A Language for Securely Referencing Persistent Information in a Federated System

Jed Liu Andrew C. Myers
Department of Computer Science
Cornell University
Ithaca, New York, USA
liujed@cs.cornell.edu andru@cs.cornell.edu

Abstract

Referential integrity, which guarantees that named resources can be accessed when referenced, is an important property for reliability and security. In distributed systems, however, the attempt to provide referential integrity can itself lead to security vulnerabilities that are not currently well understood. This paper identifies three kinds of *referential security* vulnerabilities related to the referential integrity of distributed, persistent information. Security conditions corresponding to the absence of these vulnerabilities are formalized. A language model is used to capture the key aspects of programming distributed systems with named, persistent resources in the presence of an adversary. The referential security of distributed systems is proved to be enforced by a new type system.

1 Introduction

To make programming manageable, distributed systems are increasingly being implemented using high-level languages and libraries that present distributed resources as language-level objects. This approach goes back to research platforms such as Argus [15], Emerald [4], and Network Objects [3], but is now applied widely in commercial programming using middleware platforms such as CORBA [19], in more recent object-relational mapping (ORM) systems such as Hibernate [11] and other Java Persistence API (JPA) [6] implementations, and in modern JavaScript ORM libraries [5].

Distributed systems naturally cross trust domains; it is often why they are distributed in the first place. Running a program on a federated platform composed of differently trusted distributed nodes creates security vulnerabilities that are not immediately apparent at the high level of abstraction at which the programmer is operating. Some of these vulnerabilities have been addressed by prior work; for example, the Fabric system [16] provides a high-level, Java-like abstraction for distributed programming, while using information-flow control to enforce both confidentiality and integrity properties.

In this paper, we identify three new security goals relating to the security of references that cross trust domains. Cross-domain references are a common feature not only of high-level distributed programming models, but of distributed systems in general. For example, web pages hyperlink to other pages, and relational-database tuples can contain foreign keys referring to other tuples. Regardless of the kind of system, security and reliability vulnerabilities are created when references cross trust boundaries, because they introduce dependencies between different parts of the system. This paper identifies some of these *referential vulnerabilities,* formally characterizes them, and explores a language-based approach to modeling, analyzing, and preventing them.

The first goal is *referential integrity.* A system has referential integrity if a reference can be relied upon to continue pointing to the same object. Referential integrity fails when that object is deleted while the reference still exists, resulting in a *dangling reference*, or when the reference points to a different object altogether.

Referential integrity appears in many guises. We use the term in a more general sense than in the database literature, where referential integrity is an important aspect of the relational model [7]. For example, the web lacks referential integrity: the referent of a hyperlink can be deleted, leading to the familiar "404" error. Referential integrity is also an important property for programming-language design; in programming languages that lack referential integrity, such

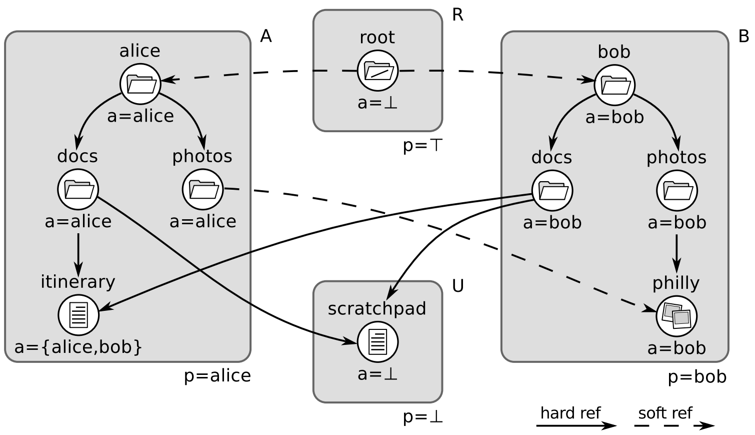

Figure 1: Directory example

as C, dangling pointers are a serious problem. Today, many languages have automatic garbage collection, allowing the automatic reclamation of memory while preserving referential integrity.

While absolute referential integrity is desirable, it cannot be achieved in a federated system: referential integrity is necessarily limited by the trustworthiness of the node (or nodes) storing the referent object. Therefore, this paper generalizes referential integrity to systems where nodes are partially trusted.

Our second goal is *intentional persistence.* With referential integrity, a reference to an object is a promise to the referrer that the object will not move or disappear: it must be persistent. Therefore, reachability implies persistence, as in various object-oriented databases (e.g., [1, 17]) and in marshaling mechanisms such as Java serialization. However, if all reachable objects are persistent, objects can become *accidentally persistent* because they are unexpectedly reachable. Accidental persistence can inflate resource consumption, leading to poor performance and system failure. This problem is familiar to those who have used Java serialization. Intentional persistence entails the absence of accidental persistence.

The third goal of this paper is immunity against *storage attacks*. Referential integrity prevents discarding reachable objects. But this gives an adversary a means to mount a denial-of-service attack. The adversary creates references to objects intended to be discarded, preventing reclamation and perhaps exhausting available storage space.

This paper formalizes these three goals as *referential security properties*, corresponding to the absence of referential vulnerabilities. This is done in the context of a simple programming language that captures the key elements of distributed programming in a federated system with persistent information and pointers. A novel type system is defined and is proved to enforce these security properties.

The rest of this paper is structured as follows. Section 2 describes the language model. Section 3 presents security policies for reasoning about the three vulnerabilities. Section 4 introduces the programming language $\lambda_{persist}$, which abstractly describes distributed programming with persistence and distrust. The language is defined formally in Sections 5 and 6. Section 7 defines the adversary model. Section 8 formalizes the desired security conditions, and proves that the type system of $\lambda_{persist}$ soundly enforces them. Related work is discussed in Section 9, and Section 10 concludes.

2 Language model

2.1 Modeling distributed computing as a language

We model distributed computing using a core programming language that we call $\lambda_{persist}$. In $\lambda_{persist}$, persistence, distribution, and communication are implicit but are constrained by policy annotations. Programs in $\lambda_{persist}$ are assumed to be mapped onto distributed host nodes in some way that agrees with these annotations. This mapping could be done manually by the programmer, or automatically by a compiler, à la Jif/split [23].

This implicit translation to a distributed implementation means that some apparently ordinary source-level operations may be implemented using distributed communication and computation. For example, function application may be implemented as a remote procedure call. Similarly, following references at the language level may involve communication between nodes to fetch referenced objects.

Although the concrete mapping from source-level constructs onto host nodes is left implicit, we can nevertheless faithfully evaluate the security of source-level computations. The key is to ensure that the system is secure under *any* possible concrete mapping that is consistent with the policy annotations in the source program. That is, any given computation or information might be located on any host that satisfies the source-level security constraints. A technical contribution of this paper is to develop an effective system of such source-level constraints, expressed as a type system.

Although we refer to $\lambda_{persist}$ as a source language, little attempt is made to make it congenial to actual programming. In particular, the type annotations introduced would be onerous in practice. They could be inferred automatically using standard constraint-solving techniques for inequations over $\mathcal{L}$, but we leave this to future work. One can view the type system as describing a program (or system) analysis, and the formal results of this paper as a demonstration that this analysis achieves its security goals.

2.2 Objects and references

Persistent objects are modeled in $\lambda_{persist}$ as records with mutable fields. The fields of an object can point to other objects through references. References contain the names of these mutable objects. References are not assignable as in ML [18]; imperative updates are achieved by assigning to mutable fields.

The language has two types of references: hard and soft. A *hard reference* is one with referential integrity: a promise that the referenced object will not be destroyed if its host is trustworthy. Because of this promise, hard references can only be created by trusted code. A *soft reference* does not create an obligation to maintain the referenced object. The language models a garbage collector that may destroy objects reachable only via soft references. When following a soft reference or an untrusted hard reference, a program must be prepared to handle a failure in case the referenced object no longer exists. Hard links in Unix and references in Java are examples of hard references. URLs, Unix symbolic links, and Java SoftReference objects are examples of soft references.

This simple data model can represent many different kinds of systems, such as distributed objects, databases, and the web. The shared directory structure shown in Figure 1 serves as a running example. Alice and Bob are traveling together and are using the system to share photos and itineraries. The root directory is kept on a host R. Alice and Bob keep their directory objects on their own hosts, A and B, respectively. To share sightseeing ideas, they use a common scratchpad stored on host U. Solid arrows in the figure represent hard references, and dashed arrows are soft references. The a and p annotations are policies, which we now explain.

3 Policies for persistent programming

3.1 Persistence policies

Referential integrity ensures that a pointer can be followed to its referent—that there are no dangling pointers. In a federated system, referential integrity cannot be absolute, because the referenced object may be located on an untrusted, perhaps maliciously controlled, host machine. Therefore, referential integrity must be constrained by the degree of trust in the referenced host. This constraint is expressed by assigning each object a *persistence policy* describing how much it can be trusted to remain in existence.

The precise form of the persistence policy is left abstract in this paper. Persistence policies p are assumed to be drawn from a bounded lattice $(\mathcal{L}, \preccurlyeq, \bot, \top)$ of *policy levels*. If $p_1 \preccurlyeq p_2$ for two persistence policies p_1 and p_2, then p_2 describes objects that are at least as persistent as those described by p_1.

Persistence policies have a simple, concrete interpretation. Absent replication, objects are located only on host nodes that are trusted to enforce their persistence policies, so a persistence policy p corresponds to a set of sufficiently trusted host nodes H_p. Therefore, if $p_1 \preccurlyeq p_2$, then p_2 must be enforceable by a smaller set of hosts: $H_{p_1} \supseteq H_{p_2}$. In fact, it is reasonable to think of a policy p as simply a set of hosts.

In Figure 1, the root directory has persistence policy $\top$, which only host R is trusted to enforce. Alice has a user directory and a persistence policy alice. While R is trusted to enforce this policy, she has chosen to use her own host A. Similarly, Bob's directory is on host B. The shared scratchpad is kept on an untrusted host U, which can only enforce the persistence policy $\bot$.

Persistence policies are integrated into the type system of $\lambda_{persist}$. The type of an object reference includes a lower bound on the persistence policy of its referent; the type system ensures that the persistence of an object is always at least as high as that of any reference pointing to it. Programs can therefore use the persistence of a reference to determine whether the reference can be trusted to be intact. This rule enables sound reasoning about persistence and referential integrity as the graph of objects is traversed.

For example, in Figure 1, while Alice and Bob both have a hard reference to the scratchpad, they must be prepared for a persistence failure when using the references. The type system of $\lambda_{persist}$ will ensure their code handles such a failure. Any reference to the scratchpad must have a type with $\perp$ persistence, because it can be no higher than the $\perp$ persistence of the scratchpad itself.

Whether a hard reference can be trusted to be intact depends on context. In Figure 1, Alice and Bob both have a hard reference to the itinerary. Because Alice trusts her own persistence level, if either reference is typed with alice persistence, then she can use it without worrying about a persistence failure. However, unless Bob trusts Alice, he would need to be prepared for such a failure when using the references.

Soft references also have types with persistence levels, and hence might be trusted. Trusted soft references can be promoted to trusted hard references. Therefore, soft references are distinct from untrusted hard references.

In $\lambda_{persist}$, persistence is defined not by reachability, but by policy. This resolves by fiat one of the three problems identified earlier: accidental persistence. Accidents are avoided by allowing programmers to express their intention explicitly. An object that is not intended to be persistent is prevented from being treated as a persistent object.

3.2 Characterizing the adversary

Security involves an adversary, and is always predicated on assumptions about the power of the adversary. In the kind of decentralized, federated system under consideration, the adversary is assumed to control some of the nodes in the system.

Different participants in a distributed system may have their own viewpoints about who the adversary is, yet all participants need security assurance. Therefore, a given adversary is modeled as a point α in the lattice of persistence policy levels. In the host-set interpretation of persistence policies, α defines the set of trusted hosts that the adversary does not control. The adversary is assumed to have the power to delete (i.e., violate the persistence of) an object if its persistence is not α or higher (i.e., $\alpha \not\preccurlyeq p$), because the object might be stored at a host node controlled by the adversary. Other actions by the adversary are modeled by special evaluation rules (see Section 7).

The formal results for the security properties enforced by $\lambda_{persist}$ treat the adversary as an arbitrary parameter. Therefore, these properties hold for any adversary.

3.3 Storage attacks and authority policies

We introduce the idea of *storage attacks*, in which a malicious adversary tries to prevent reclamation of object storage by exploiting the enforcement of referential integrity. For example, in Figure 1, Bob has shared with Alice an album containing the photos he has so far taken during their trip. Bob does not consider the album to be private, so others may create references to his album, as Alice has done. However, an adversary that creates a hard reference to this album can prevent Bob from reclaiming its storage.

To prevent such storage attacks, we ensure that hard references can be created only in sufficiently trusted code. We introduce *creation authority* to abstractly define this power to create new references. This is the only action requiring some form of authority in this paper, so for brevity, we refer to creation authority simply as *authority*.

Like persistence policies, authority policies a are assumed to be drawn from a bounded lattice $(\mathcal{L}, \preccurlyeq)$ of policy levels. Without loss of expressive power, they are assumed to be drawn from the same lattice as persistence policies. Authority prevents storage attacks because hard references can only be created to objects whose authority policy a is less than or equal to the authority a_p of the process; that is, $a \preccurlyeq a_p$.

A hard reference is a reference that should have referential integrity, so creating hard references requires authority. The adversary is assumed to have some ability to create hard references, described by its authority level α. Soft references do not keep an object alive, so no creation authority is required to create a soft reference.

In Figure 1, the root directory has the authority policy $\perp$, so anyone can create a hard reference to it. Bob's philly album is large, so he has given it the authority policy bob; only he can create hard references that prevent the album from being deleted. Therefore, Alice's reference to the album must be soft. Alice has drafted an itinerary, giving it the authority policy $\{\mathsf{alice}, \mathsf{bob}\}$ to indicate she will persist the document for as long as Bob requires. Bob's reference to the itinerary, therefore, can be hard.

	Integrity	Authority	Persistence	Set of hosts
$\top$ "High"	Trusted, Untainted: No one can affect data	"superuser": No one can make a hard reference	Persistent: No one can delete object	No host nodes
$\bot$ "Low"	Untrusted, Tainted: Anyone can affect data	"anyone": Anyone can make a hard reference	Transient: Anyone can delete object	All host nodes

Figure 2: Interpretations of the extremal policy labels

It may sound odd to posit control over creation of references. But a reference with referential integrity is a contract between the referrer and the referent. For example, the node containing the referent is obligated to notify the referrer if the object moves. Entering into a contract requires agreement by both parties, so it is reasonable for the node containing the referent to refuse the creation of a reference.

3.4 Integrity

Thus far, the powers of the adversary include creating references to low-authority objects and destroying objects with low persistence. Because the adversary may control some nodes, the adversary can also change the state of objects located at these nodes. This may in turn affect code running on nodes not controlled by the adversary, if the adversary supplies inputs to that code, or if it affects the decision to run that code.

Integrity policies describe limitations on these effects of the adversary. Integrity policies w are drawn from a bounded lattice $(\mathcal{L}, \preccurlyeq)$ of policy levels; without loss of expressive power, it is assumed to be the same lattice as for persistence and authority policies. In fact, we can think of the persistence and authority levels of an object as the integrity of other, implicit attributes of the object. For persistence, this implicit attribute is the existence of the object itself. For authority, the attribute is the set of incoming references to the object. This unifying view of different policies as different aspects of integrity explains why all three kinds of policies can come from the same lattice.

The ordering $\preccurlyeq$ corresponds to increasing integrity. If $w_1 \preccurlyeq w_2$, an information flow from level w_2 to w_1 would be secure: more-trusted information would be affecting less-trusted information.[1] In $\lambda_{persist}$, each variable and each field of an object has an associated integrity level describing how trusted it is, and hence how powerful an adversary must be to damage it. The integrity of a reference is the integrity of the field or variable it was read from.

Figure 2 summarizes the interpretation of the three kinds of policies.

3.5 Integrity of dereferences and interaction with garbage collection

An adversary can directly affect the result of a dereference in two ways. First, if the reference has low integrity, the adversary can alter it to point to a different object. Second, if the referent has low persistence, the adversary can delete it. Therefore, the integrity of any dereference can be no higher than the integrity and persistence annotations on the reference. In Figure 1, if Alice follows the reference from her docs directory to the scratchpad, she obtains an untrusted result; the untrusted host U influences the result by choosing whether to delete the scratchpad object.

More subtly, the adversary can manipulate hard references to influence the garbage collector, and thereby *indirectly* affect the result of a dereference. For example, in Figure 3a, Alice is following her soft reference to Bob's lyon album. Bob has marked lyon as only requiring low authority, allowing the untrusted, adversarial host U to create a hard reference, and thereby preventing lyon from being garbage-collected. Therefore, Alice's dereference must succeed.

However, in Figure 3b, the adversary U has removed its reference. Subsequently, lyon has been garbage-collected, and Alice's dereference fails. The adversary has indirectly affected the outcome of the dereference. To account for this, the integrity of Alice's dereference must be no higher than the authority required by lyon.

[1]This ordering is the opposite of the "upside-down" ordering typically seen in work on information-flow security [2], and corresponds to the trust ordering [16].

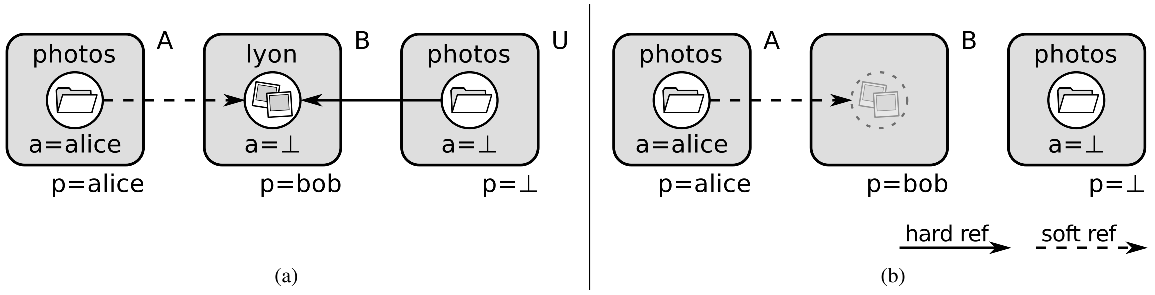

Figure 3: Authority affects integrity of dereferences. Alice is following her soft reference to the lyon album. An adversary can affect the outcome of the dereference, because the album has low authority. (a) The untrusted host U has a hard reference preventing lyon from being garbage collected; Alice's dereference succeeds. (b) Host U has removed its hard reference, allowing lyon to be garbage collected; Alice's dereference fails.

Variables	$x, y \in \mathsf{Var}$	Policy levels	$w, a, p, \ell \in \mathcal{L}$
Memory locations	$m \in \mathsf{Mem}$	PC labels	$pc ::= w$
Labeled record types	$S ::= \{\overrightarrow{x_i : \tau_i}\}_s$	Storage labels	$s ::= (a, p)$
Labeled ref types	$R ::= \{\overrightarrow{x_i : \tau_i}\}_r$	Reference labels	$r ::= (a^+, a^-, p)$
Base types	$b ::= \mathsf{bool} \mid \tau_1 \xrightarrow{pc} \tau_2 \mid R \mid \mathsf{soft}\ R$	Types	$\tau ::= b_w \mid \mathbf{1}$
Values	$v, u ::= x \mid \mathsf{true} \mid \mathsf{false} \mid * \mid m^S \mid \mathsf{soft}\ m^S \mid \lambda(x:\tau)[pc].e\ \ (\mid \bot_p)$		
Terms	$e ::= v \mid v_1\ v_2 \mid \mathsf{if}\ v_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \mid \{\overrightarrow{x_i = v_i}\}^S \mid v.x$		
	$\mid v_1.x := v_2 \mid \mathsf{soft}\ e \mid e_1 \Vert e_2 \mid \mathsf{exists}\ v\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2 \mid \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$		

Figure 4: Syntax of $\lambda^0_{persist}$. Parenthesized productions only appear at run time.

4 Types for persistent programming

To formalize the ideas presented in the previous section, we introduce the $\lambda_{persist}$ language, an extension to the simply typed lambda calculus. Figure 4 gives part of the formal syntax of $\lambda_{persist}$. Its type system prevents referential vulnerabilities by integrating policies for persistence, authority, and integrity into types. Accidental persistence is prevented because persistence is determined by policies expressing the programmer's intent, rather than by reachability. Referential integrity is maintained by a $\lambda_{persist}$ program with respect to a particular adversary if following hard references whose persistence and integrity are above the level of the adversary never leads to an object that has been destroyed by the adversary or garbage-collected. Storage attacks are prevented if the adversary is unable to change the set of high-authority objects that are reachable through hard references.

4.1 Labels

We assume a bounded lattice $(\mathcal{L}, \preccurlyeq, \bot, \top)$ of *policy levels*, from which integrity (w), authority (a), and persistence policies (p) are drawn.

Objects and reference values are annotated with *storage labels* consisting of a creation authority policy and a persistence policy. All non-unit types τ consist of a base type b along with an integrity policy annotation w; fields and variables thereby acquire integrity policies, because they are part of their types. Objects do not have their own integrity labels because all of their state is in their fields, which do have labels.

The program-counter label pc [9] is an integrity level indicating the degree to which the program's control flow has been tainted by untrusted data. This label restricts the side effects of code.

4.2 Example

Suppose we want to create a hierarchical, distributed directory structure, such as in Figure 1. Each directory maps names to either strings, representing ordinary files, or to other directories, and contains a reference to its parent directory (elided in the figure). To faithfully model ordinary filesystems, directories higher in the hierarchy should be more persistent: if they are destroyed, so is everything below.

A fully general directory structure would require augmenting $\lambda_{persist}$ with recursive and dependent types; for simplicity, these features have been omitted from $\lambda_{persist}$ because they do not appear to add interesting issues. However, we can capture the security of a general directory structure by using $\lambda_{persist}$ records to build a fixed-depth directory structure with a fixed set of entry names for each directory.

4.3 Modeling objects and references

The security policies of $\lambda_{persist}$ are about objects and references to them. Therefore, $\lambda_{persist}$ extends the lambda calculus with records that represent the content of objects. The record $\{\overrightarrow{x_i = v_i}\}$ comprises a set of fields $\overrightarrow{x_i}$ with corresponding values $\overrightarrow{v_i}$. Records are not values in the language; instead, they are accessed via references m^S, where m is the identity of the object and $S = \{\overrightarrow{x_i : \tau_i}\}_s$ gives its base record type. The *storage label* s is a pair (a, p). The *authority label* a is an upper bound on the authority required to create a new reference to the referent object.

References to objects have labeled reference types $\{\overrightarrow{x_i : \tau_i}\}_r$. A reference label r is a triple (a^+, a^-, p) that gives upper and lower bounds on the authority required by the referent, and a lower bound on the persistence of the referent. The *upper authority label* a^+ restricts reference copying to prevent storage attacks. The *lower authority label* a^- prevents the adversary from exploiting garbage collection to damage integrity (Section 3.5), by tainting the integrity of dereferencing soft references.

4.4 Modeling distributed systems

The goal of the $\lambda_{persist}$ language is to model a distributed system in which code is running at different host nodes. A single program written in $\lambda_{persist}$ is intended to represent such a system. The key to modeling distributed, federated computation faithfully is that different parts of the program can be annotated with different integrity labels, representing the trust that has been placed in that part of the code. To model a set of computations (subprograms $\overrightarrow{e_i}$) executing at different nodes, the individual computations are composed in parallel ($e_1 \parallel \cdots \parallel e_n$) into a single $\lambda_{persist}$ program.

From the viewpoint of a given principal in the system, code with a low integrity label, relative to that principal, can be replaced by any code at all. For the purposes of evaluating the security of the system, this code is in effect erased and replaced by the adversary. Therefore the single-program representation faithfully models a distributed system containing an adversary.

5 Accidental persistence and storage attacks

We present $\lambda_{persist}$ in two phases. In this section, we present $\lambda^0_{persist}$, a simplified subset of $\lambda_{persist}$ that prevents accidental persistence and storage attacks.

5.1 Syntax of $\lambda^0_{persist}$

Figure 4 gives the syntax of $\lambda^0_{persist}$. The names x and y range over variable names Var; m ranges over a space of memory addresses Mem; w, a, p, and ℓ range over the lattice $\mathcal{L}$ of policy levels; and s and r range over the space of storage labels $\mathcal{L}^2$ and reference labels $\mathcal{L}^3$, respectively.

Types in $\lambda^0_{persist}$ consist of base types with an integrity label (b_w), and the unit type $\mathbf{1}$, which needs no integrity label. Base types include booleans, functions, and two kinds of references to mutable records: hard (R) and soft ($\mathsf{soft}\ R$). The metavariable R denotes a labeled reference type.

The type $\tau_1 \xrightarrow{pc} \tau_2$ is a function type with a pc annotation that is a lower bound on the pc label of the caller. It gives an upper bound on the integrity of data the function affects, on the authority level of references the function creates, and on the authority level of references in the function body and in the closure environment.

Values include variables x, booleans true and false, the unit value $*$, record-typed memory locations (references) m^S, soft references $\mathsf{soft}\ m^S$, and functions $\lambda(x{:}\tau)[pc].\,e$. A function $\lambda(x{:}\tau)[pc].\,e$ has one argument x with type τ. The pc component has the same meaning as that in function types. At run time, p-persistence failures $\bot_p$ can also appear as values.

Terms include values v and u, applications $v_1\ v_2$, if expressions $\mathsf{if}\ v_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3$, record constructors $\{\overrightarrow{x_i = v_i}\}^S$, field selections $v.x$, field assignments $v_1.x := v_2$, soft references $\mathsf{soft}\ e$, parallel composition $e_1 \parallel e_2$, soft-reference tests $\mathsf{exists}\ v\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2$, and let expressions $\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$.

[Apply] $$\langle(\lambda(x{:}\tau)[pc].\,e)\ v, M\rangle \xrightarrow{e} \langle e\{v/x\}, M\rangle$$

[Let] $$\frac{\forall p.\ v \neq \bot_p}{\langle \mathsf{let}\ x = v\ \mathsf{in}\ e, M\rangle \xrightarrow{e} \langle e\{v/x\}, M\rangle}$$

[If-True] $$\langle \mathsf{if\ true\ then}\ e_1\ \mathsf{else}\ e_2, M\rangle \xrightarrow{e} \langle e_1, M\rangle$$

[If-False] $$\langle \mathsf{if\ false\ then}\ e_1\ \mathsf{else}\ e_2, M\rangle \xrightarrow{e} \langle e_2, M\rangle$$

[Create] $$\frac{m = \mathsf{newloc}(M)}{\left\langle \{\overrightarrow{x_i = v_i}\}^S, M\right\rangle \xrightarrow{e} \left\langle m^S, M[m^S \mapsto \{\overrightarrow{x_i = v_i}\}]\right\rangle}$$

[Parallel-Result] $$\langle v_1 \parallel v_2, M\rangle \xrightarrow{e} \langle *, M\rangle$$

[Select] $$\frac{M(m^S) = \{\overrightarrow{x_i = v_i}\}}{\left\langle m^S.x_c, M\right\rangle \xrightarrow{e} \langle v_c, M\rangle}$$

[Assign] $$\frac{M(m^S) \neq \bot \qquad \forall p.\ v \neq \bot_p}{\left\langle m^S.x_c := v, M\right\rangle \xrightarrow{e} \left\langle *, M[m^S.x_c \mapsto v]\right\rangle}$$

[Dangle-Select] $$\frac{M(m^S) = \bot \qquad p = \mathsf{persist}(m^S)}{\left\langle m^S.x_c, M\right\rangle \xrightarrow{e} \langle \bot_p, M\rangle}$$

[Dangle-Assign] $$\frac{M(m^S) = \bot \qquad p = \mathsf{persist}(m^S)}{\left\langle m^S.x_c := v, M\right\rangle \xrightarrow{e} \langle \bot_p, M\rangle}$$

[Exists-True] $$\frac{M(m^S) \neq \bot}{\left\langle \mathsf{exists\ soft}\ m^S\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2, M\right\rangle \xrightarrow{e} \left\langle e_1\{m^S/x\}, M\right\rangle}$$

[Exists-False] $$\frac{M(m^S) = \bot}{\left\langle \mathsf{exists\ soft}\ m^S\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2, M\right\rangle \xrightarrow{e} \langle e_2, M\rangle}$$

[Eval-Context] $$\frac{\langle e, M\rangle \xrightarrow{e} \left\langle e', M'\right\rangle}{\langle E[e], M\rangle \xrightarrow{e} \left\langle E[e'], M'\right\rangle}$$

[Fail-Prop] $$\langle F[\bot_p], M\rangle \xrightarrow{e} \langle \bot_p, M\rangle$$

$$E ::= \mathsf{soft}\ [\cdot] \mid \mathsf{let}\ x = [\cdot]\ \mathsf{in}\ e \mid [\cdot] \parallel e \mid e \parallel [\cdot] \qquad F ::= \mathsf{soft}\ [\cdot] \mid \mathsf{let}\ x = [\cdot]\ \mathsf{in}\ e$$

[Prog-Step] $$\frac{\langle e, M\rangle \xrightarrow{e} \left\langle e', M'\right\rangle}{\langle e, M\rangle \rightarrow \left\langle e', M'\right\rangle}$$

[GC] $$\frac{\mathsf{gc}(G, \langle e, M\rangle)}{\langle e, M\rangle \rightarrow \langle e, M[G \mapsto \bot]\rangle}$$

Figure 5: Small-step operational semantics for nonadversarial execution of $\lambda^0_{persist}$.

5.2 Example

Returning to the directory example in Figure 1, Bob can add to the itinerary with the code below. It starts at the root of the directory structure, traverses down to the itinerary, and invokes an add method to add a museum.

```
let home = root.bob
in exists home as bob:
      let docs = bob.docs
      in docs.itinerary.add "Rodin Museum"
   else: ...
```

The garbage collector may have snapped the soft reference home to Bob's home directory, so exists is used to determine whether the reference is still valid. If so, the body of the exists is evaluated with bob bound to a hard reference to the home directory.[2] (This reference can be created because the *pc* label at this point has sufficient creation authority.) The second select expression, bob.docs, dereferences the hard reference.

5.3 Operational semantics of $\lambda^0_{persist}$

Figure 5 gives the small-step operational semantics of $\lambda^0_{persist}$. The notation $e\{v/x\}$ denotes capture-avoiding substitution of value v for variable x in expression e. A failed or garbage-collected memory location contains value $\bot$. Most of the operational semantics rules are straightforward, but a few deserve more explanation.

[2]To avoid a race with the garbage collector, an implementation of exists should first optimistically create the hard reference, then check its validity before exposing it to the program.

Let M represent a memory: a finite partial map from typed memory locations m^S to closed record values. Let $\langle e, M\rangle$ be a system configuration. A small evaluation step is a transition from $\langle e, M\rangle$ to another configuration $\langle e', M'\rangle$, written $\langle e, M\rangle \rightarrow \langle e', M'\rangle$.

To avoid using undefined memory locations, we restrict the form of $\langle e, M\rangle$. Let $\mathsf{locs}(e)$ represent the set of locations appearing explicitly in e. A memory M is well-formed only if every address m appears at most once in $\mathsf{dom}(M)$, and for any location m^S in $\mathsf{dom}(M)$, $\mathsf{locs}(M(m^S)) \subseteq \mathsf{dom}(M)$. A configuration $\langle e, M\rangle$ is well-formed only if M is well-formed, $\mathsf{locs}(e) \subseteq \mathsf{dom}(M)$, and e has no free variables. Evaluation preserves well-formed configurations (see Lemma 10 in Section 8.3).

Though the operational semantics refer to complete record types, only their persistence labels are needed at run time. These labels are only used to determine the level of persistence failure that occurs when dereferencing a dangling reference (rules Dangle-Select and Dangle-Assign), so run-time overhead should be small.

The record constructor $\{\overrightarrow{x_i = v_i}\}^S$ (rule Create) creates a new memory location m^S to hold the record. The component S specifies the base type and storage label of the record. The storage label governs at what nodes the object can be created. The function $\mathsf{newloc}(M)$ deterministically generates a fresh memory location. If $\mathsf{address\text{-}space}(M)$ represents the set of location names in M (i.e., $\{m : \exists S. m^S \in \mathsf{dom}(M)\}$), then $\mathsf{newloc}(M) \notin \mathsf{address\text{-}space}(M)$ and $\mathsf{newloc}(M') = \mathsf{newloc}(M)$ if $\mathsf{address\text{-}space}(M') = \mathsf{address\text{-}space}(M)$.

Parallel-composition expressions $e_1 \parallel e_2$ evaluate to the unit value (rule Parallel-Result).

The field-selection expression $v.x$ (rules Select and Dangle-Select) evaluates v to a memory location m^S. If the location has not failed, the result of the selection is the value of the field x of the record at that location. Otherwise, a p-persistence failure occurs, where p is the persistence level of m^S, written $p = \mathsf{persist}(m^S)$.

The field-assignment expression $v_1.x := v_2$ evaluates v_1 to a memory location m^S (rules Assign and Dangle-Assign) If the location has not failed, v_2 is assigned into the field x of the record at that location; otherwise, a p-persistence failure occurs (where $p = \mathsf{persist}(m^S)$). The notation $M[m^S.x_c \mapsto v]$ denotes the memory resulting from updating with value v the field x_c of the record at location m^S.

Persistence failures propagate outward dynamically (Fail-Prop) until the whole program fails. The full $\lambda_{persist}$ language, defined in Section 6, can handle these failures.

In rule Eval-Context, E represents an ordinary evaluation context, whereas in rule Fail-Prop, F gives the contexts from which persistence failures propagate. Contexts are given as a term with a single hole (denoted by $[\cdot]$) in redex position. The syntax of E specifies the evaluation order.

The soft-reference expression $\mathsf{soft}\ e$ evaluates e to a hard reference and turns it into a soft reference. The soft-reference test $(\mathsf{exists}\ v\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2)$ promotes the soft reference v (if valid) to a hard reference bound to x and evaluates e_1. If the reference is invalid, e_2 is evaluated instead.

In rule GC, the notation $\mathsf{gc}(G, \langle e, M\rangle)$ means that G is a set of locations that is *collectible*. G is considered collectible if it has no GC roots (i.e., hard references in e), and no location outside G has a hard reference into G. This is defined formally as follows.

Definition 1 (GC roots)**.** *A location m^S is a* GC root *in an expression e, written* $\mathsf{root}(m^S, e)$*, if it is a hard reference in e. This is formally defined by the following inference rules:*

$$[\text{R1}]\ \frac{}{\mathsf{root}(m^S, m^S)} \qquad [\text{R2}]\ \frac{\mathsf{root}(m^S, e) \qquad \forall m_0^{S_0}.\ e \neq m_0^{S_0}}{\mathsf{root}(m^S, \mathsf{soft}\ e)} \qquad [\text{R3}]\ \frac{\exists i.\ \mathsf{root}(m^S, v_i)}{\mathsf{root}(m^S, \{\overrightarrow{x_i = v_i}\}^{S'})}$$

$$[\text{R4}]\ \frac{\mathsf{root}(m^S, v)}{\mathsf{root}(m^S, v.x)} \qquad [\text{R5}]\ \frac{\exists i.\ \mathsf{root}(m^S, v_i)}{\mathsf{root}(m^S, v_1.x := v_2)} \qquad [\text{R6}]\ \frac{\mathsf{root}(m^S, e)}{\mathsf{root}(m^S, \lambda(x{:}\tau)[pc].e)}$$

$$[\text{R7}]\ \frac{\exists i.\ \mathsf{root}(m^S, v_i)}{\mathsf{root}(m^S, v_1\ v_2)} \qquad [\text{R8}]\ \frac{\exists i.\ \mathsf{root}(m^S, e_i)}{\mathsf{root}(m^S, \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2)} \qquad [\text{R9}]\ \frac{\exists i.\ \mathsf{root}(m^S, e_i)}{\mathsf{root}(m^S, e_1 \parallel e_2)}$$

$$[\text{R10}]\ \frac{\exists i.\ \mathsf{root}(m^S, e_i)}{\mathsf{root}(m^S, \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3)} \qquad [\text{R11}]\ \frac{\exists i.\ \mathsf{root}(m^S, e_i)}{\mathsf{root}(m^S, \mathsf{exists}\ e_1\ \mathsf{as}\ x : e_2\ \mathsf{else}\ e_3)}$$

$\lambda_{persist}$, defined in Section 6, adds R12; and $[\lambda_{persist}]$, defined in Section 7, adds R13:

$$[\text{R12}]\ \frac{\exists i.\ \mathsf{root}(m^S, e_i)}{\mathsf{root}(m^S, \mathsf{try}\ e_1\ \mathsf{catch}\ p{:}\ e_2)} \qquad [\text{R13}]\ \frac{\mathsf{root}(m^S, e)}{\mathsf{root}(m^S, [e])}$$

$$[\mathrm{S1}]\quad \frac{n > m}{\vdash \{x_1:\tau_1,\ldots,x_n:\tau_n\}_r \le \{x_1:\tau_1,\ldots,x_m:\tau_m\}_r} \qquad [\mathrm{S2}]\quad \frac{\vdash R_1 \le R_2}{\vdash \mathsf{soft}\ R_1 \le \mathsf{soft}\ R_2}$$

$$[\mathrm{S3}]\quad \frac{\vdash b_1 \le b_2 \qquad \vdash w_2 \preccurlyeq w_1}{\vdash (b_1)_{w_1} \le (b_2)_{w_2}} \qquad [\mathrm{S4}]\quad \frac{\vdash \tau_2 \le \tau_1 \qquad \vdash \tau_1' \le \tau_2' \qquad \vdash pc_1 \preccurlyeq pc_2}{\vdash \tau_1 \xrightarrow{pc_1} \tau_1' \le \tau_2 \xrightarrow{pc_2} \tau_2'}$$

$$[\mathrm{S5}]\quad \frac{\vdash a_1^+ \preccurlyeq a_2^+ \qquad \vdash a_2^- \preccurlyeq a_1^- \qquad \vdash p_2 \preccurlyeq p_1}{\vdash \{\overrightarrow{x_i:\tau_i}\}_{(a_1^+,a_1^-,p_1)} \le \{\overrightarrow{x_i:\tau_i}\}_{(a_2^+,a_2^-,p_2)}}$$

Figure 6: Subtyping rules for $\lambda^0_{persist}$

Definition 2 (Collectible groups)**.** *A set of locations G is a* collectible group *in a configuration $\langle e,M\rangle$, written $\mathsf{gc}(G,\langle e,M\rangle)$, if it does not contain any roots of e, and no location outside G has a hard reference into G.*

$$\begin{aligned}&\mathsf{gc}(G,\langle e,M\rangle) \overset{def.}{\iff}\\ &\quad G \subseteq \mathsf{dom}(M)\\ &\quad \wedge (\nexists m^S \in G.\ \mathsf{root}(m^S,e))\\ &\quad \wedge \forall m_0^{S_0} \in \mathsf{dom}(M).\ (M(m_0^{S_0}) \neq \bot \wedge \exists m^S \in G.\ \mathsf{root}(m^S, M(m_0^{S_0}))) \Rightarrow m_0^{S_0} \in G\end{aligned}$$

5.4 Subtyping in $\lambda^0_{persist}$

The subtyping judgment $\vdash \tau_1 \le \tau_2$ states that any value of type τ_1 can be treated as a value of type τ_2. Subtyping in $\lambda^0_{persist}$ is the least reflexive and transitive relation consistent with the rules given in Figure 6. Rule S1 gives standard width subtyping on records. Because records are mutable, there is no depth subtyping.

Subtyping on soft references is covariant (rule S2). While hard references may be soundly used as soft references, this is omitted for simplicity. Rule S3 gives contravariant subtyping on integrity labels. Rule S4 gives standard subtyping on functions; the additional pc component is covariant. These are the opposite of the rules typically seen in work on information-flow security, accounting for our use of the trust ordering.

Rule S5 gives subtyping for labeled reference types. Subtyping is covariant on the a^+ component of the reference label and contravariant on the other two components. This ensures the bounds specified by the reference label of the subtype are at least as precise as those of the supertype.

5.5 Static semantics of $\lambda^0_{persist}$

Typing rules for $\lambda^0_{persist}$ are given in Figure 7. The notation $\mathsf{auth}^+(r)$, $\mathsf{auth}^-(r)$, and $\mathsf{persist}(r)$ give the upper authority (a^+), lower authority (a^-), and persistence (p) component of a reference label r, respectively. The notation $\mathsf{auth}^+(s)$ and $\mathsf{persist}(s)$ give the authority and persistence component of a storage label, respectively. The notation $\mathsf{auth}^+(\tau)$, defined below, gives the authority level needed to create a hard reference to a value of type τ. The integrity of τ is written $\mathsf{integ}(\tau)$, and $\tau \sqcap \ell$ denotes the type obtained by tainting (meeting) the integrity of τ with ℓ:

$$\begin{array}{ccc} & \mathsf{auth}^+(\mathsf{bool}) = \mathsf{auth}^+(\mathbf{1}) = \mathsf{auth}^+(\mathsf{soft}\ R) = \bot & \\ \mathsf{integ}(b_w) = w & (b_w) \sqcap \ell = b_{w \sqcap \ell} & \mathsf{auth}^+(\tau_1 \xrightarrow{pc} \tau_2) = pc \\ \mathsf{integ}(\mathbf{1}) = \top & \mathbf{1} \sqcap \ell = \mathbf{1} & \mathsf{auth}^+(\{\overrightarrow{x_i:\tau_i}\}_s) = \mathsf{auth}^+(s) \end{array}$$

The typing context includes a *type assignment* Γ and the program-counter label pc. Γ is a finite partial map from variables x to types τ, expressed as a finite list of $x{:}\tau$ entries. We write $x{:}\tau \in \Gamma$ and $\Gamma(x) = \tau$ interchangeably. For an expression e that is well-typed in a context $\Gamma;pc$, the type checker produces a type τ. The typing assertion $\Gamma;pc \vdash e : \tau$, therefore, means that the expression e has type τ under type assignment Γ with program-counter label pc.

