation Stacks—The Forth Shall Be First. *SIGARCH Computer Architecture News* (1994).

Henry G. Baker. 1994b. Minimizing Reference Count Updating with Deferred Anchored Pointers for Functional Data Structures. *SIGPLAN Notices* (1994).

Henry G. Baker. 1995. 'Use-Once' Variables and Linear Objects — Storage Management, Reflection, and Multi-Threading. *SIGPLAN Notices* (1995).

Marek Baranowski, Shaobo He, and Zvonimir Rakamarić. 2018. Verifying Rust Programs with SMACK. In *Automated Technology for Verification and Analysis*.

Sergio Benitez. 2016. Short Paper: Rusty Types for Solid Safety. In *Workshop on Programming Languages and Analysis for Security*.

Sandrine Blazy and Xavier Leroy. 2009. Mechanized semantics for the Clight subset of the C language. *Journal of Automated Reasoning* 43, 3 (2009).

David G. Clarke, John M. Potter, and James Noble. 1998. Ownership Types for Flexible Alias Protection. In *ACM Symposium on Object Oriented Programming: Systems, Languages, and Applications (OOPSLA)*.

Karl Crary, David Walker, and Greg Morrisett. 1999. Typed Memory Management in a Calculus of Capabilities. In *ACM Symposium on Principles of Programming Languages (POPL), San Antonio, Texas*.

Matthias Felleisen and Robert Hieb. 1992. The Revised Report on the Syntactic Theories of Sequential Control and State. *Theoretical Computer Science* (1992).

Matthew Fluet, Greg Morrisett, and Amal Ahmed. 2006. Linear Regions Are All You Need. In *European Symposium on Programming (ESOP)*.

Jean-Yves Girard. 1987. Linear Logic. *Theoretical Computer Science* (1987).

Dan Grossman, Greg Morrisett, Trevor Jim, Michael Hicks, Yanling Wang, and James Cheney. 2002. Region-Based Memory Management in Cyclone. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Berlin, Germany*.

Unsafe Code Guidelines Working Group. 2019. Unsafe Code Guidelines. https://github.com/rust-rfcs/unsafe-code-guidelines. Accessed: 2019-02-22.

Arjun Guha, Claudiu Saftoiu, and Shriram Krishnamurthi. 2010. The Essence of JavaScript. In *European Conference on Object-Oriented Programming (ECOOP)*.

Atsushi Igarashi, Benjamin C. Pierce, and Philip Wadler. 2001. Featherweight Java: A Minimal Core Calculus for Java and GJ. *ACM Transactions on Programming Languages and Systems* (2001).

Ralf Jung, Hoang-Hai Dang, Jeehoon Kang, and Derek Dreyer. 2019. Stacked Borrows: An Aliasing Model for Rust. *Proc. ACM Program. Lang.* 4, POPL, Article 41 (Dec. 2019), 32 pages. https://doi.org/10.1145/3371109

Ralf Jung, Jacques-Henri Jourdan, Robbert Krebbers, and Derek Dreyer. 2018a. RustBelt: Securing the Foundations of the Rust Programming Language. In *ACM Symposium on Principles of Programming Languages (POPL), Los Angeles, California*.

Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Aleš Bizjak, Lars Birkedal, and Derek Dreyer. 2018b. Iris from the Ground Up: A Modular Foundation for Higher-Order Concurrent Separation Logic. In *Journal of Functional Programming*.

Yves Lafont. 1988. The Linear Abstract Machine. *Theoretical Computer Science* (1988).

Chris Lattner and Vikram Adve. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization* (Palo Alto, California) *(CGO '04)*. IEEE Computer Society, Washington, DC, USA. http://dl.acm.org/citation.cfm?id=977395.977673

Nicholas D. Matsakis. 2016. Non-lexical lifetimes: introduction. http://smallcultfollowing.com/babysteps/blog/2016/04/27/non-lexical-lifetimes-introduction/. Accessed: 2019-02-28.

Nicholas D. Matsakis. 2018. An alias-based formulation of the borrow checker. http://smallcultfollowing.com/babysteps/blog/2018/04/27/an-alias-based-formulation-of-the-borrow-checker/.

Nicholas D. Matsakis and Contributors. 2020. Polonius. https://rust-lang.github.io/polonius/.

Robin Milner. 1978. A Theory of Type Polymorphism in Programming. *J. Comput. System Sci.* (1978).

Naftaly Minsky. 1996. Towards Alias-Free Pointers. In *European Conference on Object-Oriented Programming (ECOOP)*.

Greg Morrisett, Amal Ahmed, and Matthew Fluet. 2007. L3: A Linear Language with Locations. *Fundamenta Informaticae* (2007).

Peter Müller, Malte Schwerhoff, and Alexander J. Summers. 2016. Viper: A Verification Infrastructure for Permission-Based Reasoning. In *Verification, Model Checking, and Abstract Interpretation (VMCAI)*.

Guillaume Munch-Maccagnoni. 2018. Resource Polymorphism. *CoRR* abs/1803.02796 (2018). arXiv:1803.02796 http://arxiv.org/abs/1803.02796

James Noble, Jan Vitek, and John Potter. 1998. Flexible Alias Protection. In *European Conference on Object-Oriented Programming (ECOOP)*.

David J. Pearce. 2021. A Lightweight Formalism for Reference Lifetimes and Borrowing in Rust. *ACM Trans. Program. Lang. Syst.* 43, 1, Article 3 (April 2021), 73 pages. https://doi.org/10.1145/3443420

François Pottier and Jonathan Protzenko. 2013. Programming with Permissions in Mezzo. In *International Conference on Functional Programming (ICFP), Boston, Massachusetts*.

Eric Reed. 2015. *Patina: A formalization of the Rust programming language*. Master's thesis. University of Washington.

John C. Reynolds. 2002. Separation Logic: A Logic for Shared Mutable Data Structures. In *IEEE Symposium on Logic in Computer Science (LICS), Copenhagen, Denmark*.

Bjarne Stroustrup. 1994. *The Design and Evolution of C++*. Addison-Wesley.

Gerald Jay Sussman and Guy Lewis Steele. 1975. *Scheme: An Interpreter for Extended Lambda Calculus*. Technical Report AI Memo No. 349. Massachusetts Institute of Technology, Cambridge, UK.

Mads Tofte and Jean-Pierre Talpin. 1994. Implementation of the Typed Call-by-Value λ-calculus using a Stack of Regions. In *ACM Symposium on Principles of Programming Languages (POPL), Portland, Oregon*.

Mads Tofte and Jean-Pierre Talpin. 1997. Region-Based Memory Management. *Information and Computation* (1997).

John Toman, Stuart Pernsteiner, and Emina Torlak. 2015. CRust: A Bounded Verifier for Rust. In *IEEE/ACM International Conference on Automated Software Engineering*.

Aaron Turon, Konrad Borowski, Hidehito Yabuuchi, and Dan Aloni. 2017. Non-Lexical Lifetimes. https://github.com/rust-lang/rfcs/blob/master/text/2094-nll.md. Accessed: 2019-02-28.

Sebastian Ullrich. 2016. *Simple Verification of Rust Programs via Functional Purification.* Master's thesis. Karlsruhe Institute of Technology.

Philip Wadler. 1991. Is there a use for linear logic?. In *ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation (PEPM).*

David Wakeling and Colin Runciman. 1991. Linearity and Laziness. In *ACM Symposium on Functional Programming Languages and Computer Architecture (FPCA).*

Aaron Weiss, Daniel Patterson, and Amal Ahmed. 2018. Rust Distilled: An Expressive Tower of Languages. *ML Family Workshop* (2018).

Aaron Weiss, Daniel Patterson, Nicholas D. Matsakis, and Amal Ahmed. 2019. Oxide: The Essence of Rust. *arXiv e-prints*, Article arXiv:1903.00982 (Mar 2019), arXiv:1903.00982 pages. arXiv:1903.00982 [cs.PL]

Andrew K. Wright and Matthias Felleisen. 1992. A Syntactic Approach to Type Soundness. *Information and Computation* (1992).

Jianzhou Zhao, Santosh Nagarakatte, Milo M. K. Martin, and Steve Zdancewic. 2012. Formalizing the LLVM Intermediate Representation for Verified Program Transformations. In *ACM Symposium on Principles of Programming Languages (POPL), Philadelphia, Pennsylvania.*

A OXIDE SYNTAX

Variables	x	Functions	f	Type Vars.	α	Frame Vars.	φ
Concrete Regions	r	Abstract Regions	ϱ	Strings	$\texttt{str}$	Naturals	m, n, k

Path	q	$::=$	$\epsilon \mid n.q$
Places	π	$::=$	$x.q$
Place Exprs.	p	$::=$	$x \mid *p \mid p.n$
Place Expr. Contexts	$p^{\square}$	$::=$	$\square \mid *p^{\square} \mid p^{\square}.n$
Regions	ρ	$::=$	$\varrho \mid r$
Ownership Qualifiers	ω	$::=$	$\texttt{shrd} \mid \texttt{uniq}$
Region Rewriting Modes	μ	$::=$	$+ \mid \boxplus \mid =$
Loans	ℓ	$::=$	${}^{\omega}p$
Kinds	κ	$::=$	$\star \mid \mathsf{RGN} \mid \mathsf{FRM}$
Base Types	τ^{B}	$::=$	$\texttt{bool} \mid \texttt{u32} \mid \texttt{unit}$
Sized Types	τ^{SI}	$::=$	$\tau^{\text{B}} \mid \alpha \mid \&\rho\ \omega\ \tau^{\text{XI}} \mid [\tau^{\text{SI}};\ n] \mid (\tau_1^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}}) \mid \texttt{Either<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}\texttt{>}$
		$\mid$	$\forall\texttt{<}\overline{\varphi},\ \overline{\varrho},\ \overline{\alpha}\texttt{>}(\tau_1^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}}) \xrightarrow{\Phi} \tau_r^{\text{SI}}\ \texttt{where}\ \overline{\varrho_1 : \varrho_2}$
Maybe Unsized Types	τ^{XI}	$::=$	$\tau^{\text{SI}} \mid [\tau^{\text{SI}}]$
Dead Types	τ^{SD}	$::=$	${\tau^{\text{SI}}}^{\dagger} \mid (\tau_1^{\text{SD}},\ \ldots,\ \tau_n^{\text{SD}})$
Maybe Dead Types	τ^{SX}	$::=$	$\tau^{\text{SI}} \mid \tau^{\text{SD}} \mid (\tau_1^{\text{SX}},\ \ldots,\ \tau_n^{\text{SX}})$
Types	τ	$::=$	$\tau^{\text{XI}} \mid \tau^{\text{SX}}$
Constants	c	$::=$	$(\,) \mid n \mid \texttt{true} \mid \texttt{false}$
Expressions	e	$::=$	$c \mid p \mid \&r\ \omega\ p \mid \&r\ \omega\ p[e] \mid \&r\ \omega\ p[e_1..e_2] \mid p := e$
		$\mid$	$\texttt{letrgn}\ \texttt{<}r\texttt{>}\ \{\ e\ \} \mid \texttt{let}\ x : \tau^{\text{SI}} = e_1;\ e_2 \mid e_1;\ e_2$
		$\mid$	$\lvert x_1 : \tau_1^{\text{SI}},\ \ldots,\ x_n : \tau_n^{\text{SI}} \rvert \ \rightarrow\ \tau_r^{\text{SI}}\ \{\ e\ \} \mid e_f\texttt{::<}\overline{\Phi},\ \overline{\rho},\ \overline{\tau^{\text{SI}}}\texttt{>}(e_1,\ \ldots,\ e_n)$
		$\mid$	$\texttt{if}\ e_1\ \{\ e_2\ \}\ \texttt{else}\ \{\ e_3\ \} \mid (e_1,\ \ldots,\ e_n) \mid [e_1,\ \ldots,\ e_n]$
		$\mid$	$p[e] \mid \texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \} \mid \texttt{while}\ e_1\ \{\ e_2\ \} \mid \texttt{abort!(str)}$
		$\mid$	$\texttt{Left::<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}\texttt{>}(e) \mid \texttt{Right::<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}\texttt{>}(e)$
		$\mid$	$\texttt{match}\ e\ \{\ \texttt{Left}(x_1) \Rightarrow e_1,\ \texttt{Right}(x_2) \Rightarrow e_2\ \}$
Frame Expressions	Φ	$::=$	$\varphi \mid \mathcal{F}$
Global Environment	Σ	$::=$	$\bullet \mid \Sigma,\ \varepsilon$
Global Entries	ε	$::=$	$\texttt{fn}\ f\texttt{<}\overline{\varphi},\ \overline{\varrho},\ \overline{\alpha}\texttt{>}(x_1 : \tau_1^{\text{SI}},\ \ldots,\ x_n : \tau_n^{\text{SI}})\ \rightarrow\ \tau_r^{\text{SI}}\ \texttt{where}\ \overline{\varrho_1 : \varrho_2}\ \{\ e\ \}$
Type Environment	Δ	$::=$	$\bullet \mid \Delta,\ \alpha : \star \mid \Delta,\ \varrho : \mathsf{RGN} \mid \Delta,\ \varphi : \mathsf{FRM} \mid \Delta,\ \varrho :> \varrho'$
Frame Typing	$\mathcal{F}$	$::=$	$\bullet \mid \mathcal{F},\ x\ :\ \tau^{\text{SX}} \mid \mathcal{F},\ r \mapsto \{\ \overline{\ell}\ \}$
Stack Typing	Γ	$::=$	$\bullet \mid \Gamma \natural \mathcal{F}$
Continuation Typing	Θ	$::=$	$\bullet \mid \Theta,\ \tau^{\text{SI}}$

B STATICS

B.1 Well-Formedness Judgments

$\boxed{\vdash \Sigma}$
read: "Σ is well-formed"

WF-GlobalEnv

$$\frac{\forall \varepsilon \in \Sigma.\ \Sigma \vdash \varepsilon}{\vdash \Sigma}$$

$\boxed{\Sigma \vdash \varepsilon}$
read: "ε is a well-formed function definition in Σ"

WF-FunctionDefinition

$$\frac{\begin{array}{c}\Delta = \overline{\varphi : \mathtt{FRM}},\ \overline{\varrho : \mathtt{RGN}},\ \overline{\varrho_1 :> \varrho_2},\ \overline{\alpha : \star} \qquad \{\,\overline{\varrho_1}\,\} \subseteq \{\,\overline{\varrho}\,\} \qquad \{\,\overline{\varrho_2}\,\} \subseteq \{\,\overline{\varrho}\,\} \\ \Sigma;\ \Delta;\ \bullet \natural\, x_1\ :\ \tau_1^{\text{SI}},\ \ldots,\ x_n\ :\ \tau_n^{\text{SI}};\ \bullet \vdash \boxed{e} : \tau_f^{\text{SI}} \Rightarrow \Gamma' \qquad \Delta;\ \bullet;\ \Theta \vdash^{\mu} \tau_f^{\text{SI}} \rightsquigarrow \tau_r^{\text{SI}} \dashv \bullet\end{array}}{\Sigma \vdash \mathtt{fn}\, f \mathtt{<} \overline{\varphi},\ \overline{\varrho},\ \overline{\alpha} \mathtt{>} (x_1 : \tau_1^{\text{SI}},\ \ldots,\ x_n : \tau_n^{\text{SI}})\ \rightarrow\ \tau_r^{\text{SI}}\ \mathtt{where}\, \overline{\varrho_1 : \varrho_2}\ \{\ e\ \}}$$

$\boxed{\vdash \Delta}$
read: "Δ is well-formed"

WF-TVarEmpty

$$\frac{}{\vdash \bullet}$$

WF-TVarExtendEnv

$$\frac{}{\vdash \Delta,\ \varphi : \mathtt{FRM}}$$

WF-TVarExtendRegion

$$\frac{}{\vdash \Delta,\ \varrho : \mathtt{RGN}}$$

WF-TVarExtendType

$$\frac{}{\vdash \Delta,\ \alpha : \star}$$

WF-TVarExtendOutlives

$$\frac{\varrho_1 : \mathtt{RGN} \in \Delta \qquad \varrho_2 : \mathtt{RGN} \in \Delta}{\vdash \Delta,\ \varrho_1 :> \varrho_2}$$

$\boxed{\Sigma;\ \Delta \vdash \Gamma}$
read: "Γ is well-formed under Σ and Δ"

WF-EmptyStackTyping

$$\frac{}{\Sigma;\ \Delta \vdash \bullet}$$

WF-StackTyping

$$\frac{\begin{array}{c}\Sigma;\ \Delta \vdash \Gamma \qquad \mathtt{places}(\mathcal{F}) \subseteq \mathrm{dom}(\Gamma \natural \mathcal{F}) \\ \mathrm{dom}(\mathcal{F}) \# \mathrm{dom}(\Gamma) \qquad \forall x\ :\ \tau \in \mathcal{F}.\ \Sigma;\ \Delta;\ \Gamma \natural \mathcal{F} \vdash \tau \\ \forall \tau \in \mathrm{cod}(\mathcal{F}).\ \forall r \in \text{free-regions}(\tau).\ \forall \tau' \in \mathrm{dom}(\Gamma).\ r \text{ does not occur outside of a closure in } \tau' \\ \forall r \mapsto \{\,\overline{\ell}\,\} \in \mathcal{F}.\ \forall^{\omega} p\ \in \{\,\overline{\ell}\,\}.\ \exists \tau^{\text{XI}}.\ \Delta;\ \Gamma \natural \mathcal{F} \vdash_{\omega} p : \tau^{\text{XI}}\end{array}}{\Sigma;\ \Delta \vdash \Gamma \natural \mathcal{F}}$$

$\boxed{\Sigma;\ \Delta;\ \Gamma \vdash \Theta}$

read: "Θ is well-formed under Σ, Δ, and Γ"

WF-EmptyTemporaryTyping
$$\frac{}{\Sigma;\ \Delta;\ \Gamma \vdash \bullet}$$

WF-TemporaryTyping
$$\frac{\Sigma;\ \Delta;\ \Gamma \vdash \Theta \qquad \Sigma;\ \Delta;\ \Gamma \vdash \tau \qquad \forall r \in \text{free-regions}(\tau^{\text{SI}}).\ \nexists \tau_b^{\text{SI}} \in \text{cod}(\Gamma).\ r \in \text{free-regions}(\tau_b^{\text{SI}}) \wedge \Gamma(r) = \emptyset}{\Sigma;\ \Delta;\ \Gamma \vdash \Theta,\ \tau^{\text{SI}}}$$

$\boxed{\vdash \Sigma;\ \Delta;\ \Gamma;\ \Theta}$

read: "Σ, Δ, and Γ are well-formed."

WF-Environments
$$\frac{\vdash \Sigma \qquad \vdash \Delta \qquad \Sigma;\ \Delta \vdash \Gamma \qquad \Sigma;\ \Delta;\ \Gamma \vdash \Theta}{\vdash \Sigma;\ \Delta;\ \Gamma;\ \Theta}$$

$\boxed{\Sigma;\ \Delta;\ \Gamma \vdash \Phi}$

read: "Φ is a well-formed captured environment"

WF-EnvVar
$$\frac{\Delta(\varphi) = \texttt{FRM}}{\Sigma;\ \Delta;\ \Gamma \vdash \varphi}$$

WF-Env
$$\frac{\Sigma;\ \Delta \vdash \Gamma \natural \mathcal{F}_c}{\Sigma;\ \Delta;\ \Gamma \vdash \mathcal{F}_c}$$

$\boxed{\Delta;\ \Gamma \vdash \rho}$

read: "ρ is a well-formed region"

WF-ConcreteRegion
$$\frac{r \in \text{dom}(\Gamma)}{\Delta;\ \Gamma \vdash r}$$

WF-AbstractRegion
$$\frac{\Delta(\varrho) = \texttt{RGN}}{\Delta;\ \Gamma \vdash \varrho}$$

$\boxed{\Sigma;\ \Delta;\ \Gamma \vdash \tau}$

read: "τ is a well-formed type under Σ, Δ, and Γ"

WF-BaseType
$$\frac{}{\Sigma;\ \Delta;\ \Gamma \vdash \tau^{\text{B}}}$$

WF-TVar
$$\frac{\Delta(\alpha) = \star}{\Sigma;\ \Delta;\ \Gamma \vdash \alpha}$$

WF-Ref
$$\frac{\Delta;\ \Gamma \vdash \rho \qquad \Sigma;\ \Delta;\ \Gamma \vdash \tau^{\text{XI}}}{\Sigma;\ \Delta;\ \Gamma \vdash \&\rho\ \omega\ \tau^{\text{XI}}}$$

WF-Tuple
$$\frac{\forall i \in \{\ 1\ \ldots\ n\ \}.\ \Sigma;\ \Delta;\ \Gamma \vdash \tau_i^{\text{SX}}}{\Sigma;\ \Delta;\ \Gamma \vdash (\tau_1^{\text{SX}},\ \ldots,\ \tau_n^{\text{SX}})}$$

WF-Function
$$\frac{\begin{array}{c}\forall r \in \text{free-regions}(\tau_r^{\text{SI}}).\ \forall \tau' \in \text{dom}(\Gamma).\ r \text{ does not occur outside of a closure in } \tau' \\ \Sigma;\ \Delta;\ \Gamma \vdash \Phi \qquad \Sigma;\ \Delta,\ \overline{\varphi : \texttt{FRM}},\ \overline{\varrho : \texttt{RGN}},\ \overline{\varrho_1 :> \varrho_2},\ \overline{\alpha : \star};\ \Gamma \vdash \tau_r^{\text{SI}} \\ \forall i \in \{\ 1\ \ldots\ n\ \}.\ \Sigma;\ \Delta,\ \overline{\varphi : \texttt{FRM}},\ \overline{\varrho : \texttt{RGN}},\ \overline{\varrho_1 :> \varrho_2},\ \overline{\alpha : \star};\ \Gamma \vdash \tau_i^{\text{SI}}\end{array}}{\Sigma;\ \Delta;\ \Gamma \vdash \forall\!<\!\overline{\varphi},\ \overline{\varrho},\ \overline{\alpha}\!>\!(\tau_1^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}}) \xrightarrow{\Phi} \tau_r^{\text{SI}} \texttt{ where } \overline{\varrho_1 : \varrho_2}}$$

WF-Uninit
$$\frac{}{\Sigma;\ \Delta;\ \Gamma \vdash {\tau^{\text{SI}}}^{\dagger}}$$

WF-Array
$$\frac{\Sigma;\ \Delta;\ \Gamma \vdash \tau^{\text{SI}}}{\Sigma;\ \Delta;\ \Gamma \vdash [\tau^{\text{SI}};\ n]}$$

WF-Slice
$$\frac{\Sigma;\ \Delta;\ \Gamma \vdash \tau^{\text{SI}}}{\Sigma;\ \Delta;\ \Gamma \vdash [\tau^{\text{SI}}]}$$

B.2 Region Rewriting & Outlives Relations

$$\boxed{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma'}$$

read: "terms at the type τ_1 under Δ and Γ can be rewritten according to μ as type τ_2 under Γ'"

$$\textsc{RR-Refl}\quad \frac{}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_1 \dashv \Gamma}$$

$$\textsc{RR-Trans}\quad \frac{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma' \qquad \Delta;\ \Gamma';\ \Theta \vdash^{\mu} \tau_2 \rightsquigarrow \tau_3 \dashv \Gamma''}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_3 \dashv \Gamma''}$$

$$\textsc{RR-Array}\quad \frac{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma'}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} [\tau_1;\ n] \rightsquigarrow [\tau_2;\ n] \dashv \Gamma'}$$

$$\textsc{RR-Slice}\quad \frac{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma'}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} [\tau_1] \rightsquigarrow [\tau_2] \dashv \Gamma'}$$

$$\textsc{RR-Reference}\quad \frac{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \rho_1 :> \rho_2 \dashv \Gamma' \qquad \Delta;\ \Gamma';\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma''}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \&\rho_1\ \omega\ \tau_1 \rightsquigarrow \&\rho_2\ \omega\ \tau_2 \dashv \Gamma''}$$

$$\textsc{RR-Tuple}\quad \frac{\forall i \in \{\, 1 \ldots n \,\}.\ \Delta;\ \Gamma_{n-1};\ \Theta \vdash^{\mu} \tau_i \rightsquigarrow \tau_i' \dashv \Gamma_i}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} (\tau_1\ \ldots\ \tau_n) \rightsquigarrow (\tau_1'\ \ldots\ \tau_n') \dashv \Gamma_n}$$

$$\textsc{RR-Dead}\quad \frac{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1^{\text{SI}} \rightsquigarrow \tau_2^{\text{SI}} \dashv \Gamma'}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1^{\text{SI}} \rightsquigarrow \tau_2^{\text{SI}^{\dagger}} \dashv \Gamma}$$

$$\boxed{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \rho_1 :> \rho_2 \dashv \Gamma'}$$

read: "ρ_1 outlives ρ_2 under Δ and Γ, and can be rewritten according to μ under the environment Γ'"

$$\textsc{OL-Refl}\quad \frac{}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \rho :> \rho \dashv \Gamma}$$

$$\textsc{OL-BothAbstract}\quad \frac{\varrho_1 : \texttt{RGN} \in \Delta \qquad \varrho_2 : \texttt{RGN} \in \Delta \qquad \varrho_1 :> \varrho_2 \in \Delta}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \varrho_1 :> \varrho_2 \dashv \Gamma}$$

$$\textsc{OL-Trans}\quad \frac{\begin{array}{c}\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \varrho_1 :> \varrho_2 \dashv \Gamma' \\ \Delta;\ \Gamma';\ \Theta \vdash^{\mu} \varrho_2 :> \varrho_3 \dashv \Gamma''\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \varrho_1 :> \varrho_3 \dashv \Gamma''}$$

$$\textsc{OL-CombineConcrete}\quad \frac{\begin{array}{ccc}\Gamma \vdash r_1\ \texttt{rnrb} & \Gamma \vdash r_2\ \texttt{rnrb} & \Gamma;\ \Theta \vdash \{\, r_1,\ r_2 \,\}\ \texttt{clrs} \\ \multicolumn{2}{c}{r_1\ \texttt{occurs before}\ r_2\ \texttt{in}\ \Gamma} & \{\, \overline{\ell} \,\} = \Gamma(r_1) \cup \Gamma(r_2)\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash^{+} r_1 :> r_2 \dashv \Gamma[r_2 \mapsto \{\, \overline{\ell} \,\}]}$$

$$\textsc{OL-CombineConcreteUnrestricted}\quad \frac{\begin{array}{c}\Gamma \vdash r_1\ \texttt{rnrb} \qquad \Gamma \vdash r_2\ \texttt{rnrb} \\ r_1\ \texttt{occurs before}\ r_2\ \texttt{in}\ \Gamma \qquad \{\, \overline{\ell} \,\} = \Gamma(r_1) \cup \Gamma(r_2)\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash^{\boxplus} r_1 :> r_2 \dashv \Gamma[r_2 \mapsto \{\, \overline{\ell} \,\}]}$$

$$\textsc{OL-CheckConcrete}\quad \frac{\begin{array}{c}\Gamma \vdash r_1\ \texttt{rnrb} \qquad \Gamma \vdash r_2\ \texttt{rnrb} \\ r_1\ \texttt{occurs before}\ r_2\ \texttt{in}\ \Gamma\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash^{=} r_1 :> r_2 \dashv \Gamma}$$

$$\textsc{OL-AbstractConcrete}\quad \frac{\varrho : \texttt{RGN} \in \Delta \qquad r \in \text{dom}(\Gamma)}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \varrho :> r \dashv \Gamma}$$

$$\textsc{OL-ConcreteAbstract}\quad \frac{\begin{array}{c}\Gamma_{1,0}(r) = \{\, \overline{{}^{\omega}p}^{\,n} \,\} \neq \emptyset \qquad \forall i \in \{\, 1 \ldots n \,\}.\ \nexists \pi.\ p_i = \pi \qquad \forall i \in \{\, 1 \ldots n \,\}.\ \Delta;\ \Gamma_0 \vdash_{\texttt{shrd}} p_i : \_,\ \overline{\rho_i}^{\,m_i} \\ \varrho : \texttt{RGN} \in \Delta \qquad \forall i \in \{\, 1 \ldots n \,\}. \forall j \in \{\, 1 \ldots m_i \,\}.\ \Delta;\ \Gamma_{i,j-1};\ \Theta \vdash^{\mu} \rho_{i,j} :> \varrho \dashv \Gamma_{i,j}\end{array}}{\Delta;\ \Gamma_{1,0};\ \Theta \vdash^{\mu} r :> \varrho \dashv \Gamma_{n,m_n}}$$

$$\boxed{\Delta;\ \Gamma;\ \Theta \vdash \overline{\rho_1 :> \rho_2} \dashv \Gamma'}$$

$$\textsc{OL-Bounds}\quad \frac{\forall i \in \{\, 1 \ldots n \,\}.\ \Delta;\ \Gamma_{i-1};\ \Theta \vdash^{\mu} \rho_i :> \rho_i' \dashv \Gamma_i}{\Delta;\ \Gamma_0;\ \Theta \vdash \overline{\rho :> \rho'} \dashv \Gamma_n}$$

B.3 Ownership Safety

$\boxed{\Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi}} p \Rightarrow \{\ \overline{{}^{\omega}p'}\ \}}$ where $\Delta;\ \Gamma;\ \Theta \vdash_{\omega} p \Rightarrow \{\ \overline{{}^{\omega}p'}\ \}$ means $\Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\bullet} p \Rightarrow \{\ \overline{{}^{\omega}p}\ \}$.

read: "p is ω-safe under Δ and Γ, with reborrow exclusion list $\overline{\pi}$, and may point to any of the loans in $\overline{{}^{\omega}p}$"

O-SafePlace

$$\frac{\begin{array}{c}\forall r' \mapsto \{\ \overline{\ell}\ \} \in \text{regions}(\Gamma,\ \Theta).\ (\forall\, {}^{\omega'}p^{\square}[\pi']\ \in \{\ \overline{\ell}\ \}.(\omega = \mathsf{uniq} \vee \omega' = \mathsf{uniq}) \implies \pi' \# \pi) \\ \vee\ (\exists \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \wedge \nexists \&r'\ \omega'\ \tau' \in \Theta\ \wedge (\forall \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\ \overline{\pi_e}\ \}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi_e}} \pi \Rightarrow \{\ {}^{\omega}\pi\ \}}$$

O-Deref

$$\frac{\begin{array}{c}\Gamma(\pi) = \&r\ \omega_{\pi}\ \tau_{\pi} \qquad \Gamma(r) = \{\ \overline{{}^{\omega'}p}^{n}\ \} \qquad \text{excl} = \{\ \pi_j \text{ where } j \in \{\ 1,\ \dots\ n\ \} \mid p_j = p_j^{\square}[*\pi_j]\ \} \qquad \omega \lesssim \omega_{\pi} \\ \forall i \in \{\ 1\ \dots\ n\ \}.\ \Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi_e},\,\text{excl},\,\pi} p^{\square}[p_i] \Rightarrow \{\ \overline{{}^{\omega}p_i'}\ \} \\ \forall r' \mapsto \{\ \overline{\ell}\ \} \in \text{regions}(\Gamma,\ \Theta).\ (\forall\, {}^{\omega'}p''\ \in \{\ \overline{\ell}\ \}.(\omega = \mathsf{uniq} \vee \omega' = \mathsf{uniq}) \implies p'' \# p^{\square}[*\pi]) \\ \vee\ (\exists \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \wedge \nexists \&r'\ \omega'\ \tau' \in \Theta\ \wedge (\forall \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\ \overline{\pi_e},\ \text{excl},\ \pi\ \}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi_e}} p^{\square}[*\pi] \Rightarrow \{\ \overline{{}^{\omega}p_1'},\ \dots\ \overline{{}^{\omega}p_n'},\ {}^{\omega}p^{\square}[*\pi]\ \}}$$

O-DerefAbs

$$\frac{\begin{array}{c}\Gamma(\pi) = \&\varrho\ \omega_{\pi}\ \tau_{\pi} \qquad \Delta;\ \Gamma \vdash_{\omega} p^{\square}[*\pi] : \tau \qquad \omega \lesssim \omega_{\pi} \\ \forall r' \mapsto \{\ \overline{\ell}\ \} \in \text{regions}(\Gamma,\ \Theta).\ (\forall\, {}^{\omega'}p\ \in \{\ \overline{\ell}\ \}.(\omega = \mathsf{uniq} \vee \omega' = \mathsf{uniq}) \implies p \# p^{\square}[*\pi]) \\ \vee\ (\exists \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \wedge \nexists \&r'\ \omega'\ \tau' \in \Theta\ \wedge (\forall \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\ \overline{\pi_e},\ \pi\ \}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi_e}} p^{\square}[*\pi] \Rightarrow \{\ {}^{\omega}p^{\square}[*\pi]\ \}}$$

B.4 Typing

$\boxed{\Sigma; \Delta; \Gamma; \Theta \vdash e : \tau \Rightarrow \Gamma'}$ where $\vdash \Sigma; \Delta; \Gamma; \Theta$ and $\Sigma; \Delta; \Gamma' \vdash \tau$
read: "e has type τ under Σ, Δ, and Γ, producing output context Γ'"

T-Move

$$\dfrac{\begin{array}{c}\Delta; \Gamma; \Theta \vdash_{\mathtt{uniq}} \pi \Rightarrow \{\ {}^{\mathtt{uniq}}\pi\ \} \\ \Gamma(\pi) = \tau^{\mathrm{SI}} \qquad \mathtt{noncopyable}_{\Sigma}\ \tau^{\mathrm{SI}}\end{array}}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{\pi} : \tau^{\mathrm{SI}} \Rightarrow \Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}]}$$

T-Copy

$$\dfrac{\begin{array}{c}\Delta; \Gamma; \Theta \vdash_{\mathtt{shrd}} p \Rightarrow \{\ \overline{\ell}\ \} \\ \Delta; \Gamma \vdash_{\mathtt{shrd}} p : \tau^{\mathrm{SI}} \qquad \mathtt{copyable}_{\Sigma}\ \tau^{\mathrm{SI}}\end{array}}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{p} : \tau^{\mathrm{SI}} \Rightarrow \Gamma}$$

T-Borrow

$$\dfrac{\begin{array}{c}\Gamma(r) = \emptyset \qquad \Gamma; \Theta \vdash r\ \mathtt{rnic} \\ \Delta; \Gamma; \Theta \vdash_{\omega} p \Rightarrow \{\ \overline{\ell}\ \} \qquad \Delta; \Gamma \vdash_{\omega} p : \tau^{\mathrm{XI}}\end{array}}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{\&r\ \omega\ p} : \&r\ \omega\ \tau^{\mathrm{XI}} \Rightarrow \Gamma[r \mapsto \{\ \overline{\ell}\ \}]}$$

T-BorrowIndex

$$\dfrac{\begin{array}{c}\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{e} : \mathtt{u32} \Rightarrow \Gamma' \qquad \Gamma'(r) = \emptyset \qquad \Gamma'; \Theta \vdash r\ \mathtt{rnic} \\ \Delta; \Gamma'; \Theta \vdash_{\omega} p \Rightarrow \{\ \overline{\ell}\ \} \qquad \Delta; \Gamma' \vdash_{\omega} p : \tau^{\mathrm{XI}} \\ \tau^{\mathrm{XI}} = [\tau^{\mathrm{SI}};\ n] \lor \tau^{\mathrm{XI}} = [\tau^{\mathrm{SI}}]\end{array}}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{\&r\ \omega\ p[e]} : \&r\ \omega\ \tau^{\mathrm{SI}} \Rightarrow \Gamma'[r \mapsto \{\ \overline{\ell}\ \}]}$$

T-BorrowSlice

$$\dfrac{\begin{array}{c}\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{e_1} : \mathtt{u32} \Rightarrow \Gamma_1 \qquad \Sigma; \Delta; \Gamma_1; \Theta \vdash \boxed{e_2} : \mathtt{u32} \Rightarrow \Gamma_2 \\ \Gamma_2(r) = \emptyset \qquad \Gamma_2; \Theta \vdash r\ \mathtt{rnic} \qquad \Delta; \Gamma_2; \Theta \vdash_{\omega} p \Rightarrow \{\ \overline{\ell}\ \} \qquad \Delta; \Gamma_2 \vdash_{\omega} p : [\tau^{\mathrm{SI}}]\end{array}}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{\&r\ \omega\ p[e_1..e_2]} : \&r\ \omega\ [\tau^{\mathrm{SI}}] \Rightarrow \Gamma_2[r \mapsto \{\ \overline{\ell}\ \}]}$$

T-IndexCopy

$$\dfrac{\begin{array}{c}\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{e} : \mathtt{u32} \Rightarrow \Gamma' \qquad \Delta; \Gamma'; \Theta \vdash_{\mathtt{shrd}} p \Rightarrow \{\ \overline{\ell}\ \} \\ \Delta; \Gamma' \vdash_{\mathtt{shrd}} p : \tau^{\mathrm{XI}} \qquad \tau^{\mathrm{XI}} = [\tau^{\mathrm{SI}};\ n] \lor \tau^{\mathrm{XI}} = [\tau^{\mathrm{SI}}] \qquad \mathtt{copyable}_{\Sigma}\ \tau^{\mathrm{SI}}\end{array}}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{p[e]} : \tau^{\mathrm{SI}} \Rightarrow \Gamma'}$$

T-Seq

$$\dfrac{\begin{array}{c}\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{e_1} : \tau_1^{\mathrm{SI}} \Rightarrow \Gamma_1 \\ \Sigma; \Delta; \text{gc-loans}_{\Theta}(\Gamma_1); \Theta \vdash \boxed{e_2} : \tau_2^{\mathrm{SI}} \Rightarrow \Gamma_2\end{array}}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{e_1;\ e_2} : \tau_2^{\mathrm{SI}} \Rightarrow \Gamma_2}$$

T-Branch

$$\dfrac{\begin{array}{c}\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{e_1} : \mathtt{bool} \Rightarrow \Gamma_1 \qquad \Sigma; \Delta; \Gamma_1; \Theta \vdash \boxed{e_2} : \tau_2^{\mathrm{SI}} \Rightarrow \Gamma_2 \\ \Sigma; \Delta; \Gamma_1; \Theta \vdash \boxed{e_3} : \tau_3^{\mathrm{SI}} \Rightarrow \Gamma_3 \qquad \tau^{\mathrm{SI}} = \tau_2^{\mathrm{SI}} \lor \tau^{\mathrm{SI}} = \tau_3^{\mathrm{SI}} \\ \Delta;\ \Gamma_2;\ \Theta \vdash^{+} \tau_2^{\mathrm{SI}} \rightsquigarrow \tau^{\mathrm{SI}} \dashv \Gamma_2' \qquad \Delta;\ \Gamma_3;\ \Theta \vdash^{+} \tau_3^{\mathrm{SI}} \rightsquigarrow \tau^{\mathrm{SI}} \dashv \Gamma_3' \qquad \Gamma_2' \Cup \Gamma_3' = \Gamma'\end{array}}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{\mathtt{if}\ e_1\ \{\ e_2\ \}\ \mathtt{else}\ \{\ e_3\ \}} : \tau^{\mathrm{SI}} \Rightarrow \Gamma'}$$

T-Let

$$\dfrac{\begin{array}{c}\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{e_1} : \tau_1^{\mathrm{SI}} \Rightarrow \Gamma_1 \qquad \Delta;\ \Gamma_1;\ \Theta \vdash^{+} \tau_1^{\mathrm{SI}} \rightsquigarrow \tau_a^{\mathrm{SI}} \dashv \Gamma_1' \\ \forall r \in \text{free-regions}(\tau_a^{\mathrm{SI}}).\ \Gamma_1' \vdash r\ \mathtt{rnrb} \qquad \Sigma; \Delta; \text{gc-loans}_{\Theta}(\Gamma_1',\ x\ :\ \tau_a^{\mathrm{SI}}); \Theta \vdash \boxed{e_2} : \tau_2^{\mathrm{SI}} \Rightarrow \Gamma_2,\ x\ :\ \tau^{\mathrm{SD}}\end{array}}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{\mathtt{let}\ x : \tau_a^{\mathrm{SI}} = e_1;\ e_2} : \tau_2^{\mathrm{SI}} \Rightarrow \Gamma_2}$$

T-LetRegion

$$\dfrac{\Sigma; \Delta; \Gamma,\ r \mapsto \{\}; \Theta \vdash \boxed{e} : \tau^{\mathrm{SI}} \Rightarrow \Gamma',\ r \mapsto \{\overline{\ell}\}}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{\mathtt{letrgn}\ \mathtt{<}r\mathtt{>}\ \{\ e\ \}} : \tau^{\mathrm{SI}} \Rightarrow \Gamma'}$$

T-AssignDeref

$$\dfrac{\begin{array}{c}\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{e} : \tau_n^{\mathrm{SI}} \Rightarrow \Gamma_1 \qquad \Delta;\ \Gamma_1 \vdash_{\mathtt{uniq}} p : \tau_o^{\mathrm{SI}} \\ \Delta;\ \Gamma_1;\ \Theta \vdash^{+} \tau_n^{\mathrm{SI}} \rightsquigarrow \tau_o^{\mathrm{SI}} \dashv \Gamma' \qquad \Delta; \Gamma'; \Theta \vdash_{\mathtt{uniq}} p \Rightarrow \{\ \overline{\ell}\ \}\end{array}}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{p \mathrel{:=} e} : \mathtt{unit} \Rightarrow \Gamma'}$$

T-Assign

$$\dfrac{\begin{array}{c}\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{e} : \tau^{\mathrm{SI}} \Rightarrow \Gamma_1 \qquad \Gamma_1(\pi) = \tau^{\mathrm{SX}} \qquad \tau^{\mathrm{SX}} = \&r\ \omega\ \tau^{\mathrm{XI}} \implies r \text{ is unique to } \pi \text{ in } \Gamma_1 \\ \Delta;\ \Gamma_1 \unrhd *\pi;\ \Theta \vdash^{=} \tau^{\mathrm{SI}} \rightsquigarrow \tau^{\mathrm{SX}} \dashv \Gamma' \qquad (\tau^{\mathrm{SX}} = \tau^{\mathrm{SD}} \lor \Delta; \Gamma'; \Theta \vdash_{\mathtt{uniq}} \pi \Rightarrow \{\ {}^{\mathtt{uniq}}\pi\ \})\end{array}}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{\pi \mathrel{:=} e} : \mathtt{unit} \Rightarrow \Gamma'[\pi \mapsto \tau^{\mathrm{SI}}]}$$

T-While

$$\frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1} : \texttt{bool} \Rightarrow \Gamma_1 \qquad \Sigma;\ \Delta;\ \Gamma_1;\ \Theta \vdash \boxed{e_2} : \texttt{unit} \Rightarrow \Gamma_2 \\ \Sigma;\ \Delta;\ \Gamma_2;\ \Theta \vdash \boxed{e_1} : \texttt{bool} \Rightarrow \Gamma_2 \qquad \Sigma;\ \Delta;\ \Gamma_2;\ \Theta \vdash \boxed{e_2} : \texttt{unit} \Rightarrow \Gamma_2\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{while}\ e_1\ \{\ e_2\ \}} : \texttt{unit} \Rightarrow \Gamma_2}$$

T-ForArray

$$\frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1} : [\tau^{\text{SI}};\ n] \Rightarrow \Gamma_1 \qquad \forall r \in \text{free-regions}(\tau^{\text{SI}}).\ \Gamma_1 \vdash r\ \texttt{rnrb} \\ \Sigma;\ \Delta;\ \Gamma_1\,,\ x\ :\ \tau^{\text{SI}};\ \Theta \vdash \boxed{e_2} : \texttt{unit} \Rightarrow \Gamma_1\,,\ x\ :\ \tau^{\text{SD}}\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \}} : \texttt{unit} \Rightarrow \Gamma_1}$$

T-ForSlice

$$\frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1} : \&\rho\ \omega\ [\tau^{\text{SI}}] \Rightarrow \Gamma_1 \qquad \forall r \in \text{free-regions}(\&\rho\ \omega\ \tau^{\text{SI}}).\ \Gamma_1 \vdash r\ \texttt{rnrb} \\ \Sigma;\ \Delta;\ \Gamma_1\,,\ x\ :\ \&\rho\ \omega\ \tau^{\text{SI}};\ \Theta \vdash \boxed{e_2} : \texttt{unit} \Rightarrow \Gamma_1\,,\ x\ :\ \tau_1^{\text{SX}}\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \}} : \texttt{unit} \Rightarrow \Gamma_2}$$

T-Function

$$\frac{\Sigma(f) = \texttt{fn}\ f\texttt{<}\overline{\varphi},\ \overline{\varrho},\ \overline{\alpha}\texttt{>}(x_1 : \tau_1^{\text{SI}},\ \ldots,\ x_n : \tau_n^{\text{SI}})\ \rightarrow\ \tau_r^{\text{SI}}\ \texttt{where}\ \overline{\varrho_1 : \varrho_2}\ \{\ e\ \}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{f} : \forall\texttt{<}\overline{\varphi},\ \overline{\varrho},\ \overline{\alpha}\texttt{>}(\tau_1^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}})\ \rightarrow\ \tau_r^{\text{SI}}\ \texttt{where}\ \overline{\varrho_1 : \varrho_2} \Rightarrow \Gamma}$$

T-Closure

$$\frac{\begin{array}{c}\text{free-vars}(e) \setminus \overline{x} = \overline{x_f} \qquad \text{free-nc-vars}_\Gamma(e) \setminus \overline{x} = \overline{x_{nc}} \\ \overline{r} = \overline{\text{free-regions}(\Gamma(x_f))},\ (\text{free-regions}(e) \setminus (\overline{\text{free-regions}(\tau^{\text{SI}})}, \text{free-regions}(\tau_r^{\text{SI}}))) \\ \mathcal{F}_c = \overline{r \mapsto \Gamma(r)},\ \overline{x_f\ :\ \Gamma(x_f)} \qquad \forall r_p \in \bigcup_{i=1}^{n} \text{free-regions}(\tau_i^{\text{SI}}) \cup \text{free-regions}(\tau_r^{\text{SI}}).\ \Gamma(r_p) = \emptyset \\ \Sigma;\ \Delta;\ \Gamma[\overline{x_{nc} \mapsto \Gamma(x_{nc})^{\dagger}}] \natural \mathcal{F}_c\,,\ x_1\ :\ \tau_1^{\text{SI}},\ \ldots,\ x_n\ :\ \tau_n^{\text{SI}};\ \Theta \vdash \boxed{e} : \tau_r^{\text{SI}} \Rightarrow \Gamma' \natural \mathcal{F}\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{|x_1 : \tau_1^{\text{SI}},\ \ldots,\ x_n : \tau_n^{\text{SI}}|\ \rightarrow\ \tau_r^{\text{SI}}\ \{\ e\ \}} : (\tau_1^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}})\ \xrightarrow{\mathcal{F}_c}\ \tau_r^{\text{SI}} \Rightarrow \Gamma'}$$

T-AppFunction

$$\frac{\begin{array}{c}\overline{\Sigma;\ \Delta;\ \Gamma \vdash \Phi} \qquad \overline{\Delta;\ \Gamma \vdash \rho} \qquad \overline{\Sigma;\ \Delta;\ \Gamma \vdash \tau^{\text{SI}}} \qquad \delta = \cdot\ \overline{[{}^{\Phi}/_{\varphi}]}\ \overline{[{}^{\rho}/_{\varrho}]}\ \overline{[{}^{\tau^{\text{SI}}}/_{\alpha}]} \\ \Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_f} : \forall\texttt{<}\overline{\varphi},\ \overline{\varrho},\ \overline{\alpha}\texttt{>}(\tau_1^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}})\ \rightarrow\ \tau_f^{\text{SI}}\ \texttt{where}\ \overline{\varrho_1 : \varrho_2} \Rightarrow \Gamma_0 \\ \forall i \in \{\ 1\ \ldots\ n\ \}.\ \Sigma;\ \Delta;\ \Gamma_{i-1};\ \Theta,\ \delta(\tau_1^{\text{SI}})\ \ldots\ \delta(\tau_{i-1}^{\text{SI}}) \vdash \boxed{e_i} : \delta(\tau_i^{\text{SI}}) \Rightarrow \Gamma_i \\ \forall i \in \{\ 1\ \ldots\ n\ \}.\ \forall r \in \text{free-regions}(\tau_i^{\text{SI}}).\ \Gamma_n' \vdash r\ \texttt{rnrb} \qquad \overline{\Delta;\ \Gamma_n;\ \Theta \vdash \varrho_2\ \overline{[{}^{\rho}/_{\varrho}]} \mathrel{:>} \varrho_1\ \overline{[{}^{\rho}/_{\varrho}]} \dashv \Gamma_b}\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_f\texttt{::<}\overline{\Phi},\ \overline{\rho},\ \overline{\tau^{\text{SI}}}\texttt{>}(e_1,\ \ldots,\ e_n)} : \delta(\tau_f^{\text{SI}}) \Rightarrow \Gamma_b}$$

T-AppClosure

$$\frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_f} : \forall\texttt{<>}(\tau_1^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}})\ \xrightarrow{\Phi_c}\ \tau_f^{\text{SI}} \Rightarrow \Gamma_0 \\ \forall i \in \{\ 1\ \ldots\ n\ \}.\ \Sigma;\ \Delta;\ \Gamma_{i-1};\ \Theta,\ \tau_1^{\text{SI}}\ \ldots\ \tau_{i-1}^{\text{SI}} \vdash \boxed{e_i} : \tau_{i'}^{\text{SI}} \Rightarrow \Gamma_i \qquad \Delta;\ \Gamma_i;\ \Theta \vdash^{\boxplus} \tau_{i'}^{\text{SI}} \rightsquigarrow \tau_i^{\text{SI}} \dashv \Gamma_i' \\ \forall i \in \{\ 1\ \ldots\ n\ \}.\ \forall r \in \text{free-regions}(\tau_i^{\text{SI}}).\ \Gamma_n' \vdash r\ \texttt{rnrb}\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_f(e_1,\ \ldots,\ e_n)} : \tau_f^{\text{SI}} \Rightarrow \Gamma_n'}$$

T-Abort

$$\frac{}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{abort!(str)}} : \tau^{\text{SX}} \Rightarrow \Gamma}$$

T-Unit

$$\frac{}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{()} : \texttt{unit} \Rightarrow \Gamma}$$

T-u32

$$\frac{}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{n} : \texttt{u32} \Rightarrow \Gamma}$$

T-True

$$\frac{}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{true}} : \texttt{bool} \Rightarrow \Gamma}$$

T-False

$$\frac{}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{false}} : \texttt{bool} \Rightarrow \Gamma}$$

T-Tuple

$$\frac{\forall i \in \{\ 1\ \ldots\ n\ \}.\ \Sigma;\ \Delta;\ \Gamma_{i-1};\ \Theta,\ \tau_1^{\text{SI}},\ \ldots,\ \tau_{i-1}^{\text{SI}} \vdash \boxed{e_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_i}{\Sigma;\ \Delta;\ \Gamma_0;\ \Theta \vdash \boxed{(e_1,\ \ldots,\ e_n)} : (\tau_1^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}}) \Rightarrow \Gamma_n}$$

T-Array

$$\frac{\forall i \in \{\, 1 \,\dots\, n \,\}.\ \Sigma;\ \Delta;\ \Gamma_{i-1};\ \Theta,\ \tau_1^{\text{SI}},\ \dots,\ \tau_{i-1}^{\text{SI}} \vdash \boxed{e_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_i}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{[e_1,\ \dots,\ e_n]} : [\tau^{\text{SI}};\ n] \Rightarrow \Gamma_n}$$

T-Slice

$$\frac{\forall i \in \{\, 1 \,\dots\, n \,\}.\ \Sigma;\ \Delta;\ \Gamma_{i-1};\ \Theta,\ \tau_1^{\text{SI}},\ \dots,\ \tau_{i-1}^{\text{SI}} \vdash \boxed{e_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_i}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{[\![e_1,\ \dots,\ e_n]\!]} : [\tau^{\text{SI}}] \Rightarrow \Gamma_n}$$

T-Drop

$$\frac{\Gamma(\pi) = \tau_\pi^{\text{SI}} \qquad \Sigma;\ \Delta;\ \Gamma[\pi \mapsto \tau_\pi^{\text{SI}^\dagger}];\ \Theta \vdash \boxed{e} : \tau^{\text{SX}} \Rightarrow \Gamma_f}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau^{\text{SX}} \Rightarrow \Gamma_f}$$

T-Left

$$\frac{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau_1^{\text{SI}} \Rightarrow \Gamma'}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{Left::<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}\texttt{>}(e)} : \texttt{Either<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}\texttt{>} \Rightarrow \Gamma'}$$

T-Right

$$\frac{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau_2^{\text{SI}} \Rightarrow \Gamma'}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{Right::<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}\texttt{>}(e)} : \texttt{Either<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}\texttt{>} \Rightarrow \Gamma'}$$

T-Match

$$\frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \texttt{Either<}\tau_l^{\text{SI}},\ \tau_r^{\text{SI}}\texttt{>} \Rightarrow \Gamma' \qquad \forall r \in \text{free-regions}(\texttt{Either<}\tau_l^{\text{SI}},\ \tau_r^{\text{SI}}\texttt{>}).\ \Gamma' \vdash r\ \texttt{rnrb} \\ \Sigma;\ \Delta;\ \Gamma',\ x_1 : \tau_l^{\text{SI}};\ \Theta \vdash \boxed{e_1} : \tau_1^{\text{SI}} \Rightarrow \Gamma_1,\ x_1 : \tau_l^{\text{SD}} \\ \Sigma;\ \Delta;\ \Gamma',\ x_2 : \tau_r^{\text{SI}};\ \Theta \vdash \boxed{e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2,\ x_2 : \tau_r^{\text{SD}} \qquad \tau^{\text{SI}} = \tau_1^{\text{SI}} \lor \tau^{\text{SI}} = \tau_2^{\text{SI}} \\ \Delta;\ \Gamma_1;\ \Theta \vdash^{+} \tau_1^{\text{SI}} \rightsquigarrow \tau^{\text{SI}} \dashv \Gamma_1' \qquad \Delta;\ \Gamma_2;\ \Theta \vdash^{+} \tau_2^{\text{SI}} \rightsquigarrow \tau^{\text{SI}} \dashv \Gamma_2' \qquad \Gamma_1' \uplus \Gamma_2' = \Gamma'\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{match}\, e\ \{\, \texttt{Left}(x_1) \Rightarrow e_1,\ \texttt{Right}(x_2) \Rightarrow e_2 \,\}} : \tau^{\text{SI}} \Rightarrow \Gamma'}$$

B.5 Additional Judgments

$\boxed{\omega \lesssim \omega'}$
read: "ω is less than ω' in the qualifier ordering"

QO-Refl

$$\frac{}{\omega \lesssim \omega}$$

QO-ShrdUniq

$$\frac{}{\texttt{shrd} \lesssim \texttt{uniq}}$$

$\boxed{\Sigma;\ \Delta \vdash \Gamma \triangleright \Gamma'}$
read: "Γ is related to Γ' under Σ and Δ"

R-Env

$$\frac{\begin{array}{c}\vdash \Sigma;\ \Delta;\ \Gamma;\ \bullet \qquad \vdash \Sigma;\ \Delta;\ \Gamma';\ \bullet \qquad \text{dom}(\Gamma) = \text{dom}(\Gamma') \\ \forall x : \tau \in \Gamma'.\ \forall r \text{ that occurs in } \tau.\ \Gamma(r) = \Gamma'(r) \\ \forall r \in \text{dom}(\Gamma).\ \Gamma(r) = \Gamma'(r) \lor \Gamma'(r) = \emptyset \\ \forall \pi \in \text{dom}(\text{explode}(\Gamma)).\ \Gamma'(\pi) = \Gamma(\pi) \lor \Gamma'(\pi) = \Gamma(\pi)^\dagger\end{array}}{\Sigma;\ \Delta \vdash \Gamma \triangleright \Gamma'}$$

$\boxed{\Delta;\ \Gamma \vdash_{\omega} p : \tau,\ \{\, \overline{\rho}\, \}}$

read: "p in an ω context has type τ under Δ and Γ, passing through the regions in $\overline{\rho}$"

TC-Var
$$\frac{\Gamma(x) = \tau^{\text{SI}}}{\Delta;\ \Gamma \vdash_{\omega} x : \tau^{\text{SI}},\ \emptyset}$$

TC-Proj
$$\frac{\Delta;\ \Gamma \vdash_{\omega} p : (\tau_1^{\text{SI}}, \ldots, \tau_i^{\text{SI}}, \ldots, \tau_n^{\text{SI}}),\ \{\, \overline{\rho_p}\, \}}{\Delta;\ \Gamma \vdash_{\omega} p.i : \tau_i^{\text{SI}},\ \{\, \overline{\rho_p}\, \}}$$

TC-Deref
$$\frac{\Delta;\ \Gamma \vdash_{\omega} p : \&\rho\ \omega'\ \tau^{\text{XI}},\ \{\, \overline{\rho_p}\, \} \qquad \omega \lesssim \omega'}{\Delta;\ \Gamma \vdash_{\omega} *p : \tau^{\text{XI}},\ \{\, \overline{\rho_p},\ \rho\, \}}$$

$\boxed{\Delta;\ \Gamma \vdash_{\omega} p : \tau}$

read: "p in an ω context has type τ under Δ and Γ"

$$\Delta;\ \Gamma \vdash_{\omega} p : \tau = \Delta;\ \Gamma \vdash_{\omega} p : \tau,\ \_$$

$\boxed{\Sigma;\ \Gamma \vdash \overline{v} : \Theta}$

read: "the given values $\overline{v}$ satisfy Θ under Σ and Γ"

WF-Temporaries
$$\frac{\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma;\ \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma}{\Sigma;\ \Gamma \vdash v_1, \ldots, v_n : \tau_1^{\text{SI}}, \ldots, \tau_n^{\text{SI}}}$$

$\boxed{\Gamma \vdash r\ \texttt{rnrb}}$

read: "the region r is not reborrowed in Γ"

NRB-Region
$$\frac{\forall \pi\ :\ \&r\ \omega\ \tau^{\text{XI}} \in \text{explode}(\Gamma).\ \nexists r'.\ {}^{\omega} *\pi\ \in \Gamma(r')}{\Gamma \vdash r\ \texttt{rnrb}}$$

$\boxed{\Gamma;\ \Theta \vdash \{\, \overline{r}\, \}\ \texttt{clrs}}$

read: "the regions $\overline{r}$ follow the closure restriction in Θ or Γ"

CLS-Restriction
$$\frac{\begin{array}{c}\forall \tau \in \text{cod}(\Gamma) \cup \Theta.\ \forall\texttt{<\_>}(\tau_1^{\text{SI}}, \ldots, \tau_n^{\text{SI}}) \to \tau_r^{\text{SI}}\ \texttt{occurs in}\ \tau \implies \\ (\{\, \overline{r}\, \} \subseteq \bigcup_{i=1}^{n} \text{free-regions}(\tau_i^{\text{SI}}) \cup \text{free-regions}(\tau_r^{\text{SI}}) \lor (\{\, \overline{r}\, \} \cap \bigcup_{i=1}^{n} \text{free-regions}(\tau_i^{\text{SI}}) \cup \text{free-regions}(\tau_r^{\text{SI}})) = \emptyset)\end{array}}{\Gamma;\ \Theta \vdash \{\, \overline{r}\, \}\ \texttt{clrs}}$$

$\boxed{\Gamma;\ \Theta \vdash r\ \texttt{rnic}}$

read: "the region r is not in a closure's signature in Θ or Γ"

CLS-RegionNotIn
$$\frac{\forall \tau \in \text{cod}(\Gamma) \cup \Theta.\ \forall\texttt{<\_>}(\tau_1^{\text{SI}}, \ldots, \tau_n^{\text{SI}}) \to \tau_r^{\text{SI}}\ \texttt{occurs in}\ \tau \implies r \notin \bigcup_{i=1}^{n} \text{free-regions}(\tau_i^{\text{SI}}) \cup \text{free-regions}(\tau_r^{\text{SI}})}{\Gamma;\ \Theta \vdash r\ \texttt{rnic}}$$

C METAFUNCTIONS

free-nc-vars$_{\sigma}(e)$ = all the variables x free in e which are bound to values in σ that are non-copyable.
free-nc-vars$_{\Gamma}(e)$ = all the variables x free in e which are bound to types in Γ that are non-copyable.
$\pi_1 \# \pi_2$ = π_1 is not a prefix of π_2 and π_2 is not a prefix of π_1 and $\pi_1 \neq \pi_2$.

$p_1 \# p_2 = p_1 = p_1^{\square}[\pi_1]$ and $p_2 = p_2^{\square}[\pi_2]$ and $\pi_1 \# \pi_2$.

$\boxed{\Gamma_1 \Cup \Gamma_2}$

$$\begin{array}{lcl} (\Gamma_1 \natural \mathcal{F}_1) \Cup (\Gamma_2 \natural \mathcal{F}_2) & = & (\Gamma_1 \Cup \Gamma_2) \natural (\mathcal{F}_1 \Cup \mathcal{F}_2) \\ \bullet \Cup \bullet & = & \bullet \end{array}$$

$\boxed{\mathcal{F}_1 \Cup \mathcal{F}_2}$

$$\begin{array}{lcl} (\mathcal{F}_1, x : \tau) \Cup (\mathcal{F}_2, x : \tau) & = & (\mathcal{F}_1 \Cup \mathcal{F}_2), x : \tau \\ (\mathcal{F}_1, r \mapsto \{\, \overline{\ell} \,\}) \Cup (\mathcal{F}_2, r \mapsto \{\, \overline{\ell'} \,\}) & = & (\mathcal{F}_1 \Cup \mathcal{F}_2), r \mapsto \{\, \overline{\ell}, \overline{\ell'} \,\} \\ \bullet \Cup \bullet & = & \bullet \end{array}$$

$\boxed{\mathsf{places}(\Gamma) = \{\, \overline{\pi} \,\}}$

$$\begin{array}{l} \mathsf{places}\ (\bullet) = \emptyset \\ \mathsf{places}\ (\Gamma,\ r \mapsto \{\, \overline{{}^{\omega}p} \,\}) = \{\, \pi \mid {}^{\omega_i}p_i \in \{\, \overline{{}^{\omega}p} \,\} \wedge (p_i = \pi \vee p_i = p^{\square}[*\pi]) \,\} \cup \mathsf{places}\ \Gamma \\ \mathsf{places}\ (\Gamma,\ x : \tau) = \mathsf{places}\ (\Gamma \natural \bullet) = \mathsf{places}\ (\Gamma) \end{array}$$

$\boxed{v.q \rightsquigarrow C \boxplus v}$

$$\textsc{DV-End}\quad \frac{}{v.\epsilon \rightsquigarrow \square \boxplus v} \qquad \textsc{DV-Projection}\quad \frac{v_i.q \rightsquigarrow C \boxplus v}{(v_0, \ldots, v_i, \ldots, v_n).i.q \rightsquigarrow (v_0, \ldots, C, \ldots, v_n) \boxplus v}$$

$\boxed{\sigma[\pi \mapsto v]}$

$$\begin{array}{l} \sigma[x.q \mapsto v] = \sigma[x \mapsto C[v]] \\ \quad \textbf{where}\ \ \sigma(x).q \rightsquigarrow C \boxplus \_ \end{array}$$

$\boxed{\sigma(\pi)}$

$$\begin{array}{l} \sigma(x.q) = v \\ \quad \textbf{where}\ \ \sigma(x).q \rightsquigarrow \_ \boxplus v \end{array}$$

$\boxed{\tau.q \rightsquigarrow \tau_{\square} \boxplus \tau}$

$$\textsc{D-End}\quad \frac{}{\tau.\epsilon \rightsquigarrow \square \boxplus \tau} \qquad \textsc{D-Projection}\quad \frac{\tau_i.q \rightsquigarrow \tau_{\square} \boxplus \tau}{(\tau_0, \ldots, \tau_i, \ldots, \tau_n).i.q \rightsquigarrow (\tau_0, \ldots, \tau_{\square}, \ldots, \tau_n) \boxplus \tau}$$

$\boxed{\Gamma[\pi \mapsto \tau]}$

$$\begin{array}{l} \Gamma[x.q \mapsto \tau] = \Gamma[x \mapsto \tau_{\square}[\tau]] \\ \quad \textbf{where}\ \ \Gamma(x).q \rightsquigarrow \tau_{\square} \boxplus \_ \end{array}$$

$\boxed{\Gamma(\pi)}$

$\Gamma(x.q) = \tau$
  where $\Gamma(x).q \rightsquigarrow \_ \boxplus \tau$

$\boxed{\texttt{noncopyable}_{\Sigma}\ \tau}$

$$\begin{aligned}
\texttt{noncopyable}_{\Sigma}\ \tau^{\mathrm{B}} &= \bot \\
\texttt{noncopyable}_{\Sigma}\ \alpha &= \top \\
\texttt{noncopyable}_{\Sigma}\ \&\_\ \texttt{uniq}\ \_ &= \top \\
\texttt{noncopyable}_{\Sigma}\ \&\_\ \texttt{shrd}\ \_ &= \bot \\
\texttt{noncopyable}_{\Sigma}\ \forall\texttt{<}\_\texttt{>}(\_)\ \xrightarrow{\_}\ \_ &= \bot \\
\texttt{noncopyable}_{\Sigma}\ [\tau;\ \_] &= \texttt{noncopyable}_{\Sigma}\ \tau \\
\texttt{noncopyable}_{\Sigma}\ [\tau] &= \texttt{noncopyable}_{\Sigma}\ \tau \\
\texttt{noncopyable}_{\Sigma}\ (\tau, \ldots) &= \texttt{noncopyable}_{\Sigma}\ \tau \lor \ldots
\end{aligned}$$

$\boxed{\texttt{copyable}_{\Sigma}\ \tau}$

$$\texttt{copyable}_{\Sigma}\ \tau = \neg\, \texttt{noncopyable}_{\Sigma}\ \tau$$

$\boxed{\mathrm{explode}(\Gamma)}$

$\mathrm{explode}(\Gamma) = \bigcup_{x\,:\,\tau \,\in\, \Gamma} \mathrm{explode}(x\ :\ \tau)$

$\boxed{\mathrm{explode}(\pi\ :\ \tau)}$

$$\begin{aligned}
\texttt{explode}\ \pi\ :\ \tau^{\mathrm{B}} &= \{\ \pi\ :\ \tau^{\mathrm{B}}\ \} \\
\texttt{explode}\ \pi\ :\ \alpha &= \{\ \pi\ :\ \alpha\ \} \\
\texttt{explode}\ \pi\ :\ \&\rho\ \omega\ \tau^{\mathrm{XI}} &= \{\ \pi\ :\ \&\rho\ \omega\ \tau^{\mathrm{XI}}\ \} \\
\texttt{explode}\ \pi\ :\ [\tau^{\mathrm{SI}};\ n] &= \{\ \pi\ :\ [\tau^{\mathrm{SI}};\ n]\ \} \\
\texttt{explode}\ \pi\ :\ (\tau_1^{\mathrm{SX}}\ \ldots\ \tau_n^{\mathrm{SX}}) &= \bigcup_{i\,\in\,1\ldots n} \mathrm{explode}(\pi.i\ :\ \tau_i^{\mathrm{SX}})
\end{aligned}$$

$$\texttt{explode}\ \pi\ :\ \forall\texttt{<}\overline{\varphi},\ \overline{\varrho},\ \overline{\alpha}\texttt{>}(\tau_1^{\mathrm{SI}},\ \ldots,\ \tau_n^{\mathrm{SI}})\ \xrightarrow{\Phi}\ \tau_r^{\mathrm{SI}}\ \texttt{where}\ \overline{\varrho_1 : \varrho_2} = \{\ \pi\ :\ \forall\texttt{<}\overline{\varphi},\ \overline{\varrho},\ \overline{\alpha}\texttt{>}(\tau_1^{\mathrm{SI}},\ \ldots,\ \tau_n^{\mathrm{SI}})\ \xrightarrow{\Phi}\ \tau_r^{\mathrm{SI}}\ \texttt{where}\ \overline{\varrho_1 : \varrho_2}\ \}$$

$$\texttt{explode}\ \pi\ :\ {\tau^{\mathrm{SI}}}^{\dagger} = \{\ \pi\ :\ {\tau^{\mathrm{SI}}}^{\dagger}\ \}$$

$\boxed{r_1\ \texttt{occurs before}\ r_2\ \texttt{in}\ \Gamma}$

OC-OccursBase
$$\frac{r_1 \in \mathrm{dom}(\Gamma)}{r_1\ \texttt{occurs before}\ r_2\ \texttt{in}\ \Gamma,\ r_2 \mapsto \{\ \overline{\ell}\ \}}$$

OC-OccursExtendFrame
$$\frac{r_1\ \texttt{occurs before}\ r_2\ \texttt{in}\ \Gamma}{r_1\ \texttt{occurs before}\ r_2\ \texttt{in}\ \Gamma,\ \mathcal{F}'}$$

OC-OccursNewFrame
$$\frac{r_1\ \texttt{occurs before}\ r_2\ \texttt{in}\ \Gamma}{r_1\ \texttt{occurs before}\ r_2\ \texttt{in}\ \Gamma \natural \mathcal{F}}$$

$\boxed{\text{gc-loans}_{\Theta}(\Gamma)}$

$\text{gc-loans}_{\Theta}(\Gamma) = \Gamma[\overline{r \mapsto \emptyset}]$
  where $\overline{r} = \{\ r \in \mathrm{dom}(\Gamma)\ |\ \forall\tau \in \mathrm{cod}(\Theta) \cup \mathrm{cod}(\Gamma).\ r$ does not occur in $\tau\ \}$

$\boxed{r \text{ is unique to } \pi \text{ in } \Gamma}$

r is unique to π in $\Gamma = \forall\pi'\ :\ \&r'\ \omega'\ \tau_2^{\mathrm{XI}} \in \mathrm{explode}(\Gamma).\ \pi = \pi' \lor r \neq r'$

$\boxed{\Gamma \unrhd p}$

$\Gamma \unrhd p = \Gamma'$ where $\mathrm{dom}(\Gamma) = \mathrm{dom}(\Gamma')$ and
$\forall r.\ \Gamma'(r) = \{\ {}^{\omega}p'\ \in \Gamma(r)\ |\ p' \neq p^{\square}[p]\ \}$ and

$\forall \pi.\ \Gamma(\pi) = \Gamma'(\pi)$

$\boxed{\Gamma \trianglerighteq \{\,\overline{p}\,\}}$

$\Gamma \trianglerighteq \{\,p,\ \overline{p}\,\} = (\Gamma \trianglerighteq p) \trianglerighteq \{\,\overline{p}\,\}$
$\Gamma \trianglerighteq \emptyset = \Gamma$

$\boxed{\text{regions}(\Gamma,\ \Theta) = \{\,\overline{r \mapsto \{\,\overline{\ell}\,\}}\,\}}$

$\text{regions}(\Gamma,\ \Theta) = \{\,r \mapsto \{\,\overline{\ell}\,\} \in \Gamma\,\} \cup \{\,r \mapsto \{\,\overline{\ell}\,\} \in \mathcal{F}_c \mid \exists \tau^{\text{XI}} \in \Gamma \lor \exists \tau^{\text{XI}} \in \Theta.\ (\tau_1^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}}) \xrightarrow{\mathcal{F}_c} \tau_r^{\text{SI}}\ \mathtt{occurs\ in}\ \tau^{\text{XI}}\,\}$

D DYNAMICS

$$\begin{array}{llcl}
\text{Referent} & \mathcal{R} & ::= & x \mid \mathcal{R}.n \mid \mathcal{R}[n] \mid \mathcal{R}[n_1..n_2] \\
\text{Referent Context} & \mathcal{R}^{\square} & ::= & \square \mid \mathcal{R}^{\square}.n \mid \mathcal{R}^{\square}[n] \mid \mathcal{R}^{\square}[n_1..n_2] \\
\text{Expressions} & e & ::= & \ldots\ \mid \mathtt{framed}\ e \mid \mathtt{shift}\ e \mid [\![v_1,\ \ldots,\ v_n]\!] \mid \mathtt{dead} \mid \mathtt{ptr}\ \mathcal{R} \\
 & & \mid & \langle \varsigma,\ |x_1 : \tau_1^{\text{SI}},\ \ldots,\ x_n : \tau_n^{\text{SI}}| \to \tau_r^{\text{SI}}\ \{\ e\ \}\ \rangle \\
 & & & \\
\text{Values} & v & ::= & c \mid (v_1,\ \ldots,\ v_n) \mid [v_1,\ \ldots,\ v_n] \mid [\![v_1,\ \ldots,\ v_n]\!] \mid f \mid \mathtt{dead} \mid \mathtt{ptr}\ \mathcal{R} \\
 & & \mid & \langle \varsigma,\ |x_1 : \tau_1^{\text{SI}},\ \ldots,\ x_n : \tau_n^{\text{SI}}| \to \tau_r^{\text{SI}}\ \{\ e\ \}\ \rangle \\
\text{Eval. Contexts} & C & ::= & \square \\
 & & \mid & \&\rho\ \omega\ p[C] \mid \&\rho\ \omega\ p[C..\hat{e}] \mid \&\rho\ \omega\ p[v..C] \\
 & & \mid & \mathtt{let}\ x : \tau^{\text{SI}} = C;\ e \mid \mathtt{letrgn}\ {<}r{>}\ \{\ C\ \} \\
 & & \mid & p \coloneqq C \mid C;\ e \mid \mathtt{framed}\ C \\
 & & \mid & \mathtt{shift}\ C \mid \mathtt{shiftprov}\ C \\
 & & \mid & C{::}{<}\overline{\Phi},\ \overline{\rho},\ \overline{\tau^{\text{SI}}}{>}(\hat{e}_1,\ \ldots,\ \hat{e}_n) \\
 & & \mid & v{::}{<}\overline{\Phi},\ \overline{\rho},\ \overline{\tau^{\text{SI}}}{>}(v_1,\ \ldots,\ v_m,\ C,\ \hat{e}_1,\ \ldots,\ \hat{e}_n) \\
 & & \mid & p[C] \mid \mathtt{if}\ C\ \{\ e_1\ \}\ \mathtt{else}\ \{\ e_2\ \} \\
 & & \mid & \mathtt{for}\ x\ \mathtt{in}\ C\ \{\ e\ \} \\
 & & \mid & (v_1,\ \ldots,\ v_m,\ C,\ \hat{e}_1,\ \ldots,\ \hat{e}_n) \\
 & & \mid & [v_1,\ \ldots,\ v_m,\ C,\ \hat{e}_1,\ \ldots,\ \hat{e}_n] \\
 & & \mid & \mathtt{Left}{::}{<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}{>}(C) \mid \mathtt{Right}{::}{<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}{>}(C) \\
 & & \mid & \mathtt{match}\ C\ \ \{\ \mathtt{Left}(x_1) \Rightarrow e_1,\ \mathtt{Right}(x_2) \Rightarrow e_2\ \} \\
\text{Value Contexts} & \mathcal{V} & ::= & \square \mid (v_1,\ \ldots,\ \mathcal{V},\ \ldots,\ v_n) \mid [v_1,\ \ldots,\ \mathcal{V}_1,\ \ldots,\ \mathcal{V}_m,\ \ldots,\ v_n] \\
\text{Stacks} & \sigma & ::= & \bullet \mid \sigma \natural \varsigma \\
\text{Stack Frame} & \varsigma & ::= & \bullet \mid \varsigma,\ x \mapsto v
\end{array}$$

$\boxed{\Sigma;\ \Gamma \vdash \mathcal{R} : \tau^{\text{XI}}}$

$$\textsc{WF-RefId}\ \ \frac{\Gamma(x) = \tau^{\text{SI}}}{\Sigma;\ \Gamma \vdash x : \tau^{\text{SI}}}
\qquad
\textsc{WF-RefProjection}\ \ \frac{\Sigma;\ \Gamma \vdash \mathcal{R} : (\tau_0^{\text{SI}},\ \ldots,\ \tau_i^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}})}{\Sigma;\ \Gamma \vdash \mathcal{R}.i : \tau_i^{\text{SI}}}
\qquad
\textsc{WF-RefIndexArray}\ \ \frac{\Sigma;\ \Gamma \vdash \mathcal{R} : [\tau^{\text{SI}};\ n] \qquad 0 \leq i < n}{\Sigma;\ \Gamma \vdash \mathcal{R}[i] : \tau^{\text{SI}}}$$

$$\textsc{WF-RefIndexSlice}\ \ \frac{\Sigma;\ \Gamma \vdash \mathcal{R} : [\tau^{\text{SI}}]}{\Sigma;\ \Gamma \vdash \mathcal{R}[i] : \tau^{\text{SI}}}
\qquad
\textsc{WF-RefSliceArray}\ \ \frac{\Sigma;\ \Gamma \vdash \mathcal{R} : [\tau^{\text{SI}};\ n] \qquad 0 \leq i \leq j < n}{\Sigma;\ \Gamma \vdash \mathcal{R}[i..j] : [\tau^{\text{SI}}]}
\qquad
\textsc{WF-RefSliceSlice}\ \ \frac{\Sigma;\ \Gamma \vdash \mathcal{R} : [\tau^{\text{SI}}] \qquad i \leq j}{\Sigma;\ \Gamma \vdash \mathcal{R}[i..j] : [\tau^{\text{SI}}]}$$

$$\boxed{\sigma \vdash \mathcal{R} \Downarrow \mathcal{V} \times v}$$

ER-Id
$$\frac{\sigma(x) = v}{\sigma \vdash x \Downarrow \square \times v}$$

ER-Projection
$$\frac{\sigma \vdash \mathcal{R} \Downarrow \mathcal{V} \times (v_0\,,\,\ldots\,,\,v_i\,,\,\ldots\,,\,v_n)}{\sigma \vdash \mathcal{R}.i \Downarrow \mathcal{V}[(v_0\,,\,\ldots\,,\,\square,\,\ldots\,,\,v_n)] \times v_i}$$

ER-IndexArray
$$\frac{\sigma \vdash \mathcal{R} \Downarrow \mathcal{V} \times [v_0\,,\,\ldots\,,\,v_i\,,\,\ldots\,,\,v_n]}{\sigma \vdash \mathcal{R}[i] \Downarrow \mathcal{V}[[v_0\,,\,\ldots\,,\,\square,\,\ldots\,,\,v_n]] \times v_i}$$

ER-IndexSlice
$$\frac{\sigma \vdash \mathcal{R} \Downarrow \mathcal{V} \times \llbracket v_0\,,\,\ldots\,,\,v_k\,,\,\ldots\,,\,v_n \rrbracket}{\sigma \vdash \mathcal{R}[k] \Downarrow \mathcal{V}[v_0]\ \ldots\ [\square]\ \ldots\ [v_n] \times v_{i+k}}$$

ER-SliceArray
$$\frac{\sigma \vdash \mathcal{R} \Downarrow \mathcal{V} \times [v_0\,,\,\ldots\,,\,v_i\,,\,\ldots\,,\,v_j\,,\,\ldots\,,\,v_n]}{\sigma \vdash \mathcal{R}[i..j] \Downarrow \mathcal{V}[[v_0\,,\,\ldots\,,\,\square^{j-i+1}\,,\,\ldots\,,\,v_n]] \times \llbracket v_i\,,\,\ldots\,,\,v_j \rrbracket}$$

ER-SliceSlice
$$\frac{\sigma \vdash \mathcal{R} \Downarrow \mathcal{V} \times \llbracket v_0\,,\,\ldots\,,\,v_i\,,\,\ldots\,,\,v_j\,,\,\ldots\,,\,v_n \rrbracket}{\sigma \vdash \mathcal{R}[i..j] \Downarrow \mathcal{V}[v_0]\ \ldots\ [\square]\ \ldots\ [\square]\ \ldots\ [v_n] \times \llbracket v_i\,,\,\ldots\,,\,v_j \rrbracket}$$

$$\boxed{\sigma \vdash p \Downarrow \mathcal{R} \mapsto \mathcal{V}[v]}$$

read: "p computes to $\mathcal{R}$, which maps to v in σ."

Let $\sigma \vdash p^{\square}[x] \Downarrow \mathcal{R} \mapsto \mathcal{V}[v] = \sigma \vdash p^{\square} \times x \Downarrow (\mathcal{R}, \mathcal{V}, v)$.

$$\boxed{\sigma \vdash p^{\square} \times \mathcal{R} \Downarrow (\mathcal{R}', \mathcal{V}, v)}$$

read: "$\mathcal{R}$ in a context $p^{\square}$ computes to $\mathcal{R}'$ which maps to v in σ with a context of $\mathcal{V}$."

P-Referent
$$\frac{\sigma \vdash \mathcal{R} \Downarrow \mathcal{V} \times v}{\sigma \vdash \square \times \mathcal{R} \Downarrow (\mathcal{R}, \mathcal{V}, v)}$$

P-Proj
$$\frac{\sigma \vdash p^{\square} \times \mathcal{R}_1 \Downarrow (\mathcal{R}_2, \mathcal{V}, (v_0\,,\,\ldots\,,\,v_i\,,\,\ldots\,,\,v_n))}{\sigma \vdash p^{\square}[\square.i] \times \mathcal{R}_1 \Downarrow (\mathcal{R}_2.i, \mathcal{V}[(v_0\,,\,\ldots\,,\,\square,\,\ldots\,,\,v_n)], v_i)}$$

P-DerefPtr
$$\frac{\begin{array}{c}\sigma \vdash \square \times \mathcal{R}_1 \Downarrow (\_, \mathsf{ptr}\ \pi,) \\ \sigma \vdash p^{\square} \times \pi \Downarrow (\mathcal{R}_2, \mathcal{V}, v)\end{array}}{\sigma \vdash p^{\square}[*\square] \times \mathcal{R}_1 \Downarrow (\mathcal{R}_2, \mathcal{V}, v)}$$

P-DerefIndexPtrArray
$$\frac{\begin{array}{c}\sigma \vdash \square \times \mathcal{R}_1 \Downarrow (\_, \mathsf{ptr}\ \mathcal{R}_2[i],) \\ \sigma \vdash p^{\square} \times \mathcal{R}_2 \Downarrow (\mathcal{R}_3, \mathcal{V}, [v_0\,,\,\ldots\,,\,v_i\,,\,\ldots\,,\,v_n])\end{array}}{\sigma \vdash p^{\square}[*\square] \times \mathcal{R}_1 \Downarrow (\mathcal{R}_3[i], \mathcal{V}[[v_0\,,\,\ldots\,,\,\square,\,\ldots\,,\,v_n]], v_i)}$$

P-DerefIndexPtrSlice
$$\frac{\begin{array}{c}\sigma \vdash \square \times \mathcal{R}_1 \Downarrow (\_, \mathsf{ptr}\ \mathcal{R}_2[i],) \\ \sigma \vdash p^{\square} \times \mathcal{R}_2 \Downarrow (\mathcal{R}_3, \mathcal{V}, \llbracket v_0\,,\,\ldots\,,\,v_i\,,\,\ldots\,,\,v_n \rrbracket)\end{array}}{\sigma \vdash p^{\square}[*\square] \times \mathcal{R}_1 \Downarrow (\mathcal{R}_3[i], \mathcal{V}[v_0]\ \ldots\ [\square]\ \ldots\ [v_n], v_i)}$$

P-DerefSlicePtrArray
$$\frac{\begin{array}{c}\sigma \vdash \square \times \mathcal{R}_1 \Downarrow (\_, \mathsf{ptr}\ \mathcal{R}_2[i..j],) \\ \sigma \vdash p^{\square} \times \mathcal{R}_2 \Downarrow (\mathcal{R}_3, \mathcal{V}, [v_0\,,\,\ldots\,,\,v_i\,,\,\ldots\,,\,v_j\,,\,\ldots\,,\,v_n])\end{array}}{\sigma \vdash p^{\square}[*\square] \times \mathcal{R}_1 \Downarrow (\mathcal{R}_3[i..j], \mathcal{V}[[v_0\,,\,\ldots\,,\,\square^{j-i+1}\,,\,\ldots\,,\,v_n]], \llbracket v_i\,,\,\ldots\,,\,v_j \rrbracket)}$$

P-DerefSlicePtrSlice
$$\frac{\begin{array}{c}\sigma \vdash \square \times \mathcal{R}_1 \Downarrow (\_, \mathsf{ptr}\ \mathcal{R}_2[i..j],) \\ \sigma \vdash p^{\square} \times \mathcal{R}_2 \Downarrow (\mathcal{R}_3, \mathcal{V}, \llbracket v_0\,,\,\ldots\,,\,v_i\,,\,\ldots\,,\,v_j\,,\,\ldots\,,\,v_n \rrbracket)\end{array}}{\sigma \vdash p^{\square}[*\square] \times \mathcal{R}_1 \Downarrow (\mathcal{R}_3[i..j], \mathcal{V}[v_0]\ \ldots\ [\square]\ \ldots\ [\square]\ \ldots\ [v_n], \llbracket v_i\,,\,\ldots\,,\,v_j \rrbracket)}$$

$\boxed{\Sigma \vdash \sigma : \Gamma}$
read: “σ satisfies Γ under global context Σ”

WF-StackEmpty

$$\frac{}{\Sigma \vdash \bullet : \bullet}$$

WF-StackFrame

$$\frac{\Sigma \vdash \sigma : \Gamma \qquad \mathrm{dom}(\varsigma) = \mathrm{dom}(\mathcal{F})|_x \\ \forall x \in \mathrm{dom}(\varsigma).\ \Sigma; \bullet; \Gamma \natural \mathcal{F}; \bullet \vdash \boxed{(\sigma \natural \varsigma)(x)} : (\Gamma \natural \mathcal{F})(x) \Rightarrow \Gamma \natural \mathcal{F}}{\Sigma \vdash \sigma \natural \varsigma : \Gamma \natural \mathcal{F}}$$

$\boxed{\Sigma; \Gamma \vdash \varsigma : \mathcal{F}_c}$
read: “ς satisfies $\mathcal{F}_c$ under Σ and Γ”

WF-Frame

$$\frac{\mathrm{dom}(\varsigma) = \mathrm{dom}(\mathcal{F}_c)|_x \qquad \forall x \in \mathrm{dom}(\varsigma).\ \Sigma; \bullet; \Gamma \natural \mathcal{F}_c; \bullet \vdash \boxed{\varsigma(x)} : \mathcal{F}_c(x) \Rightarrow \Gamma \natural \mathcal{F}_c}{\Sigma; \Gamma \vdash \varsigma : \mathcal{F}_c}$$

$\boxed{\Sigma; \Delta; \Gamma; \Theta \vdash e : \tau \Rightarrow \Gamma'}$ where $\vdash \Sigma; \Delta; \Gamma; \Theta$ and $\Sigma; \Delta; \Gamma' \vdash \tau$

...

T-Shift

$$\frac{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{e} : \tau^{\text{SI}} \Rightarrow \Gamma',\ x\ :\ \tau^{\text{SD}}}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{\texttt{shift}\ e} : \tau^{\text{SI}} \Rightarrow \Gamma'}$$

T-Framed

$$\frac{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{e} : \tau^{\text{SI}} \Rightarrow \Gamma' \natural \mathcal{F}'}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{\texttt{framed}\ e} : \tau^{\text{SI}} \Rightarrow \Gamma'}$$

T-Pointer

$$\frac{\Sigma; \Gamma \vdash \mathcal{R}^{\square}[\pi] : \tau^{\text{XI}} \qquad {}^{\omega}\pi\ \in \Gamma(r)}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{\texttt{ptr}\ \mathcal{R}^{\square}[\pi]} : \&r\ \omega\ \tau^{\text{XI}} \Rightarrow \Gamma}$$

T-ClosureValue

$$\frac{\mathrm{free\text{-}vars}(e) \setminus \overline{x} = \overline{x_f} = \mathrm{dom}(\mathcal{F}_c)|_{\text{VAR}} \\ \overline{r} = \overline{\mathrm{free\text{-}regions}(\Gamma(x_f))},\ (\mathrm{free\text{-}regions}(e) \setminus (\overline{\mathrm{free\text{-}regions}(\tau^{\text{SI}})}, \mathrm{free\text{-}regions}(\tau_r^{\text{SI}}))) = \mathrm{dom}(\mathcal{F}_c)|_{\text{RGN}} \\ \Sigma; \Gamma \vdash \varsigma_c : \mathcal{F}_c \qquad \Sigma; \Delta; \Gamma \natural \mathcal{F}_c,\ x_1\ :\ \tau_1^{\text{SI}},\ \ldots,\ x_n\ :\ \tau_n^{\text{SI}}; \Theta \vdash \boxed{e} : \tau_r^{\text{SI}} \Rightarrow \Gamma' \natural \mathcal{F}}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{\langle \varsigma_c,\ |x_1 : \tau_1^{\text{SI}},\ \ldots,\ x_n : \tau_n^{\text{SI}}|\ \to\ \tau_r^{\text{SI}}\ \{\ e\ \}\ \rangle} : (\tau_1^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}})\ \overset{\mathcal{F}_c}{\to}\ \tau_r^{\text{SI}} \Rightarrow \Gamma}$$

T-Dead

$$\frac{}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{v} : \tau^{\text{SI}^{\dagger}} \Rightarrow \Gamma}$$

$\boxed{\Sigma \vdash (\sigma; \boxed{e}) \to (\sigma'; \boxed{e'})}$
read: “σ and e step to σ' and e' under Σ”

E-Move

$$\frac{\sigma \vdash \pi \Downarrow \pi \mapsto \_[v]}{\Sigma \vdash (\sigma; \boxed{\pi}) \to (\sigma[\pi \mapsto \mathsf{dead}]; \boxed{v})}$$

E-Copy

$$\frac{\sigma \vdash p \Downarrow \_ \mapsto \_[v]}{\Sigma \vdash (\sigma; \boxed{p}) \to (\sigma; \boxed{v})}$$

E-Borrow

$$\frac{\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[\_]}{\Sigma \vdash (\sigma; \boxed{\&r\ \omega\ p}) \to (\sigma; \boxed{\texttt{ptr}\ \mathcal{R}})}$$

E-BorrowIndex

$$\frac{\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[[v_0,\ \ldots,\ v_n]] \qquad 0 \le n_i \le n}{\Sigma \vdash (\sigma; \boxed{\&r\ \omega\ p[n_i]}) \to (\sigma; \boxed{\texttt{ptr}\ \mathcal{R}[n_i]})}$$

E-BorrowSlice

$$\frac{\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[[v_0,\ \ldots,\ v_n]] \qquad 0 \le n_1 \le n_2 \le n}{\Sigma \vdash (\sigma; \boxed{\&r\ \omega\ p[n_1..n_2]}) \to (\sigma; \boxed{\texttt{ptr}\ \mathcal{R}[n_1..n_2]})}$$

E-BorrowIndexOOB
$$\frac{\sigma \vdash p \Downarrow \_ \mapsto \_[[v_0\,,\ \ldots\ ,\ v_n]] \qquad n_i < 0 \lor n_i > n}{\Sigma \vdash (\sigma;\ \boxed{\&r\ \omega\ *p[n_i]}) \to (\sigma;\ \boxed{\texttt{abort!}(\text{“attempted to index out of bounds”})})}$$

E-BorrowSliceOOB
$$\frac{\sigma \vdash p \Downarrow \_ \mapsto \_[[v_0\,,\ \ldots\ ,\ v_n]] \qquad n_1 < 0 \lor n_1 > n \lor n_2 < 0 \lor n_2 > n \lor n_1 > n_2}{\Sigma \vdash (\sigma;\ \boxed{\&r\ \omega\ p[n_1..n_2]}) \to (\sigma;\ \boxed{\texttt{abort!}(\text{“attempted to slice out of bounds”})})}$$

E-IndexCopy
$$\frac{\sigma \vdash p \Downarrow \_ \mapsto \_[[v_0\,,\ \ldots\ ,\ v_{n_i}\,,\ \ldots\ ,\ v_n]]}{\Sigma \vdash (\sigma;\ \boxed{p[n_i]}) \to (\sigma;\ \boxed{v_{n_i}})}$$

E-IndexCopyOOB
$$\frac{\sigma \vdash p \Downarrow \_ \mapsto \_[[v_0\,,\ \ldots\ ,\ v_n]] \qquad n_i < 0 \lor n_i > n}{\Sigma \vdash (\sigma;\ \boxed{p[n_i]}) \to (\sigma;\ \boxed{\texttt{abort!}(\text{“attempted to index out of bounds”})})}$$

E-Framed
$$\frac{}{\Sigma \vdash (\sigma \natural \varsigma;\ \boxed{\texttt{framed}\ v}) \to (\sigma;\ \boxed{v})}$$

E-Shift
$$\frac{}{\Sigma \vdash (\sigma\,,\ x \mapsto v';\ \boxed{\texttt{shift}\ v}) \to (\sigma;\ \boxed{v})}$$

E-IfTrue
$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{if true}\ \{\ e_1\ \}\ \texttt{else}\ \{\ e_2\ \}}) \to (\sigma;\ \boxed{e_1})}$$

E-IfFalse
$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{if false}\ \{\ e_1\ \}\ \texttt{else}\ \{\ e_2\ \}}) \to (\sigma;\ \boxed{e_2})}$$

E-MatchLeft
$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{match Left::<}\tau_1^{\text{SI}}\,,\ \tau_2^{\text{SI}}\texttt{>}(v)\ \ \{\ \texttt{Left}(x_1) \Rightarrow e_1\,,\ \texttt{Right}(x_2) \Rightarrow e_2\ \}}) \to (\sigma\,,\ x_1 \mapsto v;\ \boxed{\texttt{shift}\ e_1})}$$

E-MatchRight
$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{match Right::<}\tau_1^{\text{SI}}\,,\ \tau_2^{\text{SI}}\texttt{>}(v)\ \ \{\ \texttt{Left}(x_1) \Rightarrow e_1\,,\ \texttt{Right}(x_2) \Rightarrow e_2\ \}}) \to (\sigma\,,\ x_2 \mapsto v;\ \boxed{\texttt{shift}\ e_2})}$$

E-LetRegion
$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{letrgn}\ \texttt{<}r\texttt{>}\ \{\ v\ \}}) \to (\sigma;\ \boxed{v})}$$

E-Let
$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{let}\ x : \tau_a^{\text{SI}} = v;\ e}) \to (\sigma\,,\ x \mapsto v;\ \boxed{\texttt{shift}\ e})}$$

E-Seq
$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{v;\ e}) \to (\sigma;\ \boxed{e})}$$

E-Assign
$$\frac{\sigma \vdash p \Downarrow \mathcal{R} \mapsto \mathcal{V}[\_] \qquad \mathcal{R} = \mathcal{R}^{\square}[x]}{\Sigma \vdash (\sigma;\ \boxed{p \coloneqq v}) \to (\sigma[x \mapsto \mathcal{V}[v]];\ \boxed{(\,)})}$$

E-While
$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{while}\ e_1\ \{\ e_2\ \}}) \to (\sigma;\ \boxed{\texttt{if}\ e_1\ \{\ e_2;\ \texttt{while}\ e_1\ \{\ e_2\ \}\ \}\ \texttt{else}\ \{\ (\,)\ \}})}$$

E-ForArray
$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{for}\ x\ \texttt{in}\ [v_0\,,\ \ldots\ ,\ v_n]\ \{\ e\ \}}) \to (\sigma\,,\ x \mapsto v_0;\ \boxed{\texttt{shift}\ e;\ \texttt{for}\ x\ \texttt{in}\ [v_1\,,\ \ldots\ ,\ v_n]\ \{\ e\ \}})}$$

E-ForSlice
$$\frac{\sigma \vdash \mathcal{R} \Downarrow \_ \mapsto \_[[v_1\,,\ \ldots\ ,\ v_i\,,\ \ldots\ ,\ v_j\,,\ \ldots\ ,\ v_n]] \qquad i < j \qquad i' = i + 1}{\Sigma \vdash (\sigma;\ \boxed{\texttt{for}\ x\ \texttt{in}\ \texttt{ptr}\ \mathcal{R}[i..j]\ \{\ e\ \}}) \to (\sigma\,,\ x \mapsto \texttt{ptr}\ \mathcal{R}[i];\ \boxed{\texttt{shift}\ e;\ \texttt{for}\ x\ \texttt{in}\ \texttt{ptr}\ \mathcal{R}[i'..j]\ \{\ e\ \}})}$$

E-ForEmptyArray
$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{for}\ x\ \texttt{in}\ [\,]\ \{\ e\ \}}) \to (\sigma;\ \boxed{(\,)})}$$

E-ForEmptySlice
$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{for}\ x\ \texttt{in}\ \texttt{ptr}\ \pi[n..n]\ \{\ e\ \}}) \to (\sigma;\ \boxed{(\,)})}$$

E-Closure
$$\frac{\overline{x_f} = \text{free-vars}(e) \qquad \overline{x_{nc}} = \text{free-nc-vars}_{\sigma}(e) \qquad \varsigma_c = \sigma \mid_{\overline{x_f}}}{\Sigma \vdash (\sigma;\ \boxed{|x_1 : \tau_1^{\text{s}}\,,\ \ldots\ ,\ x_n : \tau_n^{\text{s}}|\ \to\ \tau_r^{\text{s}}\ \{\ e\ \}}) \to (\sigma[\overline{x_{nc} \mapsto \text{dead}}];\ \boxed{\langle \varsigma_c\,,\ \ |x_1 : \tau_1^{\text{s}}\,,\ \ldots\ ,\ x_n : \tau_n^{\text{s}}|\ \to\ \tau_r^{\text{s}}\ \{\ e\ \}\ \rangle})}$$

$$\textsc{E-AppClosure}\quad \dfrac{v_f = \langle \varsigma_c,\ \ |x_1 : \tau^{\mathsf{s}}_1,\ \ldots,\ x_n : \tau^{\mathsf{s}}_n| \ \rightarrow\ \tau^{\mathsf{s}}_r\ \{\ e\ \}\ \rangle}{\Sigma \vdash (\sigma;\ \boxed{v_f(v_1,\ \ldots,\ v_n)}\ ) \longrightarrow (\sigma \natural \varsigma_c,\ x_1 \mapsto v_1,\ \ldots,\ x_n \mapsto v_n;\ \boxed{\texttt{framed}\ e}\ )}$$

$$\textsc{E-AppFunction}\quad \dfrac{\Sigma(f) = \texttt{fn}\ f\texttt{<}\overline{\varphi},\ \overline{\varrho},\ \overline{\alpha}\texttt{>}(x_1 : \tau^{\mathsf{s}}_1,\ \ldots,\ x_n : \tau^{\mathsf{s}}_n)\ \rightarrow\ \tau^{\mathsf{s}}_r\ \texttt{where}\ \overline{\varrho : \varrho'}\ \{\ e\ \}}{\Sigma \vdash (\sigma;\ \boxed{f\texttt{::<}\overline{\Phi},\ \overline{r'},\ \overline{\tau^{\mathsf{s}}}\texttt{>}(v_1,\ \ldots,\ v_n)}\ ) \longrightarrow (\sigma \natural x_1 \mapsto v_1,\ \ldots,\ x_n \mapsto v_n;\ \boxed{\texttt{framed}\ e[^{\overline{\Phi}}/_{\overline{\varphi}}][^{\overline{r'}}/_{\overline{\varrho}}][^{\overline{\tau^{\mathsf{s}}}}/_{\overline{\alpha}}]}\ )}$$

$$\textsc{E-EvalCtx}\quad \dfrac{\Sigma \vdash (\sigma;\ \boxed{e}\ ) \longrightarrow (\sigma';\ \boxed{e'}\ )}{\Sigma \vdash (\sigma;\ \boxed{C[e]}\ ) \longrightarrow (\sigma';\ \boxed{C[e']}\ )}$$

$$\textsc{E-EvalCtxAbort}\quad \dfrac{}{\Sigma \vdash (\sigma;\ \boxed{C[\texttt{abort!(str)}]}\ ) \longrightarrow (\sigma;\ \boxed{\texttt{abort!(str)}}\ )}$$

E METATHEORY

E.1 Standard Lemmas

LEMMA E.1 (CANONICAL FORMS). *If* $\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau \Rightarrow \Gamma$ *then*

(1) *if* $\tau = \texttt{bool}$, *then* $v = \texttt{true}$ *or* $v = \texttt{false}$.
(2) *if* $\tau = \texttt{u32}$, *then* $v = n$.
(3) *if* $\tau = \texttt{unit}$, *then* $v = (\,)$.
(4) *if* $\tau = \&\rho\ \omega\ \tau^{SI}$, *then* v *is of the form* $\texttt{ptr}\ \mathcal{R}$.
(5) *if* $\tau = \&\rho\ \omega\ [\tau^{SI}]$, *then* v *is of the form* $\texttt{ptr}\ \mathcal{R}[i..j]$.
(6) *if* $\tau = \forall{<}\overline{\varphi},\ \overline{\varrho},\ \overline{\alpha}{>}(\tau_1^{SI},\ \dots,\ \tau_n^{SI}) \rightarrow \tau_r^{SI}$ $\texttt{where}\ \overline{\varrho_1 : \varrho_2}$, *then* v *is of the form* f.
(7) *if* $\tau = (\tau_1^{SI},\ \dots,\ \tau_n^{SI}) \overset{\mathcal{F}}{\rightarrow} \tau_r^{SI}$, *then* v *is of the form* $\langle \sigma,\ |x_1 : \tau_1^{SI},\ \dots,\ x_n : \tau_n^{SI}| \rightarrow \tau_r^{SI}\ \{\ e\ \}\ \rangle$.
(8) *if* $\tau = [\tau';\ n]$, *then* v *is of the form* $[v_1,\ \dots,\ v_n]$.
(9) *if* $\tau = [\tau']$, *then* v *is of the form* $[\![v_1,\ \dots,\ v_n]\!]$.
(10) *if* $\tau = (\tau_1,\ \dots,\ \tau_n)$, *then* v *is of the form* $(v_1,\ \dots,\ v_n)$.
(11) *if* $\tau = \texttt{Either}{<}\tau_1,\ \tau_2{>}$, *then* v *is of either the form* $\texttt{Left}{::}{<}\tau_1,\ \tau_2{>}(v')$ *or* $\texttt{Right}{::}{<}\tau_1,\ \tau_2{>}(v')$.

PROOF. By inspection of the grammar of values and typing rules. □

LEMMA E.2 (PRESERVATION OF TYPES UNDER SUBSTITUTION).

(1) *If* $\Sigma;\ \Delta,\ \alpha : \star;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau \Rightarrow \Gamma'$ *and* $\Sigma;\ \Delta;\ \Gamma \vdash \tau'$, *then* $\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e[^{\tau'}/_{\alpha}]} : \tau[^{\tau'}/_{\alpha}] \Rightarrow \Gamma'[^{\tau'}/_{\alpha}]$
(2) *If* $\Sigma;\ \Delta,\ \varrho : \texttt{RGN};\ \Gamma;\ \Theta \vdash \boxed{e} : \tau \Rightarrow \Gamma'$ *and* $\Delta;\ \Gamma \vdash \rho$, *then* $\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e[^{\rho}/_{\varrho}]} : \tau[^{\rho}/_{\varrho}] \Rightarrow \Gamma'[^{\rho}/_{\varrho}]$
(3) *If* $\Sigma;\ \Delta,\ \varphi{:}\texttt{FRM};\ \Gamma;\ \Theta \vdash \boxed{e} : \tau \Rightarrow \Gamma'$ *and* $\Sigma;\ \Delta;\ \Gamma \vdash \Phi$, *then* $\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e[^{\Phi}/_{\varphi}]} : \tau[^{\Phi}/_{\varphi}] \Rightarrow \Gamma'[^{\Phi}/_{\varphi}]$

PROOF. By induction on the typing derivation. □

E.2 Referent Lemmas

LEMMA E.3 (WELL-FORMED REFERENCES EVALUATE TO WELL-TYPED VALUES). *If* $\Sigma;\ \Gamma \vdash \mathcal{R} : \tau^{XI}$ *and* $\Sigma \vdash \sigma : \Gamma$, *then* $\sigma \vdash \mathcal{R} \Downarrow \mathcal{V} \times v$.

PROOF. We proceed by induction on $\Sigma;\ \Gamma \vdash \mathcal{R} : \tau^{\text{XI}}$. There are six cases: WF-REFID, WF-REFPROJ, WF-REFINDEXARRAY, WF-REFINDEXSLICE, WF-REFSLICEARRAY, and WF-REFSLICESLICE. Each of these cases has a corresponding evaluation rule:

WF-REFID
$$\frac{\Gamma(x) = \tau^{\text{SI}}}{\Sigma;\ \Gamma \vdash x : \tau^{\text{SI}}}$$

ER-ID
$$\frac{\sigma(x) = v}{\sigma \vdash x \Downarrow \square \times v}$$

For the base case, we consider the frame of Γ which contains x. By inversion of WF-STACKFRAME for the portion of the derivation $\Sigma \vdash \sigma : \Gamma$ pertaining to that frame, we have $\forall x \in \text{dom}(\varsigma).\ \Sigma;\ \bullet;\ \Gamma \natural \mathcal{F};\ \bullet \vdash \boxed{(\sigma \natural \varsigma)(x)} : (\Gamma \natural \mathcal{F})(x) \Rightarrow \Gamma \natural \mathcal{F}$. Focusing on our particular x, we have both that $\sigma(x) = v$ and that $\Sigma;\ \bullet;\ \Gamma \natural \mathcal{F};\ \bullet \vdash \boxed{v} : (\Gamma \natural \mathcal{F})(x) \Rightarrow \Gamma \natural \mathcal{F}$, finishing the case. The remaining cases follow:

WF-RefProjection
$$\frac{\Sigma;\ \Gamma \vdash \mathcal{R} : (\tau_0^{\text{SI}},\ \dots,\ \tau_i^{\text{SI}},\ \dots,\ \tau_n^{\text{SI}})}{\Sigma;\ \Gamma \vdash \mathcal{R}.i : \tau_i^{\text{SI}}}$$

ER-Projection
$$\frac{\sigma \vdash \mathcal{R} \Downarrow \mathcal{V} \times (v_0,\ \dots,\ v_i,\ \dots,\ v_n)}{\sigma \vdash \mathcal{R}.i \Downarrow \mathcal{V}[(v_0,\ \dots,\ \square,\ \dots,\ v_n)] \times v_i}$$

WF-RefIndexArray
$$\frac{\Sigma;\ \Gamma \vdash \mathcal{R} : [\tau^{\text{SI}};\ n] \qquad 0 \le i < n}{\Sigma;\ \Gamma \vdash \mathcal{R}[i] : \tau^{\text{SI}}}$$

ER-IndexArray
$$\frac{\sigma \vdash \mathcal{R} \Downarrow \mathcal{V} \times [v_0,\ \dots,\ v_i,\ \dots,\ v_n]}{\sigma \vdash \mathcal{R}[i] \Downarrow \mathcal{V}[[v_0,\ \dots,\ \square,\ \dots,\ v_n]] \times v_i}$$

WF-RefIndexSlice
$$\frac{\Sigma;\ \Gamma \vdash \mathcal{R} : [\tau^{\text{SI}}]}{\Sigma;\ \Gamma \vdash \mathcal{R}[i] : \tau^{\text{SI}}}$$

ER-IndexSlice
$$\frac{\sigma \vdash \mathcal{R} \Downarrow \mathcal{V} \times [\![v_0,\ \dots,\ v_k,\ \dots,\ v_n]\!]}{\sigma \vdash \mathcal{R}[k] \Downarrow \mathcal{V}[v_0]\ \dots\ [\square]\ \dots\ [v_n] \times v_{i+k}}$$

WF-RefSliceArray
$$\frac{\Sigma;\ \Gamma \vdash \mathcal{R} : [\tau^{\text{SI}};\ n] \qquad 0 \le i \le j < n}{\Sigma;\ \Gamma \vdash \mathcal{R}[i..j] : [\tau^{\text{SI}}]}$$

ER-SliceArray
$$\frac{\sigma \vdash \mathcal{R} \Downarrow \mathcal{V} \times [v_0,\ \dots,\ v_i,\ \dots,\ v_j,\ \dots,\ v_n]}{\sigma \vdash \mathcal{R}[i..j] \Downarrow \mathcal{V}[[v_0,\ \dots,\ \square^{j-i+1},\ \dots,\ v_n]] \times [\![v_i,\ \dots,\ v_j]\!]}$$

WF-RefSliceSlice
$$\frac{\Sigma;\ \Gamma \vdash \mathcal{R} : [\tau^{\text{SI}}] \qquad i \le j}{\Sigma;\ \Gamma \vdash \mathcal{R}[i..j] : [\tau^{\text{SI}}]}$$

ER-SliceSlice
$$\frac{\sigma \vdash \mathcal{R} \Downarrow \mathcal{V} \times [\![v_0,\ \dots,\ v_i,\ \dots,\ v_j,\ \dots,\ v_n]\!]}{\sigma \vdash \mathcal{R}[i..j] \Downarrow \mathcal{V}[v_0]\ \dots\ [\square]\ \dots\ [\square]\ \dots\ [v_n] \times [\![v_i,\ \dots,\ v_j]\!]}$$

The proof for each case is identical: apply the induction hypothesis and then Lemma E.1 and then the evaluation rule on the right. For the well-typed portion, apply inversion on the typing rule for the appropriate value. □

Lemma E.4 (Place Expressions Reduce). *If* $\Delta;\ \Gamma \vdash_\omega p : \tau^{XI}$ *and* $\Sigma \vdash \sigma : \Gamma$, *then* $\sigma \vdash p \Downarrow \mathcal{R} \mapsto \mathcal{V}[v]$ *and* $\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau^{XI} \Rightarrow \Gamma$.

Proof. We proceed by induction on $\Delta;\ \Gamma \vdash_\omega p : \tau^{\text{XI}}$. There are three cases: TC-Var, TC-Proj, and TC-Deref.

TC-Var
$$\frac{\Gamma(x) = \tau^{\text{SI}}}{\Delta;\ \Gamma \vdash_\omega x : \tau^{\text{SI}},\ \emptyset}$$

P-Referent
$$\frac{\sigma \vdash \mathcal{R} \Downarrow \mathcal{V} \times v}{\sigma \vdash \square \times \mathcal{R} \Downarrow (\mathcal{R}, \mathcal{V}, v)}$$

For TC-Var, we consider the piece of the derivation for $\Sigma \vdash \sigma : \Gamma$ (from our premise) for the frame containing x. By inversion on WF-StackFrame, we have $\forall x \in \text{dom}(\varsigma).\ \Sigma;\ \bullet;\ \Gamma \natural \mathcal{F};\ \bullet \vdash \boxed{(\sigma \natural \varsigma)(x)} : (\Gamma \natural \mathcal{F})(x) \Rightarrow \Gamma \natural \mathcal{F}$. This immediately gives us that $\sigma(x) = v$ and that $\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau^{\text{XI}} \Rightarrow \Gamma$. To construct our premise for P-Referent, we apply ER-Id to $\sigma(x) = v$.

TC-Proj
$$\frac{\Delta;\ \Gamma \vdash_\omega p : (\tau_1^{\text{SI}},\ \dots,\ \tau_i^{\text{SI}},\ \dots,\ \tau_n^{\text{SI}}),\ \{\ \overline{\rho_p}\ \}}{\Delta;\ \Gamma \vdash_\omega p.i : \tau_i^{\text{SI}},\ \{\ \overline{\rho_p}\ \}}$$

P-Proj
$$\frac{\sigma \vdash p^\square \times \mathcal{R}_1 \Downarrow (\mathcal{R}_2, \mathcal{V}, (v_0,\ \dots,\ v_i,\ \dots,\ v_n))}{\sigma \vdash p^\square[\square.i] \times \mathcal{R}_1 \Downarrow (\mathcal{R}_2.i, \mathcal{V}[(v_0,\ \dots,\ \square,\ \dots,\ v_n)], v_i)}$$

For TC-Proj, we apply our induction hypothesis to $\Delta;\ \Gamma \vdash_\omega p : (\tau_1^{\text{SI}},\ \dots,\ \tau_i^{\text{SI}},\ \dots,\ \tau_n^{\text{SI}}),\ \{\overline{\rho_p}\}$ from the premise of TC-Proj and get $\sigma \vdash p \Downarrow \mathcal{R} \mapsto \mathcal{V}[v]$ and $\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{v} : (\tau_1^{\text{SI}},\ \dots,\ \tau_i^{\text{SI}},\ \dots,\ \tau_n^{\text{SI}}) \Rightarrow \Gamma$. Then, by Lemma E.1, we know that v must be of the form $(v_1,\ \dots,\ v_i,\ \dots,\ v_n)$. We can use this and the definition of $\sigma \vdash p \Downarrow \mathcal{R} \mapsto \mathcal{V}[v]$ to get $\sigma \vdash p^\square \times x \Downarrow (\mathcal{R}, \mathcal{V}, (v_1,\ \dots,\ v_i,\ \dots,\ v_n))$ (where $p^\square[x] = p$). This is precisely the premise of P-Proj and thus we can use that. We also have by inversion of T-Tuple for $\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{v} : (\tau_1^{\text{SI}},\ \dots,\ \tau_i^{\text{SI}},\ \dots,\ \tau_n^{\text{SI}}) \Rightarrow \Gamma$ that $\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma$.

$$\text{TC-Deref}\quad \frac{\Delta;\ \Gamma \vdash_\omega p : \&\rho\ \omega'\ \tau^{\text{XI}},\ \{\ \overline{\rho_p}\ \} \qquad \omega \lesssim \omega'}{\Delta;\ \Gamma \vdash_\omega *p : \tau^{\text{XI}},\ \{\ \overline{\rho_p},\ \rho\ \}}$$

For TC-Deref, we apply our induction hypothesis to $\Delta;\ \Gamma \vdash_\omega p : \&\rho\ \omega'\ \tau^{\text{XI}},\ \{\ \overline{\rho_p}\ \}$ to get $\sigma \vdash p \Downarrow \mathcal{R} \mapsto \mathcal{V}[v]$ and $\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{v} : \&\rho\ \omega'\ \tau^{\text{XI}} \Rightarrow \Gamma$. Then, by Lemma E.1, we know that v must of the form $\mathtt{ptr}\ \mathcal{R}$. We now have five subcases to consider depending on whether $\mathcal{R}$ is of π, $\mathcal{R}_3[i]$, or $\mathcal{R}[i..j]$, and for the latter two, whether τ^{XI} is $[\tau^{\text{SI}};\ n]$ or $[\tau^{\text{SI}}]$.

$$\text{P-DerefPtr}\quad \frac{\begin{array}{c}\sigma \vdash \square \times \mathcal{R}_1 \Downarrow (\_, \mathtt{ptr}\ \pi,) \\ \sigma \vdash p^\square \times \pi \Downarrow (\mathcal{R}_2, \mathcal{V}, v)\end{array}}{\sigma \vdash p^\square[*\square] \times \mathcal{R}_1 \Downarrow (\mathcal{R}_2, \mathcal{V}, v)}$$

$$\text{P-DerefIndexPtrArray}\quad \frac{\begin{array}{c}\sigma \vdash \square \times \mathcal{R}_1 \Downarrow (\_, \mathtt{ptr}\ \mathcal{R}_2[i],) \\ \sigma \vdash p^\square \times \mathcal{R}_2 \Downarrow (\mathcal{R}_3, \mathcal{V}, [v_0,\ \dots,\ v_i,\ \dots,\ v_n])\end{array}}{\sigma \vdash p^\square[*\square] \times \mathcal{R}_1 \Downarrow (\mathcal{R}_3[i], \mathcal{V}[[v_0,\ \dots,\ \square,\ \dots,\ v_n]], v_i)}$$

$$\text{P-DerefIndexPtrSlice}\quad \frac{\begin{array}{c}\sigma \vdash \square \times \mathcal{R}_1 \Downarrow (\_, \mathtt{ptr}\ \mathcal{R}_2[i],) \\ \sigma \vdash p^\square \times \mathcal{R}_2 \Downarrow (\mathcal{R}_3, \mathcal{V}, \llbracket v_0,\ \dots,\ v_i,\ \dots,\ v_n \rrbracket)\end{array}}{\sigma \vdash p^\square[*\square] \times \mathcal{R}_1 \Downarrow (\mathcal{R}_3[i], \mathcal{V}[v_0]\ \dots\ [\square]\ \dots\ [v_n], v_i)}$$

$$\text{P-DerefSlicePtrArray}\quad \frac{\begin{array}{c}\sigma \vdash \square \times \mathcal{R}_1 \Downarrow (\_, \mathtt{ptr}\ \mathcal{R}_2[i..j],) \\ \sigma \vdash p^\square \times \mathcal{R}_2 \Downarrow (\mathcal{R}_3, \mathcal{V}, [v_0,\ \dots,\ v_i,\ \dots,\ v_j,\ \dots,\ v_n])\end{array}}{\sigma \vdash p^\square[*\square] \times \mathcal{R}_1 \Downarrow (\mathcal{R}_3[i..j], \mathcal{V}[[v_0,\ \dots,\ \square^{j-i+1},\ \dots,\ v_n]], \llbracket v_i,\ \dots,\ v_j \rrbracket)}$$

$$\text{P-DerefSlicePtrSlice}\quad \frac{\begin{array}{c}\sigma \vdash \square \times \mathcal{R}_1 \Downarrow (\_, \mathtt{ptr}\ \mathcal{R}_2[i..j],) \\ \sigma \vdash p^\square \times \mathcal{R}_2 \Downarrow (\mathcal{R}_3, \mathcal{V}, \llbracket v_0,\ \dots,\ v_i,\ \dots,\ v_j,\ \dots,\ v_n \rrbracket)\end{array}}{\sigma \vdash p^\square[*\square] \times \mathcal{R}_1 \Downarrow (\mathcal{R}_3[i..j], \mathcal{V}[v_0]\ \dots\ [\square]\ \dots\ [\square]\ \dots\ [v_n], \llbracket v_i,\ \dots,\ v_j \rrbracket)}$$

In all these cases, we know structurally that $p^\square = \square$ since TC-Deref has no context outside of the dereference. So, for each of them, we need to be able to show $\square \vdash \mathcal{R} \Downarrow \mathcal{R}' \mapsto \mathcal{V}[v']$. Inversion on T-Pointer gives us $\Sigma;\ \Gamma \vdash \mathcal{R} : \tau^{\text{XI}}$. We can then apply Lemma E.3 to get $\Sigma \vdash \Gamma \Downarrow \mathcal{V} \times v$. Then, we can apply P-Referent to this to produce the derivation we need to apply the appropriate rule. For P-DerefIndexPtrArray and P-DerefSlicePtrArray, we apply Lemma E.1 to get that the value is an array. For P-DerefIndexPtrSlice and P-DerefSlicePtrSlice, we apply Lemma E.1 to get that the value is a slice value. □

Lemma E.5 (Reduced Place Expressions Produce Valid Referents). *If* $\Sigma \vdash \sigma : \Gamma$ *and* $\sigma \vdash p \Downarrow \mathcal{R}^\square[\pi] \mapsto \mathcal{V}[v]$, *then* $\Sigma;\ \Gamma \vdash \mathcal{R}^\square[\pi] : \tau^{\text{XI}}$.

Proof. We start by rewriting $\sigma \vdash p \Downarrow \mathcal{R}^\square[\pi] \mapsto \mathcal{V}[v]$ with its definition to get $\sigma \vdash p^\square \times x \Downarrow (\mathcal{R}^\square[\pi], \mathcal{V}, v)$ where $p = p^\square[x]$. We then proceed by induction by cases (note this means our induction hypothesis is really about the rewritten form).

$$\text{P-Referent}\quad \frac{\sigma \vdash \mathcal{R} \Downarrow \mathcal{V} \times v}{\sigma \vdash \square \times \mathcal{R} \Downarrow (\mathcal{R}, \mathcal{V}, v)} \qquad \text{WF-RefId}\quad \frac{\Gamma(x) = \tau^{\text{SI}}}{\Sigma;\ \Gamma \vdash x : \tau^{\text{SI}}}$$

P-Referent only applies if the context is $\square$ which is only the case if our original place expression was x. We can rewrite with this knowledge to see that we really have $\sigma \vdash x \Downarrow \_ \times v$ in our premise. Inversion on ER-Id gives us $\sigma(v) =$ Then, we consider the frame of Γ which contains x. By inversion of WF-StackFrame for the portion of the derivation $\Sigma \vdash \sigma : \Gamma$ pertaining to that frame, we have $\forall x \in \text{dom}(\varsigma).\ \Sigma;\ \bullet;\ \Gamma \natural \mathcal{F};\ \bullet \vdash \boxed{(\sigma \natural \varsigma)(x)} : (\Gamma \natural \mathcal{F})(x) \Rightarrow \Gamma \natural \mathcal{F}$. Focusing on our particular x, we have both that $\Gamma(x) = v$. We can then apply WF-RefId.

$$\text{P-Proj}\quad \frac{\sigma \vdash p^{\square} \times \mathcal{R}_1 \Downarrow (\mathcal{R}_2, \mathcal{V}, (v_0\,,\ \ldots\,,\ v_i\,,\ \ldots\,,\ v_n))}{\sigma \vdash p^{\square}[\square.i] \times \mathcal{R}_1 \Downarrow (\mathcal{R}_2.i, \mathcal{V}[(v_0\,,\ \ldots\,,\ \square,\ \ldots\,,\ v_n)], v_i)} \qquad \text{WF-RefProjection}\quad \frac{\Sigma;\ \Gamma \vdash \mathcal{R} : (\tau_0^{\text{SI}}\,,\ \ldots\,,\ \tau_i^{\text{SI}}\,,\ \ldots\,,\ \tau_n^{\text{SI}})}{\Sigma;\ \Gamma \vdash \mathcal{R}.i : \tau_i^{\text{SI}}}$$

Applying the induction hypothesis to $\sigma \vdash p^{\square} \times \mathcal{R}_1 \Downarrow (\mathcal{R}_2, \_, (v_0\,,\ \ldots\,,\ v_i\,,\ \ldots\,,\ v_n))$ gives us $\Sigma;\ \Gamma \vdash \mathcal{R}_2 : (\tau_0^{\text{SI}}\,,\ \ldots\,,\ \tau_i^{\text{SI}}\,,\ \ldots\,,\ \tau_n^{\text{SI}})$. We can then apply WF-RefProjection.

$$\text{P-DerefPtr}\quad \frac{\sigma \vdash \square \times \mathcal{R}_1 \Downarrow (\_, \mathtt{ptr}\ \pi,) \qquad \sigma \vdash p^{\square} \times \pi \Downarrow (\mathcal{R}_2, \mathcal{V}, v)}{\sigma \vdash p^{\square}[*\square] \times \mathcal{R}_1 \Downarrow (\mathcal{R}_2, \mathcal{V}, v)}$$

Applying the induction hypothesis to $\sigma \vdash p^{\square} \times \pi \Downarrow (\mathcal{R}_2, \_, v)$ gives us $\Sigma;\ \Gamma \vdash \mathcal{R}_2 : \tau^{\text{XI}}$.

$$\text{P-DerefIndexPtrArray}\quad \frac{\sigma \vdash \square \times \mathcal{R}_1 \Downarrow (\_, \mathtt{ptr}\ \mathcal{R}_2[i],) \qquad \sigma \vdash p^{\square} \times \mathcal{R}_2 \Downarrow (\mathcal{R}_3, \mathcal{V}, [v_0\,,\ \ldots\,,\ v_i\,,\ \ldots\,,\ v_n])}{\sigma \vdash p^{\square}[*\square] \times \mathcal{R}_1 \Downarrow (\mathcal{R}_3[i], \mathcal{V}[[v_0\,,\ \ldots\,,\ \square,\ \ldots\,,\ v_n]], v_i)} \qquad \text{WF-RefIndexArray}\quad \frac{\Sigma;\ \Gamma \vdash \mathcal{R} : [\tau^{\text{SI}};\ n] \qquad 0 \le i < n}{\Sigma;\ \Gamma \vdash \mathcal{R}[i] : \tau^{\text{SI}}}$$

Applying the induction hypothesis to $\sigma \vdash p^{\square} \times \mathcal{R}_2 \Downarrow (\mathcal{R}_3, \_, [v_0\,,\ \ldots\,,\ v_i\,,\ \ldots\,,\ v_n])$ gives us $\Sigma;\ \Gamma \vdash \mathcal{R}_3 : [\tau^{\text{SI}};\ n]$. Then, we can apply WF-RefIndexArray to get $\Sigma;\ \Gamma \vdash \mathcal{R}_3[i] : \tau^{\text{SI}}$.

$$\text{P-DerefIndexPtrSlice}\quad \frac{\sigma \vdash \square \times \mathcal{R}_1 \Downarrow (\_, \mathtt{ptr}\ \mathcal{R}_2[i],) \qquad \sigma \vdash p^{\square} \times \mathcal{R}_2 \Downarrow (\mathcal{R}_3, \mathcal{V}, [\![v_0\,,\ \ldots\,,\ v_i\,,\ \ldots\,,\ v_n]\!])}{\sigma \vdash p^{\square}[*\square] \times \mathcal{R}_1 \Downarrow (\mathcal{R}_3[i], \mathcal{V}[v_0]\ \ldots\ [\square]\ \ldots\ [v_n], v_i)} \qquad \text{WF-RefIndexSlice}\quad \frac{\Sigma;\ \Gamma \vdash \mathcal{R} : [\tau^{\text{SI}}]}{\Sigma;\ \Gamma \vdash \mathcal{R}[i] : \tau^{\text{SI}}}$$

Applying the induction hypothesis to $\sigma \vdash p^{\square} \times \mathcal{R}_2 \Downarrow (\mathcal{R}_3, \_, [\![v_0\,,\ \ldots\,,\ v_i\,,\ \ldots\,,\ v_n]\!])$ gives us $\Sigma;\ \Gamma \vdash \mathcal{R}_3 : [\tau^{\text{SI}}]$. Then, we can apply WF-RefIndexSlice to get $\Sigma;\ \Gamma \vdash \mathcal{R}_3[i] : \tau^{\text{SI}}$.

$$\text{P-DerefSlicePtrArray}\quad \frac{\sigma \vdash \square \times \mathcal{R}_1 \Downarrow (\_, \mathtt{ptr}\ \mathcal{R}_2[i..j],) \qquad \sigma \vdash p^{\square} \times \mathcal{R}_2 \Downarrow (\mathcal{R}_3, \mathcal{V}, [v_0\,,\ \ldots\,,\ v_i\,,\ \ldots\,,\ v_j\,,\ \ldots\,,\ v_n])}{\sigma \vdash p^{\square}[*\square] \times \mathcal{R}_1 \Downarrow (\mathcal{R}_3[i..j], \mathcal{V}[[v_0\,,\ \ldots\,,\ \square^{j-i+1}\,,\ \ldots\,,\ v_n]], [\![v_i\,,\ \ldots\,,\ v_j]\!])}$$

$$\text{WF-RefSliceArray}\quad \frac{\Sigma;\ \Gamma \vdash \mathcal{R} : [\tau^{\text{SI}};\ n] \qquad 0 \le i \le j < n}{\Sigma;\ \Gamma \vdash \mathcal{R}[i..j] : [\tau^{\text{SI}}]}$$

Applying the induction hypothesis to $\sigma \vdash p^{\square} \times \mathcal{R}_2 \Downarrow (\mathcal{R}_3, \_, [v_0\,,\ \ldots\,,\ v_i\,,\ \ldots\,,\ v_j\,,\ \ldots\,,\ v_n])$ gives us $\Sigma;\ \Gamma \vdash \mathcal{R}_3 : [\tau^{\text{SI}};\ n]$. Then, we can apply WF-RefSliceArray to get $\Sigma;\ \Gamma \vdash \mathcal{R}_3[i..j] : \tau^{\text{SI}}$.

P-DerefSlicePtrSlice

$$\frac{\sigma \vdash \square \times \mathcal{R}_1 \Downarrow (\_, \texttt{ptr}\ \mathcal{R}_2[i..j],) \qquad \sigma \vdash p^{\square} \times \mathcal{R}_2 \Downarrow (\mathcal{R}_3, \mathcal{V}, [\![v_0\,,\ \ldots\,,\ v_i\,,\ \ldots\,,\ v_j\,,\ \ldots\,,\ v_n]\!])}{\sigma \vdash p^{\square}[*\square] \times \mathcal{R}_1 \Downarrow (\mathcal{R}_3[i..j], \mathcal{V}[v_0]\ \ldots\ [\square]\ \ldots\ [\square]\ \ldots\ [v_n], [\![v_i\,,\ \ldots\,,\ v_j]\!])}$$

WF-RefSliceSlice

$$\frac{\Sigma; \Gamma \vdash \mathcal{R} : [\tau^{\text{SI}}] \qquad i \leq j}{\Sigma; \Gamma \vdash \mathcal{R}[i..j] : [\tau^{\text{SI}}]}$$

Applying the induction hypothesis to $\sigma \vdash p^{\square} \times \mathcal{R}_2 \Downarrow (\mathcal{R}_3, \_, [\![v_0\,,\ \ldots\,,\ v_i\,,\ \ldots\,,\ v_j\,,\ \ldots\,,\ v_n]\!])$ gives us $\Sigma; \Gamma \vdash \mathcal{R}_3 : [\tau^{\text{SI}}]$. Then, we can apply WF-RefSliceSlice to get $\Sigma; \Gamma \vdash \mathcal{R}_3[i..j] : \tau^{\text{SI}}$. □

Lemma E.6 (Reduced Place Expressions Have Roots in Loan Sets). *If $\Sigma \vdash \sigma : \Gamma$, $\sigma \vdash p \Downarrow \mathcal{R}^{\square}[\pi] \mapsto \mathcal{V}[v]$, and $\bullet; \Gamma; \Theta \vdash_{\omega} p \Rightarrow \{\, \overline{\ell}\, \}$, then $\mathcal{R} = \mathcal{R}^{\square}[\pi]$ and ${}^{\omega}\pi \in \{\, \overline{\ell}\, \}$.*

Proof. We proceed by induction on $\bullet; \Gamma; \Theta \vdash_{\omega} p \Rightarrow \{\, \overline{\ell}\, \}$. There are ordinarily three cases: O-SafePlace, O-Deref, and O-DerefAbs. However, O-DerefAbs requires the type variable context to contain entries, and thus can be immediately discharged by contradiction. This leaves us with only O-SafePlace and O-Deref.