Most of the typing rules are standard rules, extended to ensure that the pc is sufficiently high to obtain any hard references that may result from evaluating subexpressions (e.g., premise $\vdash \mathsf{auth}^+(\tau) \preccurlyeq pc$ in Rule T-IF), and that the pc is suitably tainted.

$$[\text{T-BOOL}]\quad \frac{b \in \{\mathsf{true},\mathsf{false}\}}{\Gamma;pc \vdash b : \mathsf{bool}_\top} \qquad [\text{T-UNIT}]\quad \Gamma;pc \vdash * : \mathbf{1}$$

$$[\text{T-VAR}]\quad \frac{\Gamma(x) = \tau}{\Gamma;pc \vdash x : \tau} \qquad [\text{T-BOT}]\quad \frac{p \neq \top}{\Gamma;pc \vdash \bot_p : \tau}$$

$$[\text{T-LOC}]\quad \frac{\vdash_{wf} S : \mathsf{rectype} \qquad S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}}{\Gamma;pc \vdash m^S : (\{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top} \qquad [\text{T-SOFT}]\quad \frac{\Gamma;pc \vdash e : R_w}{\Gamma;pc \vdash \mathsf{soft}\ e : (\mathsf{soft}\ R)_w}$$

$$[\text{T-IF}]\quad \frac{\begin{array}{c}\Gamma;pc \vdash v : \mathsf{bool}_w \\ \Gamma;pc \sqcap w \vdash e_i : \tau\ ^{(\forall i)} \\ \vdash \mathsf{auth}^+(\tau) \preccurlyeq pc \sqcap w\end{array}}{\Gamma;pc \vdash \mathsf{if}\ v\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau \sqcap w} \qquad [\text{T-PLL}]\quad \frac{\begin{array}{c}\Gamma;pc \vdash e_i : \tau_i\ ^{(\forall i)} \\ \vdash \mathsf{auth}^+(\tau_i) \preccurlyeq pc\ ^{(\forall i)}\end{array}}{\Gamma;pc \vdash e_1 \parallel e_2 : \mathbf{1}}$$

$$[\text{T-ABS}]\quad \frac{\begin{array}{c}\Gamma, x{:}\tau'; pc' \vdash e : \tau \\ \vdash_{wf} (\tau' \xrightarrow{pc'} \tau)_\top : \mathsf{type} \qquad \vdash pc' \preccurlyeq pc\end{array}}{\Gamma;pc \vdash \lambda(x{:}\tau')[pc'].e : (\tau' \xrightarrow{pc'} \tau)_\top} \qquad [\text{T-APP}]\quad \frac{\begin{array}{c}\Gamma;pc \vdash v_1 : (\tau' \xrightarrow{pc'} \tau)_w \\ \Gamma;pc \vdash v_2 : \tau' \\ \vdash pc' \preccurlyeq pc \sqcap w\end{array}}{\Gamma;pc \vdash v_1\ v_2 : \tau \sqcap w}$$

$$[\text{T-REC}]\quad \frac{\begin{array}{c}\vdash_{wf} S : \mathsf{rectype} \qquad S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)} \qquad \Gamma;pc \vdash v_i : \tau_i'\ ^{(\forall i)} \\ \vdash \tau_i' \leq \tau_i\ ^{(\forall i)} \qquad \vdash \mathsf{auth}^+(\tau_i') \preccurlyeq pc\ ^{(\forall i)} \qquad \vdash \mathsf{integ}(\tau_i) \preccurlyeq pc\ ^{(\forall i)} \qquad \vdash p \preccurlyeq pc\end{array}}{\Gamma;pc \vdash \{\overrightarrow{x_i = v_i}\}^S : (\{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top}$$

$$[\text{T-SEL}]\quad \frac{\begin{array}{c}\Gamma;pc \vdash v : (\{\overrightarrow{x_i : \tau_i}\}_r)_w \\ \vdash \mathsf{auth}^+(r) \preccurlyeq pc \\ w' = w \sqcap \mathsf{persist}(r)\end{array}}{\Gamma;pc \vdash v.x_c : \tau_c \sqcap w'} \qquad [\text{T-ASGN}]\quad \frac{\begin{array}{c}\Gamma;pc \vdash v_1 : (\{\overrightarrow{x_i : \tau_i}\}_r)_w \\ \vdash \mathsf{auth}^+(r) \preccurlyeq pc \\ \Gamma;pc \vdash v_2 : \tau \\ \vdash \tau \sqcap pc \sqcap w \leq \tau_c \\ \vdash \mathsf{auth}^+(\tau) \preccurlyeq pc \sqcap w\end{array}}{\Gamma;pc \vdash v_1.x_c := v_2 : \mathbf{1}}$$

$$[\text{T-EXISTS}]\quad \frac{\begin{array}{c}\Gamma;pc \vdash v : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_r)_w \qquad \vdash \mathsf{auth}^+(r) \preccurlyeq pc \sqcap w \\ w' = \mathsf{auth}^-(r) \sqcap \mathsf{persist}(r) \sqcap w \qquad \Gamma, x{:}(\{\overrightarrow{x_i : \tau_i}\}_r)_w; pc \sqcap w' \vdash e_1 : \tau \\ \Gamma;pc \sqcap w' \vdash e_2 : \tau \qquad \vdash \mathsf{auth}^+(\tau) \preccurlyeq pc \sqcap w'\end{array}}{\Gamma;pc \vdash \mathsf{exists}\ v\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2 : \tau \sqcap w'}$$

$$[\text{T-LET}]\quad \frac{\begin{array}{c}\Gamma;pc \vdash e_1 : \tau' \qquad \vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \\ w = \mathsf{integ}(\tau') \qquad pc' = pc \sqcap w \\ \Gamma, x{:}\tau'; pc' \vdash e_2 : \tau \qquad \vdash \mathsf{auth}^+(\tau) \preccurlyeq pc'\end{array}}{\Gamma;pc \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau \sqcap w} \qquad [\text{T-SUB}]\quad \frac{\Gamma;pc \vdash e : \tau' \qquad \vdash \tau' \leq \tau}{\Gamma;pc \vdash e : \tau}$$

Figure 7: Typing rules for $\lambda^0_{persist}$

Rule T-BOT says persistence failures can have any well-formed type.

Rule T-ABS checks function values. It ensures that the function's program-counter label pc' accurately summarizes the authority levels of the references contained in the closure, and that the pc is high enough to create this closure. The body is checked with program-counter label pc', so in rule T-APP, the function can only be used by code with sufficient integrity.

Rule T-REC checks the creation of records. It requires that the annotation S be well-formed. Also, the pc must be high enough to create any hard references that appear in the fields, and to write to the fields themselves.

When using a hard reference v_1, the pc must have sufficient authority to possess v_1 (premise $\vdash \mathsf{auth}^+(r) \preccurlyeq pc$ in rules T-SEL and T-ASGN). When assigning through v_1, hard references contained in the assigned value v_2 also require authority. Since the integrity and persistence of v_1 can affect whether the assignment succeeds, we taint the pc with these labels before comparing with the authority requirement of v_2.

Rule T-EXISTS checks soft-reference validity tests. It ensures that the pc has the authority to promote the reference from soft to hard (premise $\vdash \mathsf{auth}^+(r) \preccurlyeq pc$).

$$[\mathrm{WT1}] \quad \vdash_{wf} \mathsf{bool}_w : \mathsf{type} \qquad [\mathrm{WT2}] \quad \frac{\vdash pc \preccurlyeq w \qquad \vdash_{wf} \tau_1 : \mathsf{type} \qquad \vdash_{wf} \tau_2 : \mathsf{type} \qquad \vdash \mathsf{auth}^+(\tau_1) \sqcup \mathsf{auth}^+(\tau_2) \preccurlyeq pc}{\vdash_{wf} (\tau_1 \xrightarrow{pc} \tau_2)_w : \mathsf{type}}$$

$$[\mathrm{WT3}] \quad \vdash_{wf} \mathbf{1} : \mathsf{type} \qquad [\mathrm{WT4}] \quad \frac{\vdash_{wf} (\{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top : \mathsf{type} \qquad \vdash \mathsf{integ}(\tau_i) \preccurlyeq p \ ^{(\forall i)}}{\vdash_{wf} \{\overrightarrow{x_i : \tau_i}\}_{(a,p)} : \mathsf{rectype}}$$

$$[\mathrm{WT5}] \quad \frac{\vdash_{wf} R_\top : \mathsf{type}}{\vdash_{wf} (\mathsf{soft}\ R)_w : \mathsf{type}} \qquad [\mathrm{WT6}] \quad \frac{\vdash_{wf} \tau_i : \mathsf{type} \ ^{(\forall i)} \qquad \vdash \mathsf{auth}^+(\tau_i) \preccurlyeq a^+ \ ^{(\forall i)} \qquad \vdash a^+ \preccurlyeq w \sqcap p \qquad \vdash a^- \preccurlyeq a^+}{\vdash_{wf} (\{\overrightarrow{x_i : \tau_i}\}_{(a^+,a^-,p)})_w : \mathsf{type}}$$

Figure 8: Well-formedness of types

The rules for determining the well-formedness of types are given in Figure 8. In rule WT6, a reference type $(\{\overrightarrow{x_i : \tau_i}\}_{(a^+,a^-,p)})_w$ is well-formed only if the upper authority label a^+ is an upper bound on the authority levels of the field types τ_i. This ensures that the upper authority label is an accurate summary of the authority required by the fields. We also require a^+ be bounded from above by the integrity w of the reference, since low-integrity data should not influence the creation of high-authority references. To ensure hosts are able to create hard references to the objects they store, we also require $\mathsf{auth}^+(r)$ to be bounded from above by the persistence level p of the record.

6 Ensuring referential integrity

In a distributed system, references can span trust domains, so to be secure and reliable, program code must in general be ready to encounter a dangling reference, one perhaps created by the adversary. Therefore, we extend $\lambda^0_{persist}$ with *persistence-failure handlers* to obtain the full $\lambda_{persist}$ language (see Appendix A.1 for its full syntax). The type system of $\lambda_{persist}$ forces the programmer to be aware of and to handle all potential failures.

One approach is to handle the failure immediately upon using a broken reference. However, because low-persistence references may be used frequently, this would result in lots of duplicated failure-handling code.

Instead, $\lambda_{persist}$ factors out failure-handling code from ordinary code by treating failures as a kind of exception. The value of $(\mathsf{try}\ e_1\ \mathsf{catch}\ p\text{:}\ e_2)$ is the value of evaluating e_1. If a dangling reference at persistence level p or higher is encountered, the failure handler e_2 is evaluated instead. A try expression creates a context (e_1) in which the programmer can write simpler code under the assumption that certain persistence failures are impossible, yet without sacrificing the property that all failures are handled.

6.1 Persistence handler levels

To track the failures that the current context can handle, a *set* of persistence levels $\mathcal{H}$ is used.It provides lower bounds on the persistence levels of hard references that may be directly dereferenced. Functions $\lambda(x{:}\tau)[pc;\mathcal{H}].e$ and function types $\tau_1 \xrightarrow{pc,\mathcal{H}} \tau_2$ are extended with an $\mathcal{H}$ component, which is an upper bound on the $\mathcal{H}$ levels of the caller.

Formally, $\mathcal{H}$ is drawn from the bounded meet-semilattice given by the upper powerdomain [22] of persistence levels. The elements of the powerdomain are the finitely generated subsets[3] of $\mathcal{L}$, modulo equivalence relation (1) below. The ordering on these elements is given by (2). If we choose maximal sets to represent the equivalence classes, then the meet operation is set union.

$$A \sim B \overset{def.}{\iff} \forall \ell \in \mathcal{L}.\ ((\exists a \in A.\ a \preccurlyeq \ell) \Leftrightarrow (\exists b \in B.\ b \preccurlyeq \ell)) \tag{1}$$

$$A \preccurlyeq_{\wp} B \overset{def.}{\iff} \forall b \in B.\ \exists a \in A.\ a \preccurlyeq b \tag{2}$$

[3]Non-empty subsets that either are finite or contain $\perp$.

$$\left[\begin{array}{c}\text{SOFT-}\\\text{SELECT}\end{array}\right]\quad \frac{\langle m^S.x_c, M\rangle \xrightarrow{\mathsf{e}} \langle v, M\rangle}{\langle (\mathsf{soft}\ m^S).x_c, M\rangle \xrightarrow{\mathsf{e}} \langle v, M\rangle} \qquad \left[\begin{array}{c}\text{SOFT-}\\\text{ASSIGN}\end{array}\right]\quad \frac{\langle m^S.x_c := v, M\rangle \xrightarrow{\mathsf{e}} \langle v', M'\rangle}{\langle (\mathsf{soft}\ m^S).x_c := v, M\rangle \xrightarrow{\mathsf{e}} \langle v', M'\rangle}$$

$$\left[\begin{array}{c}\text{TRY-}\\\text{VAL}\end{array}\right]\quad \frac{\forall p'.\ v \neq \bot_{p'}}{\langle \mathsf{try}\ v\ \mathsf{catch}\ p\colon e, M\rangle \xrightarrow{\mathsf{e}} \langle v, M\rangle} \qquad \left[\begin{array}{c}\text{TRY-}\\\text{CATCH}\end{array}\right]\quad \frac{p \preccurlyeq p'}{\langle \mathsf{try}\ \bot_{p'}\ \mathsf{catch}\ p\colon e, M\rangle \xrightarrow{\mathsf{e}} \langle e, M\rangle}$$

$$\left[\begin{array}{c}\text{TRY-}\\\text{ESC}\end{array}\right]\quad \frac{p \not\preccurlyeq p'}{\langle \mathsf{try}\ \bot_{p'}\ \mathsf{catch}\ p\colon e, M\rangle \xrightarrow{\mathsf{e}} \langle \bot_{p'}, M\rangle} \qquad E ::= \ldots \mid \mathsf{try}\ [\cdot]\ \mathsf{catch}\ p\colon e$$

$$\left[\begin{array}{c}\text{T-SOFT-}\\\text{SELECT}\end{array}\right]\quad \frac{\begin{array}{c}\Gamma;pc;\mathcal{H} \vdash v : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_r)_w, \top \qquad \vdash \mathsf{auth}^+(\tau_c) \preccurlyeq pc\\ p = \mathsf{auth}^-(r) \sqcap \mathsf{persist}(r) \sqcap w \qquad \vdash \mathcal{H} \preccurlyeq p\end{array}}{\Gamma;pc;\mathcal{H} \vdash v.x_c : \tau_c \sqcap p, p}$$

$$\left[\begin{array}{c}\text{T-SOFT-}\\\text{ASSIGN}\end{array}\right]\quad \frac{\begin{array}{c}\Gamma;pc;\mathcal{H} \vdash v_1 : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_r)_w, \top \qquad p = \mathsf{auth}^-(r) \sqcap \mathsf{persist}(r) \sqcap w\\ \Gamma;pc;\mathcal{H} \vdash v_2 : \tau, \top \qquad \vdash \tau \sqcap pc \sqcap p \leq \tau_c \qquad \vdash \mathsf{auth}^+(\tau) \preccurlyeq pc \sqcap p \qquad \vdash \mathcal{H} \preccurlyeq p\end{array}}{\Gamma;pc;\mathcal{H} \vdash v_1.x_c := v_2 : \mathbf{1}, p}$$

$$[\text{T-TRY}]\quad \frac{\begin{array}{c}\Gamma;pc;\mathcal{H}, p \vdash e_1 : \tau, \mathcal{X}_1 \qquad w = \displaystyle\prod_{p' \in \mathcal{X}_1} (p \sqcup p')\\ \Gamma;pc \sqcap w \sqcap \mathsf{integ}(\tau);\mathcal{H} \vdash e_2 : \tau, \mathcal{X}_2 \qquad \vdash \mathsf{auth}^+(\tau) \preccurlyeq pc\end{array}}{\Gamma;pc;\mathcal{H} \vdash \mathsf{try}\ e_1\ \mathsf{catch}\ p\colon e_2 : \tau \sqcap w, (\mathcal{X}_1/p) \sqcap \mathcal{X}_2}$$

Figure 9: Additional small-step evaluation and typing rules for $\lambda_{persist}$

6.2 Example

Returning to the directory example in Figure 1, Alice can add a place to the list of sightseeing ideas with the code below. This code starts at Alice's docs directory, traverses the reference to the scratchpad, and invokes an add method to add a museum.

```
let pad = docs.scratchpad
in try pad.add "Rodin Museum" catch ⊥: ...
```

The expression pad.add follows a hard reference to the scratchpad. Despite the hard reference, a try is needed because Alice does not trust host U to persist the scratchpad.

6.3 Static and dynamic semantics of $\lambda_{persist}$

The small-step operational semantics of $\lambda_{persist}$ extends that of $\lambda^0_{persist}$ with the rules in the top of Figure 9. Failures propagate outward dynamically (TRY-ESC) until either they are handled by a failure handler (TRY-CATCH), or the whole program fails. Appendix A.2 gives the full operational semantics for $\lambda_{persist}$.

The subtyping rules are the same as for $\lambda^0_{persist}$, except that function subtyping is also contravariant on the $\mathcal{H}$ component. Full subtyping rules are given in Appendix A.3.

The typing rules for $\lambda_{persist}$ extend those for $\lambda^0_{persist}$. They augment the typing context with a *handler environment* $\mathcal{H}$, indicating the set of persistence failures the evaluation context can handle. For an expression e that is well-typed in a context $\Gamma;pc;\mathcal{H}$, typing judgments additionally produce an effect $\mathcal{X}$, which is a set indicating the persistence failures that can occur during evaluation. The typing assertion $\Gamma;pc;\mathcal{H} \vdash e : \tau, \mathcal{X}$, therefore, means that the expression e has type τ and effect $\mathcal{X}$ under type assignment Γ, current program-counter label pc, and handler environment $\mathcal{H}$.

The typing rules for $\lambda^0_{persist}$ are converted straightforwardly to thread $\mathcal{H}$ and $\mathcal{X}$ through typing judgments. Rules T-SEL and T-ASGN gain premises to ensure the context has a suitable handler in case dereferences fail. Appendix A.4 gives the full set of converted rules.

The bottom of Figure 9 gives three new typing rules. T-SOFT-SELECT and T-SOFT-ASSIGN check direct uses of soft references. They taint the integrity of the dereference with $\mathsf{auth}^-(r)$ because the result of the dereference is affected by those able to pin the referent in memory by creating a hard reference (Section 3.5). Rule T-TRY checks try expressions. To reflect the installation of a p-persistence handler, p is added to the handler environment $\mathcal{H}$ when checking e_1. The value w in the typing rule is a conservative summary of the persistence errors that can occur while evaluating e_1 and are not handled by the p-persistence handler. Because evaluation of e_2 depends on the result of e_1, the pc label for evaluating e_2 is tainted by w. In this rule, the notation $\mathcal{X}/p$ denotes the subset of persistence errors $\mathcal{X}$

$$[\textsc{Create}]\quad \frac{m = \mathsf{newloc}(M) \qquad S = \{\overrightarrow{x_i : \tau_i}\}_s \qquad v_i' = v_i \blacktriangleright_\alpha \tau_i}{\left\langle \{\overrightarrow{x_i = v_i}\}^S, M \right\rangle \xrightarrow{\mathsf{e}} \left\langle m^S, M[m^S \mapsto \{\overrightarrow{x_i = v_i'}\}] \right\rangle}$$

$$[\textsc{Select}]\quad \frac{\begin{array}{c} M(m^S) = \{\overrightarrow{x_i = v_i}\} \\ S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)} \end{array}}{\left\langle m^S.x_c, M \right\rangle \xrightarrow{\mathsf{e}} \langle v_c \blacktriangleright_\alpha p, M \rangle} \qquad [\textsc{Assign}]\quad \frac{\begin{array}{c} M(m^S) \neq \bot \qquad \forall p'.\ v \neq \bot_{p'} \\ S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)} \\ M' = M[m^S.x_c \mapsto v \blacktriangleright_\alpha \tau_c] \end{array}}{\left\langle m^S.x_c := v, M \right\rangle \xrightarrow{\mathsf{e}} \left\langle * \blacktriangleright_\alpha p, M' \right\rangle}$$

$$\begin{bmatrix}\textsc{Dangle-}\\\textsc{Select}\end{bmatrix}\quad \frac{\begin{array}{c} M(m^S) = \bot \\ S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)} \end{array}}{\left\langle m^S.x_c, M \right\rangle \xrightarrow{\mathsf{e}} \langle \bot_p \blacktriangleright_\alpha p, M \rangle} \qquad \begin{bmatrix}\textsc{Dangle-}\\\textsc{Assign}\end{bmatrix}\quad \frac{\begin{array}{c} M(m^S) = \bot \\ S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)} \end{array}}{\left\langle m^S.x_c := v, M \right\rangle \xrightarrow{\mathsf{e}} \langle \bot_p \blacktriangleright_\alpha p, M \rangle}$$

$$\begin{bmatrix}\textsc{Soft-}\\\textsc{Select}\end{bmatrix}\quad \frac{\left\langle m^S.x_c, M \right\rangle \xrightarrow{\mathsf{e}} \langle v, M \rangle \qquad S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}}{\left\langle (\mathsf{soft}\ m^S).x_c, M \right\rangle \xrightarrow{\mathsf{e}} \langle v \blacktriangleright_\alpha (a \sqcap p), M \rangle}$$

$$\begin{bmatrix}\textsc{Soft-}\\\textsc{Assign}\end{bmatrix}\quad \frac{\left\langle m^S.x_c := v, M \right\rangle \xrightarrow{\mathsf{e}} \left\langle v', M' \right\rangle \qquad S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}}{\left\langle (\mathsf{soft}\ m^S).x_c := v, M \right\rangle \xrightarrow{\mathsf{e}} \left\langle v' \blacktriangleright_\alpha (a \sqcap p), M' \right\rangle}$$

$$\begin{bmatrix}\textsc{Exists-}\\\textsc{True}\end{bmatrix}\quad \frac{M(m^S) \neq \bot \qquad S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}}{\left\langle \mathsf{exists\ soft}\ m^S\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2, M \right\rangle \xrightarrow{\mathsf{e}} \left\langle (e_1\{m^S/x\}) \blacktriangleright_\alpha (a \sqcap p), M \right\rangle}$$

$$\begin{bmatrix}\textsc{Exists-}\\\textsc{False}\end{bmatrix}\quad \frac{M(m^S) \neq \bot \qquad S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}}{\left\langle \mathsf{exists\ soft}\ m^S\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2, M \right\rangle \xrightarrow{\mathsf{e}} \langle e_2 \blacktriangleright_\alpha (a \sqcap p), M \rangle}$$

$$\begin{bmatrix}\textsc{Bracket-}\\\textsc{Select}\end{bmatrix}\quad \left\langle [m^S].x_c, M \right\rangle \xrightarrow{\mathsf{e}} \left\langle [m^S.x_c], M \right\rangle \qquad \begin{bmatrix}\textsc{Bracket-}\\\textsc{Assign}\end{bmatrix}\quad \left\langle [m^S].x_c := v, M \right\rangle \xrightarrow{\mathsf{e}} \left\langle [m^S.x_c := v], M \right\rangle$$

$$\begin{bmatrix}\textsc{Bracket-}\\\textsc{Soft-Select}\end{bmatrix}\quad \begin{array}{l} \left\langle [\mathsf{soft}\ m^S].x_c, M \right\rangle \\ \quad \xrightarrow{\mathsf{e}} \left\langle [(\mathsf{soft}\ m^S).x_c], M \right\rangle \end{array} \qquad \begin{bmatrix}\textsc{Bracket-}\\\textsc{Soft-Assign}\end{bmatrix}\quad \begin{array}{l} \left\langle [\mathsf{soft}\ m^S].x_c := v, M \right\rangle \\ \quad \xrightarrow{\mathsf{e}} \left\langle [(\mathsf{soft}\ m^S).x_c := v], M \right\rangle \end{array}$$

$$\begin{bmatrix}\textsc{Bracket-}\\\textsc{Soft}\end{bmatrix}\quad \left\langle \mathsf{soft}\ [m^S], M \right\rangle \xrightarrow{\mathsf{e}} \left\langle [\mathsf{soft}\ m^S], M \right\rangle \qquad \begin{bmatrix}\textsc{Double-}\\\textsc{Bracket}\end{bmatrix}\quad \langle [[v]], M \rangle \xrightarrow{\mathsf{e}} \langle [v], M \rangle$$

$$\begin{bmatrix}\textsc{Bracket-}\\\textsc{Exists}\end{bmatrix}\quad \begin{array}{l} \langle \mathsf{exists}\ [v]\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2, M \rangle \\ \quad \xrightarrow{\mathsf{e}} \langle [\mathsf{exists}\ v\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2], M \rangle \end{array} \qquad \begin{bmatrix}\textsc{Bracket-}\\\textsc{Apply}\end{bmatrix}\quad \begin{array}{l} \langle [\lambda(x{:}\tau)[pc;\mathcal{H}].e]\ v, M \rangle \\ \quad \xrightarrow{\mathsf{e}} \langle [(\lambda(x{:}\tau)[pc;\mathcal{H}].e)\ v], M \rangle \end{array}$$

$$\begin{bmatrix}\textsc{Bracket-}\\\textsc{Try}\end{bmatrix}\quad \begin{array}{l} \langle \mathsf{try}\ [v]\ \mathsf{catch}\ p{:}\ e, M \rangle \\ \quad \xrightarrow{\mathsf{e}} \langle [\mathsf{try}\ v\ \mathsf{catch}\ p{:}\ e], M \rangle \end{array} \qquad \begin{bmatrix}\textsc{Bracket-}\\\textsc{If}\end{bmatrix}\quad \begin{array}{l} \langle \mathsf{if}\ [v]\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2, M \rangle \\ \quad \xrightarrow{\mathsf{e}} \langle [\mathsf{if}\ v\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2], M \rangle \end{array}$$

$$\begin{bmatrix}\textsc{Bracket-}\\\textsc{Let}\end{bmatrix}\quad \frac{\forall p.\ v \neq \bot_p}{\langle \mathsf{let}\ x = [v]\ \mathsf{in}\ e, M \rangle \xrightarrow{\mathsf{e}} \langle [e\{[v]/x\}], M \rangle}$$

$$\begin{bmatrix}\textsc{Bracket-}\\\textsc{Context}\end{bmatrix}\quad \frac{\langle e, M \rangle \xrightarrow{\mathsf{e}} \left\langle e', M' \right\rangle}{\langle [e], M \rangle \xrightarrow{\mathsf{e}} \left\langle [e'], M' \right\rangle} \qquad \begin{bmatrix}\textsc{Bracket-}\\\textsc{Fail}\end{bmatrix}\quad \langle F[[\bot_p]], M \rangle \xrightarrow{\mathsf{e}} \langle [\bot_p], M \rangle$$

Figure 10: Small-step operational semantics extensions for ordinary execution of $[\lambda_{persist}]$

not handled by p. Formally, this is defined as follows.

$$\mathcal{H}/p \triangleq \{p' \in \mathcal{H} : p \not\preccurlyeq p'\}$$

7 The power of the adversary

Possible actions of the adversary are modeled by extending the operational semantics of Figure 5 with more transitions. To support reasoning about what an adversary may have affected in a partially evaluated program, $\lambda_{persist}$ is also augmented to include bracketed expressions, resulting in the language $[\lambda_{persist}]$. The term $[e]$ represents an expression e that may have been influenced by the adversary, and $[v]$ is an influenced value. The operational semantics is extended by adding rules that propagate these brackets in the obvious manner. (Doubly bracketed values are considered expressions, not values.)

The extended syntax and the rule for typing bracketed terms appear below. Extensions for the operational semantics appear in Figure 10.

$$\begin{array}{ll} \text{Values} & v ::= \ldots \mid [v] \\ \text{Terms} & e ::= \ldots \mid [e] \end{array} \qquad [\text{T-}\textsc{Bracket}]\quad \frac{\Gamma; pc \sqcap \ell; \mathcal{H} \vdash e : \tau, \mathcal{X} \qquad \alpha \not\preccurlyeq \ell \qquad \vdash \mathsf{auth}^+(\tau) \preccurlyeq pc \sqcap \ell}{\Gamma; pc; \mathcal{H} \vdash [e] : \tau \sqcap \ell, \mathcal{X}}$$

Rules CREATE, SELECT, DANGLE-SELECT, ASSIGN, DANGLE-ASSIGN, SOFT-SELECT, SOFT-ASSIGN, EXISTS-TRUE, and EXISTS-FALSE are amended to ensure low-integrity expressions are bracketed. To do this, they use the

$$[\alpha\text{-}\textsc{Create}] \quad \frac{\begin{array}{c} m = \mathsf{newloc}(M) \qquad \varnothing;\top;\top \vdash \{\overrightarrow{x_i = [v_i]}\}^S : R_\top, \top \\ \vdash^{\alpha}_{[wf]} M[m^S \mapsto \{\overrightarrow{x_i = [v_i]}\}] \qquad \alpha \not\leq \mathsf{persist}(S) \end{array}}{\langle e, M\rangle \rightsquigarrow_\alpha \left\langle e, M[m^S \mapsto \{\overrightarrow{x_i = [v_i]}\}]\right\rangle}$$

$$[\alpha\text{-}\textsc{Assign}] \quad \frac{\begin{array}{c} m^S \in \mathsf{dom}(M) \qquad M(m^S) \neq \bot \\ S = \{\overrightarrow{x_i : \tau_i}\}_s \qquad \varnothing;\top;\top \vdash [v] : \tau_c, \top \\ \vdash^{\alpha}_{[wf]} M[m^S.x_c \mapsto [v]] \end{array}}{\langle e, M\rangle \rightsquigarrow_\alpha \left\langle e, M[m^S.x_c \mapsto [v]]\right\rangle} \qquad [\alpha\text{-}\textsc{Forget}] \quad \frac{\begin{array}{c} m^S \in \mathsf{dom}(M) \\ \alpha \not\leq \mathsf{persist}(S) \end{array}}{\langle e, M\rangle \rightsquigarrow_\alpha \left\langle e, M[m^S \mapsto \bot]\right\rangle}$$

Figure 11: Effects caused by the α-adversary

auto-bracketing function $e \blacktriangleright_\alpha \ell$, which is formally defined below. The notation $e \blacktriangleright_\alpha \tau$ is shorthand for $e \blacktriangleright_\alpha \mathsf{integ}(\tau)$.

$$e \blacktriangleright_\alpha \ell = \begin{cases} e, & \text{if } \vdash \alpha \leq \ell \text{ or } \exists e'.\ e = [e']; \\ [e], & \text{otherwise.} \end{cases}$$

Also, rules Try-Val and Let are amended to prevent a transition when v is bracketed, and rules that disallow transitions on bottom values ($\bot_p$) are amended to prevent transitions on bracketed bottom values.

The adversary is powerful, as shown by the transitions defined in Figure 11. Adversaries may create new records, modify existing records, or remove records from memory altogether, but their ability is bounded by an integrity label $\alpha \in \mathcal{L}$. Such an α-*adversary* has all creation authority except α and higher, can modify any record field except those with α (or higher) integrity, and can delete any record except those with α (or higher) persistence. A small evaluation step taken in the presence of an α-adversary is a transition from a machine configuration $\langle e, M\rangle$ to another configuration $\langle e', M'\rangle$, written $\langle e, M\rangle \to_\alpha \langle e', M'\rangle$.

It is important to know that any evaluation of a program in the original language can be simulated in the augmented language, which amounts to showing that the rules cover all the ways that brackets can appear. This is proved straightforwardly by induction on the evaluation rules (see Lemma 1 in Section 8).

The adversary's transitions embody a simplifying assumption that the adversary can only create well-typed values. While it is reasonable to allow the adversary to create ill-typed values, an implementation with run-time type checking can catch ill-typed values when they cross between hosts and replace them with well-typed default values.

Rule α-Create lets the adversary create records at new memory locations. The premise $\varnothing;\top;\top \vdash \{\overrightarrow{x_i = [v_i]}\}^S : R_\top, \top$ ensures that the records are well-typed values and that new hard references satisfy the restrictions on the adversary. The premise $\vdash^{\alpha}_{[wf]} M[m^S \mapsto \{\overrightarrow{x_i = [v_i]}\}]$ ensures that the resulting memory is well-formed (formally defined in Section 8.1), so the adversary cannot create references to unknown memory locations.

Rule α-Assign lets the adversary modify existing records. The premise $M(m^S) \neq \bot$ ensures that the record being modified still exists in memory. The premise $\varnothing;\top;\top \vdash [v] : \tau_c, \top$ ensures that the assignment is well-typed and that new hard references satisfy the restrictions on the adversary. The premise $\vdash^{\alpha}_{[wf]} M[m^S.x_c \mapsto [v]]$ ensures that the resulting memory is well-formed, so the adversary cannot create references to unknown memory locations.

Rule α-Forget lets the adversary drop records from memory. The premise $\alpha \not\leq \mathsf{persist}(S)$ restricts the persistence level of dropped records.

8 Results

The goal of $\lambda_{persist}$ is to prevent accidental persistence and to ensure that the adversary cannot damage referential integrity or cause storage attacks. Accidental persistence is prevented by the use of persistence policies. We now show how to formalize the other security properties and prove that $\lambda_{persist}$ satisfies these properties.

8.1 Well-formedness

A well-formed $\lambda_{persist}$ memory M, written $\vdash_{wf} M$, maps typed locations to record values with the same type. This is defined formally as follows.

Definition 3 (Well-formed $\lambda_{persist}$ memory)**.** *A $\lambda_{persist}$ memory M is well-formed, written $\vdash_{wf} M$, if each record stored in M satisfies two conditions: the record's type corresponds to the type of the record's location in M, and every location mentioned in the record is valid in M.*

More precisely, whenever M maps a reference m^S to a record value $\{\overrightarrow{x_i = v_i}\}$,

- *S is well-formed,*
- *each v_i has the appropriate type (as specified by S),*
- *the locations mentioned in the field values v_i are in the domain of M:*

$$\begin{aligned} \vdash_{wf} M \overset{def.}{\iff} & (M(m^S) = \{\overrightarrow{x_i = v_i}\} \wedge S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)} \\ & \Rightarrow \ \vdash_{wf} S : \mathsf{rectype} \\ & \quad \wedge (\forall i.\ \varnothing; \top; \top \vdash v_i : \tau_i, \top) \\ & \quad \wedge \mathsf{locs}(\{\overrightarrow{x_i = v_i}\}) \subseteq \mathsf{dom}(M)) \end{aligned}$$

Our notion of a well-formed configuration relies on knowing when a location is noncollectible (cannot be garbage collected), which we now define.

Definition 4 (Noncollectible locations)**.** *A memory location m^S is* noncollectible *in a configuration $\langle e,M\rangle$, written $\mathsf{nc}(m^S, \langle e,M\rangle)$, if it is reachable from a GC root of e through a path of hard references.*

This is defined formally by the following induction rules:

$$[\text{NC1}]\ \frac{\mathsf{root}(m^S, e)}{\mathsf{nc}(m^S, \langle e,M\rangle)} \qquad [\text{NC2}]\ \frac{\mathsf{root}(m_1^{S_1}, e) \qquad M(m_1^{S_1}) = \{\overrightarrow{x_i = v_i}\} \qquad \exists c.\ \mathsf{nc}(m^S, \langle v_c, M\rangle)}{\mathsf{nc}(m^S, \langle e,M\rangle)}$$

Well-formedness of configurations, defined below, is parameterized on an adversary α.

Definition 5 (Well-formed $\lambda_{persist}$ configuration)**.** *A $\lambda_{persist}$ configuration $\langle e,M\rangle$ is well-formed, written $\vdash_{wf}^{\alpha} \langle e,M\rangle$, if the following all hold:*

- *M is well-formed;*
- *the locations mentioned in e are valid in M;*
- *no noncollectible high-persistence location is deleted; and*
- *if G is a minimal collectible group in which a high-persistence location is deleted, then all locations in G are also deleted.*

Formally,

$$\begin{aligned} \vdash_{wf}^{\alpha} \langle e,M\rangle \overset{def.}{\iff} & \vdash_{wf} M \wedge \mathsf{locs}(e) \subseteq \mathsf{dom}(M) \\ & \wedge (\forall m^S.\mathsf{nc}(m^S, \langle e,M\rangle) \wedge \vdash \alpha \preccurlyeq \mathsf{persist}(S) \\ & \quad \Rightarrow M(m^S) \neq \bot) \\ & \wedge (\forall G.\ \mathsf{gc}(G, \langle e,M\rangle) \wedge (\nexists G' \subseteq G.\ \mathsf{gc}(G', \langle e,M\rangle)) \\ & \quad \wedge (\exists m_0^{S_0} \in G.\ \vdash \alpha \preccurlyeq \mathsf{persist}(S_0) \wedge M(m_0^{S_0}) = \bot) \\ & \quad \Rightarrow \forall m^S \in G.\ M(m^S) = \bot) \end{aligned}$$

A $\lambda_{persist}$ configuration is well-formed in a nonadversarial setting, written $\vdash_{wf} \langle e,M\rangle$, if it is well-formed with respect to the $\bot$ adversary.

Corresponding well-formedness conditions are defined similarly for $[\lambda_{persist}]$ memories (written $\vdash_{[wf]}^{\alpha} M$ and $\vdash_{[wf]} M$) and for $[\lambda_{persist}]$ configurations (written $\vdash_{[wf]}^{\alpha} \langle e,M\rangle$ and $\vdash_{[wf]} \langle e,M\rangle$). Well-formedness of $[\lambda_{persist}]$ memories is parameterized on an α-adversary, because values appearing in low-integrity record fields must be bracketed.

Definition 6 (Well-formed $[\lambda_{persist}]$ memory)**.** *A $[\lambda_{persist}]$ memory M is well-formed with respect to an adversary α (written $\vdash^{\alpha}_{[wf]} M$) if it is a well-formed $\lambda_{persist}$ memory and all low-integrity field values are bracketed:*

$$\vdash^{\alpha}_{[wf]} M \overset{def.}{\iff} \vdash_{wf} M \wedge (M(m^S) = \{\overrightarrow{x_i = v_i}\} \wedge S = \{\overrightarrow{x_i : \tau_i}\}_s \wedge \alpha \not\leq \mathsf{integ}(\tau_i) \\ \Rightarrow \exists v'.\ v_i = [v'])$$

Definition 7 (Well-formed $[\lambda_{persist}]$ configuration)**.** *A $[\lambda_{persist}]$ configuration $\langle e, M\rangle$ is well-formed if it is a well-formed $\lambda_{persist}$ configuration with a well-formed $[\lambda_{persist}]$ memory. A configuration is well-formed in a nonadversarial setting if it is well-formed in the presence of a $\perp$ adversary.*

$$\vdash^{\alpha}_{[wf]} \langle e, M\rangle \overset{def.}{\iff} \vdash^{\alpha}_{[wf]} M \wedge \vdash^{\alpha}_{wf} \langle e, M\rangle$$
$$\vdash_{[wf]} \langle e, M\rangle \overset{def.}{\iff} \vdash^{\perp}_{[wf]} \langle e, M\rangle$$

8.2 Completeness of $[\lambda_{persist}]$ evaluation

For $[\lambda_{persist}]$ to be adequate for reasoning about referential security properties of $\lambda_{persist}$, we must be able to simulate any $\lambda_{persist}$ execution in $[\lambda_{persist}]$. This is a completeness property. To do this, we first need to define what it means for a $[\lambda_{persist}]$ configuration to simulate a $\lambda_{persist}$ configuration. This involves defining a correspondence on expressions and heaps between the two languages. We write $e_1 \lesssim e_2$ to denote that the $\lambda_{persist}$ term e_1 corresponds to the $[\lambda_{persist}]$ term e_2. The terms correspond if erasing brackets from e_2 produces e_1.

Definition 8 (Correspondence between $[\lambda_{persist}]$ and $\lambda_{persist}$ expressions)**.** *A $[\lambda_{persist}]$ expression e is related to a $\lambda_{persist}$ expression e', written $\vdash e \lesssim e'$, if the two are equal when brackets in e are removed. This is formally defined by the following induction rules.*

$$\frac{e = e}{\vdash e \lesssim e} \qquad \frac{\vdash e \lesssim e'}{\vdash [e] \lesssim e'} \qquad \frac{\vdash e \lesssim e'}{\vdash \lambda(x{:}\tau)[pc;\mathcal{H}].e \lesssim \lambda(x{:}\tau)[pc;\mathcal{H}].e'} \qquad \frac{\vdash v_i \lesssim u_i \ (\forall i)}{\vdash v_1\ v_2 \lesssim u_1\ u_2}$$

$$\frac{\vdash e_i \lesssim e_i' \ (\forall i)}{\vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \lesssim \mathsf{if}\ e_1'\ \mathsf{then}\ e_2'\ \mathsf{else}\ e_3'} \qquad \frac{\vdash v_i \lesssim u_i \ (\forall i)}{\vdash \{\overrightarrow{x_i = v_i}\}^S \lesssim \{\overrightarrow{x_i = u_i}\}^S} \qquad \frac{\vdash v \lesssim u}{\vdash v.x \lesssim u.x} \qquad \frac{\vdash v_i \lesssim u_i \ (\forall i)}{\vdash v_1.x := v_2 \lesssim u_1.x := u_2}$$

$$\frac{\vdash e \lesssim e'}{\vdash \mathsf{soft}\ e \lesssim \mathsf{soft}\ e'} \qquad \frac{\vdash e_i \lesssim e_i' \ (\forall i)}{\vdash e_1 \parallel e_2 \lesssim e_1' \parallel e_2'} \qquad \frac{\vdash e_i \lesssim e_i' \ (\forall i)}{\vdash \mathsf{exists}\ e_1\ \mathsf{as}\ x : e_2\ \mathsf{else}\ e_3 \lesssim \mathsf{exists}\ e_1'\ \mathsf{as}\ x : e_2'\ \mathsf{else}\ e_3'}$$

$$\frac{\vdash e_i \lesssim e_i' \ (\forall i)}{\vdash \mathsf{try}\ e_1\ \mathsf{catch}\ p{:}\ e_2 \lesssim \mathsf{try}\ e_1'\ \mathsf{catch}\ p{:}\ e_2'} \qquad \frac{\vdash e_i \lesssim e_i' \ (\forall i)}{\vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \lesssim \mathsf{let}\ x = e_1'\ \mathsf{in}\ e_2'}$$

This correspondence induces one on heaps. Together, the two give the correspondence on configurations.