O-SafePlace

$$\frac{\begin{array}{c}\forall r' \mapsto \{\, \overline{\ell}\, \} \in \text{regions}(\Gamma, \Theta).\ (\forall\, {}^{\omega'}p^{\square}[\pi'] \in \{\, \overline{\ell}\, \}.(\omega = \texttt{uniq} \vee \omega' = \texttt{uniq}) \implies \pi' \mathbin{\#} \pi) \\ \vee\ (\exists \pi' : \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \wedge \nexists \&r'\ \omega'\ \tau' \in \Theta\ \wedge (\forall \pi' : \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\, \overline{\pi_e}\, \}))\end{array}}{\Delta; \Gamma; \Theta \vdash_{\omega}^{\overline{\pi_e}} \pi \Rightarrow \{\ {}^{\omega}\pi\ \}}$$

O-SafePlace tells us that our p is in fact a place π meaning that it does not contain any dereferences. As such, we know that $\sigma \vdash p \Downarrow \mathcal{R}^{\square}[\pi] \mapsto \mathcal{V}[v]$ must have been derived using a combination of P-Referent and P-Proj corresponding to the structure of π. The resulting referent in such a case is precisely π (meaning $\mathcal{R}^{\square} = \square$), which we know is in the output immediately from the definition of O-SafePlace.

O-Deref

$$\frac{\begin{array}{c}\Gamma(\pi) = \&r\ \omega_{\pi}\ \tau_{\pi} \qquad \Gamma(r) = \{\, \overline{{}^{\omega'}p}^{\,n}\, \} \qquad \text{excl} = \{\, \pi_j \text{ where } j \in \{\, 1, \ldots\ n\, \} \mid p_j = p_j^{\square}[*\pi_j]\, \} \qquad \omega \lesssim \omega_{\pi} \\ \forall i \in \{\, 1\ \ldots\ n\, \}.\ \Delta; \Gamma; \Theta \vdash_{\omega}^{\overline{\pi_e}, \text{excl}, \pi} p^{\square}[p_i] \Rightarrow \{\, \overline{{}^{\omega}p_i'}\, \} \\ \forall r' \mapsto \{\, \overline{\ell}\, \} \in \text{regions}(\Gamma, \Theta).\ (\forall\, {}^{\omega'}p'' \in \{\, \overline{\ell}\, \}.(\omega = \texttt{uniq} \vee \omega' = \texttt{uniq}) \implies p'' \mathbin{\#} p^{\square}[*\pi]) \\ \vee\ (\exists \pi' : \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \wedge \nexists \&r'\ \omega'\ \tau' \in \Theta\ \wedge (\forall \pi' : \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\, \overline{\pi_e}, \text{excl}, \pi\, \}))\end{array}}{\Delta; \Gamma; \Theta \vdash_{\omega}^{\overline{\pi_e}} p^{\square}[*\pi] \Rightarrow \{\, \overline{{}^{\omega}p_1'}, \ \ldots\ \overline{{}^{\omega}p_n'},\ {}^{\omega}p^{\square}[*\pi]\, \}}$$

In the premise of O-Deref, we have a number of ownership safety derivations corresponding to each of the loans for the pointer being dereferenced. Since we know we have a dereference, we know that we must have derived $\sigma \vdash p \Downarrow \mathcal{R}^{\square}[\pi] \mapsto \mathcal{V}[v]$ using one of the five dereference rules at the appropriate point (P-DerefPtr, P-DerefIndexPtrArray, P-DerefIndexPtrSlice, P-DerefSlicePtrArray, and P-DerefSlicePtrSlice). Each of which share a common premise (at least when sufficiently generalized): $\sigma \vdash p^{\square} \times \mathcal{R}_2 \Downarrow (\mathcal{R}_3, \mathcal{V}, v)$. Here, $\mathcal{R}_2$ corresponds to the referent of the pointer we are dereferencing. As such, we know that one of the derivations of ownership safety corresponds to that particular referent. So, we can apply our induction hypothesis and get that ${}^{\omega}\pi \in \{\, \overline{{}^{\omega}p_i'}\, \}$ for the appropriate ownership safety derivation numbered i. The final output is the union of all of these sets, and thus we can generalize to ${}^{\omega}\pi \in \{\, \overline{{}^{\omega}p_1'}\,,\ \ldots\,,\ \overline{{}^{\omega}p_n'}\,,\ \ldots\,,\ {}^{\omega}p^{\square}[*\pi]\, \}$. □

E.3 Preservation under Region Rewriting Lemmas

Lemma E.7 (Ownership Safety is Preserved under Region Rewriting). *If $\bullet; \Gamma; \Theta \vdash_{\omega}^{\overline{\pi_e}} p \Rightarrow \{\, \overline{\ell'}\, \}$ and $\bullet; \Gamma; \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma'$ and $\{\, \overline{\pi_e}\, \} \subseteq \{\, \overline{\pi_e'}\, \}$ then $\bullet; \Gamma'; \Theta \vdash_{\omega}^{\overline{\pi_e}} p \Rightarrow \{\, \overline{\ell''}\, \}$.*

Proof. We proceed by induction on the region rewriting judgement. We note that if μ is $=$, then by inspection of the outlives judgement, $\Gamma = \Gamma'$, so the proof follows immediately from the premise. So consider when μ is $+$. The only case that doesn't follow immediately by induction and application of premises is RR-Reference, and in this case the only interesting part of the proof is the outlives constraint.

Proceeding by induction on the outlives constraint, the only interesting case is OL-CombineConcrete.

OL-CombineConcrete

$$\frac{\begin{array}{ccc}\Gamma \vdash r_1\ \texttt{rnrb} & \Gamma \vdash r_2\ \texttt{rnrb} & \Gamma;\ \Theta \vdash \{\, r_1,\ r_2 \,\}\ \texttt{clrs} \\ \multicolumn{2}{c}{r_1\ \texttt{occurs before}\ r_2\ \texttt{in}\ \Gamma} & \{\, \overline{\ell} \,\} = \Gamma(r_1) \cup \Gamma(r_2)\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash^{+} r_1 :> r_2 \dashv \Gamma[r_2 \mapsto \{\, \overline{\ell} \,\}]}$$

We want to show that $\Delta;\ \Gamma[r_2 \mapsto \{\overline{\ell}\}];\ \Theta \vdash_{\omega}^{\overline{\pi_e}} p \Rightarrow \{\overline{\ell''}\}$. Proceed by induction on the ownership safety judgement in the premise.

O-SafePlace

$$\frac{\begin{array}{c}\forall r' \mapsto \{\, \overline{\ell} \,\} \in \text{regions}(\Gamma,\ \Theta).\ (\forall\, {}^{\omega'}p^{\square}[\pi'] \in \{\, \overline{\ell} \,\}.(\omega = \texttt{uniq} \lor \omega' = \texttt{uniq}) \implies \pi' \,\#\, \pi) \\ \lor\ (\exists \pi' : \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \land \nexists \&r'\ \omega'\ \tau' \in \Theta\ \land (\forall \pi' : \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\, \overline{\pi_e} \,\}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi_e}} \pi \Rightarrow \{\ {}^{\omega}\pi\ \}}$$

Let r' be an arbitrary region. There are two cases to prove, depending which part of the disjunction is true for the premise $\Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi_e}} \pi \Rightarrow \{\, \overline{\ell'} \,\}$.

If the first part was true, then we need to show that $\forall\, {}^{\omega'}p^{\square}[\pi'] \in \Gamma[r_2 \mapsto \{\, \overline{\ell''} \,\}](r').\ (\omega = \texttt{uniq} \lor \omega' = \texttt{uniq}) \implies \pi' \,\#\, \pi$. This is only interesting when $r' = r_2$. Using the fact that $\Gamma[r_2 \mapsto \{\, \overline{\ell} \,\}](r_2) = \Gamma(r_1) \cup \Gamma(r_2)$, we need to show $\forall\, {}^{\omega'}p^{\square}[\pi'] \in \Gamma(r_1) \cup \Gamma(r_2).\ (\omega = \texttt{uniq} \lor \omega' = \texttt{uniq}) \implies \pi' \,\#\, \pi$. This is immediate if we can show that r_1 and r_2 are not excluded. This is immediate from the region not reborrowed judgement. For r_1 or r_2 to be excluded, for each reference π'' that has r_1 or r_2, there would have to be a loan of the form $p^{\square}[*\pi'']$, but such loans are precisely what the region not reborrowed judgement excludes.

If the second part was true, then we can prove the second part immediately from the hypothesis, because the types of references are unchanged, Θ is unchanged, and the exclusion list can only grow.

O-Deref

$$\frac{\begin{array}{c}\Gamma(\pi) = \&r\ \omega_\pi\ \tau_\pi \qquad \Gamma(r) = \{\ \overline{{}^{\omega'}p}^{\,n}\ \} \qquad \text{excl} = \{\, \pi_j \text{ where } j \in \{\, 1,\ \ldots\ n \,\} \mid p_j = p_j^{\square}[*\pi_j] \,\} \qquad \omega \lesssim \omega_\pi \\ \forall i \in \{\, 1\ \ldots\ n \,\}.\ \Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi_e},\, \text{excl},\, \pi} p^{\square}[p_i] \Rightarrow \{\ \overline{{}^{\omega}p_i'}\ \} \\ \forall r' \mapsto \{\, \overline{\ell} \,\} \in \text{regions}(\Gamma,\ \Theta).\ (\forall\, {}^{\omega'}p'' \in \{\, \overline{\ell} \,\}.(\omega = \texttt{uniq} \lor \omega' = \texttt{uniq}) \implies p'' \,\#\, p^{\square}[*\pi]) \\ \lor\ (\exists \pi' : \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \land \nexists \&r'\ \omega'\ \tau' \in \Theta\ \land (\forall \pi' : \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\, \overline{\pi_e},\ \text{excl},\ \pi \,\}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi_e}} p^{\square}[*\pi] \Rightarrow \{\ \overline{{}^{\omega}p_1'}\ ,\ \ldots\ \overline{{}^{\omega}p_n'}\ ,\ {}^{\omega}p^{\square}[*\pi]\ \}}$$

Firstly, note that the exclusion list will be equal if $r \neq r_2$, and will be potentially larger if $r = r_2$. Therefore we can immediately apply our induction hypothesis to get ownership safety for $p^{\square}[p_i]$ under $\Gamma[r_2 \mapsto \overline{\ell}]$.

For the rest of the case, apply identical reasoning to that in the O-SafePlace case.

$$\textsc{O-DerefAbs}\quad \frac{\begin{array}{c}\Gamma(\pi) = \&\varrho\ \omega_\pi\ \tau_\pi \qquad \Delta;\ \Gamma \vdash_\omega p^\square[*\pi] : \tau \qquad \omega \lesssim \omega_\pi \\ \forall r' \mapsto \{\,\overline{\ell}\,\} \in \text{regions}(\Gamma,\ \Theta).\ (\forall^{\omega'} p\ \in \{\,\overline{\ell}\,\}.(\omega = \mathtt{uniq} \vee \omega' = \mathtt{uniq}) \implies p \,\#\, p^\square[*\pi]) \\ \vee\ (\exists \pi' : \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \wedge \nexists \&r'\ \omega'\ \tau' \in \Theta\ \wedge (\forall \pi' : \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\,\overline{\pi_e},\ \pi\,\}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash_\omega^{\overline{\pi_e}} p^\square[*\pi] \Rightarrow \{\ {}^\omega p^\square[*\pi]\ \}}$$

Since $\Delta = \bullet$, there are no valid reference types that have an abstract region, meaning the first hypothesis is a contradiction.

□

Lemma E.8 (Type Computation is Preserved under Region Rewriting). *If* $\bullet;\ \Gamma \natural \mathcal{F} \vdash_\omega p : \tau^{SI}$ *and* $\bullet;\ \Gamma;\ \Theta \vdash^\mu \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma'$ *then* $\bullet;\ \Gamma' \natural \mathcal{F} \vdash_\omega p : \tau^{SI}$.

Proof. The proof is immediate by inspection of the type computation judgement, because the only things considered in the judgement are the types of places in Γ, which cannot change through the region rewriting judgement (in other words, $\text{dom}(\Gamma) = \text{dom}(\Gamma')$). □

Lemma E.9 (Outlives is Preserved under Region Rewriting). *If* $\bullet;\ \Gamma \natural \mathcal{F};\ \Theta \vdash^\mu r_1 :> r_2 \dashv \Gamma_o \natural \mathcal{F}_o$ *and* $\bullet;\ \Gamma;\ \Theta \vdash^\mu \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma'$ *then* $\bullet;\ \Gamma' \natural \mathcal{F};\ \Theta \vdash^\mu r_1 :> r_2 \dashv \Gamma_o' \natural \mathcal{F}_o'$.

Proof. We proceed by induction on the outlives judgement. The only interesting cases are OL-CombineConcrete and OL-CheckConcrete. In both cases, the only non immediate premise is the region not reborrowed judgement. Proceed by induction over the region rewriting hypothesis, and in the interesting case RR-Reference, proceed by induction over the outlives judgement. In this case, the only interesting case is when the loan sets in Γ potentially change, OL-CombineConcrete. But note that no new loans are generated, only loans are copied into other sets. For this reason, the region not reborrowed judgements we already have are sufficient, because these loans that are now potentially in two loan sets were already found to not contain any problematic reborrows. □

Lemma E.10 (Region Rewriting is Preserved under Region Rewriting). *If* $\bullet;\ \Gamma \natural \mathcal{F};\ \Theta \vdash^\mu \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma_o \natural \mathcal{F}_o$ *and* $\bullet;\ \Gamma;\ \Theta \vdash^\mu \tau_1' \rightsquigarrow \tau_2' \dashv \Gamma'$ *then* $\bullet;\ \Gamma' \natural \mathcal{F};\ \Theta \vdash^\mu \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma_o' \natural \mathcal{F}_o'$.

Proof. Proceed by induction over the the region rewriting judgement. The only interesting case is RR-Reference, for which we just apply Lemma E.9. □

Lemma E.11 (Region Rewriting is Preserved by Garbage Collecting Loans). *If* $\bullet;\ \Gamma;\ \Theta \vdash^\mu \tau_1' \rightsquigarrow \tau_2' \dashv \Gamma'$ *then* $\bullet;\ \textit{gc-loans}_\Theta(\Gamma);\ \Theta \vdash^\mu \tau_1' \rightsquigarrow \tau_2' \dashv \Gamma''$.

Proof. We proceed by induction over the region rewriting judgement, in which the only interesting case is RR-Reference. We then proceed by induction over the outlives relation, in which the only interesting cases are OL-CombineConcrete and OL-CheckConcrete. The only interesting part of the judgement is the region not reborrowed, and this is immediate because garbage collection will only potentially remove some loans. □

Lemma E.12 (Closure Body Typing is Preserved under Region Rewriting). *If* $\Sigma;\ \bullet;\ \Gamma \natural \mathcal{F};\ \Theta \vdash \boxed{e} : \tau \Rightarrow \Gamma_o \natural \mathcal{F}_o$ *and* $\bullet;\ \Gamma;\ \Theta \vdash^\mu \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma'$ *then* $\Sigma;\ \bullet;\ \Gamma' \natural \mathcal{F};\ \Theta \vdash \boxed{e} : \tau \Rightarrow \Gamma_o' \natural \mathcal{F}_o'$ *and* $\bullet;\ \Gamma_o';\ \Theta \vdash^\mu \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma_o''$.

Proof. Proceed by induction over the typing derivation for e.

The T-Abort, T-Function, T-Unit, T-u32, T-True, and T-False cases follow immediately.

The T-LetRegion, T-While, T-Closure, T-Tuple, T-Array, T-Slice, T-Drop, T-Left, and T-Right cases all follow immediately from the induction hypothesis.

The T-Seq case follows from the induction hypothesis and Lemma E.11.

The T-Branch, T-Let, and T-Match cases follow from the induction hypothesis, Lemma E.10, and Lemma E.11. Note the reborrow restriction follows immediately from the fact that rewriting can at most union together loan sets, which means the overall loans considered for the region not reborrowed judgement are the same in the context after rewriting.

The T-Move, T-Copy, T-Borrow, T-BorrowIndex, T-BorrowSlice, T-IndexCopy, T-ForArray, and T-ForSlice cases follow from the induction hypothesis, Lemma E.7, and Lemma E.8.

The T-AppFunction and T-AppClosure cases follow from the induction hypothesis, Lemma E.9, and the fact that context, region, and type well formedness aren't affected by changes in the loan sets. □

Lemma E.13 (Value Typing is Preserved under Region Rewriting). *If* $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau \Rightarrow \Gamma$ *and* $\bullet;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma'$ *then* $\Sigma;\ \bullet;\ \Gamma';\ \Theta \vdash \boxed{v} : \tau \Rightarrow \Gamma'$.

Proof. We proceed by induction on the value typing.

T-Pointer
$$\frac{\Sigma;\ \Gamma \vdash \mathcal{R}^{\square}[\pi] : \tau^{\text{XI}} \qquad {}^{\omega}\pi \in \Gamma(r)}{\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{\texttt{ptr}\ \mathcal{R}^{\square}[\pi]} : \&r\ \omega\ \tau^{\text{XI}} \Rightarrow \Gamma}$$

The T-Pointer case is immediate, because by inspection of the referent well formedness, there is no reliance on loan sets, and the loan is preserved since by inspection of the rewriting judgement, the loan sets either stay the same or potentially grow.

T-ClosureValue
$$\frac{\begin{array}{c}\text{free-vars}(e) \setminus \overline{x} = \overline{x_f} = \text{dom}(\mathcal{F}_c)|_{\text{VAR}} \\ \overline{r} = \overline{\text{free-regions}(\Gamma(x_f))},\ (\text{free-regions}(e) \setminus (\overline{\text{free-regions}(\tau^{\text{SI}})}, \text{free-regions}(\tau_r^{\text{SI}}))) = \text{dom}(\mathcal{F}_c)|_{\text{RGN}} \\ \Sigma;\ \Gamma \vdash \varsigma_c : \mathcal{F}_c \qquad \Sigma;\ \Delta;\ \Gamma \natural \mathcal{F}_c,\ x_1 : \tau_1^{\text{SI}}, \ldots, x_n : \tau_n^{\text{SI}};\ \Theta \vdash \boxed{e} : \tau_r^{\text{SI}} \Rightarrow \Gamma' \natural \mathcal{F}\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\langle \varsigma_c,\ |x_1 : \tau_1^{\text{SI}}, \ldots, x_n : \tau_n^{\text{SI}}| \to \tau_r^{\text{SI}}\ \{\, e\, \}\, \rangle} : (\tau_1^{\text{SI}}, \ldots, \tau_n^{\text{SI}}) \xrightarrow{\mathcal{F}_c} \tau_r^{\text{SI}} \Rightarrow \Gamma}$$

First, we invert the stack frame typing hypothesis to get that $\forall x \in \text{dom}(\varsigma).\ \Sigma;\ \bullet;\ \Gamma \natural \mathcal{F}_c;\ \Theta \vdash \boxed{\varsigma(x)} : \mathcal{F}_c(x) \Rightarrow \Gamma \natural \mathcal{F}_c$. We can apply the induction hypothesis to each of these statements, and apply WF-Frame to get $\Sigma;\ \Gamma' \vdash \varsigma_c : \mathcal{F}_c$.

For the typing of the body, we can apply Lemma E.12. □

Lemma E.14 (Stack Well-Formedness is Preserved under Region Rewriting). *If* $\Sigma \vdash \sigma : \Gamma$ *and* $\bullet;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma'$ *then* $\Sigma \vdash \sigma : \Gamma'$.

Proof. We proceed by induction on the stack typing derivation.

WF-StackFrame
$$\frac{\begin{array}{c}\Sigma \vdash \sigma : \Gamma \qquad \text{dom}(\varsigma) = \text{dom}(\mathcal{F})|_{x} \\ \forall x \in \text{dom}(\varsigma).\ \Sigma;\ \bullet;\ \Gamma \natural \mathcal{F};\ \bullet \vdash \boxed{(\sigma \natural \varsigma)(x)} : (\Gamma \natural \mathcal{F})(x) \Rightarrow \Gamma \natural \mathcal{F}\end{array}}{\Sigma \vdash \sigma \natural \varsigma : \Gamma \natural \mathcal{F}}$$

WF-StackEmpty
$$\frac{}{\Sigma \vdash \bullet : \bullet}$$

The WF-StackEmpty case is immediate. In the WF-StackFrame case, we get the well formedness in the premise from our induction hypothesis. What's left to show is that for all of the values v in the stack frame, they remain well typed in Γ'. This follows from applying Lemma E.13. □

E.4 Preservation under Drops and Garbage Collection Lemmas

Lemma E.15 (Values Change Environments in Limited Ways). *If* $\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau \Rightarrow \Gamma'$, *then* $\Sigma;\ \Delta \vdash \Gamma \rhd \Gamma'$.

Proof. We proceed by induction on the structure of the typing derivation. Since we assume that the expression being typed is a value, we need only consider the cases that can be used to type a value.

For many cases, the output environments are precisely the input environments, and thus this holds immediately. These cases are T-Unit, T-u32, T-True, T-False, T-Pointer, T-Function, T-ClosureValue, and T-Dead.

For T-Tuple, T-Array, T-Left, and T-Right, knowing that we have a value means that all of the subterms are themselves values, and thus we can apply our induction hypothesis to them in sequence (relying on the transitivity of $\lesssim$ for stack typings).

This leaves us with one remaining case: T-Drop.

$$\textsc{T-Drop}\quad \frac{\Gamma(\pi) = \tau_\pi^{\text{SI}} \qquad \Sigma;\ \Delta;\ \Gamma[\pi \mapsto \tau_\pi^{\text{SI}^\dagger}];\ \Theta \vdash \boxed{e} : \tau^{\text{SX}} \Rightarrow \Gamma_f}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau^{\text{SX}} \Rightarrow \Gamma_f}$$

For T-Drop, we apply our induction hypothesis to $\Sigma;\ \Delta;\ \Gamma[\pi \mapsto \tau_\pi^{\text{SI}^\dagger}];\ \Theta \vdash \boxed{e} : \tau^{\text{SX}} \Rightarrow \Gamma_f$ which tells us that $\Sigma;\ \Delta \vdash \Gamma[\pi \mapsto \tau_\pi^{\text{SI}^\dagger}] \rhd \Gamma_f$. Then, by R-Env, we have that $\Sigma;\ \Delta \vdash \Gamma \rhd \Gamma[\pi \mapsto \tau_\pi^{\text{SI}^\dagger}]$. Then, by transitivity, we have $\Sigma;\ \Delta \vdash \Gamma \rhd \Gamma_f$. □

Lemma E.16 (Type Computation is Preserved in Related Environments). *If* $\Sigma;\ \Delta \vdash \Gamma \rhd \Gamma'$ *and* $\Delta;\ \Gamma \vdash_\omega p^\square[\pi] : \tau,\ \{\, \overline{\rho}\,\}$ *and* $\Gamma(\pi) = \Gamma'(\pi)$, *then* $\Delta;\ \Gamma' \vdash_\omega p^\square[\pi] : \tau,\ \{\, \overline{\rho}\,\}$.

Proof. We proceed by induction on the type computation derivation. TC-Var follows immediately by the same type hypothesis, and TC-Proj follows from applying the induction hypothesis. All that is left is TC-Deref.

$$\textsc{TC-Deref}\quad \frac{\Delta;\ \Gamma \vdash_\omega p : \&\rho\ \omega'\ \tau^{\text{XI}},\ \{\, \overline{\rho_p}\,\} \qquad \omega \lesssim \omega'}{\Delta;\ \Gamma \vdash_\omega *p : \tau^{\text{XI}},\ \{\, \overline{\rho_p},\ \rho\,\}}$$

First, we can apply the induction hypothesis to get the type computation for p. Then, all that's left is to show the outlives constraint, but this is immediate because Δ is unchanged and both Γ and Γ' have the exact same domains.

□

Lemma E.17 (Ownership Safety Preserved in Related Environments). *If* $\Delta; \Gamma; \Theta \vdash_\omega^{\overline{\pi_e}} p \Rightarrow \{\overline{\ell}\}$ *and* $\Sigma;\ \Delta \vdash \Gamma \rhd \Gamma'$ *and* $\Delta;\ \Gamma' \vdash_\omega p : \tau^{XI}$ *and* $p = p^\square[\pi_p]$ *and* $\Gamma(\pi_p) = \Gamma'(\pi_p)$, *then* $\Delta;\ \Gamma';\ \Theta \vdash_\omega^{\overline{\pi_e}} p \Rightarrow \{\, \overline{\ell}\,\}$.

Proof. We proceed by induction on the ω-safety derivation, for which there are three cases to consider.

O-SafePlace
$$\frac{\begin{array}{c}\forall r' \mapsto \{\, \overline{\ell} \,\} \in \text{regions}(\Gamma,\, \Theta).\ (\forall {}^{\omega'} p^{\square}[\pi'] \ \in \{\, \overline{\ell} \,\}.(\omega = \texttt{uniq} \lor \omega' = \texttt{uniq}) \implies \pi' \,\#\, \pi) \\ \lor\ (\exists \pi' \ : \ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma) \ \land \nexists \&r'\ \omega'\ \tau' \in \Theta \ \land (\forall \pi' \ : \ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\, \overline{\pi_e} \,\}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi_e}} \pi \Rightarrow \{\ {}^{\omega}\pi\ \}}$$

We'd like to show that O-SafePlace can be applied with context Γ'. First, note that for any r', if the right side of the or is true for Γ with $\overline{\pi}$ then it will be true for Γ' with $\overline{\pi}$. That is, if all of the pointers with region r' in Γ are in the exclusion list $\overline{\pi}$, then all of the pointers with region r' in Γ' are also in the exclusion list $\overline{\pi}$. Note that Θ is unchanged between the two. Therefore, the only cases we need to consider are where r' occurs in pointers in Γ and Γ' that do not occur in $\overline{\pi}$.

Since the only allowed change to loan sets is emptying, and an emptied loan set has the left side of the disjunction as vacuously true, and if the loan set is the same we have the condition from the ownership safety in the premise, we are done.

O-Deref
$$\frac{\begin{array}{c}\Gamma(\pi) = \&r\ \omega_{\pi}\ \tau_{\pi} \qquad \Gamma(r) = \{\ \overline{{}^{\omega'}p}^{\,n}\ \} \qquad \text{excl} = \{\, \pi_j \text{ where } j \in \{\, 1,\ \ldots\ n \,\} \mid p_j = p_j^{\square}[*\pi_j] \,\} \qquad \omega \lesssim \omega_{\pi} \\ \forall i \in \{\, 1\ \ldots\ n \,\}.\ \Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi_e},\, \text{excl},\, \pi} p^{\square}[p_i] \Rightarrow \{\ \overline{{}^{\omega}p_i'}\ \} \\ \forall r' \mapsto \{\, \overline{\ell} \,\} \in \text{regions}(\Gamma,\, \Theta).\ (\forall {}^{\omega'} p'' \ \in \{\, \overline{\ell} \,\}.(\omega = \texttt{uniq} \lor \omega' = \texttt{uniq}) \implies p'' \,\#\, p^{\square}[*\pi]) \\ \lor\ (\exists \pi' \ : \ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma) \ \land \nexists \&r'\ \omega'\ \tau' \in \Theta \ \land (\forall \pi' \ : \ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\, \overline{\pi_e},\ \text{excl},\ \pi \,\}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi_e}} p^{\square}[*\pi] \Rightarrow \{\ \overline{{}^{\omega}p_1'}\ ,\ \ldots\ \overline{{}^{\omega}p_n'}\ ,\ {}^{\omega}p^{\square}[*\pi]\ \}}$$

Firstly, we have that $\Gamma(\pi_i) = \Gamma'(\pi_i)$, because $\Gamma'(\pi_i)$ must be an initialized type by the type computation premise, and the only changes in types between Γ and Γ' allowed by the environment relation is dropping some types to uninitialized.

Second, note that $\Gamma'(r) = \Gamma(r)$ since $\Gamma'(\pi)$ being a reference with region r means we can't empty the loan set. So we proceed by applying the induction hypothesis for all n loans, noting that the type computation requirement follows from the well formedness of Γ'.

Finally, we have to show the statement about no conflicting loans, but here the argument is identical to that in the O-SafePlace case. If the loan set is empty then we're done, otherwise we just use the ownership safety premise.

O-DerefAbs
$$\frac{\begin{array}{c}\Gamma(\pi) = \&\varrho\ \omega_{\pi}\ \tau_{\pi} \qquad \Delta;\ \Gamma \vdash_{\omega} p^{\square}[*\pi] : \tau \qquad \omega \lesssim \omega_{\pi} \\ \forall r' \mapsto \{\, \overline{\ell} \,\} \in \text{regions}(\Gamma,\, \Theta).\ (\forall {}^{\omega'} p \ \in \{\, \overline{\ell} \,\}.(\omega = \texttt{uniq} \lor \omega' = \texttt{uniq}) \implies p \,\#\, p^{\square}[*\pi]) \\ \lor\ (\exists \pi' \ : \ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma) \ \land \nexists \&r'\ \omega'\ \tau' \in \Theta \ \land (\forall \pi' \ : \ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\, \overline{\pi_e},\ \pi \,\}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi_e}} p^{\square}[*\pi] \Rightarrow \{\ {}^{\omega}p^{\square}[*\pi]\ \}}$$

This case proceeds similarly to the O-Deref case, but with an added application of Lemma E.16 to get the type computation, and no application of any induction hypothesis. □

Lemma E.18 (Types Are Well Formed in Related Environments). *If* $\Sigma;\ \Delta \vdash \Gamma \rhd \Gamma'$ *and* $\Sigma;\ \Delta;\ \Gamma \vdash \tau^{XI}$ *and* $\forall r$ *that occur in* τ^{XI}, $\Gamma(r) = \Gamma'(r)$, *then* $\Sigma;\ \Delta;\ \Gamma' \vdash \tau^{XI}$.

Proof. We proceed by induction on the type well formedness derivation. The only case that doesn't follow directly from induction and the fact that Δ and Θ are unchanged between the two related environments is WF-Ref.

$$\textsc{WF-Ref}\quad \frac{\Delta;\ \Gamma \vdash \rho \qquad \Sigma;\ \Delta;\ \Gamma \vdash \tau^{\text{XI}}}{\Sigma;\ \Delta;\ \Gamma \vdash \&\rho\ \omega\ \tau^{\text{XI}}}$$

Firstly we apply our induction hypothesis to get that $\Sigma;\ \Delta;\ \Gamma' \vdash \tau_p^{\text{XI}}$. What's left to show is the loan set condition on r. If $\Gamma'(r) = \emptyset$, then we're done. Otherwise, we just need that the type computation still holds, which we get from Lemma E.16. We know the places in these place expressions all have the same type in Γ and Γ' because between these two contexts the only changes allowed that could cause problems here are dropping one of these places, but then Γ' would not be well formed since there would be an invalid loan.

□

Lemma E.19 (Related Environments Remain Well-Formed). *If* $\Sigma;\ \Delta\ \vdash\ \Gamma \rhd \Gamma'$ *and* $\vdash \Sigma;\ \Delta;\ \Gamma \natural \mathcal{F}_c;\ \Theta$ *then* $\vdash \Sigma;\ \Delta;\ \Gamma' \natural \mathcal{F}_c;\ \Theta$.

Proof. From the well formedness of $\Gamma \natural \mathcal{F}_c$, we know that the places and disjointness conditions both hold. We also know that the occurs in restriction holds, because we at most have the same alive types. By Lemma E.18, noting that $\Sigma;\ \Delta \vdash \Gamma \natural \mathcal{F}_c \rhd \Gamma' \natural \mathcal{F}_c$ is immediate, we know that the types remain well formed in the environment. We also have the well formedness of Γ' as a premise of the related environments judgement. All that's left to show is the loan set condition. But for this all we have to show is that each place computes to some type, which follows from Lemma E.16. We know the types of the places in each place expression remain the same because the only allowed changes between Γ and Γ' are that places can be dropped and loan sets emptied, but if one such place was dropped, then Γ' would have not been well formed. □

Lemma E.20 (Related Input Environments Produce Similar Output Environments). *If:*

- $\Sigma;\ \Delta;\ \Gamma_1;\ \Theta \vdash \boxed{e_1} : \tau_1 \Rightarrow \Gamma_2$
- $\Sigma;\ \Delta;\ \Gamma_1;\ \Theta \vdash \boxed{e_2} : \tau_2 \Rightarrow \Gamma_3$
- $\Sigma;\ \Delta \vdash \Gamma_1 \rhd \Gamma_1'$
- $\Sigma;\ \Delta;\ \Gamma_1';\ \Theta \vdash \boxed{e_1} : \tau_1 \Rightarrow \Gamma_2'$
- $\Sigma;\ \Delta;\ \Gamma_1';\ \Theta \vdash \boxed{e_2} : \tau_2 \Rightarrow \Gamma_3'$
- $\Sigma;\ \Delta \vdash \Gamma_2 \rhd \Gamma_2'$
- $\Sigma;\ \Delta \vdash \Gamma_3 \rhd \Gamma_3'$
- $\forall x \in dom(\Gamma_2)$, $\Gamma_2(x) = \Gamma_3(x)$ *and* $\Gamma_2'(x) = \Gamma_3'(x)$
- $\forall r$ *that occur in* e_1 *or* e_2 *or* τ_1 *or* τ_2, $\Gamma_1(r) = \Gamma_1'(r)$

then $\forall r \in dom(\Gamma_1)$, *if* $\Gamma_2'(r) = \emptyset$ *and* $\Gamma_3'(r) \neq \emptyset$, *then* $\Gamma_2(r) = \emptyset$, *and if* $\Gamma_3'(r) = \emptyset$ *and* $\Gamma_2'(r) \neq \emptyset$, *then* $\Gamma_3(r) = \emptyset$.

Proof. The proofs for both statements in the conclusion follow identically, so without loss of generality it suffices to show that if $\Gamma_2'(r) = \emptyset$ and $\Gamma_3'(r) \neq \emptyset$, then $\Gamma_2(r) = \emptyset$. Note there are two cases to consider: that the loan set was empty all along, or that the loan set was at some point non empty, but then got garbage collected.

First, at some point between Γ_1' and Γ_2', r mapped to a non empty set of loans but then was garbage collected. In this case, Γ_2' must not contain any references that contain r, since otherwise it would have been invalid to garbage collect r. But then since Γ_2' and Γ_3' agree on types, it must be the case that it was also garbage collected in Γ_3', which is a contradiction with the fact that $\Gamma_3'(r)$ is non empty, so this case is impossible.

Second, at each step of the derivation between Γ_1' and Γ_2', r mapped to empty. If $\Gamma_1(r)$ also was empty, then this means $\Gamma_2(r)$ is also empty, and we're done. Otherwise, r was garbage collected between Γ_1 and Γ'_1. But then r must be free in e_2 for loans to have been added between Γ'_1 and Γ'_3, which means the loan set could not have been emptied between Γ_1 and Γ'_1, which is a contradiction. □

Lemma E.21 (Outlives Preserves Related Environments). *If* $\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \rho_1 :> \rho_2 \dashv \Gamma_o$, *and* $\Sigma;\ \Delta \vdash \Gamma \triangleright \Gamma'$ *and* $\vdash \Sigma;\ \Delta;\ \Gamma_o;\ \Theta$ *and* $\Gamma(\rho_1) = \Gamma'(\rho_1)$ *and* $\Gamma(\rho_2) = \Gamma'(\rho_2)$, *then* $\Delta;\ \Gamma';\ \Theta \vdash^{\mu} \rho_1 :> \rho_2 \dashv \Gamma'_o$, *and* $\Sigma;\ \Delta \vdash \Gamma_o \triangleright \Gamma'_o$. *and* $\Gamma_o(\rho_1) = \Gamma'_o(\rho_1)$ *and* $\Gamma_o(\rho_2) = \Gamma'_o(\rho_2)$

Proof. Proceed by induction on the outlives derivation. OL-Refl, OL-Trans, OL-AbstractConcrete, and OL-BothAbstract are immediate.

OL-ConcreteAbstract follows from additionally applying Lemma E.16. The condition on the place having the same type follows from the fact that p is a loan and $\Gamma'(r)$ is not emptied, so we could not have dropped the place.

OL-CheckConcrete is immediate, because the occurs before condition is unaffected since the domains are equal, and the region not reborrowed judgement is unaffected by adding loans that are already in other loan sets.

This leaves two cases which proceed similarly: OL-CombineConcrete and OL-CombineConcreteUnrestricted.

$$\textsc{OL-CombineConcrete}\quad \frac{\begin{array}{ccc}\Gamma \vdash r_1\ \texttt{rnrb} & \Gamma \vdash r_2\ \texttt{rnrb} & \Gamma;\ \Theta \vdash \{\, r_1,\ r_2 \,\}\ \texttt{clrs} \\ r_1\ \texttt{occurs before}\ r_2\ \texttt{in}\ \Gamma & & \{\, \overline{\ell}\, \} = \Gamma(r_1) \cup \Gamma(r_2)\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash^{+} r_1 :> r_2 \dashv \Gamma[r_2 \mapsto \{\, \overline{\ell}\, \}]}$$

Since $\Gamma'(r_1) = \Gamma(r_1)$ and $\Gamma'(r_2) = \Gamma(r_2)$, $\Gamma'(r_1) \cup \Gamma'(r_2) = \Gamma(r_1) \cup \Gamma'(r_2)$. The region not reborrowed judgement is unaffected by adding loans that are already in other loan sets, so those conditions are also still true. The rest of the conditions are immediate: the equality on r_1 and r_2's loan sets, the closure restriction since types are at most the same, and well formedness. □

Lemma E.22 (Related Environments Preserved by Region Rewriting). *If* $\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1^{SI} \rightsquigarrow \tau_2^{SI} \dashv \Gamma_o$, *and* $\Sigma;\ \Delta \vdash \Gamma \triangleright \Gamma'$ *and* $\vdash \Sigma;\ \Delta;\ \Gamma_o;\ \Theta$ *and* $\forall r$ *that occur in* τ_1^{SI} *or* τ_2^{SI}, $\Gamma(r) = \Gamma'(r)$, *then* $\Delta;\ \Gamma';\ \Theta \vdash^{\mu} \tau_1^{SI} \rightsquigarrow \tau_2^{SI} \dashv \Gamma'_o$, *and* $\Sigma;\ \Delta \vdash \Gamma_o \triangleright \Gamma'_o$, *and* $\forall r$ *that occur in* τ_1^{SI} *or* τ_2^{SI}, $\Gamma_o(r) = \Gamma'_o(r)$.

Proof. Proceed by induction on the region rewriting derivation. The only interesting case is RR-Reference, which proceeds by Lemma E.21 in addition to applying the induction hypothesis. □

Lemma E.23 (Expression Typing Preserved in Related Environments). *Let e be a surface expression as defined on page 1. If* $\Sigma;\ \Delta;\ \Gamma \natural \mathcal{F};\ \Theta \vdash \boxed{e} : \tau \Rightarrow \Gamma_o \natural \mathcal{F}_o$ *and* $\Sigma;\ \Delta \vdash \Gamma \natural \mathcal{F} \triangleright \Gamma' \natural \mathcal{F}$ *and* *free-vars*$(e) = \overline{x_f} \subseteq$ *dom*$(\mathcal{F})|_x$ *and* $\forall r \in$ *free-regions*(e). $r \in$ *dom*$(\mathcal{F})$, *and* $\forall r$ *that occur a type in* $\overline{\mathcal{F}(x_f)}$, $\Gamma(r) = \Gamma'(r)$ *then* $\Sigma;\ \Delta;\ \Gamma' \natural \mathcal{F};\ \Theta \vdash \boxed{e} : \tau \Rightarrow \Gamma'_o \natural \mathcal{F}_o$ *and* $\Sigma;\ \Delta \vdash \Gamma_o \natural \mathcal{F}_o \triangleright \Gamma'_o \natural \mathcal{F}_o$ *and* $\forall r$ *that occur a type in* $\overline{\mathcal{F}(x_f)}$, $\Gamma_o(r) = \Gamma'_o(r)$.

Proof. Proceed by induction on the typing derivation for e. In the cases of T-Abort, T-Function, T-Unit, T-u32, T-True, and T-False, the results are immediate.

In the cases of T-LetRegion, T-While, T-ForArray, T-ForSlice, T-Closure, T-Left, and T-Right, they all follow immediately from induction hypotheses.

For each of the following cases, the convention is that the statement in the box is our assumption, and we want to prove the same statement with Γ' replaced for each Γ.

T-Tuple

$$\frac{\forall i \in \{\, 1 \,\ldots\, n \,\}.\ \Sigma;\ \Delta;\ \Gamma_{i-1} \natural \mathcal{F}_{i-1};\ \Theta,\ \tau_1^{\text{SI}},\ \ldots,\ \tau_{i-1}^{\text{SI}} \vdash \boxed{e_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_i \natural \mathcal{F}_i}{\Sigma;\ \Delta;\ \Gamma_0 \natural \mathcal{F}_0;\ \Theta \vdash \boxed{(e_1,\ \ldots,\ e_n)} : (\tau_1^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}}) \Rightarrow \Gamma_n \natural \mathcal{F}_n}$$

We have n induction hypotheses, each giving us the properties for input context $\Gamma'_{i-1} \natural \mathcal{F}'_{i-1}$ and output context $\Gamma'_i \natural \mathcal{F}'_i$.

Given these resulting n typing judgements, we get from applying T-Tuple that $\Sigma;\ \Delta;\ \Gamma'_0 \natural \mathcal{F}_0;\ \Theta \vdash \boxed{(e_1,\ \ldots,\ e_n)} : (\tau_1^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}}) \Rightarrow \Gamma'_o \natural \mathcal{F}'_o$, as well as the related environments judgement $\Sigma;\ \Delta \vdash \Gamma_n \natural \mathcal{F}_n \rhd \Gamma'_n \natural \mathcal{F}'_n$.

The cases for T-Array and T-Slice proceed identically. This reasoning is also used in the T-App case.

T-Branch

$$\frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma \natural \mathcal{F};\ \Theta \vdash \boxed{e_1} : \texttt{bool} \Rightarrow \Gamma_1 \natural \mathcal{F}_1 \qquad \Sigma;\ \Delta;\ \Gamma_1 \natural \mathcal{F}_1;\ \Theta \vdash \boxed{e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2 \natural \mathcal{F}_2 \\ \Sigma;\ \Delta;\ \Gamma_1 \natural \mathcal{F}_1;\ \Theta \vdash \boxed{e_3} : \tau_3^{\text{SI}} \Rightarrow \Gamma_3 \natural \mathcal{F}_3 \qquad \tau^{\text{SI}} = \tau_2^{\text{SI}} \vee \tau^{\text{SI}} = \tau_3^{\text{SI}} \\ \Delta;\ \Gamma_2 \natural \mathcal{F}_2;\ \Theta \vdash^{\mu} \tau_2^{\text{SI}} \leadsto \tau^{\text{SI}} \dashv \Gamma_{2s} \natural \mathcal{F}_{2s} \\ \Delta;\ \Gamma_3 \natural \mathcal{F}_3;\ \Theta \vdash^{\mu} \tau_3^{\text{SI}} \leadsto \tau^{\text{SI}} \dashv \Gamma_{3s} \natural \mathcal{F}_{3s} \qquad \Gamma_{2s} \natural \mathcal{F}_{2s} \Cup \Gamma_{3s} \natural \mathcal{F}_{3s} = \Gamma_o \natural \mathcal{F}_o\end{array}}{\Sigma;\ \Delta;\ \Gamma \natural \mathcal{F};\ \Theta \vdash \boxed{\texttt{if}\ e_1\ \{\ e_2\ \}\ \texttt{else}\ \{\ e_3\ \}} : \tau^{\text{SI}} \Rightarrow \Gamma_o \natural \mathcal{F}_o}$$

By our induction hypothesis we get that $\Sigma;\ \Delta;\ \Gamma' \natural \mathcal{F};\ \Theta \vdash \boxed{e_1} : \texttt{bool} \Rightarrow \Gamma'_1 \natural \mathcal{F}_1$, and $\Sigma;\ \Delta;\ \Gamma'_1 \natural \mathcal{F}_1;\ \Theta \vdash \boxed{e_2} : \texttt{bool} \Rightarrow \Gamma'_2 \natural \mathcal{F}_2$, and $\Sigma;\ \Delta;\ \Gamma'_1 \natural \mathcal{F}_1;\ \Theta \vdash \boxed{e_3} : \texttt{bool} \Rightarrow \Gamma'_3 \natural \mathcal{F}_3$ with $\Sigma;\ \Delta \vdash \Gamma_2 \natural \mathcal{F}_2 \rhd \Gamma'_2 \natural \mathcal{F}_2$ and $\Sigma;\ \Delta \vdash \Gamma_3 \natural \mathcal{F}_3 \rhd \Gamma'_3 \natural \mathcal{F}_3$.

Next we want to show that $\Delta;\ \Gamma'_2 \natural \mathcal{F}_2;\ \Theta \vdash^{\mu} \tau_2^{\text{SI}} \leadsto \tau^{\text{SI}} \dashv \Gamma'_{2s} \natural \mathcal{F}_{2s}$, $\Sigma;\ \Delta \vdash \Gamma_{2s} \natural \mathcal{F}_{2s} \rhd \Gamma'_{2s} \natural \mathcal{F}_{2s}$, $\Delta;\ \Gamma'_3 \natural \mathcal{F}_3;\ \Theta \vdash^{\mu} \tau_3^{\text{SI}} \leadsto \tau^{\text{SI}} \dashv \Gamma'_{3s} \natural \mathcal{F}_{3s}$, and $\Sigma;\ \Delta \vdash \Gamma_{3s} \natural \mathcal{F}_{3s} \rhd \Gamma'_{3s} \natural \mathcal{F}_{3s}$, which all follow from applying Lemma E.22. To do this lemma application, we just need to show that for all r in τ_1^{SI}, τ_2^{SI} and τ^{SI}, $\Gamma(r) = \Gamma'(r)$ which follows from the premise.

Finally, we need to show that $\Sigma;\ \Delta \vdash \Gamma_o \natural \mathcal{F}_o \rhd \Gamma'_o \natural \mathcal{F}_o$. The well formedness condition on Γ'_o follows immediately since all types are the same as in Γ'_2 and Γ'_3 and the loan sets are just unioned, meaning reference types remain valid and we can compute types for all loans.

The equal or empty condition follows from the fact that Γ'_2 and Γ'_3 both agree on types by Lemma E.20, which means they drop exactly the same entries. For any regions emptied, either the same regions are emptied, or the region was emptied in the corresponding smaller context Γ_2 or Γ_3. Otherwise the loan sets are untouched.

Finally, both of these are preserved when adding on the same frame, so we're done.

T-Match

$$\frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma \natural \mathcal{F};\ \Theta \vdash \boxed{e} : \texttt{Either<}\tau_l^{\text{SI}},\ \tau_r^{\text{SI}}\texttt{>} \Rightarrow \Gamma_1 \natural \mathcal{F}_1 \qquad \forall r \in \text{free-regions}(\texttt{Either<}\tau_l^{\text{SI}},\ \tau_r^{\text{SI}}\texttt{>}).\ \Gamma_1 \natural \mathcal{F}_1 \vdash r\ \texttt{rnrb} \\ \Sigma;\ \Delta;\ \Gamma_1 \natural \mathcal{F}_1,\ x_1 : \tau_l^{\text{SI}};\ \Theta \vdash \boxed{e_1} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2 \natural \mathcal{F}_2,\ x_1 : \tau_l^{\text{SD}} \\ \Sigma;\ \Delta;\ \Gamma_1 \natural \mathcal{F}_1,\ x_2 : \tau_r^{\text{SI}};\ \Theta \vdash \boxed{e_2} : \tau_3^{\text{SI}} \Rightarrow \Gamma_3 \natural \mathcal{F}_3,\ x_2 : \tau_r^{\text{SD}} \qquad \tau^{\text{SI}} = \tau_2^{\text{SI}} \vee \tau^{\text{SI}} = \tau_3^{\text{SI}} \\ \Delta;\ \Gamma_2 \natural \mathcal{F}_2;\ \Theta \vdash^{\mu} \tau_2^{\text{SI}} \leadsto \tau^{\text{SI}} \dashv \Gamma_{2s} \natural \mathcal{F}_{2s} \\ \Delta;\ \Gamma_3 \natural \mathcal{F}_3;\ \Theta \vdash^{\mu} \tau_3^{\text{SI}} \leadsto \tau^{\text{SI}} \dashv \Gamma_{3s} \natural \mathcal{F}_{3s} \qquad \Gamma_{2s} \natural \mathcal{F}_{2s} \Cup \Gamma_{3s} \natural \mathcal{F}_{3s} = \Gamma_o \natural \mathcal{F}_o\end{array}}{\Sigma;\ \Delta;\ \Gamma \natural \mathcal{F};\ \Theta \vdash \boxed{\texttt{match}\ e\ \{\ \texttt{Left}(x_1) \Rightarrow e_1,\ \texttt{Right}(x_2) \Rightarrow e_2\ \}} : \tau^{\text{SI}} \Rightarrow \Gamma_o \natural \mathcal{F}_o}$$

This case follows almost identically to the T-Branch case above. The only structural difference is that the expression typing judgements for e_1 and e_2 have x_1 and x_2 respectively in their environments, but we know we can remove x_1 and x_2 from each side and keep the contexts well formed, since nothing that comes before x_1 or x_2 can refer to it, and we know that τ_1^{SI} and τ_2^{SI} cannot in any way refer to x_1 or x_2 because we have from the region rewriting judgements in the premises that the types are well formed in $\Gamma_2 \natural \mathcal{F}_2$ and $\Gamma_3 \natural \mathcal{F}_3$ respectively. We also need to show that the region not reborrowed judgement still holds, but this is immediate because at most Γ'_1 has the same types as Γ_1.

T-Let

$$\frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma \natural \mathcal{F};\ \Theta \vdash \boxed{e_1} : \tau_1^{\text{SI}} \Rightarrow \Gamma_1 \natural \mathcal{F}_1 \qquad \Delta;\ \Gamma_1 \natural \mathcal{F}_1;\ \Theta \vdash^{\mu} \tau_1^{\text{SI}} \rightsquigarrow \tau_a^{\text{SI}} \dashv \Gamma_{1s} \natural \mathcal{F}_{1s} \\ \forall r \in \text{free-regions}(\tau_a^{\text{SI}}).\ \Gamma_{1s} \natural \mathcal{F}_{1s} \vdash r \ \texttt{rnrb} \\ \Sigma;\ \Delta;\ \text{gc-loans}_{\Theta}(\Gamma_{1s} \natural \mathcal{F}_{1s},\ x : \tau_a^{\text{SI}});\ \Theta \vdash \boxed{e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2 \natural \mathcal{F}_2,\ x : \tau^{\text{SD}}\end{array}}{\Sigma;\ \Delta;\ \Gamma \natural \mathcal{F};\ \Theta \vdash \boxed{\texttt{let}\ x : \tau_a^{\text{SI}} = e_1;\ e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2 \natural \mathcal{F}_2}$$

Firstly, we apply our induction hypothesis to get that e_1 is well typed with input environment $\Gamma' \natural \mathcal{F}$ and output environment $\Gamma'_1 \natural \mathcal{F}_1$ with $\Sigma;\ \Delta \vdash \Gamma_1 \natural \mathcal{F}_1 \rhd \Gamma'_1 \natural \mathcal{F}_1$. Then, we apply Lemma E.22 to get $\Sigma;\ \Delta \vdash \Gamma_{1s} \natural \mathcal{F}_{1s} \rhd \Gamma'_{1s} \natural \mathcal{F}_{1s}$. In order to apply this lemma we need to know that for any r that occur in τ_1^{SI} or τ_a^{SI}, $\Gamma_1 \natural \mathcal{F}_1(r) = \Gamma'_1 \natural \mathcal{F}_1(r)$, which we have as a conclusion from the previous application of the induction hypothesis.

The region not reborrowed judgement holds immediately, because at most Γ'_1 has the same types as Γ_1.

To apply our induction hypothesis on e_2 and continue the case, we need that $\Sigma;\ \Delta \vdash \text{gc-loans}_{\Theta}(\Gamma_{1s} \natural \mathcal{F}_{1s}, x : \tau_a^{\text{SI}}) \rhd \text{gc-loans}_{\Theta}(\Gamma'_{1s} \natural \mathcal{F}_{1s}, x : \tau_a^{\text{SI}})$. But this is immediate by definition since gcloans can only empty loan sets for regions for which there are no types that contain them, which is allowed by R-Env.

Our final obligation to apply the induction hypothesis is that for any r that occurs in a type in $\mathcal{F}_{1s}$ but is not in $\mathcal{F}_{1s}$, we need that $\text{gc-loans}_{\Theta}(\Gamma_{1s} \natural \mathcal{F}_{1s})(r) = \text{gc-loans}_{\Theta}(\Gamma'_{1s} \natural \mathcal{F}_{1s})(r)$. We already have that $\Gamma_{1s} \natural \mathcal{F}_{1s}(r) = \Gamma'_{1s} \natural \mathcal{F}_{1s}(r)$, so we just need to know that $\exists \pi : \tau \in \Gamma_{1s}$, where r occurs in τ, and $\Gamma_{1s} \natural \mathcal{F}_{1s}(\pi) = \Gamma_{1s} \natural \mathcal{F}'_{1s}(\pi)$. But we said that r is contained in a type in $\mathcal{F}_{1s}$, so the place for that type is one such place, so we cannot empty the loan set.

T-Seq

$$\frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma \natural \mathcal{F};\ \Theta \vdash \boxed{e_1} : \tau_1^{\text{SI}} \Rightarrow \Gamma \natural \mathcal{F}_1 \\ \Sigma;\ \Delta;\ \text{gc-loans}_{\Theta}(\Gamma_1 \natural \mathcal{F}_1);\ \Theta \vdash \boxed{e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2 \natural \mathcal{F}_2\end{array}}{\Sigma;\ \Delta;\ \Gamma \natural \mathcal{F};\ \Theta \vdash \boxed{e_1;\ e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2 \natural \mathcal{F}_2}$$

Firstly, we apply our induction hypothesis to get that e_1 is well typed with input environment $\Gamma' \natural \mathcal{F}$ and output environment $\Gamma'_1 \natural \mathcal{F}_1$, with $\Sigma;\ \Delta \vdash \Gamma_1 \natural \mathcal{F}_1 \rhd \Gamma'_1 \natural \mathcal{F}_1$. We need to know that $\Sigma;\ \Delta \vdash \text{gc-loans}_{\Theta}(\Gamma_1 \natural \mathcal{F}_1) \rhd \text{gc-loans}_{\Theta}(\Gamma'_1 \natural \mathcal{F}_1)$ before we can apply our induction hypothesis to finish the proof. But this fact is trivial by the definitions, since gc-loans can only empty regions that are not in initialized types in the context, which is allowed in R-Env.

Our final obligation to apply the induction hypothesis is that for any r that occurs in a type in $\mathcal{F}_1$ but is not in $\mathcal{F}_1$, we need that $\text{gc-loans}_{\Theta}(\Gamma_1 \natural \mathcal{F}_1)(r) = \text{gc-loans}_{\Theta}(\Gamma'_1 \natural \mathcal{F}_1)(r)$. We already have that $\Gamma_1 \natural \mathcal{F}_1(r) = \Gamma'_1 \natural \mathcal{F}_1(r)$, so we just need to know that $\exists \pi : \tau \in \Gamma_1$, where r occurs in τ, and $\Gamma_1 \natural \mathcal{F}_1(\pi) = \Gamma'_1 \natural \mathcal{F}_1(\pi)$. But since r occurs in a type in $\mathcal{F}_1$, the place that maps to that type is such a place.

T-Drop

$$\frac{\Gamma(\pi) = \tau_\pi^{\text{SI}} \qquad \Sigma;\ \Delta;\ (\Gamma \natural \mathcal{F})[\pi \mapsto \tau_\pi^{\text{SI}^\dagger}];\ \Theta \vdash \boxed{e} : \tau^{\text{SX}} \Rightarrow \Gamma_o \natural \mathcal{F}_o}{\Sigma;\ \Delta;\ \Gamma \natural \mathcal{F};\ \Theta \vdash \boxed{e} : \tau^{\text{SX}} \Rightarrow \Gamma_o \natural \mathcal{F}_o}$$

In order to apply our induction hypothesis and finish the case, we only need to show that $\Sigma;\ \Delta \vdash (\Gamma \natural \mathcal{F})[\pi \mapsto \tau_{\pi}^{\text{SI}^{\dagger}}] \rhd (\Gamma' \natural \mathcal{F})[\pi \mapsto \tau_{\pi}^{\text{SI}^{\dagger}}]$, which is immediate by the definition of related contexts. Note that $(\Gamma' \natural \mathcal{F})[\pi \mapsto \tau_{\pi}^{\text{SI}^{\dagger}}]$ is well formed because $(\Gamma \natural \mathcal{F})[\pi \mapsto \tau_{\pi}^{\text{SI}^{\dagger}}]$ is well formed. There cannot be any loans to π because in the $(\Gamma' \natural \mathcal{F})[\pi \mapsto \tau_{\pi}^{\text{SI}^{\dagger}}]$ because those loans would be there in $(\Gamma \natural \mathcal{F})[\pi \mapsto \tau_{\pi}^{\text{SI}^{\dagger}}]$.

T-App

$$\frac{\begin{array}{c}\overline{\Sigma;\ \Delta;\ \Gamma \natural \mathcal{F} \vdash \Phi} \qquad \overline{\Delta; \Gamma \natural \mathcal{F} \vdash \rho} \qquad \overline{\Sigma;\ \Delta;\ \Gamma \natural \mathcal{F} \vdash \tau^{\text{SI}}} \qquad \delta = \cdot\ \overline{[^{\Phi}/_{\varphi}]}\ \overline{[^{\rho}/_{\varrho}]}\ \overline{[^{\tau^{\text{SI}}}/_{\alpha}]} \\ \Sigma;\ \Delta;\ \Gamma \natural \mathcal{F};\ \Theta \vdash \boxed{\hat{e}_f} : \forall{<}\overline{\varphi},\ \overline{\varrho},\ \overline{\alpha}{>}(\tau_1^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}}) \overset{\Phi_c}{\rightarrow} \tau_f^{\text{SI}} \text{ where } \overline{\varrho_1 : \varrho_2} \Rightarrow \Gamma_0 \natural \mathcal{F}_0 \\ \forall i \in \{\, 1 \ldots n \,\}.\ \Sigma;\ \Delta;\ \Gamma_{i-1} \natural \mathcal{F}_{i-1};\ \Theta,\ \delta(\tau_1^{\text{SI}})\ \ldots\ \delta(\tau_{i-1}^{\text{SI}}) \vdash \boxed{\hat{e}_i} : \delta(\tau_i^{\text{SI}}) \Rightarrow \Gamma_i \natural \mathcal{F}_i \\ \forall i \in \{\, 1 \ldots n \,\}.\ \forall r \in \text{free-regions}(\tau_i^{\text{SI}}).\ \Gamma_n \natural \mathcal{F}_n \vdash r \ \mathsf{rnrb} \qquad \Delta;\ \Gamma_n \natural \mathcal{F}_n;\ \Theta \vdash \overline{\varrho_2\ \overline{[^{\rho}/_{\varrho}]} :> \varrho_1\ \overline{[^{\rho}/_{\varrho}]}} \dashv \Gamma_b \natural \mathcal{F}_b\end{array}}{\Sigma;\ \Delta;\ \Gamma \natural \mathcal{F};\ \Theta \vdash \boxed{\hat{e}_f{::}{<}\overline{\Phi},\ \overline{\rho},\ \overline{\tau^{\text{SI}}}{>}(\hat{e}_1,\ \ldots,\ \hat{e}_n)} : \tau_f^{\text{SI}}\ \overline{[^{\Phi}/_{\varphi}]}\ \overline{[^{\rho}/_{\varrho}]}\ \overline{[^{\tau^{\text{SI}}}/_{\alpha}]} \Rightarrow \Gamma_b \natural \mathcal{F}_b}$$

In the case of T-App, we firstly must prove the well formedness properties:

- $\Sigma;\ \Delta;\ \Gamma' \natural \mathcal{F} \vdash \Phi$. Since Δ is unchanged, WF-Env is the only interesting case.

WF-Env

$$\frac{\Sigma;\ \Delta \vdash \Gamma \natural \mathcal{F} \natural \mathcal{F}_c}{\Sigma;\ \Delta;\ \Gamma \natural \mathcal{F} \vdash \mathcal{F}_c}$$

  Let $\Phi = \mathcal{F}_e$. We want to show that $\vdash \Sigma;\ \Delta;\ \Gamma' \natural \mathcal{F} \natural \mathcal{F}_c;\ \Theta$ given $\vdash \Sigma;\ \Delta;\ \Gamma \natural \mathcal{F} \natural \mathcal{F}_c;\ \Theta$, which is immediate from Lemma E.19.
- $\Delta;\ \Gamma' \natural \mathcal{F} \vdash \rho$, which is immediate from the premises since related loan environments have the same domains and Δ is the same.
- $\Sigma;\ \Delta;\ \Gamma' \natural \mathcal{F} \vdash \tau^{\text{SI}}$, which is immediate from Lemma E.18. We just need that for the regions that occur in the type, their loan sets are unchanged, but we get that from the premise, because the function argument is either: locally defined, in which case it can only use and produce types accessible in the context; an argument, in which case its arguments are also part of the argument type; or a global function, in which case these types do not contain any non abstract regions which are replaced with concrete regions all in $\mathcal{F}$.

For the rest of the application case, we can apply our induction hypothesis on the function and the arguments, additionally applying the substitution lemma, Lemma E.2, where needed. The region not reborrowed condition is true by the fact that at most the types between Γ_n and Γ_n' are the same. The last part about outlives follows from Lemma E.21, where we have the condition on the loan sets from the conclusion of the application of the induction hypothesis.

T-AppClosure

$$\frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma \natural \mathcal{F};\ \Theta \vdash \boxed{e_f} : \forall{<>}(\tau_1^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}}) \overset{\Phi_c}{\rightarrow} \tau_f^{\text{SI}} \Rightarrow \Gamma_{0s} \natural \mathcal{F}_{0s} \\ \forall i \in \{\, 1 \ldots n \,\}.\ \Sigma;\ \Delta;\ \Gamma_{i-1s} \natural \mathcal{F}_{i-1s};\ \Theta,\ \tau_1^{\text{SI}}\ \ldots\ \tau_{i-1}^{\text{SI}} \vdash \boxed{e_i} : \tau_{i'}^{\text{SI}} \Rightarrow \Gamma_i \natural \mathcal{F}_i \qquad \Delta;\ \Gamma_i \natural \mathcal{F}_i;\ \Theta \vdash^{\boxplus} \tau_{i'}^{\text{SI}} \rightsquigarrow \tau_i^{\text{SI}} \dashv \Gamma_{is} \natural \mathcal{F}_{is} \\ \forall i \in \{\, 1 \ldots n \,\}.\ \forall r \in \text{free-regions}(\tau_i^{\text{SI}}).\ \Gamma_{ns} \natural \mathcal{F}_{ns} \vdash r \ \mathsf{rnrb}\end{array}}{\Sigma;\ \Delta;\ \Gamma \natural \mathcal{F};\ \Theta \vdash \boxed{e_f(e_1,\ \ldots,\ e_n)} : \tau_f^{\text{SI}} \Rightarrow \Gamma_{ns} \natural \mathcal{F}_{ns}}$$

In the case of T-AppClosure, we follow a very similar procedure to T-AppFunction, but with an empty substituion, and the addition that we need to apply Lemma E.22 to handle the rewriting.

In the cases of T-Move, T-Copy, T-Borrow, T-BorrowIndex, T-BorrowSlice, IndexCopy, they all follow from the induction hypothesis and additionally applying Lemma E.17 and Lemma E.16. Note we get the place having the right type requirement for T-Move from the fact that the place must be in $\mathcal{F}$ since it is a free variable.

The remaining cases of T-Assign and T-AssignDeref proceed similarly. Firstly, we apply the induction hypothesis on the expression, then Lemma E.17 and Lemma E.16, and finally we get well formedness and relatedness on the output environment by applying Lemma E.22. Note we get the place having the same type requirement for type computation from the fact that the place must be in $\mathcal{F}$ since it is a free variable.

□

Lemma E.24 (Referent Well Formedness Preserved in Related Environments). *If* $\Sigma;\ \Delta \vdash \Gamma \triangleright \Gamma'$ *and* $\Sigma;\ \Gamma \vdash \mathcal{R}^{\square}[\pi] : \tau^{XI}$ *and* $\Gamma(\pi) = \Gamma'(\pi)$, *then* $\Sigma;\ \Gamma' \vdash \mathcal{R}^{\square}[\pi] : \tau^{XI}$.

Proof. Proceed by induction on the referent validity derivation. The only case that doesn't follow immediately from premises and the induction hypothesis in WF-RefId, which follows from the equal types premise. □

Lemma E.25 (Value Typing Preserved in Related Environments). *If* $\Sigma;\ \bullet \vdash \Gamma \triangleright \Gamma'$, *and* $\Sigma;\ \bullet;\ \Gamma;\ \bullet \vdash \boxed{v} : \Gamma(x) \Rightarrow \Gamma$, *then* $\Sigma;\ \bullet;\ \Gamma';\ \bullet \vdash \boxed{v} : \Gamma'(x) \Rightarrow \Gamma'$.

Proof. Proceed by simultaneous induction on the typing derivation and the stack frame well formedness. Since we know the expression is already a value, we restrict ourselves only to those cases that type values: T-Unit, T-u32, T-True, T-False, T-Tuple, T-Array, T-Left, T-Right, T-Dead, T-Pointer, and T-ClosureValue.

For T-Unit, T-u32, T-Dead, T-True, and T-False, this holds trivially. For T-Tuple, T-Array, T-Left, and T-Right, this holds directly by repeated application of our induction hypothesis. This leaves us with four cases.

T-ClosureValue

$$\frac{\begin{array}{c}\text{free-vars}(e) \setminus \overline{x} = \overline{x_f} = \text{dom}(\mathcal{F}_c)|_{\text{VAR}} \\ \overline{r} = \overline{\text{free-regions}(\Gamma(x_f))},\ (\text{free-regions}(e) \setminus (\overline{\text{free-regions}(\tau^{\text{SI}})}, \text{free-regions}(\tau_r^{\text{SI}}))) = \text{dom}(\mathcal{F}_c)|_{\text{RGN}} \\ \Sigma;\ \Gamma \vdash \varsigma_c : \mathcal{F}_c \qquad \Sigma;\ \Delta;\ \Gamma \natural \mathcal{F}_c,\ x_1 : \tau_1^{\text{SI}}, \ldots, x_n : \tau_n^{\text{SI}};\ \Theta \vdash \boxed{e} : \tau_r^{\text{SI}} \Rightarrow \Gamma' \natural \mathcal{F}\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\langle \varsigma_c,\ |x_1 : \tau_1^{\text{SI}}, \ldots, x_n : \tau_n^{\text{SI}}| \rightarrow \tau_r^{\text{SI}}\ \{\ e\ \}\ \rangle} : (\tau_1^{\text{SI}}, \ldots, \tau_n^{\text{SI}}) \overset{\mathcal{F}_c}{\rightarrow} \tau_r^{\text{SI}} \Rightarrow \Gamma}$$

For the T-ClosureValue case, firstly we want to show $\Sigma;\ \Gamma' \vdash \varsigma : \mathcal{F}_c$. This follows immediatedly from applying the induction hypothesis for each value.

Then to finish the closure case, it suffices to show
$\Sigma;\ \bullet;\ \Gamma' \natural \mathcal{F}_c;\ \bullet \vdash \boxed{e} : \tau_r^{\text{SI}} \Rightarrow \Gamma_o' \natural \mathcal{F}$, which follows immediately from Lemma E.23.

T-Pointer

$$\frac{\Sigma;\ \Gamma \vdash \mathcal{R}^{\square}[\pi] : \tau^{\text{XI}} \qquad {}^{\omega}\pi \in \Gamma(r)}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{ptr}\ \mathcal{R}^{\square}[\pi]} : \&r\ \omega\ \tau^{\text{XI}} \Rightarrow \Gamma}$$

If x was dropped, then $\Gamma'(x) = \Gamma(x)^{\dagger}$. Then the proof follows immediately from T-Dead.

If x was not dropped, then $\Gamma(x) = \Gamma'(x)$. All that is left to show is that that the referent remains well formed, and the loan ${}^{\omega}\pi$ is in $\Gamma'(r)$. The first condition follows from Lemma E.24. The second condition is immediate because the only potential changes allowed in the related environment to loan sets is emptying the loan sets of regions if there's no references with the region in their type, and this particular reference is a reference with the region, so emptying the loan set is ruled out. □

Lemma E.26 (Value Typing Fixed on Output Environments). *If* $\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau \Rightarrow \Gamma'$, *then* $\Sigma;\ \Delta;\ \Gamma';\ \Theta \vdash \boxed{v} : \tau \Rightarrow \Gamma'$.

Proof. Immediate by induction on the typing derivation. The only non immediate case is T-Pointer, where we also need to apply Lemma E.24. □

Lemma E.27 (Stack Validity is Preserved in Related Environments). *If* $\Sigma \vdash \sigma : \Gamma$ *and* $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma'$, *then* $\Sigma \vdash \sigma : \Gamma'$.

Proof. We proceed by induction over the well typedness of the store.

WF-StackFrame

$$\frac{\Sigma \vdash \sigma : \Gamma \qquad \mathrm{dom}(\varsigma) = \mathrm{dom}(\mathcal{F})|_x \qquad \forall x \in \mathrm{dom}(\varsigma).\ \Sigma;\ \bullet;\ \Gamma \natural \mathcal{F};\ \bullet \vdash \boxed{(\sigma \natural \varsigma)(x)} : (\Gamma \natural \mathcal{F})(x) \Rightarrow \Gamma \natural \mathcal{F}}{\Sigma \vdash \sigma \natural \varsigma : \Gamma \natural \mathcal{F}}$$

WF-StackEmpty

$$\frac{}{\Sigma \vdash \bullet : \bullet}$$

The interesting case is when the stack is non empty. Then we have that $\Sigma \vdash \sigma : \Gamma'$ and want to show that $\Sigma \vdash \sigma \natural \varsigma : \Gamma' \natural \mathcal{F}$. The requirement on the domain is immediate since related environments have the same domains. What's left to show is that the values in the store remain well typed under the new environment. This follows from repeated applications of Lemma E.25 □

E.5 Preservation When Popping a Stack Frame Lemmas

Lemma E.28 (Stack Validity is Preserved When Popping A Stack Frame). *If* $\Sigma \vdash \sigma \natural \varsigma : \Gamma \natural \mathcal{F}$, *then* $\Sigma \vdash \sigma : \Gamma$.

Proof. Immediate by inversion on WF-StackFrame which gives us $\Sigma \vdash \sigma : \Gamma$. □

E.6 Preservation under Well Typed Extension Lemmas

Lemma E.29 (Ownership Safety is Preserved under Well-Typed Extensions). *If*

(1) $\Sigma;\ \bullet;\ \Gamma \natural \mathcal{F} \vdash \tau^{SI}_x$
(2) $\bullet;\ \Gamma \natural \mathcal{F};\ \Theta \vdash^{\overline{\pi}}_{\omega} p \Rightarrow \{\ \overline{\ell}\ \}$
(3) $\texttt{root-of}(p) \in dom(\mathcal{F})$
(4) $\forall r \in \mathit{free\text{-}regions}(\tau^{SI}_x).\ \Gamma \vdash r\ \texttt{rnrb}$
(5) $\forall r \in \mathit{free\text{-}regions}(\tau^{SI}_x).\ \forall \pi_i \in \{\ \overline{\pi}\ \}.\ \Gamma(\pi_i) = \&r''\ \omega\ \tau^{XI} \implies r'' \neq r$

then $\bullet;\ \Gamma,\ x\ :\ \tau^{SI}_x \natural \mathcal{F};\ \Theta \vdash^{\overline{\pi}}_{\omega} p \Rightarrow \{\ \overline{\ell}\ \}$.

Proof. We proceed by induction on the ownership safety derivation.

O-SafePlace

$$\frac{\forall r' \mapsto \{\ \overline{\ell}\ \} \in \mathrm{regions}(\Gamma,\ \Theta).\ (\forall^{\omega'} p^{\square}[\pi'] \in \{\ \overline{\ell}\ \}.(\omega = \texttt{uniq} \lor \omega' = \texttt{uniq}) \implies \pi' \# \pi) \lor (\exists \pi'\ :\ \&r'\ \omega'\ \tau' \in \mathrm{explode}(\Gamma)\ \land \nexists \&r'\ \omega'\ \tau' \in \Theta\ \land (\forall \pi'\ :\ \&r'\ \omega'\ \tau' \in \mathrm{explode}(\Gamma).\ \pi' \in \{\ \overline{\pi_e}\ \}))}{\Delta;\ \Gamma;\ \Theta \vdash^{\overline{\pi_e}}_{\omega} \pi \Rightarrow \{\ {}^{\omega}\pi\ \}}$$

We'd like to apply O-SafePlace to show $\bullet;\ \Gamma,\ x\ :\ \tau^{\mathrm{SI}}_x \natural \mathcal{F};\ \Theta \vdash^{\overline{\pi_e}}_{\omega} \pi \Rightarrow \{\ {}^{\omega}\pi\ \}$. Let $r' \mapsto \{\ \overline{\ell}\ \} \in \Gamma$. Note that necessarily $r' \mapsto \{\ \overline{\ell}\ \} \in \Gamma,\ x\ :\ \tau^{\mathrm{SI}}_x$.

If $\forall\, {}^{\omega'}p^{\square}[\pi'] \in \{\, \overline{\ell}\, \}.(\omega = \texttt{uniq} \vee \omega' = \texttt{uniq}) \implies \pi' \mathbin{\#} \pi$ held in our original Γ, then it still holds in the extended typing $\Gamma,\ x\ :\ \tau_x^{\text{SI}} \natural \mathcal{F}$ since the loan sets are unchanged between the two stack typings.

Otherwise, we must have used the second clause $\exists \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \wedge\ \nexists \&r'\ \omega'\ \tau' \in \Theta\ \wedge (\forall \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\, \overline{\pi_e}\, \})$ in the first place. To show that this is still true, we need to show that nothing in our newly-bound x shares a type with a reborrowed reference which would end up in our exclusion list. The reason for this is that the failure condition for this is that with such a reference now bound at (or reachable within) x, we could violate the universally-quantified portion of this clause. Fortunately, we have from our premise that all the regions that appear in the type τ_x^{SI} are distinct from the ones in the exclusion list ($\forall r \in \text{free-regions}(\tau_x^{\text{SI}}).\ \forall \pi_i \in \{\, \overline{\pi}\, \}.\ \Gamma(\pi_i) = \&r''\ \omega\ \tau^{\text{XI}} \implies r'' \neq r$). Thus, we know this cannot be the case.

O-Deref

$$\frac{\begin{array}{c}\Gamma(\pi) = \&r\ \omega_\pi\ \tau_\pi \qquad \Gamma(r) = \{\, \overline{{}^{\omega'}p}^{\,n}\, \} \qquad \text{excl} = \{\, \pi_j \text{ where } j \in \{\, 1, \ldots\, n\, \} \mid p_j = p_j^{\square}[*\pi_j]\, \} \qquad \omega \lesssim \omega_\pi \\ \forall i \in \{\, 1 \ldots n\, \}.\ \Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi_e},\, \text{excl},\, \pi} p^{\square}[p_i] \Rightarrow \{\, \overline{{}^{\omega}p_i'}\, \} \\ \forall r' \mapsto \{\, \overline{\ell}\, \} \in \text{regions}(\Gamma,\ \Theta).\ (\forall\, {}^{\omega'}p'' \in \{\, \overline{\ell}\, \}.(\omega = \texttt{uniq} \vee \omega' = \texttt{uniq}) \implies p'' \mathbin{\#} p^{\square}[*\pi]) \\ \vee\ (\exists \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \wedge \nexists \&r'\ \omega'\ \tau' \in \Theta\ \wedge (\forall \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\, \overline{\pi_e},\ \text{excl},\ \pi\, \}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi_e}} p^{\square}[*\pi] \Rightarrow \{\, \overline{{}^{\omega}p_1'},\ \ldots\ \overline{{}^{\omega}p_n'},\ {}^{\omega}p^{\square}[*\pi]\, \}}$$

We'd like to apply O-Deref to show $\bullet;\ \Gamma,\ x\ :\ \tau_x^{\text{SI}} \natural \mathcal{F};\ \Theta \vdash_{\omega}^{\overline{\pi_e}} \pi \Rightarrow \{\, {}^{\omega}\pi\, \}$. This requires us to show $(\Gamma,\ x\ :\ \tau_x^{\text{SI}} \natural \mathcal{F})(\pi) = \&r\ \omega_\pi\ \tau_\pi$ and $(\Gamma,\ x\ :\ \tau_x^{\text{SI}} \natural \mathcal{F})(r) = \{\, \overline{{}^{\omega'}p}^{\,n}\, \}$. The former follows from the disjointedness assumption for x, i.e. that x is disjoint from all existing identifiers in Γ. The latter follows from the fact that no loan sets are changed between the two stack typings. Since the loan set is unchanged, we also have that the new extension for the exclusion list `excl` is the same. This leaves us with two pieces to show. First, that the recursive uses of ownership safety still succeed (for which we will use the induction hypothesis) and that our last obligation holds (which follows much as it did for O-SafePlace).

For the inductive cases, we can very nearly just apply the induction hypothesis, but we first must show that our exclusion list invariant applies for the extensions to the exclusion list. That is, we have that $\forall r \in \text{free-regions}(\tau_x^{\text{SI}}).\ \forall \pi_i \in \{\, \overline{\pi_e}\, \}.\ \Gamma(\pi_i) = \&r''\ \omega\ \tau^{\text{XI}} \implies r'' \neq r$, and we need to show $\forall r \in \text{free-regions}(\tau_x^{\text{SI}}).\ \forall \pi_i \in \{\, \text{excl},\ \pi\, \}.\ \Gamma(\pi_i) = \&r''\ \omega\ \tau^{\text{XI}} \implies r'' \neq r$. The exclusion extension `excl` is constructed by looking specifically at the reborrow loans associated with the region r. Since we know that $\forall r \in \text{free-regions}(\tau_x^{\text{SI}}).\ \forall \pi\ :\ \&r\ \omega\ \tau^{\text{XI}} \in \text{explode}(\Gamma).\ \nexists r'.\ {}^{\omega} *\pi \in \Gamma(r')$ (from inversion of NRB-Region), it follows directly that none of the places in `excl` can have a reference type with an region in τ_x^{SI}. If they did, that would mean syntactically that $\Gamma(r)$ contains a loan for $*\pi$ which would give us a contradiction.

For the last obligation, let $r' \mapsto \{\, \overline{\ell}\, \} \in \Gamma$. Note that necessarily $r' \mapsto \{\, \overline{\ell}\, \} \in \Gamma,\ x\ :\ \tau_x^{\text{SI}}$.

If $\forall\, {}^{\omega'}p'' \in \{\, \overline{\ell}\, \}.(\omega = \texttt{uniq} \vee \omega' = \texttt{uniq}) \implies p'' \mathbin{\#} p^{\square}[*\pi]$ held in our original Γ, then it still holds in the extended typing $\Gamma,\ x\ :\ \tau_x^{\text{SI}} \natural \mathcal{F}$ since the loan sets are unchanged between the two stack typings.

Otherwise, we must have used the second clause $\exists \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \wedge\ \nexists \&r'\ \omega'\ \tau' \in \Theta\ \wedge (\forall \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\, \overline{\pi_e},\ \text{excl},\ \pi\, \})$ in the first place. To show that this is still true, we need to show that nothing in our newly-bound x shares a type with a reborrowed reference which would end up in our exclusion list. The reason for this is that the failure condition for this is that with such a reference now bound at (or reachable within) x, we could violate the universally-quantified portion of this clause. Fortunately, we have from our premise that all the regions that appear in the type τ_x^{SI} are distinct from the ones in the exclusion list ($\forall r \in \text{free-regions}(\tau_x^{\text{SI}}).\ \forall \pi_i \in \{\,\overline{\pi}\,\}.\ \Gamma(\pi_i) = \&r''\ \omega\ \tau^{\text{XI}} \implies r'' \neq r$). Thus, we know this cannot be the case.

$$\textsc{O-DerefAbs}\quad \frac{\begin{array}{c}\Gamma(\pi) = \&\varrho\ \omega_\pi\ \tau_\pi \qquad \Delta;\ \Gamma \vdash_\omega p^\square[*\pi] : \tau \qquad \omega \lesssim \omega_\pi \\ \forall r' \mapsto \{\,\overline{\ell}\,\} \in \text{regions}(\Gamma,\ \Theta).\ (\forall^{\omega'} p\ \in \{\,\overline{\ell}\,\}.(\omega = \texttt{uniq} \lor \omega' = \texttt{uniq}) \implies p \,\#\, p^\square[*\pi]) \\ \lor\ (\exists \pi' : \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \land \nexists \&r'\ \omega'\ \tau' \in \Theta\ \land (\forall \pi' : \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\,\overline{\pi_e},\ \pi\,\}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash_\omega^{\overline{\pi_e}} p^\square[*\pi] \Rightarrow \{\ {}^\omega p^\square[*\pi]\ \}}$$

Since $\Delta = \bullet$, there are no valid reference types that have an abstract region, meaning the first hypothesis is a contradiction. □

Lemma E.30 (Type Computation is Preserved under Well-Typed Extensions). *If* $\Sigma;\ \bullet;\ \Gamma \vdash \tau_x^{SI}$ *and* $\bullet;\ \Gamma \vdash_\omega p : \tau,\ \{\,\overline{\rho}\,\}$ *then* $\bullet;\ \Gamma,\ \tau_x^{SI} \vdash_\omega p : \tau,\ \{\,\overline{\rho}\,\}$.

Proof. We proceed by induction on the type computation. This gives us three cases, TC-Var, TC-Proj and TC-Deref. In TC-Var, the lookup yields the same type based on the assumption that our new binding is disjoint from our existing ones. TC-Proj and TC-Deref proceed directly from the induction hypothesis. □

Lemma E.31 (Outlives is Preserved under Well-Typed Extensions). *If* $\Sigma;\ \bullet;\ \Gamma \vdash \tau_x^{SI}$ *and* $\forall r \in$ *free-regions*$(\tau_x^{SI}).\ \Gamma \vdash r\ \texttt{rnrb}$ *and* $\bullet;\ \Gamma;\ \Theta \vdash^\mu \rho_1 :> \rho_2 \dashv \Gamma'$ *then* $\bullet;\ \Gamma,\ x : \tau_x^{SI};\ \Theta \vdash^\mu \rho_1 :> \rho_2 \dashv \Gamma',\ x : \tau_x^{SI}$

Proof. We proceed by induction on the outlives judgment. This gives us six cases, OL-Refl, OL-Trans, OL-BothAbstract, OL-CombineConcrete, OL-ConcreteAbstract and OL-AbstractConcrete.

OL-Refl, OL-BothAbstract and OL-AbstractConcrete are immediate.

OL-Trans follows from the induction hypothesis.

OL-ConcreteAbstract follows from the induction hypothesis and Lemma E.30.

This leaves OL-CombineConcrete as the most interesting case. Here we use $\forall r \in \text{free-regions}(\tau_x^{\text{SI}}).\ \Gamma \vdash r\ \texttt{rnrb}$ from our premise and note that since this holds for arbitrary regions r, we know that it holds for both r_1 and r_2 in the premise of OL-CombineConcrete and thus we are done. □

Lemma E.32 (Region Rewriting is Preserved under Well-Typed Extensions). *If* $\Sigma;\ \bullet;\ \Gamma \vdash \tau_x^{SI}$ *and* $\forall r \in$ *free-regions*$(\tau_x^{SI}).\ \Gamma \vdash r\ \texttt{rnrb}$ *and* $\bullet;\ \Gamma;\ \Theta \vdash^\mu \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma'$ *then* $\bullet;\ \Gamma,\ x : \tau_x^{SI};\ \Theta \vdash^\mu \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma',\ x : \tau_x^{SI}$.

Proof. We proceed by induction on the region rewriting judgment. This gives us seven cases, RR-Refl, RR-Trans, RR-Array, RR-Slice, RR-Reference, RR-Tuple, and RR-Dead.

RR-Refl is immediate.

RR-Trans, RR-Array, RR-Slice, RR-Tuple and RR-Dead all follow directly from the induction hypothesis.

This leaves RR-Reference which follows from Lemma E.31 and the induction hypothesis. □

Lemma E.33 (Closure Bodies are Well-Typed under Well-Typed Extensions). *If* $\Sigma;\ \bullet;\ \Gamma \vdash \tau_x^{SI}$ *and* $\forall r \in$ *free-regions*$(\tau_x^{SI}).\ \Gamma \vdash r\ \texttt{rnrb}$ *and* $\Sigma;\ \bullet;\ \Gamma \natural \mathcal{F};\ \Theta \vdash \boxed{e} : \tau^{SI} \Rightarrow \Gamma' \natural \mathcal{F}'$, *then* $\Sigma;\ \bullet;\ \Gamma,\ x : \tau_x^{SI} \natural \mathcal{F};\ \Theta \vdash \boxed{e} : \tau^{SI} \Rightarrow \Gamma',\ x : \tau_x^{SI} \natural \mathcal{F}'$ *and* $\forall r \in$ *free-regions*$(\tau_x^{SI}).\ \forall \pi : \&r\ \omega\ \tau^{XI} \in$ *explode*$(\Gamma').\ \nexists r'.\ {}^\omega{*\pi} \in \Gamma'(r')$.

Proof. We proceed by induction on the typing derivation.

T-Function, T-Abort, T-Unit, T-u32, T-True, T-False, and T-Dead are all immediate.

T-LetRegion, T-While, T-ForArray, T-ForSlice, T-Closure, T-Tuple, T-Slice, T-Drop, T-Left, T-Right, and T-Shift, T-Framed, and T-ClosureValue all follow directly from the induction hypothesis.

For T-Move, T-Copy, T-Borrow, T-BorrowIndex, T-BorrowSlice, and T-IndexCopy, we rely on Lemma E.29 and the induction hypothesis for almost all of our obligations. For all of them except T-Move, we also have to show that the type computation for p still works in the extended stack typing. This follows from Lemma E.30.

For T-Branch and T-Match, we use the induction hypothesis in conjunction with Lemma E.32 to get most of the premises. In the end, we also need to deal with the union between the output environments from the two region rewriting derivations. Fortunately, we know that by definition this operation unions corresponding loan sets for the same region and as such creates no new loans leaving our not-reborrowed property intact.

T-Seq proceeds almost directly based on just the induction hypothesis, but with the required note that garbage collecting loans can only remove loans and thus leaves our not-reborrowed property intact. T-Let follows similarly to T-Seq but also requires the use of Lemma E.32.

T-Assign and T-AssignDeref follow from the induction hypothesis combined with Lemma E.32 and Lemma E.29.

T-App follows from the induction hypothesis and Lemma E.31.

T-Pointer follows from Lemma E.30. □

Lemma E.34 (Values are Well-Typed under Well-Typed Extensions). *If* $\Sigma;\ \bullet;\ \Gamma \vdash \tau_x^{SI}$ *and* $\forall r \in$ *free-regions*(τ_x^{SI}). $\Gamma \vdash r\ \texttt{rnrb}$ *and* $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau^{SI} \Rightarrow \Gamma$*, then* $\Sigma;\ \bullet;\ \Gamma\,,\ x\ :\ \tau_x^{SI};\ \Theta \vdash \boxed{v} : \tau^{SI} \Rightarrow \Gamma\,,\ x\ :\ \tau_x^{SI}$.

Proof. We proceed by induction on the typing derivation. The only non-immediate case is T-ClosureValue.

T-ClosureValue

$$\frac{\begin{array}{c}\text{free-vars}(e) \setminus \overline{x} = \overline{x_f} = \text{dom}(\mathcal{F}_c)|_{\text{VAR}} \\ \overline{r} = \overline{\text{free-regions}(\Gamma(x_f))}\,,\ (\text{free-regions}(e) \setminus (\overline{\text{free-regions}(\tau^{\text{SI}}), \text{free-regions}(\tau_r^{\text{SI}})})) = \text{dom}(\mathcal{F}_c)|_{\text{RGN}} \\ \Sigma;\ \Gamma \vdash \varsigma_c : \mathcal{F}_c \qquad \Sigma;\ \Delta;\ \Gamma \natural \mathcal{F}_c\,,\ x_1\ :\ \tau_1^{\text{SI}}\,,\ \ldots\,,\ x_n\ :\ \tau_n^{\text{SI}};\ \Theta \vdash \boxed{e} : \tau_r^{\text{SI}} \Rightarrow \Gamma' \natural \mathcal{F}\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\langle \varsigma_c\,,\ |x_1 : \tau_1^{\text{SI}}\,,\ \ldots\,,\ x_n : \tau_n^{\text{SI}}|\ \rightarrow\ \tau_r^{\text{SI}}\ \{\ e\ \}\ \rangle} : (\tau_1^{\text{SI}}\,,\ \ldots\,,\ \tau_n^{\text{SI}}) \overset{\mathcal{F}_c}{\rightarrow} \tau_r^{\text{SI}} \Rightarrow \Gamma}$$

First, we invert the stack frame typing hypothesis to get that $\forall x \in \text{dom}(\varsigma)$. $\Sigma;\ \bullet;\ \Gamma \natural \mathcal{F}_c;\ \Theta \vdash \boxed{\varsigma(x)} : \mathcal{F}_c(x) \Rightarrow \Gamma \natural \mathcal{F}_c$. We can apply the induction hypothesis to each of these statements, and apply WF-Frame to get $\Sigma;\ \Gamma\,,\ x\ :\ \tau_x^{\text{SI}} \vdash \varsigma_c : \mathcal{F}_c$.

Next, we need to show that $\Sigma;\ \bullet;\ \Gamma\,,\ x\ :\ \tau_x^{\text{SI}} \natural \mathcal{F}_c\,,\ x_1\ :\ \tau_1^{\text{SI}}\,,\ \ldots\,,\ x_n\ :\ \tau_n^{\text{SI}};\ \Theta \vdash \boxed{e} : \tau_r^{\text{SI}} \Rightarrow \Gamma'' \natural \mathcal{F}'$ for some Γ'' and $\mathcal{F}'$. We get this by applying Lemma E.33 to $\Sigma;\ \bullet;\ \Gamma \natural \mathcal{F}_c\,,\ x_1\ :\ \tau_1^{\text{SI}}\,,\ \ldots\,,\ x_n\ :\ \tau_n^{\text{SI}};\ \Theta \vdash \boxed{e} : \tau_r^{\text{SI}} \Rightarrow \Gamma' \natural \mathcal{F}'$ (from the premise of T-ClosureValue). □

Lemma E.35 (Stack Validity is Preserved under Well-Typed Extensions). *If* $\Sigma \vdash \sigma : \Gamma$ *and* $\forall r \in$ *free-regions*(τ_x^{SI}). $\Gamma \vdash r\ \texttt{rnrb}$ *and* $\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau^{SI} \Rightarrow \Gamma$*, then* $\Sigma \vdash \sigma\,,\ x \mapsto v : \Gamma\,,\ x\ :\ \tau^{SI}$.

Proof. This proof follows directly from the definition of WF-StackFrame.

WF-StackFrame

$$\frac{\begin{array}{c}\Sigma \vdash \sigma : \Gamma \qquad \text{dom}(\varsigma) = \text{dom}(\mathcal{F})|_x \\ \forall x \in \text{dom}(\varsigma).\ \Sigma;\ \bullet;\ \Gamma \natural \mathcal{F};\ \bullet \vdash \boxed{(\sigma \natural \varsigma)(x)} : (\Gamma \natural \mathcal{F})(x) \Rightarrow \Gamma \natural \mathcal{F}\end{array}}{\Sigma \vdash \sigma \natural \varsigma : \Gamma \natural \mathcal{F}}$$

In particular, inversion of WF-StackFrame on $\Sigma \vdash \sigma : \Gamma$ gives us well-formedness for the remainder of the stack, $\text{dom}(\varsigma) = \text{dom}(\mathcal{F})|_x$ and $\forall x \in \text{dom}(\varsigma).\ \Sigma;\ \bullet;\ \Gamma \natural \mathcal{F};\ \bullet \vdash \boxed{(\sigma \natural \varsigma)(x)} : (\Gamma \natural \mathcal{F})(x) \Rightarrow \Gamma \natural \mathcal{F}$. We can then apply Lemma E.34 to each of these derivations to get $\forall x \in \text{dom}(\varsigma,\ x \mapsto v).\ \Sigma;\ \bullet;\ \Gamma \natural \mathcal{F},\ x\ :\ \tau^{\text{SI}};\ \bullet \vdash \boxed{(\varsigma,\ x \mapsto v)(x)} : (\Gamma \natural \mathcal{F},\ x\ :\ \tau^{\text{SI}})(x) \Rightarrow \Gamma \natural \mathcal{F},\ x\ :\ \tau^{\text{SI}}$. We can then see that the well-formedness of the remainder of the stack is unaffected, and that the domains when extended with x remain equal. The last obligation is to show that the v is well-typed in the current stack typing, but we already have that from our premise. Thus, we can apply WF-StackFrame with the extended stack to get $\Sigma \vdash \sigma,\ x \mapsto v : \Gamma,\ x\ :\ \tau^{\text{SI}}$. □

E.7 Preservation after Assignment Lemmas

Lemma E.36 (Ownership Safety is Preserved after Assignment). *If* $\Gamma(\pi_a) = \tau^{SX}$ *and* $\Delta;\ \Gamma \trianglerighteq *\pi_a;\ \Theta \vdash^{=} \tau^{SI} \rightsquigarrow \tau^{SX} \dashv \Gamma'$ *and* τ^{SX} *is unique to* π_a *in* Γ *and* $\bullet;\ \Gamma \natural \mathcal{F};\ \Theta \vdash_{\omega}^{\overline{\pi_o}} p \Rightarrow \{\ \overline{\ell}\ \}$ *and* $\{\ \overline{\pi_n}\ \} = \{\ \overline{\pi_o}\ \} \setminus \pi_a$*, then* $\bullet;\ \textit{gc-loans}_{\Theta}(\Gamma'[\pi_a \mapsto \tau^{SI}]) \natural \mathcal{F};\ \Theta \vdash_{\omega}^{\overline{\pi_n}} p \Rightarrow \{\ \overline{\ell}'\ \}$.

Proof. We proceed by induction on the ownership safety derivation.

O-SafePlace

$$\frac{\begin{array}{c}\forall r' \mapsto \{\ \overline{\ell}\ \} \in \text{regions}(\Gamma,\ \Theta).\ (\forall\, {}^{\omega'}p^{\square}[\pi']\ \in \{\ \overline{\ell}\ \}.(\omega = \texttt{uniq} \lor \omega' = \texttt{uniq}) \implies \pi' \# \pi) \\ \lor\ (\exists \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \land \nexists \&r'\ \omega'\ \tau' \in \Theta\ \land (\forall \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\ \overline{\pi_e}\ \}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi_e}} \pi \Rightarrow \{\ {}^{\omega}\pi\ \}}$$

Consider an arbitrary region r from the domain of $\Gamma \natural \mathcal{F}$. For this r, we wish to show that either of the two clauses in O-SafePlace which were previously true are maintained after going through $\Delta;\ \Gamma \trianglerighteq *\pi_a;\ \Theta \vdash^{=} \tau^{\text{SI}} \rightsquigarrow \tau^{\text{SX}} \dashv \Gamma'$ and $\text{gc-loans}_{\Theta}(\cdot)$, which kills loans prefixed by $*\pi$, checks that the new type for π_a is compatible with its old type, and clears out loans associated with its outermost region. So, we will consider each clause as a separate case.

We will first consider the case where we have $\forall\, {}^{\omega'}p^{\square}[\pi']\ \in \{\ \overline{\ell}\ \}.(\omega = \texttt{uniq} \lor \omega' = \texttt{uniq}) \implies \pi' \# \pi$. In this case, we know by definition of $\trianglerighteq$ that $\forall r \in \text{dom}(\Gamma).\ (\Gamma \trianglerighteq *\pi_a)(r) \subseteq \Gamma(r)$. We then know, again by definition (see OL-CheckConcrete), that $\forall r \in \text{dom}(\Gamma).\ (\Gamma \trianglerighteq *p)(r) = \Gamma'(r)$. As such, we know that the $\trianglerighteq$ and $\text{gc-loans}_{\Theta}(\cdot)$ has at most shrank the obligations in this case to having fewer disjointedness obligations, and it is otherwise unchanged.

This leaves us to consider the second case $\exists \pi'\ :\ \&r'\,\omega'\,\tau' \in \text{explode}(\Gamma)\ \land\ \nexists \&r'\,\omega'\,\tau' \in \Theta\ \land(\forall \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\ \overline{\pi_e}\ \})$. Recall that, definitionally, neither $\trianglerighteq$ nor the region rewriting judgment change variable bindings and their associated types in the environment (instead both affect only the loan sets associated with regions, though the latter does not when run in the checking mode =). Thus, we know that $\text{explode}(\Gamma) = \text{explode}(\Gamma')$. We then need to consider two distinct possibilities for how the exclusion list has changed. We know from the premise that $\{\overline{\pi_n}\} = \{\overline{\pi_o}\}\setminus\pi_a$ which means that either the two sets are exactly identical (when $\pi_a \notin \{\ \overline{\pi_o}\ \}$) or smaller by π_a in particular (when $\pi_a \in \{\ \overline{\pi_o}\ \}$). In the former case, the exclusion list is unchanged which means the whole clause is true for every r' in Γ' for which it was true in Γ. In the latter case, the regions r' is in the type of π_a which *has* been removed from the exclusion list $\{\ \overline{\pi_n}\ \}$. Thus, we need to show for the loans $\{\ \overline{\ell}\ \}$ associated with r' that $\forall\, {}^{\omega'}p^{\square}[\pi']\ \in \{\ \overline{\ell}\ \}.(\omega = \texttt{uniq} \lor \omega' = \texttt{uniq}) \implies \pi' \# \pi$. By definition, τ^{SX} is unique to π in Γ tells us that the outermost region r' is unique to the type τ^{SX} and place π_a, and thus when we replace it with τ^{SI}, we ensure that r' does not occur in any type in Γ'. Thus, the surrounding call of $\text{gc-loans}_{\Theta}(\cdot)$ necessarily clears out the loan set meaning that the set associated with r' is always empty in new environment, meaning the disjointness condition from O-SafePlace holds trivially.

O-Deref

$$\dfrac{\begin{array}{c}\Gamma(\pi) = \&r\ \omega_\pi\ \tau_\pi \qquad \Gamma(r) = \{\ \overline{{}^{\omega'}p}^{\,n}\ \} \qquad \mathrm{excl} = \{\ \pi_j \text{ where } j \in \{\ 1,\ \ldots\ n\ \} \mid p_j = p^\square_j[*\pi_j]\ \} \qquad \omega \lesssim \omega_\pi \\ \forall i \in \{\ 1\ \ldots\ n\ \}.\ \Delta;\ \Gamma;\ \Theta \vdash^{\overline{\pi_e},\,\mathrm{excl},\,\pi}_{\omega} p^\square[p_i] \Rightarrow \{\ \overline{{}^{\omega}p'_i}\ \} \\ \forall r' \mapsto \{\ \bar{\ell}\ \} \in \mathrm{regions}(\Gamma,\ \Theta).\ (\forall {}^{\omega'}p'' \in \{\ \bar{\ell}\ \}.(\omega = \mathtt{uniq} \lor \omega' = \mathtt{uniq}) \implies p'' \# p^\square[*\pi]) \\ \lor\ (\exists \pi' : \&r'\ \omega'\ \tau' \in \mathrm{explode}(\Gamma)\ \land \nexists \&r'\ \omega'\ \tau' \in \Theta\ \land (\forall \pi' : \&r'\ \omega'\ \tau' \in \mathrm{explode}(\Gamma).\ \pi' \in \{\ \overline{\pi_e},\ \mathrm{excl},\ \pi\ \})) \end{array}}{\Delta;\ \Gamma;\ \Theta \vdash^{\overline{\pi_e}}_{\omega} p^\square[*\pi] \Rightarrow \{\ \overline{{}^{\omega}p'_1},\ \ldots\ \overline{{}^{\omega}p'_n},\ {}^{\omega}p^\square[*\pi]\ \}}$$

We want to produce a new derivation using O-Deref for $\bullet;\ \Gamma \trianglerighteq *\pi_a \natural \mathcal{F};\ \Theta \vdash^{\overline{\pi_e}}_{\omega} p^\square[*\pi] \Rightarrow \{\ \overline{\ell''}\ \}$. We have $(\Gamma \natural \mathcal{F})(\pi) = \&r\ \omega_\pi\ \tau_\pi$ from the premise of O-Deref. We then know by the definition of $\trianglerighteq$ that $(\Gamma \trianglerighteq *\pi_a \natural \mathcal{F})(\pi) = \&r\ \omega_\pi\ \tau_\pi$ since $\trianglerighteq$ only affects the loan set portion of the codomain of its input environment. We also know from the premise of O-Deref that $(\Gamma \natural \mathcal{F})(r) = \{\ \overline{{}^{\omega'}p}^{\,n}\ \}$. Again, by the definition of $\trianglerighteq$, we have that $(\Gamma \trianglerighteq *\pi_a \natural \mathcal{F})(r) = \{\ \overline{{}^{\omega^k}p^k}^{\,m}\ \}$ where we know $m \le n$ and $\{\ \overline{{}^{\omega^k}p^k}^{\,m}\ \} \subseteq \{\ \overline{{}^{\omega'}p}^{\,n}\ \}$.

We know by the definition of $\trianglerighteq$ also that every place expression p_d in $\{\ \overline{{}^{\omega'}p}^{\,n}\ \} \setminus \{\ \overline{{}^{\omega^k}p^k}^{\,m}\ \}$ can be decomposed into $p^\square_d[*\pi_a]$. This means that for computing the set excl, it is either the same or has shrunk by precisely π_a. This lines up with our induction hypothesis which we apply to each of $\forall i \in \{\ 1\ \ldots\ n\ \}.\ \bullet;\ \Gamma;\ \Theta \vdash^{\overline{\pi_e},\,\mathrm{excl},\,\pi}_{\omega} p^\square[p_i] \Rightarrow \{\ \overline{{}^{\omega}p'_i}\ \}$ from the premise of O-Deref. This gives us $\forall i \in \{\ 1\ \ldots\ n\ \}.\ \bullet;\ \Gamma \trianglerighteq *\pi_a;\ \Theta \vdash^{\overline{\pi_e},\,\mathrm{excl},\,\pi}_{\omega} p^\square[p_i] \Rightarrow \{\ \overline{{}^{\omega}p''_i}\ \}$.

Finally, for the last premise of O-Deref, the proof precedes identically to the case for O-SafePlace since the obligation is exactly the same.

O-DerefAbs

$$\dfrac{\begin{array}{c}\Gamma(\pi) = \&\varrho\ \omega_\pi\ \tau_\pi \qquad \Delta;\ \Gamma \vdash_{\omega} p^\square[*\pi] : \tau \qquad \omega \lesssim \omega_\pi \\ \forall r' \mapsto \{\ \bar{\ell}\ \} \in \mathrm{regions}(\Gamma,\ \Theta).\ (\forall {}^{\omega'}p \in \{\ \bar{\ell}\ \}.(\omega = \mathtt{uniq} \lor \omega' = \mathtt{uniq}) \implies p \# p^\square[*\pi]) \\ \lor\ (\exists \pi' : \&r'\ \omega'\ \tau' \in \mathrm{explode}(\Gamma)\ \land \nexists \&r'\ \omega'\ \tau' \in \Theta\ \land (\forall \pi' : \&r'\ \omega'\ \tau' \in \mathrm{explode}(\Gamma).\ \pi' \in \{\ \overline{\pi_e},\ \pi\ \})) \end{array}}{\Delta;\ \Gamma;\ \Theta \vdash^{\overline{\pi_e}}_{\omega} p^\square[*\pi] \Rightarrow \{\ {}^{\omega}p^\square[*\pi]\ \}}$$

Since $\Delta = \bullet$, there are no valid reference types that have an abstract region, meaning the first hypothesis is a contradiction. □

Lemma E.37 (Outlives is Preserved after Assignment). *If* $\Gamma(\pi_a) = \tau^{SX}$ *and* $\Delta;\ \Gamma \trianglerighteq *\pi_a;\ \Theta \vdash^{=} \tau^{SI} \rightsquigarrow \tau^{SX} \dashv \Gamma'$ *and* $\bullet;\ \Gamma;\ \Theta \vdash^{\mu} \rho_1 :> \rho_2 \dashv \Gamma_o$, *then* $\bullet;\ \Gamma';\ \Theta \vdash^{\mu} \rho_1 :> \rho_2 \dashv \Gamma'_o$.

Proof. We proceed by induction on the outlives judgment. This gives us seven cases, OL-Refl, OL-Trans, OL-BothAbstract, OL-CombineConcrete, OL-CombineConcrete, OL-ConcreteAbstract and OL-AbstractConcrete.

OL-Refl, OL-BothAbstract and OL-AbstractConcrete are immediate.

OL-Trans follows from the induction hypothesis.

OL-ConcreteAbstract follows from the induction hypothesis and noting that the type computation does not depend on the contents of loan sets.

This leaves OL-CombineConcreteUnrestricted, OL-CombineConcrete and OL-CheckConcrete as the most interesting cases. We note that the checking mode = corresponds to making no changes to the environment, thus $\Gamma' = \Gamma \trianglerighteq *\pi_a$. Then, for each, we note that the value of each associated loan set in the input environment only has an effect on the output environment and not whether or not the rule applies. Thus, since we know that Γ' (compared to Γ) has had some loans removed (those rooted at $*\pi_a$), then we can still produce a derivation, only with a different, potentially smaller output. □

Lemma E.38 (Region Rewriting is Preserved after Assignment). *If* $\Gamma(\pi_a) = \tau^{SX}$ *and* $\Delta;\ \Gamma \rhd *\pi_a;\ \Theta \vdash^{=} \tau^{SI} \rightsquigarrow \tau^{SX} \dashv \Gamma'$ *and* $\bullet;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma_o$, *then* $\bullet;\ \Gamma';\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma'_o$.

Proof. We proceed by induction on the region rewriting judgment. This gives us seven cases, RR-Refl, RR-Trans, RR-Array, RR-Slice, RR-Reference, RR-Tuple, and RR-Dead.

RR-Refl is immediate.

RR-Trans, RR-Array, RR-Slice, RR-Tuple and RR-Dead all follow directly from the induction hypothesis.

This leaves RR-Reference which follows from Lemma E.37 and the induction hypothesis. □

Lemma E.39 (Expressions are Well-Typed after Assignment). *If* $\Gamma(\pi_a) = \tau^{SX}$ *and* $\Delta;\ \Gamma \rhd *\pi_a;\ \Theta \vdash^{=} \tau^{SI} \rightsquigarrow \tau^{SX} \dashv \Gamma'$ *and* $\bullet; \Gamma'; \Theta \vdash_{\texttt{uniq}} \pi_a \Rightarrow \{\ ^{\texttt{uniq}}\pi_a\ \}$ *and* $\Sigma;\ \bullet;\ \Gamma \natural \mathcal{F};\ \Theta \vdash \boxed{e} : \tau^{SI} \Rightarrow \Gamma_o \natural \mathcal{F}'$, *then* $\Sigma;\ \bullet;\ \textit{gc-loans}_{\Theta}(\Gamma'[\pi_a \mapsto \tau^{SI}]) \natural \mathcal{F};\ \Theta \vdash \boxed{e} : \tau^{SI} \Rightarrow \Gamma'_o[\pi_a \mapsto \tau^{SI}] \natural \mathcal{F}''$.

Proof. We proceed by induction on the typing derivation.

T-Function, T-Abort, T-Unit, T-u32, T-True, T-False, and T-Dead are all immediate.

T-Seq, T-LetRegion, T-While, T-ForArray, T-ForSlice, T-Closure, T-Tuple, T-Slice, T-Drop, T-Left, T-Right, and T-Shift, T-Framed, and T-ClosureValue all follow directly from the induction hypothesis.

For T-Move, T-Copy, T-Borrow, T-BorrowIndex, T-BorrowSlice, and T-IndexCopy, we rely on Lemma E.36 and the induction hypothesis for almost all of our obligations. For all of them except T-Move, we also have to show that the type computation for p still works in the updated environment. Since we know that π_a is ownership safe from our premise, we know that p is disjoint from π_a and thus the type update could not affect its type computation. Otherwise, the only difference is in loan sets associated with regions, and thus does not affect type computation.

T-Pointer requires the same argument about type computation as in T-Borrow, but does not need the additional lemmas or the induction hypothesis.

For T-Branch and T-Match, we use the induction hypothesis in conjunction with Lemma E.38 to get most of

the premises. In the end, we also need to deal with the union between the output environments from the two rewriting derivations. Fortunately, we know that by definition this operation unions corresponding loan sets for the same region and so commutes with gc-loans$_{\Theta}(\cdot)$ and the type update.

T-Let follows similarly to T-Branch and T-Match using the induction hypothesis in conjunction with Lemma E.38 without the need to address a combined environment.

T-Assign and T-AssignDeref follow from the induction hypothesis combined with Lemma E.38 and Lemma E.36.

T-App follows from the induction hypothesis and Lemma E.37. □

Lemma E.40 (Values are Well-Typed after Assignment). *If* $\Gamma(\pi_a) = \tau^{SX}$ *and* $\Delta;\ \Gamma \rhd *\pi_a;\ \Theta \vdash^{=} \tau^{SI} \rightsquigarrow \tau^{SX} \dashv \Gamma'$ *and* $\bullet;\ \Gamma';\ \Theta \vdash_{\texttt{uniq}} \pi_a \Rightarrow \{\ ^{\texttt{uniq}}\pi_a\ \}$ *and* $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau \Rightarrow \Gamma$, *then* $\Sigma;\ \bullet;\ \textit{gc-loans}_{\Theta}(\Gamma'[\pi_a \mapsto \tau^{SI}]);\ \Theta \vdash \boxed{v} : \tau \Rightarrow \textit{gc-loans}_{\Theta}(\Gamma'[\pi_a \mapsto \tau^{SI}])$.

Proof. We proceed by induction on the value typing derivation. The only non-immediate cases are T-Pointer and T-ClosureValue.

T-Pointer

$$\frac{\Sigma;\ \Gamma \vdash \mathcal{R}^{\square}[\pi] : \tau^{\text{XI}} \qquad {}^{\omega}\pi \in \Gamma(r)}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{ptr}\ \mathcal{R}^{\square}[\pi]} : \&r\ \omega\ \tau^{\text{XI}} \Rightarrow \Gamma}$$

In T-Pointer, we have a requirement that ${}^{\omega}\pi \in \Gamma(r)$ which could potentially be affected by the kill rules. However, note that the definition of $\trianglerighteq$ is such that we only remove loans of the form $*\pi_a$ which necessarily cannot match this loan which has no dereference in it. Thus, we know that ${}^{\omega}\pi \in (\Gamma \trianglerighteq *\pi)(r)$ and thus, $\Sigma; \bullet; \Gamma \trianglerighteq *\pi_a; \Theta \vdash \boxed{\texttt{ptr}\ \mathcal{R}^{\square}[\pi]} : \tau^{\text{SI}} \Rightarrow \Gamma \trianglerighteq *\pi_a$. We then know that the checking mode = for rewriting does not change the output environment, and thus, $\Gamma' = \Gamma \trianglerighteq *\pi_a$. Then, we know that $\pi \neq \pi_a$ (this would otherwise conflict with the ownership safety derivation for π_a in our premise), so the type update for π_a does not impact T-Pointer. Finally, the call to $\text{gc-loans}_{\Theta}(\cdot)$ clears out any unused regions, but the region r here is still in use and thus not changed.

T-ClosureValue

$$\dfrac{\begin{array}{c}\text{free-vars}(e) \setminus \overline{x} = \overline{x_f} = \text{dom}(\mathcal{F}_c)|_{\text{VAR}} \\ \overline{r} = \overline{\text{free-regions}(\Gamma(x_f))},\ (\text{free-regions}(e) \setminus (\overline{\text{free-regions}(\tau^{\text{SI}})}, \text{free-regions}(\tau_r^{\text{SI}}))) = \text{dom}(\mathcal{F}_c)|_{\text{RGN}} \\ \Sigma; \Gamma \vdash \varsigma_c : \mathcal{F}_c \qquad \Sigma; \Delta; \Gamma \natural \mathcal{F}_c,\ x_1 : \tau_1^{\text{SI}}, \ldots, x_n : \tau_n^{\text{SI}}; \Theta \vdash \boxed{e} : \tau_r^{\text{SI}} \Rightarrow \Gamma' \natural \mathcal{F}\end{array}}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{\langle \varsigma_c,\ |x_1 : \tau_1^{\text{SI}}, \ldots, x_n : \tau_n^{\text{SI}}| \to \tau_r^{\text{SI}}\ \{\, e \,\}\ \rangle} : (\tau_1^{\text{SI}}, \ldots, \tau_n^{\text{SI}}) \overset{\mathcal{F}_c}{\to} \tau_r^{\text{SI}} \Rightarrow \Gamma}$$

First, we invert the stack frame typing hypothesis to get that $\forall x \in \text{dom}(\varsigma).\ \Sigma; \bullet; \Gamma \natural \mathcal{F}_c; \Theta \vdash \boxed{\varsigma(x)} : \mathcal{F}_c(x) \Rightarrow \Gamma \natural \mathcal{F}_c$. We can apply the induction hypothesis to each of these statements, and apply WF-Frame to get $\Sigma;\ \text{gc-loans}_{\Theta}(\Gamma'[\pi_a \mapsto \tau^{\text{SI}}]) \vdash \varsigma_c : \mathcal{F}_c$.

Next, we need to show that $\Sigma; \bullet; \text{gc-loans}_{\Theta}(\Gamma'[\pi_a \mapsto \tau^{\text{SI}}]) \natural \mathcal{F}_c,\ x_1 : \tau_1^{\text{SI}}, \ldots, x_n : \tau_n^{\text{SI}}; \Theta \vdash \boxed{e} : \tau_r^{\text{SI}} \Rightarrow \Gamma'' \natural \mathcal{F}'$ for some Γ'' and $\mathcal{F}'$. We get this by applying Lemma E.39 to $\Sigma; \bullet; \Gamma \natural \mathcal{F}_c,\ x_1 : \tau_1^{\text{SI}}, \ldots, x_n : \tau_n^{\text{SI}}; \Theta \vdash \boxed{e} : \tau_r^{\text{SI}} \Rightarrow \Gamma' \natural \mathcal{F}'$ from the premise of T-ClosureValue. □

Lemma E.41 (Stack Validity is Preserved after Assignment). *If $\Sigma \vdash \sigma : \Gamma$ and $\Gamma(\pi) = \tau^{SX}$ and $\Sigma; \bullet; \Gamma; \Theta \vdash \boxed{v} : \tau^{SI} \Rightarrow \Gamma$ and $\bullet; \Gamma \trianglerighteq *\pi; \Theta \vdash^{\mu} \tau^{SI} \rightsquigarrow \tau^{SX} \dashv \Gamma'$ and $\bullet; \Gamma'; \Theta \vdash_{\texttt{uniq}} \pi \Rightarrow \{\ {}^{\texttt{uniq}}\pi\ \}$ and $\sigma \vdash \pi \Downarrow \pi \mapsto \mathcal{V}[\_]$ and $\pi = x.q$, then $\Sigma \vdash \sigma[x \mapsto \mathcal{V}[v]] : \text{gc-loans}_{\Theta}(\Gamma'[\pi \mapsto \tau^{SI}])$.*

Proof. The proof proceeds by induction on the stack validity judgment $\Sigma \vdash \sigma : \Gamma$ which has two cases, WF-StackEmpty and WF-StackFrame.

WF-StackEmpty

$$\dfrac{}{\Sigma \vdash \bullet : \bullet}$$

In this case, the stack is empty and therefore, we have a contradiction since our premise says that $\Gamma(\pi) = \tau^{\text{SX}}$, but $\Gamma = \bullet$ and $\bullet(\pi)$ necessarily fails.

WF-StackFrame

$$\dfrac{\begin{array}{c}\Sigma \vdash \sigma : \Gamma \qquad \text{dom}(\varsigma) = \text{dom}(\mathcal{F})|_x \\ \forall x \in \text{dom}(\varsigma).\ \Sigma; \bullet; \Gamma \natural \mathcal{F}; \bullet \vdash \boxed{(\sigma \natural \varsigma)(x)} : (\Gamma \natural \mathcal{F})(x) \Rightarrow \Gamma \natural \mathcal{F}\end{array}}{\Sigma \vdash \sigma \natural \varsigma : \Gamma \natural \mathcal{F}}$$

In the premise of WF-StackFrame, we have a collection of typing judgments for values stored in the stack. This naturally leads us to another case split: either x (the root of π from $\pi = x.q$) is in the current frame or it is not.

If x is not in the current frame, we apply our induction hypothesis to $\Sigma \vdash \sigma : \Gamma$ to get $\Sigma \vdash \sigma[x \mapsto \mathcal{V}[v]] : \Gamma'[\pi \mapsto \tau^{\text{SI}}]$. Then, we apply WF-StackFrame with the same typing judgments we already have to reach our overall conclusion of $\Sigma \vdash (\sigma \natural \varsigma)[x \mapsto \mathcal{V}[v]] : (\Gamma' \natural \mathcal{F})[\pi \mapsto \tau^{\text{SI}}]$ (noting that substituting inside or outside is definitionally equal when we know that $x \notin \text{dom}(\sigma)$).

If x is in the current frame, then we apply Lemma E.13 to $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau^{\text{SI}} \Rightarrow \Gamma$ and $\bullet;\ \Gamma;\ \Theta \vdash^{\mu} \tau^{\text{SI}} \rightsquigarrow \tau^{\text{SX}} \dashv \Gamma'$ (both from our premise) to get $\Sigma;\ \bullet;\ \Gamma';\ \Theta \vdash \boxed{v} : \tau^{\text{SI}} \Rightarrow \Gamma'$. Then, we note that it would be a well-formedness violation for this value to depend on x itself (since that would mean it was a cyclical reference) and thus, we can get that $\Sigma;\ \bullet;\ \Gamma'[\pi \mapsto \tau^{\text{SI}}];\ \Theta \vdash \boxed{v} : \tau^{\text{SI}} \Rightarrow \Gamma'[\pi \mapsto \tau^{\text{SI}}]$. Finally, we can garbage collect the loans from the old type to get $\Sigma;\ \bullet;\ \text{gc-loans}_{\Theta}(\Gamma'[\pi \mapsto \tau^{\text{SI}}]);\ \Theta \vdash \boxed{v} : \tau^{\text{SI}} \Rightarrow \text{gc-loans}_{\Theta}(\Gamma'[\pi \mapsto \tau^{\text{SI}}])$

For the other typing judgments in this frame, we apply Lemma E.40 to get $\forall x \in \text{dom}(\varsigma).\ \Sigma;\ \bullet;\ \text{gc-loans}_{\Theta}(\Gamma'[\pi \mapsto \tau^{\text{SI}}]);\ \bullet \vdash \boxed{(\sigma \natural \varsigma)(x)} : (\Gamma' \natural \mathcal{F})(x) \Rightarrow \text{gc-loans}_{\Theta}(\Gamma'[\pi \mapsto \tau^{\text{SI}}])$. Thus, we can apply WF-StackFrame to conclude $\Sigma \vdash \sigma[x \mapsto \mathcal{V}[v]] : \text{gc-loans}_{\Theta}(\Gamma'[\pi \mapsto \tau^{\text{SI}}])$. □

E.8 Values are Well-Types at Rewritten Types Lemma

Lemma E.42 (Values are Well-Typed At Rewritten Types). *If* $\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau^{SI} \Rightarrow \Gamma_i$ *and* $\Delta;\ \Gamma_i;\ \Theta \vdash^{\mu} \tau^{SI} \rightsquigarrow \tau^{SI\prime} \dashv \Gamma'$, *then* $\Sigma;\ \Delta;\ \Gamma';\ \Theta \vdash \boxed{v} : \tau^{SI\prime} \Rightarrow \Gamma'$.

Proof. We proceed by induction on the value typing relation.

In the case of T-Tuple, we need to apply the induction hypothesis for each entry which has a changed type, and Lemma E.13 for each entry which does not.

In the case of T-Array, we just apply the induction hypothesis to each entry.

T-Pointer

$$\frac{\Sigma;\ \Gamma \vdash \mathcal{R}^{\square}[\pi] : \tau^{\text{XI}} \qquad {}^{\omega}\pi \in \Gamma(r)}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{ptr}\ \mathcal{R}^{\square}[\pi]} : \&r\ \omega\ \tau^{\text{XI}} \Rightarrow \Gamma}$$

For the T-Pointer case, we proceed by induction on the region rewriting judgement. The only interesting cases are for reference types. From there, we proceed by induction on the outlives relation, for which the only interesting case is OL-CombineConcrete.

OL-CombineConcrete

$$\frac{\begin{array}{ccc}\Gamma \vdash r_1\ \texttt{rnrb} & \Gamma \vdash r_2\ \texttt{rnrb} & \Gamma;\ \Theta \vdash \{\, r_1,\ r_2 \,\}\ \texttt{clrs} \\ \multicolumn{2}{c}{r_1\ \texttt{occurs before}\ r_2\ \texttt{in}\ \Gamma} & \{\, \overline{\ell} \,\} = \Gamma(r_1) \cup \Gamma(r_2)\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash^{+} r_1 :> r_2 \dashv \Gamma[r_2 \mapsto \{\, \overline{\ell} \,\}]}$$

The T-Pointer case is immediate. We know that the referent type is preserved since we do not change any types in the context, and we know the loan is preserved since loan sets only grow.

In all other cases, we know the types cannot change, which means $\Gamma = \Gamma'$, so we are done. □

E.9 Function Definitions are Self-Contained Lemma

Lemma E.43 (Function Definitions are Self-Contained).
If $\vdash \Sigma;\ \bullet;\ \Gamma;\ \Theta$ *and* $\Sigma(f) = \texttt{fn}\ f\texttt{<}\overline{\varphi},\ \overline{\varrho},\ \overline{\alpha}\texttt{>}(x_1 : \tau_1^{SI},\ \ldots,\ x_n : \tau_n^{SI}) \rightarrow \tau_r^{SI}\ \texttt{where}\ \overline{\varrho_1 : \varrho_2}\ \{\, e \,\}$, *then* $\Sigma;\ \overline{\varphi : \texttt{FRM}},\ \overline{\varrho : \texttt{RGN}},\ \overline{\varrho_1 :> \varrho_2},\ \overline{\alpha : \star};\ \Gamma \natural\, x_1 : \tau_1^{SI},\ \ldots,\ x_n : \tau_n^{SI};\ \Theta \vdash \boxed{\texttt{framed}\ e} : \tau_f^{SI} \Rightarrow \Gamma$.

Proof. Begin by noting that WF-FunctionDefinition gives us that $\Sigma;\ \overline{\varphi : \texttt{FRM}},\ \overline{\varrho : \texttt{RGN}},\ \overline{\varrho_1 :> \varrho_2},\ \overline{\alpha : \star};\ \bullet \natural\, x_1 : \tau_1^{\text{SI}},\ \ldots,\ x_n : \tau_n^{\text{SI}};\ \bullet \vdash \boxed{e} : \tau_f^{\text{SI}} \Rightarrow \Gamma'$. We also have by inspection of the typing rules that $\Gamma' = \bullet \natural \mathcal{F}'$ for some frame $\mathcal{F}'$. Then by T-Framed, it suffices to show that $\Sigma;\ \overline{\varphi : \texttt{FRM}},\ \overline{\varrho : \texttt{RGN}},\ \overline{\varrho_1 :> \varrho_2},\ \overline{\alpha : \star};\ \Gamma \natural\, x_1 : \tau_1^{\text{SI}},\ \ldots,\ x_n : \tau_n^{\text{SI}};\ \Theta \vdash \boxed{e} : \tau_f^{\text{SI}} \Rightarrow \Gamma \natural \mathcal{F}'$. But note that this is immediate. The typing derivation with $\bullet$ and the current frame means that there's absolutely no reliance on context outside $x_1, \ldots x_n$, and these places are necessarily completely disjoint from places in Γ since any regions in their types must be abstract. □

E.10 Subset Related Environments Lemma

Lemma E.44 (Outlives Produces Subset-Related Environments). *If* $\Delta;\ \Gamma;\ \Theta \vdash^{\mu} r_1 :> r_2 \dashv \Gamma'$, *then* $\forall r.\ \Gamma(r) \subseteq \Gamma'(r)$.

Proof. The proof proceeds by induction on the outlives relation $\Delta;\ \Gamma;\ \Theta \vdash^{\mu} r_1 :> r_2 \dashv \Gamma'$. We will consider each case.

$$\textsc{OL-Refl}\quad \frac{}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \rho :> \rho \dashv \Gamma}$$

$$\textsc{OL-BothAbstract}\quad \frac{\varrho_1 : \mathsf{RGN} \in \Delta \qquad \varrho_2 : \mathsf{RGN} \in \Delta \qquad \varrho_1 :> \varrho_2 \in \Delta}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \varrho_1 :> \varrho_2 \dashv \Gamma}$$

$$\textsc{OL-AbstractConcrete}\quad \frac{\varrho : \mathsf{RGN} \in \Delta \qquad r \in \mathrm{dom}(\Gamma)}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \varrho :> r \dashv \Gamma}$$

$$\textsc{OL-CheckConcrete}\quad \frac{\Gamma \vdash r_1\ \mathsf{rnrb} \qquad \Gamma \vdash r_2\ \mathsf{rnrb} \qquad r_1\ \mathsf{occurs\ before}\ r_2\ \mathsf{in}\ \Gamma}{\Delta;\ \Gamma;\ \Theta \vdash^{=} r_1 :> r_2 \dashv \Gamma}$$

Each of OL-Refl, OL-BothAbstract, OL-AbstractConcrete, and OL-CheckConcrete are immediate since $\Gamma' = \Gamma$.

$$\textsc{OL-Trans}\quad \frac{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \varrho_1 :> \varrho_2 \dashv \Gamma' \qquad \Delta;\ \Gamma';\ \Theta \vdash^{\mu} \varrho_2 :> \varrho_3 \dashv \Gamma''}{\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \varrho_1 :> \varrho_3 \dashv \Gamma''}$$

$$\textsc{OL-ConcreteAbstract}\quad \frac{\begin{array}{c}\Gamma_{1,0}(r) = \{\ \overline{{}^{\omega}p}^{\,n}\ \} \neq \emptyset \qquad \forall i \in \{\ 1 \ldots n\ \}.\ \nexists \pi.\ p_i = \pi \qquad \forall i \in \{\ 1 \ldots n\ \}.\ \Delta;\ \Gamma_0 \vdash_{\mathsf{shrd}} p_i : \_,\ \overline{\rho_i}^{\,m_i} \\ \varrho : \mathsf{RGN} \in \Delta \qquad \forall i \in \{\ 1 \ldots n\ \}. \forall j \in \{\ 1 \ldots m_i\ \}.\ \Delta;\ \Gamma_{i,j-1};\ \Theta \vdash^{\mu} \rho_{i,j} :> \varrho \dashv \Gamma_{i,j}\end{array}}{\Delta;\ \Gamma_{1,0};\ \Theta \vdash^{\mu} r :> \varrho \dashv \Gamma_{n,m_n}}$$

Both OL-Trans and OL-ConcreteAbstract follow by applying the induction hypothesis to all instances of the outlives judgment in their premise and then relying on transitivity of subset.

$$\textsc{OL-CombineConcrete}\quad \frac{\Gamma \vdash r_1\ \mathsf{rnrb} \qquad \Gamma \vdash r_2\ \mathsf{rnrb} \qquad \Gamma;\ \Theta \vdash \{\ r_1,\ r_2\ \}\ \mathsf{clrs} \qquad r_1\ \mathsf{occurs\ before}\ r_2\ \mathsf{in}\ \Gamma \qquad \{\ \overline{\ell}\ \} = \Gamma(r_1) \cup \Gamma(r_2)}{\Delta;\ \Gamma;\ \Theta \vdash^{+} r_1 :> r_2 \dashv \Gamma[r_2 \mapsto \{\ \overline{\ell}\ \}]}$$

$$\textsc{OL-CombineConcreteUnrestricted}\quad \frac{\Gamma \vdash r_1\ \mathsf{rnrb} \qquad \Gamma \vdash r_2\ \mathsf{rnrb} \qquad r_1\ \mathsf{occurs\ before}\ r_2\ \mathsf{in}\ \Gamma \qquad \{\ \overline{\ell}\ \} = \Gamma(r_1) \cup \Gamma(r_2)}{\Delta;\ \Gamma;\ \Theta \vdash^{\boxplus} r_1 :> r_2 \dashv \Gamma[r_2 \mapsto \{\ \overline{\ell}\ \}]}$$

For OL-CombineConcrete and OL-CombineConcreteUnrestricted, the conclusion is almost immediate since Γ' is very nearly Γ. However, it differs in the loan set for one particular region r_2. Fortunately, its new loan set in Γ' is the union of its loan set with the loan set for r_1 and thus we immediately have $\Gamma(r_2) \subseteq \Gamma'(r_2)$. □

Lemma E.45 (Region Rewriting Produces Subset-Related Environments). *If* $\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma'$, *then* $\forall r.\ \Gamma(r) \subseteq \Gamma'(r)$.