Definition 9 (Correspondence between $[\lambda_{persist}]$ and $\lambda_{persist}$ memories)**.** *A $[\lambda_{persist}]$ memory M_1 is related to a $\lambda_{persist}$ memory M_2, written $\vdash M_1 \lesssim M_2$, if they map the same set of locations, and the memories map each location to values that are related.*

$$\begin{aligned}\vdash M_1 \lesssim M_2 \overset{def.}{\iff} & \mathsf{dom}(M_1) = \mathsf{dom}(M_2) \\ & \wedge \forall m^S \in \mathsf{dom}(M_1).\ M_1(m^S) = M_2(m^S) = \perp \\ & \qquad \vee \vdash M_1(m^S) \lesssim M_2(m^S)\end{aligned}$$

Definition 10 (Correspondence between $[\lambda_{persist}]$ and $\lambda_{persist}$ configurations)**.** *A $[\lambda_{persist}]$ configuration $\langle e_1, M_1\rangle$ is related to a $\lambda_{persist}$ configuration $\langle e_2, M_2\rangle$, written $\vdash \langle e_1, M_1\rangle \lesssim \langle e_2, M_2\rangle$, if both the expressions and the memories are related:*

$$\vdash \langle e_1, M_1\rangle \lesssim \langle e_2, M_2\rangle \overset{def.}{\iff} \vdash e_1 \lesssim e_2 \wedge \vdash M_1 \lesssim M_2$$

We can now state and prove the completeness of $[\lambda_{persist}]$.

Lemma 1 (Completeness of $[\lambda_{persist}]$ with respect to $\lambda_{persist}$)**.** *Let $\langle e_1, M_1\rangle$ be a well-formed $\lambda_{persist}$ configuration with e_1 well-typed. Let $\langle e_2, M_2\rangle$ be a well-formed $[\lambda_{persist}]$ configuration corresponding to $\langle e_1, M_1\rangle$, with e_2 well-typed. If $\langle e_1, M_1\rangle$ takes a $\rightarrow$ transition to $\langle e_1', M_1'\rangle$, then there exists a configuration $\langle e_2', M_2'\rangle$ corresponding to $\langle e_1', M_1'\rangle$ such that $\langle e_2, M_2\rangle \rightarrow_\alpha^* \langle e_2', M_2'\rangle$.*

$$\begin{aligned}
&\vdash_{wf}^{\alpha} \langle e_1, M_1\rangle \wedge \varnothing; pc; \mathcal{H} \vdash e_1 : \tau, \mathcal{X} \\
&\wedge \vdash_{[wf]}^{\alpha} \langle e_2, M_2\rangle \wedge \varnothing; pc; \mathcal{H} \vdash e_2 : \tau, \mathcal{X} \\
&\wedge \vdash \langle e_1, M_1\rangle \lesssim \langle e_2, M_2\rangle \wedge \langle e_1, M_1\rangle \rightarrow \left\langle e_1', M_1'\right\rangle \\
&\qquad \Rightarrow \exists e_2', M_2'.\, \langle e_2, M_2\rangle \rightarrow_\alpha^* \left\langle e_2', M_2'\right\rangle \wedge \left\langle e_1', M_1'\right\rangle \lesssim \left\langle e_2', M_2'\right\rangle
\end{aligned}$$

Proof. Induction on the derivation of $\langle e_1, M_1\rangle \rightarrow_\alpha \langle e_1', M_1'\rangle$. □

8.3 Soundness of $[\lambda_{persist}]$ type system

We prove the type system sound with the usual method, via type preservation (Lemma 9) and progress (Lemma 13). Because we are only concerned with well-formed configurations, it is important to know that they are preserved by the operational semantics. This is captured by Lemma 10.

We first show some preliminary results for weakening the typing context.

Lemma 2 (Type-environment weakening)**.** *Extra type assumptions can be safely added to typing contexts:* $\Gamma; pc; \mathcal{H} \vdash e : \tau, \mathcal{X} \wedge x \notin \mathsf{FV}(e) \Rightarrow \Gamma, x{:}\tau'; pc; \mathcal{H} \vdash e : \tau, \mathcal{X}$.

Proof. By induction on the derivation of $\Gamma; pc; \mathcal{H} \vdash e : \tau, \mathcal{X}$. □

Lemma 3 (*pc* weakening)**.** *The pc label in a typing context can be safely raised.*

$$\vdash pc \preccurlyeq pc' \wedge \Gamma; pc; \mathcal{H} \vdash e : \tau, \mathcal{X} \Rightarrow \Gamma; pc'; \mathcal{H} \vdash e : \tau, \mathcal{X}$$

Proof. By induction on the derivation of $\Gamma; pc; \mathcal{H} \vdash e : \tau, \mathcal{X}$.

Rules T-BOOL, T-UNIT, T-LOC, T-BOTTOM, and T-VAR are trivial base cases. Rule T-ABS follows from the transitivity of $\preccurlyeq$. Rules T-SOFT, T-PARALLEL, and T-SUBSUME follow from the induction hypothesis.

Rules T-RECORD, T-SELECT, T-SOFT-SELECT, T-ASSIGN, T-SOFT-ASSIGN, T-EXISTS, T-APP, T-LET, T-TRY, T-IF, and T-BRACKET follow from the induction hypothesis and the transitivity of $\preccurlyeq$. In rules T-ASSIGN and T-SOFT-ASSIGN, we need to show $\vdash \tau \sqcap pc' \sqcap p \leq \tau_c$. By assumption, we have $\vdash \tau \sqcap pc \sqcap p \leq \tau_c$. Let $b_w = \tau$. Then using S3, we can show

$$\frac{\vdash b \leq b \qquad \vdash w \sqcap pc \sqcap p \preccurlyeq w \sqcap pc' \sqcap p}{\vdash \tau \sqcap pc' \sqcap p \leq \tau \sqcap pc \sqcap p},$$

and the result follows from transitivity of subtyping. □

Lemma 4 (Handler weakening)**.** *Extra handler assumptions can be safely added to the typing context.*

$$\vdash \mathcal{H}' \preccurlyeq \mathcal{H} \wedge \Gamma; pc; \mathcal{H} \vdash e : \tau, \mathcal{X} \Rightarrow \Gamma; pc; \mathcal{H}' \vdash e : \tau, \mathcal{X}$$

Proof. By induction on the derivation of $\Gamma; pc; \mathcal{H} \vdash e : \tau, \mathcal{X}$.

Rules T-BOOL, T-UNIT, T-LOC, T-BOTTOM, T-VAR, T-ABS, and T-PARALLEL are trivial base cases.

Rules T-SOFT, T-RECORD, T-SELECT, T-ASSIGN, T-SOFT-SELECT, T-SOFT-ASSIGN, T-EXISTS, T-APP, T-LET, T-IF, T-SUBSUME, and T-BRACKET follow from the induction hypothesis.

Rule T-TRY follows from the induction hypothesis and the fact that $\mathcal{H}' \cup \{p\} \preccurlyeq \mathcal{H} \cup \{p\}$. □

Corollary 5 summarizes these results.

Corollary 5 (Context weakening)**.** *Suppose $\Gamma; pc; \mathcal{H} \vdash e : \tau, \mathcal{X}$ with $x \notin \mathsf{FV}(e)$, $pc \preccurlyeq pc'$ and $\mathcal{H}' \preccurlyeq \mathcal{H}$. Then*

$$\Gamma, x{:}\tau'; pc'; \mathcal{H}' \vdash e : \tau, \mathcal{X}$$

We can now prove the substitution lemma.

Lemma 6 (Substitution)**.** $\Gamma, x{:}\tau'; pc; \mathcal{H} \vdash e : \tau, \mathcal{X} \wedge \varnothing; pc; \mathcal{H} \vdash v : \tau', \top \Rightarrow \Gamma; pc; \mathcal{H} \vdash e\{v/x\} : \tau, \mathcal{X}$.

Proof. By induction on the derivation of $\Gamma, x:\tau'; pc; \mathcal{H} \vdash e : \tau, \mathcal{X}$. Note that since v was typed in an empty type assignment ($\Gamma = \varnothing$), we must have $\mathsf{FV}(v) = \varnothing$.

Rules T-BOOL, T-UNIT, T-LOC, and T-BOTTOM are trivial base cases.

Rules T-SOFT, T-RECORD, T-SELECT, T-ASSIGN, T-SOFT-SELECT, T-SOFT-ASSIGN, T-APP, T-PARALLEL, T-TRY, T-IF, T-SUBSUME, and T-BRACKET follow from the definition of substitution and the induction hypothesis.

Case T-VAR:

Suppose $e = x$. Then $e\{v/x\} = v$ and $\tau' = \tau$ and the result holds in this case via Corollary 5 and T-SUBSUME. Alternatively, suppose $e = y \neq x$. In this case, $e\{v/x\} = e$ and the result holds trivially.

Case T-ABS ($e = \lambda(y{:}\tau_1)[pc_1; \mathcal{H}_1].e_1$):

If $y = x$, then $e\{v/x\} = e$ and the result holds trivially. Alternatively, suppose $y \neq x$. We have $\mathsf{FV}(v) = \varnothing$, so $e\{v/x\} = \lambda(y{:}\tau_1)[pc_1; \mathcal{H}_1].e_1\{v/x\}$. Let $\Gamma' = \Gamma, x{:}\tau'$. From the typing of e, we have

$$\frac{\begin{array}{c}\Gamma, x{:}\tau', y{:}\tau_1; pc_1; \mathcal{H}_1 \vdash e_1 : \tau_2, \mathcal{H}_1 \\ \vdash_{wf} (\tau_1 \xrightarrow{pc_1, \mathcal{H}_1} \tau_2)_\top : \mathsf{type} \qquad \vdash pc_1 \preccurlyeq pc\end{array}}{\Gamma, x{:}\tau'; pc; \mathcal{H} \vdash \lambda(y{:}\tau_1)[pc_1; \mathcal{H}_1].e_1 : (\tau_1 \xrightarrow{pc_1, \mathcal{H}_1} \tau_2)_\top, \top}\,,$$

where $\tau = (\tau_1 \xrightarrow{pc_1, \mathcal{H}_1} \tau_2)_\top$ and $\mathcal{X} = \top$. So, we know $\Gamma, x{:}\tau', y{:}\tau_1; pc_1; \mathcal{H}_1 \vdash e_1 : \tau_2, \mathcal{H}_1$. Therefore, by the induction hypothesis, we have $\Gamma, y{:}\tau_1; pc_1; \mathcal{H}_1 \vdash e_1\{v/x\} : \tau_2, \mathcal{H}_1$. So the result holds in this case via an application of T-ABS:

$$\frac{\begin{array}{c}\Gamma, y{:}\tau_1; pc_1; \mathcal{H}_1 \vdash e_1\{v/x\} : \tau_2, \mathcal{H}_1 \\ \vdash_{wf} (\tau_1 \xrightarrow{pc_1, \mathcal{H}_1} \tau_2)_\top : \mathsf{type} \qquad \vdash pc_1 \preccurlyeq pc\end{array}}{\Gamma; pc; \mathcal{H} \vdash \lambda(y{:}\tau_1)[pc_1; \mathcal{H}_1].e_1\{v/x\} : (\tau_1 \xrightarrow{pc_1, \mathcal{H}_1} \tau_2)_\top, \top}\,.$$

Case T-EXISTS ($\mathsf{exists}\ u\ \mathsf{as}\ y : e_1\ \mathsf{else}\ e_2$):

From the typing of e, we have

$$\frac{\begin{array}{c}\Gamma, x{:}\tau'; pc; \mathcal{H} \vdash u : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_r)_w, \top \qquad \vdash \mathsf{auth}^+(r) \preccurlyeq pc \sqcap w \qquad w' = \mathsf{auth}^-(r) \sqcap \mathsf{persist}(r) \sqcap w \\ \Gamma, x{:}\tau', y{:}(\{\overrightarrow{x_i : \tau_i}\}_r)_w; pc'; \mathcal{H} \vdash e_1 : \tau'', \mathcal{X}_1 \qquad \Gamma, x{:}\tau'; pc'; \mathcal{H} \vdash e_2 : \tau'', \mathcal{X}_2 \qquad \vdash \mathsf{auth}^+(\tau'') \preccurlyeq pc'\end{array}}{\Gamma, x{:}\tau'; pc; \mathcal{H} \vdash \mathsf{exists}\ u\ \mathsf{as}\ y : e_1\ \mathsf{else}\ e_2 : \tau'' \sqcap w', \mathcal{X}_1 \sqcap \mathcal{X}_2}\,,$$

where $pc' = pc \sqcap w'$, $\tau = \tau'' \sqcap w'$, and $\mathcal{X} = \mathcal{X}_1 \sqcap \mathcal{X}_2$. By the induction hypothesis, we therefore have

$$\Gamma; pc; \mathcal{H} \vdash u\{v/x\} : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_r)_w, \top \tag{3}$$
$$\Gamma; pc'; \mathcal{H} \vdash e_2\{v/x\} : \tau'', \mathcal{X}_2 \tag{4}$$

Suppose $y = x$. Then $e\{v/x\} = \mathsf{exists}\ u\{v/x\}\ \mathsf{as}\ y : e_1\ \mathsf{else}\ e_2\{v/x\}$, so from (3) and (4), the result holds in this case via an application of T-EXISTS:

$$\frac{\begin{array}{c}\Gamma; pc; \mathcal{H} \vdash u\{v/x\} : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_r)_w, \top \qquad \vdash \mathsf{auth}^+(r) \preccurlyeq pc \sqcap w \qquad w' = \mathsf{auth}^-(r) \sqcap \mathsf{persist}(r) \sqcap w \\ \Gamma, y{:}(\{\overrightarrow{x_i : \tau_i}\}_r)_w; pc'; \mathcal{H} \vdash e_1 : \tau'', \mathcal{X}_1 \qquad \Gamma; pc'; \mathcal{H} \vdash e_2\{v/x\} : \tau'', \mathcal{X}_2 \qquad \vdash \mathsf{auth}^+(\tau'') \preccurlyeq pc'\end{array}}{\Gamma; pc; \mathcal{H} \vdash \mathsf{exists}\ u\{v/x\}\ \mathsf{as}\ y : e_1\ \mathsf{else}\ e_2\{v/x\} : \tau'' \sqcap w', \mathcal{X}_1 \sqcap \mathcal{X}_2}\,.$$

Alternatively, suppose $y \neq x$. We have $\mathsf{FV}(v) = \varnothing$, so $e\{v/x\} = \mathsf{exists}\ u\{v/x\}\ \mathsf{as}\ y : e_1\{v/x\}\ \mathsf{else}\ e_2\{v/x\}$. From the typing of e, we know $\Gamma, x{:}\tau', y{:}(\{\overrightarrow{x_i : \tau_i}\}_r)_w; pc'; \mathcal{H} \vdash e_1 : \tau'', \mathcal{X}_1$. By the induction hypothesis, we have $\Gamma, y{:}(\{\overrightarrow{x_i : \tau_i}\}_r)_w; pc'; \mathcal{H} \vdash e_1\{v/x\} : \tau'', \mathcal{X}_1$. From this, (3), and (4), the result holds in this case via an application of T-EXISTS:

$$\frac{\begin{array}{c}\Gamma; pc; \mathcal{H} \vdash u\{v/x\} : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_r)_w, \top \qquad \vdash \mathsf{auth}^+(r) \preccurlyeq pc \sqcap w \qquad w' = \mathsf{auth}^-(r) \sqcap \mathsf{persist}(r) \sqcap w \\ \Gamma, y{:}(\{\overrightarrow{x_i : \tau_i}\}_r)_w; pc'; \mathcal{H} \vdash e_1\{v/x\} : \tau'', \mathcal{X}_1 \qquad \Gamma; pc'; \mathcal{H} \vdash e_2\{v/x\} : \tau'', \mathcal{X}_2 \qquad \vdash \mathsf{auth}^+(\tau'') \preccurlyeq pc'\end{array}}{\Gamma; pc; \mathcal{H} \vdash \mathsf{exists}\ u\{v/x\}\ \mathsf{as}\ y : e_1\{v/x\}\ \mathsf{else}\ e_2\{v/x\} : \tau'' \sqcap w', \mathcal{X}_1 \sqcap \mathcal{X}_2}\,.$$

Case T-LET ($e = \mathsf{let}\ y = e_1\ \mathsf{in}\ e_2$):

From the typing of e, we have

$$\frac{\begin{array}{ccc}\Gamma,x{:}\tau';pc;\mathcal{H} \vdash e_1 : \tau_1, \mathcal{X}_1 & \vdash \mathsf{auth}^+(\tau_1) \preccurlyeq pc & w = (\bigsqcap \mathcal{X}_1) \sqcap \mathsf{integ}(\tau_1) \\ pc' = pc \sqcap w & \Gamma,x{:}\tau',y{:}\tau_1;pc';\mathcal{H} \vdash e_2 : \tau_2, \mathcal{X}_2 & \vdash \mathsf{auth}^+(\tau_2) \preccurlyeq pc'\end{array}}{\Gamma,x{:}\tau';pc;\mathcal{H} \vdash \mathsf{let}\ y = e_1\ \mathsf{in}\ e_2 : \tau_2 \sqcap w, \mathcal{X}_1 \sqcap \mathcal{X}_2},$$

where $\tau = \tau_2 \sqcap w$ and $\mathcal{X} = \mathcal{X}_1 \sqcap \mathcal{X}_2$. By the induction hypothesis, we therefore have

$$\Gamma;pc;\mathcal{H} \vdash e_1\{v/x\} : \tau_1, \mathcal{X}_1. \tag{5}$$

Suppose $y = x$. Then $e\{v/x\} = \mathsf{let}\ x = e_1\{v/x\}\ \mathsf{in}\ e_2$, so from (5), the result holds in this case via an application of T-LET:

$$\frac{\begin{array}{ccc}\Gamma;pc;\mathcal{H} \vdash e_1\{v/x\} : \tau_1, \mathcal{X}_1 & \vdash \mathsf{auth}^+(\tau_1) \preccurlyeq pc & w = (\bigsqcap \mathcal{X}_1) \sqcap \mathsf{integ}(\tau_1) \\ pc' = pc \sqcap w & \Gamma,y{:}\tau_1;pc';\mathcal{H} \vdash e_2 : \tau_2, \mathcal{X}_2 & \vdash \mathsf{auth}^+(\tau_2) \preccurlyeq pc'\end{array}}{\Gamma;pc;\mathcal{H} \vdash \mathsf{let}\ y = e_1\{v/x\}\ \mathsf{in}\ e_2 : \tau_2 \sqcap w, \mathcal{X}_1 \sqcap \mathcal{X}_2}.$$

Alternatively, suppose $y \neq x$. We have $\mathsf{FV}(v) = \varnothing$, so $e\{v/x\} = \mathsf{let}\ y = e_1\{v/x\}\ \mathsf{in}\ e_2\{v/x\}$. From the typing of e, we know $\Gamma,x{:}\tau',y{:}\tau_1;pc';\mathcal{H} \vdash e_2 : \tau_2, \mathcal{X}_2$. By the induction hypothesis, we have $\Gamma,y{:}\tau_1;pc';\mathcal{H} \vdash e_2\{v/x\} : \tau_2, \mathcal{X}_2$. From this and (5), the result holds in this case via an application of T-LET:

$$\frac{\begin{array}{ccc}\Gamma;pc;\mathcal{H} \vdash e_1\{v/x\} : \tau_1, \mathcal{X}_1 & \vdash \mathsf{auth}^+(\tau_1) \preccurlyeq pc & w = (\bigsqcap \mathcal{X}_1) \sqcap \mathsf{integ}(\tau_1) \\ pc' = pc \sqcap w & \Gamma,y{:}\tau_1;pc';\mathcal{H} \vdash e_2\{v/x\} : \tau_2, \mathcal{X}_2 & \vdash \mathsf{auth}^+(\tau_2) \preccurlyeq pc'\end{array}}{\Gamma;pc;\mathcal{H} \vdash \mathsf{let}\ y = e_1\{v/x\}\ \mathsf{in}\ e_2\{v/x\} : \tau_2 \sqcap w, \mathcal{X}_1 \sqcap \mathcal{X}_2}.$$

□

Lemma 7 (Effect bound)**.** *The effect of a well-typed expression is bounded from below by its handler environment.*

$$\Gamma;pc;\mathcal{H} \vdash e : \tau, \mathcal{X} \Rightarrow \mathcal{H} \preccurlyeq \mathcal{X}$$

Proof. By induction on the derivation of $\Gamma;pc;\mathcal{H} \vdash e : \tau, \mathcal{X}$. Rule T-TRY relies on the easily proved fact that if $\mathcal{H} \cup \{p\} \preccurlyeq \mathcal{X}_1$, then $\mathcal{H} \preccurlyeq \mathcal{X}_1/p$. □

Lemma 8 (Value typing)**.** *The handler environment is irrelevant for typing non-bottom values. Such a value v can also be typed with any pc that has the authority of the references that appear in v, and can have any effect bounded from below by the handler environment. Formally,*

$$\begin{array}{l}\Gamma;pc;\mathcal{H} \vdash v : \tau, \mathcal{X} \wedge (\forall p.\ v \neq \bot_p \wedge v \neq [\bot_p]) \\ \wedge \vdash \mathsf{auth}^+(\tau) \preccurlyeq pc' \wedge \vdash \mathcal{H}' \preccurlyeq \mathcal{X}' \\ \quad \Rightarrow \Gamma;pc';\mathcal{H}' \vdash v : \tau, \mathcal{X}'\end{array}$$

Proof. By induction on the derivation of $\Gamma;pc;\mathcal{H} \vdash v : \tau, \mathcal{X}$. □

We are now ready to prove type preservation for $[\lambda_{persist}]$.

Lemma 9 (Type preservation)**.** *Let M be a well-formed memory. Let e be an expression with type τ and effect $\mathcal{X}$. Let $\alpha \in \mathcal{L}$. If the configuration $\langle e,M \rangle$ takes an $\rightarrow_\alpha$ transition, then the new expression e' will also have type τ and effect $\mathcal{X}$:*

$$\begin{array}{l}\vdash^{\alpha}_{[wf]} M \wedge \varnothing;pc;\mathcal{H} \vdash e : \tau, \mathcal{X} \\ \wedge \langle e,M \rangle \rightarrow_\alpha \langle e',M' \rangle \\ \quad \Rightarrow \varnothing;pc;\mathcal{H} \vdash e' : \tau, \mathcal{X}.\end{array}$$

Proof. By induction on the derivation of $\varnothing;pc;\mathcal{H} \vdash e : \tau, \mathcal{X}$.

Given $\langle e,M \rangle \rightarrow_\alpha \langle e',M' \rangle$, the proof proceeds by cases according to the evaluation rules. For cases GC, α-CREATE, α-ASSIGN, and α-FORGET, we have $e = e'$, so the result follows trivially.

By Lemma 7, we have $\mathcal{H} \preccurlyeq \mathcal{X}$.

Case CREATE ($\langle \{\overrightarrow{x_i = v_i}\}^S, M\rangle \to_\alpha \left\langle m^S, M[m^S \mapsto \{\overrightarrow{x_i = v_i'}\}]\right\rangle$, where m is fresh, $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$, and $v_i' = v_i \blacktriangleright_\alpha \tau_i$):

We have $\varnothing; pc; \mathcal{H} \vdash \{\overrightarrow{x_i = v_i}\}^S : R_\top, \top$ with $R = \{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)}$ and $\vdash_{wf} S : \mathsf{rectype}$. We need to show $\varnothing; pc; \mathcal{H} \vdash m^S : R_\top, \top$. This is given by trivial application of T-LOC.

Case APPLY ($\langle(\lambda(x{:}\tau_1)[pc_1; \mathcal{H}_1].e_1)\ v, M\rangle \to_\alpha \langle e_1\{v/x\}, M\rangle$):

From the typing of e, we have

$$\varnothing; pc; \mathcal{H} \vdash \lambda(x{:}\tau_1)[pc_1; \mathcal{H}_1].e_1 : (\tau_1 \xrightarrow{pc_1, \mathcal{H}_1} \tau)_\top, \top \tag{6}$$

and

$$\varnothing; pc; \mathcal{H} \vdash v : \tau_1, \top \tag{7}$$

with $\vdash \mathcal{H} \preccurlyeq \mathcal{H}_1$ and $\mathcal{H}_1 = \mathcal{X}$. We need to show $\varnothing; pc; \mathcal{H} \vdash e_1\{v/x\} : \tau, \mathcal{H}_1$. From the derivation of (6), we know

$$x{:}\tau_1; pc_1; \mathcal{H}_1 \vdash e_1 : \tau, \mathcal{H}_1$$

with $\vdash pc_1 \preccurlyeq pc$. Applying Corollary 5 to this, we get

$$x{:}\tau_1; pc; \mathcal{H} \vdash e_1 : \tau, \mathcal{H}_1.$$

The result follows from this and (7) via Lemma 6.

Case SELECT ($\langle m^S.x_c, M\rangle \to_\alpha \langle v_c \blacktriangleright_\alpha p, M\rangle$, where $M(m^S) = \{\overrightarrow{x_i = v_i}\}$ and $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$):

We have $\varnothing; pc; \mathcal{H} \vdash m^S.x_c : \tau_c \sqcap p, p$ and need to show $\varnothing; pc; \mathcal{H} \vdash v_c \blacktriangleright_\alpha p : \tau_c \sqcap p, p$.

From the typing of e, we know $\varnothing; pc; \mathcal{H} \vdash m^S : (\{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top, \top$ and $\vdash a \preccurlyeq pc$ with $\vdash_{wf} S : \mathsf{rectype}$. Therefore, the following holds:

- $\vdash \mathsf{auth}^+(\tau_c) \preccurlyeq a \preccurlyeq pc$ and
- $\vdash \mathsf{auth}^+(\tau_c) \preccurlyeq p$.

So, we know $\vdash \mathsf{auth}^+(\tau_c) \preccurlyeq pc \sqcap p$.

By Lemma 7, we have $\vdash \mathcal{H} \preccurlyeq p$. Since M is well-formed, we also know that $\varnothing; \top; \top \vdash v_c : \tau_c, \top$, so by Lemma 8, we have

$$\varnothing; pc \sqcap p; \mathcal{H} \vdash v_c : \tau_c, p. \tag{8}$$

Suppose $\vdash \alpha \preccurlyeq p$ or v_c is bracketed. Then $v_c \blacktriangleright_\alpha p = v_c$ and the result follows from (8) via Corollary 5 and T-SUBSUME.

Otherwise, $\alpha \not\preccurlyeq p$ and v_c is unbracketed. So $v_c \blacktriangleright_\alpha p = [v_c]$, and the result follows via T-BRACKET:

$$\frac{\varnothing; pc \sqcap p; \mathcal{H} \vdash v_c : \tau_c, p \qquad \alpha \not\preccurlyeq p \qquad \vdash \mathsf{auth}^+(\tau_c) \preccurlyeq pc \sqcap p}{\varnothing; pc; \mathcal{H} \vdash [v_c] : \tau_c \sqcap p, p}.$$

Case DANGLE-SELECT ($\langle m^S.x_c, M\rangle \to_\alpha \langle \bot_p \blacktriangleright_\alpha p, M\rangle$, where $M(m^S) = \bot$ and $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$):

We have $\varnothing; pc; \mathcal{H} \vdash m^S.x_c : \tau_c \sqcap p, p$ and need to show $\varnothing; pc; \mathcal{H} \vdash \bot_p \blacktriangleright_\alpha p : \tau_c \sqcap p, p$.

From the typing of e, we know $\varnothing; pc; \mathcal{H} \vdash m^S : (\{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top, \top$ and $\vdash a \preccurlyeq pc$ with $\vdash_{wf} S : \mathsf{rectype}$. Therefore, the following holds:

- $\vdash \mathsf{auth}^+(\tau_c) \preccurlyeq a \preccurlyeq pc$ and
- $\vdash \mathsf{auth}^+(\tau_c) \preccurlyeq p$.

So, we know $\vdash \mathsf{auth}^+(\tau_c) \preccurlyeq pc \sqcap p$.

By Lemma 7, we know $\vdash \mathcal{H} \preccurlyeq p$, so by T-BOTTOM, we have

$$\varnothing; pc \sqcap p; \mathcal{H} \vdash \bot_p : \tau_c, p. \tag{9}$$

Suppose $\vdash \alpha \preccurlyeq p$. Then $\bot_p \blacktriangleright_\alpha p = \bot_p$ and the result follows from (8) via Corollary 5 and T-SUBSUME.

Otherwise, $\alpha \not\preccurlyeq p$. So $\bot_p \blacktriangleright_\alpha p = [\bot_p]$, and the result follows via T-BRACKET:

$$\frac{\varnothing; pc \sqcap p; \mathcal{H} \vdash \bot_p : \tau_c, p \qquad \alpha \not\preccurlyeq p \qquad \vdash \mathsf{auth}^+(\tau_c) \preccurlyeq pc \sqcap p}{\varnothing; pc; \mathcal{H} \vdash [\bot_p] : \tau_c \sqcap p, p}.$$

Case SOFT-SELECT ($\langle (\mathsf{soft}\, m^S).x_c, M\rangle \to_\alpha \langle v \blacktriangleright_\alpha (a \sqcap p), M\rangle$, where $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$ and $\langle m^S.x_c, M\rangle \xrightarrow{\mathsf{e}} \langle v, M\rangle$):

From the typing of e, we have $\varnothing; pc; \mathcal{H} \vdash (\mathsf{soft}\, m^S).x_c : \tau_c \sqcap p', p$ and $\vdash \mathsf{auth}^+(\tau_c) \preccurlyeq pc$, where $p' = a \sqcap p$. So $\tau = \tau_c \sqcap p'$ and $\mathcal{X} = p$. We need to show $\varnothing; pc; \mathcal{H} \vdash v \blacktriangleright_\alpha p' : \tau_c \sqcap p', p$.

From the typing of e, it follows that $\vdash_{wf} S : \mathsf{rectype}$, and therefore, we know $\vdash \mathsf{auth}^+(\tau_c) \preccurlyeq p'$. Since we also know $\vdash \mathsf{auth}^+(\tau_c) \preccurlyeq pc$, we therefore have $\vdash \mathsf{auth}^+(\tau_c) \preccurlyeq pc \sqcap p'$.

We proceed by cases according to the evaluation rules for $\langle m^S.x_c, M\rangle \xrightarrow{\mathsf{e}} \langle v, M\rangle$.

Sub-case SELECT ($\langle m^S.x_c, M\rangle \xrightarrow{\mathsf{e}} \langle v_c \blacktriangleright_\alpha p, M\rangle$, where $M(m^S) = \{\overrightarrow{x_i = v_i}\}$):

We have $v = v_c \blacktriangleright_\alpha p$.

By Lemma 7, we have $\vdash \mathcal{H} \preccurlyeq p$. Since M is well-formed, we also know that $\varnothing; \top; \top \vdash v_c : \tau_c, \top$, so by Lemma 8, we have

$$\varnothing; pc \sqcap p'; \mathcal{H} \vdash v_c : \tau_c, p. \tag{10}$$

Suppose $\vdash \alpha \preccurlyeq p'$. Then $\vdash \alpha \preccurlyeq p$, so $v \blacktriangleright_\alpha p' = v = v_c \blacktriangleright_\alpha p = v_c$. Similarly, if v_c is bracketed, then $v \blacktriangleright_\alpha p' = v_c$. In these cases, the result follows from (10) via Corollary 5 and T-SUBSUME.

Otherwise, $\alpha \not\preccurlyeq p'$ and v_c is unbracketed. So $v \blacktriangleright_\alpha p' = [v_c]$ and the result follows via T-BRACKET:

$$\frac{\varnothing; pc \sqcap p'; \mathcal{H} \vdash v_c : \tau_c, p \qquad \alpha \not\preccurlyeq p' \qquad \vdash \mathsf{auth}^+(\tau_c) \preccurlyeq pc \sqcap p'}{\varnothing; pc; \mathcal{H} \vdash [v_c] : \tau_c \sqcap p', p}.$$

Sub-case DANGLE-SELECT ($\langle m^S.x_c, M\rangle \xrightarrow{\mathsf{e}} \langle \bot_p \blacktriangleright_\alpha p, M\rangle$):

We have $v = \bot_p \blacktriangleright_\alpha p$.

By Lemma 7, we know $\vdash \mathcal{H} \preccurlyeq p$, and we have $\vdash p' \preccurlyeq p$ by definition, so by T-BOTTOM, we have $\varnothing; pc \sqcap p'; \mathcal{H} \vdash \bot_p : \tau_c \sqcap p', p$.

Suppose $\vdash \alpha \preccurlyeq p'$. Then $\vdash \alpha \preccurlyeq p$, so $v \blacktriangleright_\alpha p' = v = \bot_p \blacktriangleright_\alpha p = \bot_p$, and the result follows via Corollary 5.

Otherwise, $\alpha \not\preccurlyeq p'$, so $v \blacktriangleright_\alpha p' = [\bot_p]$, and the result follows via T-BRACKET:

$$\frac{\varnothing; pc \sqcap p'; \mathcal{H} \vdash \bot_p : \tau_c \sqcap p', p \qquad \alpha \not\preccurlyeq p' \qquad \vdash \mathsf{auth}^+(\tau_c) \preccurlyeq pc \sqcap p'}{\varnothing; pc; \mathcal{H} \vdash [\bot_p] : \tau_c \sqcap p', p}.$$

Case ASSIGN ($\langle m^S.x_c := v, M\rangle \to_\alpha \langle * \blacktriangleright_\alpha p, M[m^S.x_c \mapsto v']\rangle$, where $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$):

From the typing of e, we have $\varnothing; pc; \mathcal{H} \vdash m^S.x_c := v : \mathbf{1}, p$ and need to show $\varnothing; pc; \mathcal{H} \vdash * \blacktriangleright_\alpha p : \mathbf{1}, p$. By Lemma 7, we have $\vdash \mathcal{H} \preccurlyeq p$, so from T-UNIT and T-SUBSUME, we have $\varnothing; pc \sqcap p; \mathcal{H} \vdash * : \mathbf{1}, p$.

If $\vdash \alpha \preccurlyeq p$, then $* \blacktriangleright_\alpha p = *$, and the result follows by Corollary 5.

Otherwise, $\alpha \not\preccurlyeq p$ and $* \blacktriangleright_\alpha p = [*]$, and the result follows via T-BRACKET.

Case DANGLE-ASSIGN ($\langle m^S.x_c := v, M\rangle \to_\alpha \langle \bot_p \blacktriangleright_\alpha p, M\rangle$, where $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$):

From the typing of e, we have $\varnothing; pc; \mathcal{H} \vdash m^S.x_c := v : \mathbf{1}, p$ and we need to show $\varnothing; pc; \mathcal{H} \vdash \bot_p \blacktriangleright_\alpha p : \mathbf{1}, p$. By Lemma 7, we have $\vdash \mathcal{H} \preccurlyeq p$, so by T-BOTTOM, we have $\varnothing; pc \sqcap p; \mathcal{H} \vdash \bot_p : \mathbf{1}, p$.

If $\vdash \alpha \preccurlyeq p$, then $\bot_p \blacktriangleright_\alpha p = \bot_p$, and the result follows by Corollary 5.

Otherwise, $\alpha \not\preccurlyeq p$ and $\bot_p \blacktriangleright_\alpha p = [\bot_p]$, and the result follows via T-BRACKET.

Case SOFT-ASSIGN ($\langle(\mathsf{soft}\ m^S).x_c := v, M\rangle \to_\alpha \langle v' \blacktriangleright_\alpha (a \sqcap p), M'\rangle$, where $\langle m^S.x_c := v, M\rangle \xrightarrow{\mathsf{e}} \langle v', M'\rangle$ and $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$):

From the typing of e, we have $\varnothing; pc; \mathcal{H} \vdash (\mathsf{soft}\ m^S).x_c := v : \mathbf{1}, p$. By Lemma 7, we have $\vdash \mathcal{H} \preccurlyeq p$. Let $p' = a \sqcap p$. We need to show $\varnothing; pc; \mathcal{H} \vdash v' \blacktriangleright_\alpha p' : \mathbf{1}, p$. We proceed by cases according to the evaluation rules for $\langle m^S.x_c := v, M\rangle \xrightarrow{\mathsf{e}} \langle v', M'\rangle$.

Sub-case ASSIGN ($\langle m^S.x_c := v, M\rangle \xrightarrow{\mathsf{e}} \langle * \blacktriangleright_\alpha p, M'\rangle$):

We have $v' = * \blacktriangleright_\alpha p$.

If $\vdash \alpha \preccurlyeq p'$, then $\vdash \alpha \preccurlyeq p$, so $v' \blacktriangleright_\alpha p' = v' = * \blacktriangleright_\alpha p = *$. The result follows from T-UNIT and T-SUBSUME.

Otherwise, $\alpha \not\preccurlyeq p'$, so $v' \blacktriangleright_\alpha p' = [*]$. The result follows from T-UNIT, T-BRACKET, and T-SUBSUME.

Sub-case DANGLE-ASSIGN ($\langle m^S.x_c := v, M\rangle \xrightarrow{\mathsf{e}} \langle \bot_p \blacktriangleright_\alpha p, M\rangle$):

We have $v' = \bot_p \blacktriangleright_\alpha p$.

If $\vdash \alpha \preccurlyeq p'$, then $\vdash \alpha \preccurlyeq p$, so $v' \blacktriangleright_\alpha p' = v' = \bot_p \blacktriangleright_\alpha p = \bot_p$. The result follows from T-BOTTOM.

Otherwise, $\alpha \not\preccurlyeq p'$, so $v' \blacktriangleright_\alpha p' = [\bot_p]$. The result follows from T-BOTTOM and T-BRACKET.

Case EXISTS-TRUE ($\langle \mathsf{exists\ soft}\ m^S\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2, M\rangle \to_\alpha \langle (e_1\{m^S/x\}) \blacktriangleright_\alpha w, M\rangle$, where $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$ and $w = a \sqcap p$):

From the typing of e, we have

$$\frac{\begin{array}{c}\varnothing; pc; \mathcal{H} \vdash \mathsf{soft}\ m^S : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top, \top \qquad \vdash a \preccurlyeq pc \qquad w = a \sqcap p \\ x{:}(\{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top; pc \sqcap w; \mathcal{H} \vdash e_1 : \tau', \mathcal{X}_1 \qquad \varnothing; pc \sqcap w; \mathcal{H} \vdash e_2 : \tau', \mathcal{X}_2 \qquad \vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap w\end{array}}{\varnothing; pc; \mathcal{H} \vdash \mathsf{exists\ soft}\ m^S\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2 : \tau' \sqcap w, \mathcal{X}_1 \sqcap \mathcal{X}_2}$$

where $\tau = \tau' \sqcap w$ and $\mathcal{X} = \mathcal{X}_1 \sqcap \mathcal{X}_2$. We need to show

$$\varnothing; pc; \mathcal{H} \vdash (e_1\{m^S/x\}) \blacktriangleright_\alpha w : \tau, \mathcal{X}.$$

By T-LOC, we have $\varnothing; pc \sqcap w; \mathcal{H} \vdash m^S : (\{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top, \top$. Therefore, by Lemma 6, we know $\varnothing; pc \sqcap w; \mathcal{H} \vdash e_1\{m^S/x\} : \tau', \mathcal{X}_1$. Also, by Lemma 7, we know $\vdash \mathcal{H} \preccurlyeq \mathcal{X}_1 \sqcap \mathcal{X}_2$, so by T-SUBSUME, we know

$$\varnothing; pc \sqcap w; \mathcal{H} \vdash e_1\{m^S/x\} : \tau, \mathcal{X}.$$

Suppose $\vdash \alpha \preccurlyeq w$. Then $(e_1\{m^S/x\}) \blacktriangleright_\alpha w = e_1\{m^S/x\}$. The result therefore follows by Corollary 5.

Otherwise, we have $\alpha \not\preccurlyeq w$, so $(e_1\{m^S/x\}) \blacktriangleright_\alpha w = [e_1\{m^S/x\}]$. The result therefore follows via T-BRACKET:

$$\frac{\varnothing; pc \sqcap w; \mathcal{H} \vdash e_1\{m^S/x\} : \tau' \sqcap w, \mathcal{X} \qquad \alpha \not\preccurlyeq w \qquad \vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap w}{\varnothing; pc; \mathcal{H} \vdash [e_1\{m^S/x\}] : \tau' \sqcap w, \mathcal{X}}$$

Case EXISTS-FALSE ($\langle \mathsf{exists\ soft}\ m^S\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2, M\rangle \to_\alpha \langle e_2 \blacktriangleright_\alpha w, M\rangle$, where $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$ and $w = a \sqcap p$):

From the typing of e, we have

$$\frac{\begin{array}{c}\varnothing; pc; \mathcal{H} \vdash \mathsf{soft}\ m^S : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top, \top \qquad \vdash a \preccurlyeq pc \qquad w = a \sqcap p \\ x{:}(\{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top; pc \sqcap w; \mathcal{H} \vdash e_1 : \tau', \mathcal{X}_1 \qquad \varnothing; pc \sqcap w; \mathcal{H} \vdash e_2 : \tau', \mathcal{X}_2 \qquad \vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap w\end{array}}{\varnothing; pc; \mathcal{H} \vdash \mathsf{exists\ soft}\ m^S\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2 : \tau' \sqcap w, \mathcal{X}_1 \sqcap \mathcal{X}_2}$$

where $\tau = \tau' \sqcap w$ and $\mathcal{X} = \mathcal{X}_1 \sqcap \mathcal{X}_2$. We need to show $\varnothing; pc; \mathcal{H} \vdash e_2 \blacktriangleright_\alpha w : \tau, \mathcal{X}$.

Suppose $\vdash \alpha \preccurlyeq w$. Then $e_2 \blacktriangleright_\alpha w = e_2$. The result follows from Corollary 5 and T-SUBSUME.

Otherwise, $\alpha \not\preccurlyeq w$, so $e_2 \blacktriangleright_\alpha w = [e_2]$. By Lemma 7, we know $\vdash \mathcal{H} \preccurlyeq \mathcal{X}_1 \sqcap \mathcal{X}_2$. The result follows via T-BRACKET and T-SUBSUME:

$$\frac{\dfrac{\varnothing; pc \sqcap w; \mathcal{H} \vdash e_2 : \tau', \mathcal{X}_2 \qquad \alpha \not\preccurlyeq w \qquad \vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap w}{\varnothing; pc; \mathcal{H} \vdash [e_2] : \tau' \sqcap w, \mathcal{X}_2} \qquad \vdash \mathcal{H} \preccurlyeq \mathcal{X}_1 \sqcap \mathcal{X}_2 \qquad \vdash \mathcal{X}_1 \sqcap \mathcal{X}_2 \preccurlyeq \mathcal{X}_2}{\varnothing; pc; \mathcal{H} \vdash [e_2] : \tau' \sqcap w, \mathcal{X}_1 \sqcap \mathcal{X}_2}$$

Case TRY-VAL ($\langle \mathsf{try}\ v\ \mathsf{catch}\ p{:}\ e_1, M\rangle \to_\alpha \langle v, M\rangle$, where $v \neq \bot_{p'}$ and $v \neq [\bot_{p'}]$ for all p'):

From the typing of e, we have

$$\varnothing;pc;\mathcal{H},p \vdash v : \tau, \mathcal{X}_1,$$

for some $\mathcal{X}_1$. Since v is a non-bottom value, the result $\varnothing;pc;\mathcal{H} \vdash v : \tau, \mathcal{X}$ follows via Lemma 8.