Proof. This proof proceeds by induction on the region rewriting relation $\Delta;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma'$. We will consider each case. For RR-Refl and RR-Uninit, the output environment Γ' is precisely Γ and thus the result is immediate. For RR-Trans, RR-Array, RR-Slice and RR-Tuple, the result follows from applying the induction hypothesis to every region rewriting derivation in their premise and combining the results by transitivity of subset. This leaves us with one more interesting case, RR-Reference. For this case, apply Lemma E.44 to the outlives derivation in the premise. Then, apply our induction hypothesis to the region rewriting derivation in their premise. Finally, combine the two by transitivity of subset. □

Lemma E.46 (Frame Typing Union Produces Subset-Related Environments). *If* $\mathcal{F} = \mathcal{F}_1 \Cup \mathcal{F}_2$, *then* $\forall r.\ \mathcal{F}_1(r) \subseteq \mathcal{F}(r)$ *and* $\forall r.\ \mathcal{F}_2(r) \subseteq \mathcal{F}(r)$.

Proof. First, note that the definition of $\Cup$ is symmetric and thus we will only prove the first conclusion, the second proceeding immediately the same in all cases.

We proceed by induction over the frame typing. For $\bullet \Cup \bullet$, the case follows immediately. For $\mathcal{F}_1\,,\ x\ :\ \tau \Cup \mathcal{F}_2\,,\ x\ :\ \tau$, the result follows directly from applying the induction hypothesis to $\mathcal{F}_1 \Cup \mathcal{F}_2$. The last case is the interesting one, $\mathcal{F}_1\,,\ r \mapsto \overline{\ell} \Cup \mathcal{F}_2\,,\ r \mapsto \overline{\ell'}$. In this case, we can apply our induction hypothesis to get $\forall r' \in \text{dom}(\mathcal{F}_1), \mathcal{F}_1(r') \subseteq \mathcal{F}_2(r')$. Now we just need that $\{\ \overline{\ell}\ \} \subseteq \mathcal{F}(r)$. But this holds immediately since $\mathcal{F}(r) = \{\ \overline{\ell}\ \} \cup \{\ \overline{\ell'}\ \}$, so we're done. □

Lemma E.47 (Stack Typing Union Produces Subset-Related Environments). *If* $\Gamma = \Gamma_1 \Cup \Gamma_2$, *then* $\forall r.\ \Gamma_1(r) \subseteq \Gamma(r)$ *and* $\forall r.\ \Gamma_2(r) \subseteq \Gamma(r)$.

Proof. First, note that the definition of $\Cup$ is symmetric and thus we will only prove the first conclusion, the second proceeding immediately the same in all cases.

We proceed by induction over the Stack Typing.

For $\bullet \Cup \bullet$, the result is trivial and thus immediate. For $\Gamma_1 \natural \mathcal{F} \Cup \Gamma_2 \natural \mathcal{F}$, we apply the induction hypothesis and Lemma E.46. □

Lemma E.48 (Subset-Related Frames are also Frame Typing Union Related). *If* $\forall r \in \mathcal{F}.\ \mathcal{F}'(r) \subseteq \mathcal{F}(r)$ *and* $dom(\mathcal{F}) = dom(\mathcal{F}')$, *then* $\exists \mathcal{F}_o$ *such that* $\mathcal{F} = \mathcal{F}' \Cup \mathcal{F}_o$.

Proof. We proceed by induction over the frame typing. For the $\bullet$ case, the proof follows immediately.

For $\mathcal{F} = \mathcal{F}_i, x : \tau$ and $\mathcal{F}' = \mathcal{F}_i', x : \tau$, we just apply the induction hypothesis on $\mathcal{F}_i$ and $\mathcal{F}_i'$, and add $x : \tau$ to $\mathcal{F}_o$.

For $\mathcal{F} = \mathcal{F}_i, r \mapsto \{\ \overline{\ell}\ \}$ and $\mathcal{F}' = \mathcal{F}_i', r \mapsto \{\ \overline{\ell}'\ \}$, we apply the induction hypothesis, and add on $r \mapsto \{\ \overline{\ell}\ \} \setminus \{\ \overline{\ell}'\ \}$, which is well defined because from our premise we have $\{\ \overline{\ell}'\ \} \subseteq \{\ \overline{\ell}\ \}$. □

Lemma E.49 (Subset-Related Environments are also Stack Typing Union Related). *If* $\forall r \in \Gamma.\ \Gamma'(r) \subseteq \Gamma(r)$ *and* $dom(\Gamma) = dom(\Gamma')$, *then* $\exists \Gamma_o$ *such that* $\Gamma = \Gamma' \Cup \Gamma_o$.

Proof. Proceed by induction on the stack typing. In the $\bullet$ case, the proof is immediate. In the $\Gamma = \Gamma_i \natural \mathcal{F}$ case, we apply Lemma E.48 and the induction hypothesis. □

E.11 Preservation in More Precise Environments Lemmas

Lemma E.50 (Type Computation Preserved in More Precise Environments). *If* $dom(\Gamma) = dom(\Gamma')$, *and* $\forall x.\ \Gamma(x) = \Gamma'(x)$ *and* $\forall r.\ \Gamma'(r) \subseteq \Gamma(r)$ *and* $\Delta;\ \Gamma \vdash_\omega p : \tau^{XI}$ *then* $\Delta;\ \Gamma' \vdash_\omega p : \tau^{XI}$.

Proof. Proceed by induction over the type computation judgement with Γ. The only non-immediate case is TC-Deref.

TC-Deref
$$\frac{\Delta;\ \Gamma \vdash_\omega p : \&\rho\ \omega'\ \tau^{\text{XI}},\ \{\ \overline{\rho_p}\ \} \qquad \omega \lesssim \omega'}{\Delta;\ \Gamma \vdash_\omega *p : \tau^{\text{XI}},\ \{\ \overline{\rho_p}\,,\ \rho\ \}}$$

This follows from the induction hypothesis and Lemma E.44.

□

Lemma E.51 (Type Well-Formedness Preserved in More Precise Environments). *If* $dom(\Gamma) = dom(\Gamma')$, *and* $\forall r.\ \Gamma'(r) \subseteq \Gamma(r)$ *and* $\Sigma;\ \Delta;\ \Gamma \vdash \tau^{SI}$ *then* $\Sigma;\ \Delta;\ \Gamma' \vdash \tau^{SI}$.

Proof. Proceed by induction over the type well formedness under Γ. The only case that isn't immediate or doesn't proceed directly from the induction hypothesis is WF-Ref.

WF-Ref
$$\frac{\Delta;\ \Gamma \vdash \rho \qquad \Sigma;\ \Delta;\ \Gamma \vdash \tau^{\text{XI}}}{\Sigma;\ \Delta;\ \Gamma \vdash \&\rho\ \omega\ \tau^{\text{XI}}}$$

We can apply the induction hypothesis to get the premise that τ^{XI} is well formed. For our other premise, we need to show that our region is still well-formed. We can look at this by cases. If the region ρ is local, we can apply WF-LocalRegion since we know $\text{dom}(\Gamma) = \text{dom}(\Gamma')$. If the region ρ is abstract, we can apply WF-AbstractRegion since we know that Δ is unchanged. □

Lemma E.52 (Stack Typing Validity Preserved in More Precise Environments). *If* $dom(\Gamma) = dom(\Gamma')$, *and* $\forall r.\ \Gamma'(r) \subseteq \Gamma(r)$ *and* $\Sigma;\ \Delta \vdash \Gamma$ *then* $\Sigma;\ \Delta \vdash \Gamma'$

Proof. Proceed by induction over the stack typing validity judgement for Γ. The empty case is trivial, so the interesting case is WF-StackTyping.

WF-StackTyping
$$\frac{\begin{array}{c}\Sigma;\ \Delta \vdash \Gamma \qquad \texttt{places}(\mathcal{F}) \subseteq \text{dom}(\Gamma \natural \mathcal{F}) \\ \text{dom}(\mathcal{F}) \# \text{dom}(\Gamma) \qquad \forall x : \tau \in \mathcal{F}.\ \Sigma;\ \Delta;\ \Gamma \natural \mathcal{F} \vdash \tau \\ \forall \tau \in \text{cod}(\mathcal{F}).\ \forall r \in \text{free-regions}(\tau).\ \forall \tau' \in \text{dom}(\Gamma).\ r \text{ does not occur outside of a closure in } \tau' \\ \forall r \mapsto \{\ \overline{\ell}\ \} \in \mathcal{F}.\ \forall^{\omega} p \in \{\ \overline{\ell}\ \}.\ \exists \tau^{\text{XI}}.\ \Delta;\ \Gamma \natural \mathcal{F} \vdash_{\omega} p : \tau^{\text{XI}}\end{array}}{\Sigma;\ \Delta \vdash \Gamma \natural \mathcal{F}}$$

We get $\Sigma;\ \Delta \vdash \Gamma'$ by induction. We get the type well formedness from Lemma E.51. We get the type computation from Lemma E.50, and that's all we needed to show.

□

Lemma E.53 (Ownership Safety Preserved in More Precise Environments). *If* $dom(\Gamma) = dom(\Gamma')$, *and* $\forall r.\ \Gamma'(r) \subseteq \Gamma(r)$ *and* $\Delta;\ \Gamma;\ \Theta \vdash_{\omega} p \Rightarrow \{\ \overline{\ell}\ \}$ *then* $\Delta;\ \Gamma';\ \Theta \vdash_{\omega} p \Rightarrow \{\ \overline{\ell'}\ \}$ *and* $\{\ \overline{\ell'}\ \} \subseteq \{\ \overline{\ell}\ \}$.

Proof. The proof proceeds by induction on the ownership safety judgment $\Delta;\ \Gamma;\ \Theta \vdash_{\omega} p \Rightarrow \{\overline{\ell}\}$. This gives us three cases: O-SafePlace, O-Deref, and O-DerefAbs.

O-SafePlace
$$\frac{\begin{array}{c}\forall r' \mapsto \{\ \overline{\ell}\ \} \in \text{regions}(\Gamma,\ \Theta).\ (\forall^{\omega'} p^{\square}[\pi'] \in \{\ \overline{\ell}\ \}.(\omega = \texttt{uniq} \lor \omega' = \texttt{uniq}) \implies \pi' \# \pi) \\ \lor\ (\exists \pi' : \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \land \nexists \&r'\ \omega'\ \tau' \in \Theta\ \land (\forall \pi' : \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\ \overline{\pi_e}\ \}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi_e}} \pi \Rightarrow \{\ {}^{\omega}\pi\ \}}$$

We need to show $\Delta;\ \Gamma';\ \Theta \vdash_{\omega} p \Rightarrow \{\ {}^{\omega}\pi\ \}$ and that $\{\ {}^{\omega}\pi\ \} \subseteq \{\ {}^{\omega}\pi\ \}$. The latter is immediate from the definition of subset which leaves us with the former. For the former, we'll correspondingly wish to apply O-SafePlace but using Γ' as our context. This means we need to show that $\forall r' \mapsto \{\ \overline{\ell}\ \} \in \Gamma'.\ (\forall^{\omega'} p \in \{\ \overline{\ell}\ \}.(\omega = \texttt{uniq} \lor \omega' = \texttt{uniq}) \implies \pi' \# \pi) \lor (\exists \pi' : \&r'\ \omega'\ \tau' \in$

$\Gamma \ \wedge (\forall \pi' : \&r' \ \omega' \ \tau' \in \Gamma.\ \pi' \in \{ \overline{\pi_e} \}))$. Fortunately, from $\mathrm{dom}(\Gamma) = \mathrm{dom}(\Gamma')$, we know that for every $r' \in \mathrm{dom}(\Gamma)$, $r' \in \mathrm{dom}(\Gamma')$, and further that $\Gamma'(r') \subseteq \Gamma(r')$. Thus, for each r', we know there are only potentially fewer loans to show if the obligation was met using the clause of $\forall {}^{\omega'}p \in \{ \overline{\ell} \}.(\omega = \mathsf{uniq} \vee \omega' = \mathsf{uniq}) \implies \pi' \# \pi$. If the obligation was met using the other clause, note that Γ and Γ' can only differ in the loan sets they associate with any given region and so the exact fact must still be true for Γ'.

O-Deref

$$\dfrac{\begin{array}{c}\Gamma(\pi) = \&r\ \omega_\pi\ \tau_\pi \qquad \Gamma(r) = \{ \overline{{}^{\omega'}p}^{\,n} \} \qquad \mathrm{excl} = \{ \pi_j \text{ where } j \in \{ 1, \ldots n \} \mid p_j = p^\square_j[*\pi_j] \} \qquad \omega \lesssim \omega_\pi \\ \forall i \in \{ 1 \ldots n \}.\ \Delta;\ \Gamma;\ \Theta \vdash_\omega^{\overline{\pi_e}, \mathrm{excl}, \pi} p^\square[p_i] \Rightarrow \{ \overline{{}^{\omega}p'_i} \} \\ \forall r' \mapsto \{ \overline{\ell} \} \in \mathrm{regions}(\Gamma, \Theta).\ (\forall {}^{\omega'}p'' \in \{ \overline{\ell} \}.(\omega = \mathsf{uniq} \vee \omega' = \mathsf{uniq}) \implies p'' \# p^\square[*\pi]) \\ \vee\ (\exists \pi' : \&r'\ \omega'\ \tau' \in \mathrm{explode}(\Gamma)\ \wedge \nexists \&r'\ \omega'\ \tau' \in \Theta\ \wedge (\forall \pi' : \&r'\ \omega'\ \tau' \in \mathrm{explode}(\Gamma).\ \pi' \in \{ \overline{\pi_e}, \mathrm{excl}, \pi \}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash_\omega^{\overline{\pi_e}} p^\square[*\pi] \Rightarrow \{ \overline{{}^{\omega}p'_1}, \ \ldots\ \overline{{}^{\omega}p'_n}, {}^{\omega}p^\square[*\pi] \}}$$

This case proceeds much like the O-SafePlace case in terms of meeting the direct ownership safety criterion (the last premise of O-Deref) for the new derivation using O-Deref with Γ'. It differs only in that we need also apply our induction hypothesis to each of the n derivations of ownership safety used in $\forall i \in \{ 1 \ \ldots \ n \}.\ \Delta;\ \Gamma;\ \Theta \vdash_\omega^{\overline{\pi_e}, \overline{\pi_i}, \pi} p^\square[p_i] \Rightarrow \{ \overline{{}^{\omega}p'_i} \}$. This gives us $\forall i \in \{ 1 \ \ldots \ n \}.\ \Delta;\ \Gamma';\ \Theta \vdash_\omega^{\overline{\pi_e}, \overline{\pi_i}, \pi} p^\square[p_i] \Rightarrow \{ \overline{{}^{\omega}p''_i} \}$ and $\forall i \in \{ 1 \ \ldots \ n \}.\ \{ \overline{{}^{\omega}p''_i} \} \subseteq \{ \overline{{}^{\omega}p'_i} \}$. The former combined with the same reasoning from the O-SafePlace case gives us $\Delta;\ \Gamma';\ \Theta \vdash_\omega p^\square[*\pi] \Rightarrow \{ \overline{{}^{\omega}p''_1}, \ \ldots\ \overline{{}^{\omega}p''_n}, {}^{\omega}p^\square[*\pi] \}$ and the latter allows us to conclude $\{ \overline{{}^{\omega}p''_1}, \ \ldots\ \overline{{}^{\omega}p''_n}, {}^{\omega}p^\square[*\pi] \} \subseteq \{ \overline{{}^{\omega}p'_1}, \ \ldots\ \overline{{}^{\omega}p'_n}, {}^{\omega}p^\square[*\pi] \}$ since we know that each individual collection of loans has the subset relation from above and the whole set is simply their union.

O-DerefAbs

$$\dfrac{\begin{array}{c}\Gamma(\pi) = \&\varrho\ \omega_\pi\ \tau_\pi \qquad \Delta;\ \Gamma \vdash_\omega p^\square[*\pi] : \tau \qquad \omega \lesssim \omega_\pi \\ \forall r' \mapsto \{ \overline{\ell} \} \in \mathrm{regions}(\Gamma, \Theta).\ (\forall {}^{\omega'}p \in \{ \overline{\ell} \}.(\omega = \mathsf{uniq} \vee \omega' = \mathsf{uniq}) \implies p \# p^\square[*\pi]) \\ \vee\ (\exists \pi' : \&r'\ \omega'\ \tau' \in \mathrm{explode}(\Gamma)\ \wedge \nexists \&r'\ \omega'\ \tau' \in \Theta\ \wedge (\forall \pi' : \&r'\ \omega'\ \tau' \in \mathrm{explode}(\Gamma).\ \pi' \in \{ \overline{\pi_e}, \pi \}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash_\omega^{\overline{\pi_e}} p^\square[*\pi] \Rightarrow \{ {}^{\omega}p^\square[*\pi] \}}$$

This case proceeds identically to the case for O-SafePlace. We need to show $\Delta;\ \Gamma';\ \Theta \vdash_\omega p \Rightarrow \{ {}^{\omega}p^\square[*\pi] \}$ and that $\{ \{ {}^{\omega}p^\square[*\pi] \} \} \subseteq \{ \{ {}^{\omega}p^\square[*\pi] \} \}$. The latter is immediate from the definition of subset which leaves us with the former. For the former, we'll correspondingly wish to apply O-DerefAbs but using Γ' as our context. This means we need to show that $\forall r' \mapsto \{ \overline{\ell} \} \in \Gamma'.\ (\forall {}^{\omega'}p \in \{ \overline{\ell} \}.(\omega = \mathsf{uniq} \vee \omega' = \mathsf{uniq}) \implies \pi' \# p^\square[*\pi]) \vee\ (\exists \pi' : \&r'\ \omega'\ \tau' \in \Gamma\ \wedge (\forall \pi' : \&r'\ \omega'\ \tau' \in \Gamma.\ \pi' \in \{ \overline{\pi_e} \}))$. Fortunately, from $\mathrm{dom}(\Gamma) = \mathrm{dom}(\Gamma')$, we know that for every $r' \in \mathrm{dom}(\Gamma)$, $r' \in \mathrm{dom}(\Gamma')$, and further that $\Gamma'(r') \subseteq \Gamma(r')$. Thus, for each r', we know there are only potentially fewer loans to show if the obligation was met using the clause of $\forall {}^{\omega'}p \in \{ \overline{\ell} \}.(\omega = \mathsf{uniq} \vee \omega' = \mathsf{uniq}) \implies \pi' \# \pi$. If the obligation was met using the other clause, note that Γ and Γ' can only differ in the loan sets they associate with any given region and so the exact fact must still be true for Γ'. □

Lemma E.54 (Expressions Remain Well-Typed in More Precise Environments). *If* $\mathrm{dom}(\Gamma) = \mathrm{dom}(\Gamma')$, *and* $\forall r.\ \Gamma'(r) \subseteq \Gamma(r)$ *and* $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau \Rightarrow \Gamma_f$ *then* $\Sigma;\ \bullet;\ \Gamma';\ \Theta \vdash \boxed{e} : \tau \Rightarrow \Gamma'_f$ *and* $\mathrm{dom}(\Gamma_f) = \mathrm{dom}(\Gamma'_f)$, *and* $\forall r.\ \Gamma'_f(r) \subseteq \Gamma_f(r)$

Proof. We proceed by induction over the expression typing.

T-Move, T-Copy, and T-Borrow all follow from the Lemma E.53 and the Lemma E.50.

T-BorrowIndex, T-BorrowSlice, and T-IndexCopy all follow from the induction hypothesis, Lemma E.53, and the Lemma E.50.

T-Seq follows from the observation that garbage collecting loans will preserve subsets and clear in exactly both or neither, and from the induction hypothesis.

T-Branch and T-Match follow from applying the induction hypothesis, Lemma E.45, Lemma E.47.

T-Let follows from the same observation about garbage collection in the T-Seq case, the induction hypothesis, and Lemma E.45.

T-Assign follows from the induction hypothesis, Lemma E.53, Lemma E.47.

T-AssignDeref both follow from the induction hypothesis, follows from the induction hypothesis, Lemma E.53, Lemma E.47, and Lemma E.50.

T-AppFunction follows from the induction hypothesis, and a few pieces about the well-formedness of the instantiations happening in T-AppFunction (namely, frame expressions, regions, and types). For the frame expression validity, we consider each case and note that WF-EnvVar depends only on Δ which is unchanged and that WF-Env appeals to stack typing validity and so it suffices to show that that still holds in our more precise environment which we do by appealing to Lemma E.52. For the region validity, there are again two cases to consider WF-LocalProv which applies if the domain of Γ is the same as the domain of Γ' which we have directly from our premise and WF-AbstractProv which depends only on Δ which is unchanged. For the type validity, we appeal to Lemma E.51.

T-AppClosure follows from the induction hypothesis and Lemma E.45.

T-LetRegion, T-While, T-ForArray, T-ForSlice, T-Closure, T-Tuple, T-Array, T-Slice, T-Drop, T-Left, T-Right follow immediately from the induction hypothesis.

T-Function, T-Abort, T-Unit, T-U32, T-True, and T-False are immediate.

□

E.12 Preservation under Safe Loan Updates Lemmas

Lemma E.55 (Ownership Safety Produces Non Conflicting Loans). *If*

(1) $p^{\square}_{r_p}[p_{r_p}] \in \{\, \overline{\ell} \,\}$, *where* $r_p \mapsto \{\, \overline{\ell} \,\} \in \mathit{regions}(\Gamma,\ \Theta)$
(2) $\Gamma(r_b) = \emptyset$
(3) $\Gamma[r_b \mapsto \{\, \overline{\ell_b} \,\}] \vdash r_p\ \mathsf{rnrb}$
(4) $\bullet;\ \Gamma;\ \Theta \vdash^{\overline{\pi_b}}_{\omega} p_b \Rightarrow \{\, \overline{\ell_b} \,\}$

then $\forall\, {}^{\omega}p' \in \{\, \overline{\ell_b} \,\}.\ p_{r_p} \# p'$.

Proof. Proceed by induction on the ownership safety judgement for p in the premise.

O-SafePlace

$$\frac{\begin{array}{c}\forall r' \mapsto \{\, \overline{\ell} \,\} \in \text{regions}(\Gamma,\ \Theta).\ (\forall\, {}^{\omega'}p^{\square}[\pi'] \in \{\, \overline{\ell} \,\}.(\omega = \mathsf{uniq} \lor \omega' = \mathsf{uniq}) \implies \pi' \# \pi) \\ \lor\ (\exists \pi' : \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \land \nexists \&r'\ \omega'\ \tau' \in \Theta\ \land (\forall \pi' : \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\, \overline{\pi_e} \,\}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash^{\overline{\pi_e}}_{\omega} \pi \Rightarrow \{\, {}^{\omega}\pi \,\}}$$

We need to show that $p_{r_p} \# \pi$. Since $r_p \in \text{dom}(\text{regions}(\Gamma,\ \Theta))$, we know from the hypothesis of O-SafePlace that either r_p is excluded, or all loans in it are disjoint from π.

r_p cannot have been exluded because $\Gamma \vdash r_p\ \mathsf{rnrb}$.

This just leaves the case where all loans in $\Gamma(r_p)$ are disjoint from π. Let π_{r_p} be the inner place of p_{r_p}. More formally, $p_{r_p} = p^{\square}[\pi_{r_p}]$. By this disjointness, we know $p^{\square}_{r_p}[p^{\square}[\pi_{r_p}]] \# \pi$, which directly implies $p^{\square}[\pi_{r_p}] \# \pi$, which is what we wanted to show.

O-Deref

$$\frac{\begin{array}{c}\Gamma(\pi) = \&r\ \omega_\pi\ \tau_\pi \qquad \Gamma(r) = \{\ \overline{{}^{\omega'}p}^{\,n}\ \} \qquad \text{excl} = \{\ \pi_j \text{ where } j \in \{\ 1, \dots\ n\ \} \mid p_j = p_j^{\square}[*\pi_j]\ \} \qquad \omega \lesssim \omega_\pi \\ \forall i \in \{\ 1\ \dots\ n\ \}.\ \Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi_e},\,\text{excl},\,\pi} p^{\square}[p_i] \Rightarrow \{\ \overline{{}^{\omega}p_i'}\ \} \\ \forall r' \mapsto \{\ \overline{\ell}\ \} \in \text{regions}(\Gamma,\ \Theta).\ (\forall\, {}^{\omega'}p''\ \in \{\ \overline{\ell}\ \}.(\omega = \texttt{uniq} \vee \omega' = \texttt{uniq}) \implies p'' \# p^{\square}[*\pi]) \\ \vee\ (\exists \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \wedge \nexists \&r'\ \omega'\ \tau' \in \Theta\ \wedge (\forall \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\ \overline{\pi_e},\ \text{excl},\ \pi\ \}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi_e}} p^{\square}[*\pi] \Rightarrow \{\ \overline{{}^{\omega}p_1'},\ \dots\ \overline{{}^{\omega}p_n'},\ {}^{\omega}p^{\square}[*\pi]\ \}}$$

We need to show that $p^{\square}[*\pi] \# p_{r_p}$, and that $\forall i,\ {}^{\omega}p\ \in \{\ \overline{{}^{\omega}p_i'}\ \}.\ p_{r_p} \# p$.
The latter we get from applying the induction hypothesis.
The former follows from the same reasoning as in the previous case.

O-DerefAbs

$$\frac{\begin{array}{c}\Gamma(\pi) = \&\varrho\ \omega_\pi\ \tau_\pi \qquad \Delta;\ \Gamma \vdash_\omega p^{\square}[*\pi] : \tau \qquad \omega \lesssim \omega_\pi \\ \forall r' \mapsto \{\ \overline{\ell}\ \} \in \text{regions}(\Gamma,\ \Theta).\ (\forall\, {}^{\omega'}p\ \in \{\ \overline{\ell}\ \}.(\omega = \texttt{uniq} \vee \omega' = \texttt{uniq}) \implies p \# p^{\square}[*\pi]) \\ \vee\ (\exists \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \wedge \nexists \&r'\ \omega'\ \tau' \in \Theta\ \wedge (\forall \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\ \overline{\pi_e},\ \pi\ \}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi_e}} p^{\square}[*\pi] \Rightarrow \{\ {}^{\omega}p^{\square}[*\pi]\ \}}$$

Since $\Delta = \bullet$, there are no valid reference types that have an abstract region, meaning the first hypothesis is a contradiction. □

Lemma E.56 (Ownership Safety is Preserved under Safe Loan Updates). *If*

(1) $\bullet;\ \Gamma;\ \Theta \vdash_{\omega_b} p_b \Rightarrow \{\ \overline{\ell_b}\ \}$
(2) $\bullet;\ \Gamma \natural \mathcal{F};\ \Theta \vdash_{\omega}^{\overline{\pi_1}} p \Rightarrow \{\ \overline{\ell}\ \}$
(3) *and* $\Gamma(r_b) = \{\}$
(4) $\Gamma[r_b \mapsto \{\ \overline{\ell_b}\ \}] \vdash r_p$ $\texttt{rnrb}$
(5) $\pi_1 = \pi_2$ *or* $\pi_1 = \pi_2 \cup \{\ \pi \mid p^{\square}[*\pi] \in \{\ \overline{\ell_b}\ \}\ \}$
(6) *either* $\texttt{root-of}(p) \in dom(\mathcal{F})$ *or* $\exists r_p \in dom(\Gamma \natural \mathcal{F}), p^{\square}, p'$.
(7) $p = p^{\square}[p']$
(8) $p' \in \Gamma \natural \mathcal{F}(r_p)$
(9) $\forall r \in dom(\mathcal{F}),\ {}^{\omega}p\ \in \mathcal{F}(r)$. *either* $\texttt{root-of}(p) \in dom(\mathcal{F})$, *or* $\exists r' \mapsto \{\ \overline{\ell}\ \} \in regions(\Gamma,\ \Theta)$. ${}^{\omega}p\ \in \{\ \overline{\ell}\ \}$.
(In english, loans in the closure's frame come from the current frame or a closure's captured frame in Γ *or* Θ*)*

then $\bullet;\ \Gamma[r_b \mapsto \overline{\ell_b}] \natural \mathcal{F};\ \Theta \vdash_{\omega}^{\overline{\pi_2}} p \Rightarrow \{\ \overline{\ell'}\ \}$.

Proof. Proceed by induction on the ownership safety judgement for p in the premise.

O-SafePlace

$$\frac{\begin{array}{c}\forall r' \mapsto \{\ \overline{\ell}\ \} \in \text{regions}(\Gamma,\ \Theta).\ (\forall\, {}^{\omega'}p^{\square}[\pi']\ \in \{\ \overline{\ell}\ \}.(\omega = \texttt{uniq} \vee \omega' = \texttt{uniq}) \implies \pi' \# \pi) \\ \vee\ (\exists \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \wedge \nexists \&r'\ \omega'\ \tau' \in \Theta\ \wedge (\forall \pi'\ :\ \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\ \overline{\pi_e}\ \}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash_{\omega}^{\overline{\pi_e}} \pi \Rightarrow \{\ {}^{\omega}\pi\ \}}$$

Let r' be an arbitrary region. If the disjunction was proven using the right part, which talks about the exclusion list, then we can prove it again the same way, since none of the types are changed and the exclusion list $\overline{\pi_2}$ includes all of $\overline{\pi_1}$ in either case.

If the disjunction was proven using the left part, the only interesting case is when $r' = r_b$. If `root-of`$(\pi) \in \mathrm{dom}(\mathcal{F})$, then we're done, because all loans in $\{ \overline{\ell_b} \}$ are disjoint just by the well formedness of the environment Γ.

Otherwise, `root-of`$(\pi) \in \mathrm{dom}(\Gamma)$ and $\exists r_p$ such that $\pi \in \Gamma \natural \mathcal{F}(r_p)$, and we want to show that $\forall {}^{\omega'} p^{\square}[\pi'] \in \{ \overline{\ell_b} \}, \pi \# \pi'$.

If $r_p \in \mathrm{dom}(\mathcal{F})$, then we're done because each loan in r_p either comes from $\mathrm{dom}(\mathcal{F})$, in which case disjointness is immediate, or it comes from a loan mapping in Γ or Θ, in which case we can apply Lemma E.55 to finish the proof.

Otherwise, $r_p \in \mathrm{dom}(\Gamma)$. By the well formedness of $\Gamma[r_b \mapsto \{ \overline{\ell_b} \}] \natural \mathcal{F}, \Gamma[r_b \mapsto \{ \overline{\ell_b} \}] \vdash r'$ rnrb. Given all of this we can apply Lemma E.55 to finish the proof.

O-Deref

$$\frac{\begin{array}{c}\Gamma(\pi) = \&r\ \omega_\pi\ \tau_\pi \qquad \Gamma(r) = \{ \overline{{}^{\omega'}p}^{\,n} \} \qquad \mathrm{excl} = \{ \pi_j \text{ where } j \in \{ 1, \ldots n \} \mid p_j = p_j^{\square}[*\pi_j] \} \qquad \omega \lesssim \omega_\pi \\ \forall i \in \{ 1 \ldots n \}.\ \Delta; \Gamma; \Theta \vdash_{\omega}^{\overline{\pi_e}, \mathrm{excl}, \pi} p^{\square}[p_i] \Rightarrow \{ \overline{{}^{\omega}p_i'} \} \\ \forall r' \mapsto \{ \overline{\ell} \} \in \mathrm{regions}(\Gamma, \Theta).\ (\forall {}^{\omega'}p'' \in \{ \overline{\ell} \}.(\omega = \mathtt{uniq} \lor \omega' = \mathtt{uniq}) \implies p'' \# p^{\square}[*\pi]) \\ \lor (\exists \pi' : \&r'\ \omega'\ \tau' \in \mathrm{explode}(\Gamma) \land \nexists \&r'\ \omega'\ \tau' \in \Theta \land (\forall \pi' : \&r'\ \omega'\ \tau' \in \mathrm{explode}(\Gamma).\ \pi' \in \{ \overline{\pi_e}, \mathrm{excl}, \pi \}))\end{array}}{\Delta; \Gamma; \Theta \vdash_{\omega}^{\overline{\pi_e}} p^{\square}[*\pi] \Rightarrow \{ \overline{{}^{\omega}p_1'}, \ \ldots \ \overline{{}^{\omega}p_n'}, {}^{\omega}p^{\square}[*\pi] \}}$$

Firstly, we would like to apply our induction hypothesis. In order to do so, we need to show that the new `excl` is either the same, or only has places from $\{ \overline{\ell_b} \}$ added. If $r \neq r_b$, then $\Gamma \natural \mathcal{F}(r) = \Gamma[r_b \mapsto \{ \overline{\ell_b} \}] \natural \mathcal{F}$, so the exclusion list is the same. If $r = r_b$, then `excl` was empty, and now includes the places $\{ \pi \mid p^{\square}[*\pi] \in \{ \overline{\ell_b} \} \}$. We also need to show that either that either the place expression is in the domain of $\mathcal{F}$ or that it has a sub place expression in a loan set, but this is immediate from our hypotheses. So in either case we satisfy the necessary hypothesis and can apply the induction hypothesis.

Whats left to show is the disjointness or exclusion condition, which follows identically to the reasoning in the previous case.

O-DerefAbs

$$\frac{\begin{array}{c}\Gamma(\pi) = \&\varrho\ \omega_\pi\ \tau_\pi \qquad \Delta; \Gamma \vdash_{\omega} p^{\square}[*\pi] : \tau \qquad \omega \lesssim \omega_\pi \\ \forall r' \mapsto \{ \overline{\ell} \} \in \mathrm{regions}(\Gamma, \Theta).\ (\forall {}^{\omega'}p \in \{ \overline{\ell} \}.(\omega = \mathtt{uniq} \lor \omega' = \mathtt{uniq}) \implies p \# p^{\square}[*\pi]) \\ \lor (\exists \pi' : \&r'\ \omega'\ \tau' \in \mathrm{explode}(\Gamma) \land \nexists \&r'\ \omega'\ \tau' \in \Theta \land (\forall \pi' : \&r'\ \omega'\ \tau' \in \mathrm{explode}(\Gamma).\ \pi' \in \{ \overline{\pi_e}, \pi \}))\end{array}}{\Delta; \Gamma; \Theta \vdash_{\omega}^{\overline{\pi_e}} p^{\square}[*\pi] \Rightarrow \{ {}^{\omega}p^{\square}[*\pi] \}}$$

Since $\Delta = \bullet$, there are no valid reference types that have an abstract region, meaning the first hypothesis is a contradiction. □

Lemma E.57 (Ownership Safety is Preserved after Environment Union). *If* $\bullet; \Gamma_1; \Theta \vdash_{\omega}^{\overline{\pi_1}} p \Rightarrow \{ \overline{\ell} \}$ *and* $\bullet; \Gamma_2; \Theta \vdash_{\omega}^{\overline{\pi_2}} p \Rightarrow \{ \overline{\ell}' \}$ *then* $\bullet; \Gamma_1 \Cup \Gamma_2; \Theta \vdash_{\omega}^{\overline{\pi_1}, \overline{\pi_2}} p \Rightarrow \{ \overline{\ell}'' \}$ *and* $\pi_1 = \pi_2$ *or* $\pi_1 = \pi_2 \cup \{ \pi \mid p^{\square}[*\pi] \in \{ \overline{\ell_b} \} \}$.

Proof. Proceed by induction on the ownership safety judgements in the premise. Note that they both have the same sequence of proof rule applications, because the judgement is inductive over p.

O-SafePlace

$$\frac{\begin{array}{c}\forall r' \mapsto \{ \overline{\ell} \} \in \mathrm{regions}(\Gamma, \Theta).\ (\forall {}^{\omega'}p^{\square}[\pi'] \in \{ \overline{\ell} \}.(\omega = \mathtt{uniq} \lor \omega' = \mathtt{uniq}) \implies \pi' \# \pi) \\ \lor (\exists \pi' : \&r'\ \omega'\ \tau' \in \mathrm{explode}(\Gamma) \land \nexists \&r'\ \omega'\ \tau' \in \Theta \land (\forall \pi' : \&r'\ \omega'\ \tau' \in \mathrm{explode}(\Gamma).\ \pi' \in \{ \overline{\pi_e} \}))\end{array}}{\Delta; \Gamma; \Theta \vdash_{\omega}^{\overline{\pi_e}} \pi \Rightarrow \{ {}^{\omega}\pi \}}$$

Let r be an arbitrary region.

If either the derivation with Γ_1 or Γ_2 used the second part of the disjunction, we can proceed by the second part of the disjunction.

Otherwise, both derivations used the first part of the disjunction. Since $\Gamma_1 \Cup \Gamma_2(r) = \Gamma_1(r) \cup \Gamma_2(r)$, we can just combine these two facts and proceed by the first part of the disjunction.

$$\textsc{O-Deref}\quad \frac{\begin{array}{c}\Gamma(\pi) = \&r\ \omega_\pi\ \tau_\pi \qquad \Gamma(r) = \{\ \overline{{}^{\omega'}p}^{\,n}\ \} \qquad \text{excl} = \{\ \pi_j \text{ where } j \in \{\ 1, \dots\ n\ \} \mid p_j = p_j^\square[*\pi_j]\ \} \qquad \omega \lesssim \omega_\pi \\ \forall i \in \{\ 1\ \dots\ n\ \}.\ \Delta;\ \Gamma;\ \Theta \vdash_\omega^{\overline{\pi_e},\, \text{excl},\, \pi} p^\square[p_i] \Rightarrow \{\ \overline{{}^{\omega}p_i'}\ \} \\ \forall r' \mapsto \{\ \overline{\ell}\ \} \in \text{regions}(\Gamma,\ \Theta).\ (\forall {}^{\omega'}p'' \in \{\ \overline{\ell}\ \}.(\omega = \texttt{uniq} \vee \omega' = \texttt{uniq}) \implies p'' \# p^\square[*\pi]) \\ \vee\ (\exists \pi' : \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \wedge \nexists \&r'\ \omega'\ \tau' \in \Theta\ \wedge (\forall \pi' : \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\ \overline{\pi_e},\ \text{excl},\ \pi\ \}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash_\omega^{\overline{\pi_e}} p^\square[*\pi] \Rightarrow \{\ \overline{{}^{\omega}p_1'}\ ,\ \dots\ \overline{{}^{\omega}p_n'}\ ,\ {}^{\omega}p^\square[*\pi]\ \}}$$

In order to apply the induction hypothesis, we need show that our new `excl` is the union of the two from the derivations. This is immediate because $\Gamma_1 \Cup \Gamma_2(r) = \Gamma_1(r) \cup \Gamma_2(r)$.

To finish the case, follow the same reasoning from the previous case for the disjunction.

$$\textsc{O-DerefAbs}\quad \frac{\begin{array}{c}\Gamma(\pi) = \&\varrho\ \omega_\pi\ \tau_\pi \qquad \Delta;\ \Gamma \vdash_\omega p^\square[*\pi] : \tau \qquad \omega \lesssim \omega_\pi \\ \forall r' \mapsto \{\ \overline{\ell}\ \} \in \text{regions}(\Gamma,\ \Theta).\ (\forall {}^{\omega'}p \in \{\ \overline{\ell}\ \}.(\omega = \texttt{uniq} \vee \omega' = \texttt{uniq}) \implies p \# p^\square[*\pi]) \\ \vee\ (\exists \pi' : \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma)\ \wedge \nexists \&r'\ \omega'\ \tau' \in \Theta\ \wedge (\forall \pi' : \&r'\ \omega'\ \tau' \in \text{explode}(\Gamma).\ \pi' \in \{\ \overline{\pi_e},\ \pi\ \}))\end{array}}{\Delta;\ \Gamma;\ \Theta \vdash_\omega^{\overline{\pi_e}} p^\square[*\pi] \Rightarrow \{\ {}^{\omega}p^\square[*\pi]\ \}}$$

Since $\Delta = \bullet$, there are no valid reference types that have an abstract region, meaning the first hypothesis is a contradiction.

□

Lemma E.58 (Ownership Safety is Preserved after Type Checking a Closure Body). *If*

(1) $\bullet;\ \Gamma;\ \Theta \vdash_\omega p_b \Rightarrow \{\ \overline{\ell_b}\ \}$
(2) $\Gamma(r) = \{\}$
(3) $\forall x \in \mathit{free\text{-}vars}(e).\ x \in \mathit{dom}(\mathcal{F})$
(4) $\forall r \in \mathit{free\text{-}regions}(e).\ r \in \mathit{dom}(\mathcal{F})$
(5) $\forall r \in \mathit{dom}(\mathcal{F}),\ {}^{\omega}p \in \mathcal{F}(r).\texttt{root-of}(p) \in \mathit{dom}(\mathcal{F}) \vee \exists r' \mapsto \{\overline{\ell}\} \in \mathit{regions}(\Gamma,\ \Theta).\ {}^{\omega}p \in \{\overline{\ell}\}$. *(In English, loans in the closure's frame come from the current frame or a closure's captured frame in* Γ *or* Θ*)*
(6) $\Sigma;\ \bullet;\ \Gamma \natural \mathcal{F};\ \Theta \vdash \boxed{e} : \tau^{SI} \Rightarrow \Gamma_o \natural \mathcal{F}_o$

then

(1) $\bullet;\ \Gamma_o;\ \Theta \vdash_\omega p_b \Rightarrow \{\ \overline{\ell_b'}\ \}$
(2) $\forall r \in \mathit{dom}(\mathcal{F}_o),\ {}^{\omega}p \in \mathcal{F}_o(r).\texttt{root-of}(p) \in \mathit{dom}(\mathcal{F}_o) \vee \exists r' \mapsto \{\ \overline{\ell}\ \} \in \mathit{regions}(\Gamma_o,\ \Theta).\ {}^{\omega}p \in \{\ \overline{\ell}\ \}$.
(In English, loans in the closure's frame come from the current frame or a closure's captured frame in Γ_o *or* Θ*)*

Proof. In the T-Move case, the context is updated, but since $\pi \in \text{dom}(\mathcal{F})$, $\Gamma_o = \Gamma$, so the conclusions follow from the premises.

In the T-Copy, T-Function, T-Abort, T-Unit, T-u32, T-True, and T-False cases, $\Gamma_o = \Gamma$ so the conclusions follow from the premises.

In the T-IndexCopy, T-LetRegion, T-While, T-ForArray, T-ForSlice, T-Closure, T-Tuple, T-Array, T-Slice, T-Drop, T-Left, and T-Right cases, the proof is immediate from the induction hypothesis.

In the T-Borrow, T-BorrowIndex, and T-BorrowSlice cases, the proof follows from the induction hypothesis and Lemma E.56. The last condition is the restriction on loans in $\mathcal{F}$, but this is immediate by inspection of the ownership safety judgement. The only way to create new loans is to directly borrow a place, in which case we'd have that the loan is in the domain of $\mathcal{F}$, and otherwise the loans originated from the loan set in $\Gamma \natural \mathcal{F}$ of a reference being reborrowed, and we already know the property for $\Gamma \natural \mathcal{F}$.

In the T-Seq case, the proof follows from the induction hypothesis and Lemma E.17 (note that `gc-loans` does not change any types in the environment and produces related environments).

In the T-Branch and T-Match cases, the proof follows from the induction hypothesis, Lemma E.7, and Lemma E.57. We also need to show that rewriting preserves the restriction on loans in $\mathcal{F}$, but this is immediate because the most that rewriting can do is union together loan sets. The rest of the cases that use rewriting also use this same reasoning.

In the T-Let case, the proof follows from the induction hypothesis, Lemma E.7, and Lemma E.17 (note that `gc-loans` does not change any types in the environment and produces related environments).

In the T-AssignDeref case the proof follows from the induction hypothesis, and Lemma E.7.

In the T-Assign case, the proof follows from the induction hypothesis, Lemma E.7, and Lemma E.36.

In the T-App case, the proof follows from the induction hypothesis and Lemma E.7. □

Lemma E.59 (Outlives is Preserved under Safe Loan Updates). *If*

(1) $\bullet;\ \Gamma;\ \Theta \vdash_{\omega} p \Rightarrow \{\ \overline{\ell}\ \}$
(2) $\Gamma(r) = \{\}$
(3) $r \notin dom(\mathcal{F})$
(4) $r \neq r_1$ *and* $r \neq r_2$
(5) $\forall \tau' \in dom(\Gamma).\ r_1$ *and* r_2 *does not occur outside of a closure in* τ'
(6) $\bullet;\ \Gamma \natural \mathcal{F};\ \Theta \vdash^{\mu} r_1 :> r_2 \dashv \Gamma' \natural \mathcal{F}'$

then $\bullet;\ \Gamma[r \mapsto \{\ \overline{\ell_b}\ \}] \natural \mathcal{F};\ \Theta \vdash^{\mu} r_1 :> r_2 \dashv \Gamma'[r \mapsto \{\ \overline{\ell_b}\ \}] \natural \mathcal{F}'$

Proof. Proceed by induction on the outlives relation. The only interesting cases are OL-CombineConcrete, OL-CombineConcreteUnrestricted, and OL-CheckConcrete. They all proceed similarly. The closure restriction is immediate from the premise because no types are changed. The region not reborrowed restriction follows from the fact that r_1 and r_2 don't occur in any types in Γ outside of a closure, which means there's no place in the domain of Γ for there to be a reborrow of in $\overline{\ell}$. □

Lemma E.60 (Rewriting is Preserved under Safe Loan Updates). *If*

(1) $\bullet;\ \Gamma;\ \Theta \vdash_{\omega} p \Rightarrow \{\ \overline{\ell}\ \}$
(2) $\Gamma(r) = \{\}$
(3) $r \notin dom(\mathcal{F})$
(4) $\forall \tau \in \mathcal{F}.\ r$ *does not occur in* τ *or* τ_1 *or* τ_2
(5) $\forall r \in$ *free-regions*$(\tau_1) \cup$ *free-regions*(τ_2). $\forall \tau' \in dom(\Gamma).\ r$ *does not occur outside of a closure in* τ'.
(6) $\bullet;\ \Gamma \natural \mathcal{F};\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma' \natural \mathcal{F}'$

then $\bullet;\ \Gamma[r \mapsto \{\ \overline{\ell_b}\ \}] \natural \mathcal{F};\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma'[r \mapsto \{\ \overline{\ell_b}\ \}] \natural \mathcal{F}'$.

Proof. Proceed by induction on the rewriting derivation. The only interesting case is RR-Reference, in which case we apply Lemma E.59 and the induction hypothesis. The other cases all follow immediately or from the induction hypothesis. □

Lemma E.61 (Closure Bodies are Well-Typed under Safe Loan Updates). *If*

(1) $\Delta;\, \Gamma;\, \Theta \vdash_{\omega} p \Rightarrow \{\, \overline{\ell}\, \}$
(2) $\Gamma(r) = \{\}$
(3) $r \notin dom(\mathcal{F})$
(4) $\forall \tau \in \mathcal{F}.\ r$ *does not occur in* τ
(5) $\forall x \in free\text{-}vars(e).\ x \in dom(\mathcal{F})$
(6) $\forall r \in free\text{-}regions(e).\ r \in dom(\mathcal{F})$
(7) $\forall r \in dom(\mathcal{F}),\ {}^{\omega}p\ \in \mathcal{F}(r).\texttt{root-of}(p) \in dom(\mathcal{F}) \vee \exists r' \mapsto \{\overline{\ell}\} \in regions(\Gamma,\, \Theta).\ {}^{\omega}p\ \in \{\overline{\ell}\}$.
(In English, loans in the closure's frame come from the current frame or a closure's captured frame in Γ *or* Θ*)*
(8) $\forall r \in free\text{-}regions(\tau^{SI}).\ \forall \tau' \in dom(\Gamma).\ r$ *does not occur in* τ'
(9) $\Sigma;\, \bullet;\, \Gamma \natural \mathcal{F};\, \Theta \vdash \boxed{e} : \tau^{SI} \Rightarrow \Gamma_o \natural \mathcal{F}_o$
(10) $\Sigma;\, \bullet \vdash \Gamma[r \mapsto \{\, \overline{\ell}\, \}] \natural \mathcal{F}$

then

(1) $\Sigma;\, \bullet;\, \Gamma[r \mapsto \{\, \overline{\ell}\, \}] \natural \mathcal{F};\, \Theta \vdash \boxed{e} : \tau^{SI} \Rightarrow \Gamma_o[r \mapsto \{\, \overline{\ell}\, \}] \natural \mathcal{F}_o$.
(2) $\forall r \in dom(\mathcal{F}_o),\ {}^{\omega}p\ \in \mathcal{F}_o(r).\texttt{root-of}(p) \in dom(\mathcal{F}_o) \vee \exists r' \mapsto \{\, \overline{\ell}\, \} \in regions(\Gamma_o,\, \Theta).$ ${}^{\omega}p \in \{\, \overline{\ell}\, \}$.
(In English, loans in the closure's frame come from the current frame or a closure's captured frame in Γ_o *or* Θ*)*

Proof. Proceed by induction over the typing derivation for e.

In T-Move and T-Copy, and T-Borrow cases, we can apply Lemma E.56 and note that type computation is unaffected by changes in loan sets.

In the T-Borrow case, we can apply Lemma E.56 and note that type computation is unaffected by changes in loan sets. The last condition is the restriction on loans in $\mathcal{F}$, but this is immediate by inspection of the ownership safety judgement. The only way to create new loans is to directly borrow a place, in which case we'd have that the loan is in the domain of $\mathcal{F}$, and otherwise the loans originated from the loan set in $\Gamma \natural \mathcal{F}$ of a reference being reborrowed, and we already know the loan set restriction for $\Gamma \natural \mathcal{F}$. The rest of the cases that involve borrowing use similar reasoning.

In the T-BorrowIndex and T-IndexCopy cases, we can apply Lemma E.56, the induction hypothesis, and the note about type computation.

In the T-BorrowSlice case, we can apply Lemma E.56, the induction hypothesis, the note about type computation, and Lemma E.58.

In the T-Seq case, we can apply the induction hypothesis, for which we need to apply Lemma E.58 and use the fact that garbage collection produces related environments with Lemma E.17.

In the T-Branch case we can apply the induction hypothesis, Lemma E.58, and Lemma E.60. We also need to show that rewriting preserves the restriction on loans in $\mathcal{F}$, but this is immediate because the most that rewriting can do is union together loan sets. The rest of the cases that use rewriting also use this same reasoning.

In the T-Let case, we can apply the induction hypothesis, Lemma E.58, Lemma E.60, and the fact that garbage collection produces related environments with Lemma E.17. The last obligation is the region not reborrowed condition. Note that by environment well formedness and our hypothesis, any free regions in the types are not in any non closure types. Therefore, there are no places in Γ

with the region for the new loans in $\overline{\ell}$ to even contain a reborrow of, meaning the regions are not reborrowed. The other cases which require region not reborrowed proceed by the same reasoning.

In the T-LetRegion, T-While, T-Closure, T-Tuple, T-Array, T-Slice, T-Left, and T-Right cases, the proofs follows from the induction hypothesis and Lemma E.58.

In the T-AssignDeref case, we can apply the induction hypothesis, Lemma E.58, the fact that type computation is unaffected by loan updates, Lemma E.60, and Lemma E.56.

In the T-Assign case, we can apply the induction hypothesis, Lemma E.58, the fact that type computation is unaffected by loan updates, Lemma E.60, and Lemma E.56. The last obligation is the unique to judgement, which is unaffected by loan updates.

In the T-ForArray and T-ForSlice cases, we can apply the induction hypothesis and Lemma E.58. The remaining region not reborrowed obligation follows from the same reasoning in the T-Let case.

In the T-Function, T-Abort, T-Unit, T-u32, T-True, and T-False cases, the proof is immediate.

In the T-App case, we can apply the induction hypothesis, Lemma E.58 and Lemma E.60. Note that the well formedness judgements are unaffected by loan updates. The last obligation is the region not reborrowed judgement, follows from the same reasoning in the T-Let case.

In the T-Drop case, we can apply the induction hypothesis, Lemma E.58, and Lemma E.17, noting that making the type of a place dead produces a related environment.

In the T-Match case we can apply the induction hypothesis, Lemma E.58, and Lemma E.60. The last obligation is the region not reborrowed judgement, follows from the same reasoning in the T-Let case. □

Lemma E.62 (Values are Well-Typed under Safe Loan Updates). *If* $\bullet;\ \Gamma;\ \Theta \vdash_\omega p \Rightarrow \{\,\overline{\ell}\,\}$ *and* $\Gamma;\ \Theta \vdash r \ \mathtt{rnic}$ *and* $\Gamma(r) = \emptyset$ *and* $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau^{SI} \Rightarrow \Gamma$*, then* $\Sigma;\ \bullet;\ \Gamma[r \mapsto \{\,\overline{\ell}\,\}];\ \Theta \vdash \boxed{v} : \tau^{SI} \Rightarrow \Gamma[r \mapsto \{\,\overline{\ell}\,\}]$.

Proof. We proceed by induction on the value typing relation.

For T-u32, T-True, T-False, the result is immediate.

For T-Tuple and T-Array, we apply the induction hypothesis to each entry.

T-Pointer
$$\frac{\Sigma;\ \Gamma \vdash \mathcal{R}^{\square}[\pi] : \tau^{\text{XI}} \qquad {}^{\omega}\pi \in \Gamma(r)}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\mathtt{ptr}\ \mathcal{R}^{\square}[\pi]} : \&r\ \omega\ \tau^{\text{XI}} \Rightarrow \Gamma}$$

In the T-Pointer case, both judgements in the premise are unaffected by taking an empty loan set and adding loans to it, so the case is immediate.

T-ClosureValue
$$\frac{\begin{array}{c}\text{free-vars}(e) \setminus \overline{x} = \overline{x_f} = \text{dom}(\mathcal{F}_c)|_{\text{VAR}} \\ \overline{r} = \overline{\text{free-regions}(\Gamma(x_f))},\ (\text{free-regions}(e) \setminus (\overline{\text{free-regions}(\tau^{\text{SI}})}, \text{free-regions}(\tau_r^{\text{SI}}))) = \text{dom}(\mathcal{F}_c)|_{\text{RGN}} \\ \Sigma;\ \Gamma \vdash \varsigma_c : \mathcal{F}_c \qquad \Sigma;\ \Delta;\ \Gamma \natural \mathcal{F}_c,\ x_1 : \tau_1^{\text{SI}}, \ldots, x_n : \tau_n^{\text{SI}};\ \Theta \vdash \boxed{e} : \tau_r^{\text{SI}} \Rightarrow \Gamma' \natural \mathcal{F}\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\langle \varsigma_c,\ |x_1 : \tau_1^{\text{SI}}, \ldots, x_n : \tau_n^{\text{SI}}| \to \tau_r^{\text{SI}}\ \{\ e\ \}\ \rangle} : (\tau_1^{\text{SI}}, \ldots, \tau_n^{\text{SI}}) \overset{\mathcal{F}_c}{\to} \tau_r^{\text{SI}} \Rightarrow \Gamma}$$

In the T-ClosureValue case, first we invert the stack frame typing hypothesis to get that $\forall x \in \text{dom}(\varsigma).\ \Sigma;\ \bullet;\ \Gamma \natural \mathcal{F}_c;\ \Theta \vdash \boxed{\varsigma(x)} : \mathcal{F}_c(x) \Rightarrow \Gamma \natural \mathcal{F}_c$. We can apply the induction hypothesis to each of these statements, and apply WF-Frame to get $\Sigma;\ \Gamma[r \mapsto \{\,\overline{\ell}\,\}] \vdash \varsigma_c : \mathcal{F}_c$.

Next we need to show that the body remains well typed. This follows from Lemma E.61.

In all other value cases, the typing judgement holds immediately. □

Lemma E.63 (Stack Validity is Preserved under Safe Loan Updates). *If* $\bullet;\ \Gamma;\ \Theta \vdash_{\omega} p \Rightarrow \{\,\overline{\ell}\,\}$ *and* $\Gamma;\ \Theta \vdash r\ \texttt{rnic}$ *and and* $\Gamma(r) = \emptyset$ *and* $\Sigma \vdash \sigma : \Gamma$*, then* $\Sigma \vdash \sigma : \Gamma[r \mapsto \{\,\overline{\ell}\,\}]$.

Proof. We proceed by induction on the stack validity. There are two cases, WF-StackEmpty, and WF-StackFrame. WF-StackEmpty is impossible, since we already know that r is in Γ.

$$\textsc{WF-StackFrame}\quad \frac{\Sigma \vdash \sigma : \Gamma \qquad \mathrm{dom}(\varsigma) = \mathrm{dom}(\mathcal{F})|_x \qquad \forall x \in \mathrm{dom}(\varsigma).\ \Sigma;\ \bullet;\ \Gamma \natural \mathcal{F};\ \bullet \vdash \boxed{(\sigma \natural \varsigma)(x)} : (\Gamma \natural \mathcal{F})(x) \Rightarrow \Gamma \natural \mathcal{F}}{\Sigma \vdash \sigma \natural \varsigma : \Gamma \natural \mathcal{F}} \qquad \textsc{WF-StackEmpty}\quad \frac{}{\Sigma \vdash \bullet : \bullet}$$

In the case of WF-StackFrame, we have to show that the values remain well-typed in the updated environment. For the remaining Γ', if $r \in \Gamma'$, then we apply the induction hypothesis, otherwise we just use the derivation from the premise.

To show that the values in the stack are still well typed in $\Gamma[r \mapsto \{\overline{\ell}\}]$, we apply Lemma E.62. □

E.13 Preservation of Rewriting under Parallel Type Checking Lemmas

Lemma E.64 (Region Rewriting is Preserved by Parallel Loan Updates). *If*

(1) $\bullet;\ \Gamma';\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma_s$
(2) $\bullet;\ \Gamma;\ \Theta, \tau_1, \Theta' \vdash_{\omega} p \Rightarrow \{\,\overline{\ell}\,\}$
(3) $\bullet;\ \Gamma';\ \Theta, \tau_2, \Theta' \vdash_{\omega} p \Rightarrow \{\,\overline{\ell}'\,\}$
(4) $dom(\Gamma) = dom(\Gamma')$ *and* $dom(\Gamma_o) = dom(\Gamma_o')$
(5) $\forall r \in dom(\Gamma).\ \Gamma'(r) \subseteq \Gamma(r)$
(6) $\forall r \in dom(\Gamma_o).\ \Gamma_o'(r) \subseteq \Gamma_o(r)$

then $\bullet;\ \Gamma'[r \mapsto \{\,\overline{\ell}'\,\}];\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma_s'$.

Proof. Proceed by induction on the rewriting derivation. The only interesting case is RR-Reference, in which case we proceed by induction on the outlives derivation.

The interesting cases are OL-CombineConcrete, OL-CombineConcreteUnrestricted, and OL-CheckConcrete since the type variable environment is guaranteed to be empty and the other cases all involve abstract regions. The only interesting obligations are the region not reborrowed ones, since the types in the environments are unchanged. This amounts to showing that for all loans in $\{\,\overline{\ell}'\,\}$, none are reborrows of references that have either r_1 or r_2 as their region.

Assume one such loan ${}^{\omega'} * \pi \in \overline{\ell}'$ exists. Assume without loss of generality that $\pi : \&r_1\,\omega''\,\tau \in \Gamma$. By the well formedness of Θ, τ_1, Θ', since τ_1 contains r_1, it must be the case that $\Gamma(r_1) \neq \emptyset$. When checking ownership safety for $*\pi$, we'll need to show that excluding π, there are no conflicts in r' with any lons in r'. But since the exclusion clause doesn't exclude when we have a reference in theta, and τ_1 contains r_1, it must be the case that r_1 will not be excluded, and we will then find loan conflicts, which is a contradiction. □

Lemma E.65 (Region Rewriting is Preserved by Type Checking Parallel Expressions). *If*

(1) $\bullet;\ \Gamma';\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma_s$
(2) $\Sigma;\ \bullet;\ \Gamma;\ \Theta, \tau_1, \Theta' \vdash \boxed{e} : \tau \Rightarrow \Gamma_o$
(3) $\Sigma;\ \bullet;\ \Gamma';\ \Theta, \tau_2, \Theta' \vdash \boxed{e} : \tau \Rightarrow \Gamma_o'$
(4) $dom(\Gamma) = dom(\Gamma')$ *and* $dom(\Gamma_o) = dom(\Gamma_o')$
(5) $\forall r \in dom(\Gamma).\ \Gamma'(r) \subseteq \Gamma(r)$
(6) $\forall r \in dom(\Gamma_o).\ \Gamma_o'(r) \subseteq \Gamma_o(r)$

then $\bullet;\ \Gamma'_o;\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma'_s$.

Proof. Proceed by induction on the typing derivation using Γ' (note that since the typing derivation is by the structure of e, we can simultaneously induct on the typing derivation using Γ).

In the T-Copy, T-Function, T-Abort, T-Unit, T-u32, T-True, and T-False cases, the proof is immediate, with $\Gamma_o = \Gamma$ and $\Gamma'' = \Gamma'$.

In the T-IndexCopy, T-LetRegion, T-While, T-ForArray, T-ForSlice, T-Closure, T-Tuple, T-Array, T-Slice, T-Left, and T-Right cases, the proof follows immediately from the induction hypothesis.

In the T-Move case, the proof is immediate because making a type a dead does not add additional obligations in the rewriting judgement.

In the T-Borrow, T-BorrowIndex, and T-BorrowSlice cases, we proceed by the induction hypothesis and apply Lemma E.64.

In the T-Seq case, we just need to show that garbage collecting loans preserves rewriting. But this is immediate, because garbage collection loans can only clear loan sets, which makes the requirements in rewriting strictly easier since it could only *remove* reborrows, not add them.

In the T-AssignDeref, T-AppFunction, and T-AppClosure cases, we first apply the induction hypothesis. Note that the output of the region rewriting at most only combines loan sets. As such, the region rewriting is preserved, because the main condition, the region not reborrowed requirement on the regions in the types, will still consider the same set of loans. For all region rewriting cases below, use this same reasoning.

In the T-Assign case, we apply the induction hypothesis and reason about the rewriting as above, but we additionally need to know that the type update maintains the region not reborrowed and closure restrictions. For both, it's immediate because the rewriting in the hypothesis of the typing rule will check the same restrictions.

In the T-Branch and T-Match cases, we apply the induction hypothesis, the reasoning above about rewriting, and the fact that $\Cup$ only unions together the loan sets from Γ_2 and Γ_3, both of which had the rewriting restrictions true by the induction hypothesis.

In the T-Let case, we again apply the induction hypothesis and the reasoning above about rewriting, but additionally use the same reasoning as the garbage collection case as well.

In the T-Drop case, we just need to show that rewriting is preserved by making a place dead in order to apply the induction hypothesis. This is immediate though, because all of the obligations in rewriting are either the same difficulty or made easier by making a place dead. □

Lemma E.66 (Outlives Still Holds with Smaller Continuation Contexts). *If* $\bullet;\ \Gamma;\ \Theta, \tau \vdash^{\mu} r_1 :> r_2 \dashv \Gamma'$ *then* $\bullet;\ \Gamma;\ \Theta \vdash^{\mu} r_1 :> r_2 \dashv \Gamma'$.

Proof. Proceed by induction on the outlives judgement. The only interesting cases are the OL-CombineConcrete, OL-CombineConcreteUnrestricted, and OL-CheckConcrete cases which all proceed similarly. The main obligations, the region not reborrowed and closure restriction judgements, are immediate since they are either unaffected by or have strictly fewer obligations in the smaller temporary typing Θ. □

Lemma E.67 (Region Rewriting Still Holds with Smaller Continuation Contexts). *If* $\bullet;\ \Gamma;\ \Theta, \tau \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma'$ *then* $\bullet;\ \Gamma;\ \Theta \vdash^{\mu} \tau_1 \rightsquigarrow \tau_2 \dashv \Gamma'$.