Case TRY-CATCH ($\langle \mathsf{try}\ \bot_p\ \mathsf{catch}\ p'{:}\ e_2, M\rangle \to_\alpha \langle e_2, M\rangle$, where $\vdash p' \preccurlyeq p$):

We have $p' \preccurlyeq p$, so from the typing of e, we have

$$\varnothing;pc \sqcap p \sqcap \mathsf{integ}(\tau_1);\mathcal{H} \vdash e_2 : \tau_1, \mathcal{X},$$

where $\tau = \tau_1 \sqcap p$. We need to show $\varnothing;pc;\mathcal{H} \vdash e_2 : \tau, \mathcal{X}$. This follows from Corollary 5 and T-SUBSUME.

Case TRY-ESC ($\langle \mathsf{try}\ \bot_p\ \mathsf{catch}\ p'{:}\ e_2, M\rangle \to_\alpha \langle \bot_p, M\rangle$, where $p' \not\preccurlyeq p$):

We have $p' \not\preccurlyeq p$, so from the typing of e, it follows that $\mathcal{H} \preccurlyeq \mathcal{X} \preccurlyeq p$. We need to show $\varnothing;pc;\mathcal{H} \vdash \bot_p : \tau, \mathcal{X}$. This follows from T-BOTTOM and T-SUBSUME.

Case PARALLEL-RESULT ($\langle v_1 \parallel v_2, M\rangle \to_\alpha \langle *, M\rangle$):

From the typing of e, we have $\tau = \mathbf{1}$ and $\mathcal{X} = \top$. The result $\varnothing;pc;\mathcal{H} \vdash * : \mathbf{1}, \top$ follows trivially from T-UNIT.

Case IF-TRUE ($\langle \mathsf{if\ true\ then}\ e_1\ \mathsf{else}\ e_2), M\rangle \to_\alpha \langle e_1, M\rangle$):

From the typing of e, we have $\varnothing;pc;\mathcal{H} \vdash e_1 : \tau, \mathcal{X}_1$, where $\mathcal{X} \preccurlyeq \mathcal{X}_1$. We need to show $\varnothing;pc;\mathcal{H} \vdash e_1 : \tau, \mathcal{X}$, which follows from Lemma 7 and T-SUBSUME.

Case IF-FALSE ($\langle \mathsf{if\ false\ then}\ e_1\ \mathsf{else}\ e_2, M\rangle \to_\alpha \langle e_2, M\rangle$):

From the typing of e, we have $\varnothing;pc;\mathcal{H} \vdash e_2 : \tau, \mathcal{X}_2$, where $\mathcal{X} \preccurlyeq \mathcal{X}_2$. We need to show $\varnothing;pc;\mathcal{H} \vdash e_2 : \tau, \mathcal{X}$, which follows from Lemma 7 and T-SUBSUME.

Case LET ($\langle \mathsf{let}\ x = v\ \mathsf{in}\ e_1, M\rangle \to_\alpha \langle e_1\{v/x\}, M\rangle$, where $v \notin \{\bot_p, [\bot_p]\}$ for all p):

Since v is a non-bottom value, by Lemma 8, it can be typed with $\top$ effect. From the typing of e, then, we have

$$\varnothing;pc;\mathcal{H} \vdash v : \tau_1, \top$$

and

$$x{:}\tau_1;pc \sqcap \mathsf{integ}(\tau_1);\mathcal{H} \vdash e_1 : \tau, \mathcal{X}.$$

We need to show $\varnothing;pc;\mathcal{H} \vdash e_1\{v/x\} : \tau, \mathcal{X}$, which follows from Corollary 5 and Lemma 6.

Case EVAL-CONTEXT ($\langle E[e_1], M\rangle \to_\alpha \langle E[e_1'], M'\rangle$, where $\langle e_1, M\rangle \xrightarrow{\mathsf{e}} \langle e_1', M'\rangle$):

We need to show $\varnothing;pc;\mathcal{H} \vdash E[e_1'] : \tau, \mathcal{X}$. We proceed by cases according to the structure of $E[e_1]$.

Case $\mathsf{soft}\ e_1$:

From the typing of e, we have $\varnothing;pc;\mathcal{H} \vdash e_1 : R_w, \mathcal{X}$, where $\tau = (\mathsf{soft}\ R)_w$. By the induction hypothesis, we have $\varnothing;pc;\mathcal{H} \vdash e_1' : R_w, \mathcal{X}$, so the result follows via T-SOFT.

Cases $e_1 \parallel e_2$ and $e_2 \parallel e_1$:

We show case $e_1 \parallel e_2$. The other case follows symmetrically. From the typing of e, we have $\varnothing;pc;\top \vdash e_1 : \tau_1, \top$. By the induction hypothesis, we have $\varnothing;pc;\top \vdash e_1' : \tau_1, \top$, so the result follows via T-PARALLEL.

Case $\mathsf{try}\ e_1\ \mathsf{catch}\ p{:}\ e_2$:

From the typing of e, we have $\varnothing;pc;\mathcal{H},p \vdash e_1 : \tau_1, \mathcal{X}_1$, where $\tau = \tau_1 \sqcap w$ with $w = \bigsqcap_{p' \in \mathcal{X}_1}(p \sqcup p')$ and $\mathcal{X} = (\mathcal{X}_1/p) \sqcap \mathcal{X}_2$. By the induction hypothesis, we have $\varnothing;pc;\mathcal{H},p \vdash e_1' : \tau_1, \mathcal{X}_1$, so the result follows via T-TRY.

Case $\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$:

From the typing of e, we have $\varnothing;pc;\mathcal{H} \vdash e_1 : \tau_1, \mathcal{X}_1$, where $\vdash \mathsf{auth}^+(\tau_1) \preccurlyeq pc$ and $\vdash \mathcal{X}_1 \preccurlyeq \mathcal{X}$. By the induction hypothesis, we have $\varnothing;pc;\mathcal{H} \vdash e_1' : \tau_1, \mathcal{X}_1$, so the result follows via T-LET.

Case FAIL-PROP ($\langle F[\bot_p], M\rangle \rightarrow_\alpha \langle \bot_p, M\rangle$):

We have $\varnothing; pc; \mathcal{H} \vdash e : \tau, \mathcal{X}$ and need to show $\varnothing; pc; \mathcal{H} \vdash \bot_p : \tau, \mathcal{X}$. From the typing of e, it follows that $\vdash \mathcal{H} \preccurlyeq \mathcal{X} \preccurlyeq p \neq \top$, so the result follows from T-BOTTOM and T-SUBSUME.

Case BRACKET-SELECT ($\langle [m^S].x_c, M\rangle \rightarrow_\alpha \langle [m^S.x_c], M\rangle$):

Without loss of generality, assume $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$. Then, from the typing of e, we have

$$\dfrac{\dfrac{\begin{array}{c}\varnothing; pc \sqcap w; \mathcal{H} \vdash m^S : (\{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top, \top \\ \alpha \not\preccurlyeq w \qquad \vdash a \preccurlyeq pc \sqcap w\end{array}}{\varnothing; pc; \mathcal{H} \vdash [m^S] : (\{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_w, \top} \qquad \vdash a \preccurlyeq pc \qquad w' = w \sqcap p \qquad \vdash \mathcal{H} \preccurlyeq p}{\varnothing; pc; \mathcal{H} \vdash [m^S].x_c : \tau_c \sqcap w', p},$$

where $\tau = \tau_c \sqcap w'$ and $\mathcal{X} = p$. We need to show $\varnothing; pc; \mathcal{H} \vdash [m^S.x_c] : \tau_c \sqcap w', p$.

By the typing of m^S, we know $\vdash_{wf} S : \mathsf{rectype}$, and so, $\vdash \mathsf{auth}^+(\tau_c) \preccurlyeq a \preccurlyeq pc \sqcap w$. Therefore, the result follows from an application of T-SELECT, followed by T-BRACKET:

$$\dfrac{\dfrac{\begin{array}{c}\varnothing; pc \sqcap w; \mathcal{H} \vdash m^S : (\{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top, \top \\ \vdash a \preccurlyeq pc \sqcap w \qquad \vdash \mathcal{H} \preccurlyeq p\end{array}}{\varnothing; pc \sqcap w; \mathcal{H} \vdash m^S.x_c : \tau_c \sqcap p, p} \qquad \alpha \not\preccurlyeq w \qquad \vdash \mathsf{auth}^+(\tau_c) \preccurlyeq pc \sqcap w}{\varnothing; pc; \mathcal{H} \vdash [m^S.x_c] : \tau_c \sqcap w', p}.$$

Case BRACKET-SOFT-SELECT ($\langle [\mathsf{soft}\ m^S].x_c, M\rangle \rightarrow_\alpha \langle [(\mathsf{soft}\ m^S).x_c], M\rangle$):

Without loss of generality, assume $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$. Then, from the typing of e, we have

$$\dfrac{\dfrac{\begin{array}{c}\varnothing; pc \sqcap w; \mathcal{H} \vdash \mathsf{soft}\ m^S : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top, \top \\ \alpha \not\preccurlyeq w \qquad \vdash a \preccurlyeq pc \sqcap w\end{array}}{\varnothing; pc; \mathcal{H} \vdash [\mathsf{soft}\ m^S] : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_w, \top} \qquad \vdash \mathsf{auth}^+(\tau_c) \preccurlyeq pc \qquad w' = w \sqcap a \sqcap p \qquad \vdash \mathcal{H} \preccurlyeq p}{\varnothing; pc; \mathcal{H} \vdash [\mathsf{soft}\ m^S].x_c : \tau_c \sqcap w', p},$$

where $\tau = \tau_c \sqcap w'$ and $\mathcal{X} = p$. We need to show $\varnothing; pc; \mathcal{H} \vdash [(\mathsf{soft}\ m^S).x_c] : \tau_c \sqcap w', p$.

By the typing of m^S, we know $\vdash_{wf} S : \mathsf{rectype}$, and so, $\vdash \mathsf{auth}^+(\tau_c) \preccurlyeq a \preccurlyeq pc \sqcap w$. Therefore, the result follows from T-SOFT-SELECT and T-BRACKET:

$$\dfrac{\dfrac{\begin{array}{c}\varnothing; pc \sqcap w; \mathcal{H} \vdash \mathsf{soft}\ m^S : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top, \top \\ \vdash \mathsf{auth}^+(\tau_c) \preccurlyeq pc \sqcap w \qquad w' = w \sqcap a \sqcap p \qquad \vdash \mathcal{H} \preccurlyeq p\end{array}}{\varnothing; pc \sqcap w; \mathcal{H} \vdash (\mathsf{soft}\ m^S).x_c : \tau_c \sqcap w', p} \qquad \alpha \not\preccurlyeq w \qquad \vdash \mathsf{auth}^+(\tau_c) \preccurlyeq pc \sqcap w}{\varnothing; pc; \mathcal{H} \vdash [(\mathsf{soft}\ m^S).x_c] : \tau_c \sqcap w', p}.$$

Case BRACKET-ASSIGN ($\langle [m^S].x_c := v, M\rangle \rightarrow_\alpha \langle [m^S.x_c := v], M\rangle$):

Without loss of generality, assume $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$. Then, from the typing of e, we have

$$\dfrac{\begin{array}{c}\dfrac{\begin{array}{c}\varnothing; pc \sqcap w; \mathcal{H} \vdash m^S : (\{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top, \top \\ \alpha \not\preccurlyeq w \qquad \vdash a \preccurlyeq pc \sqcap w\end{array}}{\varnothing; pc; \mathcal{H} \vdash [m^S] : (\{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_w, \top} \qquad \vdash a \preccurlyeq pc \qquad \varnothing; pc; \mathcal{H} \vdash v : \tau', \top \\ \vdash \tau' \sqcap pc \sqcap w \leq \tau_c \qquad \vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap w \qquad \vdash \mathcal{H} \preccurlyeq p\end{array}}{\varnothing; pc; \mathcal{H} \vdash [m^S].x_c := v : \mathbf{1}, p},$$

where $\tau = \mathbf{1}$ and $\mathcal{X} = p$. We need to show $\varnothing; pc; \mathcal{H} \vdash [m^S.x_c := v] : \mathbf{1}, p$.

From $\vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap w$ and Lemma 8, we know $\varnothing; pc \sqcap w; \mathcal{H} \vdash v : \tau', \top$. The result then follows from an application of T-ASSIGN followed by T-BRACKET:

$$\frac{\dfrac{\begin{array}{cc} \varnothing; pc \sqcap w; \mathcal{H} \vdash m^S : (\{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top, \top & \vdash a \preccurlyeq pc \\ \varnothing; pc \sqcap w; \mathcal{H} \vdash v : \tau', \top & \vdash \tau' \sqcap pc \sqcap w \leq \tau_c \\ \vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap w & \vdash \mathcal{H} \preccurlyeq p \end{array}}{\varnothing; pc \sqcap w; \mathcal{H} \vdash m^S.x_c := v : \mathbf{1}, p} \qquad \alpha \not\preccurlyeq w}{\varnothing; pc; \mathcal{H} \vdash [m^S.x_c := v] : \mathbf{1}, p}\ .$$

Case BRACKET-SOFT-ASSIGN
($\langle [\mathsf{soft}\ m^S].x_c := v, M\rangle \rightarrow_\alpha \langle [(\mathsf{soft}\ m^S).x_c := v], M\rangle$):

Without loss of generality, assume $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$. Then, from the typing of e, we have

$$\frac{\begin{array}{cc} \dfrac{\begin{array}{c} \varnothing; pc \sqcap w; \mathcal{H} \vdash \mathsf{soft}\ m^S : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top, \top \\ \alpha \not\preccurlyeq w \qquad \vdash a \preccurlyeq pc \sqcap w \end{array}}{\varnothing; pc; \mathcal{H} \vdash [\mathsf{soft}\ m^S] : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_w, \top} & \varnothing; pc; \mathcal{H} \vdash v : \tau', \top \\ \vdash \tau' \sqcap pc \sqcap w \leq \tau_c \qquad \vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap w & \vdash \mathcal{H} \preccurlyeq p \end{array}}{\varnothing; pc; \mathcal{H} \vdash [\mathsf{soft}\ m^S].x_c := v : *, p}\ ,$$

where $\tau = \mathbf{1}$ and $\mathcal{X} = p$. We need to show $\varnothing; pc; \mathcal{H} \vdash [(\mathsf{soft}\ m^S).x_c := v] : \mathbf{1}, p$.

From $\vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap w$ and Lemma 8, we know $\varnothing; pc \sqcap w; \mathcal{H} \vdash v : \tau', \top$. The result then follows from T-SOFT-ASSIGN and T-BRACKET:

$$\frac{\dfrac{\begin{array}{c} \varnothing; pc \sqcap w; \mathcal{H} \vdash \mathsf{soft}\ m^S : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top, \top \\ \varnothing; pc \sqcap w; \mathcal{H} \vdash v : \tau, \top \qquad \vdash \tau' \sqcap pc \sqcap w \leq \tau_c \qquad \vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap w \qquad \vdash \mathcal{H} \preccurlyeq p \end{array}}{\varnothing; pc \sqcap w; \mathcal{H} \vdash (\mathsf{soft}\ m^S).x_c := v : \mathbf{1}, p} \qquad \alpha \not\preccurlyeq w}{\varnothing; pc; \mathcal{H} \vdash [(\mathsf{soft}\ m^S).x_c := v] : \mathbf{1}, p}\ .$$

Case BRACKET-SOFT ($\langle \mathsf{soft}\ [m^S], M\rangle \rightarrow_\alpha \langle [\mathsf{soft}\ m^S], M\rangle$):

Without loss of generality, assume $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$. Then, from the typing of e, we have

$$\frac{\dfrac{\begin{array}{c} \varnothing; pc \sqcap w; \mathcal{H} \vdash m^S : (\{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top, \top \\ \alpha \not\preccurlyeq w \qquad \vdash a \preccurlyeq pc \sqcap w \end{array}}{\varnothing; pc; \mathcal{H} \vdash [m^S] : (\{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_w, \top}}{\varnothing; pc; \mathcal{H} \vdash \mathsf{soft}\ [m^S] : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_w, \top}\ ,$$

where $\tau = (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_w$ and $\mathcal{X} = \top$. We need to show $\varnothing; pc; \mathcal{H} \vdash [\mathsf{soft}\ m^S] : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_w, \top$. This follows via an application of T-SOFT followed by T-BRACKET:

$$\frac{\dfrac{\varnothing; pc \sqcap w; \mathcal{H} \vdash m^S : (\{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top, \top}{\varnothing; pc \sqcap w; \mathcal{H} \vdash \mathsf{soft}\ m^S : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top, \top} \qquad \alpha \not\preccurlyeq w}{\varnothing; pc; \mathcal{H} \vdash [\mathsf{soft}\ m^S] : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_w, \top}\ .$$

Case BRACKET-EXISTS
($\langle \mathsf{exists}\ [v]\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2, M\rangle \rightarrow_\alpha \langle [\mathsf{exists}\ v\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2], M\rangle$):

From the typing of e, we have

$$\frac{\begin{array}{c} \dfrac{\begin{array}{c} \varnothing; pc \sqcap \ell; \mathcal{H} \vdash v : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_r)_w, \top \\ \alpha \not\preccurlyeq \ell \qquad \vdash \mathsf{auth}^+(r) \preccurlyeq pc \sqcap \ell \end{array}}{\varnothing; pc; \mathcal{H} \vdash [v] : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_r)_{w \sqcap \ell}, \top} \qquad \vdash \mathsf{auth}^+(r) \preccurlyeq pc \sqcap w \sqcap \ell \qquad w' = \mathsf{auth}^-(r) \sqcap \mathsf{persist}(r) \sqcap w \sqcap \ell \\ x : (\{\overrightarrow{x_i : \tau_i}\}_r)_{w \sqcap \ell}; pc \sqcap w'; \mathcal{H} \vdash e_1 : \tau', \mathcal{X}_1 \qquad \varnothing; pc \sqcap w'; \mathcal{H} \vdash e_2 : \tau', \mathcal{X}_2 \qquad \vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap w' \end{array}}{\varnothing; pc; \mathcal{H} \vdash \mathsf{exists}\ [v]\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2 : \tau' \sqcap w', \mathcal{X}_1 \sqcap \mathcal{X}_2}$$

where $\tau = \tau' \sqcap w'$ and $\mathcal{X} = \mathcal{X}_1 \sqcap \mathcal{X}_2$. We need to show $\varnothing;pc;\mathcal{H} \vdash [\mathsf{exists}\ v\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2] : \tau' \sqcap w', \mathcal{X}_1 \sqcap \mathcal{X}_2$. To do this, we need to know that $x : (\{\overrightarrow{x_i : \tau_i}\}_r)_w;pc \sqcap w';\mathcal{H} \vdash e_1 : \tau', \mathcal{X}_1$, which can be demonstrated by an easy induction on the derivation of $x : (\{\overrightarrow{x_i : \tau_i}\}_r)_{w \sqcap \ell};pc \sqcap w';\mathcal{H} \vdash e_1 : \tau', \mathcal{X}_1$. The result therefore follows via an application of T-EXISTS and T-BRACKET.

$$\dfrac{\dfrac{\begin{array}{c}\varnothing;pc \sqcap \ell;\mathcal{H} \vdash v : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_r)_w, \top \qquad \vdash \mathsf{auth}^+(r) \preccurlyeq pc \sqcap w \sqcap \ell \qquad w'' = \mathsf{auth}^-(r) \sqcap \mathsf{persist}(r) \sqcap w \\ x : (\{\overrightarrow{x_i : \tau_i}\}_r)_w;pc \sqcap \ell \sqcap w'';\mathcal{H} \vdash e_1 : \tau', \mathcal{X}_1 \qquad \varnothing;pc \sqcap \ell \sqcap w'';\mathcal{H} \vdash e_2 : \tau', \mathcal{X}_2 \qquad \vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap \ell \sqcap w''\end{array}}{\varnothing;pc \sqcap \ell;\mathcal{H} \vdash \mathsf{exists}\ v\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2 : \tau' \sqcap w'', \mathcal{X}_1 \sqcap \mathcal{X}_2} \\ \alpha \not\preccurlyeq \ell \qquad \vdash \mathsf{auth}^+(\tau' \sqcap w'') \preccurlyeq pc \sqcap \ell \sqcap w'' \preccurlyeq pc \sqcap \ell}{\varnothing;pc;\mathcal{H} \vdash [\mathsf{exists}\ v\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2] : \tau' \sqcap w', \mathcal{X}_1 \sqcap \mathcal{X}_2}$$

(Note that $w' = \ell \sqcap w''$.)

Case BRACKET-APPLY
$(\langle [\lambda(x{:}\tau_1)[pc_1;\mathcal{H}_1].e_1]\ v_2, M\rangle \to_\alpha \langle [(\lambda(x{:}\tau_1)[pc_1;\mathcal{H}_1].e_1)\ v_2], M\rangle)$:

From the typing of e, we have

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}x{:}\tau_1;pc_1;\mathcal{H}_1 \vdash e_1 : \tau', \mathcal{H}_1 \\ \vdash_{wf} (\tau_1 \xrightarrow{pc_1,\mathcal{H}_1} \tau')_\top : \mathsf{type} \qquad \vdash pc_1 \preccurlyeq pc \sqcap w\end{array}}{\varnothing;pc \sqcap w;\mathcal{H} \vdash \lambda(x{:}\tau_1)[pc_1;\mathcal{H}_1].e_1 : (\tau_1 \xrightarrow{pc_1,\mathcal{H}_1} \tau')_\top, \top} \\ \alpha \not\preccurlyeq w \qquad \vdash pc_1 \preccurlyeq pc \sqcap w}{\begin{array}{c}\varnothing;pc;\mathcal{H} \vdash [\lambda(x{:}\tau_1)[pc_1;\mathcal{H}_1].e_1] : (\tau_1 \xrightarrow{pc_1,\mathcal{H}_1} \tau')_w, \top \\ \varnothing;pc;\mathcal{H} \vdash v_2 : \tau_1, \top \qquad \vdash pc_1 \preccurlyeq pc \sqcap w \qquad \vdash \mathcal{H} \preccurlyeq \mathcal{H}_1\end{array}}}{\varnothing;pc;\mathcal{H} \vdash [\lambda(x{:}\tau_1)[pc_1;\mathcal{H}_1].e_1]\ v_2 : \tau' \sqcap w, \mathcal{H}_1} \quad ,$$

where $\tau = \tau' \sqcap w$ and $\mathcal{X} = \mathcal{H}_1$. We need to show $\varnothing;pc;\mathcal{H} \vdash [(\lambda(x{:}\tau_1)[pc_1;\mathcal{H}_1].e_1)\ v_2] : \tau' \sqcap w, \mathcal{H}_1$.

By WT2, from $\vdash_{wf} (\tau_1 \xrightarrow{pc_1,\mathcal{H}_1} \tau')_\top : \mathsf{type}$, we know $\vdash \mathsf{auth}^+(\tau_1) \sqcup \mathsf{auth}^+(\tau') \preccurlyeq pc_1$. Since we also know from the above derivation that $\vdash pc_1 \preccurlyeq pc \sqcap w$, it therefore follows that

$$\vdash \mathsf{auth}^+(\tau_1) \preccurlyeq pc \sqcap w \tag{11}$$

and $\vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap w$. From the above derivation, we also have $\varnothing;pc;\mathcal{H} \vdash v_2 : \tau_1, \top$. Applying Lemma 8 to this and (11), we have $\varnothing;pc \sqcap w;\mathcal{H} \vdash v_2 : \tau_1, \top$. The result then follows from an application of T-APP followed by T-BRACKET:

$$\dfrac{\dfrac{\begin{array}{c}\varnothing;pc \sqcap w;\mathcal{H} \vdash \lambda(x{:}\tau_1)[pc_1;\mathcal{H}_1].e_1 : (\tau_1 \xrightarrow{pc_1,\mathcal{H}_1} \tau')_\top, \top \\ \varnothing;pc \sqcap w;\mathcal{H} \vdash v_2 : \tau_1, \top \qquad \vdash pc_1 \preccurlyeq pc \sqcap w \qquad \vdash \mathcal{H} \preccurlyeq \mathcal{H}_1\end{array}}{\varnothing;pc \sqcap w;\mathcal{H} \vdash (\lambda(x{:}\tau_1)[pc_1;\mathcal{H}_1].e_1)\ v_2 : \tau', \mathcal{H}_1} \qquad \alpha \not\preccurlyeq w \qquad \vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap w}{\varnothing;pc;\mathcal{H} \vdash [(\lambda(x{:}\tau_1)[pc_1;\mathcal{H}_1].e_1)\ v_2] : \tau' \sqcap w, \mathcal{H}_1} \quad .$$

Case BRACKET-TRY $(\langle \mathsf{try}\ [v]\ \mathsf{catch}\ p{:}\ e_2, M\rangle \to_\alpha \langle [\mathsf{try}\ v\ \mathsf{catch}\ p{:}\ e_2], M\rangle)$:

From the typing of e, we have

$$\dfrac{\dfrac{\begin{array}{c}\varnothing;pc \sqcap \ell;\mathcal{H},p \vdash v : \tau', \mathcal{X}_1 \\ \alpha \not\preccurlyeq \ell \qquad \vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap \ell\end{array}}{\varnothing;pc;\mathcal{H},p \vdash [v] : \tau' \sqcap \ell, \mathcal{X}_1} \qquad w = \prod_{p' \in \mathcal{X}_1} (p \sqcup p') \\ \varnothing;pc \sqcap w \sqcap \mathsf{integ}(\tau' \sqcap \ell);\mathcal{H} \vdash e_2 : \tau' \sqcap \ell, \mathcal{X}_2 \qquad \vdash \mathsf{auth}^+(\tau' \sqcap \ell) \preccurlyeq pc}{\varnothing;pc;\mathcal{H} \vdash \mathsf{try}\ [v]\ \mathsf{catch}\ p{:}\ e_2 : \tau' \sqcap \ell \sqcap w, (\mathcal{X}_1/p) \sqcap \mathcal{X}_2} \quad ,$$

where $\tau = \tau' \sqcap \ell \sqcap w$ and $\mathcal{X} = (\mathcal{X}_1/p) \sqcap \mathcal{X}_2$.

We need to show

$$\varnothing;pc;\mathcal{H} \vdash [\mathsf{try}\ v\ \mathsf{catch}\ p\colon e_2] : \tau' \sqcap \ell \sqcap w, (\mathcal{X}_1/p) \sqcap \mathcal{X}_2.$$

Note that $\mathsf{integ}(\tau' \sqcap \ell) = \mathsf{integ}(\tau') \sqcap \ell$ and $\mathsf{auth}^+(\tau') = \mathsf{auth}^+(\tau' \sqcap \ell) = \mathsf{auth}^+(\tau' \sqcap \ell \sqcap w)$. The result then follows from S3, T-SUBSUME, T-TRY, and T-BRACKET:

$$\dfrac{\begin{array}{c}\dfrac{\begin{array}{c}\dfrac{\varnothing;pc\sqcap\ell;\mathcal{H},p \vdash v:\tau',\mathcal{X}_1 \qquad \dfrac{\vdash \mathsf{integ}(\tau')\sqcap\ell \preccurlyeq \mathsf{integ}(\tau')}{\vdash \tau' \le \tau'\sqcap\ell}}{\varnothing;pc\sqcap\ell;\mathcal{H},p \vdash v:\tau'\sqcap\ell,\mathcal{X}_1} \qquad w = \displaystyle\bigsqcap_{p'\in\mathcal{X}_1}(p\sqcup p') \\ \varnothing;pc\sqcap\ell\sqcap w\sqcap\mathsf{integ}(\tau'\sqcap\ell);\mathcal{H}\vdash e_2:\tau'\sqcap\ell,\mathcal{X}_2 \qquad \vdash \mathsf{auth}^+(\tau'\sqcap\ell)\preccurlyeq pc\sqcap\ell\end{array}}{\varnothing;pc\sqcap\ell;\mathcal{H}\vdash \mathsf{try}\ v\ \mathsf{catch}\ p\colon e_2 : \tau'\sqcap\ell\sqcap w,(\mathcal{X}_1/p)\sqcap\mathcal{X}_2} \\ \alpha \not\preccurlyeq \ell \qquad \vdash \mathsf{auth}^+(\tau'\sqcap\ell\sqcap w)\preccurlyeq pc\sqcap\ell\end{array}}{\varnothing;pc;\mathcal{H}\vdash [\mathsf{try}\ v\ \mathsf{catch}\ p\colon e_2]:\tau'\sqcap\ell\sqcap w,(\mathcal{X}_1/p)\sqcap\mathcal{X}_2}\ .$$

Case BRACKET-IF ($\langle \mathsf{if}\ [v]\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2, M\rangle \to_\alpha \langle [\mathsf{if}\ v\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2], M\rangle$):

From the typing of e, we have

$$\dfrac{\dfrac{\varnothing;pc\sqcap\ell;\mathcal{H}\vdash v:\mathsf{bool}_w,\top \qquad \alpha\not\preccurlyeq\ell}{\varnothing;pc;\mathcal{H}\vdash[v]:\mathsf{bool}_{w\sqcap\ell},\top} \qquad \varnothing;pc\sqcap w\sqcap\ell;\mathcal{H}\vdash e_i:\tau',\mathcal{X}_i\ ^{(\forall i)} \qquad \vdash\mathsf{auth}^+(\tau')\preccurlyeq pc\sqcap w\sqcap\ell}{\varnothing;pc;\mathcal{H}\vdash\mathsf{if}\ [v]\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2:\tau'\sqcap w\sqcap\ell,\mathcal{X}_1\sqcap\mathcal{X}_2}\ ,$$

where $\tau = \tau' \sqcap w \sqcap \ell$ and $\mathcal{X} = \mathcal{X}_1 \sqcap \mathcal{X}_2$. We need to show $\varnothing;pc;\mathcal{H} \vdash [\mathsf{if}\ v\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2] : \tau'\sqcap w\sqcap\ell, \mathcal{X}_1\sqcap\mathcal{X}_2$.

Since $\mathsf{auth}^+(\tau'\sqcap w) = \mathsf{auth}^+(\tau')$ and $\vdash pc\sqcap w\sqcap\ell \preccurlyeq pc\sqcap\ell$, we therefore have $\vdash \mathsf{auth}^+(\tau'\sqcap w)\preccurlyeq pc\sqcap\ell$. The result then follows from T-IF and T-BRACKET:

$$\dfrac{\dfrac{\begin{array}{c}\varnothing;pc\sqcap\ell;\mathcal{H}\vdash v:\mathsf{bool}_w,\top \\ \varnothing;pc\sqcap\ell\sqcap w;\mathcal{H}\vdash e_i:\tau',\mathcal{X}_i\ ^{(\forall i)} \qquad \vdash\mathsf{auth}^+(\tau')\preccurlyeq pc\sqcap\ell\sqcap w\end{array}}{\varnothing;pc\sqcap\ell;\mathcal{H}\vdash\mathsf{if}\ v\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2:\tau'\sqcap w,\mathcal{X}_1\sqcap\mathcal{X}_2} \qquad \alpha\not\preccurlyeq\ell \qquad \vdash\mathsf{auth}^+(\tau'\sqcap w)\preccurlyeq pc\sqcap\ell}{\varnothing;pc;\mathcal{H}\vdash[\mathsf{if}\ v\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2]:\tau'\sqcap w\sqcap\ell,\mathcal{X}_1\sqcap\mathcal{X}_2}\ .$$

Case BRACKET-LET ($\langle \mathsf{let}\ x = [v]\ \mathsf{in}\ e_1, M\rangle \to_\alpha \langle [e_1\{[v]/x\}], M\rangle$, where $v \ne \bot_p$ for all p):

By Lemma 8, we know $[v]$ can be typed with $\top$ effect. Therefore, from the typing of e, we have

$$\dfrac{\begin{array}{c}\dfrac{\begin{array}{c}\varnothing;pc\sqcap\ell;\mathcal{H}\vdash v:\tau'',\top \\ \alpha\not\preccurlyeq\ell \qquad \vdash\mathsf{auth}^+(\tau'')\preccurlyeq pc\sqcap\ell\end{array}}{\varnothing;pc;\mathcal{H}\vdash[v]:\tau''\sqcap\ell,\top} \qquad \vdash\mathsf{auth}^+(\tau''\sqcap\ell)\preccurlyeq pc \\ w=\mathsf{integ}(\tau''\sqcap\ell) \qquad x{:}\tau''\sqcap\ell;pc\sqcap w;\mathcal{H}\vdash e_1:\tau',\mathcal{X} \qquad \vdash\mathsf{auth}^+(\tau')\preccurlyeq pc\sqcap w\end{array}}{\varnothing;pc;\mathcal{H}\vdash\mathsf{let}\ x=[v]\ \mathsf{in}\ e_1:\tau'\sqcap w,\mathcal{X}}\ ,$$

where $\tau = \tau' \sqcap w$. We need to show $\varnothing;pc;\mathcal{H} \vdash [e_1\{[v]/x\}] : \tau'\sqcap w, \mathcal{X}$.

From the derivation above, we know $\varnothing;pc;\mathcal{H}\vdash[v]:\tau''\sqcap\ell,\top$. Since $\mathsf{auth}^+(\tau''\sqcap\ell) = \mathsf{auth}^+(\tau'')$, from $\vdash\mathsf{auth}^+(\tau'')\preccurlyeq pc\sqcap\ell$ above, we have $\vdash\mathsf{auth}^+(\tau''\sqcap\ell)\preccurlyeq pc\sqcap\ell$. Therefore, by Lemma 8, we know

$$\varnothing;pc\sqcap\ell;\mathcal{H}\vdash[v]:\tau''\sqcap\ell,\top. \tag{12}$$

From the derivation above, we also know $x{:}\tau''\sqcap\ell;pc\sqcap w;\mathcal{H}\vdash e_1:\tau',\mathcal{X}$. By definition, $\vdash w\preccurlyeq\ell$, so by Corollary 5, we know $x{:}\tau''\sqcap\ell;pc\sqcap\ell;\mathcal{H}\vdash e_1:\tau',\mathcal{X}$. Applying Lemma 6 to this and (12), we have

$$\varnothing;pc\sqcap\ell;\mathcal{H}\vdash e_1\{[v]/x\}:\tau'\sqcap w,\mathcal{X}.$$

Since $\mathsf{auth}^+(\tau' \sqcap w) = \mathsf{auth}^+(\tau')$ and $\vdash w \preccurlyeq \ell$, from $\vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap w$ above, we have

$$\vdash \mathsf{auth}^+(\tau' \sqcap w) \preccurlyeq pc \sqcap \ell.$$

Finally, by Lemma 7, we have $\vdash \mathcal{H} \preccurlyeq \mathcal{X}$.

The result, then, follows from T-SUBSUME and T-BRACKET:

$$\frac{\dfrac{\begin{array}{c}\varnothing;pc \sqcap \ell;\mathcal{H} \vdash e_1\{[v]/x\} : \tau' \sqcap w, \mathcal{X} \\ \vdash \mathcal{H} \preccurlyeq \mathcal{X}\end{array}}{\varnothing;pc \sqcap \ell;\mathcal{H} \vdash e_1\{[v]/x\} : \tau' \sqcap w, \mathcal{X}} \qquad \alpha \not\preccurlyeq \ell \qquad \vdash \mathsf{auth}^+(\tau' \sqcap w) \preccurlyeq pc \sqcap \ell}{\varnothing;pc;\mathcal{H} \vdash [e_1\{[v]/x\}] : \tau' \sqcap w \sqcap \ell, \mathcal{X}}\ .$$

Case DOUBLE-BRACKET ($\langle [[v]], M\rangle \to_\alpha \langle [v], M\rangle$):

From the typing of e, we have

$$\frac{\varnothing;pc \sqcap \ell;\mathcal{H} \vdash [v] : \tau', \mathcal{X} \qquad \alpha \not\preccurlyeq \ell \qquad \vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap \ell}{\varnothing;pc;\mathcal{H} \vdash [[v]] : \tau' \sqcap \ell, \mathcal{X}}\ ,$$

where $\tau = \tau' \sqcap \ell$. We need to show $\varnothing;pc;\mathcal{H} \vdash [v] : \tau' \sqcap \ell, \mathcal{X}$. By Corollary 5, we know $\varnothing;pc;\mathcal{H} \vdash [v] : \tau', \mathcal{X}$. The result then follows via S3 and T-SUBSUME:

$$\frac{\varnothing;pc;\mathcal{H} \vdash [v] : \tau', \mathcal{X} \qquad \dfrac{\vdash \mathsf{integ}(\tau') \sqcap \ell \preccurlyeq \mathsf{integ}(\tau')}{\vdash \tau' \leq \tau' \sqcap \ell}}{\varnothing;pc;\mathcal{H} \vdash [v] : \tau' \sqcap \ell, \mathcal{X}}\ .$$

Case BRACKET-CONTEXT ($\langle [e_1], M\rangle \to_\alpha \langle [e_1'], M'\rangle$, where $\langle e_1, M\rangle \to_\alpha \langle e_1', M'\rangle$):

From the typing of e, we have

$$\frac{\varnothing;pc \sqcap \ell;\mathcal{H} \vdash e_1 : \tau', \mathcal{X} \qquad \alpha \not\preccurlyeq \ell \qquad \vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap \ell}{\varnothing;pc;\mathcal{H} \vdash [e_1] : \tau' \sqcap \ell, \mathcal{X}}\ ,$$

where $\tau = \tau' \sqcap \ell$. We need to show $\varnothing;pc;\mathcal{H} \vdash [e_1'] : \tau, \mathcal{X}$.

By the induction hypothesis, we have $\varnothing;pc \sqcap \ell;\mathcal{H} \vdash e_1' : \tau', \mathcal{X}$. The result follows by T-BRACKET:

$$\frac{\varnothing;pc \sqcap \ell;\mathcal{H} \vdash e_1' : \tau', \mathcal{X} \qquad \alpha \not\preccurlyeq \ell \qquad \vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap \ell}{\varnothing;pc;\mathcal{H} \vdash [e_1'] : \tau' \sqcap \ell, \mathcal{X}}\ .$$

Case BRACKET-FAIL ($\langle F[[\bot_p]], M\rangle \to_\alpha \langle [\bot_p], M\rangle$):

We need to show $\varnothing;pc;\mathcal{H} \vdash [\bot_p] : \tau, \mathcal{X}$. We proceed by cases according to the structure of $F[[\bot_p]]$.

Case $\mathsf{soft}\ [\bot_p]$:

From the typing of e, we have

$$\frac{\dfrac{\dfrac{p \neq \top \qquad \vdash \mathcal{H} \preccurlyeq p}{\varnothing;pc \sqcap w;\mathcal{H} \vdash \bot_p : R_\top, p} \qquad \alpha \not\preccurlyeq w \qquad \vdash \mathsf{auth}^+(R_\top) \preccurlyeq pc \sqcap w}{\varnothing;pc;\mathcal{H} \vdash [\bot_p] : R_w, p}}{\varnothing;pc;\mathcal{H} \vdash \mathsf{soft}\ [\bot_p] : (\mathsf{soft}\ R)_w, p}\ ,$$

where $\tau = (\mathsf{soft}\ R)_w$ and $\mathcal{X} = p$. The result follows via T-BOTTOM and T-BRACKET:

$$\frac{\dfrac{p \neq \top \qquad \vdash \mathcal{H} \preccurlyeq p}{\varnothing;pc \sqcap w;\mathcal{H} \vdash \bot_p : (\mathsf{soft}\ R)_\top, p} \qquad \alpha \not\preccurlyeq w \qquad \vdash \mathsf{auth}^+((\mathsf{soft}\ R)_\top) \preccurlyeq pc \sqcap w}{\varnothing;pc;\mathcal{H} \vdash [\bot_p] : (\mathsf{soft}\ R)_w, p}\ .$$

Case let $x = [\bot_p]$ in e_2:

From the typing of e, we have

$$\dfrac{\begin{array}{c}\dfrac{\dfrac{p \neq \top \qquad \vdash \mathcal{H} \preccurlyeq p}{\varnothing; pc \sqcap \ell; \mathcal{H} \vdash \bot_p : \tau_1, p} \qquad \alpha \not\preccurlyeq \ell \qquad \vdash \mathsf{auth}^+(\tau_1) \preccurlyeq pc \sqcap \ell}{\varnothing; pc; \mathcal{H} \vdash [\bot_p] : \tau_1 \sqcap \ell, p} \\ \vdash \mathsf{auth}^+(\tau_1 \sqcap \ell) \preccurlyeq pc \qquad w = p \sqcap \mathsf{integ}(\tau_1 \sqcap \ell) \\ x{:}\tau_1 \sqcap \ell; pc \sqcap w; \mathcal{H} \vdash e_2 : \tau_2, \mathcal{X}_2 \qquad \vdash \mathsf{auth}^+(\tau_2) \preccurlyeq pc \sqcap w\end{array}}{\varnothing; pc; \mathcal{H} \vdash \mathsf{let}\ x = [\bot_p]\ \mathsf{in}\ e_2 : \tau_2 \sqcap w, \mathcal{X}_2 \sqcap p} \quad ,$$

where $\tau = \tau_2 \sqcap w$ and $\mathcal{X} = \mathcal{X}_2 \sqcap p$. By construction, $\vdash w \preccurlyeq \ell$, so we know $\alpha \not\preccurlyeq w$. By Lemma 7, we also know $\vdash \mathcal{H} \preccurlyeq \mathcal{X}_2 \sqcap p$. The result follows via T-BOTTOM, T-BRACKET and T-SUBSUME:

$$\dfrac{\dfrac{\begin{array}{c}\dfrac{p \neq \top \qquad \vdash \mathcal{H} \preccurlyeq p}{\varnothing; pc \sqcap w; \mathcal{H} \vdash \bot_p : \tau_2, p} \\ \alpha \not\preccurlyeq w \qquad \vdash \mathsf{auth}^+(\tau_2) \preccurlyeq pc \sqcap w\end{array}}{\varnothing; pc; \mathcal{H} \vdash [\bot_p] : \tau_2 \sqcap w, p} \qquad \vdash \mathcal{H} \preccurlyeq \mathcal{X}_2 \sqcap p \qquad \vdash \mathcal{X}_2 \sqcap p \preccurlyeq \mathcal{X}_2}{\varnothing; pc; \mathcal{H} \vdash [\bot_p] : \tau_2 \sqcap w, \mathcal{X}_2 \sqcap p} \quad .$$

□

We now show that well-formedness is also preserved during execution.

Lemma 10 (Well-formedness preservation)**.** *If $\langle e, M\rangle$ is a well-formed configuration wherein e is well-typed, and $\langle e, M\rangle$ takes an $\rightarrow_\alpha$ transition, then the new configuration $\langle e', M'\rangle$ will also be well-formed:*

$$\vdash^{\alpha}_{[wf]} \langle e, M\rangle \wedge \varnothing; pc; \mathcal{H} \vdash e : \tau, \mathcal{X} \wedge \langle e, M\rangle \rightarrow_\alpha \langle e', M'\rangle \Rightarrow \vdash^{\alpha}_{[wf]} \langle e', M'\rangle .$$

Proof. First, note that $\mathsf{dom}(M) \subseteq \mathsf{dom}(M')$. This can be shown by an easy induction on the derivation of $\langle e, M\rangle \rightarrow_\alpha \langle e', M'\rangle$.

We prove the lemma by induction on the derivation of $\langle e, M\rangle \rightarrow_\alpha \langle e', M'\rangle$. Given $\langle e, M\rangle \rightarrow_\alpha \langle e', M'\rangle$, the proof proceeds by cases according to the evaluation rules.

Rules DANGLE-SELECT, DANGLE-ASSIGN, PARALLEL-RESULT, TRY-ESC, FAIL-PROP, and BRACKET-FAIL, are trivial base cases in which $\mathsf{locs}(e') = \varnothing$ and $M' = M$. Rules SELECT, and SOFT-SELECT follow from the definition of $\vdash^{\alpha}_{[wf]} M$.

Rules APPLY, EXISTS-TRUE, EXISTS-FALSE, TRY-VAL, TRY-CATCH, IF-TRUE, IF-FALSE, LET, BRACKET-SELECT, BRACKET-SOFT-SELECT, BRACKET-ASSIGN, BRACKET-SOFT-ASSIGN, BRACKET-SOFT, BRACKET-EXISTS, BRACKET-APPLY, BRACKET-TRY, BRACKET-IF, BRACKET-LET, and DOUBLE-BRACKET follow because in these cases, $\mathsf{locs}(e') \subseteq \mathsf{locs}(e)$, $M' = M$, and $\mathsf{root}(m^S, e') \Rightarrow \mathsf{root}(m^S, e)$.

Rule GC follows from the fact that $\vdash^{\alpha}_{[wf]} M$ and that minimal collectible groups must be disjoint.

Rules α-CREATE and α-ASSIGN follow trivially from the transition rules.

Rule α-FORGET follows from the fact that $\vdash^{\alpha}_{[wf]} M$.

Case CREATE ($\langle \{\overrightarrow{x_i = v_i}\}^S, M\rangle \rightarrow_\alpha \langle m^S, M[m^S \mapsto \{\overrightarrow{x_i = v_i \blacktriangleright_\alpha \tau_i}\}]\rangle$, where m^S is fresh and $S = \{\overrightarrow{x_i : \tau_i}\}_s$):

We show $\varnothing; \top; \top \vdash v_i : \tau_i, \top$ for arbitrary i; the rest of this case follows from $\vdash^{\alpha}_{[wf]} \langle e, M\rangle$.

From the typing of e, we have $\varnothing; pc; \mathcal{H} \vdash v_i : \tau_i, \top$. From this, the result follows via Lemma 8 and T-BRACKET.

Case ASSIGN ($\langle m^S.x_c := v, M\rangle \rightarrow_\alpha \langle * \blacktriangleright_\alpha p, M[m^S \mapsto v \blacktriangleright_\alpha \tau_c]\rangle$, where $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$):

From the typing of e and by T-SUBSUME, we have $\varnothing; pc; \mathcal{H} \vdash v : \tau_c, \top$. By Lemma 8, we have $\varnothing; \top; \top \vdash v : \tau_c, \top$, and the rest of this case follows from $\vdash^{\alpha}_{[wf]} \langle e, M\rangle$.

Case SOFT-ASSIGN ($\langle (\mathsf{soft}\ m^S).x_c := v, M\rangle \rightarrow_\alpha \langle v' \blacktriangleright_\alpha (a \sqcap p), M'\rangle$, where $\langle m^S.x_c := v, M\rangle \xrightarrow{\mathsf{e}} \langle v', M'\rangle$ and $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$):

If $M(m^S) = \bot$, then we have $\mathsf{locs}(e') = \varnothing$ and $M' = M$, and the result follows.

Otherwise, $M(m^S) \neq \bot$, so from the definition of $\vdash^{\alpha}_{[wf]} \langle (\mathsf{soft}\ m^S).x_c := v, M\rangle$, we can obtain $\vdash^{\alpha}_{[wf]} \langle m^S.x_c := v, M\rangle$, and the rest follows similarly to the previous case.

Case EVAL-CONTEXT ($\langle E[e_1],M\rangle \to_\alpha \langle E[e_1'],M'\rangle$, where $\langle e_1,M\rangle \xrightarrow{\mathsf{e}} \langle e_1',M'\rangle$):

Since $\vdash^{\alpha}_{[wf]} \langle e,M\rangle$, we know $\vdash^{\alpha}_{[wf]} \langle e_1,M\rangle$. Proceeding by cases according to the structure of $E[e_1]$, in each case we have $\varnothing;pc_1;\mathcal{H}_1 \vdash e_1 : \tau_1,\mathcal{X}_1$ (for some pc_1, $\mathcal{H}_1$, τ_1, and $\mathcal{X}_1$). So, by the induction hypothesis, we have $\vdash^{\alpha}_{[wf]} \langle e_1',M'\rangle$. From this and the fact that $\vdash^{\alpha}_{[wf]} \langle e,M\rangle$, the result follows.

Case BRACKET-CONTEXT ($\langle [e_1],M\rangle \to_\alpha \langle [e_1'],M'\rangle$, where $\langle e_1,M\rangle \xrightarrow{\mathsf{e}} \langle e_1',M'\rangle$):

Since $\vdash^{\alpha}_{[wf]} \langle e,M\rangle$, we know $\vdash^{\alpha}_{[wf]} \langle e_1,M\rangle$. From the typing of e, we have $\varnothing;pc';\mathcal{H} \vdash e_1 : \tau_1,\mathcal{X}$, for some appropriate pc' and τ_1. So, by the induction hypothesis, we have $\vdash^{\alpha}_{[wf]} \langle e_1',M'\rangle$, and therefore $\vdash^{\alpha}_{[wf]} \langle e',M'\rangle$.

□

Corollary 11 (Preservation)**.**

$$\begin{aligned}&\vdash^{\alpha}_{[wf]} \langle e,M\rangle \wedge \varnothing;pc;\mathcal{H} \vdash e : \tau,\mathcal{X} \wedge \langle e,M\rangle \to^{*}_{\alpha} \langle e',M'\rangle\\ &\quad\Rightarrow \vdash^{\alpha}_{[wf]} \langle e',M'\rangle \wedge \varnothing;pc;\mathcal{H} \vdash e' : \tau,\mathcal{X}\end{aligned}$$

Proof. This follows from Lemmas 9 and 10 by induction on the number of $\to_\alpha$ transitions taken. □

Corollary 12 (Preservation (nonadversarial execution))**.**

$$\begin{aligned}&\vdash_{[wf]} \langle e,M\rangle \wedge \varnothing;pc;\mathcal{H} \vdash e : \tau,\mathcal{X} \wedge \langle e,M\rangle \to^{*} \langle e',M'\rangle\\ &\quad\Rightarrow \vdash_{[wf]} \langle e',M'\rangle \wedge \varnothing;pc;\mathcal{H} \vdash e' : \tau,\mathcal{X}\end{aligned}$$

Proof. This follows by Corollary 11 and the definition of $\vdash_{[wf]} \langle e,M\rangle$. □

Lemma 13 (Progress)**.** *Let $\langle e,M\rangle$ be a configuration wherein e has type τ and effect $\mathcal{X}$, and the locations appearing in e are mapped by M. Then either e is a value, or $\langle e,M\rangle$ can take an $\xrightarrow{\mathsf{e}}$ transition:*

$$\begin{aligned}&\varnothing;pc;\mathcal{H} \vdash e : \tau,\mathcal{X} \wedge \mathsf{locs}(e) \subseteq \mathsf{dom}(M)\\ &\quad\Rightarrow e \text{ is a value} \vee \exists e',M'.\ \langle e,M\rangle \xrightarrow{\mathsf{e}} \langle e',M'\rangle\end{aligned}$$

(Doubly bracketed values are considered expressions and not values.)

Proof. By induction on the derivation of $\varnothing;pc;\mathcal{H} \vdash e : \tau,\mathcal{X}$. We proceed by cases according to the syntax of e. Note that since e is typed in an empty type context ($\Gamma = \varnothing$), we must have $\mathsf{FV}(e) = \varnothing$.

Case $e = v$:

Trivial since e is a value.

Case $e = v_1\ v_2$:

From the typing of e, we know that v_1 is a value with an arrow type and $\top$ effect, so it is either an abstraction $\lambda(x{:}\tau_1)[pc_1;\mathcal{H}_1].e_1$ or a bracketed value $[v_1']$. In the former case, by APPLY, we have

$$\langle e,M\rangle = \langle (\lambda(x{:}\tau_1)[pc_1;\mathcal{H}_1].e_1)\ v_2,M\rangle \xrightarrow{\mathsf{e}} \langle e_1\{v_2/x\},M\rangle\,.$$

In the latter case, by BRACKET-APPLY, we have $\langle e,M\rangle = \langle [v_1']\ v_2,M\rangle \xrightarrow{\mathsf{e}} \langle [v_1'\ v_2],M\rangle$.

Case $e = \mathsf{if}\ v_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3$:

From the typing of e, we know that v_1 is a value with bool type and $\top$ effect, so either $v_1 = \mathsf{true}$, $v_1 = \mathsf{false}$, or v_1 is a bracketed value $[v_1']$. If $v_1 = \mathsf{true}$, then by IF-TRUE, we have $\langle e,M\rangle = \langle \mathsf{if\ true\ then}\ e_2\ \mathsf{else}\ e_3,M\rangle \xrightarrow{\mathsf{e}} \langle e_2,M\rangle$. If $v_1 = \mathsf{false}$, then by IF-FALSE, we have $\langle e,M\rangle = \langle \mathsf{if\ false\ then}\ e_2\ \mathsf{else}\ e_3,M\rangle \xrightarrow{\mathsf{e}} \langle e_3,M\rangle$. Otherwise, $v_1 = [v_1']$ and by BRACKET-IF, we have $\langle e,M\rangle = \langle \mathsf{if}\ [v_1']\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3,M\rangle \xrightarrow{\mathsf{e}} \langle [\mathsf{if}\ v_1'\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3],M\rangle$.

Case $e = \{\overrightarrow{x_i = v_i}\}^S$:

Trivial by CREATE.

Case $e = v.x_c$:

From the typing of e, we know v is a value with record type and $\top$ effect, so it is either a bracketed hard reference ($v = [m^S]$), a bracketed soft reference ($v = [\mathsf{soft}\ m^S]$), a hard reference ($v = m^S$), or a soft reference ($v = \mathsf{soft}\ m^S$).

In the first case ($v = [m^S]$), by BRACKET-SELECT, we have $\langle e,M\rangle = \langle [m^S].x_c,M\rangle \xrightarrow{\mathsf{e}} \langle [m^S.x_c],M\rangle$. In the second case ($v = [\mathsf{soft}\ m^S]$), by BRACKET-SOFT-SELECT, we have $\langle e,M\rangle = \langle [\mathsf{soft}\ m^S].x_c,M\rangle \xrightarrow{\mathsf{e}} \langle [(\mathsf{soft}\ m^S).x_c],M\rangle$.

In the third ($v = m^S$) and fourth ($v = \mathsf{soft}\ m^S$) cases, assume without loss of generality $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$. Since $\mathsf{locs}(e) \subseteq \mathsf{dom}(M)$, we must have $m^S \in \mathsf{dom}(M)$. There are two sub-cases to consider. In each sub-case, we will show $\langle m^S.x_c,M\rangle \xrightarrow{\mathsf{e}} \langle v'',M\rangle$, for some v'', thereby proving the case of a hard reference ($v = m^S$). The case of a soft reference ($v = \mathsf{soft}\ m^S$) follows by SOFT-SELECT: $\langle (\mathsf{soft}\ m^S).x_c,M\rangle \xrightarrow{\mathsf{e}} \langle v'' \blacktriangleright_\alpha (a \sqcap p),M\rangle$.

1. Case $M(m^S) \neq \bot$:
   Without loss of generality, assume $M(m^S) = \{\overrightarrow{x_i = v_i}\}$. Then, by SELECT,
   $$\langle m^S.x_c,M\rangle \xrightarrow{\mathsf{e}} \langle v_c \blacktriangleright_\alpha p,M\rangle.$$
2. Case $M(m^S) = \bot$:
   By DANGLE-SELECT, $\langle m^S.x_c,M\rangle \xrightarrow{\mathsf{e}} \langle \bot_p \blacktriangleright_\alpha p,M\rangle$.

Case $e = v.x_c := v'$:

From the typing of e, we know that v is a value with record type and $\top$ effect, so it must be either a bracketed hard reference ($v = [m^S]$), a bracketed soft reference ($v = [\mathsf{soft}\ m^S]$), a hard reference ($v = m^S$), or a soft reference ($v = \mathsf{soft}\ m^S$).

In the first case ($v = [m^S]$), by BRACKET-ASSIGN, we have $\langle e,M\rangle = \langle [m^S].x_c := v',M\rangle \xrightarrow{\mathsf{e}} \langle [m^S.x_c := v'],M\rangle$.

In the second case ($v = [\mathsf{soft}\ m^S]$), by BRACKET-SOFT-ASSIGN, we have $\langle e,M\rangle = \langle [\mathsf{soft}\ m^S].x_c := v',M\rangle \xrightarrow{\mathsf{e}} \langle [(\mathsf{soft}\ m^S).x_c := v'],M\rangle$.

For the remaining cases, suppose $v = m^S$ or $v = \mathsf{soft}\ m^S$. Then, since $\mathsf{locs}(e) \subseteq \mathsf{dom}(M)$, we must have $m^S \in \mathsf{dom}(M)$. Without loss of generality, assume $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$. From the typing of e, we also know that v' is a value with $\top$ effect, so $v' \neq \bot_{p'}$ and $v' \neq [\bot_{p'}]$ for all p'. There are two sub-cases to consider. In each sub-case, we will show $\langle m^S.x_c := v',M\rangle \xrightarrow{\mathsf{e}} \langle v''',M'\rangle$, for some v''' and some M', thereby proving the case of a hard reference ($v = m^S$). The case of a soft reference ($v = \mathsf{soft}\ m^S$) follows by SOFT-ASSIGN: $\langle (\mathsf{soft}\ m^S).x_c := v',M\rangle \xrightarrow{\mathsf{e}} \langle v''' \blacktriangleright_\alpha (a \sqcap p),M'\rangle$.

1. Case $M(m^S) \neq \bot$:
   Without loss of generality, assume $M(m^S) = \{\overrightarrow{x_i = v_i}\}$. Then, by ASSIGN,
   $$\langle m^S.x_c := v',M\rangle \xrightarrow{\mathsf{e}} \langle * \blacktriangleright_\alpha p,M[m^S.x_c \mapsto v' \blacktriangleright_\alpha \tau_c]\rangle.$$
2. Case $M(m^S) = \bot$:
   By DANGLE-ASSIGN, $\langle m^S.x_c := v',M\rangle \xrightarrow{\mathsf{e}} \langle \bot_p \blacktriangleright_\alpha p,M\rangle$.

Case $e = \mathsf{exists}\ v\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2$:

From the typing of e, we know that v is a value with soft reference type, so it is either a bracketed value ($v = [v']$) or a soft reference ($v = \mathsf{soft}\ m^S$).

In the first case ($v = [v']$), by BRACKET-EXISTS, we have

$$\langle e,M\rangle = \langle \mathsf{exists}\ [v']\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2,M\rangle \xrightarrow{\mathsf{e}} \langle [\mathsf{exists}\ v'\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2],M\rangle$$

The second case ($v = \mathsf{soft}\ m^S$) splits into two subcases. Assume $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$ without loss of generality. If $M(m^S) \neq \bot$, then by EXISTS-TRUE, we have

$$\langle e,M\rangle = \left\langle \mathsf{exists\ soft}\ m^S\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2, M\right\rangle \xrightarrow{\mathsf{e}} \left\langle (e_1\{m^S/x\}) \blacktriangleright_\alpha (a \sqcap p), M\right\rangle$$

Otherwise, $M(m^S) = \bot$, and by EXISTS-FALSE, we have

$$\langle e,M\rangle = \left\langle \mathsf{exists\ soft}\ m^S\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2, M\right\rangle \xrightarrow{\mathsf{e}} \langle e_2 \blacktriangleright_\alpha (a \sqcap p), M\rangle$$

Case $e = \mathsf{soft}\ e_1$:

From the typing of e, we know that $\varnothing;pc;\mathcal{H} \vdash e_1 : R_w, \mathcal{X}$, where $\tau = (\mathsf{soft}\ R)_w$. Since $\mathsf{locs}(e) \subseteq \mathsf{dom}(M)$, we must also have $\mathsf{locs}(e_1) \subseteq \mathsf{dom}(M)$, so by the induction hypothesis, either e_1 is a value or $\langle e_1, M\rangle \xrightarrow{\mathsf{e}} \langle e_1', M'\rangle$, for some e_1' and M'. There are four cases to consider:

1. e_1 is a value $\bot_p$,
2. e_1 is a bracketed value $[v]$,
3. e_1 is some other value, and
4. $\langle e_1, M\rangle \xrightarrow{\mathsf{e}} \langle e_1', M'\rangle$.

In case 1, by FAIL-PROP, we have $\langle e,M\rangle = \left\langle \mathsf{soft}\ \bot_p, M\right\rangle \xrightarrow{\mathsf{e}} \left\langle \bot_p, M\right\rangle$.

In case 2, by BRACKET-SOFT, we have $\langle e,M\rangle = \langle \mathsf{soft}\ [v], M\rangle \xrightarrow{\mathsf{e}} \langle [\mathsf{soft}\ v], M\rangle$.

In case 3, e_1 must be a hard reference m^S, so $e = \mathsf{soft}\ m^S$ is also a value.

In case 4, by EVAL-CONTEXT, we have $\langle e,M\rangle = \langle \mathsf{soft}\ e_1, M\rangle \xrightarrow{\mathsf{e}} \langle \mathsf{soft}\ e_1', M'\rangle$.

Case $e = e_1 \parallel e_2$:

From the typing of e, we have $\varnothing;pc;\top \vdash e_i : \tau_i, \top$ for $i \in \{1,2\}$. Since $\mathsf{locs}(e) \subseteq \mathsf{dom}(M)$, we must also have $\mathsf{locs}(e_i) \subseteq \mathsf{dom}(M)$, so by the induction hypothesis, either e_1 and e_2 are both values, or (without loss of generality) $\langle e_1, M\rangle \xrightarrow{\mathsf{e}} \langle e_1', M'\rangle$, for some e_1' and M'.

If e_1 and e_2 are both values, then by PARALLEL-RESULT, we have $\langle e,M\rangle = \langle e_1 \parallel e_2, M\rangle \xrightarrow{\mathsf{e}} \langle *, M\rangle$.

Otherwise, by EVAL-CONTEXT, we have $\langle e,M\rangle = \langle e_1 \parallel e_2, M\rangle \xrightarrow{\mathsf{e}} \langle e_1' \parallel e_2, M'\rangle$.

Case $e = \mathsf{try}\ e_1\ \mathsf{catch}\ p\colon e_2$:

From the typing of e, we have $\varnothing;pc;\mathcal{H}, p \vdash e_1 : \tau_1, \mathcal{X}_1$, where $\tau = \tau_1 \sqcap w$ with $w = \bigsqcap_{p' \in \mathcal{X}_1} (p \sqcup p')$. Since $\mathsf{locs}(e) \subseteq \mathsf{dom}(M)$, we must also have $\mathsf{locs}(e_1) \subseteq \mathsf{dom}(M)$, so by the induction hypothesis, either e_1 is a value or $\langle e_1, M\rangle \xrightarrow{\mathsf{e}} \langle e_1', M'\rangle$, for some e_1' and M'. There are five cases to consider:

1. e_1 is a value $\bot_{p'}$ and $\vdash p \preccurlyeq p'$,
2. e_1 is a value $\bot_{p'}$ and $p \not\preccurlyeq p'$,
3. e_1 is a bracketed value $[v]$,
4. e_1 is some other value v, and
5. $\langle e_1, M\rangle \xrightarrow{\mathsf{e}} \langle e_1', M'\rangle$.

In case 1, by TRY-CATCH, we have $\langle e,M\rangle = \left\langle \mathsf{try}\ \bot_{p'}\ \mathsf{catch}\ p\colon e_2, M\right\rangle \xrightarrow{\mathsf{e}} \langle e_2, M\rangle$.

In case 2, by TRY-ESC, we have $\langle e,M\rangle = \left\langle \mathsf{try}\ \bot_{p'}\ \mathsf{catch}\ p\colon e_2, M\right\rangle \xrightarrow{\mathsf{e}} \left\langle \bot_{p'}, M\right\rangle$.

In case 3, by BRACKET-TRY, we have $\langle e,M\rangle = \langle \mathsf{try}\ [v]\ \mathsf{catch}\ p\colon e_2, M\rangle \xrightarrow{\mathsf{e}} \langle [\mathsf{try}\ v\ \mathsf{catch}\ p\colon e_2], M\rangle$.

In case 4, by TRY-VAL, we have $\langle e,M\rangle = \langle \mathsf{try}\ v\ \mathsf{catch}\ p\colon e_2, M\rangle \xrightarrow{\mathsf{e}} \langle v, M\rangle$.

Finally, in case 5, by EVAL-CONTEXT, we have

$$\langle e,M\rangle = \langle \mathsf{try}\ e_1\ \mathsf{catch}\ p\colon e_2, M\rangle \xrightarrow{\mathsf{e}} \left\langle \mathsf{try}\ e_1'\ \mathsf{catch}\ p\colon e_2, M'\right\rangle$$

Case $e = \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$:

From the typing of e, we have $\varnothing; pc; \mathcal{H} \vdash e_1 : \tau_1, \mathcal{X}_1$. Since $\mathsf{locs}(e) \subseteq \mathsf{dom}(M)$, we must also have $\mathsf{locs}(e_1) \subseteq \mathsf{dom}(M)$, so by the induction hypothesis, either e_1 is a value or $\langle e_1, M\rangle \xrightarrow{\mathsf{e}} \langle e_1', M'\rangle$, for some e_1' and M'. There are five cases to consider:

1. e_1 is a bottom value $\bot_p$,
2. e_1 is a bracketed bottom value $[\bot_p]$,
3. e_1 is some other bracketed value $[v]$,
4. e_1 is some other value v, and
5. $\langle e_1, M\rangle \xrightarrow{\mathsf{e}} \langle e_1', M'\rangle$.

In case 1, by FAIL-PROP, we have $\langle e, M\rangle = \big\langle \mathsf{let}\ x = \bot_p\ \mathsf{in}\ e_2, M\big\rangle \xrightarrow{\mathsf{e}} \big\langle \bot_p, M\big\rangle$.

In case 2, by BRACKET-FAIL, we have $\langle e, M\rangle = \big\langle \mathsf{let}\ x = [\bot_p]\ \mathsf{in}\ e_2, M\big\rangle \xrightarrow{\mathsf{e}} \big\langle [\bot_p], M\big\rangle$.

In case 3, by BRACKET-LET, we have $\langle e, M\rangle = \langle \mathsf{let}\ x = [v]\ \mathsf{in}\ e_2, M\rangle \xrightarrow{\mathsf{e}} \langle [e_2\{[v]/x\}], M\rangle$.

In case 4, by LET, we have $\langle e, M\rangle = \langle \mathsf{let}\ x = v\ \mathsf{in}\ e_2, M\rangle \xrightarrow{\mathsf{e}} \langle e_2\{v/x\}, M\rangle$.

In case 5, by EVAL-CONTEXT, we have $\langle e, M\rangle = \langle \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2, M\rangle \xrightarrow{\mathsf{e}} \langle \mathsf{let}\ x = e_1'\ \mathsf{in}\ e_2, M'\rangle$.

Case $e = [e']$:

From the typing of e, we have $\varnothing; pc'; \mathcal{H} \vdash e' : \tau', \mathcal{X}$. Since $\mathsf{locs}(e) \subseteq \mathsf{dom}(M)$, we must also have $\mathsf{locs}(e') \subseteq \mathsf{dom}(M)$, so by the induction hypothesis, either e' is a value or $\langle e', M\rangle \xrightarrow{\mathsf{e}} \langle e'', M'\rangle$.

If e' is a bracketed value $[v]$, then by DOUBLE-BRACKET, we have $\langle e, M\rangle = \langle [[v]], M\rangle \xrightarrow{\mathsf{e}} \langle [v], M\rangle$.

If e' is some other value v, then $e = [v]$ is a value.

Otherwise, by BRACKET-CONTEXT, we have $\langle e, M\rangle = \langle [e'], M\rangle \xrightarrow{\mathsf{e}} \langle [e''], M'\rangle$.

□

Corollary 14 (Soundness of $[\lambda_{persist}]$).

$$\begin{aligned} &\vdash^{\alpha}_{[wf]} \langle e, M\rangle \wedge \varnothing; pc; \mathcal{H} \vdash e : \tau, \mathcal{X} \\ &\quad \Rightarrow \langle e, M\rangle \Uparrow \vee \exists v \in \mathsf{Val}, M'.\ \langle e, M\rangle \xrightarrow{\mathsf{e}}{}^{*} \langle v, M'\rangle \end{aligned}$$

Proof. This follows from Corollary 11 and Lemma 13 by induction on the number of $\xrightarrow{\mathsf{e}}$ transitions taken. □

8.4 Security relation

The key to proving both referential integrity and immunity to storage attacks is to show that the adversary cannot meaningfully influence the high-integrity parts of the program and memory. This property is similar to noninterference [10], and similarly can be expressed using a security relation on configurations. Two configurations are related if they agree on all high-integrity parts of the program and of the memory.

The property states that for any execution influenced by the adversary, there is a corresponding, related execution in which the adversary is not present. Hence, the adversary's influence is not significant. More precisely, each configuration $\langle e_1, M_1\rangle$ reached via the language augmented by adversarial transitions must be related to some configuration $\langle e_2, M_2\rangle$ reachable by purely nonadversarial execution. This is a possibilistic security property, which is problematic for confidentiality properties [21], but is acceptable for integrity.

Because the two executions being compared operate on different heaps, with the adversary behaving differently in the two executions, the addresses chosen during record allocation may differ. However, the structure of the high-integrity part of the heap should still correspond. A *high-integrity homomorphism* ϕ is used to relate corresponding locations in the two heaps that are high-integrity or high-persistence. High-integrity homomorphisms are injective, preserve location types, and are isomorphisms on both high-integrity and high-persistence locations.

$$\frac{}{* \approx^{\phi}_{\alpha} *} \qquad \frac{}{x \approx^{\phi}_{\alpha} x} \qquad \frac{b \in \{\mathsf{true}, \mathsf{false}\}}{b \approx^{\phi}_{\alpha} b} \qquad \frac{\phi(m_1^S) = m_2^S}{m_1^S \approx^{\phi}_{\alpha} m_2^S} \qquad \frac{e \approx^{\phi}_{\alpha} e'}{\lambda(x{:}\tau)[pc;\mathcal{H}].\,e \approx^{\phi}_{\alpha} \lambda(x{:}\tau)[pc;\mathcal{H}].\,e'}$$

$$\frac{}{\bot_p \approx^{\phi}_{\alpha} \bot_p} \qquad \frac{v_i \approx^{\phi}_{\alpha} v_i' \ (\forall i)}{v_1\ v_2 \approx^{\phi}_{\alpha} v_1'\ v_2'} \qquad \frac{e_i \approx^{\phi}_{\alpha} e_i' \ (\forall i)}{\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \approx^{\phi}_{\alpha} \mathsf{if}\ e_1'\ \mathsf{then}\ e_2'\ \mathsf{else}\ e_3'} \qquad \frac{v_i \approx^{\phi}_{\alpha} v_i' \ (\forall i)}{\{\overrightarrow{x_i = v_i}\}^S \approx^{\phi}_{\alpha} \{\overrightarrow{x_i = v_i'}\}^S}$$

$$\frac{v \approx^{\phi}_{\alpha} v'}{v.x \approx^{\phi}_{\alpha} v'.x} \qquad \frac{v_i \approx^{\phi}_{\alpha} v_i' \ (\forall i)}{v_1.x := v_2 \approx^{\phi}_{\alpha} v_1'.x := v_2'} \qquad \frac{e \approx^{\phi}_{\alpha} e'}{\mathsf{soft}\ e \approx^{\phi}_{\alpha} \mathsf{soft}\ e'} \qquad \frac{}{[e] \approx^{\phi}_{\alpha} [e']} \qquad \frac{e_i \approx^{\phi}_{\alpha} e_i' \ (\forall i)}{e_1 \parallel e_2 \approx^{\phi}_{\alpha} e_1' \parallel e_2'}$$

$$\frac{e_i \approx^{\phi}_{\alpha} e_i' \ (\forall i)}{\mathsf{exists}\ e_1\ \mathsf{as}\ x : e_2\ \mathsf{else}\ e_3 \approx^{\phi}_{\alpha} \mathsf{exists}\ e_1'\ \mathsf{as}\ x : e_2'\ \mathsf{else}\ e_3'} \qquad \frac{e_i \approx^{\phi}_{\alpha} e_i' \ (\forall i)}{\mathsf{try}\ e_1\ \mathsf{catch}\ p{:}\ e_2 \approx^{\phi}_{\alpha} \mathsf{try}\ e_1'\ \mathsf{catch}\ p{:}\ e_2'}$$

$$\frac{e_i \approx^{\phi}_{\alpha} e_i' \ (\forall i)}{\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \approx^{\phi}_{\alpha} \mathsf{let}\ x = e_1'\ \mathsf{in}\ e_2'}$$

Figure 12: Security relation on expressions in $[\lambda_{persist}]$

Definition 11 (High-integrity homomorphism)**.** *An injective partial function* $\phi : \mathsf{dom}(M_1) \rightharpoonup \mathsf{dom}(M_2)$ *is a* high-integrity homomorphism from M_1 to M_2 *if it satisfies the following:*

- *Injective:* $m_1^{S_1} \neq m_2^{S_2} \wedge \{m_1^{S_1}, m_2^{S_2}\} \subseteq \mathsf{dom}(\phi) \Rightarrow \phi(m_1^{S_1}) \neq \phi(m_2^{S_2})$;
- *Type-preserving:* $m_2^{S_2} = \phi(m_1^{S_1}) \Rightarrow S_1 = S_2$; *and*
- *Isomorphous when the domain and range are restricted to the high-integrity and high-persistence locations in* M_1 *and* M_2*:*
$$\phi|_D : D \rightarrowtail\!\!\!\!\twoheadrightarrow R,$$
where $D = \{m^S \in \mathsf{dom}(M_1) \mid \ \vdash \alpha \preccurlyeq \mathsf{integ}(S) \vee \vdash \alpha \preccurlyeq \mathsf{persist}(S)\}$ *and* $R = \{m^S \in \mathsf{dom}(M_2) \mid \ \vdash \alpha \preccurlyeq \mathsf{integ}(S) \vee \vdash \alpha \preccurlyeq \mathsf{persist}(S)\}$. *The notation* $\mathsf{integ}(S)$ *denotes the integrity of a record with type S: the least upper bound of its fields' integrity.*
$$\mathsf{integ}(\{\overrightarrow{x_i : \tau_i}\}_s) = \bigsqcup_i \mathsf{integ}(\tau_i)$$

We are now ready to define our security relation on expressions. An expression e_1 is considered to be related to e_2 via a high-integrity homomorphism ϕ, written $e_1 \approx^{\phi}_{\alpha} e_2$, if e_1 is equal to e_2 (modulo bracketed expressions) when the memory locations in e_1 are transformed via ϕ. This is defined formally in Figure 12.

To ensure that high-integrity dereferences yield related results, we also define a security relation on memories: M_1 and M_2 are related via ϕ, written $M_1 \approx^{\phi}_{\alpha} M_2$, if two conditions hold for each location $m^S \in \mathsf{dom}(\phi)$. If m^S is not deleted, then $\phi(m^S)$ maps to a related record. Otherwise, if m^S is deleted, high-authority, and high-persistence, then so is $\phi(m^S)$.

Definition 12 (Security relation on memories)**.** *Two memories,* M_1 *and* M_2*, are* ϕ*-related, written* $M_1 \approx^{\phi}_{\alpha} M_2$*, if for any location* $m^S \in \mathsf{dom}(\phi)$,

1. *if* m^S *is not deleted, then* $\phi(m^S)$ *maps to a* ϕ*-related record; and*
2. *if* m^S *is a deleted high-authority, high-persistence location, then so is* $\phi(m^S)$.

$$\begin{aligned} M_1 \approx^{\phi}_{\alpha} M_2 \overset{def.}{\iff} {} & \forall m^S \in \mathsf{dom}(\phi). \\ & (M_1(m^S) \neq \bot \Rightarrow M_2(\phi(m^S)) \neq \bot \wedge M_1(m^S) \approx^{\phi}_{\alpha} M_2(\phi(m^S))) \\ & \wedge (\vdash \alpha \preccurlyeq \mathsf{auth}^{+}(S) \sqcap \mathsf{persist}(S) \wedge M_1(m^S) = \bot \Rightarrow M_2(\phi(m^S)) = \bot) \end{aligned}$$

These two security relations induce a security relation on configurations:

$$\langle e_1, M_1 \rangle \approx^{\phi}_{\alpha} \langle e_2, M_2 \rangle \overset{def.}{\iff} e_1 \approx^{\phi}_{\alpha} e_2 \wedge M_1 \approx^{\phi}_{\alpha} M_2.$$

A $[\lambda_{persist}]$ program has limited adversary influence if related initial configurations produce related final configurations. We now see that the language $[\lambda_{persist}]$ enforces security, because all well-formed programs do have limited adversary influence.

8.5 Referential integrity

Theorem 1 formalizes the referential integrity result, showing that the adversary has limited influence on program execution: execution in the presence of an adversary is ϕ-related to a nonadversarial execution.

Theorem 1 (Referential integrity)**.** *Assume $\langle e_1, M_1 \rangle$ is a well-formed configuration and $\langle e_2, M_2 \rangle$ is a well-formed, nonadversarial, ϕ-related configuration, such that e_1 and e_2 have type τ and M_2 is well-formed:*

$$\begin{gathered} \vdash^{\alpha}_{[wf]} \langle e_1, M_1 \rangle \ \wedge\ \vdash_{[wf]} \langle e_2, M_2 \rangle \ \wedge\ \langle e_1, M_1 \rangle \approx^{\phi}_{\alpha} \langle e_2, M_2 \rangle \\ \wedge\ \varnothing; pc; \mathcal{H} \vdash e_1 : \tau, \mathcal{X} \ \wedge\ \varnothing; pc; \mathcal{H} \vdash e_2 : \tau, \mathcal{X} \ \wedge\ \vdash^{\alpha}_{[wf]} M_2 \end{gathered}$$

Suppose $\langle e_1, M_1 \rangle$ takes some number of steps in the presence of an adversary to another configuration $\langle e_1', M_1' \rangle$. Then either $\langle e_2, M_2 \rangle$ diverges, or it can take some number of steps in the absence of an adversary to another configuration $\langle e_2', M_2' \rangle$ and there exists a high-integrity homomorphism ϕ' from M_1' to M_2' that extends ϕ, such that $\langle e_1', M_1' \rangle$ is related to $\langle e_2', M_2' \rangle$ via ϕ':

$$\begin{aligned} &\langle e_1, M_1 \rangle \to^{*}_{\alpha} \langle e_1', M_1' \rangle \wedge \neg \langle e_2, M_2 \rangle \Uparrow \\ &\qquad \Rightarrow \exists e_2', M_2', \phi'.\ \langle e_2, M_2 \rangle \to^{*} \langle e_2', M_2' \rangle \wedge \langle e_1', M_1' \rangle \approx^{\phi'}_{\alpha} \langle e_2', M_2' \rangle \wedge \phi = \phi'|_{\mathsf{dom}(\phi)} \end{aligned}$$

To prove this, we first need to prove a few preliminary results about garbage collection, given by Lemmas 15–25.

Lemma 15. *Let e be well-typed in a low-integrity context. Assume that if e is a memory location, then it is low-authority. If m^S is a GC root in e, then m^S must be low-authority:*

$$\begin{aligned} &\Gamma; pc; \mathcal{H} \vdash e : \tau, \mathcal{X} \wedge \alpha \not\leq pc \\ &\wedge (e \in \mathsf{dom}(M) \Rightarrow \alpha \not\leq \mathsf{auth}^{+}(\tau)) \wedge \mathsf{root}(m^S, e) \\ &\qquad \Rightarrow \alpha \not\leq \mathsf{auth}^{+}(S) \end{aligned}$$

Proof. By induction on the derivation of $\mathsf{root}(m^S, e)$.

Case R1 ($e = m^S$):

Without loss of generality, assume $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$, where $\alpha \not\leq a$. We have $\tau = (\{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_{\top}$, so $\mathsf{auth}^{+}(\tau) = \mathsf{auth}^{+}(S)$ and the result follows trivially.

Case R2 ($e = \mathsf{soft}\ e'$, where $\forall m_1^{S'}.\ e' \neq m_1^{S'}$):

From the derivation of $\mathsf{root}(m^S, e)$, we have $\mathsf{root}(m^S, e')$. From the typing of e, we have $\Gamma; pc; \mathcal{H} \vdash e' : R_w, \mathcal{X}$. Therefore, the induction hypothesis applies, and the result follows.

Case R3 ($e = \{\overrightarrow{x_i = v_i}\}^{S'}$):

From the derivation of $\mathsf{root}(m^S, e)$, we have $\mathsf{root}(m^S, v_c)$ for some c.

From the typing of e, we know $\Gamma; pc; \mathcal{H} \vdash v_c : \tau', \top$, where $\vdash \mathsf{auth}^{+}(\tau') \leq pc$. From this, we also know $\alpha \not\leq \mathsf{auth}^{+}(\tau')$.

Therefore, the induction hypothesis applies, and the result follows.

Case R4 ($e = v.x$):

From the derivation of $\mathsf{root}(m^S, e)$, we have $\mathsf{root}(m^S, v)$.

From the typing of e, we have $\Gamma; pc; \mathcal{H} \vdash v : \tau', \top$, where either $\tau' = (\{\overrightarrow{x_i : \tau_i}\}_{(a^+, a^-, p)})_w$ with $\vdash a^+ \preccurlyeq pc$, or $\tau' = (\mathsf{soft}\ R)_w$ for some R. We therefore know $\alpha \not\preccurlyeq \mathsf{auth}^+(\tau')$.

Therefore, the induction hypothesis applies, and the result follows.

Case R5 ($e = v_1.x := v_2$):

From the derivation of $\mathsf{root}(m^S, e)$, we have $\mathsf{root}(m^S, v_i)$, for some $i \in \{1, 2\}$.

From the typing of e, we know $\Gamma; pc; \mathcal{H} \vdash v_1 : \tau', \top$ (where either $\tau' = (\{\overrightarrow{x_i : \tau_i}\}_{(a^+, a^-, p)})_w$ with $\vdash a^+ \preccurlyeq pc$, or $\tau' = (\mathsf{soft}\ R)_w$ for some R) and $\Gamma; pc; \mathcal{H} \vdash v_2 : \tau'', \top$, where $\vdash \mathsf{auth}^+(\tau'') \preccurlyeq pc$. Therefore, $\alpha \not\preccurlyeq \mathsf{auth}^+(\tau')$ and $\alpha \not\preccurlyeq \mathsf{auth}^+(\tau'')$. So, the induction hypothesis applies for $i \in \{1, 2\}$, and the result follows.

Case R6 ($e = \lambda(x{:}\tau')[pc'; \mathcal{H}'].e'$):

From the typing of e, we know $\vdash_{wf} \tau : \mathsf{type}$, $\vdash pc' \preccurlyeq pc$, and $\Gamma, x{:}\tau'; pc'; \mathcal{H}' \vdash e' : \tau'', \mathcal{H}'$, where $\tau = \tau' \xrightarrow{pc', \mathcal{H}'} \tau''$. Since $\alpha \not\preccurlyeq pc$, we have $\alpha \not\preccurlyeq pc'$. From $\vdash_{wf} \tau : \mathsf{type}$, we know $\vdash \mathsf{auth}^+(\tau'') \preccurlyeq pc'$, so we therefore know $\alpha \not\preccurlyeq \mathsf{auth}^+(\tau'')$. From the derivation of $\mathsf{root}(m^S, e)$, we know $\mathsf{root}(m^S, e')$. Therefore, the induction hypothesis applies, and the result follows.

Case R7 ($e = v_1\ v_2$):

From the derivation of $\mathsf{root}(m^S, e)$, we have $\mathsf{root}(m^S, v_i)$ for some $i \in \{1, 2\}$.

From the typing of e, we know $\Gamma; pc; \mathcal{H} \vdash v_1 : (\tau' \xrightarrow{pc', \mathcal{H}'} \tau)_w, \top$, $\Gamma; pc; \mathcal{H} \vdash v_2 : \tau', \top$, and $\vdash_{wf} (\tau' \xrightarrow{pc', \mathcal{H}'} \tau)_w : \mathsf{type}$, with $\vdash pc' \preccurlyeq pc$. Therefore, $\alpha \not\preccurlyeq pc'$.

From the derivation of $\vdash_{wf} (\tau' \xrightarrow{pc', \mathcal{H}'} \tau)_w : \mathsf{type}$, we have $\vdash \mathsf{auth}^+(\tau') \preccurlyeq pc$ and $\vdash \mathsf{auth}^+(\tau) \preccurlyeq pc$. We therefore know $\alpha \not\preccurlyeq \mathsf{auth}^+(\tau')$.