Proof. Proceed by induction on the rewriting judgement. The only interesting case is RR-Reference, in which case we just apply Lemma E.66. □

E.14 Progress

Lemma E.68 (Progress). *If* $\Sigma; \bullet; \Gamma; \Theta \vdash \boxed{e} : \tau^{SI} \Rightarrow \Gamma'$ *and* $\Sigma \vdash \sigma : \Gamma$ *, then either* e *is a value,* e *is an* `abort!` *(…), or* $\exists\, \sigma', e'.\ \Sigma \vdash (\sigma; \boxed{e}) \rightarrow (\sigma'; \boxed{e'})$.

Proof. We proceed by induction on the derivation $\Sigma; \bullet; \Gamma; \Theta \vdash \boxed{e} : \tau \Rightarrow \Gamma'$.

Case T-Move:

From premise:

$$\textsc{T-Move}\quad \frac{\Delta; \Gamma; \Theta \vdash_{\texttt{uniq}} \pi \Rightarrow \{\ {}^{\texttt{uniq}}\pi\ \} \qquad \Gamma(\pi) = \tau^{\text{SI}} \qquad \texttt{noncopyable}_{\Sigma}\ \tau^{\text{SI}}}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{\pi} : \tau^{\text{SI}} \Rightarrow \Gamma[\pi \mapsto \tau^{\text{SI}^{\dagger}}]}$$

We want to step with:

$$\textsc{E-Move}\quad \frac{\sigma \vdash \pi \Downarrow \pi \mapsto \_[v]}{\Sigma \vdash (\sigma; \boxed{\pi}) \rightarrow (\sigma[\pi \mapsto \texttt{dead}]; \boxed{v})}$$

Applying Lemma E.4 to $\Delta;\ \texttt{uniq};\ \Gamma \vdash_{\pi} \{\ {}^{\texttt{uniq}}\pi\ \} \Rightarrow$, $\Delta;\ \Gamma \vdash_{\texttt{uniq}} \pi : \tau^{\text{SI}}$ (from $\Gamma(\pi) = \tau^{\text{SI}}$ by TC-Place), and $\Sigma \vdash \sigma : \Gamma$ to conclude that $\sigma \vdash \pi \Downarrow \_ \mapsto \_[v]$. Thus, we can step with E-Move.

Case T-Copy:

From premise:

$$\textsc{T-Copy}\quad \frac{\Delta; \Gamma; \Theta \vdash_{\texttt{shrd}} p \Rightarrow \{\ \overline{\ell}\ \} \qquad \Delta; \Gamma \vdash_{\texttt{shrd}} p : \tau^{\text{SI}} \qquad \texttt{copyable}_{\Sigma}\ \tau^{\text{SI}}}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{p} : \tau^{\text{SI}} \Rightarrow \Gamma}$$

We want to step with:

$$\textsc{E-Copy}\quad \frac{\sigma \vdash p \Downarrow \_ \mapsto \_[v]}{\Sigma \vdash (\sigma; \boxed{p}) \rightarrow (\sigma; \boxed{v})}$$

Applying Lemma E.4 to $\Delta;\ \texttt{shrd};\ \Gamma \vdash_{p} \{\ \overline{\ell}\ \} \Rightarrow$, $\Delta;\ \Gamma \vdash_{\texttt{shrd}} p : \tau^{\text{SI}}$, and $\Sigma \vdash \sigma : \Gamma$ to conclude that $\sigma \vdash p \Downarrow \_ \mapsto \_[v]$. Thus, we can step with E-Copy.

Case T-Borrow:

From premise:

$$\textsc{T-Borrow}\quad \frac{\Gamma(r) = \emptyset \qquad \Gamma; \Theta \vdash r\ \texttt{rnic} \qquad \Delta; \Gamma; \Theta \vdash_{\omega} p \Rightarrow \{\ \overline{\ell}\ \} \qquad \Delta; \Gamma \vdash_{\omega} p : \tau^{\text{XI}}}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{\&r\ \omega\ p} : \&r\ \omega\ \tau^{\text{XI}} \Rightarrow \Gamma[r \mapsto \{\ \overline{\ell}\ \}]}$$

We want to step with:

$$\textsc{E-Borrow}\quad \frac{\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[\_]}{\Sigma \vdash (\sigma; \boxed{\&r\ \omega\ p}) \rightarrow (\sigma; \boxed{\texttt{ptr}\ \mathcal{R}})}$$

Applying Lemma E.4 to $\Delta;\ \omega;\ \Gamma \vdash_{p} \{\ \overline{\ell}\ \} \Rightarrow$, $\Delta;\ \Gamma \vdash_{\omega} p : \tau^{\text{XI}}$, and $\Sigma \vdash \sigma : \Gamma$ to conclude that $\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[\_]$. Thus, we can step with E-Borrow.

Case T-BorrowIndex:

From premise:

$$\textsc{T-BorrowIndex}\quad \frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \texttt{u32} \Rightarrow \Gamma' \qquad \Gamma'(r) = \emptyset \qquad \Gamma';\ \Theta \vdash r\ \texttt{rnic} \\ \Delta;\ \Gamma';\ \Theta \vdash_{\omega} p \Rightarrow \{\ \overline{\ell}\ \} \qquad \Delta;\ \Gamma' \vdash_{\omega} p : \tau^{\text{XI}} \\ \tau^{\text{XI}} = [\tau^{\text{SI}};\ n] \vee \tau^{\text{XI}} = [\tau^{\text{SI}}]\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\&r\ \omega\ p[e]} : \&r\ \omega\ \tau^{\text{SI}} \Rightarrow \Gamma'[r \mapsto \{\ \overline{\ell}\ \}]}$$

We proceed based on whether or not e is a value. If it is not, we can decompose our expression into the evaluation context $\&\rho\ \omega\ p[\square]$ and redex e. Then, by applying our induction hypothesis to the typing derivation for e, we know either that e is an `abort!` expression or it e steps to some e'. In the former case, we can step with E-EvalCtxAbort. In the latter case, we can plug e' back into our evaluation context and step with E-EvalCtx.

If e is a value, we would like to step with one of:

$$\textsc{E-BorrowIndex}\quad \frac{\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[[v_0,\ \ldots,\ v_n]] \qquad 0 \leq n_i \leq n}{\Sigma \vdash (\sigma;\ \boxed{\&r\ \omega\ p[n_i]}\ ) \rightarrow (\sigma;\ \boxed{\texttt{ptr}\ \mathcal{R}[n_i]}\ )}$$

$$\textsc{E-BorrowIndexOOB}\quad \frac{\sigma \vdash p \Downarrow \_ \mapsto \_[[v_0,\ \ldots,\ v_n]] \qquad n_i < 0 \vee n_i > n}{\Sigma \vdash (\sigma;\ \boxed{\&r\ \omega\ *p[n_i]}\ ) \rightarrow (\sigma;\ \boxed{\texttt{abort!}(\text{“attempted to index out of bounds”})}\ )}$$

Since e is a value, we can apply Lemma E.15 to get $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma'$. Applying Lemma E.27, then gives us $\Sigma \vdash \sigma : \Gamma'$.

Then, we can apply Lemma E.4 to $\Delta;\ \omega;\ \Gamma' \vdash_p \{\ \overline{\ell}\ \} \Rightarrow,\ \Delta;\ \Gamma' \vdash_{\omega} p : \tau^{\text{XI}}$, and $\Sigma \vdash \sigma : \Gamma'$ to get $\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[v]$. By Lemma E.1, we know that $v = [v_0,\ \ldots,\ v_n]$ since the type tells us the shape of the resultant value.

Since we wish to step with one of E-BorrowIndex and E-BorrowIndexOOB, we should observe that we now have their shared requirement: $\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[[v_0,\ \ldots,\ v_n]]$. Their other obligations are a bounds check which together are a tautology (i.e. one of them must hold). Thus, we can step with the appropriate rule based on whether or not the bounds check succeeds.

Case T-BorrowSlice:

From premise:

$$\textsc{T-BorrowSlice}\quad \frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1} : \texttt{u32} \Rightarrow \Gamma_1 \qquad \Sigma;\ \Delta;\ \Gamma_1;\ \Theta \vdash \boxed{e_2} : \texttt{u32} \Rightarrow \Gamma_2 \\ \Gamma_2(r) = \emptyset \qquad \Gamma_2;\ \Theta \vdash r\ \texttt{rnic} \qquad \Delta;\ \Gamma_2;\ \Theta \vdash_{\omega} p \Rightarrow \{\ \overline{\ell}\ \} \qquad \Delta;\ \Gamma_2 \vdash_{\omega} p : [\tau^{\text{SI}}]\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\&r\ \omega\ p[e_1..e_2]} : \&r\ \omega\ [\tau^{\text{SI}}] \Rightarrow \Gamma_2[r \mapsto \{\ \overline{\ell}\ \}]}$$

The proof proceeds along similar lines as for T-BorrowIndex. We proceed based on whether or not e_1 and e_2 are values.

If e_1 is not a value, then we can decompose our whole expression into the evaluation context $\&\rho\ \omega\ p[\square..e_2]$ and redex e_1. Then, by applying our induction hypothesis to e_1, we know either that e_1 steps to some e_1' or is an `abort!` expression. In the former case, this satisfies our requirement since we can plug e' back into our evaluation context. In the latter case, we can step with E-EvalCtxAbort.

If e_1 is a value and e_2 is not a value, then we can decompose our whole expression into the evaluation context $\&\rho\ \omega\ p[v_1..\square]$ and redex e_2. Then, by applying our induction hypothesis to e_2, we know either that e_2 steps to some e_2' or is an `abort!` expression. In the former case, this satisfies our requirement since we can plug e' back into our evaluation context. In the latter case, we can step with E-EvalCtxAbort.

If e_1 and e_2 are values, we would like to step with one of:

$$\textsc{E-BorrowSlice}\quad \frac{\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[[v_0\,,\ \ldots\,,\ v_n]] \qquad 0 \le n_1 \le n_2 \le n}{\Sigma \vdash (\sigma;\ \boxed{\&r\ \omega\ p[n_1..n_2]}\,) \to (\sigma;\ \boxed{\texttt{ptr}\ \mathcal{R}[n_1..n_2]}\,)}$$

$$\textsc{E-BorrowSliceOOB}\quad \frac{\sigma \vdash p \Downarrow \_ \mapsto \_[[v_0\,,\ \ldots\,,\ v_n]] \qquad n_1 < 0 \lor n_1 > n \lor n_2 < 0 \lor n_2 > n \lor n_1 > n_2}{\Sigma \vdash (\sigma;\ \boxed{\&r\ \omega\ p[n_1..n_2]}\,) \to (\sigma;\ \boxed{\texttt{abort!}(\text{“attempted to slice out of bounds”})}\,)}$$

Since e_1 is a value, we can apply Lemma E.15 to get $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma_1$. Then, since e_2 is also a value, we can apply Lemma E.15 to get $\Sigma;\ \bullet \vdash \Gamma_1 \rhd \Gamma_2$. Then, by transitivity, we get $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma_2$. Then, applying Lemma E.27 gives us $\Sigma \vdash \sigma : \Gamma_2$.

Then, we can apply Lemma E.4 to $\Delta;\ \omega;\ \Gamma_2 \vdash_p \{\,\overline{\ell}\,\} \Rightarrow$, $\Delta;\ \Gamma_2 \vdash_\omega p : [\tau^{\text{SI}}]$, and $\Sigma \vdash \sigma : \Gamma_2$ to get $\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[v]$. By Lemma E.1, we know that $v = [v_0\,,\ \ldots\,,\ v_n]$ since the type tells us the shape of the resultant value.

Since we wish to step with one of E-BorrowSlice and E-BorrowSliceOOB, we should observe that we now have their shared requirement: $\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[[v_0\,,\ \ldots\,,\ v_n]]$. Their other obligations are a bounds check which together are a tautology (i.e. one of them must hold). Thus, we can step with the appropriate rule based on whether or not the bounds check succeeds.

Case T-IndexCopy:

From premise:

$$\textsc{T-IndexCopy}\quad \frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \texttt{u32} \Rightarrow \Gamma' \qquad \Delta;\ \Gamma';\ \Theta \vdash_{\texttt{shrd}} p \Rightarrow \{\,\overline{\ell}\,\} \\ \Delta;\ \Gamma' \vdash_{\texttt{shrd}} p : \tau^{\text{XI}} \qquad \tau^{\text{XI}} = [\tau^{\text{SI}};\ n] \lor \tau^{\text{XI}} = [\tau^{\text{SI}}] \qquad \texttt{copyable}_\Sigma\ \tau^{\text{SI}}\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{p[e]} : \tau^{\text{SI}} \Rightarrow \Gamma'}$$

We proceed based on whether or not e is a value. If it is not, we can decompose our expression into the evaluation context $p[\square]$ and redex e. Then, by applying our induction hypothesis to e, we know either that e steps to some e' or is an `abort!` expression. In the former case, this satisfies our requirement since we can plug e' back into our evaluation context. In the latter case, we can step with E-EvalCtxAbort.

If e is a value, we would like to step with one of:

$$\textsc{E-IndexCopy}\quad \frac{\sigma \vdash p \Downarrow \_ \mapsto \_[[v_0\,,\ \ldots\,,\ v_{n_i}\,,\ \ldots\,,\ v_n]]}{\Sigma \vdash (\sigma;\ \boxed{p[n_i]}\,) \to (\sigma;\ \boxed{v_{n_i}}\,)}$$

$$\textsc{E-IndexCopyOOB}\quad \frac{\sigma \vdash p \Downarrow \_ \mapsto \_[[v_0\,,\ \ldots\,,\ v_n]] \qquad n_i < 0 \lor n_i > n}{\Sigma \vdash (\sigma;\ \boxed{p[n_i]}\,) \to (\sigma;\ \boxed{\texttt{abort!}(\text{“attempted to index out of bounds”})}\,)}$$

Since e is a value, we can apply Lemma E.15 to get $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma'$. Applying Lemma E.27, then gives us $\Sigma \vdash \sigma : \Gamma'$.

Then, we can apply Lemma E.4 to $\Delta;\ \omega;\ \Gamma' \vdash_p \{\,\overline{\ell}\,\} \Rightarrow$, $\Delta;\ \Gamma' \vdash_\omega p : \tau^{\text{XI}}$, and $\Sigma \vdash \sigma : \Gamma'$ to get $\sigma \vdash p \Downarrow \_ \mapsto \_[v]$. By Lemma E.1, we know that $v = [v_0\,,\ \ldots\,,\ v_n]$ since the type tells us the shape of the resultant value.

Since we wish to step with one of E-IndexCopy and E-IndexCopyOOB, we should observe that we now have their shared requirement: $\sigma \vdash p \Downarrow \_ \mapsto \_[[v_0, \ldots, v_n]]$. Their other obligations are a bounds check which together are a tautology (i.e. one of them must hold). Thus, we can step with the appropriate rule based on whether or not the bounds check succeeds.

Case T-Seq:

From premise:

T-Seq
$$\frac{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1} : \tau_1^{\mathrm{SI}} \Rightarrow \Gamma_1 \qquad \Sigma;\ \Delta;\ \text{gc-loans}_{\Theta}(\Gamma_1);\ \Theta \vdash \boxed{e_2} : \tau_2^{\mathrm{SI}} \Rightarrow \Gamma_2}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1;\ e_2} : \tau_2^{\mathrm{SI}} \Rightarrow \Gamma_2}$$

We proceed based on whether or not e_1 is a value. If it is not, we can decompose our expression into the evaluation context $\square;\ e_2$ and redex e_1. Then, by applying our induction hypothesis to e_1, we know either that e_1 steps to some e_1' or is an `abort!` expression. In the former case, this satisfies our requirement since we can plug e_1' back into our evaluation context. In the latter case, we can step with E-EvalCtxAbort.

If e_1 is a value, we can step with:

E-Seq
$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{v;\ e}) \rightarrow (\sigma;\ \boxed{e})}$$

Case T-Branch:

From premise:

T-Branch
$$\frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1} : \mathsf{bool} \Rightarrow \Gamma_1 \qquad \Sigma;\ \Delta;\ \Gamma_1;\ \Theta \vdash \boxed{e_2} : \tau_2^{\mathrm{SI}} \Rightarrow \Gamma_2 \\ \Sigma;\ \Delta;\ \Gamma_1;\ \Theta \vdash \boxed{e_3} : \tau_3^{\mathrm{SI}} \Rightarrow \Gamma_3 \qquad \tau^{\mathrm{SI}} = \tau_2^{\mathrm{SI}} \vee \tau^{\mathrm{SI}} = \tau_3^{\mathrm{SI}} \\ \Delta;\ \Gamma_2;\ \Theta \vdash^{+} \tau_2^{\mathrm{SI}} \rightsquigarrow \tau^{\mathrm{SI}} \dashv \Gamma_2' \qquad \Delta;\ \Gamma_3;\ \Theta \vdash^{+} \tau_3^{\mathrm{SI}} \rightsquigarrow \tau^{\mathrm{SI}} \dashv \Gamma_3' \qquad \Gamma_2' \Cup \Gamma_3' = \Gamma'\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\mathtt{if}\ e_1\ \{\ e_2\ \}\ \mathtt{else}\ \{\ e_3\ \}} : \tau^{\mathrm{SI}} \Rightarrow \Gamma'}$$

We proceed based on whether or not e_1 is a value. If it is not, we can decompose our expression into the evaluation context $\mathtt{if}\ \square\ \{\ e_2\ \}\ \mathtt{else}\ \{\ e_3\ \}$ and redex e_1. Then, by applying our induction hypothesis to e_1, we know either that e_1 steps to some e_1' or is an `abort!` expression. In the former case, this satisfies our requirement since we can plug e_1' back into our evaluation context. In the latter case, we can step with E-EvalCtxAbort.

If e_1 is a value, we would like to step with one of:

E-IfTrue
$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{\mathtt{if\ true}\ \{\ e_1\ \}\ \mathtt{else}\ \{\ e_2\ \}}) \rightarrow (\sigma;\ \boxed{e_1})}$$

E-IfFalse
$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{\mathtt{if\ false}\ \{\ e_1\ \}\ \mathtt{else}\ \{\ e_2\ \}}) \rightarrow (\sigma;\ \boxed{e_2})}$$

Since e_1 is a value, applying Lemma E.1 tells us that e_1 is either `true` or `false`. In the former case, we can step with E-IfTrue and in the latter case, we can step with E-IfFalse

Case T-Let:

From premise:

T-Let
$$\frac{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1} : \tau_1^{\text{SI}} \Rightarrow \Gamma_1 \qquad \Delta;\ \Gamma_1;\ \Theta \vdash^{+} \tau_1^{\text{SI}} \rightsquigarrow \tau_a^{\text{SI}} \dashv \Gamma_1' \qquad \forall r \in \text{free-regions}(\tau_a^{\text{SI}}).\ \Gamma_1' \vdash r\ \texttt{rnrb} \qquad \Sigma;\ \Delta;\ \text{gc-loans}_{\Theta}(\Gamma_1',\ x\ :\ \tau_a^{\text{SI}});\ \Theta \vdash \boxed{e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2,\ x\ :\ \tau^{\text{SD}}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{let}\ x : \tau_a^{\text{SI}} = e_1;\ e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2}$$

We proceed based on whether or not e_1 is a value. If it is not, we can decompose our expression into the evaluation context $\texttt{let}\ x : \tau_a^{\text{SI}} = \square;\ e_2$ and redex e_1. Then, by applying our induction hypothesis to e_1, we know either that e_1 steps to some e_1' or is an abort! expression. In the former case, this satisfies our requirement since we can plug e_1' back into our evaluation context. In the latter case, we can step with E-EvalCtxAbort.

If e_1 is a value, we can step with:

E-Let
$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{let}\ x : \tau_a^{\text{SI}} = v;\ e}) \rightarrow (\sigma,\ x \mapsto v;\ \boxed{\texttt{shift}\ e})}$$

Case T-LetRegion:

From premise:

T-LetRegion
$$\frac{\Sigma;\ \Delta;\ \Gamma,\ r \mapsto \{\};\ \Theta \vdash \boxed{e} : \tau^{\text{SI}} \Rightarrow \Gamma',\ r \mapsto \{\bar{\ell}\}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{letrgn}\ \texttt{<}r\texttt{>}\ \{\ e\ \}} : \tau^{\text{SI}} \Rightarrow \Gamma'}$$

We proceed based on whether or not e is a value. If it is not, we can decompose our expression into the evaluation context $\texttt{letrgn}\ \texttt{<}r\texttt{>}\ \{\ \square\ \}$ and redex e. Then, by applying our induction hypothesis to e, we know either that e steps to some e' or is an abort! expression. In the former case, this satisfies our requirement since we can plug e' back into our evaluation context. In the latter case, we can step with E-EvalCtxAbort.

If e is a value, we can step with:

E-LetRegion
$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{letrgn}\ \texttt{<}r\texttt{>}\ \{\ v\ \}}) \rightarrow (\sigma;\ \boxed{v})}$$

Case T-Assign:

From premise:

T-Assign
$$\frac{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau^{\text{SI}} \Rightarrow \Gamma_1 \qquad \Gamma_1(\pi) = \tau^{\text{SX}} \qquad \tau^{\text{SX}} = \&r\ \omega\ \tau^{\text{XI}} \implies r \text{ is unique to } \pi \text{ in } \Gamma_1 \qquad \Delta;\ \Gamma_1 \trianglerighteq *\pi;\ \Theta \vdash^{=} \tau^{\text{SI}} \rightsquigarrow \tau^{\text{SX}} \dashv \Gamma' \qquad (\tau^{\text{SX}} = \tau^{\text{SD}} \lor \Delta;\ \Gamma';\ \Theta \vdash_{\texttt{uniq}} \pi \Rightarrow \{\ {}^{\texttt{uniq}}\pi\ \})}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\pi := e} : \texttt{unit} \Rightarrow \Gamma'[\pi \mapsto \tau^{\text{SI}}]}$$

We proceed based on whether or not e is a value. If it is not, we can decompose our expression into the evaluation context $p := \square$ and redex e. Then, by applying our induction hypothesis to e, we know either that e steps to some e' or is an abort! expression. In the former case, this satisfies our requirement since we can plug e' back into our evaluation context. In the latter case, we can step with E-EvalCtxAbort.

If e is a value, we would like to step with:

$$\textsc{E-Assign}\quad \frac{\sigma \vdash p \Downarrow \mathcal{R} \mapsto \mathcal{V}[\_] \qquad \mathcal{R} = \mathcal{R}^{\square}[x]}{\Sigma \vdash (\sigma;\ \boxed{p := v}) \rightarrow (\sigma[x \mapsto \mathcal{V}[v]];\ \boxed{(\,)})}$$

Since e is a value, we can apply Lemma E.15 to get $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma'$. Applying Lemma E.27, then gives us $\Sigma \vdash \sigma : \Gamma'$.

Then, we can apply Lemma E.4 to $\Delta;\ \omega;\ \Gamma' \vdash_p \{\ \overline{\ell}\ \} \Rightarrow$, $\Delta;\ \Gamma' \vdash_\omega p : \tau^{\text{XI}}$, and $\Sigma \vdash \sigma : \Gamma'$ to get $\sigma \vdash p \Downarrow \mathcal{R} \mapsto \mathcal{V}[\_]$.

Case T-ForArray:

From premise:

$$\textsc{T-ForArray}\quad \frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1} : [\tau^{\text{SI}};\ n] \Rightarrow \Gamma_1 \qquad \forall r \in \text{free-regions}(\tau^{\text{SI}}).\ \Gamma_1 \vdash r\ \texttt{rnrb} \\ \Sigma;\ \Delta;\ \Gamma_1,\ x\ :\ \tau^{\text{SI}};\ \Theta \vdash \boxed{e_2} : \texttt{unit} \Rightarrow \Gamma_1,\ x\ :\ \tau^{\text{SD}}\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \}} : \texttt{unit} \Rightarrow \Gamma_1}$$

We proceed based on whether or not e_1 is a value. If it is not, we can decompose our expression into the evaluation context $\texttt{for}\ x\ \texttt{in}\ \square\ \{\ e_2\ \}$ and redex e_1. Then, by applying our induction hypothesis to e_1, we know either that e_1 steps to some e_1' or is an abort! expression. In the former case, this satisfies our requirement since we can plug e_1' back into our evaluation context. In the latter case, we can step with E-EvalCtxAbort.

If e_1 is a value, we would like to step with one of:

$$\textsc{E-ForArray}\quad \frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{for}\ x\ \texttt{in}\ [v_0,\ \dots,\ v_n]\ \{\ e\ \}}) \rightarrow (\sigma,\ x \mapsto v_0;\ \boxed{\texttt{shift}\ e;\ \texttt{for}\ x\ \texttt{in}\ [v_1,\ \dots,\ v_n]\ \{\ e\ \}})}$$

$$\textsc{E-ForEmptyArray}\quad \frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{for}\ x\ \texttt{in}\ []\ \{\ e\ \}}) \rightarrow (\sigma;\ \boxed{(\,)})}$$

Since e_1 is a value, then by Lemma E.1, we know that e_1 is of the form $[v_1,\ \dots,\ v_n]$. If $n > 0$, then we can step with E-ForArray, and if $n = 0$, then we can step with E-ForEmptyArray.

Case T-ForSlice:

From premise:

$$\textsc{T-ForSlice}\quad \frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1} : \&\rho\ \omega\ [\tau^{\text{SI}}] \Rightarrow \Gamma_1 \qquad \forall r \in \text{free-regions}(\&\rho\ \omega\ \tau^{\text{SI}}).\ \Gamma_1 \vdash r\ \texttt{rnrb} \\ \Sigma;\ \Delta;\ \Gamma_1,\ x\ :\ \&\rho\ \omega\ \tau^{\text{SI}};\ \Theta \vdash \boxed{e_2} : \texttt{unit} \Rightarrow \Gamma_1,\ x\ :\ \tau_1^{\text{SX}}\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \}} : \texttt{unit} \Rightarrow \Gamma_2}$$

We proceed based on whether or not e_1 is a value. If it is not, we can decompose our expression into the evaluation context $\texttt{for}\ x\ \texttt{in}\ \square\ \{\ e_2\ \}$ and redex e_1. Then, by applying our induction hypothesis to e_1, we know either that e_1 steps to some e_1' or is an abort! expression. In the former case, this satisfies our requirement since we can plug e_1' back into our evaluation context. In the latter case, we can step with E-EvalCtxAbort.

If e_1 is a value, we would like to step with one of:

E-ForSlice
$$\frac{\sigma \vdash \mathcal{R} \Downarrow \_ \mapsto \_[[v_1, \ldots, v_i, \ldots, v_j, \ldots, v_n]] \qquad i < j \qquad i' = i + 1}{\Sigma \vdash (\sigma;\ \boxed{\texttt{for}\ x\ \texttt{in}\ \texttt{ptr}\ \mathcal{R}[i..j]\ \{\ e\ \}}) \rightarrow (\sigma,\ x \mapsto \texttt{ptr}\ \mathcal{R}[i];\ \boxed{\texttt{shift}\ e;\ \texttt{for}\ x\ \texttt{in}\ \texttt{ptr}\ \mathcal{R}[i'..j]\ \{\ e\ \}})}$$

E-ForEmptySlice
$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{for}\ x\ \texttt{in}\ \texttt{ptr}\ \pi[n..n]\ \{\ e\ \}}) \rightarrow (\sigma;\ \boxed{(\,)})}$$

If e_1 is a value, then by Lemma E.1, we know that e_1 is of the form $\texttt{ptr}\ \mathcal{R}[n_1..n_2]$. Further, by inversion of T-Pointer for the typing derivation of e_1, we get $\Sigma;\ \Gamma \vdash \mathcal{R}[i..j] : [\tau^{\text{SI}}]$. By inversion of WF-RefSliceArray or WF-RefSliceSlice (one of which must apply since the referent ends in a slice), we know that $i \leq j$. If $i < j$, we step with E-ForSlice and if $i = j$, we step with E-ForEmptySlice.

Case T-Closure:

From premise:

T-Closure
$$\frac{\begin{array}{c}\overline{\text{free-vars}(e) \setminus \overline{x} = \overline{x_f}} \qquad \overline{\text{free-nc-vars}_\Gamma(e) \setminus \overline{x} = \overline{x_{nc}}} \\ \overline{r} = \overline{\text{free-regions}(\Gamma(x_f))},\ (\text{free-regions}(e) \setminus (\text{free-regions}(\tau^{\text{SI}}), \text{free-regions}(\tau_r^{\text{SI}}))) \\ \mathcal{F}_c = \overline{r \mapsto \Gamma(r)},\ \overline{x_f\ :\ \Gamma(x_f)} \qquad \forall r_p \in \bigcup_{i=1}^{n} \text{free-regions}(\tau_i^{\text{SI}}) \cup \text{free-regions}(\tau_r^{\text{SI}}).\ \Gamma(r_p) = \emptyset \\ \Sigma;\ \Delta;\ \Gamma[\overline{x_{nc} \mapsto \Gamma(x_{nc})^\dagger}] \natural \mathcal{F}_c,\ x_1\ :\ \tau_1^{\text{SI}}, \ldots, x_n\ :\ \tau_n^{\text{SI}};\ \Theta \vdash \boxed{e} : \tau_r^{\text{SI}} \Rightarrow \Gamma' \natural \mathcal{F}\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{|x_1 : \tau_1^{\text{SI}}, \ldots, x_n : \tau_n^{\text{SI}}| \rightarrow \tau_r^{\text{SI}}\ \{\ e\ \}} : (\tau_1^{\text{SI}}, \ldots, \tau_n^{\text{SI}}) \xrightarrow{\mathcal{F}_c} \tau_r^{\text{SI}} \Rightarrow \Gamma'}$$

We want to step with:

E-Closure
$$\frac{\overline{x_f} = \text{free-vars}(e) \qquad \overline{x_{nc}} = \text{free-nc-vars}_\sigma(e) \qquad \varsigma_c = \sigma\ |_{\overline{x_f}}}{\Sigma \vdash (\sigma;\ \boxed{|x_1 : \tau_1^{\text{S}}, \ldots, x_n : \tau_n^{\text{S}}| \rightarrow \tau_r^{\text{S}}\ \{\ e\ \}}) \rightarrow (\sigma[\overline{x_{nc} \mapsto \texttt{dead}}];\ \boxed{\langle \varsigma_c,\ |x_1 : \tau_1^{\text{S}}, \ldots, x_n : \tau_n^{\text{S}}| \rightarrow \tau_r^{\text{S}}\ \{\ e\ \}\ \rangle})}$$

Since free-vars$(\cdot)$ and free-nc-vars$_\sigma(\cdot)$ are total, we can always step with E-Closure.

Case T-AppClosure:

From premise:

T-AppClosure
$$\frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_f} : \forall\texttt{<>}(\tau_1^{\text{SI}}, \ldots, \tau_n^{\text{SI}}) \xrightarrow{\Phi_c} \tau_f^{\text{SI}} \Rightarrow \Gamma_0 \\ \forall i \in \{\ 1 \ldots n\ \}.\ \Sigma;\ \Delta;\ \Gamma_{i-1};\ \Theta,\ \tau_1^{\text{SI}} \ldots \tau_{i-1}^{\text{SI}} \vdash \boxed{e_i} : \tau_{i'}^{\text{SI}} \Rightarrow \Gamma_i \qquad \Delta;\ \Gamma_i;\ \Theta \vdash^{\boxplus} \tau_{i'}^{\text{SI}} \rightsquigarrow \tau_i^{\text{SI}} \dashv \Gamma_i' \\ \forall i \in \{\ 1 \ldots n\ \}.\ \forall r \in \text{free-regions}(\tau_i^{\text{SI}}).\ \Gamma_n' \vdash r\ \texttt{rnrb}\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_f(e_1, \ldots, e_n)} : \tau_f^{\text{SI}} \Rightarrow \Gamma_n'}$$

We proceed based on whether or not e_f is a value. If it is not, we can decompose our expression into the evaluation context $\square(e_1, \ldots, e_n)$ and redex e_f. Then, by applying our induction hypothesis to e_f, we know either that e_f steps to some e_f' or is an `abort!` expression. In the former case, this satisfies our requirement since we can plug e_f' back into our evaluation context. In the latter case, we can step with E-EvalCtxAbort.

Next, we'll proceed based on whether or not each expression e_i is a value. If any of them are not, we can decompose our expression into the evaluation context $v_f(v_1\,,\ \ldots\,,\ v_m\,,\ \square,\ e_1\,,\ \ldots\,,\ e_{n'})$ and redex e_i. Then, by applying our induction hypothesis to e_i, we know either that e_i steps to some e_i' or is an `abort!` expression. In the former case, this satisfies our requirement since we can plug e_i' back into our evaluation context. In the latter case, we can step with E-EvalCtxAbort.

If e_f is a value and every e_i is a value, we would like to step with one of:

E-AppClosure

$$\frac{v_f = \langle \varsigma_c\,,\ |x_1 : \tau_1^{\text{S}}\,,\ \ldots\,,\ x_n : \tau_n^{\text{S}}|\ \rightarrow\ \tau_r^{\text{S}}\ \{\ e\ \}\ \rangle}{\Sigma \vdash (\sigma;\ \boxed{v_f(v_1\,,\ \ldots\,,\ v_n)}\ ) \rightarrow (\sigma \natural \varsigma_c\,,\ x_1 \mapsto v_1\,,\ \ldots\,,\ x_n \mapsto v_n;\ \boxed{\texttt{framed}\ e}\ )}$$

Since e_f is a value, then by Lemma E.1, we know that it has the form $\langle \sigma_c\,,\ |x_1 : \tau_1^{\text{SI}}\,,\ \ldots\,,\ x_n : \tau_n^{\text{SI}}|\ \rightarrow\ \tau_r^{\text{SI}}\ \{\ e\ \}\ \rangle$. Then, since all of the e_i are values, then we can step using E-AppClosure.

Case T-AppFunction:

From premise:

T-AppFunction

$$\frac{\begin{array}{c}\overline{\Sigma;\ \Delta;\ \Gamma \vdash \Phi} \qquad \overline{\Delta;\ \Gamma \vdash \rho} \qquad \overline{\Sigma;\ \Delta;\ \Gamma \vdash \tau^{\text{SI}}} \qquad \delta = \cdot\ \overline{[^{\Phi}/_{\varphi}]}\ \overline{[^{\rho}/_{\varrho}]}\ \overline{[^{\tau^{\text{SI}}}/_{\alpha}]} \\ \Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_f} : \forall{<}\overline{\varphi}\,,\ \overline{\varrho}\,,\ \overline{\alpha}{>}(\tau_1^{\text{SI}}\,,\ \ldots\,,\ \tau_n^{\text{SI}})\ \rightarrow\ \tau_f^{\text{SI}}\ \texttt{where}\ \overline{\varrho_1 : \varrho_2} \Rightarrow \Gamma_0 \\ \forall i \in \{\ 1\ \ldots\ n\ \}.\ \Sigma;\ \Delta;\ \Gamma_{i-1};\ \Theta,\ \delta(\tau_1^{\text{SI}})\ \ldots\ \delta(\tau_{i-1}^{\text{SI}}) \vdash \boxed{e_i} : \delta(\tau_i^{\text{SI}}) \Rightarrow \Gamma_i \\ \forall i \in \{\ 1\ \ldots\ n\ \}.\ \forall r \in \text{free-regions}(\tau_i^{\text{SI}}).\ \Gamma_n' \vdash r\ \texttt{rnrb} \qquad \overline{\Delta;\ \Gamma_n;\ \Theta \vdash \varrho_2\ \overline{[^{\rho}/_{\varrho}]} :> \varrho_1\ \overline{[^{\rho}/_{\varrho}]} \dashv \Gamma_b}\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_f{::}{<}\overline{\Phi},\ \overline{\rho},\ \overline{\tau^{\text{SI}}}{>}(e_1\,,\ \ldots\,,\ e_n)} : \delta(\tau_f^{\text{SI}}) \Rightarrow \Gamma_b}$$

We proceed based on whether or not e_f is a value. If it is not, we can decompose our expression into the evaluation context $\square{::}{<}\overline{\Phi},\ \overline{\rho},\ \overline{\tau^{\text{SI}}}{>}(e_1\,,\ \ldots\,,\ e_n)$ and redex e_f. Then, by applying our induction hypothesis to e_f, we know either that e_f steps to some e_f' or is an `abort!` expression. In the former case, this satisfies our requirement since we can plug e_f' back into our evaluation context. In the latter case, we can step with E-EvalCtxAbort.

Next, we'll proceed based on whether or not each expression e_i is a value. If any of them are not, we can decompose our expression into the evaluation context $v_f{::}{<}\overline{\Phi},\ \overline{\rho},\ \overline{\tau^{\text{SI}}}{>}(v_1\,,\ \ldots\,,\ v_m\,,\ \square,\ e_1\,,\ \ldots\,,\ e_{n'})$ and redex e_i. Then, by applying our induction hypothesis to e_i, we know either that e_i steps to some e_i' or is an `abort!` expression. In the former case, this satisfies our requirement since we can plug e_i' back into our evaluation context. In the latter case, we can step with E-EvalCtxAbort.

If e_f is a value and every e_i is a value, we would like to step with one of:

E-AppFunction

$$\frac{\Sigma(f) = \texttt{fn}\ f{<}\overline{\varphi}\,,\ \overline{\varrho}\,,\ \overline{\alpha}{>}(x_1 : \tau_1^{\text{S}}\,,\ \ldots\,,\ x_n : \tau_n^{\text{S}})\ \rightarrow\ \tau_r^{\text{S}}\ \texttt{where}\ \overline{\varrho : \varrho'}\ \{\ e\ \}}{\Sigma \vdash (\sigma;\ \boxed{f{::}{<}\overline{\Phi},\ \overline{r'},\ \overline{\tau^{\text{S}}}{>}(v_1\,,\ \ldots\,,\ v_n)}\ ) \rightarrow (\sigma \natural x_1 \mapsto v_1\,,\ \ldots\,,\ x_n \mapsto v_n;\ \boxed{\texttt{framed}\ e[^{\overline{\Phi}}/_{\overline{\varphi}}][^{\overline{r'}}/_{\overline{\varrho}}][^{\overline{\tau^{\text{S}}}}/_{\overline{\alpha}}]}\ )}$$

Since e_f is a value, then by Lemma E.1, we know that it has the form f. Then, since all of the e_i are values, then we can step using E-AppFunction.

Case T-Unit:

From premise:

$$\frac{\text{T-Unit}}{\Sigma;\,\Delta;\,\Gamma;\,\Theta \vdash \boxed{(\,)} : \texttt{unit} \Rightarrow \Gamma}$$

By inspection of the value grammar, we know that ( ) is already a value.

Case T-u32:

From premise:

$$\frac{\text{T-u32}}{\Sigma;\,\Delta;\,\Gamma;\,\Theta \vdash \boxed{n} : \texttt{u32} \Rightarrow \Gamma}$$

By inspection of the value grammar, we know that n is already a value.

Case T-True:

From premise:

$$\frac{\text{T-True}}{\Sigma;\,\Delta;\,\Gamma;\,\Theta \vdash \boxed{\texttt{true}} : \texttt{bool} \Rightarrow \Gamma}$$

By inspection of the value grammar, we know that `true` is already a value.

Case T-False:

From premise:

$$\frac{\text{T-False}}{\Sigma;\,\Delta;\,\Gamma;\,\Theta \vdash \boxed{\texttt{false}} : \texttt{bool} \Rightarrow \Gamma}$$

By inspection of the value grammar, we know that `false` is already a value.

Case T-Tuple:

From premise:

$$\text{T-Tuple}\quad \frac{\forall i \in \{\,1 \ldots n\,\}.\ \Sigma;\,\Delta;\,\Gamma_{i-1};\,\Theta,\,\tau_1^{\text{SI}},\,\ldots,\,\tau_{i-1}^{\text{SI}} \vdash \boxed{e_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_i}{\Sigma;\,\Delta;\,\Gamma_0;\,\Theta \vdash \boxed{(e_1,\,\ldots,\,e_n)} : (\tau_1^{\text{SI}},\,\ldots,\,\tau_n^{\text{SI}}) \Rightarrow \Gamma_n}$$

We'll proceed based on whether or not each expression e_i is a value. If any of them are not, we can decompose our expression into the evaluation context $(v_1,\,\ldots,\,v_m,\,\square,\,e_1,\,\ldots,\,e_{n'})$ and redex e_i. Then, by applying our induction hypothesis to e_i, we know either that e_i steps to some e_i' or to an `abort`! expression. In either case, this satisfies our requirement, since we can plug e_i' back into our evaluation context.

If every expression e_i is a value, then the whole expression is a value by the definition of values.

Case T-Array:

From premise:

$$\textsc{T-Array}\quad \frac{\forall i \in \{\, 1 \,\ldots\, n \,\}.\ \Sigma;\, \Delta;\, \Gamma_{i-1};\, \Theta,\, \tau_1^{\text{SI}},\, \ldots,\, \tau_{i-1}^{\text{SI}} \vdash \boxed{e_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_i}{\Sigma;\, \Delta;\, \Gamma;\, \Theta \vdash \boxed{[e_1,\, \ldots,\, e_n]} : [\tau^{\text{SI}};\, n] \Rightarrow \Gamma_n}$$

We'll proceed based on whether or not each expression e_i is a value. If any of them are not, we can decompose our expression into the evaluation context $[v_1,\, \ldots,\, v_m,\, \Box,\, e_1,\, \ldots,\, e_{n'}]$ and redex e_i. Then, by applying our induction hypothesis to e_i, we know either that e_i steps to some e_i' or to an `abort!` expression. In either case, this satisfies our requirement, since we can plug e_i' back into our evaluation context.

If every expression e_i is a value, then the whole expression is a value by the definition of values.

Case T-Abort:

From premise:

$$\textsc{T-Abort}\quad \frac{}{\Sigma;\, \Delta;\, \Gamma;\, \Theta \vdash \boxed{\texttt{abort!(str)}} : \tau^{\text{SX}} \Rightarrow \Gamma}$$

By definition, `abort!`(...) is an `abort!` expression.

Case T-Framed:

From premise:

$$\textsc{T-Framed}\quad \frac{\Sigma;\, \Delta;\, \Gamma;\, \Theta \vdash \boxed{e} : \tau^{\text{SI}} \Rightarrow \Gamma' \natural \mathcal{F}'}{\Sigma;\, \Delta;\, \Gamma;\, \Theta \vdash \boxed{\texttt{framed}\ e} : \tau^{\text{SI}} \Rightarrow \Gamma'}$$

We proceed based on whether or not e is a value. If it is not, we can decompose our expression into the evaluation context `framed` $\Box$ and redex e. Then, by applying our induction hypothesis to e, we know either that e steps to some e' or to an `abort!` expression. In either case, this satisfies our requirement, since we can plug e' back into our evaluation context.

If e is a value, then we would like to step with:

$$\textsc{E-Framed}\quad \frac{}{\Sigma \vdash (\sigma \natural \varsigma;\, \boxed{\texttt{framed}\ v}) \rightarrow (\sigma;\, \boxed{v})}$$

In order to do so, we need to know $x \in \text{dom}(\sigma)$. Fortunately, we know from our assumption that $\Sigma \vdash \sigma : \Gamma$ (via WF-Stack). The premise of WF-Stack tells us that $\text{dom}(\sigma) = \text{dom}(\Gamma)$, and thus the $x \in \text{dom}(\Gamma)$ from the premise of T-Framed is sufficient to tell us that $x \in \text{dom}(\sigma)$. Thus, we can step with E-Framed.

Case T-Pointer:

From premise:

$$\textsc{T-Pointer}\quad \frac{\Sigma;\, \Gamma \vdash \mathcal{R}^{\Box}[\pi] : \tau^{\text{XI}} \qquad {}^{\omega}\pi \in \Gamma(r)}{\Sigma;\, \Delta;\, \Gamma;\, \Theta \vdash \boxed{\texttt{ptr}\ \mathcal{R}^{\Box}[\pi]} : \&r\ \omega\ \tau^{\text{XI}} \Rightarrow \Gamma}$$

By inspection of the value grammar, we know that `ptr` π is already a value.

Case T-ClosureValue:

From premise:

T-ClosureValue
$$\frac{\begin{array}{c}\text{free-vars}(e) \setminus \overline{x} = \overline{x_f} = \text{dom}(\mathcal{F}_c)|_{\text{VAR}} \\ \overline{r} = \overline{\text{free-regions}(\Gamma(x_f))},\ (\text{free-regions}(e) \setminus \overline{(\text{free-regions}(\tau^{\text{SI}})}, \text{free-regions}(\tau_r^{\text{SI}}))) = \text{dom}(\mathcal{F}_c)|_{\text{RGN}} \\ \Sigma;\ \Gamma \vdash \varsigma_c : \mathcal{F}_c \qquad \Sigma;\ \Delta;\ \Gamma \natural \mathcal{F}_c,\ x_1\ :\ \tau_1^{\text{SI}},\ \ldots,\ x_n\ :\ \tau_n^{\text{SI}};\ \Theta \vdash \boxed{e} : \tau_r^{\text{SI}} \Rightarrow \Gamma' \natural \mathcal{F}\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\langle \varsigma_c,\ \ |x_1 : \tau_1^{\text{SI}},\ \ldots,\ x_n : \tau_n^{\text{SI}}|\ \rightarrow\ \tau_r^{\text{SI}}\ \{\ e\ \}\ \rangle} : (\tau_1^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}}) \overset{\mathcal{F}_c}{\rightarrow} \tau_r^{\text{SI}} \Rightarrow \Gamma}$$

By inspection of the value grammar, we know that $\langle \sigma,\ \ |x_1 : \tau_1^{\text{SI}},\ \ldots,\ x_n : \tau_n^{\text{SI}}|\ \rightarrow\ \tau_r^{\text{SI}}\ \{\ e\ \}\ \rangle$ is already a value.

Case T-Dead:

From premise:

T-Dead
$$\frac{}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau^{\text{SI}^\dagger} \Rightarrow \Gamma}$$

The type $\tau^{\text{SI}^\dagger}$ is not in the grammar of τ^{SI}. Thus, we have a contradiction.

Case T-Drop:

From premise:

T-Drop
$$\frac{\Gamma(\pi) = \tau_\pi^{\text{SI}} \qquad \Sigma;\ \Delta;\ \Gamma[\pi \mapsto \tau_\pi^{\text{SI}^\dagger}];\ \Theta \vdash \boxed{e} : \tau^{\text{SX}} \Rightarrow \Gamma_f}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau^{\text{SX}} \Rightarrow \Gamma_f}$$

By R-Env, we have that $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma[\pi \mapsto \tau_\pi^{\text{SI}^\dagger}]$. Then, applying Lemma E.27 with $\Sigma \vdash \sigma : \Gamma$ (from our premise) gives us $\Sigma \vdash \sigma : \Gamma[\pi \mapsto \tau_\pi^{\text{SI}^\dagger}]$. We can then apply our induction hypothesis to this and $\Sigma;\ \bullet;\ \Gamma[\pi \mapsto \tau_\pi^{\text{SI}^\dagger}];\ \Theta \vdash \boxed{e} : \tau^{\text{SX}} \Rightarrow \Gamma_f$ to reach our goal.

Case T-Left:

From premise:

T-Left
$$\frac{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau_1^{\text{SI}} \Rightarrow \Gamma'}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{Left::<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}\texttt{>}(e)} : \texttt{Either<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}\texttt{>} \Rightarrow \Gamma'}$$

We proceed based on whether or not e is a value. If it is not, we can decompose our expression into the evaluation context `Left::<`τ_1^{SI}, τ_2^{SI}`>`$(\square)$ and redex e. Then, by applying our induction hypothesis to e, we know either that e steps to some e' or to an `abort`! expression. In either case, this satisfies our requirement, since we can plug e' back into our evaluation context.

If e is a value, then we know that the whole expression `Left::<`τ_1^{SI}, τ_2^{SI}`>`(v) is a value and thus we are done.

Case T-Right:

From premise:

T-Right
$$\frac{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau_2^{\text{SI}} \Rightarrow \Gamma'}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{Right::<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}\texttt{>}(e)} : \texttt{Either<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}\texttt{>} \Rightarrow \Gamma'}$$

We proceed based on whether or not e is a value. If it is not, we can decompose our expression into the evaluation context $\texttt{Right::<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}\texttt{>}(\square)$ and redex e. Then, by applying our induction hypothesis to e, we know either that e steps to some e' or to an `abort!` expression. In either case, this satisfies our requirement, since we can plug e' back into our evaluation context.

If e is a value, then we know that the whole expression $\texttt{Right::<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}\texttt{>}(v)$ is a value and thus we are done.

Case T-Match:

From premise:

T-Match
$$\frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \texttt{Either<}\tau_l^{\text{SI}},\ \tau_r^{\text{SI}}\texttt{>} \Rightarrow \Gamma' \qquad \forall r \in \text{free-regions}(\texttt{Either<}\tau_l^{\text{SI}},\ \tau_r^{\text{SI}}\texttt{>}).\ \Gamma' \vdash r\ \texttt{rnrb} \\ \Sigma;\ \Delta;\ \Gamma',\ x_1 : \tau_l^{\text{SI}};\ \Theta \vdash \boxed{e_1} : \tau_1^{\text{SI}} \Rightarrow \Gamma_1,\ x_1 : \tau_l^{\text{SD}} \\ \Sigma;\ \Delta;\ \Gamma',\ x_2 : \tau_r^{\text{SI}};\ \Theta \vdash \boxed{e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2,\ x_2 : \tau_r^{\text{SD}} \qquad \tau^{\text{SI}} = \tau_1^{\text{SI}} \lor \tau^{\text{SI}} = \tau_2^{\text{SI}} \\ \Delta;\ \Gamma_1;\ \Theta \vdash^{+} \tau_1^{\text{SI}} \rightsquigarrow \tau^{\text{SI}} \dashv \Gamma_1' \qquad \Delta;\ \Gamma_2;\ \Theta \vdash^{+} \tau_2^{\text{SI}} \rightsquigarrow \tau^{\text{SI}} \dashv \Gamma_2' \qquad \Gamma_1' \Cup \Gamma_2' = \Gamma'\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{match}\ e\ \{\ \texttt{Left}(x_1) \Rightarrow e_1,\ \texttt{Right}(x_2) \Rightarrow e_2\ \}} : \tau^{\text{SI}} \Rightarrow \Gamma'}$$

We proceed based on whether or not e is a value. If it is not, we can decompose our expression into the evaluation context $\texttt{match}\,\square\ \{\texttt{Left}(x_1) \Rightarrow e_1,\ \texttt{Right}(x_2) \Rightarrow e_2\}$ and redex e. Then, by applying our induction hypothesis to e, we know either that e steps to some e' or to an `abort!` expression. In either case, this satisfies our requirement, since we can plug e' back into our evaluation context.

If e is a value, then we would like to step with either:

E-MatchLeft
$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{match Left::<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}\texttt{>}(v)\ \{\ \texttt{Left}(x_1) \Rightarrow e_1,\ \texttt{Right}(x_2) \Rightarrow e_2\ \}}) \rightarrow (\sigma,\ x_1 \mapsto v;\ \boxed{\texttt{shift}\ e_1})}$$

E-MatchRight
$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{match Right::<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}\texttt{>}(v)\ \{\ \texttt{Left}(x_1) \Rightarrow e_1,\ \texttt{Right}(x_2) \Rightarrow e_2\ \}}) \rightarrow (\sigma,\ x_2 \mapsto v;\ \boxed{\texttt{shift}\ e_2})}$$

Since e is a value, applying Lemma E.1 tells us that e is either of the form $\texttt{Left::<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}\texttt{>}(v)$ or $\texttt{Right::<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}\texttt{>}(v)$. In the former case, we can step with E-MatchLeft and in the latter case, we can step with E-MatchRight.

E.15 Preservation

Lemma E.69 (Preservation). *If* $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau_1^{SI} \Rightarrow \Gamma_f$ *and* $\Sigma \vdash \sigma : \Gamma$ *and* $\Sigma;\ \Gamma \vdash \overline{v} : \Theta$ *and* $\Sigma \vdash (\sigma;\ \boxed{e}) \longrightarrow (\sigma';\ \boxed{e'})$, *then there exists* Γ_i *such that* $\Sigma \vdash \sigma' : \Gamma_i$ *and* $\Sigma;\ \Gamma_i \vdash \overline{v} : \Theta$ *and* $\Sigma;\ \bullet;\ \Gamma_i;\ \Theta \vdash \boxed{e'} : \tau_2^{SI} \Rightarrow \Gamma_f'$ *and* $\bullet;\ \Gamma_f';\ \Theta \vdash^{+} \tau_2^{SI} \rightsquigarrow \tau_1^{SI} \dashv \Gamma_s$ *and there exists* Γ_o *such that* $\Gamma_f = \Gamma_s \Cup \Gamma_o$.

Proof. We proceed by induction on the derivation $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau \Rightarrow \Gamma_f$

Case T-Move:

From premise:

$$\textsc{T-Move}\quad \frac{\Delta;\ \Gamma;\ \Theta \vdash_{\mathsf{uniq}} \pi \Rightarrow \{\ ^{\mathsf{uniq}}\pi\ \} \qquad \Gamma(\pi) = \tau^{\mathrm{SI}} \qquad \mathsf{noncopyable}_{\Sigma}\ \tau^{\mathrm{SI}}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\pi} : \tau^{\mathrm{SI}} \Rightarrow \Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}]}$$

Since $e = \pi$, by inspection of the reduction rules, we know that e steps with the following rule:

$$\textsc{E-Move}\quad \frac{\sigma \vdash \pi \Downarrow \pi \mapsto \_[v]}{\Sigma \vdash (\sigma;\ \boxed{\pi}) \longrightarrow (\sigma[\pi \mapsto \mathsf{dead}];\ \boxed{v})}$$

We then pick Γ_i to be $\boxed{\Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}]}$, and need to show:

$\Sigma \vdash \sigma[\pi \mapsto \mathsf{dead}] : \Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}]$	Applying Lemma E.27 to $\Sigma;\ \bullet \vdash \Gamma \triangleright \Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}]$ (immediate by R-Env) and $\Sigma \vdash \sigma : \Gamma$ (from premise) gives us $\Sigma \vdash \sigma : \Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}]$. Then, since we know $\Sigma;\ \bullet;\ \Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}];\ \Theta \vdash \boxed{\mathsf{dead}} : \tau^{\mathrm{SI}^{\dagger}} \Rightarrow \Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}]$ (by T-Dead), we can conclude $\Sigma \vdash \sigma[\pi \mapsto \mathsf{dead}] : \Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}]$.
$\Sigma;\ \Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}] \vdash \overline{v} : \Theta$	Inverting WF-Temporaries gives us $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma;\ \tau_1^{\mathrm{SI}}, \ldots, \tau_{i-1}^{\mathrm{SI}} \vdash \boxed{v_i} : \tau_i^{\mathrm{SI}} \Rightarrow \Gamma$. By R-Env, we have that $\Sigma;\ \bullet \vdash \Gamma \triangleright \Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}]$. Thus, we can apply Lemma E.25 to each of the typing judgments and then apply WF-Temporaries to get $\Sigma;\ \Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}] \vdash \overline{v} : \Theta$.
$\Sigma;\ \bullet;\ \Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}];\ \Theta \vdash \boxed{v} : \tau^{\mathrm{SI}} \Rightarrow \Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}]$	Applying Lemma E.4 to $\bullet;\ \mathsf{uniq};\ \Gamma \vdash_{\pi} \{\ ^{\mathsf{uniq}}\pi\ \} \Rightarrow$, $\bullet;\ \Gamma \vdash_{\mathsf{uniq}} \pi : \tau^{\mathrm{SI}}$ (immediate by TC-Place with $\Gamma(\pi) = \tau^{\mathrm{SI}}$), and $\Sigma \vdash \sigma : \Gamma$ gives us $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau^{\mathrm{SI}} \Rightarrow \Gamma$. Then, by applying Lemma E.25 with $\Sigma;\ \bullet \vdash \Gamma \triangleright \Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}]$ (immediate by R-Env), we can conclude $\Sigma;\ \bullet;\ \Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}];\ \Theta \vdash \boxed{v} : \tau^{\mathrm{SI}} \Rightarrow \Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}]$.
$\bullet;\ \Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}];\ \Theta \vdash^{+} \tau^{\mathrm{SI}} \rightsquigarrow \tau^{\mathrm{SI}} \dashv \Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}]$	Immediate by RR-Refl.
$\exists \Gamma_o.\Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}] \Cup \Gamma_o = \Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}]$	$\Gamma_o = \Gamma[\pi \mapsto \tau^{\mathrm{SI}^{\dagger}}]$

Case T-Copy:

From premise:

$$\textsc{T-Copy}\quad \frac{\Delta;\ \Gamma;\ \Theta \vdash_{\mathsf{shrd}} p \Rightarrow \{\,\overline{\ell}\,\} \qquad \Delta;\ \Gamma \vdash_{\mathsf{shrd}} p : \tau^{\text{SI}} \qquad \mathsf{copyable}_{\Sigma}\ \tau^{\text{SI}}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{p} : \tau^{\text{SI}} \Rightarrow \Gamma}$$

Since $e = p$, by inspection of the reduction rules, we know that e steps with the following rule:

$$\textsc{E-Copy}\quad \frac{\sigma \vdash p \Downarrow \_ \mapsto \_[v]}{\Sigma \vdash (\sigma;\ \boxed{p}) \rightarrow (\sigma;\ \boxed{v})}$$

We then pick Γ_i to be $\boxed{\Gamma}$, and need to show:

$\Sigma \vdash \sigma : \Gamma$	Immediate from our premise.
$\Sigma;\ \Gamma \vdash \overline{v} : \Theta$	Immediate from our premise.
$\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau^{\text{SI}} \Rightarrow \Gamma$	Applying Lemma E.4 to $\bullet;\ \mathsf{shrd};\ \Gamma \vdash_p \{\,\overline{\ell}\,\} \Rightarrow$, $\bullet;\ \Gamma \vdash_{\mathsf{shrd}} p : \tau^{\text{SI}}$, and $\Sigma \vdash \sigma : \Gamma$ gives us $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau^{\text{SI}} \Rightarrow \Gamma$.
$\bullet;\ \Gamma;\ \Theta \vdash^{+} \tau^{\text{SI}} \rightsquigarrow \tau^{\text{SI}} \dashv \Gamma$	Immediate by RR-Refl.
$\exists \Gamma_o . \Gamma \uplus \Gamma_o = \Gamma$	$\Gamma_o = \Gamma$

Case T-Borrow:

From premise:

$$\textsc{T-Borrow}\quad \frac{\Gamma(r) = \emptyset \qquad \Gamma;\ \Theta \vdash r\ \mathsf{rnic} \qquad \Delta;\ \Gamma;\ \Theta \vdash_{\omega} p \Rightarrow \{\,\overline{\ell}\,\} \qquad \Delta;\ \Gamma \vdash_{\omega} p : \tau^{\text{XI}}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\&r\ \omega\ p} : \&r\ \omega\ \tau^{\text{XI}} \Rightarrow \Gamma[r \mapsto \{\,\overline{\ell}\,\}]}$$

Since $e = \&\rho\ \omega\ p$, by inspection of the reduction rules, we know that e steps with the following rule:

$$\textsc{E-Borrow}\quad \frac{\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[\_]}{\Sigma \vdash (\sigma;\ \boxed{\&r\ \omega\ p}) \rightarrow (\sigma;\ \boxed{\mathsf{ptr}\ \mathcal{R}})}$$

We then pick Γ_i to be $\boxed{\Gamma[r \mapsto \{\,\overline{\ell}\,\}]}$, and need to show:

$\Sigma \vdash \sigma : \Gamma[r \mapsto \{\overline{\ell}\}]$	Apply Lemma E.63 to $\Sigma \vdash \sigma : \Gamma$ (from our premise) and $\vdash \Sigma; \bullet; \Gamma[r \mapsto \{\overline{\ell}\}]; \Theta$ (from our premise) gives us $\Sigma \vdash \sigma : \Gamma[r \mapsto \{\overline{\ell}\}]$.
$\Sigma;\ \Gamma[r \mapsto \{\overline{\ell}\}] \vdash \overline{v} : \Theta$	Inverting WF-Temporaries gives us $\forall i \in 1 \ldots n.\ \Sigma; \bullet; \Gamma; \tau^{\text{SI}}_1, \ldots, \tau^{\text{SI}}_{i-1} \vdash \boxed{v_i} : \tau^{\text{SI}}_i \Rightarrow \Gamma$. We can then apply Lemma E.62 to each of the typing judgments along with $\bullet; \Gamma; \Theta \vdash_\omega p \Rightarrow \{\overline{\ell}\}$ and $\Gamma; \Theta \vdash r$ rnic and $\Gamma(r) = \emptyset$ (all from the premise of T-Borrow) to get $\forall i \in 1 \ldots n.\ \Sigma; \bullet; \Gamma[r \mapsto \{\overline{\ell}\}]; \tau^{\text{SI}}_1, \ldots, \tau^{\text{SI}}_{i-1} \vdash \boxed{v_i} : \tau^{\text{SI}}_i \Rightarrow \Gamma[r \mapsto \{\overline{\ell}\}]$. We can then apply WF-Temporaries to get $\Sigma;\ \Gamma[r \mapsto \{\overline{\ell}\}] \vdash \overline{v} : \Theta$.
$\Sigma; \bullet; \Gamma_i; \Theta \vdash \boxed{\texttt{ptr}\ \mathcal{R}} : \&r\ \omega\ \tau^{\text{XI}} \Rightarrow \Gamma_i$	Applying Lemma E.5 to $\Sigma \vdash \sigma : \Gamma$, and $\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[\_]$ gives us $\Sigma; \Gamma \vdash \mathcal{R}^\square[\pi] : \tau^{\text{XI}}$. Then, note that referent well-formedness does not depend on the contents of loan sets. This means we can also conclude $\Sigma; \Gamma[r \mapsto \{\overline{\ell}\}] \vdash \mathcal{R}^\square[\pi] : \tau^{\text{XI}}$. Applying Lemma E.6 to $\Sigma \vdash \sigma : \Gamma$, $\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[\_]$, and $\bullet; \omega; \Gamma \vdash_p \{\overline{\ell}\} \Rightarrow$ gives us $\mathcal{R} = \mathcal{R}^\square[\pi]$ and ${}^\omega\pi \in \{\overline{\ell}\}$. Finally, we can apply T-Pointer to the two facts above to get $\Sigma; \bullet; \Gamma[r \mapsto \{\overline{\ell}\}]; \Theta \vdash \boxed{\texttt{ptr}\ \mathcal{R}} : \&r\ \omega\ \tau^{\text{XI}} \Rightarrow \Gamma[r \mapsto \{\overline{\ell}\}]$.
$\bullet;\ \Gamma_i;\ \Theta \vdash^+ \&r\ \omega\ \tau^{\text{XI}} \rightsquigarrow \&r\ \omega\ \tau^{\text{XI}} \dashv \Gamma_i$	Immediate by RR-Refl.
$\exists \Gamma_o.\Gamma_i \Cup \Gamma_o = \Gamma_i$	$\Gamma_o = \Gamma_i$

Case T-BorrowIndex:

From premise:

T-BorrowIndex

$$\frac{\begin{array}{c}\Sigma;\Delta;\Gamma;\Theta \vdash \boxed{e} : \texttt{u32} \Rightarrow \Gamma' \qquad \Gamma'(r) = \emptyset \qquad \Gamma';\Theta \vdash r\ \texttt{rnic} \\ \Delta;\Gamma';\Theta \vdash_\omega p \Rightarrow \{\overline{\ell}\} \qquad \Delta;\Gamma' \vdash_\omega p : \tau^{\text{XI}} \\ \tau^{\text{XI}} = [\tau^{\text{SI}}; n] \lor \tau^{\text{XI}} = [\tau^{\text{SI}}]\end{array}}{\Sigma;\Delta;\Gamma;\Theta \vdash \boxed{\&r\ \omega\ p[e]} : \&r\ \omega\ \tau^{\text{SI}} \Rightarrow \Gamma'[r \mapsto \{\overline{\ell}\}]}$$

Since $e = \&\rho\ \omega\ p[e_i]$, by inspection of the reduction rules, we know that e steps with the following rule:

E-BorrowIndex

$$\frac{\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[[v_0, \ldots, v_n]] \qquad 0 \le n_i \le n}{\Sigma \vdash (\sigma; \boxed{\&r\ \omega\ p[n_i]}) \rightarrow (\sigma; \boxed{\texttt{ptr}\ \mathcal{R}[n_i]})}$$

E-BorrowIndexOOB

$$\frac{\sigma \vdash p \Downarrow \_ \mapsto \_[[v_0, \ldots, v_n]] \qquad n_i < 0 \lor n_i > n}{\Sigma \vdash (\sigma; \boxed{\&r\ \omega\ *p[n_i]}) \rightarrow (\sigma; \boxed{\texttt{abort!}(\text{“attempted to index out of bounds”})})}$$

Then, for each possible rule, we'll pick Γ_i separately. The cases proceed as follows:

For E-BorrowIndex, we pick Γ_i to be $\boxed{\Gamma'[r \mapsto \{\overline{\ell}\}]}$, and need to show:

$\Sigma \vdash \sigma : \Gamma'[r \mapsto \{\,\overline{\ell}\,\}]$	Applying Lemma E.15 to the typing derivation (from T-BorrowIndex) for e (which we know is a value from E-BorrowIndex) gives us $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma'$. Then, applying Lemma E.27 to $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma'$ and $\Sigma \vdash \sigma : \Gamma$ (from premise) gives us $\Sigma \vdash \sigma : \Gamma'$. Finally, applying Lemma E.63 to $\Sigma \vdash \sigma : \Gamma'$ gives us $\Sigma \vdash \sigma : \Gamma'[r \mapsto \{\,\overline{\ell}\,\}]$.
$\Sigma;\ \Gamma'[r \mapsto \{\,\overline{\ell}\,\}] \vdash \overline{v} : \Theta$	Inverting WF-Temporaries gives us $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma;\ \tau^{\text{SI}}_1, \ldots, \tau^{\text{SI}}_{i-1} \vdash \boxed{v_i} : \tau^{\text{SI}}_i \Rightarrow \Gamma$. Applying Lemma E.15 to the typing derivation (from T-BorrowIndex) for e (which we know is a value from E-BorrowIndex) gives us $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma'$. Thus, we can apply Lemma E.25 to each of the typing judgments to get $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma';\ \bullet \vdash \boxed{v_i} : \tau^{\text{SI}}_i \Rightarrow \Gamma'$. We can then apply Lemma E.62 to each of the typing judgments along with $\bullet;\ \Gamma';\ \Theta \vdash_{\omega} p \Rightarrow \{\,\overline{\ell}\,\}$ and $\Gamma';\ \Theta \vdash r$ `rnic` and $\Gamma'(r) = \emptyset$ (all from the premise of T-BorrowIndex) to get $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma'[r \mapsto \{\,\overline{\ell}\,\}];\ \bullet \vdash \boxed{v_i} : \tau^{\text{SI}}_i \Rightarrow \Gamma'[r \mapsto \{\,\overline{\ell}\,\}]$. We can then apply WF-Temporaries to get $\Sigma;\ \Gamma'[r \mapsto \{\,\overline{\ell}\,\}] \vdash \overline{v} : \Theta$.
$\Sigma;\ \bullet;\ \Gamma_i;\ \Theta \vdash \boxed{\texttt{ptr}\ \mathcal{R}[n_i]} : \&r\ \omega\ \tau^{\text{SI}} \Rightarrow \Gamma_i$	Applying Lemma E.15 to the typing derivation (from T-BorrowIndex) for e (which we know is a value from E-BorrowIndex) gives us $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma'$. Then, applying Lemma E.27 to $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma'$ and $\Sigma \vdash \sigma : \Gamma$ (from premise) gives us $\Sigma \vdash \sigma : \Gamma'$. Applying Lemma E.5 to $\Sigma \vdash \sigma : \Gamma'$, and $\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[[v_0, \ldots, v_n]]$ gives us $\Sigma;\ \Gamma' \vdash \mathcal{R}^{\square}[\pi] : \tau^{\text{XI}}$. Then, note that referent well-formedness does not depend on the contents of loan sets. This means we can also conclude $\Sigma;\ \Gamma'[r \mapsto \{\overline{\ell}\}] \vdash \mathcal{R}^{\square}[\pi] : \tau^{\text{XI}}$. We can then apply WF-RefIndexArray or WF-RefIndexSlice to get $\Sigma;\ \Gamma'[r \mapsto \{\,\overline{\ell}\,\}] \vdash \mathcal{R}^{\square}[\pi][n_i] : \tau^{\text{XI}}$. Applying Lemma E.6 to $\Sigma \vdash \sigma : \Gamma'$, $\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[[v_0, \ldots, v_n]]$, and $\bullet;\ \omega;\ \Gamma' \vdash_p \{\overline{\ell}\} \Rightarrow$ gives us $\mathcal{R} = \mathcal{R}[\pi]$ and ${}^{\omega}\pi \in \{\,\overline{\ell}\,\}$. Finally, we can apply T-Pointer to the two facts above to get $\Sigma;\ \bullet;\ \Gamma[r \mapsto \{\,\overline{\ell}\,\}];\ \Theta \vdash \boxed{\texttt{ptr}\ \mathcal{R}[n_i]} : \&r\ \omega\ \tau^{\text{XI}} \Rightarrow \Gamma[r \mapsto \{\,\overline{\ell}\,\}]$.
$\bullet;\ \Gamma_i;\ \Theta \vdash^{+} \&r\ \omega\ \tau^{\text{SI}} \rightsquigarrow \&r\ \omega\ \tau^{\text{SI}} \dashv \Gamma_i$	Immediate by RR-Refl.
$\exists \Gamma_o . \Gamma_i \Cup \Gamma_o = \Gamma_i$	$\Gamma_o = \Gamma_i$

For E-BorrowIndexOOB, we pick Γ_i to be $\boxed{\Gamma}$, and need to show:

$\Sigma \vdash \sigma : \Gamma$	Immediate from our premise.
$\Sigma;\ \Gamma \vdash \overline{v} : \Theta$	Immediate from our premise.
$\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{\texttt{abort!}(\ldots)} : \&r\ \omega\ \tau^{\text{SI}} \Rightarrow \Gamma$	An abort! expression is well-typed (at any type) via the rule T-Abort.
$\bullet;\ \Gamma;\ \Theta \vdash^{+} \&r\ \omega\ \tau^{\text{SI}} \rightsquigarrow \&r\ \omega\ \tau^{\text{SI}} \dashv \Gamma$	Immediate by RR-Refl.
$\exists \Gamma_o . \Gamma_i \uplus \Gamma_o = \Gamma_i$	$\Gamma_o = \Gamma_i$

Case T-BorrowSlice:

From premise:

$$\frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1} : \texttt{u32} \Rightarrow \Gamma_1 \qquad \Sigma;\ \Delta;\ \Gamma_1;\ \Theta \vdash \boxed{e_2} : \texttt{u32} \Rightarrow \Gamma_2 \\ \Gamma_2(r) = \emptyset \qquad \Gamma_2;\ \Theta \vdash r\ \texttt{rnic} \qquad \Delta;\ \Gamma_2;\ \Theta \vdash_{\omega} p \Rightarrow \{\ \overline{\ell}\ \} \qquad \Delta;\ \Gamma_2 \vdash_{\omega} p : [\tau^{\text{SI}}]\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\&r\ \omega\ p[e_1..e_2]} : \&r\ \omega\ [\tau^{\text{SI}}] \Rightarrow \Gamma_2[r \mapsto \{\ \overline{\ell}\ \}]}\ \text{T-BorrowSlice}$$

Since $e = \&r\ \omega\ p[e_1..e_2]$, by inspection of the reduction rules, we know that e steps with the following rule:

$$\frac{\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[[v_0\,,\ \ldots\ ,\ v_n]] \qquad 0 \leq n_1 \leq n_2 \leq n}{\Sigma \vdash (\sigma;\ \boxed{\&r\ \omega\ p[n_1..n_2]}) \rightarrow (\sigma;\ \boxed{\texttt{ptr}\ \mathcal{R}[n_1..n_2]})}\ \text{E-BorrowSlice}$$

$$\frac{\sigma \vdash p \Downarrow \_ \mapsto \_[[v_0\,,\ \ldots\ ,\ v_n]] \qquad n_1 < 0 \vee n_1 > n \vee n_2 < 0 \vee n_2 > n \vee n_1 > n_2}{\Sigma \vdash (\sigma;\ \boxed{\&r\ \omega\ p[n_1..n_2]}) \rightarrow (\sigma;\ \boxed{\texttt{abort!}(\text{“attempted to slice out of bounds”})})}\ \text{E-BorrowSliceOOB}$$

Then, for each possible rule, we'll pick Γ_i separately. The cases proceed as follows:

For E-BorrowSlice, we pick Γ_i to be $\boxed{\Gamma_2[r \mapsto \{\ \overline{\ell}\ \}]}$, and need to show:

$\Sigma \vdash \sigma : \Gamma_2[r \mapsto \{\,\overline{\ell}\,\}]$	Applying Lemma E.15 to the typing derivation (from T-BorrowSlice) for e_1 (which we know is a value from E-BorrowSlice) gives us $\Sigma;\ \bullet\ \vdash\ \Gamma \rhd \Gamma_1$. Then, applying Lemma E.15 to the typing derivation (from T-BorrowSlice) for e_2 (which we know is a value from E-BorrowSlice) gives us $\Sigma;\ \bullet \vdash \Gamma_1 \rhd \Gamma_2$. Then, by transitivity, we have $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma_2$. Then, applying Lemma E.27 to $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma_2$ and $\Sigma \vdash \sigma : \Gamma$ (from premise) gives us $\Sigma \vdash \sigma : \Gamma_2$. Finally, applying Lemma E.63 to $\Sigma \vdash \sigma : \Gamma_2$ gives us $\Sigma \vdash \sigma : \Gamma_2[r \mapsto \{\overline{\ell}\}]$.
$\Sigma;\ \Gamma_2[r \mapsto \{\,\overline{\ell}\,\}] \vdash \overline{v} : \Theta$	Inverting WF-Temporaries gives us $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma;\ \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma$. Applying Lemma E.15 to the typing derivation (from T-BorrowSlice) for e_1 (which we know is a value from E-BorrowSlice) gives us $\Sigma;\ \bullet\ \vdash\ \Gamma \rhd \Gamma_1$. Then, applying Lemma E.15 to the typing derivation (from T-BorrowSlice) for e_2 (which we know is a value from E-BorrowSlice) gives us $\Sigma;\ \bullet \vdash \Gamma_1 \rhd \Gamma_2$. Then, by transitivity, we have $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma_2$. Then, we apply Lemma E.25 to each of the typing judgments to get $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma_2;\ \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_2$. We can then apply Lemma E.62 to each of the typing judgments along with $\bullet;\ \Gamma_2;\ \Theta \vdash_\omega p \Rightarrow \{\,\overline{\ell}\,\}$ and $\Gamma_2;\ \Theta \vdash r\ \texttt{rnic}$ and $\Gamma_2(r) = \emptyset$ (all from the premise of T-BorrowSlice) to get $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma_2[r \mapsto \{\overline{\ell}\}];\ \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_2[r \mapsto \{\overline{\ell}\}]$. We can then apply WF-Temporaries to get $\Sigma;\ \Gamma_2[r \mapsto \{\,\overline{\ell}\,\}] \vdash \overline{v} : \Theta$.
$\Sigma;\ \bullet;\ \Gamma_i;\ \Theta \vdash \boxed{\texttt{ptr}\ \mathcal{R}[n_1..n_2]} : \&r\ \omega\ [\tau^{\text{SI}}] \Rightarrow \Gamma_i$	Applying Lemma E.15 to the typing derivation (from T-BorrowSlice) for e_1 (which we know is a value from E-BorrowSlice) gives us $\Sigma;\ \bullet\ \vdash\ \Gamma \rhd \Gamma_1$. Then, applying Lemma E.15 to the typing derivation (from T-BorrowSlice) for e_2 (which we know is a value from E-BorrowSlice) gives us $\Sigma;\ \bullet \vdash \Gamma_1 \rhd \Gamma_2$. Then, by transitivity, we have $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma_2$. Then, applying Lemma E.27 to $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma_2$ and $\Sigma \vdash \sigma : \Gamma$ (from premise) gives us $\Sigma \vdash \sigma : \Gamma_2$. Applying Lemma E.5 to $\Sigma \vdash \sigma : \Gamma_2$, and $\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[[v_0, \ldots, v_n]]$ gives us $\Sigma;\ \Gamma_2 \vdash \mathcal{R}^{\square}[\pi] : \tau^{\text{XI}}$. Then, note that referent well-formedness does not depend on the contents of loan sets. This means we can also conclude $\Sigma;\ \Gamma_2[r \mapsto \{\,\overline{\ell}\,\}] \vdash \mathcal{R}^{\square}[\pi] : \tau^{\text{XI}}$. We can then apply WF-RefSliceArray or WF-RefSliceSlice to get $\Sigma;\ \Gamma'[r \mapsto \{\,\overline{\ell}\,\}] \vdash \mathcal{R}^{\square}[\pi][n_1..n_2] : \tau^{\text{XI}}$. Applying Lemma E.6 to $\Sigma \vdash \sigma : \Gamma_2$, $\sigma \vdash p \Downarrow \mathcal{R} \mapsto \_[[v_0, \ldots, v_n]]$, and $\bullet;\ \omega;\ \Gamma_2 \vdash_p \{\,\overline{\ell}\,\} \Rightarrow$ gives us ${}^{\omega}\pi \in \{\,\overline{\ell}\,\}$. Finally, we can apply T-Pointer to the two facts above to get $\Sigma;\ \bullet;\ \Gamma[r \mapsto \{\,\overline{\ell}\,\}];\ \Theta \vdash \boxed{\texttt{ptr}\ \mathcal{R}[\pi][n_1..n_2]} : \&r\ \omega\ \tau^{\text{XI}} \Rightarrow \Gamma[r \mapsto \{\,\overline{\ell}\,\}]$.
$\bullet;\ \Gamma_i;\ \Theta \vdash^{+} \&r\ \omega\ [\tau^{\text{SI}}] \rightsquigarrow \&r\ \omega\ [\tau^{\text{SI}}] \dashv \Gamma_i$	Immediate by RR-Refl.

For E-BorrowSliceOOB, we pick Γ_i to be $\boxed{\Gamma}$, and need to show:

$\Sigma \vdash \sigma : \Gamma$	Immediate from our premise.
$\Sigma;\ \Gamma \vdash \overline{v} : \Theta$	Immediate from our premise.
$\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{\texttt{abort!(...)}} : \&r\ \omega\ [\tau^{\text{SI}}] \Rightarrow \Gamma'$	An `abort!` expression is well-typed (at any type) via the rule T-Abort.
$\bullet;\ \Gamma;\ \Theta \vdash^{+} \&r\ \omega\ [\tau^{\text{SI}}] \rightsquigarrow \&r\ \omega\ [\tau^{\text{SI}}] \dashv \Gamma$	Immediate by RR-Refl.
$\exists \Gamma_o . \Gamma_i \uplus \Gamma_o = \Gamma_i$	$\Gamma_o = \Gamma_i$

Case T-IndexCopy:

From premise:

$$\textsc{T-IndexCopy}\quad \frac{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \texttt{u32} \Rightarrow \Gamma' \qquad \Delta;\ \Gamma';\ \Theta \vdash_{\texttt{shrd}} p \Rightarrow \{\ \overline{\ell}\ \} \qquad \Delta;\ \Gamma' \vdash_{\texttt{shrd}} p : \tau^{\text{XI}} \qquad \tau^{\text{XI}} = [\tau^{\text{SI}};\ n] \vee \tau^{\text{XI}} = [\tau^{\text{SI}}] \qquad \texttt{copyable}_{\Sigma}\ \tau^{\text{SI}}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{p[e]} : \tau^{\text{SI}} \Rightarrow \Gamma'}$$

Since $e = p[e_i]$, by inspection of the reduction rules, we know that e steps with the following rule:

$$\textsc{E-IndexCopy}\quad \frac{\sigma \vdash p \Downarrow \_ \mapsto \_[[v_0,\ \dots,\ v_{n_i},\ \dots,\ v_n]]}{\Sigma \vdash (\sigma;\ \boxed{p[n_i]}) \rightarrow (\sigma;\ \boxed{v_{n_i}})}$$

We then pick Γ_i to be $\boxed{\Gamma'}$, and need to show:

$\Sigma \vdash \sigma : \Gamma'$	Applying Lemma E.15 to the typing derivation (from T-IndexCopy) for e (which we know is a value from E-IndexCopy) gives us $\Sigma;\ \bullet \vdash \Gamma \triangleright \Gamma'$. Then, applying Lemma E.27 to $\Sigma;\ \bullet \vdash \Gamma \triangleright \Gamma'$ and $\Sigma \vdash \sigma : \Gamma$ (from premise) gives us $\Sigma \vdash \sigma : \Gamma'$.
$\Sigma;\ \Gamma' \vdash \overline{v} : \Theta$	Inverting WF-Temporaries gives us $\forall i \in 1 \dots n.\ \Sigma;\ \bullet;\ \Gamma;\ \tau^{\text{SI}}_1, \dots, \tau^{\text{SI}}_{i-1} \vdash \boxed{v_i} : \tau^{\text{SI}}_i \Rightarrow \Gamma$. Applying Lemma E.15 to the typing derivation (from T-IndexCopy) for e (which we know is a value from E-IndexCopy) gives us $\Sigma;\ \bullet \vdash \Gamma \triangleright \Gamma'$. Thus, we can apply Lemma E.25 to each of the typing judgments to get $\forall i \in 1\ \dots\ n.\ \Sigma;\ \bullet;\ \Gamma';\ \tau^{\text{SI}}_1,\ \dots,\ \tau^{\text{SI}}_{i-1} \vdash \boxed{v_i} : \tau^{\text{SI}}_i \Rightarrow \Gamma'$. Finally, applying WF-Temporaries gives us $\Sigma;\ \Gamma' \vdash \overline{v} : \Theta$.
$\Sigma;\ \bullet;\ \Gamma';\ \Theta \vdash \boxed{v_{n_i}} : \tau^{\text{SI}} \Rightarrow \Gamma'$	Applying Lemma E.15 to the typing derivation (from T-IndexCopy) for e (which we know is a value from E-IndexCopy) gives us $\Sigma;\ \bullet \vdash \Gamma \triangleright \Gamma'$. Then, applying Lemma E.27 to $\Sigma;\ \bullet \vdash \Gamma \triangleright \Gamma'$ and $\Sigma \vdash \sigma : \Gamma$ (from premise) gives us $\Sigma \vdash \sigma : \Gamma'$. Applying Lemma E.4 to $\bullet;\ \texttt{shrd};\ \Gamma' \vdash_p \{\ \overline{\ell}\ \} \Rightarrow,\ \bullet;\ \Gamma' \vdash_{\texttt{shrd}} p : \tau^{\text{SI}}$, and $\Sigma \vdash \sigma : \Gamma'$ T-Slice (based on whether $\tau^{\text{XI}} = [\tau^{\text{SI}};\ n]$ or $[\tau^{\text{SI}}]$ respectively), we get $\forall i \in \{\ 1\ \dots\ n\ \}.\ \Sigma;\ \bullet;\ \Gamma';\ \Theta \vdash \boxed{v_i} : \tau^{\text{SI}} \Rightarrow \Gamma'$ (after accounting for the fact that the constituent expressions are values and the output environment matches the input environment). Thus, we can pick out specifically that $\Sigma;\ \bullet;\ \Gamma';\ \Theta \vdash \boxed{v_i} : \tau^{\text{SI}} \Rightarrow \Gamma'$.
$\bullet;\ \Gamma';\ \Theta \vdash^{+} \tau^{\text{SI}} \rightsquigarrow \tau^{\text{SI}} \dashv \Gamma'$	Immediate by RR-Refl.
$\exists \Gamma_o . \Gamma' \uplus \Gamma_o = \Gamma'$	$\Gamma_o = \Gamma'$

Case T-Seq:

From premise:

$$\textsc{T-Seq}\quad \frac{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1} : \tau_1^{\text{SI}} \Rightarrow \Gamma_1 \qquad \Sigma;\ \Delta;\ \text{gc-loans}_\Theta(\Gamma_1);\ \Theta \vdash \boxed{e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1;\ e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2}$$

Since $e = e_1;\ e_2$, by inspection of the reduction rules, we know that e steps with the following rule:

$$\textsc{E-Seq}\quad \frac{}{\Sigma \vdash (\sigma;\ \boxed{v;\ e}) \rightarrow (\sigma;\ \boxed{e})}$$

We then pick Γ_i to be $\boxed{\text{gc-loans}_\Theta(\Gamma_1)}$, and need to show:

$\Sigma \vdash \sigma : \text{gc-loans}_\Theta(\Gamma_1)$	Applying Lemma E.15 to the typing derivation (from T-Seq) for e_1 (which we know is a value from E-Seq) gives us $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma_1$. Applying Lemma E.27 with this and $\Sigma \vdash \sigma : \Gamma$ (from premise) gives us $\Sigma \vdash \sigma : \Gamma_1$. By definition of R-Env, we know that $\Sigma;\ \bullet \vdash \Gamma_1 \rhd \text{gc-loans}_\Theta(\Gamma_1)$. Then, we can apply Lemma E.27 to get $\Sigma \vdash \sigma : \text{gc-loans}_\Theta(\Gamma_1)$.
$\Sigma;\ \text{gc-loans}_\Theta(\Gamma_1) \vdash \overline{v} : \Theta$	Inverting WF-Temporaries gives us $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma;\ \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma$. Applying Lemma E.15 to the typing derivation (from T-Seq) for e_1 (which we know is a value from E-Seq) gives us $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma_1$. Then, we note that definitionally $\Sigma;\ \bullet \vdash \Gamma_1 \rhd \text{gc-loans}_\Theta(\Gamma_1)$ since we can only clear loans that are not present in any of the types. Then, by transitivity, we have $\Sigma;\ \bullet \vdash \Gamma \rhd \text{gc-loans}_\Theta(\Gamma_1)$. Thus, we can apply Lemma E.25 to each of the typing judgments to get $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \text{gc-loans}_\Theta(\Gamma_1);\ \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \text{gc-loans}_\Theta(\Gamma_1)$. Finally, applying WF-Temporaries gives us $\Sigma;\ \text{gc-loans}_\Theta(\Gamma_1) \vdash \overline{v} : \Theta$.
$\Sigma;\ \bullet;\ \text{gc-loans}_\Theta(\Gamma_1);\ \Theta \vdash \boxed{e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2$	Immediate from the premise of T-Seq.
$\bullet;\ \Gamma_2;\ \Theta \vdash^{+} \tau_2^{\text{SI}} \rightsquigarrow \tau_2^{\text{SI}} \dashv \Gamma_2$	Immediate by RR-Refl.
$\exists \Gamma_o . \Gamma_2 \uplus \Gamma_o = \Gamma_2$	$\Gamma_o = \Gamma_2$

Case T-Branch:

From premise:

$$\textsc{T-Branch}\quad \frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1} : \mathsf{bool} \Rightarrow \Gamma_1 \qquad \Sigma;\ \Delta;\ \Gamma_1;\ \Theta \vdash \boxed{e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2 \\ \Sigma;\ \Delta;\ \Gamma_1;\ \Theta \vdash \boxed{e_3} : \tau_3^{\text{SI}} \Rightarrow \Gamma_3 \qquad \tau^{\text{SI}} = \tau_2^{\text{SI}} \lor \tau^{\text{SI}} = \tau_3^{\text{SI}} \\ \Delta;\ \Gamma_2;\ \Theta \vdash^{+} \tau_2^{\text{SI}} \rightsquigarrow \tau^{\text{SI}} \dashv \Gamma_2' \qquad \Delta;\ \Gamma_3;\ \Theta \vdash^{+} \tau_3^{\text{SI}} \rightsquigarrow \tau^{\text{SI}} \dashv \Gamma_3' \qquad \Gamma_2' \Cup \Gamma_3' = \Gamma'\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\mathtt{if}\ e_1\ \{\ e_2\ \}\ \mathtt{else}\ \{\ e_3\ \}} : \tau^{\text{SI}} \Rightarrow \Gamma'}$$

Since $e = \mathtt{if}\ e_1\ \{\ e_2\ \}\ \mathtt{else}\ \{\ e_3\ \}$, by inspection of the reduction rules, we know that e steps with the following rule:

$$\textsc{E-IfTrue}\quad \frac{}{\Sigma \vdash (\sigma;\ \boxed{\mathtt{if\ true}\ \{\ e_1\ \}\ \mathtt{else}\ \{\ e_2\ \}}) \rightarrow (\sigma;\ \boxed{e_1})} \qquad \textsc{E-IfFalse}\quad \frac{}{\Sigma \vdash (\sigma;\ \boxed{\mathtt{if\ false}\ \{\ e_1\ \}\ \mathtt{else}\ \{\ e_2\ \}}) \rightarrow (\sigma;\ \boxed{e_2})}$$

Then, for each possible rule, we'll pick Γ_i separately. The cases proceed as follows:
For E-IfTrue, we pick Γ_i to be $\boxed{\Gamma_1}$, and need to show:

$\Sigma \vdash \sigma : \Gamma_1$	Applying Lemma E.15 to the typing derivation (from T-Branch) for e_1 (which we know is a value from E-IfTrue) gives us $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma_1$. Then, applying Lemma E.27 with this and $\Sigma \vdash \sigma : \Gamma$ (from premise) gives us $\Sigma \vdash \sigma : \Gamma_1$.
$\Sigma;\ \Gamma_1 \vdash \overline{v} : \Theta$	Inverting WF-Temporaries gives us $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma;\ \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma$. Applying Lemma E.15 to the typing derivation (from T-Branch) for e_1 (which we know is a value from E-IfTrue) gives us $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma_1$. Thus, we can apply Lemma E.25 to each of the typing judgments to get $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma_1;\ \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_1$. Finally, applying WF-Temporaries gives us $\Sigma;\ \Gamma_1 \vdash \overline{v} : \Theta$.
$\Sigma;\ \bullet;\ \Gamma_1;\ \Theta \vdash \boxed{e_2} : \tau^{\text{SI}} \Rightarrow \Gamma_2$	Immediate from premise of T-Branch.
$\bullet;\ \Gamma_2;\ \Theta \vdash^{+} \tau_2^{\text{SI}} \rightsquigarrow \tau^{\text{SI}} \dashv \Gamma_2'$	Immediate from premise of T-Branch.
$\exists \Gamma_o. \Gamma_2' \Cup \Gamma_o = \Gamma'$	$\Gamma_o = \Gamma_3'$

For E-IfFalse, we pick Γ_i to be $\boxed{\Gamma_1}$, and need to show:

$\Sigma \vdash \sigma : \Gamma_1$	Applying Lemma E.15 to the typing derivation (from T-Branch) for e_1 (which we know is a value from E-IfFalse) gives us $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma_1$. Then, applying Lemma E.27 with this and $\Sigma \vdash \sigma : \Gamma$ (from premise) gives us $\Sigma \vdash \sigma : \Gamma_1$.
$\Sigma;\ \Gamma_1 \vdash \overline{v} : \Theta$	Inverting WF-Temporaries gives us $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma;\ \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma$. Applying Lemma E.15 to the typing derivation (from T-Branch) for e_1 (which we know is a value from E-IfFalse) gives us $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma_1$. Thus, we can apply Lemma E.25 to each of the typing judgments to get $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma_1;\ \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_1$. Finally, applying WF-Temporaries gives us $\Sigma;\ \Gamma_1 \vdash \overline{v} : \Theta$.
$\Sigma;\ \bullet;\ \Gamma_1;\ \Theta \vdash \boxed{e_3} : \tau_3^{\text{SI}} \Rightarrow \Gamma_3$	Immediate from premise of T-Branch.
$\bullet;\ \Gamma_3;\ \Theta \vdash^{+} \tau_3^{\text{SI}} \rightsquigarrow \tau^{\text{SI}} \dashv \Gamma_3'$	Immediate from premise of T-Branch.
$\exists \Gamma_o. \Gamma_3' \Cup \Gamma_o = \Gamma'$	$\Gamma_o = \Gamma_2'$ (note that $\Cup$ commutes)

Case T-MATCH:

From premise:

$$\dfrac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \mathtt{Either}\texttt{<}\tau_l^{\text{SI}},\ \tau_r^{\text{SI}}\texttt{>} \Rightarrow \Gamma' \qquad \forall r \in \text{free-regions}(\mathtt{Either}\texttt{<}\tau_l^{\text{SI}},\ \tau_r^{\text{SI}}\texttt{>}).\ \Gamma' \vdash r\ \mathtt{rnrb} \\ \Sigma;\ \Delta;\ \Gamma',\ x_1 : \tau_l^{\text{SI}};\ \Theta \vdash \boxed{e_1} : \tau_1^{\text{SI}} \Rightarrow \Gamma_1,\ x_1 : \tau_l^{\text{SD}} \\ \Sigma;\ \Delta;\ \Gamma',\ x_2 : \tau_r^{\text{SI}};\ \Theta \vdash \boxed{e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2,\ x_2 : \tau_r^{\text{SD}} \qquad \tau^{\text{SI}} = \tau_1^{\text{SI}} \lor \tau^{\text{SI}} = \tau_2^{\text{SI}} \\ \Delta;\ \Gamma_1;\ \Theta \vdash^+ \tau_1^{\text{SI}} \rightsquigarrow \tau^{\text{SI}} \dashv \Gamma_1' \qquad \Delta;\ \Gamma_2;\ \Theta \vdash^+ \tau_2^{\text{SI}} \rightsquigarrow \tau^{\text{SI}} \dashv \Gamma_2' \qquad \Gamma_1' \Cup \Gamma_2' = \Gamma'\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\mathtt{match}\ e\ \{\ \mathtt{Left}(x_1) \Rightarrow e_1,\ \mathtt{Right}(x_2) \Rightarrow e_2\ \}} : \tau^{\text{SI}} \Rightarrow \Gamma'}\ \text{T-MATCH}$$

Since $e = \mathtt{match}\ e\ \{\ \mathtt{Left}(x_1) \Rightarrow e_1,\ \mathtt{Right}(x_2) \Rightarrow e_2\ \}$, by inspection of the reduction rules, we know that e steps with the following rule:

$$\dfrac{}{\Sigma \vdash (\sigma;\ \boxed{\mathtt{match}\ \mathtt{Left}::\texttt{<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}\texttt{>}(v)\ \{\ \mathtt{Left}(x_1) \Rightarrow e_1,\ \mathtt{Right}(x_2) \Rightarrow e_2\ \}}) \rightarrow (\sigma,\ x_1 \mapsto v;\ \boxed{\mathtt{shift}\ e_1})}\ \text{E-MATCHLEFT}$$

$$\dfrac{}{\Sigma \vdash (\sigma;\ \boxed{\mathtt{match}\ \mathtt{Right}::\texttt{<}\tau_1^{\text{SI}},\ \tau_2^{\text{SI}}\texttt{>}(v)\ \{\ \mathtt{Left}(x_1) \Rightarrow e_1,\ \mathtt{Right}(x_2) \Rightarrow e_2\ \}}) \rightarrow (\sigma,\ x_2 \mapsto v;\ \boxed{\mathtt{shift}\ e_2})}\ \text{E-MATCHRIGHT}$$

Then, for each possible rule, we'll pick Γ_i separately. The cases proceed as follows:
For E-MATCHLEFT, we pick Γ_i to be $\boxed{\Gamma',\ x_1 : \tau_l^{\text{SI}}}$, and need to show:

$\Sigma \vdash \sigma,\ x_1 \mapsto v : \Gamma',\ x_1 : \tau_l^{\text{SI}}$	Applying Lemma E.15 to the typing derivation (from T-MATCH) for e (which we know is a value from E-MATCHLEFT) gives us $\Sigma;\ \bullet \vdash \Gamma \triangleright \Gamma'$. Then, applying Lemma E.27 with this and $\Sigma \vdash \sigma : \Gamma$ (from premise) gives us $\Sigma \vdash \sigma : \Gamma'$. Then, using $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau_l^{\text{SI}} \Rightarrow \Gamma'$ (which we get by inversion of T-INL in the derivation for e) with WF-STACKFRAME allows us to finally conclude $\Sigma \vdash \sigma,\ x_1 \mapsto v : \Gamma',\ x_1 : \tau_l^{\text{SI}}$.
$\Sigma;\ \Gamma',\ x_1 : \tau_l^{\text{SI}} \vdash \overline{v} : \Theta$	Inverting WF-TEMPORARIES gives us $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma;\ \tau_1^{\text{SI}},\ \ldots,\ \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma$. Applying Lemma E.15 to the typing derivation (from T-MATCH) for e (which we know is a value from E-MATCHLEFT) gives us $\Sigma;\ \bullet \vdash \Gamma \triangleright \Gamma'$. Thus, we can apply Lemma E.25 to each of the typing judgments to get $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma';\ \tau_1^{\text{SI}},\ \ldots,\ \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma'$. Applying Lemma E.34 to each of the value typing judgments and $\forall r \in \text{free-regions}(\mathtt{Either}\texttt{<}\tau_l^{\text{SI}},\ \tau_r^{\text{SI}}\texttt{>}).\ \Gamma' \vdash r\ \mathtt{rnrb}$ (from the premise of T-MATCH) gives us $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma',\ x_1 : \tau_l^{\text{SI}};\ \tau_1^{\text{SI}},\ \ldots,\ \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma',\ x_1 : \tau_l^{\text{SI}}$. Finally, applying WF-TEMPORARIES gives us $\Sigma;\ \Gamma',\ x_1 : \tau_l^{\text{SI}} \vdash \overline{v} : \Theta$.
$\Sigma;\ \bullet;\ \Gamma',\ x_1 : \tau_l^{\text{SI}};\ \Theta \vdash \boxed{e_1} : \tau_1^{\text{SI}} \Rightarrow \Gamma_1$	Immediate from applying T-SHIFT to $\Sigma;\ \bullet;\ \Gamma',\ x_1 : \tau_l^{\text{SI}};\ \Theta \vdash \boxed{e_1} : \tau_1^{\text{SI}} \Rightarrow \Gamma_1,\ x_1 : \tau_l^{\text{SD}}$ from the premise of T-MATCH.
$\bullet;\ \Gamma_1;\ \Theta \vdash^+ \tau_1^{\text{SI}} \rightsquigarrow \tau^{\text{SI}} \dashv \Gamma_1'$	Immediate from premise of T-MATCH.
$\exists \Gamma_o . \Gamma_1' \Cup \Gamma_o = \Gamma'$	$\Gamma_o = \Gamma_2'$

For E-MatchRight, we pick Γ_i to be $\boxed{\Gamma',\ x_2\ :\ \tau_r^{\text{SI}}}$, and need to show:

$\Sigma \vdash \sigma,\ x_2 \mapsto v : \Gamma',\ x_2\ :\ \tau_r^{\text{SI}}$	Applying Lemma E.15 to the typing derivation (from T-Match) for e (which we know is a value from E-MatchRight) gives us $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma'$. Then, applying Lemma E.27 with this and $\Sigma \vdash \sigma : \Gamma$ (from premise) gives us $\Sigma \vdash \sigma : \Gamma'$. Then, using $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau_r^{\text{SI}} \Rightarrow \Gamma'$ (which we get by inversion of T-Inr in the derivation for e) with WF-StackFrame allows us to finally conclude $\Sigma \vdash \sigma,\ x_2 \mapsto v : \Gamma',\ x_2\ :\ \tau_r^{\text{SI}}$.
$\Sigma;\ \Gamma',\ x_2\ :\ \tau_r^{\text{SI}} \vdash \overline{v} : \Theta$	Inverting WF-Temporaries gives us $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma;\ \tau_1^{\text{SI}},\ \ldots,\ \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma$. Applying Lemma E.15 to the typing derivation (from T-Match) for e (which we know is a value from E-MatchLeft) gives us $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma'$. Thus, we can apply Lemma E.25 to each of the typing judgments to get $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma';\ \tau_1^{\text{SI}},\ \ldots,\ \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma'$. Applying Lemma E.34 to each of the value typing judgments and $\forall r \in \text{free-regions}(\texttt{Either<}\tau_l^{\text{SI}},\ \tau_r^{\text{SI}}\texttt{>})$. $\Gamma' \vdash r\ \texttt{rnrb}$ (from the premise of T-Match) gives us $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma',\ x_2\ :\ \tau_r^{\text{SI}};\ \tau_1^{\text{SI}},\ \ldots,\ \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma',\ x_2\ :\ \tau_r^{\text{SI}}$. Finally, applying WF-Temporaries gives us $\Sigma;\ \Gamma',\ x_2\ :\ \tau_r^{\text{SI}} \vdash \overline{v} : \Theta$.
$\Sigma;\ \bullet;\ \Gamma',\ x_2\ :\ \tau_r^{\text{SI}};\ \Theta \vdash \boxed{e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2$	Immediate from applying T-Shift to $\Sigma;\ \bullet;\ \Gamma',\ x_2\ :\ \tau_r^{\text{SI}};\ \Theta \vdash \boxed{e_2} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2,\ x_2\ :\ \tau_r^{\text{SD}}$ from the premise of T-Match.
$\bullet;\ \Gamma_2;\ \Theta \vdash^{+} \tau_2^{\text{SI}} \rightsquigarrow \tau^{\text{SI}} \dashv \Gamma_2'$	Immediate from premise of T-Match.
$\exists \Gamma_o.\Gamma_2' \uplus \Gamma_o = \Gamma'$	$\Gamma_o = \Gamma_1'$ (note that $\uplus$ commutes)

Case T-Assign:

From premise:

T-Assign
$$\frac{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau^{\text{SI}} \Rightarrow \Gamma_1 \qquad \Gamma_1(\pi) = \tau^{\text{SX}} \qquad \tau^{\text{SX}} = \&r\ \omega\ \tau^{\text{XI}} \implies r \text{ is unique to } \pi \text{ in } \Gamma_1 \qquad \Delta;\ \Gamma_1 \unrhd *\pi;\ \Theta \vdash^{=} \tau^{\text{SI}} \rightsquigarrow \tau^{\text{SX}} \dashv \Gamma' \qquad (\tau^{\text{SX}} = \tau^{\text{SD}} \lor \Delta;\ \Gamma';\ \Theta \vdash_{\texttt{uniq}} \pi \Rightarrow \{\ {}^{\texttt{uniq}}\pi\ \})}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\pi := e} : \texttt{unit} \Rightarrow \Gamma'[\pi \mapsto \tau^{\text{SI}}]}$$

Since $e = \pi := e_a$, by inspection of the reduction rules, we know that e steps with the following rule:

E-Assign
$$\frac{\sigma \vdash p \Downarrow \mathcal{R} \mapsto \mathcal{V}[\_] \qquad \mathcal{R} = \mathcal{R}^{\square}[x]}{\Sigma \vdash (\sigma;\ \boxed{p := v}) \rightarrow (\sigma[x \mapsto \mathcal{V}[v]];\ \boxed{()})}$$

We then pick Γ_i to be $\boxed{\text{gc-loans}_{\Theta}(\Gamma'[\pi \mapsto \tau^{\text{SI}}])}$, and need to show:

$\Sigma \vdash \sigma[\pi \mapsto \mathcal{V}[v]] : \text{gc-loans}_\Theta(\Gamma'[\pi \mapsto \tau^{\text{SI}}])$	From our premise, we have $\Sigma \vdash \sigma : \Gamma$. Then, applying Lemma E.15 to the typing derivation $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau^{\text{SI}} \Rightarrow \Gamma_1$ (from the premise of T-Assign combined with the fact that e is a value from E-Assign) gives us $\Sigma;\ \bullet \vdash \Gamma \triangleright \Gamma_1$. Then, applying Lemma E.27 to these two facts gives us $\Sigma \vdash \sigma : \Gamma_1$. Finally, we apply Lemma E.41 to $\Sigma \vdash \sigma : \Gamma_1$ and $\Gamma_1(\pi) = \tau^{\text{SX}}$ (from premise) and $\bullet;\ \Gamma_1 \trianglerighteq \pi;\ \Theta \vdash^{=} \tau^{\text{SI}} \rightsquigarrow \tau^{\text{SX}} \dashv \Gamma'$ (from the premise of T-Assign) and $\bullet;\ \texttt{uniq};\ \Gamma' \vdash_\pi \{\ {}^{\texttt{uniq}}\pi\ \} \Rightarrow$ (from premise of T-Assign and $\sigma \vdash \pi \Downarrow \pi \mapsto \mathcal{V}[\_]$ and $\pi = x.q$ (both from the premise of E-Assign), $\Sigma;\ \bullet;\ \Gamma_1 \trianglerighteq p;\ \Theta \vdash \boxed{v} : \tau^{\text{SI}} \Rightarrow \Gamma_1 \trianglerighteq p$ (from above) and $\Sigma \vdash \sigma[x \mapsto \mathcal{V}[v]] : \text{gc-loans}_\Theta(\Gamma'[\pi \mapsto \tau^{\text{SI}}])$.
$\Sigma;\ \text{gc-loans}_\Theta(\Gamma'[\pi \mapsto \tau^{\text{SI}}]) \vdash \overline{v} : \Theta$	Inverting WF-Temporaries gives us $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma;\ \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma$. Applying Lemma E.15 to the typing derivation (from T-Let) for e_1 (which we know is a value from E-Let) gives us $\Sigma;\ \bullet \vdash \Gamma \triangleright \Gamma_1$. Then, we can apply Lemma E.25 to each of the typing judgments to get $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma_1;\ \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_1$. Then, we apply Lemma E.40 to each of these typing judgments to get $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \text{gc-loans}_\Theta(\Gamma'[\pi \mapsto \tau^{\text{SI}}]);\ \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \text{gc-loans}_\Theta(\Gamma'[\pi \mapsto \tau^{\text{SI}}])$. After which we can apply WF-Temporaries to conclude $\Sigma;\ \text{gc-loans}_\Theta(\Gamma'[\pi \mapsto \tau^{\text{SI}}]) \vdash \overline{v} : \Theta$.
$\Sigma;\ \bullet;\ \Gamma_i;\ \Theta \vdash \boxed{()} : \texttt{unit} \Rightarrow \Gamma_i$	Immediate by T-Unit.
$\bullet;\ \Gamma_i;\ \Theta \vdash^{+} \texttt{unit} \rightsquigarrow \texttt{unit} \dashv \Gamma_i$	Immediate by RR-Refl.
$\exists \Gamma_o . \Gamma_i \uplus \Gamma_o = \Gamma_i$	$\Gamma_o = \Gamma_i = \text{gc-loans}_\Theta(\Gamma'[\pi \mapsto \tau^{\text{SI}}])$

Case T-AssignDeref:

From premise:

T-AssignDeref
$$\frac{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau_n^{\text{SI}} \Rightarrow \Gamma_1 \qquad \Delta;\ \Gamma_1 \vdash_{\texttt{uniq}} p : \tau_o^{\text{SI}} \qquad \Delta;\ \Gamma_1;\ \Theta \vdash^{+} \tau_n^{\text{SI}} \rightsquigarrow \tau_o^{\text{SI}} \dashv \Gamma' \qquad \Delta;\ \Gamma';\ \Theta \vdash_{\texttt{uniq}} p \Rightarrow \{\ \overline{\ell}\ \}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{p := e} : \texttt{unit} \Rightarrow \Gamma'}$$

Since $e = p := e_a$, by inspection of the reduction rules, we know that e steps with the following rule:

E-Assign
$$\frac{\sigma \vdash p \Downarrow \mathcal{R} \mapsto \mathcal{V}[\_] \qquad \mathcal{R} = \mathcal{R}^{\square}[x]}{\Sigma \vdash (\sigma;\ \boxed{p := v}) \rightarrow (\sigma[x \mapsto \mathcal{V}[v]];\ \boxed{()})}$$

We then pick Γ_i to be $\boxed{\Gamma'}$, and need to show:

$\Sigma \vdash \sigma[\pi \mapsto \mathcal{V}[v]] : \Gamma'$	From our premise, we have $\Sigma \vdash \sigma : \Gamma$. Then, applying Lemma E.15 to the typing derivation $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau^{\text{SI}} \Rightarrow \Gamma_1$ (from the premise of T-AssignDeref combined with the fact that e is a value from E-Assign) gives us $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma_1$. Then, applying Lemma E.27 to these two facts gives us $\Sigma \vdash \sigma : \Gamma_1$. Then, applying Lemma E.14 to $\Sigma \vdash \sigma : \Gamma_1$ and $\bullet;\ \Gamma_1;\ \Theta \vdash^{+} \tau^{\text{SI}}_n \rightsquigarrow \tau^{\text{SI}}_o \dashv \Gamma'$ (from the premise of T-AssignDeref) gives us $\Sigma \vdash \sigma : \Gamma'$.
$\Sigma;\ \Gamma' \vdash \overline{v} : \Theta$	Inverting WF-Temporaries gives us $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma;\ \tau^{\text{SI}}_1, \ldots, \tau^{\text{SI}}_{i-1} \vdash \boxed{v_i} : \tau^{\text{SI}}_i \Rightarrow \Gamma$. Applying Lemma E.15 to the typing derivation (from T-AssignDeref) for e (which we know is a value from E-Assign) gives us $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma_1$. Then, we can apply Lemma E.25 to each of the typing judgments to get $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma_1;\ \tau^{\text{SI}}_1, \ldots, \tau^{\text{SI}}_{i-1} \vdash \boxed{v_i} : \tau^{\text{SI}}_i \Rightarrow \Gamma_1$. Then, applying Lemma E.13 to each of these derivations along with $\bullet;\ \Gamma_1;\ \Theta \vdash^{+} \tau^{\text{SI}}_n \rightsquigarrow \tau^{\text{SI}}_o \dashv \Gamma'$ (from the premise of T-AssignDeref) gives us $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma';\ \tau^{\text{SI}}_1, \ldots, \tau^{\text{SI}}_{i-1} \vdash \boxed{v_i} : \tau^{\text{SI}}_i \Rightarrow \Gamma'$. We can then apply WF-Temporaries to conclude $\Sigma;\ \Gamma' \vdash \overline{v} : \Theta$.
$\Sigma;\ \bullet;\ \Gamma';\ \Theta \vdash \boxed{()} : \texttt{unit} \Rightarrow \Gamma'$	Immediate by T-Unit.
$\bullet;\ \Gamma';\ \Theta \vdash^{+} \texttt{unit} \rightsquigarrow \texttt{unit} \dashv \Gamma'$	Immediate by RR-Refl.
$\exists \Gamma_o . \Gamma' \uplus \Gamma_o = \Gamma'$	$\Gamma_o = \Gamma'$

Case T-Let:

From premise:

T-Let

$$\dfrac{\begin{array}{cc}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1} : \tau^{\text{SI}}_1 \Rightarrow \Gamma_1 & \Delta;\ \Gamma_1;\ \Theta \vdash^{+} \tau^{\text{SI}}_1 \rightsquigarrow \tau^{\text{SI}}_a \dashv \Gamma'_1 \\ \forall r \in \text{free-regions}(\tau^{\text{SI}}_a).\ \Gamma'_1 \vdash r\ \texttt{rnrb} & \Sigma;\ \Delta;\ \text{gc-loans}_{\Theta}(\Gamma'_1,\ x : \tau^{\text{SI}}_a);\ \Theta \vdash \boxed{e_2} : \tau^{\text{SI}}_2 \Rightarrow \Gamma_2,\ x : \tau^{\text{SD}}\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{let}\ x : \tau^{\text{SI}}_a = e_1;\ e_2} : \tau^{\text{SI}}_2 \Rightarrow \Gamma_2}$$

Since $e = \texttt{let}\ x : \tau^{\text{SI}} = e_1;\ e_2$, by inspection of the reduction rules, we know that e steps with the following rule:

E-Let

$$\dfrac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{let}\ x : \tau^{\text{SI}}_a = v;\ e}) \to (\sigma,\ x \mapsto v;\ \boxed{\texttt{shift}\ e})}$$

We then pick Γ_i to be $\boxed{\text{gc-loans}_{\Theta}(\Gamma'_1,\ x : \tau^{\text{SI}}_a)}$, and need to show:

$\Sigma \vdash \sigma, x \mapsto v : \text{gc-loans}_\Theta(\Gamma_1', x : \tau_a^{\text{SI}})$	Applying Lemma E.15 to the typing derivation (from T-Let) for e_1 (which we know is a value from E-Let) gives us $\Sigma; \bullet \vdash \Gamma \rhd \Gamma_1$. Then, we can apply Lemma E.27 to get $\Sigma \vdash \sigma : \Gamma_1$. Then, applying Lemma E.14 to $\bullet; \Gamma_1; \Theta \vdash^+ \tau_1^{\text{SI}} \rightsquigarrow \tau_a^{\text{SI}} \dashv \Gamma_1'$ (from premise of T-Let) gives us $\Sigma \vdash \sigma : \Gamma_1'$. We can also apply Lemma E.42 to $\Sigma; \bullet; \Gamma; \Theta \vdash \boxed{v} : \tau_1^{\text{SI}} \Rightarrow \Gamma_1$ (from premise of T-Let) and $\bullet; \Gamma_1; \Theta \vdash^+ \tau_1^{\text{SI}} \rightsquigarrow \tau_a^{\text{SI}} \dashv \Gamma_1'$ (from premise of T-Let) gives us $\Sigma; \bullet; \Gamma'; \Theta \vdash \boxed{e_1} : \tau_a^{\text{SI}} \Rightarrow \Gamma'$ Then, apply Lemma E.35 to $\Sigma \vdash \sigma : \Gamma_1'$ and $\Sigma; \bullet; \Gamma'; \Theta \vdash \boxed{e_1} : \tau_a^{\text{SI}} \Rightarrow \Gamma'$ gives us $\Sigma \vdash \sigma, x \mapsto v : \Gamma_1', x : \tau_a^{\text{SI}}$. By definition of R-Env, we know that $\Sigma; \bullet \vdash \Gamma_1', x : \tau_a^{\text{SI}} \rhd \text{gc-loans}_\Theta(\Gamma_1', x : \tau_a^{\text{SI}})$. Then, we can apply Lemma E.27 to conclude $\Sigma \vdash \sigma, x \mapsto v : \text{gc-loans}_\Theta(\Gamma_1', x : \tau_a^{\text{SI}})$.
$\Sigma; \text{gc-loans}_\Theta(\Gamma_1', x : \tau_a^{\text{SI}}) \vdash \bar{v} : \Theta$	Inverting WF-Temporaries gives us $\forall i \in 1 \ldots n.\ \Sigma; \bullet; \Gamma; \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma$. Applying Lemma E.15 to the typing derivation (from T-Let) for e_1 (which we know is a value from E-Let) gives us $\Sigma; \bullet \vdash \Gamma \rhd \Gamma_1$. Then, we can apply Lemma E.25 to each of the typing judgments to get $\forall i \in 1 \ldots n.\ \Sigma; \bullet; \Gamma_1; \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_1$. Applying Lemma E.34 to each of the value typing judgments and $\forall r \in \text{free-regions}(\tau_a^{\text{SI}}).\ \Gamma_1' \vdash r\ \texttt{rnrb}$ (from the premise of T-Let) gives us $\forall i \in 1 \ldots n.\ \Sigma; \bullet; \Gamma_1, x : \tau_a^{\text{SI}}; \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_1, x : \tau_a^{\text{SI}}$. Then, we note that definitionally $\Sigma; \bullet \vdash \Gamma_1, x : \tau_a^{\text{SI}} \rhd \text{gc-loans}_\Theta(\Gamma_1, x : \tau_a^{\text{SI}})$ since we can only clear loans that are not present in any of the types. We can use this with each typing derivation in Lemma E.15, and then finally apply WF-Temporaries to get $\Sigma; \text{gc-loans}_\Theta(\Gamma_1, x : \tau_a^{\text{SI}}) \vdash \bar{v} : \Theta$.
$\Sigma; \bullet; \text{gc-loans}_\Theta(\Gamma_1', x : \tau_a^{\text{SI}}); \Theta \vdash \boxed{\texttt{shift}\ e} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2$	Immediate by applying T-Shift to the derivation $\Sigma; \bullet; \text{gc-loans}_\Theta(\Gamma_1', x : \tau_a^{\text{SI}}); \Theta \vdash \boxed{e} : \tau_2^{\text{SI}} \Rightarrow \Gamma_2, x : \tau^{\text{SD}}$ (from premise of T-Let).
$\bullet; \Gamma_2; \Theta \vdash^+ \tau_2^{\text{SI}} \rightsquigarrow \tau_2^{\text{SI}} \dashv \Gamma_2$	Immediate by RR-Refl.
$\exists \Gamma_o. \Gamma_2 \uplus \Gamma_o = \Gamma_2$	$\Gamma_o = \Gamma_2$

Case T-LetRegion:

From premise:

$$\textsc{T-LetRegion}\ \frac{\Sigma;\ \Delta;\ \Gamma\,,\ r \mapsto \{\};\ \Theta \vdash \boxed{e} : \tau^{\text{SI}} \Rightarrow \Gamma'\,,\ r \mapsto \{\overline{\ell}\}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{letrgn}\ \texttt{<}r\texttt{>}\ \{\ e\ \}} : \tau^{\text{SI}} \Rightarrow \Gamma'}$$

Since $e = \texttt{letrgn <}r\texttt{> \{}\ e\ \texttt{\}}$, by inspection of the reduction rules, we know that e steps with the following rule:

$$\textsc{E-LetRegion}\ \frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{letrgn}\ \texttt{<}r\texttt{>}\ \{\ v\ \}}) \rightarrow (\sigma;\ \boxed{v})}$$

We then pick Γ_i to be $\boxed{\Gamma'}$, and need to show:

$\Sigma \vdash \sigma : \Gamma'$	Applying Lemma E.15 to the typing derivation (from T-LetRegion) for e (which we know is a value from E-LetRegion) gives us $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma'$. Then, applying Lemma E.27 with this and $\Sigma \vdash \sigma : \Gamma$ (from premise) gives us $\Sigma \vdash \sigma : \Gamma'$.
$\Sigma;\ \Gamma' \vdash \overline{v} : \Theta$	Inverting WF-Temporaries gives us $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma;\ \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma$. Applying Lemma E.15 to the typing derivation (from T-LetRegion) for e (which we know is a value from E-LetRegion) gives us $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma'$. Thus, we can apply Lemma E.25 to each of the typing judgments to get $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma';\ \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma'$. Finally, applying WF-Temporaries gives us $\Sigma;\ \Gamma' \vdash \overline{v} : \Theta$.
$\Sigma;\ \bullet;\ \Gamma';\ \Theta \vdash \boxed{v} : \tau^{\text{SI}} \Rightarrow \Gamma'$	We know from E-LetRegion that e is a value v. Thus, we can apply Lemma E.26 to $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau^{\text{SI}} \Rightarrow \Gamma'\,, r \mapsto \{\overline{\ell}\}$ to get $\Sigma;\ \bullet;\ \Gamma'\,, r \mapsto \{\ \overline{\ell}\ \};\ \Theta \vdash \boxed{v} : \tau^{\text{SI}} \Rightarrow \Gamma'\,,\ r \mapsto \{\ \overline{\ell}\ \}$. We now wish to show that $\Sigma;\ \bullet;\ \Gamma';\ \Theta \vdash \boxed{v} : \tau^{\text{SI}} \Rightarrow \Gamma'$. By inspecting the grammar of values and their typing rules, we know that the only values who depend on the context are pointers and closure values. But by inversion on $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{\texttt{letrgn}\ \texttt{<}r\texttt{>}\ \{\ v\ \}} : \tau^{\text{SI}} \Rightarrow \Gamma'$, we know that $\Sigma;\ \bullet;\ \Gamma' \vdash \tau^{\text{SI}}$. Since the type is valid without the frame, we know that the values cannot depend on that frame. Thus, we can conclude $\Sigma;\ \bullet;\ \Gamma';\ \Theta \vdash \boxed{v} : \tau^{\text{SI}} \Rightarrow \Gamma'$.
$\bullet;\ \Gamma';\ \Theta \vdash^{+} \tau^{\text{SI}} \rightsquigarrow \tau^{\text{SI}} \dashv \Gamma'$	Immediate by RR-Refl.
$\exists \Gamma_o.\Gamma' \Cup \Gamma_o = \Gamma'$	$\Gamma_o = \Gamma'$

Case T-While:

From premise:

$$\frac{\begin{array}{cc}\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{e_1} : \texttt{bool} \Rightarrow \Gamma_1 & \Sigma; \Delta; \Gamma_1; \Theta \vdash \boxed{e_2} : \texttt{unit} \Rightarrow \Gamma_2 \\ \Sigma; \Delta; \Gamma_2; \Theta \vdash \boxed{e_1} : \texttt{bool} \Rightarrow \Gamma_2 & \Sigma; \Delta; \Gamma_2; \Theta \vdash \boxed{e_2} : \texttt{unit} \Rightarrow \Gamma_2\end{array}}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{\texttt{while}\, e_1 \,\{\, e_2 \,\}} : \texttt{unit} \Rightarrow \Gamma_2}\ \textsc{T-While}$$

Since $e = \texttt{while}\, e_1 \{ e_2 \}$, by inspection of the reduction rules, we know that e steps with the following rule:

$$\frac{}{\Sigma \vdash (\sigma; \boxed{\texttt{while}\, e_1 \,\{\, e_2 \,\}}) \to (\sigma; \boxed{\texttt{if}\, e_1 \,\{\, e_2; \texttt{while}\, e_1 \,\{\, e_2 \,\} \,\}\, \texttt{else} \,\{\, () \,\}})}\ \textsc{E-While}$$

We then pick Γ_i to be $\boxed{\Gamma}$, and need to show:

$\Sigma \vdash \sigma : \Gamma$	Immediate from our premise.
$\Sigma; \Gamma \vdash \overline{v} : \Theta$	Immediate from our premise.
$\Sigma; \bullet; \Gamma; \Theta \vdash \boxed{e'} : \texttt{unit} \Rightarrow \Gamma_2$	We would like to build a derivation to show that the expression $\texttt{if}\, e_1 \{ e_2; \texttt{while}\, e_1 \{ e_2 \} \} \texttt{else} \{ () \}$ is well-typed. We thus start by applying T-Branch. This requires us to show three things. First, $\Sigma; \bullet; \Gamma; \Theta \vdash \boxed{e_1} : \texttt{bool} \Rightarrow \Gamma_1$ which we have from the premise of T-While. Second, $\Sigma; \bullet; \Gamma_1; \Theta \vdash \boxed{e_2; \texttt{while}\, e_1 \{ e_2 \}} : \texttt{unit} \Rightarrow \Gamma_2$. We build this by applying T-Seq to $\Sigma; \bullet; \Gamma_1; \Theta \vdash \boxed{e_2} : \texttt{unit} \Rightarrow \Gamma_2$ and $\Sigma; \bullet; \Gamma_2; \Theta \vdash \boxed{\texttt{while}\, e_1 \{ e_2 \}} : \Gamma_2 \Rightarrow$. The former is directly in the premise of T-While and the latter can be built by applying T-While to $\Sigma; \bullet; \Gamma_2; \Theta \vdash \boxed{e_1} : \texttt{bool} \Rightarrow \Gamma_2$ and $\Sigma; \bullet; \Gamma_2; \Theta \vdash \boxed{e_2} : \texttt{unit} \Rightarrow \Gamma_2$, both from the premise of our original T-While. Finally, we need to show $\Sigma; \bullet; \Gamma_2; \Theta \vdash \boxed{()} : \texttt{unit} \Rightarrow \Gamma_2$, which is immediate from T-Unit.
$\bullet; \Gamma_2; \Theta \vdash^{+} \texttt{unit} \rightsquigarrow \texttt{unit} \dashv \Gamma_2$	Immediate by RR-Refl.
$\exists \Gamma_o . \Gamma_2 \Cup \Gamma_o = \Gamma_2$	$\Gamma_o = \Gamma_2$

Case T-ForArray:

From premise:

$$\frac{\begin{array}{c}\Sigma;\Delta;\Gamma;\Theta \vdash \boxed{e_1} : [\tau^{\text{SI}};\, n] \Rightarrow \Gamma_1 \qquad \forall r \in \text{free-regions}(\tau^{\text{SI}}).\ \Gamma_1 \vdash r\ \texttt{rnrb} \\ \Sigma;\Delta;\Gamma_1,\ x : \tau^{\text{SI}};\Theta \vdash \boxed{e_2} : \texttt{unit} \Rightarrow \Gamma_1,\ x : \tau^{\text{SD}}\end{array}}{\Sigma;\Delta;\Gamma;\Theta \vdash \boxed{\texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \}} : \texttt{unit} \Rightarrow \Gamma_1}\ \text{T-ForArray}$$

Since $e = \texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \}$, by inspection of the reduction rules, we know that e steps with the following rule:

$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{for}\ x\ \texttt{in}\ [v_0,\ \ldots,\ v_n]\ \{\ e\ \}}) \rightarrow (\sigma,\ x \mapsto v_0;\ \boxed{\texttt{shift}\ e;\ \texttt{for}\ x\ \texttt{in}\ [v_1,\ \ldots,\ v_n]\ \{\ e\ \}})}\ \text{E-ForArray}$$

$$\frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{for}\ x\ \texttt{in}\ []\ \{\ e\ \}}) \rightarrow (\sigma;\ \boxed{()})}\ \text{E-ForEmptyArray}$$

Then, for each possible rule, we'll pick Γ_i separately. The cases proceed as follows:
For E-ForArray, we pick Γ_i to be $\boxed{\Gamma_1,\ x : \tau^{\text{SI}}}$, and need to show:

$\Sigma \vdash \sigma, x \mapsto v_0 : \Gamma_1, x : \tau^{\text{SI}}$	Applying Lemma E.15 to the typing derivation (from T-ForArray) for e_1 (which we know is a value from E-ForArray) gives us $\Sigma; \bullet \vdash \Gamma \rhd \Gamma_1$. Then, we can apply Lemma E.27 to get $\Sigma \vdash \sigma : \Gamma_1$. Applying Lemma E.26 to the derivation $\Sigma; \bullet; \Gamma; \Theta \vdash \boxed{[v_0, \ldots, v_n]} : [\tau^{\text{SI}}; n] \Rightarrow \Gamma_1$ gives us $\Sigma; \bullet; \Gamma_1; \Theta \vdash \boxed{[v_0, \ldots, v_n]} : [\tau^{\text{SI}}; n] \Rightarrow \Gamma_1$. Then, using inversion (of T-Array), we get $\Sigma; \bullet; \Gamma_1; \Theta \vdash \boxed{v_0} : \tau^{\text{SI}} \Rightarrow \Gamma_1$. Finally, applying Lemma E.35 to $\Sigma \vdash \sigma : \Gamma_1$ and $\Sigma; \bullet; \Gamma_1; \Theta \vdash \boxed{v_0} : \tau^{\text{SI}} \Rightarrow \Gamma_1$ gives us $\Sigma \vdash \sigma, x \mapsto v_0 : \Gamma_1, x : \tau^{\text{SI}}$.
$\Sigma; \Gamma_1, x : \tau^{\text{SI}} \vdash \overline{v} : \Theta$	Inverting WF-Temporaries gives us $\forall i \in 1 \ldots n.\ \Sigma; \bullet; \Gamma; \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma$. Applying Lemma E.15 to the typing derivation (from T-ForArray) for e_1 (which we know is a value from E-ForArray) gives us $\Sigma; \bullet \vdash \Gamma \rhd \Gamma_1$. Thus, we can apply Lemma E.25 to each of the typing judgments to get $\forall i \in 1 \ldots n.\ \Sigma; \bullet; \Gamma_1; \bullet \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_1$. Applying Lemma E.34 to each of the value typing judgments and $\forall r \in \text{free-regions}(\tau^{\text{SI}}).\ \Gamma_1 \vdash r$ rnrb (from the premise of T-ForArray) gives us $\forall i \in 1 \ldots n.\ \Sigma; \bullet; \Gamma_1, x : \tau^{\text{SI}}; \bullet \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_1, x : \tau^{\text{SI}}$.
$\Sigma; \bullet; \Gamma_1, x : \tau^{\text{SI}}; \Theta \vdash \boxed{e'} : \texttt{unit} \Rightarrow \Gamma_1$	We need to build a derivation for the expression $\texttt{shift}\ e;\ \texttt{for}\ x\ \texttt{in}\ [v_1, \ldots, v_n]\ \{\ e\ \}$. The bottom of this derivation will be T-Seq which requires us to show that $\Sigma; \bullet; \Gamma_1, x : \tau^{\text{SI}}; \Theta \vdash \boxed{\texttt{shift}\ e} : \texttt{unit} \Rightarrow \Gamma_1$ and that $\Sigma; \bullet; \Gamma_1; \Theta \vdash \boxed{\texttt{for}\ x\ \texttt{in}\ [v_1, \ldots, v_n]\ \{\ e\ \}} : \texttt{unit} \Rightarrow \Gamma_1$. To show $\Sigma; \bullet; \Gamma_1, x : \tau^{\text{SI}}; \Theta \vdash \boxed{\texttt{shift}\ e} : \texttt{unit} \Rightarrow \Gamma_1$, we apply T-Shift to $\Sigma; \bullet; \Gamma_1, x : \tau^{\text{SI}}; \Theta \vdash \boxed{e} : \texttt{unit} \Rightarrow \Gamma_1, x : \tau^{\text{SD}}$ (from the premise of T-ForArray). To show $\Sigma; \bullet; \Gamma_1; \Theta \vdash \boxed{\texttt{for}\ x\ \texttt{in}\ [v_1, \ldots, v_n]\ \{\ e\ \}} : \texttt{unit} \Rightarrow \Gamma_1$, we apply Lemma E.26 to the derivation $\Sigma; \bullet; \Gamma; \Theta \vdash \boxed{[v_0, \ldots, v_n]} : [\tau^{\text{SI}}; n] \Rightarrow \Gamma_1$ to get $\Sigma; \bullet; \Gamma_1; \Theta \vdash \boxed{[v_0, \ldots, v_n]} : [\tau^{\text{SI}}; n] \Rightarrow \Gamma_1$. Then, we rewrite the derivation (inverting and then reapply T-Array) to exclude v_0 giving us $\Sigma; \bullet; \Gamma_1; \Theta \vdash \boxed{[v_1, \ldots, v_n]} : [\tau^{\text{SI}}; n-1] \Rightarrow \Gamma_1$. Finally, we apply T-ForArray using this combined with $\Sigma; \bullet; \Gamma_1; \Theta \vdash \boxed{e} : \texttt{unit} \Rightarrow \Gamma_1$ (from the premise of T-ForArray).
$\bullet; \Gamma_1; \Theta \vdash^+ \texttt{unit} \rightsquigarrow \texttt{unit} \dashv \Gamma_1$	Immediate by RR-Refl.
$\exists \Gamma_o. \Gamma_1 \Cup \Gamma_o = \Gamma_1$	$\Gamma_o = \Gamma_1$