So, the induction hypothesis applies for $i \in \{1, 2\}$, and the result follows.

Case R8 ($e = \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$):

From the derivation of $\mathsf{root}(m^S, e)$, we have $\mathsf{root}(m^S, e_i)$, for some $i \in \{1, 2\}$.

From the typing of e, we know $\Gamma; pc; \mathcal{H} \vdash e_1 : \tau', \mathcal{X}_1$ and $\Gamma, x{:}\tau'; pc'; \mathcal{H} \vdash e_2 : \tau'', \mathcal{X}_2$, where $\vdash \mathsf{auth}^+(\tau') \preccurlyeq pc$, $pc' = pc \sqcap w$, and $\vdash \mathsf{auth}^+(\tau'') \preccurlyeq pc'$, for some w. Therefore, it follows that $\alpha \not\preccurlyeq pc'$, $\alpha \not\preccurlyeq \mathsf{auth}^+(\tau')$, and $\alpha \not\preccurlyeq \mathsf{auth}^+(\tau'')$.

So, the induction hypothesis applies for $i \in \{1, 2\}$, and the result follows.

Case R9 ($e = e_1 \parallel e_2$):

From the derivation of $\mathsf{root}(m^S, e)$, we have $\mathsf{root}(m^S, e_i)$ for some $i \in \{1, 2\}$. Without loss of generality, assume $i = 1$.

From the typing of e, we know $\Gamma; pc; \top \vdash e_1 : \tau_1, \top$ and $\vdash \mathsf{auth}^+(\tau_1) \preccurlyeq pc$. From this, we also know $\alpha \not\preccurlyeq \mathsf{auth}^+(\tau_1)$.

Therefore, the induction hypothesis applies, and the result follows.

Case R10 ($e = \mathsf{if}\ v\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$):

From the typing of e, we know $\Gamma; pc; \mathcal{H} \vdash v : \mathsf{bool}_w, \top$ and $\Gamma; pc \sqcap w; \mathcal{H} \vdash e_i : \tau', \mathcal{X}_i$ (for $i \in \{1, 2\}$), where $\vdash \mathsf{auth}^+(\tau') \preccurlyeq pc$. From this, we therefore know $\alpha \not\preccurlyeq \mathsf{auth}^+(\tau')$.

From the derivation of $\mathsf{root}(m^S, e)$, we have either $\mathsf{root}(m^S, v)$ or $\mathsf{root}(m^S, e_j)$, for some $j \in \{1, 2\}$. In all cases, the induction hypothesis applies, and the result follows.

Case R11 ($e = \mathsf{exists}\ v\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2$):

From the typing of e, we know $\Gamma; pc; \mathcal{H} \vdash v : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_r)_w, \top$ and $\Gamma_i; pc \sqcap w'; \mathcal{H} \vdash e_i : \tau', \mathcal{X}_i$ (for $i \in \{1, 2\}$), where $\vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap w'$. From this, we therefore know $\alpha \not\preccurlyeq \mathsf{auth}^+(\tau')$.

From the derivation of $\mathsf{root}(m^S, e)$, we have either $\mathsf{root}(m^S, v)$ or $\mathsf{root}(m^S, e_j)$, for some $j \in \{1, 2\}$. In all cases, the induction hypothesis applies, and the result follows.

Case R12 ($e = \mathsf{try}\ e_1\ \mathsf{catch}\ p\colon e_2$):

From the derivation of $\mathsf{root}(m^S, e)$, we have $\mathsf{root}(m^S, e_i)$, for some $i \in \{1,2\}$.

From the typing of e, we know $\Gamma; pc; \mathcal{H}, p \vdash e_1 : \tau', \mathcal{X}_1$ and $\Gamma; pc'; \mathcal{H} \vdash e_2 : \tau', \mathcal{X}_2$, where $\vdash pc' \preccurlyeq pc$ and $\vdash \mathsf{auth}^+(\tau') \preccurlyeq pc$. From this, we also know $\alpha \not\preccurlyeq pc'$ and $\alpha \not\preccurlyeq \mathsf{auth}^+(\tau')$.

So, the induction hypothesis applies for $i \in \{1,2\}$, and the result follows.

Case R13 ($e = [e']$):

From the derivation of $\mathsf{root}(m^S, e)$, we have $\mathsf{root}(m^S, e')$.

From the typing of e, we know $\Gamma; pc'; \mathcal{H} \vdash e' : \tau', \mathcal{X}$, where $\vdash pc' \preccurlyeq pc$ and $\vdash \mathsf{auth}^+(\tau') \preccurlyeq pc$. From this, it follows that $\alpha \not\preccurlyeq pc'$ and $\alpha \not\preccurlyeq \mathsf{auth}^+(\tau')$.

Therefore, the induction hypothesis applies, and the result follows.

□

Lemma 16. *Suppose m^S is a high-authority location that is a GC root in the well-typed value v. Then v must have high-authority type.*

$$\begin{aligned} &\vdash \alpha \preccurlyeq \mathsf{auth}^+(S) \wedge \mathsf{root}(m^S, v) \wedge \varnothing; \top; \top \vdash v : \tau, \top \\ &\quad \Rightarrow \vdash \alpha \preccurlyeq \mathsf{auth}^+(\tau) \end{aligned}$$

Proof. Since $\mathsf{root}(m^S, v)$, the value v must either be equal to m^S, be a lambda abstraction $\lambda(x{:}\tau)[pc; \mathcal{H}].e$, or be a bracketed value $[v']$.

Case $v = m^S$:

We therefore have $\varnothing; \top; \top \vdash m^S : \tau, \top$. So $\vdash \mathsf{auth}^+(S) \preccurlyeq \mathsf{auth}^+(\tau)$. Therefore, $\vdash \alpha \preccurlyeq \mathsf{auth}^+(\tau)$, as desired.

Case $v = \lambda(x{:}\tau)[pc; \mathcal{H}].e$:

From the derivation of $\mathsf{root}(m^S, v)$, we have $\mathsf{root}(m^S, e)$. So, we must have $\tau = \tau_1 \xrightarrow{pc, \mathcal{H}} \tau_2$, for some τ_1 and some τ_2. Therefore, $\mathsf{auth}^+(\tau) = pc$.

Suppose $\alpha \not\preccurlyeq pc$. From the typing of v, we know $x{:}\tau_1; pc; \mathcal{H} \vdash e : \tau_2, \mathcal{H}$ and $\vdash_{wf} \tau_1 \xrightarrow{pc, \mathcal{H}} \tau_2 : \mathsf{type}$. Therefore, we also know $\vdash \mathsf{auth}^+(\tau_2) \preccurlyeq pc$, and hence, $\alpha \not\preccurlyeq \mathsf{auth}^+(\tau_2)$. Since $\mathsf{root}(m^S, e)$, by Lemma 15, we must have $\alpha \not\preccurlyeq \mathsf{auth}^+(S)$, a contradiction. So, we must have $\vdash \alpha \preccurlyeq pc = \mathsf{auth}^+(\tau)$, as desired.

Case $v = [v']$:

From the derivation of $\mathsf{root}(m^S, v)$, we have $\mathsf{root}(m^S, v')$. From the typing derivation of v, we know $\varnothing; \ell; \top \vdash v' : \tau', \top$ and $\vdash \mathsf{auth}^+(\tau') \preccurlyeq \ell$, where $\alpha \not\preccurlyeq \ell$. Therefore, $\alpha \not\preccurlyeq \mathsf{auth}^+(\tau')$. Since $\mathsf{root}(m^S, v')$, by Lemma 15, we must have $\alpha \not\preccurlyeq \mathsf{auth}^+(S)$, a contradiction. This case therefore holds vacuously.

□

Lemma 17. *Suppose m^S is a high-authority location that is noncollectible in the configuration $\langle v, M \rangle$, where M is well-formed and the value v has type τ. Then τ must be high-authority.*

$$\begin{aligned} &\vdash^{\alpha}_{[wf]} M \wedge \vdash \alpha \preccurlyeq \mathsf{auth}^+(S) \wedge \mathsf{nc}(m^S, \langle v, M \rangle) \wedge \varnothing; \top; \top \vdash v : \tau, \top \\ &\quad \Rightarrow \vdash \alpha \preccurlyeq \mathsf{auth}^+(\tau) \end{aligned}$$

Proof. By induction on the derivation of $\mathsf{nc}(m^S, \langle v, M \rangle)$.

Case NC1:

We have $\mathsf{root}(m^S, v)$. The result follows via Lemma 16.

Case NC2:

We have $\mathsf{root}(m_1^{S_1}, v)$, $M(m_1^{S_1}) = \{\overrightarrow{x_i = v_i}\}$, and $\mathsf{nc}(m^S, \langle v_c, M\rangle)$ for some c. Without loss of generality, assume $S_1 = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$. From $\vdash_{[wf]}^{\alpha} M$, we have $\varnothing;\top;\top \vdash v_c : \tau_c, \top$. Therefore, we can apply the induction hypothesis to obtain $\vdash \alpha \preccurlyeq \mathsf{auth}^+(\tau_c)$. From $\vdash_{[wf]}^{\alpha} M$, we know $\vdash_{wf} S_1 : \mathsf{rectype}$, and therefore, $\vdash \mathsf{auth}^+(\tau_c) \preccurlyeq a$. Hence, $\vdash \alpha \preccurlyeq a = \mathsf{auth}^+(S_1)$. The result follows via Lemma 16.

□

Corollary 18. *Suppose m^S is a high-authority location that is noncollectible in the configuration $\left\langle m_1^{S_1}, M\right\rangle$, where M is well-formed. Then $m_1^{S_1}$ is high-integrity, high-authority, and high-persistence.*

$$\vdash_{[wf]}^{\alpha} M \wedge \vdash \alpha \preccurlyeq \mathsf{auth}^+(S) \wedge \mathsf{nc}(m^S, \left\langle m_1^{S_1}, M\right\rangle)$$
$$\Rightarrow \vdash \alpha \preccurlyeq \mathsf{integ}(S_1) \sqcap \mathsf{auth}^+(S_1) \sqcap \mathsf{persist}(S_1)$$

Proof. Since $\vdash_{[wf]}^{\alpha} M$, we must have $\vdash_{wf} S_1 : \mathsf{rectype}$, and so, $\vdash \mathsf{auth}^+(S_1) \preccurlyeq \mathsf{integ}(S_1) \sqcap \mathsf{persist}(S_1)$. It therefore suffices to show that $m_1^{S_1}$ is high-authority. This follows from Lemma 17. □

Lemma 19. *Let m^S be part of a group G that is collectible in $\langle e, M\rangle$. If m^S is noncollectible in $\left\langle m_1^{S_1}, M\right\rangle$, then $m_1^{S_1}$ must also be in G.*

$$\mathsf{gc}(G, \langle e, M\rangle) \wedge m^S \in G \wedge \mathsf{nc}(m^S, \left\langle m_1^{S_1}, M\right\rangle) \Rightarrow m_1^{S_1} \in G$$

Proof. By induction on the derivation of $\mathsf{nc}(m^S, \left\langle m_1^{S_1}, M\right\rangle)$.

Case NC1:

We have $\mathsf{root}(m^S, m_1^{S_1})$. From this, it follows that $m_1^{S_1} = m^S$, and the result follows trivially.

Case NC2:

We have $M(m_1^{S_1}) = \{\overrightarrow{x_i = v_i}\}$, and $\mathsf{nc}(m^S, \langle v_c, M\rangle)$ for some c. Suppose $\mathsf{root}(m_2^{S_2}, v_c)$ for some $m_2^{S_2} \in G$. From this, we know $\mathsf{root}(m_2^{S_2}, M(m_1^{S_1}))$, and from the definition of $\mathsf{gc}(G, \langle e, M\rangle)$, we have $m_1^{S_1} \in G$, as desired.

We proceed by cases according to the derivation of $\mathsf{nc}(m^S, \langle v_c, M\rangle)$ to find such an $m_2^{S_2}$.

Case NC1:

We have $\mathsf{root}(m^S, v_c)$, so choose $m_2^{S_2} = m^S$.

Case NC2:

We have $\mathsf{root}(m_3^{S_3}, v_c)$, $M(m_3^{S_3}) = \{\overrightarrow{x_i = u_i}\}$, and $\mathsf{nc}(m^S, \langle u_{c'}, M\rangle)$ for some $m_3^{S_3}$ and some c'. It therefore follows that $\mathsf{nc}(m^S, \left\langle m_3^{S_3}, M\right\rangle)$. So, we can apply the induction hypothesis to obtain $m_3^{S_3} \in G$. Therefore, we can choose $m_2^{S_2} = m_3^{S_3}$.

□

Lemma 20. *All locations in a collectible group are collectible:*

$$\mathsf{gc}(G, \langle e, M\rangle) \wedge m^S \in G \Rightarrow \neg\mathsf{nc}(m^S, \langle e, M\rangle).$$

Proof. By contradiction. Let $m^S \in G$ be such that $\mathsf{nc}(m^S, \langle e, M\rangle)$. We proceed by cases according to the derivation of $\mathsf{nc}(m^S, \langle e, M\rangle)$.

Case NC1:

We have $\mathsf{root}(m^S, e)$. But from the definition of $\mathsf{gc}(G, \langle e, M\rangle)$, no such m^S can exist; a contradiction.

Case NC2:

We have $\mathsf{root}(m_1^{S_1}, e)$, $M(m_1^{S_1}) = \{\overrightarrow{x_i = v_i}\}$, and $\mathsf{nc}(m^S, \langle v_c, M\rangle)$ for some c. From this, it follows that $\mathsf{nc}(m^S, \left\langle m_1^{S_1}, M\right\rangle)$. Therefore, by Lemma 19, we have $m_1^{S_1} \in G$. So, from the definition of $\mathsf{gc}(G, \langle e, M\rangle)$, we have $\neg\mathsf{root}(m_1^{S_1}, e)$, a contradiction.

□

Lemma 21. *Let $C \subseteq \mathsf{dom}(M)$ be a set of locations that are collectible in a configuration $\langle e, M\rangle$:*

$$\forall m^S \in C.\ \neg\mathsf{nc}(m^S, \langle e, M\rangle).$$

Then there is a collectible group that contains C. In particular, let G be the largest superset of C such that from every location in G, some location in C is reachable through a chain of hard references:

$$\forall m_0^{S_0} \in G.\ \exists m_1^{S_1} \in C.\ \mathsf{nc}(m_1^{S_1}, \left\langle m_0^{S_0}, M\right\rangle). \tag{13}$$

Then G is a collectible group: $\mathsf{gc}(G, \langle e, M\rangle)$.

Proof. Suppose G is not a collectible group. Then either G contains a GC root in e, or there is a location outside G with a hard reference into G.

Suppose G contains a GC root m^S in e: $m^S \in G \wedge \mathsf{root}(m^S, e)$. By construction of G, let $m_1^{S_1} \in C$ be a location reachable through a chain of hard references from m^S: $\mathsf{nc}(m_1^{S_1}, \langle m^S, M\rangle)$. If $M(m^S) = \bot$, then from the derivation of $\mathsf{nc}(m_1^{S_1}, \langle m^S, M\rangle)$, we must have $m^S = m_1^{S_1}$, and so from $\mathsf{root}(m^S, e)$, we know $\mathsf{nc}(m_1^{S_1}, \langle e, M\rangle)$, a contradiction. Otherwise, assume $M(m^S) \neq \bot$ and $m^S \neq m_1^{S_1}$. Let $\overrightarrow{v_i}$ be such that $M(m^S) = \{\overrightarrow{x_i = v_i}\}$. From the derivation of $\mathsf{nc}(m_1^{S_1}, \langle m^S, M\rangle)$, we know there exists a c such that $\mathsf{nc}(m_1^{S_1}, \langle v_c, M\rangle)$. Therefore, by NC2, we have $\mathsf{nc}(m_1^{S_1}, \langle e, M\rangle)$, a contradiction.

Otherwise, let $m^S \notin G$ be such that $M(m^S)$ has a hard reference to some $m_0^{S_0} \in G$: $\mathsf{root}(m_0^{S_0}, M(m^S))$. By construction of G, let $m_1^{S_1} \in C$ be a location reachable through a chain of hard references from $m_0^{S_0}$: $\mathsf{nc}(m_1^{S_1}, \left\langle m_0^{S_0}, M\right\rangle)$.

We presently show that $\mathsf{nc}(m_1^{S_1}, \langle M(m^S), M\rangle)$. If $M(m_0^{S_0}) = \bot$, then from the derivation of $\mathsf{nc}(m_1^{S_1}, \left\langle m_0^{S_0}, M\right\rangle)$, we must have $m_0^{S_0} = m_1^{S_1}$, and so from $\mathsf{root}(m_0^{S_0}, M(m^S))$, we know $\mathsf{nc}(m_1^{S_1}, \langle M(m^S), M\rangle)$. Otherwise, assume $M(m_0^{S_0}) \neq \bot$ and $m_0^{S_0} \neq m_1^{S_1}$. Let $\overrightarrow{v_i}$ be such that $M(m_0^{S_0}) = \{\overrightarrow{x_i = v_i}\}$. From the derivation of $\mathsf{nc}(m_1^{S_1}, \left\langle m_0^{S_0}, M\right\rangle)$, we know there exists a c such that $\mathsf{nc}(m_1^{S_1}, \langle v_c, M\rangle)$. Therefore, by NC2, we have $\mathsf{nc}(m_1^{S_1}, \langle M(m^S), M\rangle)$.

So, we know $\mathsf{nc}(m_1^{S_1}, \langle M(m^S), M\rangle)$. It therefore follows that $\mathsf{nc}(m_1^{S_1}, \langle m^S, M\rangle)$. So, $G \cup \{m^S\}$ is a set larger than G satisfying property (13), a contradiction. □

Lemma 22. *Let e_1 and e_2 be well-typed related expressions, and suppose $\phi(m^S)$ is a high-authority GC root of e_2. Then m^S is a GC root of e_1.*

$$\begin{aligned}
& e_1 \approx_\alpha^\phi e_2 \wedge \Gamma; pc; \mathcal{H} \vdash e_1 : \tau, \mathcal{X} \wedge \Gamma; pc; \mathcal{H} \vdash e_2 : \tau, \mathcal{X} \\
& \wedge \vdash \alpha \preccurlyeq \mathsf{auth}^+(S) \wedge \mathsf{root}(\phi(m^S), e_2) \\
& \quad \Rightarrow \mathsf{root}(m^S, e_1)
\end{aligned}$$

Proof. By induction on the derivation of $\mathsf{root}(\phi(m^S), e_2)$. Since e_2 is well-typed, by Lemma 15, we know that case R13 holds vacuously. □

Lemma 23. *Suppose M_1 and M_2 are well-formed memories. Let ϕ be a high-integrity homomorphism such that $M_1 \approx_\alpha^\phi M_2$. Let m^S and $m_1^{S_1}$ be locations in M_1 mapped by ϕ. Assume $m_1^{S_1}$ is high-authority and high-persistence. If* $\mathsf{nc}(\phi(m_1^{S_1}), \langle \phi(m^S), M_2\rangle)$, *then* $\mathsf{nc}(m_1^{S_1}, \langle m^S, M_1\rangle)$*:*

$$\begin{aligned}
& \vdash_{[wf]}^{\alpha} M_1 \wedge \vdash_{[wf]}^{\alpha} M_2 \wedge M_1 \approx_\alpha^\phi M_2 \\
& \wedge \vdash \alpha \preccurlyeq \mathsf{auth}^+(S_1) \sqcap \mathsf{persist}(S_1) \wedge \mathsf{nc}(\phi(m_1^{S_1}), \langle \phi(m^S), M_2\rangle) \\
& \quad \Rightarrow \mathsf{nc}(m_1^{S_1}, \langle m^S, M_1\rangle).
\end{aligned}$$

Proof. By induction on the derivation of $\mathsf{nc}(\phi(m_1^{S_1}), \langle \phi(m^S), M_2 \rangle)$.

Case NC1:

We must have $\phi(m^S) = \phi(m_1^{S_1})$. Since ϕ is injective, we know $m^S = m_1^{S_1}$, so the result follows by NC1.

Case NC2:

We know there exists some $\overrightarrow{u_i}$ and some c such that $M_2(\phi(m^S)) = \{\overrightarrow{x_i = u_i}\}$ and $\mathsf{nc}(\phi(m_1^{S_1}), \langle u_c, M_2 \rangle)$. Assume $\phi(m^S) \neq \phi(m_1^{S_1})$. (Otherwise, the argument for Case NC1 applies.) Since $\phi(m_1^{S_1})$ is high-authority and $\mathsf{nc}(\phi(m_1^{S_1}), \langle \phi(m^S), M_2 \rangle)$, by Corollary 18, we know that m^S is high-integrity, high-authority and high-persistence. Since $M_1 \approx_\alpha^\phi M_2$ and $M_2(\phi(m^S)) \neq \bot$, then we must have $M_1(m^S) \neq \bot$. So let $\overrightarrow{v_i}$ be such that $M_1(m^S) = \{\overrightarrow{x_i = v_i}\}$.

We show that $\mathsf{nc}(m_1^{S_1}, \langle v_c, M_1 \rangle)$. From this, the result follows via an application of NC2:

$$\frac{\mathsf{root}(m^S, m^S) \qquad M_1(m^S) = \{\overrightarrow{x_i = v_i}\} \qquad \mathsf{nc}(m_1^{S_1}, \langle v_c, M_1 \rangle)}{\mathsf{nc}(m_1^{S_1}, \langle m^S, M_1 \rangle).}$$

Consider the two cases in the derivation of $\mathsf{nc}(\phi(m_1^{S_1}), \langle u_c, M_2 \rangle)$:

Sub-case NC1:

We have $\mathsf{root}(\phi(m_1^{S_1}), u_c)$. From $M_1 \approx_\alpha^\phi M_2$, we know $v_c \approx_\alpha^\phi u_c$. From $\vdash_{[wf]}^\alpha M_1$ and $\vdash_{[wf]}^\alpha M_2$, we know $\varnothing; \top; \top \vdash v_c : \tau_c, \top$ and $\varnothing; \top; \top \vdash u_c : \tau_c, \top$, for some τ_c. Therefore, we can apply Lemma 22 to get $\mathsf{root}(m_1^{S_1}, v_c)$, and the result follows via NC1.

Sub-case NC2:

Assume $\neg\mathsf{root}(\phi(m_1^{S_1}), u_c)$. (Otherwise, the argument in sub-case NC1 applies.)

We know there exists some $m_2^{S_2} \in \mathsf{dom}(M_2)$, some $\overrightarrow{u_i}'$ and some c' such that $\mathsf{root}(m_2^{S_2}, u_c)$, $M_2(m_2^{S_2}) = \{\overrightarrow{x_i = u_i'}\}$, and $\mathsf{nc}(\phi(m_1^{S_1}), \langle u'_{c'}, M_2 \rangle)$. From this, it follows that $\mathsf{nc}(\phi(m_1^{S_1}), \langle m_2^{S_2}, M_2 \rangle)$. Since $\phi(m_1^{S_1})$ is high-authority and $\mathsf{nc}(\phi(m_1^{S_1}), \langle m_2^{S_2}, M_2 \rangle)$, by Corollary 18, we know that $m_2^{S_2}$ is high-integrity, high-authority, and high-persistence.

Since ϕ is a high-integrity homomorphism, there exists an $m_3^{S_3}$ such that $m_2^{S_2} = \phi(m_3^{S_3})$. Therefore, we can apply the induction hypothesis to get $\mathsf{nc}(m_1^{S_1}, \langle m_3^{S_3}, M_1 \rangle)$.

We have $\mathsf{root}(\phi(m_3^{S_3}), u_c)$. From $M_1 \approx_\alpha^\phi M_2$, we know $v_c \approx_\alpha^\phi u_c$. From $\vdash_{[wf]}^\alpha M_1$ and $\vdash_{[wf]}^\alpha M_2$, we know $\varnothing; \top; \top \vdash v_c : \tau_c, \top$ and $\varnothing; \top; \top \vdash u_c : \tau_c, \top$, for some τ_c. Therefore, we can apply Lemma 22 to get $\mathsf{root}(m_3^{S_3}, v_c)$. Since $m_2^{S_2} = \phi(m_3^{S_3})$, from $\neg\mathsf{root}(\phi(m_1^{S_1}), u_c)$ and $\mathsf{root}(m_2^{S_2}, u_c)$, we know $m_3^{S_3} \neq m_1^{S_1}$. Therefore, from the derivation of $\mathsf{nc}(m_1^{S_1}, \langle m_3^{S_3}, M_1 \rangle)$, we know there exists some $\overrightarrow{v_i}'$ and some c'' such that $M_1(m_3^{S_3}) = \{\overrightarrow{x_i = v_i'}\}$ and $\mathsf{nc}(m_1^{S_1}, \langle v'_{c''}, M_1 \rangle)$.

The result follows via NC2:

$$\frac{\mathsf{root}(m_3^{S_3}, v_c) \qquad M_1(m_3^{S_3}) = \{\overrightarrow{x_i = v_i'}\} \qquad \mathsf{nc}(m_1^{S_1}, \langle v'_{c''}, M_1 \rangle)}{\mathsf{nc}(m_1^{S_1}, \langle v_c, M_1 \rangle)}$$

□

Lemma 24. *Let G be a collectible group in $\langle e, M \rangle$ with $m_1^{S_1} \in G$. If $\mathsf{nc}(m_1^{S_1}, \langle m^S, M \rangle)$, then $m^S \in G$:*

$$\mathsf{gc}(G, \langle e, M \rangle) \wedge m_1^{S_1} \in G \wedge \mathsf{nc}(m_1^{S_1}, \langle m^S, M \rangle) \Rightarrow m^S \in G$$

Proof. By induction on the derivation of $\mathsf{nc}(m_1^{S_1}, \langle m^S, M \rangle)$.

Case NC1:

We must have $m^S = m_1^{S_1}$, so the result follows trivially.

Case NC2:

We know there exists some $\overrightarrow{v_i}$ and some c such that $M(m^S) = \{\overrightarrow{x_i = v_i}\}$ and $\mathsf{nc}(m_1^{S_1}, \langle v_c, M\rangle)$. Consider the two cases in the derivation of $\mathsf{nc}(m_1^{S_1}, \langle v_c, M\rangle)$:

Sub-case NC1:

We have $\mathsf{root}(m_1^{S_1}, v_c)$, so we therefore know $\mathsf{root}(m_1^{S_1}, M(m^S))$. The result then follows from the definition of $\mathsf{gc}(G, \langle e, M\rangle)$.

Sub-case NC2:

We know there exists some location $m_2^{S_2}$ such that $\mathsf{root}(m_2^{S_2}, v_c)$ and $\mathsf{nc}(m_1^{S_1}, \left\langle m_2^{S_2}, M\right\rangle)$. So, by the induction hypothesis, we know $m_2^{S_2} \in G$. From $\mathsf{root}(m_2^{S_2}, v_c)$, we know $\mathsf{root}(m_2^{S_2}, M(m^S))$. The result then follows from the definition of $\mathsf{gc}(G, \langle e, M\rangle)$.

□

Lemma 25. *Let M_1 and M_2 be well-formed memories. Suppose $\langle e_1, M_1\rangle \approx_\alpha^\phi \langle e_2, M_2\rangle$. Let G be a collectible group in $\langle e_1, M_1\rangle$, and let $\phi(G)$ denote $\{\phi(m^S) : m^S \in G \cap \mathsf{dom}(\phi)\}$. Let C represent the high-authority, high-persistence members of $G \cap \mathsf{dom}(\phi)$:*

$$C = \{m^S \in G \cap \mathsf{dom}(\phi) : \vdash \alpha \preccurlyeq \mathsf{auth}^+(S) \sqcap \mathsf{persist}(S)\}.$$

Then there exists a set G' such that $\phi(G')$ is a subset of $\phi(G)$, is a collectible group in $\langle e_2, M_2\rangle$, and contains all members of $\phi(C)$:

$$\begin{aligned}&\vdash_{[wf]}^{\alpha} M_1 \wedge \vdash_{[wf]}^{\alpha} M_2 \wedge \langle e_1, M_1\rangle \approx_\alpha^\phi \langle e_2, M_2\rangle \wedge \mathsf{gc}(G, \langle e_1, M_1\rangle)\\ &\quad \Rightarrow \exists G'.\, \mathsf{gc}(\phi(G'), \langle e_2, M_2\rangle) \wedge \phi(C) \subseteq \phi(G') \subseteq \phi(G)\end{aligned}$$

Proof. First, we show that $\phi(C)$ is a set of collectible locations. Suppose it isn't. Then let $m^S \in G \cap \mathsf{dom}(\phi)$ be such that $\phi(m^S) \in \phi(C)$ is noncollectible: $\mathsf{nc}(\phi(m^S), \langle e_2, M_2\rangle)$. Since $\phi(m^S)$ is high-authority and high-persistence, by induction on the derivation of $\mathsf{nc}(\phi(m^S), \langle e_2, M_2\rangle)$, we can show that m^S must be noncollectible: $\mathsf{nc}(m^S, \langle e_1, M_1\rangle)$. Therefore, by Lemma 20, G cannot be a collectible group, a contradiction.

Let $G' \subseteq \mathsf{dom}(\phi)$ be such that $\phi(G')$ is the largest superset of $\phi(C)$ such that from every location in $\phi(G')$, some location in $\phi(C)$ is reachable through a chain of hard references: $\forall m_0^{S_0} \in \phi(G').\ \exists m_1^{S_1} \in \phi(C).\ \mathsf{nc}(m_1^{S_1}, \left\langle m_0^{S_0}, M_2\right\rangle)$. By Lemma 21, we know $\phi(G')$ is a collectible group.

We now show that $\phi(G')$ is also a subset of $\phi(G)$ by showing $G' \subseteq G$. Suppose $m^S \in G'$. By construction of G', let $m_1^{S_1} \in C$ be such that $\mathsf{nc}(\phi(m_1^{S_1}), \langle \phi(m^S), M_2\rangle)$. From this, by Lemma 23, we know $\mathsf{nc}(m_1^{S_1}, \langle m^S, M_1\rangle)$. By Lemma 24, then, we must have $m^S \in G$. So $G' \subseteq G$. □

Lemmas 26–29 prove a few results about the preservation of the security relation under various conditions

Lemma 26 (Relation substitution)**.** *If $e_1 \approx_\alpha^\phi e_2$ and $e_1' \approx_\alpha^\phi e_2'$, then $e_1\{e_1'/x\} \approx_\alpha^\phi e_2\{e_2'/x\}$.*

Proof. By induction on the derivation of $e_1 \approx_\alpha^\phi e_2$. □

Lemma 27 (Security-relation preservation under auto-bracketing)**.** *Auto-bracketing preserves the security relation:*

$$e_1 \approx_\alpha^\phi e_2 \Rightarrow e_1 \blacktriangleright_\alpha \tau \approx_\alpha^\phi e_2 \blacktriangleright_\alpha \tau.$$

Proof. By induction on the derivation of $e_1 \approx_\alpha^\phi e_2$. □

Lemma 28. *Suppose $e_1 \approx_\alpha^\phi e_2$. Let m^S be such that $m^S \notin \mathsf{locs}(e_1)$ and let $\phi' = \phi[m^S \mapsto m_1^S]$. Then $e_1 \approx_\alpha^{\phi'} e_2$.*

Proof. By induction on the derivation of $e_1 \approx_\alpha^\phi e_2$. □

Lemma 29. *Evaluating in a low-integrity context preserves the security relation on memories.*

Let $\langle e_1, M_1\rangle$ be a well-formed configuration, wherein e_1 is well-typed in a low-integrity context. Let M_2 be a well-formed memory and suppose $\langle e_1, M_1\rangle \xrightarrow{\mathsf{e}}{}^{*} \langle e_1', M_1'\rangle$.

$$\begin{aligned}&\vdash^{\alpha}_{[wf]} \langle e_1, M_1\rangle \wedge \alpha \not\leqslant pc \wedge \varnothing;pc;\mathcal{H} \vdash e_1 : \tau, \mathcal{X}\\ &\wedge \vdash^{\alpha}_{[wf]} M_2 \wedge \langle e_1, M_1\rangle \xrightarrow{\mathsf{e}}{}^{*} \left\langle e_1', M_1'\right\rangle.\end{aligned}$$

Then the following holds.

1. $M_1 \approx^{\phi}_{\alpha} M_2 \Rightarrow M_1' \approx^{\phi}_{\alpha} M_2$ *and*
2. $\vdash_{[wf]} \langle e_1, M_1\rangle \wedge M_2 \approx^{\phi}_{\alpha} M_1 \Rightarrow M_2 \approx^{\phi}_{\alpha} M_1'$.

Proof. If we can show this is true for the case where a single $\xrightarrow{\mathsf{e}}$ step is taken, then the rest follows by induction on the number of $\xrightarrow{\mathsf{e}}$ steps taken. (We know the induction hypothesis will apply because of Corollary 11 and Corollary 12.)

We show the single-step case by induction on the derivation of $\langle e_1, M_1\rangle \xrightarrow{\mathsf{e}} \langle e_1', M_1'\rangle$. The proof proceeds by cases according to the evaluation rules.

In cases SELECT, DANGLE-SELECT, SOFT-SELECT, DANGLE-ASSIGN, APPLY, EXISTS-TRUE, EXISTS-FALSE, TRY-VAL, TRY-CATCH, TRY-ESC, PARALLEL-RESULT, IF-TRUE, IF-FALSE, LET, FAIL-PROP, BRACKET-SELECT, BRACKET-SOFT-SELECT, BRACKET-ASSIGN, BRACKET-SOFT-ASSIGN, BRACKET-SOFT, BRACKET-EXISTS, BRACKET-APPLY, BRACKET-TRY, BRACKET-IF, BRACKET-LET, DOUBLE-BRACKET, and BRACKET-FAIL, the result holds trivially, since $M_1' = M_1$.

Case CREATE $\left(\left\langle\{\overrightarrow{x_i = v_i}\}^S, M_1\right\rangle \xrightarrow{\mathsf{e}} \left\langle m^S, M_1\left[m^S \mapsto \{\overrightarrow{x_i = v_i \blacktriangleright_{\alpha} \tau_i}\}\right]\right\rangle\right.$, where m is fresh and $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$):

1. Suppose $M_1 \approx^{\phi}_{\alpha} M_2$. We show that $M_1' \approx^{\phi}_{\alpha} M_2$.
   From the derivation of $\varnothing;pc;\mathcal{H} \vdash e_1 : \tau, \mathcal{X}$, we know $\vdash p \leqslant pc$ and $\vdash \mathsf{integ}(\tau_i) \leqslant pc$ for all i. Therefore, we know m^S is neither high-integrity nor high-persistence. The result then follows from the fact that $m^S \notin \mathsf{dom}(\phi)$ and the assumption $M_1 \approx^{\phi}_{\alpha} M_2$.
2. Suppose $\vdash_{[wf]} \langle e_1, M_1\rangle$ and $M_2 \approx^{\phi}_{\alpha} M_1$. We show that $M_2 \approx^{\phi}_{\alpha} M_1'$.
   This follows from $M_2 \approx^{\phi}_{\alpha} M_1$.

Case ASSIGN $\left(\left\langle m^S.x_c := v, M_1\right\rangle \xrightarrow{\mathsf{e}} \left\langle * \blacktriangleright_{\alpha} p, M_1[m^S.x_c \mapsto v \blacktriangleright_{\alpha} \tau_c]\right\rangle\right.$, where $M_1(m^S) \neq \bot$ and $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$):

Let $\overrightarrow{v_i}'$ and $\overrightarrow{u_i}$ be such that $M_1'(m^S) = \{\overrightarrow{x_i = v_i'}\}$ and $M_2(\phi(m^S)) = \{\overrightarrow{x_i = u_i}\}$. We therefore have $v_c' = v \blacktriangleright_{\alpha} \tau_c$.

From $\varnothing;pc;\mathcal{H} \vdash m^S.x_c := v : \tau, \mathcal{X}$, we know $\vdash \tau' \sqcap pc \leq \tau_c$, for some τ'. We therefore know $\vdash \mathsf{integ}(\tau_c) \leqslant pc$. So, from $\alpha \not\leqslant pc$, we have $\alpha \not\leqslant \mathsf{integ}(\tau_c)$. Therefore, v_c' is a bracketed value: there exists a v_c'' such that $v_c' = [v_c'']$.

From $\vdash^{\alpha}_{[wf]} M_2$ and $\alpha \not\leqslant \mathsf{integ}(\tau_c)$, we also know u_c is a bracketed value: $u_c = [u_c']$, for some u_c'.

1. Suppose $M_1 \approx^{\phi}_{\alpha} M_2$. We show that $M_1' \approx^{\phi}_{\alpha} M_2$.
   Assume $m^S \in \mathsf{dom}(\phi)$ (otherwise, the result follows directly from $M_1 \approx^{\phi}_{\alpha} M_2$). Since $M_1 \approx^{\phi}_{\alpha} M_2$, it suffices to show $v_c' \approx^{\phi}_{\alpha} u_c$. This is trivial, since $v_c' = [v_c'']$ and $u_c = [u_c']$.
2. Suppose $\vdash_{[wf]} \langle e_1, M_1\rangle$ and $M_2 \approx^{\phi}_{\alpha} M_1$. We show that $M_2 \approx^{\phi}_{\alpha} M_1'$.
   Assume $m^S \in \mathsf{im}(\phi)$ (otherwise, the result follows directly from $M_2 \approx^{\phi}_{\alpha} M_1$). Since $M_2 \approx^{\phi}_{\alpha} M_1$, it suffices to show $u_c \approx^{\phi}_{\alpha} v_c'$. This is trivial, since $u_c = [u_c']$ and $v_c' = [v_c'']$.

Case SOFT-ASSIGN $\left(\left\langle(\mathsf{soft}\ m^S).x_c := v, M_1\right\rangle \xrightarrow{\mathsf{e}} \langle e_1' \blacktriangleright_{\alpha} (a \sqcap p), M_1'\rangle\right.$, where $\left\langle m^S.x_c := v, M_1\right\rangle \xrightarrow{\mathsf{e}} \langle e_1', M_1'\rangle$ and $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$):

We proceed by cases according to the evaluation rules for $\left\langle m^S.x_c := v, M_1\right\rangle \xrightarrow{\mathsf{e}} \langle e_1', M_1'\rangle$.

Sub-case ASSIGN ($\langle m^S.x_c := v, M_1\rangle \xrightarrow{e} \langle * \blacktriangleright_\alpha p, M_1[m^S.x_c \mapsto v \blacktriangleright_\alpha \tau_c]\rangle$):

Let $\overrightarrow{v_i}'$ and $\overrightarrow{u_i}$ be such that $M_1'(m^S) = \{\overrightarrow{x_i = v_i'}\}$ and $M_2(\phi(m^S)) = \{\overrightarrow{x_i = u_i}\}$. We therefore have $v_c' = v \blacktriangleright_\alpha \tau_c$. From $\varnothing;pc;\mathcal{H} \vdash (\mathsf{soft}\, m^S).x_c := v : \tau, \mathcal{X}$, we know $\vdash \tau' \sqcap pc \le \tau_c$, for some τ'. We therefore know $\vdash \mathsf{integ}(\tau_c) \preccurlyeq pc$. So, from $\alpha \not\preccurlyeq pc$, we have $\alpha \not\preccurlyeq \mathsf{integ}(\tau_c)$. Therefore, v_c' is a bracketed value: there exists a v_c'' such that $v_c' = [v_c'']$.

From $\vdash^\alpha_{[wf]} M_2$ and $\alpha \not\preccurlyeq \mathsf{integ}(\tau_c)$, we also know u_c is a bracketed value: $u_c = [u_c']$, for some u_c'.

1. Suppose $M_1 \approx^\phi_\alpha M_2$. We show $M_1' \approx^\phi_\alpha M_2$.
   Assume $m^S \in \mathsf{dom}(\phi)$ (otherwise, the result follows directly from $M_1 \approx^\phi_\alpha M_2$). Since $M_1 \approx^\phi_\alpha M_2$, it suffices to show $v_c' \approx^\phi_\alpha u_c$. This is trivial, since $v_c' = [v_c'']$ and $u_c = [u_c']$.
2. Suppose $\vdash_{[wf]} \langle e_1, M_1\rangle$ and $M_2 \approx^\phi_\alpha M_1$. We show $M_2 \approx^\phi_\alpha M_1'$.
   Assume $m^S \in \mathsf{im}(\phi)$ (otherwise, the result follows directly from $M_2 \approx^\phi_\alpha M_1$). Since $M_2 \approx^\phi_\alpha M_1$, it suffices to show $u_c \approx^\phi_\alpha v_c'$. This is trivial, since $u_c = [u_c']$ and $v_c' = [v_c'']$.

Sub-case DANGLE-ASSIGN ($\langle m^S.x_c := v, M_1\rangle \xrightarrow{e} \langle \bot_p \blacktriangleright_\alpha p, M_1\rangle$):

The result holds trivially, since $M_1' = M_1$.

Case EVAL-CONTEXT ($\langle E[e_3], M_1\rangle \xrightarrow{e} \langle E[e_3'], M_1'\rangle$, where $\langle e_3, M_1\rangle \xrightarrow{e} \langle e_3', M_1'\rangle$):

A case analysis on the syntax of $E[\cdot]$ shows that from the derivation of $\varnothing;pc;\mathcal{H} \vdash E[e_3] : \tau, \mathcal{X}$, we know $\varnothing;pc;\mathcal{H}' \vdash e_3 : \tau', \mathcal{X}'$, for some $\mathcal{H}'$, τ', and $\mathcal{X}'$. Therefore, we can apply the induction hypothesis and obtain the result.

Case BRACKET-CONTEXT ($\langle [e_3], M_1\rangle \xrightarrow{e} \langle [e_3'], M_1'\rangle$, where $\langle e_3, M_1\rangle \xrightarrow{e} \langle e_3', M_1'\rangle$):

From the derivation of $\varnothing;pc;\mathcal{H} \vdash [e_3] : \tau, \mathcal{X}$, we know $\varnothing;pc \sqcap \ell;\mathcal{H} \vdash e_3 : \tau', \mathcal{X}$, where $\alpha \not\preccurlyeq \ell$ and $\tau = \tau' \sqcap \ell$. Therefore, we can apply the induction hypothesis and obtain the result.

□

We can now prove Theorem 1, restated below for convenience.

Theorem 1 (Referential integrity)**.** *Assume $\langle e_1, M_1\rangle$ is a well-formed configuration and $\langle e_2, M_2\rangle$ is a well-formed, nonadversarial, ϕ-related configuration, such that e_1 and e_2 have type τ and M_2 is well-formed:*

$$\vdash^\alpha_{[wf]} \langle e_1, M_1\rangle \;\wedge\; \vdash_{[wf]} \langle e_2, M_2\rangle \;\wedge\; \langle e_1, M_1\rangle \approx^\phi_\alpha \langle e_2, M_2\rangle$$
$$\wedge\; \varnothing;pc;\mathcal{H} \vdash e_1 : \tau, \mathcal{X} \;\wedge\; \varnothing;pc;\mathcal{H} \vdash e_2 : \tau, \mathcal{X} \;\wedge\; \vdash^\alpha_{[wf]} M_2$$

Suppose $\langle e_1, M_1\rangle$ takes some number of steps in the presence of an adversary to another configuration $\langle e_1', M_1'\rangle$. Then either $\langle e_2, M_2\rangle$ diverges, or it can take some number of steps in the absence of an adversary to another configuration $\langle e_2', M_2'\rangle$ and there exists a high-integrity homomorphism ϕ' from M_1' to M_2' that extends ϕ, such that $\langle e_1', M_1'\rangle$ is related to $\langle e_2', M_2'\rangle$ via ϕ':

$$\langle e_1, M_1\rangle \to^*_\alpha \langle e_1', M_1'\rangle \wedge \neg \langle e_2, M_2\rangle \Uparrow$$
$$\Rightarrow \exists e_2', M_2', \phi'.\ \langle e_2, M_2\rangle \to^* \langle e_2', M_2'\rangle \wedge \langle e_1', M_1'\rangle \approx^{\phi'}_\alpha \langle e_2', M_2'\rangle \wedge \phi = \phi'|_{\mathsf{dom}(\phi)}$$

Proof. If we can show this is true for the case where a single $\to_\alpha$ step is taken to reach $\langle e_1', M_1'\rangle$, then the rest follows by induction on the number of $\to_\alpha$ steps taken. (We know the induction hypothesis will apply because of Corollary 11 and the fact that $\vdash_{[wf]} \langle e_2, M_2\rangle \wedge \vdash^\alpha_{[wf]} M_2 \Rightarrow \vdash^\alpha_{[wf]} \langle e_2, M_2\rangle$.)

We show the single-step case by induction on the derivation of $\langle e_1, M_1\rangle \to_\alpha \langle e_1', M_1'\rangle$. The proof proceeds by cases according to the evaluation rules. For each case, we need to show two things about e_2', M_2', and ϕ':

i. Related expressions: $e_1' \approx^{\phi'}_\alpha e_2'$ and

ii. Related memories: $M_1' \approx^{\phi'}_\alpha M_2'$.

It will be obvious by its construction that ϕ' is a high-integrity homomorphism that extends ϕ.

Case CREATE ($\left\langle \{\overrightarrow{x_i = v_i}\}^S, M_1 \right\rangle \to_\alpha \left\langle m_1^S, M_1[m_1^S \mapsto \{\overrightarrow{x_i = v_i \blacktriangleright_\alpha \tau_i}\}] \right\rangle$, where m_1 is fresh and $S = \{\overrightarrow{x_i : \tau_i}\}_s$):

From $e_1 \approx_\alpha^\phi e_2$, we know $e_2 = \{\overrightarrow{x_i = u_i}\}^S$ with $v_i \approx_\alpha^\phi u_i$ for all i. By CREATE,

$$\langle e_2, M_2 \rangle \to \left\langle m_2^S, M_2[m_2^S \mapsto \{\overrightarrow{x_i = u_i \blacktriangleright_\alpha \tau_i}\}] \right\rangle,$$

where m_2 is fresh. Choose $\phi' = \phi[m_1^S \mapsto m_2^S]$.

i. We need to show $m_1^S \approx_\alpha^{\phi'} m_2^S$.
This follows by construction of ϕ'.

ii. We need to show $M_1' \approx_\alpha^{\phi'} M_2'$, where $M_1' = M_1[m_1^S \mapsto \{\overrightarrow{x_i = v_i \blacktriangleright_\alpha \tau_i}\}]$ and $M_2' = M_2[m_2^S \mapsto \{\overrightarrow{x_i = u_i \blacktriangleright_\alpha \tau_i}\}]$.
First, let $m^{S'} \in \mathsf{dom}(\phi')$ be such that $M_1'(m^{S'}) \neq \bot$. We show that $M_2'(\phi'(m^{S'})) \neq \bot$ and $M_1'(m^{S'}) \approx_\alpha^{\phi'} M_2'(\phi'(m^{S'}))$.
If $m^{S'} = m_1^S$, then $M_1'(m^{S'}) = M_1'(m_1^S) = \{\overrightarrow{x_i = v_i \blacktriangleright_\alpha \tau_i}\}$ and $M_2'(\phi'(m^{S'})) = M_2'(m_2^S) = \{\overrightarrow{x_i = u_i \blacktriangleright_\alpha \tau_i}\}$, so the result follows from the assumption $e_1 \approx_\alpha^\phi e_2$ via Lemmas 27 and 28. Otherwise, $m^{S'} \neq m_1^S$, so $M_1'(m^{S'}) = M_1(m^{S'})$ and $M_2'(\phi'(m^{S'})) = M_2(\phi(m^{S'}))$. The result therefore follows from the assumption $M_1 \approx_\alpha^\phi M_2$.
Now, let $m^{S'} \in \mathsf{dom}(\phi')$ be such that $\vdash \alpha \preccurlyeq \mathsf{auth}^+(S') \sqcap \mathsf{persist}(S')$ and $M_1'(m^{S'}) = \bot$. We show that $M_2'(\phi'(m^{S'})) = \bot$. Since $M_1'(m^{S'}) = \bot$, we must have $m^{S'} \neq m_1^S$, so $M_1'(m^{S'}) = M_1(m^{S'})$ and $M_2'(\phi'(m^{S'})) = M_2(\phi(m^{S'}))$. The result therefore follows from the assumption $M_1 \approx_\alpha^\phi M_2$.

Case SELECT ($\left\langle m_1^S.x_c, M_1 \right\rangle \to_\alpha \langle v_c \blacktriangleright_\alpha p, M_1 \rangle$, where $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$ and $M_1(m_1^S) = \{\overrightarrow{x_i = v_i}\}$):

From $e_1 \approx_\alpha^\phi e_2$, we know $e_2 = m_2^S.x_c$, where $\phi(m_1^S) = m_2^S$. Therefore, from $M_1 \approx_\alpha^\phi M_2$, we have $M_2(m_2^S) = \{\overrightarrow{x_i = u_i}\}$ for some $\overrightarrow{u_i}$, where $v_i \approx_\alpha^\phi u_i$. So, by SELECT, $\langle e_2, M_2 \rangle \to \langle u_c \blacktriangleright_\alpha p, M_2 \rangle$. Choose $\phi' = \phi$.

i. We need to show $v_c \blacktriangleright_\alpha p \approx_\alpha^\phi u_c \blacktriangleright_\alpha p$.
This follows via Lemma 27.

ii. We need to show $M_1 \approx_\alpha^\phi M_2$.
This is given.

Case DANGLE-SELECT ($\left\langle m_1^S.x_c, M_1 \right\rangle \to_\alpha \left\langle \bot_p \blacktriangleright_\alpha p, M_1 \right\rangle$, where $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$ and $M_1(m_1^S) = \bot$):

From $\vdash_{[wf]}^\alpha \left\langle m_1^S.x_c, M_1 \right\rangle$, $\mathsf{nc}(m_1^S, \left\langle m_1^S.x_c, M_1 \right\rangle)$, and $M_1(m_1^S) = \bot$, we know $\alpha \not\preccurlyeq p$. Therefore, $\bot_p \blacktriangleright_\alpha p = [\bot_p]$. From $e_1 \approx_\alpha^\phi e_2$, we know $e_2 = m_2^S.x_c$.

If $M_2(m_2^S) = \bot$, then by DANGLE-SELECT, $\langle e_2, M_2 \rangle \to \left\langle [\bot_p], M_2 \right\rangle$.

Otherwise, without loss of generality, assume $M_2(m_2^S) = \{\overrightarrow{x_i = u_i}\}$. Since $\alpha \not\preccurlyeq p$, we have $u_c \blacktriangleright_\alpha p = [u_c']$ for some u_c'. So, by SELECT, $\langle e_2, M_2 \rangle \to \langle [u_c'], M_2 \rangle$.

Choose $\phi' = \phi$.

i. We need to show that $[\bot_p] \approx_\alpha^\phi [u]$ for $u \in \{u_c', \bot_p\}$.
This is trivial.

ii. We need to show that $M_1 \approx_\alpha^\phi M_2$.
This is given.

Case SOFT-SELECT ($\left\langle (\mathsf{soft}\, m_1^S).x_c, M_1 \right\rangle \to_\alpha \langle v', M_1 \rangle$, where $\left\langle m_1^S.x_c, M_1 \right\rangle \xrightarrow{\mathsf{e}} \langle v, M_1 \rangle$, $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$), and $v' = v \blacktriangleright_\alpha (a \sqcap p)$):

From $e_1 \approx_\alpha^\phi e_2$, we know $e_2 = ((\mathsf{soft}\, m_2^S).x_c)$, where $\phi(m_1^S) = m_2^S$. If we can find u so that $\left\langle m_2^S.x_c, M_2 \right\rangle \xrightarrow{\mathsf{e}} \langle u, M_2 \rangle$, then by SOFT-SELECT, we have $\langle e_2, M_2 \rangle = \left\langle (\mathsf{soft}\, m_2^S).x_c, M_2 \right\rangle \to \langle u \blacktriangleright_\alpha (a \sqcap p), M_2 \rangle$. Choose $\phi' = \phi$.

We proceed by cases according to the evaluation rules for $\left\langle m_1^S.x_c, M_1 \right\rangle \xrightarrow{\mathsf{e}} \langle v, M_1 \rangle$.

Sub-case SELECT ($v = v_c \blacktriangleright_\alpha p$, where $M_1(m_1^S) = \{\overrightarrow{x_i = v_i}\}$):

From $M_1 \approx_\alpha^\phi M_2$, we know $M_2(m_2^S) = \{\overrightarrow{x_i = u_i}\}$ for some $\overrightarrow{u_i}$, where $v_i \approx_\alpha^\phi u_i$. So, by SELECT, we have $\langle m_2^S.x_c, M_2\rangle \xrightarrow{e} \langle u_c \blacktriangleright_\alpha p, M_2\rangle$. Therefore, $u = u_c \blacktriangleright_\alpha p$.

i. We need to show $(v_c \blacktriangleright_\alpha p) \blacktriangleright_\alpha (a \sqcap p) \approx_\alpha^\phi (u_c \blacktriangleright_\alpha p) \blacktriangleright_\alpha (a \sqcap p)$.
This follows via Lemma 27.

ii. We need to show $M_1 \approx_\alpha^\phi M_2$.
This is given.

Sub-case DANGLE-SELECT ($v = \bot_p \blacktriangleright_\alpha p$, where $M_1(m_1^S) = \bot$):

First, suppose $\vdash \alpha \preccurlyeq a \sqcap p$. Then $v \blacktriangleright_\alpha (a \sqcap p) = \bot_p$, $u \blacktriangleright_\alpha (a \sqcap p) = u$, and $\vdash \alpha \preccurlyeq p$. Therefore, we have $m_1^S \in \mathsf{dom}(\phi)$, and so, $\phi(m_1^S) = m_2^S$. From $M_1 \approx_\alpha^\phi M_2$, then, we know $M_2(m_2^S) = \bot$. So, by SELECT, we have $\langle m_2^S.x_c, M_2\rangle \xrightarrow{e} \langle \bot_p, M_2\rangle$. Therefore, $u = \bot_p$.

i. We need to show $\bot_p \approx_\alpha^\phi \bot_p$.
This is trivial.

ii. We need to show $M_1 \approx_\alpha^\phi M_2$.
This is given.

Now, suppose $\alpha \not\preccurlyeq a \sqcap p$. Then $v \blacktriangleright_\alpha (a \sqcap p) = [\bot_p]$. If $M_2(m_2^S) = \bot$, then by DANGLE-SELECT, $\langle m_2^S.x_c, M_2\rangle \xrightarrow{e} \langle \bot_p \blacktriangleright_\alpha p, M_2\rangle$. Otherwise, without loss of generality, assume $M_2(m_2^S) = \{\overrightarrow{x_i = u_i}\}$. So, by SELECT, $\langle m_2^S.x_c, M_2\rangle \xrightarrow{e} \langle u_c \blacktriangleright_\alpha p, M_2\rangle$. Therefore, $u \blacktriangleright_\alpha (a \sqcap p) \in \{(\bot_p \blacktriangleright_\alpha p) \blacktriangleright_\alpha (a \sqcap p), (u_c \blacktriangleright_\alpha p) \blacktriangleright_\alpha (a \sqcap p)\} = \{[\bot_p], [u_c']\}$, for some u_c'.

i. We need to show $[\bot_p] \approx_\alpha^\phi [u']$ for $u' \in \{u_c', \bot_p\}$.
This is trivial.

ii. We need to show $M_1 \approx_\alpha^\phi M_2$.
This is given.

Case ASSIGN ($\langle m_1^S.x_c := v, M_1\rangle \to_\alpha \langle * \blacktriangleright_\alpha p, M_1[m_1^S.x_c \mapsto v \blacktriangleright_\alpha \tau_c]\rangle$, where $M_1(m_1^S) \neq \bot$ and $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$):

From $e_1 \approx_\alpha^\phi e_2$, we know $e_2 = (m_2^S.x_c := u)$ with $\phi(m_1^S) = m_2^S$ and $v \approx_\alpha^\phi u$. From $M_1 \approx_\alpha^\phi M_2$, we have $M_2(m_2^S) \neq \bot$. So, by ASSIGN, $\langle m_2^S.x_c := u, M_2\rangle \to \langle * \blacktriangleright_\alpha p, M_2[m_2^S.x_c \mapsto u \blacktriangleright_\alpha \tau_c]\rangle$. Choose $\phi' = \phi$.

i. We need to show $* \blacktriangleright_\alpha p \approx_\alpha^\phi * \blacktriangleright_\alpha p$.
This follows via Lemma 27.

ii. We need to show $M_1[m_1^S.x_c \mapsto v \blacktriangleright_\alpha \tau_c] \approx_\alpha^\phi M_2[m_2^S.x_c \mapsto u \blacktriangleright_\alpha \tau_c]$.
First, let $m_0^{S_0} \in \mathsf{dom}(\phi)$ be such that $M_1'(m_0^{S_0}) \neq \bot$. We show that $M_2'(\phi(m_0^{S_0})) \neq \bot$ and $M_1'(m_0^{S_0}) \approx_\alpha^\phi M_2'(\phi(m_0^{S_0}))$.
If $m_0^{S_0} = m_1^S$, then $\phi(m_0^{S_0}) = m_2^S$. Let $\overrightarrow{v_i}'$ and $\overrightarrow{u_i}'$ be such that $M_1'(m_1^S) = \{\overrightarrow{x_i = v_i'}\}$ and $M_2'(m_2^S) = \{\overrightarrow{x_i = u_i'}\}$. Since $v \approx_\alpha^\phi u$, by Lemma 27, we have $v \blacktriangleright_\alpha \tau_c \approx_\alpha^\phi u \blacktriangleright_\alpha \tau_c$. Since $v_c' = v \blacktriangleright_\alpha \tau_c$ and $u_c' = u \blacktriangleright_\alpha \tau_c$, we therefore have $v_c' \approx_\alpha^\phi u_c'$. Therefore, from $M_1 \approx_\alpha^\phi M_2$, it follows that $\{\overrightarrow{x_i = v_i'}\} \approx_\alpha^\phi \{\overrightarrow{x_i = u_i'}\}$.
Otherwise, $m_0^{S_0} \neq m_1^S$, so $M_1'(m_0^{S_0}) = M_1(m_0^{S_0})$ and $M_2'(\phi(m_0^{S_0})) = M_2(\phi(m_0^{S_0}))$. The result therefore follows from $M_1 \approx_\alpha^\phi M_2$.
Now, let $m_0^{S_0} \in \mathsf{dom}(\phi)$ be such that $\vdash \alpha \preccurlyeq \mathsf{auth}^+(S_0) \sqcap \mathsf{persist}(S_0)$ and $M_1'(m_0^{S_0}) = \bot$. We show that $M_2'(\phi(m_0^{S_0})) = \bot$. This follows from $M_1 \approx_\alpha^\phi M_2$ by construction of M_1' and M_2'.

Case DANGLE-ASSIGN ($\langle m_1^S.x_c := v, M_1\rangle \to_\alpha \langle \bot_p \blacktriangleright_\alpha p, M_1\rangle$, where $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$ and $M_1(m^S) = \bot$):

From $\vdash_{[wf]}^\alpha \langle m_1^S.x_c := v, M_1\rangle$, $\mathsf{nc}(m_1^S, \langle m_1^S.x_c := v, M_1\rangle)$, and $M_1(m_1^S) = \bot$, we know $\alpha \not\preccurlyeq p$. Therefore, $\bot_p \blacktriangleright_\alpha p = [\bot_p]$.

From $e_1 \approx_\alpha^\phi e_2$, we know $e_2 = m_2^S.x_c := u$ with $\phi(m_1^S) = m_2^S$ and $v \approx_\alpha^\phi u$. If $M_2(m_2^S) = \bot$, then by DANGLE-ASSIGN, $\langle e_2, M_2\rangle \to \langle [\bot_p], M_2\rangle$.

Otherwise, $M_2(m_2^S) \neq \bot$. Since $\alpha \not\leq p$, we have $* \blacktriangleright_\alpha p = [*]$. So, by ASSIGN, $\langle e_2, M_2\rangle \to \big\langle [*], M_2[m_2^S.x_c \mapsto u \blacktriangleright_\alpha \tau_c]\big\rangle$. Choose $\phi' = \phi$.

i. We need to show $[\bot_p] \approx_\alpha^\phi [u']$ for $u' \in \{*, \bot_p\}$.
This is trivial.

ii. We need to show $M_1' \approx_\alpha^\phi M_2'$.
First, let $m_0^{S_0} \in \mathsf{dom}(\phi)$ be such that $M_1'(m_0^{S_0}) \neq \bot$. We show that $M_2'(\phi(m_0^{S_0})) \neq \bot$ and $M_1'(m_0^{S_0}) \approx_\alpha^\phi M_2'(\phi(m_0^{S_0}))$.
Since $M_1' = M_1$ and $M_1'(m_0^{S_0}) \neq \bot$, we must have $m_0^{S_0} \neq m_1^S$, so $M_1'(m_0^{S_0}) = M_1(m_0^{S_0})$ and $M_2'(\phi(m_0^{S_0})) = M_2(\phi(m_0^{S_0}))$. The result therefore follows from $M_1 \approx_\alpha^\phi M_2$.
Now, let $m_0^{S_0} \in \mathsf{dom}(\phi)$ be such that $\vdash \alpha \leqslant \mathsf{auth}^+(S_0) \sqcap \mathsf{persist}(S_0)$ and $M_1'(m_0^{S_0}) = \bot$. We show that $M_2'(\phi(m_0^{S_0})) = \bot$. This follows from $M_1 \approx_\alpha^\phi M_2$ by construction of M_2'.

Case SOFT-ASSIGN ($\big\langle (\mathsf{soft}\ m_1^S).x_c := v, M_1\big\rangle \to_\alpha \langle v' \blacktriangleright_\alpha (a \sqcap p), M_1'\rangle$, where $\big\langle m_1^S.x_c := v, M_1\big\rangle \xrightarrow{\mathsf{e}} \langle v', M_1'\rangle$ and $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$):

From $e_1 \approx_\alpha^\phi e_2$, we know $e_2 = ((\mathsf{soft}\ m_2^S).x_c := u)$ with $\phi(m_1^S) = m_2^S$ and $v \approx_\alpha^\phi u$. If we can find a configuration $\langle u', M_2'\rangle$ so that $\big\langle m_2^S.x_c := u, M_2\big\rangle \xrightarrow{\mathsf{e}} \langle u', M_2'\rangle$, then by SOFT-ASSIGN, we have $\langle e_2, M_2\rangle = \big\langle (\mathsf{soft}\ m_2^S).x_c := u, M_2\big\rangle \to \langle u' \blacktriangleright_\alpha (a \sqcap p), M_2'\rangle$. Choose $\phi' = \phi$.

We proceed by cases according to the evaluation rules for $\big\langle m_1^S.x_c := v, M_1\big\rangle \xrightarrow{\mathsf{e}} \langle v', M_1'\rangle$.

Sub-case ASSIGN ($v' = * \blacktriangleright_\alpha p$ and $M_1' = M_1[m_1^S.x_c \mapsto v \blacktriangleright_\alpha \tau_c]$, where $M_1(m_1^S) \neq \bot$):

From $M_1 \approx_\alpha^\phi M_2$, we know $M_2(m_2^S) \neq \bot$. So, by ASSIGN, we have

$$\big\langle m_2^S.x_c := u, M_2\big\rangle \xrightarrow{\mathsf{e}} \big\langle * \blacktriangleright_\alpha p, M_2[m_2^S.x_c \mapsto u \blacktriangleright_\alpha \tau_c]\big\rangle.$$

So $u' = * \blacktriangleright_\alpha p$ and $M_2' = M_2[m_2^S.x_c \mapsto u \blacktriangleright_\alpha \tau_c]$.

i. We need to show $(* \blacktriangleright_\alpha p) \blacktriangleright_\alpha (a \sqcap p) \approx_\alpha^\phi (* \blacktriangleright_\alpha p) \blacktriangleright_\alpha (a \sqcap p)$.
This follows via Lemma 27.

ii. We need to show $M_1[m_1^S.x_c \mapsto v \blacktriangleright_\alpha \tau_c] \approx_\alpha^\phi M_2[m_2^S.x_c \mapsto u \blacktriangleright_\alpha \tau_c]$.
First, let $m_0^{S_0} \in \mathsf{dom}(\phi)$ be such that $M_1'(m_0^{S_0}) \neq \bot$. We show that $M_2'(\phi(m_0^{S_0})) \neq \bot$ and $M_1'(m_0^{S_0}) \approx_\alpha^\phi M_2'(\phi(m_0^{S_0}))$.
If $m_0^{S_0} = m_1^S$, then $\phi(m_0^{S_0}) = m_2^S$. Let $\overrightarrow{v_i}'$ and $\overrightarrow{u_i}'$ be such that $M_1'(m_1^S) = \{\overrightarrow{x_i = v_i'}\}$ and $M_2'(m_2^S) = \{\overrightarrow{x_i = u_i'}\}$. Since $v \approx_\alpha^\phi u$, by Lemma 27, we have $v \blacktriangleright_\alpha \tau_c \approx_\alpha^\phi u \blacktriangleright_\alpha \tau_c$. Since $v_c' = v \blacktriangleright_\alpha \tau_c$ and $u_c' = u \blacktriangleright_\alpha \tau_c$, we therefore have $v_c' \approx_\alpha^\phi u_c'$. Therefore, from $M_1 \approx_\alpha^\phi M_2$, it follows that $\{\overrightarrow{x_i = v_i'}\} \approx_\alpha^\phi \{\overrightarrow{x_i = u_i'}\}$. Otherwise, $m_0^{S_0} \neq m_1^S$, so $M_1'(m_0^{S_0}) = M_1(m_0^{S_0})$ and $M_2'(\phi(m_0^{S_0})) = M_2(\phi(m_0^{S_0}))$. The result therefore follows from $M_1 \approx_\alpha^\phi M_2$.
Now, let $m_0^{S_0} \in \mathsf{dom}(\phi)$ be such that $\vdash \alpha \leqslant \mathsf{auth}^+(S_0) \sqcap \mathsf{persist}(S_0)$ and $M_1'(m_0^{S_0}) = \bot$. We show that $M_2'(\phi(m_0^{S_0})) = \bot$. This follows from $M_1 \approx_\alpha^\phi M_2$ by construction of M_1' and M_2'.

Sub-case DANGLE-ASSIGN ($v' = \bot_p \blacktriangleright_\alpha p$ and $M_1' = M_1$, where $M_1(m_1^S) = \bot$):

First, suppose $\vdash \alpha \leqslant a \sqcap p$. Then $v' \blacktriangleright_\alpha (a \sqcap p) = \bot_p$ and $u' \blacktriangleright_\alpha (a \sqcap p) = u'$, and from $M_1 \approx_\alpha^\phi M_2$, we know $M_2(m_2^S) = \bot$. So, by DANGLE-ASSIGN, we have $\big\langle m_2^S.x_c := u, M_2\big\rangle \xrightarrow{\mathsf{e}} \big\langle \bot_p \blacktriangleright_\alpha p, M_2\big\rangle$. Therefore, $u' = \bot_p \blacktriangleright_\alpha p = \bot_p$ and $M_2' = M_2$.

i. We need to show $\bot_p \approx_\alpha^\phi \bot_p$.
This is trivial.

ii. We need to show $M_1 \approx_\alpha^\phi M_2$.
This is given.

Now, suppose $\alpha \not\leq a \sqcap p$. Then $v' \blacktriangleright_\alpha (a \sqcap p) = [\bot_p]$.

If $M_2(m_2^S) = \bot$, then by DANGLE-ASSIGN, $\langle m_2^S.x_c := u, M_2 \rangle \xrightarrow{\mathsf{e}} \langle \bot_p \blacktriangleright_\alpha p, M_2 \rangle$. Otherwise, $M_2(m_2^S) \neq \bot$, and by ASSIGN, $\langle m_2^S.x_c := u, M_2 \rangle \xrightarrow{\mathsf{e}} \langle * \blacktriangleright_\alpha p, M_2[m_2^S.x_c \mapsto u \blacktriangleright_\alpha \tau_c] \rangle$. Therefore, $u' \blacktriangleright_\alpha (a \sqcap p) \in \{(\bot_p \blacktriangleright_\alpha p) \blacktriangleright_\alpha (a \sqcap p), (* \blacktriangleright_\alpha p) \blacktriangleright_\alpha (a \sqcap p)\} = \{[\bot_p], [*]\}$.

i. We need to show $[\bot_p] \approx_\alpha^\phi [u']$ for $u' \in \{*, \bot_p\}$.
   This is trivial.
ii. We need to show $M_1' \approx_\alpha^\phi M_2'$.
   First, let $m_0^{S_0} \in \mathsf{dom}(\phi)$ be such that $M_1'(m_0^{S_0}) \neq \bot$. We show that $M_2'(\phi(m_0^{S_0})) \neq \bot$ and $M_1'(m_0^{S_0}) \approx_\alpha^\phi M_2'(\phi(m_0^{S_0}))$.
   Since $M_1' = M_1$ and $M_1'(m_0^{S_0}) \neq \bot$, we must have $m_0^{S_0} \neq m_1^S$, so $M_1'(m_0^{S_0}) = M_1(m_0^{S_0})$ and $M_2'(\phi(m_0^{S_0})) = M_2(\phi(m_0^{S_0}))$. The result therefore follows from $M_1 \approx_\alpha^\phi M_2$.
   Now, let $m_0^{S_0} \in \mathsf{dom}(\phi)$ be such that $\vdash \alpha \leq \mathsf{persist}(S_0)$ and $M_1'(m_0^{S_0}) = \bot$. We show that $M_2'(\phi(m_0^{S_0})) = \bot$.
   This follows from $M_1 \approx_\alpha^\phi M_2$ by construction of M_2'.

Case APPLY ($\langle(\lambda(x{:}\tau)[pc;\mathcal{H}].e_3)\ v_1, M_1\rangle \to_\alpha \langle e_3\{v_1/x\}, M_1\rangle$):

From $e_1 \approx_\alpha^\phi e_2$, we know that $e_2 = ((\lambda(x{:}\tau)[pc;\mathcal{H}].e_4)\ v_2)$, where $e_3 \approx_\alpha^\phi e_4$ and $v_1 \approx_\alpha^\phi v_2$. By APPLY, we have $\langle(\lambda(x{:}\tau)[pc;\mathcal{H}].e_4)\ v_2, M_2\rangle \to \langle e_4\{v_2/x\}, M_2\rangle$. Choose $\phi' = \phi$.

i. We need to show $e_3\{v_1/x\} \approx_\alpha^\phi e_4\{v_2/x\}$.
   This follows by Lemma 26.
ii. We need to show $M_1 \approx_\alpha^\phi M_2$.
   This is given.

Case EXISTS-TRUE ($\langle \mathsf{exists\ soft}\ m_1^S\ \mathsf{as}\ x : e_3\ \mathsf{else}\ e_4, M_1 \rangle \to_\alpha \langle (e_3\{m_1^S/x\}) \blacktriangleright_\alpha (a \sqcap p), M_1 \rangle$, where $M_1(m_1^S) \neq \bot$ and $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$):

From $e_1 \approx_\alpha^\phi e_2$, we know that $e_2 = \mathsf{exists\ soft}\ m_2^S\ \mathsf{as}\ x : e_5\ \mathsf{else}\ e_6$, where $\phi(m_1^S) = m_2^S$ and $e_3 \approx_\alpha^\phi e_5$. Since $M_1 \approx_\alpha^\phi M_2$ and $M_1(m_1^S) \neq \bot$, we know $M_2(m_2^S) \neq \bot$, so by EXISTS-TRUE, we have

$$\langle \mathsf{exists\ soft}\ m_2^S\ \mathsf{as}\ x : e_5\ \mathsf{else}\ e_6, M_2 \rangle \to \langle (e_5\{m_2^S/x\}) \blacktriangleright_\alpha (a \sqcap p), M_2 \rangle$$

Choose $\phi' = \phi$.

i. We need to show $(e_3\{m_1^S/x\}) \blacktriangleright_\alpha (a \sqcap p) \approx_\alpha^\phi (e_5\{m_2^S/x\}) \blacktriangleright_\alpha (a \sqcap p)$.
   This follows by Lemmas 26 and 27.
ii. We need to show $M_1 \approx_\alpha^\phi M_2$.
   This is given.

Case EXISTS-FALSE ($\langle \mathsf{exists\ soft}\ m_1^S\ \mathsf{as}\ x : e_3\ \mathsf{else}\ e_4, M_1 \rangle \to_\alpha \langle e_4 \blacktriangleright_\alpha (a \sqcap p), M_1 \rangle$, where $S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}$ and $M_1(m_1^S) = \bot$):

From $e_1 \approx_\alpha^\phi e_2$, we know that $e_2 = \mathsf{exists\ soft}\ m_2^S\ \mathsf{as}\ x : e_5\ \mathsf{else}\ e_6$, where $\phi(m_1^S) = m_2^S$ and $e_4 \approx_\alpha^\phi e_6$. We proceed by cases according to whether $\vdash \alpha \leq a \sqcap p$.

Sub-case $\vdash \alpha \leq a \sqcap p$:

We therefore know $e_4 \blacktriangleright_\alpha (a \sqcap p) = e_4$ and $\vdash \alpha \leq p$. So, from $M_1 \approx_\alpha^\phi M_2$ and $M_1(m_1^S) = \bot$, we know $M_2(m_2^S) = \bot$, so by EXISTS-FALSE, we have $\langle \mathsf{exists\ soft}\ m_2^S\ \mathsf{as}\ x : e_5\ \mathsf{else}\ e_6, M_2 \rangle \to \langle e_6, M_2 \rangle$. Choose $\phi' = \phi$.

i. We need to show $e_4 \approx_\alpha^\phi e_6$.
   This is given.
ii. We need to show $M_1 \approx_\alpha^\phi M_2$.
   This is given.

Sub-case $\alpha \not\leq a \sqcap p$:

We therefore have $e_4 \blacktriangleright_\alpha (a \sqcap p) = [e_4']$ for some e_4'. If $M_2(m_2^S) = \bot$, then by EXISTS-FALSE, we have $\langle \mathsf{exists\ soft}\ m_2^S\ \mathsf{as}\ x : e_5\ \mathsf{else}\ e_6, M_2 \rangle \to \langle [e_6'], M_2 \rangle$, for some e_6'. Otherwise, by EXISTS-TRUE, we have $\langle \mathsf{exists\ soft}\ m_2^S\ \mathsf{as}\ x : e_5\ \mathsf{else}\ e_6, M_2 \rangle \to \langle [e_5'], M_2 \rangle$, for some e_5'. Choose $\phi' = \phi$.

i. We need to show $[e_4'] \approx_\alpha^\phi [e_2'']$ for $e_2'' \in \{e_5', e_6'\}$.
   This is trivial.
ii. We need to show $M_1 \approx_\alpha^\phi M_2$.
   This is given.

Case TRY-VAL ($\langle \mathsf{try}\ v_1\ \mathsf{catch}\ p \colon e_3, M_1 \rangle \to_\alpha \langle v_1, M_1 \rangle$, where $\forall p'.\ v_1 \neq \bot_{p'}$ and $\forall v_1'.\ v_1 \neq [v_1']$):

From $e_1 \approx_\alpha^\phi e_2$, we know that $e_2 = (\mathsf{try}\ v_2\ \mathsf{catch}\ p \colon e_4)$, where $v_1 \approx_\alpha^\phi v_2$ and $e_3 \approx_\alpha^\phi e_4$. From this, it follows that by TRY-VAL, we have $\langle \mathsf{try}\ v_2\ \mathsf{catch}\ p \colon e_4, M_2 \rangle \to \langle v_2, M_2 \rangle$. Choose $\phi' = \phi$.

i. We need to show $v_1 \approx_\alpha^\phi v_2$.
   This is given.
ii. We need to show $M_1 \approx_\alpha^\phi M_2$.
   This is given.

Case TRY-CATCH ($\langle \mathsf{try}\ \bot_{p'}\ \mathsf{catch}\ p \colon e_3, M_1 \rangle \to_\alpha \langle e_3, M_1 \rangle$, where $\vdash p \leq p'$):

From $e_1 \approx_\alpha^\phi e_2$, we know that $e_2 = (\mathsf{try}\ \bot_{p'}\ \mathsf{catch}\ p \colon e_4)$, where $e_3 \approx_\alpha^\phi e_4$. By TRY-CATCH, we have $\langle \mathsf{try}\ \bot_{p'}\ \mathsf{catch}\ p \colon e_4, M_2 \rangle \to \langle e_4, M_2 \rangle$. Choose $\phi' = \phi$.

i. We need to show $e_3 \approx_\alpha^\phi e_4$.
   This is given.
ii. We need to show $M_1 \approx_\alpha^\phi M_2$.
   This is given.

Case TRY-ESC ($\langle \mathsf{try}\ \bot_{p'}\ \mathsf{catch}\ p \colon e_3, M_1 \rangle \to_\alpha \langle \bot_{p'}, M_1 \rangle$, where $p \not\leq p'$):

From $e_1 \approx_\alpha^\phi e_2$, we know that $e_2 = (\mathsf{try}\ \bot_{p'}\ \mathsf{catch}\ p \colon e_4)$. By TRY-ESC, we have $\langle \mathsf{try}\ \bot_{p'}\ \mathsf{catch}\ p \colon e_4, M_2 \rangle \to \langle \bot_{p'}, M_2 \rangle$. Choose $\phi' = \phi$.

i. We need to show $\bot_{p'} \approx_\alpha^\phi \bot_{p'}$.
   This is trivial.
ii. We need to show $M_1 \approx_\alpha^\phi M_2$.
   This is given.

Case PARALLEL-RESULT ($\langle v_1 \parallel v_2, M_1 \rangle \to_\alpha \langle *, M_1 \rangle$):

From $e_1 \approx_\alpha^\phi e_2$, we know that $e_2 = u_1 \parallel u_2$, where $v_i \approx_\alpha^\phi u_i$ for $i \in \{1,2\}$. By PARALLEL-RESULT, we have $\langle u_1 \parallel u_2, M_2 \rangle \to \langle *, M_2 \rangle$. Choose $\phi' = \phi$.

i. We need to show $* \approx_\alpha^\phi *$.
   This is trivial.
ii. We need to show $M_1 \approx_\alpha^\phi M_2$.
   This is given.

Case IF-TRUE ($\langle \mathsf{if\ true\ then}\ e_3\ \mathsf{else}\ e_4, M_1 \rangle \to_\alpha \langle e_3, M_1 \rangle$):

From $e_1 \approx_\alpha^\phi e_2$, we know that $e_2 = (\mathsf{if\ true\ then}\ e_5\ \mathsf{else}\ e_6)$, where $e_3 \approx_\alpha^\phi e_5$. By IF-TRUE, we have $\langle \mathsf{if\ true\ then}\ e_5\ \mathsf{else}\ e_6, M_2 \rangle \to \langle e_5, M_2 \rangle$. Choose $\phi' = \phi$.

i. We need to show $e_3 \approx^{\phi}_{\alpha} e_5$.
This is given.

ii. We need to show $M_1 \approx^{\phi}_{\alpha} M_2$.
This is given.

Case IF-FALSE ($\langle \mathsf{if\ false\ then}\ e_3\ \mathsf{else}\ e_4, M_1\rangle \to_\alpha \langle e_4, M_1\rangle$):

From $e_1 \approx^{\phi}_{\alpha} e_2$, we know that $e_2 = (\mathsf{if\ false\ then}\ e_5\ \mathsf{else}\ e_6)$, where $e_4 \approx^{\phi}_{\alpha} e_6$. By IF-FALSE, we have $\langle \mathsf{if\ false\ then}\ e_5\ \mathsf{else}\ e_6, M_2\rangle \to \langle e_6, M_2\rangle$. Choose $\phi' = \phi$.

i. We need to show $e_4 \approx^{\phi}_{\alpha} e_6$.
This is given.

ii. We need to show $M_1 \approx^{\phi}_{\alpha} M_2$.
This is given.

Case LET ($\langle \mathsf{let}\ x = v_1\ \mathsf{in}\ e_3, M_1\rangle \to_\alpha \langle e_3\{v_1/x\}, M_1\rangle$, where $\forall p.\ v_1 \neq \bot_p$ and $\forall v_1'.\ v_1 \neq [v_1']$):

From $e_1 \approx^{\phi}_{\alpha} e_2$, we know that $e_2 = (\mathsf{let}\ x = v_2\ \mathsf{in}\ e_4)$, where $v_1 \approx^{\phi}_{\alpha} v_2$ and $e_3 \approx^{\phi}_{\alpha} e_4$. By LET, we have $\langle \mathsf{let}\ x = v_2\ \mathsf{in}\ e_4, M_2\rangle \to \langle e_4\{v_2/x\}, M_2\rangle$. Choose $\phi' = \phi$.

i. We need to show $e_3\{v_1/x\} \approx^{\phi}_{\alpha} e_4\{v_2/x\}$.
This follows by Lemma 26.

ii. We need to show $M_1 \approx^{\phi}_{\alpha} M_2$.
This is given.

Case EVAL-CONTEXT ($\langle E[e_3], M_1\rangle \to_\alpha \langle E[e_3'], M_1'\rangle$, where $\langle e_3, M_1\rangle \xrightarrow{\mathsf{e}} \langle e_3', M_1'\rangle$):

We proceed by cases according to the syntax of $E[\cdot]$. We only show the case $E[\cdot] = \mathsf{try}\ [\cdot]\ \mathsf{catch}\ p\colon e_4$; the other cases follow similarly.

Sub-case $E[\cdot] = (\mathsf{try}\ [\cdot]\ \mathsf{catch}\ p\colon e_4)$:

From $e_1 \approx^{\phi}_{\alpha} e_2$, we know $e_2 = (\mathsf{try}\ e_5\ \mathsf{catch}\ p\colon e_6)$, where $e_3 \approx^{\phi}_{\alpha} e_5$ and $e_4 \approx^{\phi}_{\alpha} e_6$. To apply the induction hypothesis, we need:

- $\vdash^{\alpha}_{[wf]} \langle e_3, M_1\rangle$ and $\vdash_{[wf]} \langle e_5, M_2\rangle$
These follow from
$$\vdash^{\alpha}_{[wf]} \langle \mathsf{try}\ e_3\ \mathsf{catch}\ p\colon e_4, M_1\rangle$$
and
$$\vdash_{[wf]} \langle \mathsf{try}\ e_5\ \mathsf{catch}\ p\colon e_6, M_2\rangle\,.$$
- $\varnothing; pc; \mathcal{H}' \vdash e_3 : \tau', \mathcal{X}'$ and $\varnothing; pc; \mathcal{H}' \vdash e_5 : \tau', \mathcal{X}'$, for some $\mathcal{H}'$, τ', $\mathcal{X}'$
These follow from the typing derivations for $\mathsf{try}\ e_3\ \mathsf{catch}\ p\colon e_4$ and $\mathsf{try}\ e_5\ \mathsf{catch}\ p\colon e_6$.
- $\vdash^{\alpha}_{[wf]} M_2$ and $\langle e_3, M_1\rangle \approx^{\phi}_{\alpha} \langle e_5, M_2\rangle$.
These are given.

Therefore, we can apply the induction hypothesis to get a configuration $\langle e_5', M_2'\rangle$ and a high-integrity homomorphism ϕ' from M_1' to M_2' that extends ϕ, such that $\langle e_5, M_2\rangle \to \langle e_5', M_2'\rangle$ and

$$\langle e_3', M_1'\rangle \approx^{\phi'}_{\alpha} \langle e_5', M_2'\rangle. \tag{14}$$

So, by EVAL-CONTEXT, we have

$$\langle \mathsf{try}\ e_5\ \mathsf{catch}\ p\colon e_6, M_2\rangle \to \langle \mathsf{try}\ e_5'\ \mathsf{catch}\ p\colon e_6, M_2'\rangle\,.$$

From (14), we know $\langle \mathsf{try}\ e_3'\ \mathsf{catch}\ p\colon e_4, M_1'\rangle \approx^{\phi'}_{\alpha} \langle \mathsf{try}\ e_5'\ \mathsf{catch}\ p\colon e_6, M_2'\rangle$, as desired.

In case $E[\cdot] = \mathsf{soft}\ [\cdot]$, where $e_2 = \mathsf{soft}\ e_4$, to apply the induction hypothesis, we need the additional fact that since e_3 is not a value, neither is e_4, and therefore $\vdash^{\alpha}_{[wf]} \langle \mathsf{soft}\ e_3, M_1\rangle$ and $\vdash_{[wf]} \langle \mathsf{soft}\ e_4, M_2\rangle$ imply $\vdash^{\alpha}_{[wf]} \langle e_3, M_1\rangle$ and $\vdash_{[wf]} \langle e_4, M_2\rangle$.