For E-ForEmptyArray, we pick Γ_i to be $\boxed{\Gamma}$, and need to show:

$\Sigma \vdash \sigma : \Gamma$	Immediate from our premise.
$\Sigma;\ \Gamma \vdash \overline{v} : \Theta$	Immediate from our premise.
$\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{(\,)} : \texttt{unit} \Rightarrow \Gamma$	Immediate by T-Unit.
$\bullet;\ \Gamma;\ \Theta \vdash^{+} \texttt{unit} \rightsquigarrow \texttt{unit} \dashv \Gamma$	Immediate by RR-Refl.
$\exists \Gamma_o . \Gamma \uplus \Gamma_o = \Gamma$	$\Gamma_o = \Gamma$

Case T-ForSlice:

From premise:

$$\textsc{T-ForSlice}\quad \frac{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_1} : \&\rho\ \omega\ [\tau^{\text{SI}}] \Rightarrow \Gamma_1 \qquad \forall r \in \text{free-regions}(\&\rho\ \omega\ \tau^{\text{SI}}).\ \Gamma_1 \vdash r\ \texttt{rnrb} \qquad \Sigma;\ \Delta;\ \Gamma_1,\ x\ :\ \&\rho\ \omega\ \tau^{\text{SI}};\ \Theta \vdash \boxed{e_2} : \texttt{unit} \Rightarrow \Gamma_1,\ x\ :\ \tau_1^{\text{SX}}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \}} : \texttt{unit} \Rightarrow \Gamma_2}$$

Since $e = \texttt{for}\ x\ \texttt{in}\ e_1\ \{\ e_2\ \}$, by inspection of the reduction rules, we know that e steps with the following rule:

$$\textsc{E-ForSlice}\quad \frac{\sigma \vdash \mathcal{R} \Downarrow \_ \mapsto \_[[v_1,\ \dots,\ v_i,\ \dots,\ v_j,\ \dots,\ v_n]] \qquad i < j \qquad i' = i + 1}{\Sigma \vdash (\sigma;\ \boxed{\texttt{for}\ x\ \texttt{in}\ \texttt{ptr}\ \mathcal{R}[i..j]\ \{\ e\ \}}) \rightarrow (\sigma,\ x \mapsto \texttt{ptr}\ \mathcal{R}[i];\ \boxed{\texttt{shift}\ e;\ \texttt{for}\ x\ \texttt{in}\ \texttt{ptr}\ \mathcal{R}[i'..j]\ \{\ e\ \}})}$$

$$\textsc{E-ForEmptySlice}\quad \frac{}{\Sigma \vdash (\sigma;\ \boxed{\texttt{for}\ x\ \texttt{in}\ \texttt{ptr}\ \pi[n..n]\ \{\ e\ \}}) \rightarrow (\sigma;\ \boxed{(\,)})}$$

Then, for each possible rule, we'll pick Γ_i separately. The cases proceed as follows:
For E-ForSlice, we pick Γ_i to be $\boxed{\Gamma_1,\ x\ :\ \&r\ \omega\ \tau^{\text{SI}}}$, and need to show:

$\Sigma \vdash \sigma, x \mapsto \texttt{ptr}\,\mathcal{R}[n_1] : \Gamma_1, x : \&r\ \omega\ \tau^{\text{SI}}$	Applying Lemma E.15 to the typing derivation (from T-ForSlice) for e_1 (which we know is a value from E-ForSlice) gives us $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma_1$. Then, we can apply Lemma E.27 to get $\Sigma \vdash \sigma : \Gamma_1$. Applying Lemma E.26 to the derivation $\Sigma; \bullet; \Gamma; \Theta \vdash \boxed{\texttt{ptr}\,\mathcal{R}[i..j]} : \&r\ \omega\ [\tau^{\text{SI}}] \Rightarrow \Gamma_1$ gives us $\Sigma; \bullet; \Gamma_1; \Theta \vdash \boxed{\texttt{ptr}\,\mathcal{R}[i..j]} : \&r\ \omega\ [\tau^{\text{SI}}] \Rightarrow \Gamma_1$. Then, using inversion (on T-Pointer), we get $\Sigma; \Gamma_1 \vdash \mathcal{R}[i..j] : \tau^{\text{XI}}$ (where $\tau^{\text{XI}} = [\tau^{\text{SI}}; n]$ or $[\tau^{\text{SI}}]$) and ${}^{\omega}\pi \in \Gamma_1(r)$ (where $\mathcal{R} = \mathcal{R}^{\square}[\pi]$). We can invert WF-RefSliceArray or WF-RefSliceSlice (based on τ^{XI}) for $\Sigma; \Gamma_1 \vdash \mathcal{R}[i..j] : \tau^{\text{XI}}$ and then apply WF-RefIndexArray or WF-RefIndexSlice appropriately to get $\Sigma; \Gamma_1 \vdash \mathcal{R}[i] : \tau^{\text{XI}}$. We can then use T-Pointer to get $\Sigma; \bullet; \Gamma_1; \Theta \vdash \boxed{\texttt{ptr}\,\mathcal{R}[n_1]} : \&r\ \omega\ \tau^{\text{SI}} \Rightarrow \Gamma_1$. Finally, applying Lemma E.35 to $\Sigma \vdash \sigma : \Gamma_1$ and $\Sigma; \bullet; \Gamma_1; \Theta \vdash \boxed{\texttt{ptr}\,\mathcal{R}[n_1]} : \&r\ \omega\ \tau^{\text{SI}} \Rightarrow \Gamma_1$ gives us $\Sigma \vdash \sigma, x \mapsto \texttt{ptr}\,\mathcal{R}[n_1] : \Gamma_1, x : \&r\ \omega\ \tau^{\text{SI}}$.
$\Sigma; \Gamma_1, x : \&r\ \omega\ \tau^{\text{SI}} \vdash \overline{v} : \Theta$	Inverting WF-Temporaries gives us $\forall i \in 1 \ldots n.\ \Sigma; \bullet; \Gamma; \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma$. Applying Lemma E.15 to the typing derivation (from T-ForSlice) for e_1 (which we know is a value from E-ForSlice) gives us $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma_1$. Thus, we can apply Lemma E.25 to each of the typing judgments to get $\forall i \in 1 \ldots n.\ \Sigma; \bullet; \Gamma_1; \bullet \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_1$. Applying Lemma E.34 to each of the value typing judgments and $\forall r \in \text{free-regions}(\&\rho\ \omega\ \tau^{\text{SI}})$. $\Gamma_1 \vdash r\ \texttt{rnrb}$ (from the premise of T-ForSlice) gives us $\forall i \in 1 \ldots n.\ \Sigma; \bullet; \Gamma_1, x : \&r\ \omega\ \tau^{\text{SI}}; \bullet \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_1, x : \&r\ \omega\ \tau^{\text{SI}}$.
$\Sigma; \bullet; \Gamma_1, x : \&r\ \omega\ \tau^{\text{SI}}; \Theta \vdash \boxed{e'} : \texttt{unit} \Rightarrow \Gamma_1$	We need to build a derivation for the expression $\texttt{shift}\ e$; $\texttt{for}\ x\ \texttt{in}\ \texttt{ptr}\,\mathcal{R}[n_1'..n_2]\ \{\ e\ \}$. The bottom of this derivation will be T-Seq which requires us to show that $\Sigma; \bullet; \Gamma_1, x : \tau^{\text{SI}}; \Theta \vdash \boxed{\texttt{shift}\ e} : \texttt{unit} \Rightarrow \Gamma_1$ and that $\Sigma; \bullet; \Gamma_1; \Theta \vdash \boxed{\texttt{for}\ x\ \texttt{in}\ \texttt{ptr}\,\mathcal{R}[n_1'..n_2]\ \{\ e\ \}} : \texttt{unit} \Rightarrow \Gamma_1$. To show $\Sigma; \bullet; \Gamma_1, x : \tau^{\text{SI}}; \Theta \vdash \boxed{\texttt{shift}\ e} : \texttt{unit} \Rightarrow \Gamma_1$, we apply T-Shift to $\Sigma; \bullet; \Gamma_1, x : \tau^{\text{SI}}; \Theta \vdash \boxed{e} : \texttt{unit} \Rightarrow \Gamma_1, x : \tau^{\text{SD}}$ (from the premise of T-ForSlice). To show $\Sigma; \bullet; \Gamma_1; \Theta \vdash \boxed{\texttt{for}\ x\ \texttt{in}\ \texttt{ptr}\,\mathcal{R}[n_1'..n_2]\ \{\ e\ \}} : \texttt{unit} \Rightarrow \Gamma_1$, we apply Lemma E.26 to the derivation $\Sigma; \bullet; \Gamma; \Theta \vdash \boxed{\texttt{ptr}\,\mathcal{R}[n_1..n_2]} : \&r\ \omega\ [\tau^{\text{SI}}] \Rightarrow \Gamma_1$ to get $\Sigma; \bullet; \Gamma_1; \Theta \vdash \boxed{\texttt{ptr}\,\mathcal{R}[n_1..n_2]} : \&r\ \omega\ [\tau^{\text{SI}}] \Rightarrow \Gamma_1$. Then, we rewrite the derivation (inverting and then reapply T-Pointer) to increment n_1 to n_1' giving us $\Sigma; \bullet; \Gamma_1; \Theta \vdash \boxed{\texttt{ptr}\,\mathcal{R}[n_1'..n_2]} : \&r\ \omega\ [\tau^{\text{SI}}] \Rightarrow \Gamma_1$. Finally, we apply T-ForSlice using this combined with $\Sigma; \bullet; \Gamma_1; \Theta \vdash \boxed{e} : \texttt{unit} \Rightarrow \Gamma_1$ (from the premise of T-ForSlice).
$\bullet; \Gamma_1; \Theta \vdash^{+} \texttt{unit} \rightsquigarrow \texttt{unit} \dashv \Gamma_1$	Immediate by RR-Refl.
$\exists \Gamma_o . \Gamma_1 \Cup \Gamma_o = \Gamma_1$	$\Gamma_o = \Gamma_1$

For E-ForEmptySlice, we pick Γ_i to be $\boxed{\Gamma}$, and need to show:

$\Sigma \vdash \sigma : \Gamma$	Immediate from our premise.
$\Sigma;\ \Gamma \vdash \overline{v} : \Theta$	Immediate from our premise.
$\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{()} : \texttt{unit} \Rightarrow \Gamma$	Immediate by T-Unit.
$\bullet;\ \Gamma;\ \Theta \vdash^{+} \texttt{unit} \rightsquigarrow \texttt{unit} \dashv \Gamma$	Immediate by RR-Refl.
$\exists \Gamma_o . \Gamma \Cup \Gamma_o = \Gamma$	$\Gamma_o = \Gamma$

Case T-AppFunction:

From premise:

$$\dfrac{\begin{array}{c}\text{T-AppFunction} \\ \overline{\Sigma;\ \Delta;\ \Gamma \vdash \Phi} \qquad \overline{\Delta;\ \Gamma \vdash \rho} \qquad \overline{\Sigma;\ \Delta;\ \Gamma \vdash \tau^{\text{SI}}} \qquad \delta = \cdot\ \overline{[^{\Phi}/_{\varphi}]}\ \overline{[^{\rho}/_{\varrho}]}\ \overline{[^{\tau^{\text{SI}}}/_{\alpha}]} \\ \Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_f} : \forall\texttt{<}\overline{\varphi},\ \overline{\varrho},\ \overline{\alpha}\texttt{>}(\tau^{\text{SI}}_1,\ \ldots,\ \tau^{\text{SI}}_n) \ \rightarrow\ \tau^{\text{SI}}_f\ \texttt{where}\ \overline{\varrho_1 : \varrho_2} \Rightarrow \Gamma_0 \\ \overline{\forall i \in \{\,1\ \ldots\ n\,\}.\ \Sigma;\ \Delta;\ \Gamma_{i-1};\ \Theta,\ \delta(\tau^{\text{SI}}_1)\ \ldots\ \delta(\tau^{\text{SI}}_{i-1}) \vdash \boxed{e_i} : \delta(\tau^{\text{SI}}_i) \Rightarrow \Gamma_i} \\ \forall i \in \{\,1\ \ldots\ n\,\}.\ \forall r \in \text{free-regions}(\tau^{\text{SI}}_i).\ \Gamma'_n \vdash r\ \texttt{rnrb} \qquad \Delta;\ \Gamma_n;\ \Theta \vdash \varrho_2\ \overline{[^{\rho}/_{\varrho}]} :> \varrho_1\ \overline{[^{\rho}/_{\varrho}]} \dashv \Gamma_b \end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_f\texttt{::<}\overline{\Phi},\ \overline{\rho},\ \overline{\tau^{\text{SI}}}\texttt{>}(e_1,\ \ldots,\ e_n)} : \delta(\tau^{\text{SI}}_f) \Rightarrow \Gamma_b}$$

Since $e = e_f\texttt{::<}\overline{\Phi},\ \overline{\rho'},\ \overline{\tau^{\text{SI}}}\texttt{>}(e_1,\ \ldots,\ e_n)$, by inspection of the reduction rules, we know that e steps with the following rule:

$$\dfrac{\begin{array}{c}\text{E-AppFunction} \\ \Sigma(f) = \texttt{fn}\ f\texttt{<}\overline{\varphi},\ \overline{\varrho},\ \overline{\alpha}\texttt{>}(x_1 : \tau^{\text{S}}_1,\ \ldots,\ x_n : \tau^{\text{S}}_n)\ \rightarrow\ \tau^{\text{S}}_r\ \texttt{where}\ \overline{\varrho : \varrho'}\ \{\ e\ \}\end{array}}{\Sigma \vdash (\sigma;\ \boxed{f\texttt{::<}\overline{\Phi},\ \overline{r'},\ \overline{\tau^{\text{S}}}\texttt{>}(v_1,\ \ldots,\ v_n)}) \rightarrow (\sigma \natural\, x_1 \mapsto v_1,\ \ldots,\ x_n \mapsto v_n;\ \boxed{\texttt{framed}\ e[^{\overline{\Phi}}/_{\overline{\varphi}}][^{\overline{r'}}/_{\overline{\varrho}}][^{\overline{\tau^{\text{S}}}}/_{\overline{\alpha}}]})}$$

We then pick Γ_i to be $\boxed{\Gamma_b \natural\, x_1\ :\ \delta(\tau^{\text{SI}}_1),\ \ldots,\ x_n\ :\ \delta(\tau^{\text{SI}}_n)}$, and need to show:

$\Sigma \vdash \sigma' : \Gamma_i$	Applying Lemma E.15 to the derivation for v_f gives us $\Sigma; \bullet \vdash \Gamma \rhd \Gamma_0$. Then, applying Lemma E.15 to the derivations for every v_i gives us $\forall i \in \{ 1 \dots n \}. \Sigma; \bullet \vdash \Gamma_{i-1} \rhd \Gamma_i$. Inversion on $\bullet; \Gamma_n; \Theta \vdash \overline{\varrho_2\,[\overline{\rho/\varrho}] :> \varrho_1\,[\overline{\rho/\varrho}]} \dashv \Gamma_b$ gives us a sequence of outlives relations with intermediate contexts. Applying Lemma E.21 to each of them and then combining the result by transitivity gives us $\Sigma; \bullet \vdash \Gamma_n \rhd \Gamma_b$. Combining both by transitivity, we have $\Sigma; \bullet \vdash \Gamma \rhd \Gamma_b$. We can then apply Lemma E.27 with $\Sigma \vdash \sigma : \Gamma$ to get $\Sigma \vdash \sigma : \Gamma_b$. We can apply WF-StackFrame to get $\Sigma \vdash \sigma \natural \bullet : \Gamma_b \natural \bullet$. Then, we repeatedly apply Lemma E.35 to the derivations for the arguments $(v_1 \dots v_n)$ to get $\Sigma \vdash \sigma \natural x_1 \mapsto v_1, \dots, x_n \mapsto v_n : \Gamma \natural x_1 : \delta(\tau_1^{\text{SI}}), \dots, x_n : \delta(\tau_n^{\text{SI}})$.
$\Sigma; \Gamma_i \vdash \overline{v} : \Theta$	Inverting WF-Temporaries gives us $\forall i \in 1 \dots n. \Sigma; \bullet; \Gamma; \tau_1^{\text{SI}}, \dots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma$. Applying Lemma E.15 to the derivation for v_f gives us $\Sigma; \bullet \vdash \Gamma \rhd \Gamma_0$. Then, applying Lemma E.15 to the derivations for every v_i gives us $\forall i \in \{ 1 \dots n \}. \Sigma; \bullet \vdash \Gamma_{i-1} \rhd \Gamma_i$. Inversion on $\bullet; \Gamma_n; \Theta \vdash \overline{\varrho_2\,[\overline{\rho/\varrho}] :> \varrho_1\,[\overline{\rho/\varrho}]} \dashv \Gamma_b$ gives us a sequence of outlives relations with intermediate contexts. Applying Lemma E.21 to each of them and then combining the result by transitivity gives us $\Sigma; \bullet \vdash \Gamma_n \rhd \Gamma_b$. Combining both by transitivity, we have $\Sigma; \bullet \vdash \Gamma \rhd \Gamma_b$. Thus, we can apply Lemma E.25 to each of the typing judgments to get $\forall i \in 1 \dots n. \Sigma; \bullet; \Gamma_b; \bullet \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_b$. We also immediately have that adding a new empty frame leaves all the typing derivations good since it doesn't actually bind any new names on its own. Thus, we have $\forall i \in 1 \dots n. \Sigma; \bullet; \Gamma_b \natural \bullet; \bullet \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_b \natural \bullet$. Repeatedly applying Lemma E.34 to each of the value typing judgments gives us $\forall i \in 1 \dots n. \Sigma; \bullet; \Gamma_b \natural x_1 : \delta(\tau_1^{\text{SI}}), \dots, x_n : \delta(\tau_n^{\text{SI}}); \bullet \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_b \natural x_1 : \delta(\tau_1^{\text{SI}}), \dots, x_n : \delta(\tau_n^{\text{SI}})$.
$\Sigma; \bullet; \Gamma_i; \Theta \vdash \boxed{e'} : \delta(\tau_f^{\text{SI}}) \Rightarrow \Gamma_b$	Applying Lemma E.43 with $\vdash \Sigma; \bullet; \Gamma_b; \Theta$ and $\Sigma(f) = \texttt{fn}\ f\texttt{<}\overline{\varphi}, \overline{\varrho}, \overline{\alpha}\texttt{>}(x_1 : \tau_1^{\text{SI}}, \dots, x_n : \tau_n^{\text{SI}}) \to \tau_r^{\text{SI}}\ \texttt{where}\ \overline{\varrho_1 : \varrho_2}\ \{ e \}$ (from the premise of on E-AppFunction) gives us $\Sigma; \overline{\varphi : \mathsf{FRM}}, \overline{\varrho : \mathsf{RGN}}, \overline{\varrho_1 :> \varrho_2}, \overline{\alpha : \star}; \Gamma_b \natural x_1 : \tau_1^{\text{SI}}, \dots, x_n : \tau_n^{\text{SI}}; \Theta \vdash \boxed{\texttt{framed}\ e} : \tau_f^{\text{SI}} \Rightarrow \Gamma_b$. We then repeatedly apply Lemma E.2 for all of our type, region, and environment variables to get $\Sigma; \bullet; \Gamma_b \natural x_1 : \delta(\tau_1^{\text{SI}}), \dots, x_n : \delta(\tau_n^{\text{SI}}); \Theta \vdash \boxed{\texttt{framed}\ e} : \delta(\tau_f^{\text{SI}}) \Rightarrow \Gamma_b$.
$\bullet; \Gamma_b; \Theta \vdash^{+} \tau' \rightsquigarrow \tau' \dashv \Gamma_b$	Immediate by RR-Refl.
$\exists \Gamma_o. \Gamma_b \uplus \Gamma_o = \Gamma_b$	$\Gamma_o = \Gamma_b$

Case T-AppClosure:

From premise:

T-AppClosure

$$\frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_f} : \forall\texttt{<>}(\tau_1^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}}) \xrightarrow{\Phi_c} \tau_f^{\text{SI}} \Rightarrow \Gamma_0 \\ \forall i \in \{\ 1\ \ldots\ n\ \}.\ \Sigma;\ \Delta;\ \Gamma_{i-1};\ \Theta,\ \tau_1^{\text{SI}}\ \ldots\ \tau_{i-1}^{\text{SI}} \vdash \boxed{e_i} : \tau_{i'}^{\text{SI}} \Rightarrow \Gamma_i \qquad \Delta;\ \Gamma_i;\ \Theta \vdash^{\boxplus} \tau_{i'}^{\text{SI}} \rightsquigarrow \tau_i^{\text{SI}} \dashv \Gamma_i' \\ \forall i \in \{\ 1\ \ldots\ n\ \}.\ \forall r \in \text{free-regions}(\tau_i^{\text{SI}}).\ \Gamma_n' \vdash r\ \texttt{rnrb}\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e_f(e_1,\ \ldots,\ e_n)} : \tau_f^{\text{SI}} \Rightarrow \Gamma_n'}$$

Since $e = e_f(e_1,\ \ldots,\ e_n)$, by inspection of the reduction rules, we know that e steps with the following rule:

E-AppClosure

$$\frac{v_f = \langle \varsigma_c,\ \ |x_1 : \tau_1^{\text{S}},\ \ldots,\ x_n : \tau_n^{\text{S}}|\ \rightarrow\ \tau_r^{\text{S}}\ \{\ e\ \}\ \rangle}{\Sigma \vdash (\sigma;\ \boxed{v_f(v_1,\ \ldots,\ v_n)}) \rightarrow (\sigma \natural \varsigma_c,\ x_1 \mapsto v_1,\ \ldots,\ x_n \mapsto v_n;\ \boxed{\texttt{framed}\ e})}$$

We then pick Γ_i to be $\boxed{\Gamma_n' \natural \mathcal{F}_c,\ x_1\ :\ \tau_1^{\text{SI}},\ \ldots,\ x_n\ :\ \tau_n^{\text{SI}}}$, and need to show:

$\Sigma \vdash \sigma' : \Gamma_i$	Applying Lemma E.15 to the derivation for v_f gives us $\Sigma; \bullet \vdash \Gamma \rhd \Gamma_0$. We can apply Lemma E.27 with $\Sigma \vdash \sigma : \Gamma$ to get $\Sigma \vdash \sigma : \Gamma_0$. Then, for each pair of value typing judgment and region rewriting, we follow the same pattern we will describe in terms of an arbitrary index i. First, we apply Lemma E.15 to the derivation for v_i to get $\Sigma; \bullet \vdash \Gamma_{i-1} \rhd \Gamma_i$. Then, applying Lemma E.27 gives us $\Sigma \vdash \sigma : \Gamma_i$. Then, we can apply Lemma E.14 to $\bullet;\ \Gamma_i;\ \Theta \vdash^{\boxplus} \tau^{\text{SI}}_{i'} \rightsquigarrow \tau^{\text{SI}}_{i} \dashv \Gamma'_i$ to get $\Sigma \vdash \sigma : \Gamma'_i$. Going through this for all the values in sequence gives us $\Sigma \vdash \sigma : \Gamma'_n$ Then, inversion of T-ClosureValue for the typing derivation for v_f gives us $\Sigma;\ \Gamma \vdash \varsigma_c : \mathcal{F}_c$. We can then invert WF-Frame here to get $\text{dom}(\varsigma) = \text{dom}(\mathcal{F}_c)\|_x\ \forall x \in \text{dom}(\varsigma).\ \Sigma; \bullet; \Gamma \natural \mathcal{F}_c; \Theta \vdash \boxed{\varsigma(x)} : \mathcal{F}_c(x) \Rightarrow \Gamma \natural \mathcal{F}_c$ which we can then use with $\Sigma \vdash \sigma : \Gamma_n$ in WF-StackFrame to get $\Sigma \vdash \sigma \natural \varsigma_c : \Gamma'_n \natural \mathcal{F}_c$. Finally, we repeatedly apply Lemma E.35 to the derivations for the arguments $(v_1\ \dots\ v_n)$ to get $\Sigma \vdash \sigma' : \Gamma_i$.
$\Sigma;\ \Gamma_i \vdash \overline{v'} : \Theta$	Inverting WF-Temporaries gives us $\forall j \in 1 \dots n.\ \Sigma; \bullet; \Gamma; \tau^{\text{SI}}_1, \dots, \tau^{\text{SI}}_{i-1} \vdash \boxed{v'_j} : \tau^{\text{SI}}_j \Rightarrow \Gamma$. Applying Lemma E.15 to the derivation for v_f gives us $\Sigma; \bullet \vdash \Gamma \rhd \Gamma_0$. For each of $\forall j \in 1 \dots n.\ \Sigma; \bullet; \Gamma; \bullet \vdash \boxed{v'_j} : \tau^{\text{SI}}_i \Rightarrow \Gamma$, we apply Lemma E.25 to get $\forall j \in 1 \dots n.\ \Sigma; \bullet; \Gamma_0; \bullet \vdash \boxed{v'_j} : \tau^{\text{SI}}_j \Rightarrow \Gamma_0$ From here, we have an alternating pattern of lemmas to apply to deal with the interleaved value typing and then region rewriting in the premise of T-AppClosure. For each value v_i in T-AppClosure, we'll first apply Lemma E.15 to get $\Sigma; \bullet \vdash \Gamma_{i-1} \rhd \Gamma_i$. Then, we'll apply Lemma E.25 to $\forall j \in 1 \dots n.\ \Sigma; \bullet; \Gamma_{i-1}; \bullet \vdash \boxed{v'_j} : \tau^{\text{SI}}_j \Rightarrow \Gamma_{i-1}$ to get $\forall j \in 1 \dots n.\ \Sigma; \bullet; \Gamma_i; \bullet \vdash \boxed{v'_j} : \tau^{\text{SI}}_j \Rightarrow \Gamma_i$. Then, we apply Lemma E.13 to each of these along with the rewriting $\bullet;\ \Gamma_i;\ \Theta \vdash^{\boxplus} \tau^{\text{SI}}_{i'} \rightsquigarrow \tau^{\text{SI}}_i \dashv \Gamma'_i$ to get $\forall j \in 1 \dots n.\ \Sigma; \bullet; \Gamma'_i; \bullet \vdash \boxed{v'_j} : \tau^{\text{SI}}_j \Rightarrow \Gamma'_i$. After going through this for each value v_i, we have $\forall j \in 1 \dots n.\ \Sigma; \bullet; \Gamma'_n; \bullet \vdash \boxed{v'_j} : \tau^{\text{SI}}_j \Rightarrow \Gamma'_n$. We also immediately have that adding a new empty frame leaves all the typing derivations good since it doesn't actually bind any new names on its own. Thus, we have $\forall i \in 1 \dots n.\ \Sigma; \bullet; \Gamma'_n \natural \bullet; \bullet \vdash \boxed{v_i} : \tau^{\text{SI}}_i \Rightarrow \Gamma'_n \natural \bullet$. Repeatedly applying Lemma E.34 to each of the value typing judgments and each of $\forall i \in \{1 \dots n\}.\ \forall r \in \text{free-regions}(\tau^{\text{SI}}_i).\ \Gamma_1 \vdash r\ \mathsf{rnrb}$ (from the premise of T-App) gives us $\forall i \in 1 \dots n.\ \Sigma; \bullet; \Gamma'_n \natural \mathcal{F}_c, x_1 : \tau^{\text{SI}}_1, \dots, x_n : \tau^{\text{SI}}_n; \bullet \vdash \boxed{v_i} : \tau^{\text{SI}}_i \Rightarrow \Gamma'_n \natural \mathcal{F}_c, x_1 : \tau^{\text{SI}}_1, \dots, x_n : \tau^{\text{SI}}_n$.

$\Sigma; \bullet; \Gamma_i; \Theta \vdash \boxed{e'} : \tau_f^{\text{SI}} \Rightarrow \Gamma_n'$	Consider first the typing derivation for v_f of $\Sigma; \bullet; \Gamma; \Theta \vdash \boxed{v_f} : \forall\texttt{<>}(\tau_1^{\text{SI}}, \dots, \tau_n^{\text{SI}}) \xrightarrow{\Phi_c} \tau_f^{\text{SI}} \Rightarrow \Gamma_0'$. Applying Lemma E.26 gives us $\Sigma; \bullet; \Gamma_0'; \Theta \vdash \boxed{v_f} : \forall\texttt{<>}(\tau_1^{\text{SI}}, \dots, \tau_n^{\text{SI}}) \xrightarrow{\Phi_c} \tau_f^{\text{SI}} \Rightarrow \Gamma_0'$. Then, as in the previous cases, we can apply Lemma E.15, Lemma E.25, and Lemma E.13 in sequence starting with the above typing derivation with each of the typing derivations for the arguments v_i and their subsequent corresponding region rewriting derivation. At the end, this gives us $\Sigma; \bullet; \Gamma_n'; \Theta \vdash \boxed{v_f} : \forall\texttt{<>}(\tau_1^{\text{SI}}, \dots, \tau_n^{\text{SI}}) \xrightarrow{\Phi_c} \tau_f^{\text{SI}} \Rightarrow \Gamma_n'$ Then, inversion on T-ClosureValue on this typing derivation for v_f gives us free-vars$(e) \setminus \overline{x} = \overline{x_f} = \text{dom}(\mathcal{F}_c)\vert_x$, $\overline{r} = \overline{\text{free-regions}(\Gamma(x_f))}$, free-regions$(e) = \text{dom}(\mathcal{F}_c)\vert_r$, and $\Sigma; \bullet; \Gamma_n' \natural \mathcal{F}_c, x_1 : \tau_1^{\text{SI}}, \dots, x_n : \tau_n^{\text{SI}}; \Theta \vdash \boxed{e} : \tau_r^{\text{SI}} \Rightarrow \Gamma_n' \natural \mathcal{F}$. We can then apply T-Framed to get $\Sigma; \bullet; \Gamma_n \natural \mathcal{F}_c, x_1 : \tau_1^{\text{SI}}, \dots, x_n : \tau_n^{\text{SI}}; \Theta \vdash \boxed{\texttt{framed}\ e} : \tau_f^{\text{SI}} \Rightarrow \Gamma_n$.
$\bullet; \Gamma_n'; \Theta \vdash^+ \tau_f^{\text{SI}} \rightsquigarrow \tau_f^{\text{SI}} \dashv \Gamma_n'$	Immediate by RR-Refl.
$\exists \Gamma_o . \Gamma_n' \uplus \Gamma_o = \Gamma_n'$	$\Gamma_o = \Gamma_n'$

Case T-Function:

From premise:

$$\textsc{T-Function}\quad \frac{\Sigma(f) = \texttt{fn}\ f\texttt{<}\overline{\varphi}, \overline{\varrho}, \overline{\alpha}\texttt{>}(x_1 : \tau_1^{\text{SI}}, \dots, x_n : \tau_n^{\text{SI}}) \to \tau_r^{\text{SI}}\ \texttt{where}\ \overline{\varrho_1 : \varrho_2}\ \{\, e \,\}}{\Sigma; \Delta; \Gamma; \Theta \vdash \boxed{f} : \forall\texttt{<}\overline{\varphi}, \overline{\varrho}, \overline{\alpha}\texttt{>}(\tau_1^{\text{SI}}, \dots, \tau_n^{\text{SI}}) \to \tau_r^{\text{SI}}\ \texttt{where}\ \overline{\varrho_1 : \varrho_2} \Rightarrow \Gamma}$$

Since $e = \forall\texttt{<}\overline{\rho}, \overline{\alpha}\texttt{>}(\tau_1^{\text{SI}}, \dots, \tau_n^{\text{SI}}) \to \tau_r^{\text{SI}}$, by inspection of the reduction rules, we know that e steps with the following rule:

$$\textsc{E-Function}\quad \frac{\Sigma(f) = \texttt{fn}\ f\texttt{<}\overline{\varphi}, \overline{\varrho}, \overline{\alpha}\texttt{>}(x_1 : \tau_1^{\text{S}}, \dots, x_n : \tau_n^{\text{S}}) \to \tau_r^{\text{S}}\ \texttt{where}\ \overline{\varrho : \varrho'}\ \{\, e \,\}}{\Sigma \vdash (\sigma; \boxed{f}) \to (\sigma; \boxed{\langle \bullet,\ \texttt{forall<}\overline{\varphi}, \overline{\varrho}, \overline{\alpha}\texttt{>}\ \vert x_1 : \tau_1^{\text{S}}, \dots, x_n : \tau_n^{\text{S}} \vert \to \tau_r^{\text{S}}\ \{\, e \,\} \rangle})}$$

We then pick Γ_i to be $\boxed{\Gamma}$, and need to show:

$\Sigma \vdash \sigma : \Gamma$	Immediate from our premise.
$\Sigma; \Gamma \vdash \overline{v} : \Theta$	Immediate from our premise.
$\Sigma; \bullet; \Gamma; \Theta \vdash \boxed{e'} : \tau' \Rightarrow \Gamma$	By T-ClosureValue since σ_c is empty, and we know that the body itself is well-typed as a consequence of inversion on WF-FunctionDefinition for f.
$\bullet; \Gamma; \Theta \vdash^+ \tau' \rightsquigarrow \tau \dashv \Gamma$	Immediate by RR-Refl.
$\exists \Gamma_o . \Gamma \uplus \Gamma_o = \Gamma$	$\Gamma_o = \Gamma$

Case T-Closure:

From premise:

T-Closure

$$\frac{\begin{array}{c}\text{free-vars}(e) \setminus \overline{x} = \overline{x_f} \qquad \text{free-nc-vars}_\Gamma(e) \setminus \overline{x} = \overline{x_{nc}} \\ \overline{r} = \overline{\text{free-regions}(\Gamma(x_f))},\ (\text{free-regions}(e) \setminus (\overline{\text{free-regions}(\tau^{\text{SI}})}, \text{free-regions}(\tau_r^{\text{SI}}))) \\ \mathcal{F}_c = \overline{r \mapsto \Gamma(r)},\ \overline{x_f\ :\ \Gamma(x_f)} \qquad \forall r_p \in \bigcup_{i=1}^{n} \text{free-regions}(\tau_i^{\text{SI}}) \cup \text{free-regions}(\tau_r^{\text{SI}}).\ \Gamma(r_p) = \emptyset \\ \Sigma;\ \Delta;\ \Gamma[\overline{x_{nc} \mapsto \Gamma(x_{nc})^\dagger}] \natural \mathcal{F}_c,\ x_1\ :\ \tau_1^{\text{SI}},\ \ldots,\ x_n\ :\ \tau_n^{\text{SI}};\ \Theta \vdash \boxed{e} : \tau_r^{\text{SI}} \Rightarrow \Gamma' \natural \mathcal{F}\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{|x_1 : \tau_1^{\text{SI}},\ \ldots,\ x_n : \tau_n^{\text{SI}}| \ \rightarrow\ \tau_r^{\text{SI}}\ \{\ e\ \}} : (\tau_1^{\text{SI}},\ \ldots,\ \tau_n^{\text{SI}}) \overset{\mathcal{F}_c}{\rightarrow} \tau_r^{\text{SI}} \Rightarrow \Gamma'}$$

Since $e = \texttt{forall<}\overline{\varphi},\ \overline{\rho},\ \overline{\alpha}\texttt{>}\ |x_1 : \tau_1^{\text{SI}},\ \ldots,\ x_n : \tau_n^{\text{SI}}| \ \rightarrow\ \tau_r^{\text{SI}}\ \{\ e\ \}$, by inspection of the reduction rules, we know that e steps with the following rule:

E-Closure

$$\frac{\overline{x_f} = \text{free-vars}(e) \qquad \overline{x_{nc}} = \text{free-nc-vars}_\sigma(e) \qquad \varsigma_c = \sigma \mid_{\overline{x_f}}}{\Sigma \vdash (\sigma;\ \boxed{|x_1 : \tau_1^{\text{S}},\ \ldots,\ x_n : \tau_n^{\text{S}}| \ \rightarrow\ \tau_r^{\text{S}}\ \{\ e\ \}}) \rightarrow (\sigma[\overline{x_{nc} \mapsto \texttt{dead}}];\ \boxed{\langle \varsigma_c,\ |x_1 : \tau_1^{\text{S}},\ \ldots,\ x_n : \tau_n^{\text{S}}| \ \rightarrow\ \tau_r^{\text{S}}\ \{\ e\ \}\ \rangle})}$$

We then pick Γ_i to be $\boxed{\Gamma[\overline{x_{nc} \mapsto \Gamma(x_{nc})^\dagger}]}$, and need to show:

$\Sigma \vdash \sigma[\overline{x_{nc} \mapsto \texttt{dead}}] : \Gamma[\overline{x_{nc} \mapsto \Gamma(x_{nc})^\dagger}]$	Compared to Γ, we know that $\Gamma[\overline{x_{nc} \mapsto \Gamma(x_{nc})^\dagger}]$ has more things marked dead and no other changes. Thus, we can apply R-Env to get $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma[\overline{x_{nc} \mapsto \Gamma(x_{nc})^\dagger}]$. Then, we can apply Lemma E.27 to get $\Sigma \vdash \sigma : \Gamma[\overline{x_{nc} \mapsto \Gamma(x_{nc})^\dagger}]$. Since dead is good at every dead type τ^{SD} by T-Dead, we can then build a new derivation using that rule instead for every x_{nc} that is now at a dead type. This gives us $\Sigma \vdash \sigma[\overline{x_{nc} \mapsto \texttt{dead}}] : \Gamma[\overline{x_{nc} \mapsto \Gamma(x_{nc})^\dagger}]$.
$\Sigma;\ \Gamma[\overline{x_{nc} \mapsto \Gamma(x_{nc})^\dagger}] \vdash \overline{v} : \Theta$	Inverting WF-Temporaries gives us $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma;\ \tau_1^{\text{SI}},\ \ldots,\ \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma$. By R-Env, we have that $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma[\overline{x_{nc} \mapsto \Gamma(x_{nc})^\dagger}]$. Thus, we can apply Lemma E.25 to each of the typing judgments and then apply WF-Temporaries to get $\Sigma;\ \Gamma[\overline{x_{nc} \mapsto \Gamma(x_{nc})^\dagger}] \vdash \overline{v} : \Theta$.
$\Sigma;\ \bullet;\ \Gamma_i;\ \Theta \vdash \boxed{e'} : \tau' \Rightarrow \Gamma_i$	Immediate by inversion of T-Closure and application of T-ClosureValue. Note that they have identical premises.
$\bullet;\ \Gamma_i;\ \Theta \vdash^+ \tau' \rightsquigarrow \tau' \dashv \Gamma_i$	Immediate by RR-Refl.
$\exists \Gamma_o. \Gamma_i \Cup \Gamma_o = \Gamma_i$	$\Gamma_o = \Gamma_i$

Case T-Shift:

From premise:

$$\frac{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau^{\mathrm{SI}} \Rightarrow \Gamma',\ x\ :\ \tau^{\mathrm{SD}}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{shift}\ e} : \tau^{\mathrm{SI}} \Rightarrow \Gamma'}\ \textsc{T-Shift}$$

Since $e = \texttt{shift}\ e_i$, by inspection of the reduction rules, we know that e steps with the following rule:

$$\frac{}{\Sigma \vdash (\sigma,\ x \mapsto v';\ \boxed{\texttt{shift}\ v}) \rightarrow (\sigma;\ \boxed{v})}\ \textsc{E-Shift}$$

We then pick Γ_i to be $\boxed{\Gamma'}$, and need to show:

$\Sigma \vdash \sigma : \Gamma'$	By inversion of WF-StackFrame on $\Sigma \vdash \sigma,\ x \mapsto v' : \Gamma',\ x\ :\ \tau^{\mathrm{SD}}$, we get $\Sigma \vdash \sigma : \Gamma_i$, $\mathrm{dom}(\varsigma,\ x \mapsto v') = \mathrm{dom}(\mathcal{F},\ x\ :\ \tau^{\mathrm{SD}})\vert_x$, and $\forall x \in \mathrm{dom}(\sigma \natural \varsigma,\ x \mapsto v')$. $\Sigma;\ \bullet;\ \Gamma_i \natural \mathcal{F},\ x\ :\ \tau^{\mathrm{SD}};\ \Theta \vdash \boxed{(\sigma \natural \varsigma,\ x \mapsto v')(x)} : (\Gamma_i \natural \mathcal{F}, x : \tau^{\mathrm{SD}})(x) \Rightarrow \Gamma_i \natural \mathcal{F}, x : \tau^{\mathrm{SD}}$. Note that $\Gamma_i \natural \mathcal{F} = \Gamma'$. We can then immediately see that the above implies $\mathrm{dom}(\varsigma) = \mathrm{dom}(\mathcal{F})\vert_x$ and $\forall x \in \mathrm{dom}(\sigma \natural \varsigma,\ x \mapsto v')$. $\Sigma;\ \bullet;\ \Gamma_i \natural \mathcal{F};\ \Theta \vdash \boxed{(\sigma \natural \varsigma)(x)} : (\Gamma_i \natural \mathcal{F})(x) \Rightarrow \Gamma_i \natural \mathcal{F}$. Thus, we can apply WF-StackFrame to get $\Sigma \vdash \sigma : \Gamma_i \natural \varsigma$ which can be rewritten as $\Sigma \vdash \sigma : \Gamma'$.
$\Sigma;\ \Gamma' \vdash \overline{v} : \Theta$	Inverting WF-Temporaries gives us $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma;\ \tau_1^{\mathrm{SI}}, \ldots, \tau_{i-1}^{\mathrm{SI}} \vdash \boxed{v_i} : \tau_i^{\mathrm{SI}} \Rightarrow \Gamma$. Applying Lemma E.15 to the typing derivation (from T-Shift) for e (which we know is a value from E-Shift) gives us $\Sigma;\ \bullet \vdash \Gamma \triangleright \Gamma'$. Thus, we can apply Lemma E.25 to each of the typing judgments to get $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma';\ \tau_1^{\mathrm{SI}}, \ldots, \tau_{i-1}^{\mathrm{SI}} \vdash \boxed{v_i} : \tau_i^{\mathrm{SI}} \Rightarrow \Gamma'$. Finally, applying WF-Temporaries gives us $\Sigma;\ \Gamma' \vdash \overline{v} : \Theta$.
$\Sigma;\ \bullet;\ \Gamma';\ \Theta \vdash \boxed{v} : \tau^{\mathrm{SI}} \Rightarrow \Gamma'$	We know from E-Shift that e is a value v. Thus, we can apply Lemma E.26 to $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{v} : \tau^{\mathrm{SI}} \Rightarrow \Gamma',\ x\ :\ \tau^{\mathrm{SD}}$ to get $\Sigma;\ \bullet;\ \Gamma',\ x\ :\ \tau^{\mathrm{SD}};\ \Theta \vdash \boxed{v} : \tau^{\mathrm{SI}} \Rightarrow \Gamma',\ x\ :\ \tau^{\mathrm{SD}}$. We now wish to show that $\Sigma;\ \bullet;\ \Gamma';\ \Theta \vdash \boxed{v} : \tau^{\mathrm{SI}} \Rightarrow \Gamma'$. By inspecting the grammar of values and their typing rules, we know that the only values who depend on the context are pointers and closure values. But by inversion on $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{\texttt{shift}\ v} : \tau^{\mathrm{SI}} \Rightarrow \Gamma'$, we know that $\Sigma;\ \bullet;\ \Gamma' \vdash \tau^{\mathrm{SI}}$. Since the type is valid without the frame, we know that the values cannot depend on that frame. Thus, we can conclude $\Sigma;\ \bullet;\ \Gamma';\ \Theta \vdash \boxed{v} : \tau^{\mathrm{SI}} \Rightarrow \Gamma'$.
$\bullet;\ \Gamma';\ \Theta \vdash^{+} \tau^{\mathrm{SI}} \rightsquigarrow \tau^{\mathrm{SI}} \dashv \Gamma'$	Immediate by RR-Refl.
$\exists \Gamma_o . \Gamma' \Cup \Gamma_o = \Gamma'$	$\Gamma_o = \Gamma'$

Case T-Framed:

From premise:

$$\textsc{T-Framed}\quad \frac{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau^{\text{SI}} \Rightarrow \Gamma' \natural \mathcal{F}'}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\texttt{framed}\ e} : \tau^{\text{SI}} \Rightarrow \Gamma'}$$

Since $e = \texttt{framed}\ e_i$, by inspection of the reduction rules, we know that e steps with the following rule:

$$\textsc{E-Framed}\quad \frac{}{\Sigma \vdash (\sigma \natural \varsigma;\ \boxed{\texttt{framed}\ v}) \to (\sigma;\ \boxed{v})}$$

We then pick Γ_i to be $\boxed{\Gamma'}$, and need to show:

$\Sigma \vdash \sigma : \Gamma'$	Applying Lemma E.28 to $\Sigma \vdash \sigma \natural \varsigma : \Gamma' \natural \mathcal{F}$ gives us $\Sigma \vdash \sigma : \Gamma'$.
$\Sigma;\ \Gamma' \vdash \overline{v} : \Theta$	Inverting WF-Temporaries gives us $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma;\ \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma$. Applying Lemma E.15 to the typing derivation (from T-Framed) for e (which we know is a value from E-Framed) gives us $\Sigma;\ \bullet \vdash \Gamma \rhd \Gamma'$. Thus, we can apply Lemma E.25 to each of the typing judgments to get $\forall i \in 1 \ldots n.\ \Sigma;\ \bullet;\ \Gamma';\ \tau_1^{\text{SI}}, \ldots, \tau_{i-1}^{\text{SI}} \vdash \boxed{v_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma'$. Finally, applying WF-Temporaries gives us $\Sigma;\ \Gamma' \vdash \overline{v} : \Theta$.
$\Sigma;\ \bullet;\ \Gamma';\ \Theta \vdash \boxed{v} : \tau^{\text{SI}} \Rightarrow \Gamma'$	We know from E-Framed that e is a value v. Thus, we can apply Lemma E.26 to $\Sigma;\ \bullet;\ \Gamma \natural \mathcal{F};\ \Theta \vdash \boxed{v} : \tau^{\text{SI}} \Rightarrow \Gamma' \natural \mathcal{F}'$ to get $\Sigma;\ \bullet;\ \Gamma' \natural \mathcal{F}';\ \Theta \vdash \boxed{v} : \tau^{\text{SI}} \Rightarrow \Gamma' \natural \mathcal{F}'$. We now wish to show that $\Sigma;\ \bullet;\ \Gamma';\ \Theta \vdash \boxed{v} : \tau^{\text{SI}} \Rightarrow \Gamma'$. By inspecting the grammar of values and their typing rules, we know that the only values who depend on the context are pointers and closure values. But by inversion on $\Sigma;\ \bullet;\ \Gamma;\ \Theta \vdash \boxed{\texttt{framed}\ v} : \tau^{\text{SI}} \Rightarrow \Gamma'$, we know that $\Sigma;\ \bullet;\ \Gamma' \vdash \tau^{\text{SI}}$. Since the type is valid without the frame, we know that the values cannot depend on that frame. Thus, we can conclude $\Sigma;\ \bullet;\ \Gamma';\ \Theta \vdash \boxed{v} : \tau^{\text{SI}} \Rightarrow \Gamma'$.
$\bullet;\ \Gamma';\ \Theta \vdash^{+} \tau^{\text{SI}} \rightsquigarrow \tau^{\text{SI}} \dashv \Gamma'$	Immediate by RR-Refl.
$\exists \Gamma_o.\Gamma' \uplus \Gamma_o = \Gamma'$	$\Gamma_o = \Gamma'$

Case T-BorrowIndex:

From premise:

$$\textsc{T-BorrowIndex}\quad \frac{\begin{array}{c}\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \texttt{u32} \Rightarrow \Gamma' \qquad \Gamma'(r) = \emptyset \qquad \Gamma';\ \Theta \vdash r\ \texttt{rnic} \\ \Delta;\ \Gamma';\ \Theta \vdash_{\omega} p \Rightarrow \{\ \overline{\ell}\ \} \qquad \Delta;\ \Gamma' \vdash_{\omega} p : \tau^{\text{XI}} \\ \tau^{\text{XI}} = [\tau^{\text{SI}};\ n] \lor \tau^{\text{XI}} = [\tau^{\text{SI}}]\end{array}}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{\&r\ \omega\ p[e]} : \&r\ \omega\ \tau^{\text{SI}} \Rightarrow \Gamma'[r \mapsto \{\ \overline{\ell}\ \}]}$$

Since $e = \&r\ \omega\ p[e]$, by inspection of the reduction rules, we know that e steps with the following rule:

$$\textsc{E-EvalCtx}\quad \frac{\Sigma \vdash (\sigma;\ \boxed{e}) \to (\sigma';\ \boxed{e'})}{\Sigma \vdash (\sigma;\ \boxed{\&r\ \omega\ p[e]}) \to (\sigma';\ \boxed{\&r\ \omega\ p[e']})}$$

We begin by applying the induction hypothesis on e. We get that $\exists \Gamma_i$ such that $\Sigma \vdash \sigma' : \Gamma_i$ and $\Sigma;\ \Gamma_i \vdash \overline{v} : \Theta$ and $\Sigma;\ \bullet;\ \Gamma_i;\ \Theta \vdash \boxed{e'} : \texttt{u32} \Rightarrow \Gamma'_f$ and $\bullet;\ \Gamma'_f;\ \Theta \vdash^{+} \texttt{u32} \rightsquigarrow \texttt{u32} \dashv \Gamma_s$ and there exists Γ_o such that $\Gamma' = \Gamma_s \uplus \Gamma_o$. Note that since rewriting does nothing with u32, $\Gamma'_f = \Gamma_s$. We chose Γ_i in the lemma statement to be $\boxed{\Gamma_i}$ and need to show:

$\Sigma \vdash \sigma' : \Gamma_i$	Immediate.
$\Sigma;\ \Gamma_i \vdash \overline{v} : \Theta$	Immediate
$\Sigma;\ \bullet;\ \Gamma_i;\ \Theta \vdash \boxed{\&r\ \omega\ p[e']} : \&r\ \omega\ \tau^{\text{SI}} \Rightarrow \Gamma'_f[r \mapsto \{\ \overline{\ell'}\ \}]$	We'd like to apply T-BorrowIndex. In order to do so, we still need to show $\Gamma'_f(r) = \emptyset$, $\Gamma'_f;\ \Theta \vdash r\ \texttt{rnic}$, $\Sigma;\ \Gamma'_f;\ \bullet \vdash_\omega p \Rightarrow \{\ \overline{\ell'}\ \}$, and $\bullet;\ \Gamma' \vdash_\omega p : \tau^{\text{XI}}$. $\Gamma'_f(r) = \emptyset$ is immediate because $\Gamma'_f(r) \subseteq \Gamma'(r)$. Both $\Gamma'_f;\ \Theta \vdash r\ \texttt{rnic}$ and $\bullet;\ \Gamma' \vdash_\omega p : \tau^{\text{XI}}$ follow from the fact that types are unchanged between Γ' and Γ'_f. The last obligation is $\Sigma;\ \Gamma'_f;\ \bullet \vdash_\omega p \Rightarrow \{\ \overline{\ell'}\ \}$, which follows from Lemma E.53
$\bullet;\ \Gamma'_f[r \mapsto \{\ \overline{\ell'}\ \}];\ \Theta \vdash^{+} \texttt{u32} \rightsquigarrow \texttt{u32} \dashv \Gamma_s$	Immediate, with $\Gamma_s = \Gamma'_f[r \mapsto \{\ \overline{\ell'}\ \}]$.
$\exists \Gamma_o . \Gamma_s \uplus \Gamma_o = \Gamma'[r \mapsto \{\ \overline{\ell}\ \}]$	From our application of the induction hypothesis above, we have that $\Gamma' = \Gamma'_f \uplus \Gamma_o$. As we found in the typing judgement section above, $\{\ \overline{\ell'}\ \} \subseteq \{\ \overline{\ell}\ \}$. Therefore, for this judgement, we can use $\Gamma_o[r \mapsto \{\ \overline{\ell}\ \} \setminus \{\ \overline{\ell'}\ \}]$.

Case T-Tuple:

From premise:

T-Tuple
$$\frac{\forall i \in \{\ 1 \ \dots\ n\ \}.\ \Sigma;\ \Delta;\ \Gamma_{i-1};\ \Theta,\ \tau_1^{\text{SI}},\ \dots,\ \tau_{i-1}^{\text{SI}} \vdash \boxed{e_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_i}{\Sigma;\ \Delta;\ \Gamma_0;\ \Theta \vdash \boxed{(e_1,\ \dots,\ e_n)} : (\tau_1^{\text{SI}},\ \dots,\ \tau_n^{\text{SI}}) \Rightarrow \Gamma_n}$$

Since $e = (v_1,\ \dots,\ v_{i-1}, e_i,\ \dots,\ e_n)$, by inspection of the reduction rules, we know that e steps with the following rule:

E-EvalCtx
$$\frac{\Sigma \vdash (\sigma;\ \boxed{e_i}) \rightarrow (\sigma';\ \boxed{e'_i})}{\Sigma \vdash (\sigma;\ \boxed{(v_1,\ \dots,\ v_{i-1}, e_i,\ \dots,\ e_n)}) \rightarrow (\sigma';\ \boxed{(v_1,\ \dots,\ v_{i-1}, e'_i, e_{i+1},\ \dots,\ e_n)})}$$

We want to apply our induction hypothesis to the premise of E-EvalCtx with the typing judgement $\Sigma;\ \bullet;\ \Gamma_{i-1};\ \Theta, \tau_1^{\text{SI}},\ \dots,\ \tau_{i-1}^{\text{SI}} \vdash \boxed{e_i} : \tau_i^{\text{SI}} \Rightarrow \Gamma_i$. In order to do so we need to show that $\forall\ \overline{v}.\ \Sigma;\ \Gamma_{i-1} \vdash \overline{v}, v_1,\ \dots,\ v_{i-1} : \Theta, \tau_1^{\text{SI}},\ \dots,\ \tau_{i-1}^{\text{SI}}$ given that $\Sigma;\ \Gamma_0 \vdash \overline{v} : \Theta$. So let $v \in \overline{v}$. For each v_j, where $0 < j < i$, we can repeatedly apply Lemma E.15 to get $\Sigma;\ \bullet \vdash \Gamma_{j-1} \rhd \Gamma_{i-1}$, and then apply Lemma E.25 to get that $\forall v_k \in \overline{v}.\ \Sigma;\ \bullet;\ \Gamma_{i-1};\ \Theta, \tau_1^{\text{SI}},\ \dots,\ \tau_{k-1}^{\text{SI}} \vdash \boxed{v_k} : \Theta[k] \Rightarrow \Gamma_{i-1}$.

Now we can apply the induction hypothesis to get there is some Γ'_{i-1} such that $\forall\ \overline{v}.\ \Sigma;\ \Gamma'_{i-1} \vdash \overline{v} : \Theta, \tau_1^{\text{SI}},\ \dots,\ \tau_{i-1}^{\text{SI}}$ and $\Sigma;\ \bullet;\ \Gamma'_{i-1};\ \Theta, \tau_1^{\text{SI}},\ \dots,\ \tau_{i-1}^{\text{SI}} \vdash \boxed{e'_i} : \tau_i^{\text{SI}\prime} \Rightarrow \Gamma'_i$, with $\Sigma \vdash \sigma' : \Gamma'_{i-1}$ and $\bullet;\ \Gamma'_i;\ \Theta, \tau_1^{\text{SI}},\ \dots,\ \tau_{i-1}^{\text{SI}} \vdash^{+} \tau_i^{\text{SI}\prime} \rightsquigarrow \tau_i^{\text{SI}} \dashv \Gamma_{si}$ and there is some Γ_{oi} such that $\Gamma_i = \Gamma_{si} \uplus \Gamma_{oi}$.

We then pick Γ_i in the lemma statement to be $\boxed{\Gamma'_{i-1}}$ and need to show:

$\Sigma \vdash \sigma' : \Gamma'_{i-1}$	Immediate.
$\Sigma;\ \Gamma'_{i-1} \vdash \overline{v} : \Theta$	Immediate
$\Sigma;\ \bullet;\ \Theta;\ \Gamma'_{i-1} \vdash \boxed{(v_1\,,\ \ldots\,,\ v_{i-1}, e'_i, e_{i+1}\,,\ \ldots\,,\ e_n)}$ $: (\tau^{\text{SI}}_1\,,\ \ldots\,,\ \tau^{\text{SI}}_{i-1}, \tau^{\text{SI}\prime}_i, \tau^{\text{SI}}_{i+1}\,,\ \ldots\,,\ \tau^{\text{SI}}_n) \Rightarrow \Gamma'_n$	We want to apply T-Tuple in order to typecheck this tuple expression. We firstly want to show that we can typecheck v_1 through v_{i-1} with initial input context Γ'_{i-1}. For each v_j, where $0 < j < i$, we can repeatedly apply Lemma E.15 to get $\Sigma;\ \bullet\ \vdash\ \Gamma_{j-1} \triangleright \Gamma_{i-1}$, and then apply Lemma E.25 to get that $\Sigma;\ \bullet;\ \Gamma_{i-1};\ \Theta, \tau^{\text{SI}}_1\,,\ \ldots\,,\ \tau^{\text{SI}}_{j-1} \vdash \boxed{v_j} : \tau_j \Rightarrow \Gamma_{i-1}$, and finally applying our continuation typing hypothesis gives us $\Sigma;\ \bullet;\ \Gamma'_{i-1};\ \Theta, \tau^{\text{SI}}_1, \ldots, \tau^{\text{SI}}_{j-1} \vdash \boxed{v_j} : \tau_j \Rightarrow \Gamma'_{i-1}$. Then we can use the result of our induction hypothesis, $\Sigma;\ \bullet;\ \Gamma'_{i-1};\ \Theta, \tau^{\text{SI}}_1\,,\ \ldots\,,\ \tau^{\text{SI}}_{i-1} \vdash \boxed{e'_i} : \tau^{\text{SI}\prime}_i \Rightarrow \Gamma'_i$. And finally for $e_{i+1}\,,\ \ldots\,,\ e_n$, we can repeatedly apply Lemma E.54.
$\bullet;\ \Gamma'_n;\ \Theta \vdash^+ (\tau^{\text{SI}}_1\,,\ \ldots\,,\ \tau^{\text{SI}}_{i-1}, \tau^{\text{SI}\prime}_i, \tau^{\text{SI}}_{i+1}\,,\ \ldots\,,\ \tau^{\text{SI}}_n) \rightsquigarrow (\tau^{\text{SI}}_1\,,\ \ldots\,,\ \tau^{\text{SI}}_n) \dashv \Gamma_s$	Apply RR-Tuple to RR-Refl for every pair of types that are identical, leaving just $\tau^{\text{SI}\prime}_i$ and τ^{SI}_i for which we can use $\bullet;\ \Gamma'_i;\ \Theta, \tau^{\text{SI}}_1\,,\ \ldots\,,\ \tau^{\text{SI}}_{i-1} \vdash^+ \tau^{\text{SI}\prime}_i \rightsquigarrow \tau^{\text{SI}}_i \dashv \Gamma_{si}$ from our induction hypothesis with Lemma E.65 and Lemma E.67 to get that $\bullet;\ \Gamma'_n;\ \Theta \vdash^+ \tau^{\text{SI}\prime}_i \rightsquigarrow \tau^{\text{SI}}_i \dashv \Gamma_s$.
$\exists \Gamma_o . \Gamma_s \uplus \Gamma_o = \Gamma_n$	Note from the induction hypothesis above, we have $\Gamma_i = \Gamma_{si} \uplus \Gamma_{oi}$. This means that for whatever loan sets were unioned between Γ'_i and Γ_{si}, those loan sets will be unioned between Γ'_n and Γ_s. And since Γ_i contained Γ_{si}, it will also be true that Γ_n will contain Γ_s, which means we can use $\Gamma_o = \Gamma_n \setminus \Gamma_s$.

Case T-Drop:

From premise:

$$\dfrac{\Gamma(\pi) = \tau^{\text{SI}}_{\pi} \qquad \Sigma;\ \Delta;\ \Gamma[\pi \mapsto \tau^{\text{SI}^{\dagger}}_{\pi}];\ \Theta \vdash \boxed{e} : \tau^{\text{SX}} \Rightarrow \Gamma_f}{\Sigma;\ \Delta;\ \Gamma;\ \Theta \vdash \boxed{e} : \tau^{\text{SX}} \Rightarrow \Gamma_f}\ \textsc{T-Drop}$$

Since T-Drop applies to *any* expression e, we cannot determine anything about what rule we stepped with. So, we will instead try to apply our induction hypothesis to the typing derivation in the premise of T-Drop $(\Sigma;\ \bullet;\ \Gamma[\pi \mapsto \tau^{\text{SI}^{\dagger}}_{\pi}];\ \Theta \vdash \boxed{e} : \tau^{\text{SX}} \Rightarrow \Gamma_f)$. To do this, we need to establish the premises of our inductive hypothesis.

Namely, we need to show:

(1) $\Sigma;\ \bullet;\ \Gamma[\pi \mapsto \tau^{\text{SI}^{\dagger}}_{\pi}];\ \Theta \vdash \boxed{e} : \tau^{\text{SX}} \Rightarrow \Gamma_f$,
(2) $\Sigma \vdash \sigma : \Gamma[\pi \mapsto \tau^{\text{SI}^{\dagger}}_{\pi}]$,
(3) $\Sigma \vdash (\sigma;\ \boxed{e}) \rightarrow (\sigma';\ \boxed{e'})$.

(1) follows immediately from our premise.
(2) follows almost directly from our premise, which tells us that $\Sigma \vdash \sigma : \Gamma$. We just need to show that the value at x (where x is the root of π) is still valid at its new type. Fortunately, it's old derivation works almost perfectly except for typing the part that corresponds directly to π. In this case, we can use T-Dead on the value to get the new derivation with that part of the aggregate structure at the uninitialized type $\tau^{\text{SI}^{\dagger}}_{\pi}$.
(3) follows immediately from our premise.

This allows us to use our induction hypothesis to conclude that there exists Γ_i' such that:

(5) $\Sigma \vdash \sigma' : \Gamma_i'$,
(6) $\Sigma;\ \Gamma_i' \vdash \overline{v} : \Theta$,
(7) $\Sigma;\ \bullet;\ \Gamma_i';\ \Theta \vdash \boxed{e'} : \tau' \Rightarrow \Gamma_f'$,
(8) $\bullet;\ \Gamma_f';\ \Theta \vdash^{+} \tau' \rightsquigarrow \tau^{\text{SX}} \dashv \Gamma_s$, and
(9) $\exists \Gamma_o.\ \Gamma_s \uplus \Gamma_o = \Gamma_f$.

□

E.16 Type Safety

Theorem E.70 (Type Safety). *If* $\Sigma;\ \bullet;\ \bullet;\ \bullet \vdash \boxed{e} : \tau^{SI} \Rightarrow \Gamma$ *and* $\vdash \Sigma$, *then* $\Sigma \vdash (\bullet;\ \boxed{e}) \rightarrow^{*} (\sigma';\ \boxed{v})$ *or evaluation of e aborts or diverges.*

Proof. By the interleaved use of **Progress** and **Preservation**. □