Case FAIL-PROP ($\langle F[\bot_p], M_1\rangle \rightarrow_\alpha \langle \bot_p, M_1\rangle$):

From $e_1 \approx^{\phi}_{\alpha} e_2$, we know that $e_2 = F'[\bot_p]$. By FAIL-PROP, we have $\langle F'[\bot_p], M_2\rangle \rightarrow \langle \bot_p, M_2\rangle$. Choose $\phi' = \phi$.

i. We need to show $\bot_p \approx^{\phi}_{\alpha} \bot_p$.
   This is trivial.
ii. We need to show $M_1 \approx^{\phi}_{\alpha} M_2$.
   This is given.

Case GC ($\langle e_1, M_1\rangle \rightarrow_\alpha \langle e_1, M_1[G \mapsto \bot]\rangle$, where $\mathsf{gc}(G, \langle e_1, M_1\rangle)$):

By Lemma 25, let G' be such that $C \subseteq G' \subseteq \phi(G)$ and $\mathsf{gc}(G', \langle e_2, M_2\rangle)$, where

$$C = \{\phi(m^S) : m^S \in G \cap \mathsf{dom}(\phi) \wedge \vdash \alpha \preccurlyeq \mathsf{auth}^+(S) \sqcap \mathsf{persist}(S)\}$$

and

$$\phi(G) = \{\phi(m^S) : m^S \in G \cap \mathsf{dom}(\phi)\}.$$

Then, by GC, we have $\langle e_2, M_2\rangle \rightarrow \langle e_2, M_2[G' \mapsto \bot]\rangle$. Choose $\phi' = \phi$.

i. We need to show $e_1 \approx^{\phi}_{\alpha} e_2$.
   This is given.
ii. We need to show $M_1[G \mapsto \bot] \approx^{\phi}_{\alpha} M_2[G' \mapsto \bot]$.
   First, let $m_0^{S_0} \in \mathsf{dom}(\phi)$ be such that $M_1'(m_0^{S_0}) \neq \bot$. We show that $M_2'(\phi(m_0^{S_0})) \neq \bot$ and $M_1'(m_0^{S_0}) \approx^{\phi}_{\alpha} M_2'(\phi(m_0^{S_0}))$.
   Since $M_1'(m_0^{S_0}) \neq \bot$, we know $m_0^{S_0} \notin G$ and $\phi(m_0^{S_0}) \notin G'$. Therefore, $M_1'(m_0^{S_0}) = M_1(m_0^{S_0})$ and $M_2'(\phi(m_0^{S_0})) = M_2(\phi(m_0^{S_0}))$, so the result follows from the assumption $M_1 \approx^{\phi}_{\alpha} M_2$.
   Now, let $m_0^{S_0} \in \mathsf{dom}(\phi)$ be such that $\vdash \alpha \preccurlyeq \mathsf{auth}^+(S_0) \sqcap \mathsf{persist}(S_0)$ and $M_1'(m_0^{S_0}) = \bot$. We show that $M_2'(\phi(m_0^{S_0})) = \bot$. If $m_0^{S_0} \in G$, then $\phi(m_0^{S_0}) \in G'$, and the result follows by construction of M_2'. Otherwise, $m_0^{S_0} \notin G$, and so, $\phi(m_0^{S_0}) \notin G'$. Therefore, $M_1'(m_0^{S_0}) = M_1(m_0^{S_0})$ and $M_2'(\phi(m_0^{S_0})) = M_2(\phi(m_0^{S_0}))$, so the result follows from the assumption $M_1 \approx^{\phi}_{\alpha} M_2$.

Cases BRACKET-SELECT and BRACKET-SOFT-SELECT
($\langle [v].x_c, M_1\rangle \rightarrow_\alpha \langle [v.x_c], M_1\rangle$, where $v \in \{m_1^{S_1}, \mathsf{soft}\ m_1^{S_1}\}$):

From $e_1 \approx^{\phi}_{\alpha} e_2$, we know that $e_2 = [u].x_c$. From the grammar, we must have $u \in \{m_2^{S_2}, \mathsf{soft}\ m_2^{S_2}\}$. If $u = m_2^{S_2}$, then by BRACKET-SELECT, $\langle [m_2^{S_2}].x_c, M_2\rangle \rightarrow \langle [m_2^{S_2}.x_c], M_2\rangle$. Otherwise, $u = \mathsf{soft}\ m_2^{S_2}$, and by BRACKET-SOFT-SELECT, $\langle [\mathsf{soft}\ m_2^{S_2}].x_c, M_2\rangle \rightarrow \langle [(\mathsf{soft}\ m_2^{S_2}).x_c], M_2\rangle$. Choose $\phi' = \phi$.

i. We need to show $[v.x_c] \approx^{\phi}_{\alpha} [u.x_c]$.
   This is trivial.
ii. We need to show $M_1 \approx^{\phi}_{\alpha} M_2$.
   This is given.

Case BRACKET-ASSIGN ($\langle [v_1].x_c := v_2, M_1\rangle \rightarrow_\alpha \langle [v_1.x_c := v_2], M_1\rangle$):

From $e_1 \approx^{\phi}_{\alpha} e_2$, we know that $e_2 = [u_1].x_c := u_2$. By BRACKET-ASSIGN, we have $\langle [u_1].x_c := u_2, M_2\rangle \rightarrow \langle [u_1.x_c := u_2], M_2\rangle$. Choose $\phi' = \phi$.

i. We need to show $[v_1.x_c := v_2] \approx_\alpha^\phi [u_1.x_c := u_2]$.
This is trivial.

ii. We need to show $M_1 \approx_\alpha^\phi M_2$.
This is given.

Case BRACKET-SOFT ($\langle \mathsf{soft}\ [v], M_1\rangle \to_\alpha \langle [\mathsf{soft}\ v], M_1\rangle$):

From $e_1 \approx_\alpha^\phi e_2$, we know that $e_2 = \mathsf{soft}\ [e_3]$. Since $\langle e_2, M_2\rangle$ is assumed to not diverge, $\langle e_3, M_2\rangle$ cannot diverge either. So, by Corollary 14, there is a configuration $\langle u, M_2'\rangle$ such that $\langle e_3, M_2\rangle \xrightarrow{\mathsf{e}}{}^* \langle u, M_2'\rangle$. Then by EVAL-CONTEXT, BRACKET-CONTEXT, and BRACKET-SOFT, we have

$$\langle \mathsf{soft}\ [e_3], M_2\rangle \to^* \left\langle \mathsf{soft}\ [u], M_2'\right\rangle \to \left\langle [\mathsf{soft}\ u], M_2'\right\rangle.$$

Choose $\phi' = \phi$.

i. We need to show $[\mathsf{soft}\ v] \approx_\alpha^\phi [\mathsf{soft}\ u]$.
This is trivial.

ii. We need to show $M_1 \approx_\alpha^\phi M_2'$.
This follows by Lemma 29.

Case BRACKET-EXISTS
($\langle \mathsf{exists}\ [v]\ \mathsf{as}\ x: e_3\ \mathsf{else}\ e_4, M_1\rangle \to_\alpha \langle [\mathsf{exists}\ v\ \mathsf{as}\ x: e_3\ \mathsf{else}\ e_4], M_1\rangle$):

From $e_1 \approx_\alpha^\phi e_2$, we know that $e_2 = \mathsf{exists}\ [u]\ \mathsf{as}\ x: e_5\ \mathsf{else}\ e_6$. By BRACKET-EXISTS, we have

$$\langle \mathsf{exists}\ [u]\ \mathsf{as}\ x: e_5\ \mathsf{else}\ e_6, M_2\rangle \to \langle [\mathsf{exists}\ u\ \mathsf{as}\ x: e_5\ \mathsf{else}\ e_6], M_2\rangle$$

Choose $\phi' = \phi$.

i. We need to show $[\mathsf{exists}\ v\ \mathsf{as}\ x: e_3\ \mathsf{else}\ e_4] \approx_\alpha^\phi [\mathsf{exists}\ u\ \mathsf{as}\ x: e_5\ \mathsf{else}\ e_6]$.
This is trivial.

ii. We need to show $M_1 \approx_\alpha^\phi M_2$.
This is given.

Case BRACKET-APPLY ($\langle [v_1]\ v_2, M_1\rangle \to_\alpha \langle [v_1\ v_2], M_1\rangle$):

From $e_1 \approx_\alpha^\phi e_2$, we know that $e_2 = ([u_1]\ u_2)$. By BRACKET-APPLY, we have $\langle [u_1]\ u_2, M_2\rangle \to \langle [u_1\ u_2], M_2\rangle$. Choose $\phi' = \phi$.

i. We need to show $[v_1\ v_2] \approx_\alpha^\phi [u_1\ u_2]$.
This is trivial.

ii. We need to show $M_1 \approx_\alpha^\phi M_2$.
This is given.

Case BRACKET-TRY ($\langle \mathsf{try}\ [v]\ \mathsf{catch}\ p{:}\ e_3, M_1\rangle \to_\alpha \langle [\mathsf{try}\ v\ \mathsf{catch}\ p{:}\ e_3], M_1\rangle$):

From $e_1 \approx_\alpha^\phi e_2$, we know that $e_2 = (\mathsf{try}\ [e_4]\ \mathsf{catch}\ p{:}\ e_5)$. Since $\langle e_2, M_2\rangle$ is assumed to not diverge, $\langle e_4, M_2\rangle$ cannot diverge either. So, by Corollary 14, there is a configuration $\langle u, M_2'\rangle$ such that $\langle e_4, M_2\rangle \xrightarrow{\mathsf{e}}{}^* \langle u, M_2'\rangle$. Then by EVAL-CONTEXT, BRACKET-CONTEXT, and BRACKET-TRY, we have

$$\langle \mathsf{try}\ [e_4]\ \mathsf{catch}\ p{:}\ e_5, M_2\rangle \to^* \left\langle \mathsf{try}\ [u]\ \mathsf{catch}\ p{:}\ e_5, M_2'\right\rangle \to \left\langle [\mathsf{try}\ u\ \mathsf{catch}\ p{:}\ e_5], M_2'\right\rangle.$$

Choose $\phi' = \phi$.

i. We need to show $[\mathsf{try}\ v\ \mathsf{catch}\ p{:}\ e_3] \approx_\alpha^\phi [\mathsf{try}\ u\ \mathsf{catch}\ p{:}\ e_5]$.
This is trivial.

ii. We need to show $M_1 \approx^{\phi}_{\alpha} M'_2$.
This follows by Lemma 29.

Case BRACKET-IF ($\langle \mathsf{if}\ [v]\ \mathsf{then}\ e_3\ \mathsf{else}\ e_4, M_1\rangle \to_{\alpha} \langle [\mathsf{if}\ v\ \mathsf{then}\ e_3\ \mathsf{else}\ e_4], M_1\rangle$):

From $e_1 \approx^{\phi}_{\alpha} e_2$, we know that $e_2 = (\mathsf{if}\ [u]\ \mathsf{then}\ e_5\ \mathsf{else}\ e_6)$. By BRACKET-IF, we have

$$\langle \mathsf{if}\ [u]\ \mathsf{then}\ e_5\ \mathsf{else}\ e_6, M_2\rangle \to \langle [\mathsf{if}\ u\ \mathsf{then}\ e_5\ \mathsf{else}\ e_6], M_2\rangle .$$

Choose $\phi' = \phi$.

i. We need to show $[\mathsf{if}\ v\ \mathsf{then}\ e_3\ \mathsf{else}\ e_4] \approx^{\phi}_{\alpha} [\mathsf{if}\ u\ \mathsf{then}\ e_5\ \mathsf{else}\ e_6]$.
This is trivial.

ii. We need to show $M_1 \approx^{\phi}_{\alpha} M_2$.
This is given.

Case BRACKET-LET ($\langle \mathsf{let}\ x = [v]\ \mathsf{in}\ e_3, M_1\rangle \to_{\alpha} \langle [e_3\{[v]/x\}], M_1\rangle$, where $\forall p.\ v \neq \bot_p$):

From $e_1 \approx^{\phi}_{\alpha} e_2$, we know that $e_2 = (\mathsf{let}\ x = [u]\ \mathsf{in}\ e_4)$. If $\forall p.\ u \neq \bot_p$, then by BRACKET-LET, we have

$$\langle \mathsf{let}\ x = [u]\ \mathsf{in}\ e_4, M_2\rangle \to \langle [e_4\{[u]/x\}], M_2\rangle .$$

Otherwise, let p be such that $u = \bot_p$. Then, by BRACKET-FAIL, we have

$$\langle \mathsf{let}\ x = [u]\ \mathsf{in}\ e_4, M_2\rangle \to \left\langle [\bot_p], M_2\right\rangle .$$

Choose $\phi' = \phi$.

i. We need to show $[e_3\{[v]/x\}] \approx^{\phi}_{\alpha} [e''_2]$, where $e''_2 \in \{e_4\{[u]/x\}, \bot_p\}$.
This is trivial.

ii. We need to show $M_1 \approx^{\phi}_{\alpha} M_2$.
This is given.

Case DOUBLE-BRACKET ($\langle [[v]], M_1\rangle \to_{\alpha} \langle [v], M_1\rangle$):

From $e_1 \approx^{\phi}_{\alpha} e_2$, we know that $e_2 = [e''_2]$, so we trivially have $\langle [e''_2], M_2\rangle \to^{*} \langle [e''_2], M_2\rangle$. Choose $\phi' = \phi$.

i. We need to show $[v] \approx^{\phi}_{\alpha} [e''_2]$.
This is trivial.

ii. We need to show $M_1 \approx^{\phi}_{\alpha} M_2$.
This is given.

Case BRACKET-CONTEXT ($\langle [e_3], M_1\rangle \to_{\alpha} \left\langle [e'_3], M'_1\right\rangle$, where $\langle e_3, M_1\rangle \xrightarrow{\mathsf{e}} \left\langle e'_3, M'_1\right\rangle$):

From $e_1 \approx^{\phi}_{\alpha} e_2$, we know that $e_2 = [e_4]$, and we trivially have $\langle [e_4], M_2\rangle \xrightarrow{\mathsf{e}}{}^{*} \langle [e_4], M_2\rangle$. Choose $\phi' = \phi$.

i. We need to show $[e'_3] \approx^{\phi}_{\alpha} [e_4]$.
This is trivial.

ii. We need to show $M'_1 \approx^{\phi}_{\alpha} M_2$.
This follows by Lemma 29.

Case BRACKET-FAIL ($\left\langle F[[\bot_p]], M_1\right\rangle \to_{\alpha} \left\langle [\bot_p], M_1\right\rangle$):

From $e_1 \approx^{\phi}_{\alpha} e_2$, an easy case analysis on the syntax of $F[\cdot]$ shows that $\langle e_2, M_2\rangle \to \left\langle [\bot_p], M_2\right\rangle$ via BRACKET-FAIL.
Choose $\phi' = \phi$.

i. We need to show $[\bot_p] \approx_\alpha^\phi [\bot_p]$.
   This is trivial.

ii. We need to show $M_1 \approx_\alpha^\phi M_2$.
   This is given.

Case α-CREATE ($\langle e_1, M_1\rangle \to_\alpha \left\langle e_1, M_1[m^S \mapsto \{\overrightarrow{x_i = [v_i]}\}]\right\rangle$, where m^S is fresh, $\varnothing;\top;\top \vdash \{\overrightarrow{x_i = [v_i]}\}^S : R_\top, \top$, $\vdash_{[wf]}^\alpha$ $M[m^S \mapsto \{\overrightarrow{x_i = [v_i]}\}$, and $\alpha \not\leq \mathsf{persist}(S)$) :

We trivially have $\langle e_2, M_2\rangle \to^* \langle e_2, M_2\rangle$. Choose $\phi' = \phi$.

i. We need to show $e_1 \approx_\alpha^\phi e_2$.
   This is given.

ii. We need to show $M_1[m^S \mapsto \{\overrightarrow{x_i = [v_i]}\}] \approx_\alpha^\phi M_2$.
   Since $m^S \notin \mathsf{dom}(\phi)$, this follows from the assumption $M_1 \approx_\alpha^\phi M_2$ by construction of M_1'.

Case α-ASSIGN ($\langle e_1, M_1\rangle \to_\alpha \langle e_1, M_1[m^S.x_c \mapsto [v]]\rangle$, where $m^S \in \mathsf{dom}(M_1)$, $M_1(m^S) \neq \bot$, $S = \{\overrightarrow{x_i : \tau_i}\}_s$, $\varnothing;\top;\top \vdash [v] : \tau_c, \top$, and $\vdash_{[wf]}^\alpha M_1[m^S.x_c \mapsto [v]]$):

We trivially have $\langle e_2, M_2\rangle \to^* \langle e_2, M_2\rangle$. Choose $\phi' = \phi$.

i. We need to show $e_1 \approx_\alpha^\phi e_2$.
   This is given.

ii. We need to show $M_1[m^S.x_c \mapsto [v]] \approx_\alpha^\phi M_2$.
   If $m^S \notin \mathsf{dom}(\phi)$, then this follows from the assumption $M_1 \approx_\alpha^\phi M_2$ by construction of M_1'.
   Suppose $m^S \in \mathsf{dom}(\phi)$. Then it suffices to show that $M_1'(m^S) \approx_\alpha^\phi M_2(\phi(m^S))$, since the rest follows from $M_1 \approx_\alpha^\phi M_2$.
   Let $\overrightarrow{v_i}$, $\overrightarrow{v_i'}$, and $\overrightarrow{u_i}$ be such that $M_1(m^S) = \{\overrightarrow{x_i = v_i}\}$ $M_1'(m^S) = \{\overrightarrow{x_i = v_i'}\}$ and $M_2(m^S) = \{\overrightarrow{x_i = u_i}\}$. From $M_1 \approx_\alpha^\phi M_2$, we know $v_i \approx_\alpha^\phi u_i$ for all i. By construction of M_1', we have $v_c' = [v]$ and $v_i' = v_i$ for $i \neq c$. Therefore, it remains to be shown that $[v] \approx_\alpha^\phi u_c$.
   From $\varnothing;\top;\top \vdash [v] : \tau_c, \top$, we know that $\alpha \not\leq \mathsf{integ}(\tau_c)$. Therefore, from $\vdash_{[wf]}^\alpha M_2$, we must have $u_c = [u]$, for some u. The result $[v] \approx_\alpha^\phi u_c$ then follows trivially.

Case α-FORGET ($\langle e_1, M_1\rangle \to_\alpha \langle e_1, M_1[m^S \mapsto \bot]\rangle$, where $m^S \in \mathsf{dom}(M_1)$ and $\alpha \not\leq \mathsf{persist}(S)$):

We trivially have $\langle e_2, M_2\rangle \to^* \langle e_2, M_2\rangle$. Choose $\phi' = \phi$.

i. We need to show $e_1 \approx_\alpha^\phi e_2$.
   This is given.

ii. We need to show $M_1[m^S \mapsto \bot] \approx_\alpha^\phi M_2$.
   This follows from the assumption $M_1 \approx_\alpha^\phi M_2$, since $\alpha \not\leq \mathsf{persist}(S)$.

□

8.6 Storage attacks

To formalize immunity to storage attacks, we first show that the adversary cannot cause more high-persistence locations to be allocated. Theorem 1 captures this via the security relation, since all high-persistence locations are mapped by the homomorphism. We now show that the adversary cannot cause more high-authority locations to become non-collectible. Lemma 30 says that this is also implied by Theorem 1.

Lemma 30. *Assume $\langle e_1,M_1\rangle$ is a well-formed configuration and $\langle e_2,M_2\rangle$ is a well-formed, nonadversarial, ϕ-related configuration, such that e_1 and e_2 have type τ and M_2 is well-formed:*

$$\begin{gathered}\vdash^{\alpha}_{[wf]} \langle e_1,M_1\rangle \;\wedge\; \vdash_{[wf]} \langle e_2,M_2\rangle \;\wedge\; \langle e_1,M_1\rangle \approx^{\phi}_{\alpha} \langle e_2,M_2\rangle \\ \wedge\; \varnothing;pc;\mathcal{H} \vdash e_1 : \tau,\mathcal{X} \;\wedge\; \varnothing;pc;\mathcal{H} \vdash e_2 : \tau,\mathcal{X} \;\wedge\; \vdash^{\alpha}_{[wf]} M_2\end{gathered}$$

If m^S is a high-authority, noncollectible location in $\langle e_1,M_1\rangle$, then $\phi(m^S)$ is also noncollectible in $\langle e_2,M_2\rangle$.

$$\vdash \alpha \preccurlyeq \mathsf{auth}^{+}(S) \wedge \mathsf{nc}(m^S,\langle e_1,M_1\rangle) \Rightarrow \mathsf{nc}(\phi(m^S),\langle e_2,M_2\rangle)$$

To prove this, we first prove a preliminary result about garbage-collection roots.

Lemma 31. *Let e_1 and e_2 be well-typed expressions, both of type τ, that are related via a high-integrity homomorphism ϕ. If m^S is a high-authority GC root in e_1, then $\phi(m^S)$ is also a GC root in $\langle e_2,M_2\rangle$.*

$$\begin{gathered}\Gamma;pc;\mathcal{H} \vdash e_1 : \tau,\mathcal{X} \wedge \Gamma;pc;\mathcal{H} \vdash e_2 : \tau,\mathcal{X} \\ \wedge\, e_1 \approx^{\phi}_{\alpha} e_2 \wedge \vdash \alpha \preccurlyeq \mathsf{auth}^{+}(S) \wedge \mathsf{root}(m^S,e_1) \\ \Rightarrow \mathsf{root}(\phi(m^S),e_2)\end{gathered}$$

Proof. By induction on the derivation of $\mathsf{root}(m^S,e_1)$. The proof proceeds by cases according to the syntax of e_1. The result holds vacuously in cases $e_1 = x$, $e_1 = \mathsf{true}$, $e_1 = \mathsf{false}$, $e_1 = *$, and $e_1 = \mathsf{soft}\ m_1^{S_1}$, since $\neg\mathsf{root}(m^S,e_1)$ in these cases.

Case $e_1 = m_1^{S_1}$:

Since $\mathsf{root}(m^S,m_1^{S_1})$, we must have $m_1^{S_1} = m^S$. Therefore, from $e_1 \approx^{\phi}_{\alpha} e_2$, we have $e_2 = \phi(m^S)$, and the result follows via Rule R1.

Case $e_1 = \lambda(x{:}\tau')[pc';\mathcal{H}'].e_3$:

From $e_1 \approx^{\phi}_{\alpha} e_2$, we have $e_2 = \lambda(x{:}\tau')[pc';\mathcal{H}'].e_4$, where $e_3 \approx^{\phi}_{\alpha} e_4$. From the typing of e_1 and e_2, we have $\Gamma,x{:}\tau';pc';\mathcal{H}' \vdash e_3 : \tau'',\mathcal{H}'$ and $\Gamma,x{:}\tau';pc';\mathcal{H}' \vdash e_4 : \tau'',\mathcal{H}'$, for some pc', $\mathcal{H}'$, and τ''. From the derivation of $\mathsf{root}(m^S,e_1)$, we know $\mathsf{root}(m^S,e_3)$. Therefore, we can apply the induction hypothesis to obtain $\mathsf{root}(\phi(m^S),e_4)$, and the result follows via Rule R6.

Case $e_1 = v_1\ v_2$:

From the derivation of $\mathsf{root}(m^S,e_1)$, we know $\mathsf{root}(m^S,v_1)$ or $\mathsf{root}(m^S,v_2)$. Without loss of generality, assume $\mathsf{root}(m^S,v_1)$. From $e_1 \approx^{\phi}_{\alpha} e_2$, we have $e_2 = v_3\ v_4$, where $v_1 \approx^{\phi}_{\alpha} v_3$. From the typing of e_1 and e_2, we have $\Gamma;pc;\mathcal{H} \vdash v_1 : \tau',\top$ and $\Gamma;pc;\mathcal{H} \vdash v_3 : \tau',\top$, for some τ'. Therefore, we can apply the induction hypothesis to obtain $\mathsf{root}(\phi(m^S),v_1)$, and the result follows via Rule R7.

Case $e_1 = \mathsf{if}\ e_3\ \mathsf{then}\ e_4\ \mathsf{else}\ e_5$:

From the derivation of $\mathsf{root}(m^S,e_1)$, we know $\mathsf{root}(m^S,e_k)$ for some $k \in \{3,4,5\}$. From $e_1 \approx^{\phi}_{\alpha} e_2$, we have $e_2 = \mathsf{if}\ e_3'\ \mathsf{then}\ e_4'\ \mathsf{else}\ e_5'$, where $e_i \approx^{\phi}_{\alpha} e_i'$ for $i \in \{3,4,5\}$. From the typing of e_1 and e_2, we have $\Gamma;pc';\mathcal{H} \vdash e_k : \tau',\mathcal{X}'$ and $\Gamma;pc';\mathcal{H} \vdash e_k' : \tau',\mathcal{X}'$, for some pc', τ', and $\mathcal{X}'$. Therefore, we can apply the induction hypothesis to obtain $\mathsf{root}(\phi(m^S),e_k')$, and the result follows via Rule R10.

Case $e_1 = \{\overrightarrow{x_i = v_i}\}^{S_1}$:

From the derivation of $\mathsf{root}(m^S,e_1)$, we know $\mathsf{root}(m^S,v_k)$ for some k. From $e_1 \approx^{\phi}_{\alpha} e_2$, we have $e_2 = \{\overrightarrow{x_i = u_i}\}^{S_1}$, where $v_i \approx^{\phi}_{\alpha} u_i$ for all i. From the typing of e_1 and e_2, we have $\Gamma;pc;\mathcal{H} \vdash e_k : \tau',\top$ and $\Gamma;pc;\mathcal{H} \vdash e_k' : \tau',\top$, for some τ'. Therefore, we can apply the induction hypothesis to obtain $\mathsf{root}(\phi(m^S),u_k)$, and the result follows via Rule R3.

Case $e_1 = v.x$:

From the derivation of $\mathsf{root}(m^S,e_1)$, we know $\mathsf{root}(m^S,v)$. From $e_1 \approx^{\phi}_{\alpha} e_2$, we have $e_2 = u.x$, where $v \approx^{\phi}_{\alpha} u$. From the typing of e_1 and e_2, we have $\Gamma;pc;\mathcal{H} \vdash v : \tau',\top$ and $\Gamma;pc;\mathcal{H} \vdash u : \tau',\top$, for some τ'. Therefore, we can apply the induction hypothesis to obtain $\mathsf{root}(\phi(m^S),u)$, and the result follows via Rule R4.

Case $e_1 = v_1.x := v_2$:

From the derivation of $\mathsf{root}(m^S, e_1)$, we know $\mathsf{root}(m^S, v_k)$ for some k. From $e_1 \approx^{\phi}_{\alpha} e_2$, we have $e_2 = u_1.x := u_2$, where $v_i \approx^{\phi}_{\alpha} u_i$ for $i \in \{1,2\}$. From the typing of e_1 and e_2, we have $\Gamma;pc;\mathcal{H} \vdash v_k : \tau', \top$ and $\Gamma;pc;\mathcal{H} \vdash u_k : \tau', \top$, for some τ'. Therefore, we can apply the induction hypothesis to obtain $\mathsf{root}(\phi(m^S), u_k)$, and the result follows via Rule R5.

Case $e_1 = \mathsf{soft}\ e_3$:

From the derivation of $\mathsf{root}(m^S, e_1)$, we know $\mathsf{root}(m^S, e_3)$ and e_3 is not a memory location. From $e_1 \approx^{\phi}_{\alpha} e_2$, we have $e_2 = \mathsf{soft}\ e_4$, where $e_3 \approx^{\phi}_{\alpha} e_4$; therefore, e_4 is not a memory location. From the typing of e_1 and e_2, we have $\Gamma;pc;\mathcal{H} \vdash e_3 : \tau', \mathcal{X}$ and $\Gamma;pc;\mathcal{H} \vdash e_4 : \tau', \mathcal{X}$, for some τ'. Therefore, we can apply the induction hypothesis to obtain $\mathsf{root}(\phi(m^S), e_4)$, and the result follows via Rule R2.

Case $e_1 = e_3 \parallel e_4$:

From the derivation of $\mathsf{root}(m^S, e_1)$, we know $\mathsf{root}(m^S, e_k)$ for some $k \in \{3,4\}$. From $e_1 \approx^{\phi}_{\alpha} e_2$, we have $e_2 = e'_3 \parallel e'_4$, where $e_i \approx^{\phi}_{\alpha} e'_i$ for $i \in \{3,4\}$. From the typing of e_1 and e_2, we have $\Gamma;pc;\top \vdash e_k : \tau', \top$ and $\Gamma;pc;\top \vdash e'_k : \tau', \top$, for some τ'. Therefore, we can apply the induction hypothesis to obtain $\mathsf{root}(\phi(m^S), e'_k)$, and the result follows via Rule R9.

Case $e_1 = \mathsf{exists}\ e_3\ \mathsf{as}\ x : e_4\ \mathsf{else}\ e_5$:

From the derivation of $\mathsf{root}(m^S, e_1)$, we know $\mathsf{root}(m^S, e_k)$ for some $k \in \{3,4,5\}$. From $e_1 \approx^{\phi}_{\alpha} e_2$, we have $e_2 = \mathsf{exists}\ e'_3\ \mathsf{as}\ x : e'_4\ \mathsf{else}\ e'_5$, where $e_i \approx^{\phi}_{\alpha} e'_i$ for $i \in \{3,4,5\}$. From the typing of e_1 and e_2, we have $\Gamma';pc';\mathcal{H} \vdash e_k : \tau', \mathcal{X}'$ and $\Gamma';pc';\mathcal{H} \vdash e'_k : \tau', \mathcal{X}'$, for some Γ', pc', τ' and $\mathcal{X}'$. Therefore, we can apply the induction hypothesis to obtain $\mathsf{root}(\phi(m^S), e'_k)$, and the result follows via Rule R11.

Case $e_1 = \mathsf{let}\ x = e_3\ \mathsf{in}\ e_4$:

From the derivation of $\mathsf{root}(m^S, e_1)$, we know $\mathsf{root}(m^S, e_k)$ for some $k \in \{3,4\}$. From $e_1 \approx^{\phi}_{\alpha} e_2$, we have $e_2 = \mathsf{let}\ x = e'_3\ \mathsf{in}\ e'_4$, where $e_i \approx^{\phi}_{\alpha} e'_i$ for $i \in \{3,4\}$. From the typing of e_1 and e_2, we have $\Gamma';pc';\mathcal{H} \vdash e_k : \tau', \mathcal{X}'$ and $\Gamma';pc';\mathcal{H} \vdash e'_k : \tau', \mathcal{X}'$, for some Γ', pc', τ', and $\mathcal{X}'$. Therefore, we can apply the induction hypothesis to obtain $\mathsf{root}(\phi(m^S), e'_k)$, and the result follows via Rule R8.

Case $e_1 = \mathsf{try}\ e_3\ \mathsf{catch}\ p{:}\ e_4$:

From the derivation of $\mathsf{root}(m^S, e_1)$, we know $\mathsf{root}(m^S, e_k)$ for some $k \in \{3,4\}$. From $e_1 \approx^{\phi}_{\alpha} e_2$, we have $e_2 = \mathsf{try}\ e'_3\ \mathsf{catch}\ p{:}\ e'_4$, where $e_i \approx^{\phi}_{\alpha} e'_i$ for $i \in \{3,4\}$. From the typing of e_1 and e_2, we have $\Gamma;pc';\mathcal{H} \vdash e_k : \tau', \mathcal{X}'$ and $\Gamma;pc';\mathcal{H} \vdash e'_k : \tau', \mathcal{X}'$, for some pc', τ', and $\mathcal{X}'$. Therefore, we can apply the induction hypothesis to obtain $\mathsf{root}(\phi(m^S), e'_k)$, and the result follows via Rule R12.

Case $e_1 = [e_3]$:

From the derivation of $\mathsf{root}(m^S, e_1)$, we know $\mathsf{root}(m^S, e_3)$. From the typing of e_1, we know $\Gamma;pc \sqcap \ell;\mathcal{H} \vdash e_3 : \tau', \mathcal{X}$ for some ℓ and τ', where $\alpha \not\preccurlyeq \ell$ and $\vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \sqcap \ell$. Therefore, by Lemma 15, we must have $\alpha \not\preccurlyeq \mathsf{auth}^+(S)$, which contradicts the assumption $\vdash \alpha \preccurlyeq \mathsf{auth}^+(S)$. So, this case holds vacuously.

□

We now prove Lemma 30, restated here for convenience.

Lemma 30. *Assume $\langle e_1, M_1 \rangle$ is a well-formed configuration and $\langle e_2, M_2 \rangle$ is a well-formed, nonadversarial, ϕ-related configuration, such that e_1 and e_2 have type τ and M_2 is well-formed:*

$$\vdash^{\alpha}_{[wf]} \langle e_1, M_1 \rangle \ \wedge\ \vdash_{[wf]} \langle e_2, M_2 \rangle \ \wedge\ \langle e_1, M_1 \rangle \approx^{\phi}_{\alpha} \langle e_2, M_2 \rangle$$
$$\wedge\ \varnothing;pc;\mathcal{H} \vdash e_1 : \tau, \mathcal{X} \ \wedge\ \varnothing;pc;\mathcal{H} \vdash e_2 : \tau, \mathcal{X} \ \wedge\ \vdash^{\alpha}_{[wf]} M_2$$

If m^S is a high-authority, noncollectible location in $\langle e_1, M_1 \rangle$, then $\phi(m^S)$ is also noncollectible in $\langle e_2, M_2 \rangle$.

$$\vdash \alpha \preccurlyeq \mathsf{auth}^+(S) \wedge \mathsf{nc}(m^S, \langle e_1, M_1 \rangle) \Rightarrow \mathsf{nc}(\phi(m^S), \langle e_2, M_2 \rangle)$$

Proof. By induction on the derivation of $\mathsf{nc}(m^S, \langle e_1, M_1\rangle)$.

Case NC1:

We have $\mathsf{root}(m^S, e_1)$. By Lemma 31, we therefore have $\mathsf{root}(\phi(m^S), e_2)$, and the result follows by NC1.

Case NC2:

We have $\mathsf{root}(m_1^{S_1}, e_1)$ and $\mathsf{nc}(m^S, \langle v_c, M_1\rangle)$ for some $m_1^{S_1}$, where $M_1(m_1^{S_1}) = \{\overrightarrow{x_i = v_i}\}$. It therefore follows that $\mathsf{nc}(m^S, \left\langle m_1^{S_1}, M_1\right\rangle)$, $\vdash^{\alpha}_{[wf]} \left\langle m_1^{S_1}, M_1\right\rangle$, and $\vdash^{\alpha}_{[wf]} \langle v_c, M_1\rangle$.

Assume $m_1^{S_1} \neq m^S$. (Otherwise, we would have $\mathsf{root}(m^S, e_1)$, and the argument for Case NC1 applies.)

Since m^S is high-authority and $\mathsf{nc}(m^S, \left\langle m_1^{S_1}, M_1\right\rangle)$, by Corollary 18, we know $m_1^{S_1}$ is high-integrity, high-authority, and high-persistence.

Since $\mathsf{root}(m_1^{S_1}, e_1)$ and $m_1^{S_1}$ is high-authority, by Lemma 31, we know $\mathsf{root}(\phi(m_1^{S_1}), e_2)$.

From $\langle e_1, M_1\rangle \approx^{\phi}_{\alpha} \langle e_2, M_2\rangle$, we know $M_1 \approx^{\phi}_{\alpha} M_2$. So, from $M_1(m_1^{S_1}) \neq \bot$, we have $M_2(\phi(m_1^{S_1})) = \{\overrightarrow{x_i = u_i}\}$, where $v_c \approx^{\phi}_{\alpha} u_c$, for some $\overrightarrow{u_i}$.

Since $\vdash^{\alpha}_{[wf]} M_1$ and $\vdash_{[wf]} M_2$, we also know $\varnothing; \top; \top \vdash v_c : \tau_c, \top$ and $\varnothing; \top; \top \vdash u_c : \tau_c, \top$, for some τ_c. Therefore, we can apply the induction hypothesis to get $\mathsf{nc}(\phi(m^S), \langle u_c, M_2\rangle)$.

The result follows via NC2.

□

9 Related work

This paper identifies and addresses a new problem, referential security. As a result, little prior work is closely related.

Some prior work has tried to improve referential integrity through system mechanisms, for example improving the referential integrity of web hyperlinks [8, 12]. Systems mechanisms for improving referential integrity (and other aspects of trustworthiness) are orthogonal to the language model presented here, but could be used to justify assigning persistence, integrity, and authority levels to nodes.

Liblit and Aiken [13] develop a type system for distributed data structures. Its explicit two-level hierarchy distinguishes between local pointers that are meaningful only to a single processor, and global pointers that are valid everywhere. The type system ensures that local pointers do not leak into a global context. This work was extended in [14] to add types for dealing with private vs. shared data. However, this line of work does not consider security properties that require defense against an adversary.

Riely and Hennessey study type safety in a distributed system of partially trusted mobile agents [20] but do not consider referential security.

This paper builds on prior work on language-based information-flow security, much of which is summarized by [21]. The Fabric system [16] is programmed in a high-level language that includes integrity annotations and abstracts away the locations of objects, as $\lambda_{persist}$ does. Its type system does not enforce referential security, however, so adding the features described here is an obvious next step.

10 Conclusions

Complex distributed information systems are being integrated across different organizations with only partial trust, often in the context of cloud computing. But the security properties that are desirable in distributed computing are poorly understood, and the options for enforcing security are murkier still. In fact, the desirable referential security properties are actually in tension with each other. The result is that programmers have little guidance in designing distributed systems to be secure and reliable.

This paper makes several contributions that aid in resolving this situation. The paper newly identifies and formalizes some important referential security properties. It introduces a high-level language for modeling referential security issues in a distributed system. The language introduces a way to express referential security requirements through label annotations for persistence and creation authority, which can be viewed as different aspects of integrity. The paper demonstrates how to enforce referential security, through static analysis expressed as a type system in the language. The type system is validated by formal proofs that $\lambda_{persist}$ programs enforce the new security properties.

While this paper is a useful first step, clearly there is more to be done. The type system could be enriched with more features such as parametric polymorphism, recursive and dependent types. With such extensions, an implementation would then help evaluate how well these types guide programmers designing distributed computing systems.

Acknowledgments

This research was supported in part by ONR Grants N00014-09-1-0652 and N00014-13-1-0089; by MURI grant FA9550-12-1-0400, administered by the U.S. Air Force; by NSF Grants 0541217, 0627649, and 0964409; and by a grant from Microsoft Corporation. The views and conclusions here are those of the authors and do not necessarily reflect those of ONR, the Navy, the Air Force, NSF, or Microsoft. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright annotation thereon.

We also thank Aslan Askarov, Danfeng Zhang, Owen Arden, Barbara Liskov, Mike George, and David Schulz for their suggestions about this work or its presentation.

References

[1] Malcom Atkinson, François Bancilhon, David DeWitt, Klaus Dittrich, David Maier, and Stanley Zdonik. The object-oriented database system manifesto. In *Proc. International Conference on Deductive Object Oriented Databases*, Kyoto, Japan, December 1989.

[2] K. J. Biba. Integrity considerations for secure computer systems. Technical Report ESD-TR-76-372, USAF Electronic Systems Division, Bedford, MA, April 1977.

[3] Andrew Birrell, Greg Nelson, Susan Owicki, and Edward Wobber. Network objects. In *SOSP '93*, pages 217–230, December 1993.

[4] Andrew Black, Norman Hutchinson, Eric Jul, and Henry Levy. Object structure in the Emerald system. In *OOPSLA '86*, pages 78–86, November 1986.

[5] Breeze, 2013. `www.breezejs.com`.

[6] Heiko Böck. *Java Persistence API*. Springer, 2011.

[7] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems (TODS)*, 4(4):397–434, December 1979.

[8] Hugh C. Davis. Referential integrity of links in open hypermedia systems. In *Proc. 9th ACM Conference on Hypertext and Hypermedia*, pages 207–216, 1998.

[9] Dorothy E. Denning. *Cryptography and Data Security*. Addison-Wesley, Reading, Massachusetts, 1982.

[10] Joseph A. Goguen and Jose Meseguer. Security policies and security models. In *Proc. IEEE Symp. on Security and Privacy*, pages 11–20, April 1982.

[11] Hibernate. `www.hibernate.org`.

[12] Frank Kappe. A scalable architecture for maintaining referential integrity in distributed information systems. *Journal of Universal Computer Science*, 1(2), 1995.

[13] Ben Liblit and Alexander Aiken. Type systems for distributed data structures. In *POPL*, pages 199–213, January 2000.

[14] Ben Liblit, Alexander Aiken, and Katherine A. Yelick. Type systems for distributed data sharing. In *Proc. 10th International Static Analysis Symposium*, volume 2694 of *LNCS*, San Diego, California, June 2003. Springer-Verlag.

[15] Barbara H. Liskov. The Argus language and system. In *Distributed Systems: Methods and Tools for Specification*, volume 150 of *Lecture Notes in Computer Science*, pages 343–430. Springer-Verlag Berlin, 1985.

[16] Jed Liu, Michael D. George, K. Vikram, Xin Qi, Lucas Waye, and Andrew C. Myers. Fabric: A platform for secure distributed computation and storage. In *SOSP*, pages 321–334, 2009.

[17] D. Maier and J. Stein. Development and implementation of an object-oriented DBMS. In B. Shriver and P. Wegner, editors, *Research Directions in Object-Oriented Programming*. MIT Press, 1987.

[18] Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML.* MIT Press, Cambridge, MA, 1990.

[19] OMG. *The Common Object Request Broker: Architecture and Specification*, December 1991. OMG TC Document Number 91.12.1, Revision 1.1.

[20] James Riely and Matthew Hennessy. Trust and partial typing in open systems of mobile agents. In *POPL '99*, pages 93–104, 1999.

[21] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, January 2003.

[22] Michael B. Smyth. Power domains. *Journal of Computer and System Sciences*, 16(1):23–36, 1978.

[23] Lantian Zheng, Stephen Chong, Andrew C. Myers, and Steve Zdancewic. Using replication and partitioning to build secure distributed systems. In *Proc. IEEE Symp. on Security and Privacy*, pages 236–250, May 2003.

A Appendix

A.1 Full syntax of $\lambda_{persist}$

$$
\begin{array}{rrcl|rl}
 & & & & \text{Policy levels} & w,a,p,\ell \in \mathcal{L} \\
\text{Variables} & x,y & \in & \mathsf{Var} & \text{PC labels} & pc ::= w \\
\text{Memory locations} & m & \in & \mathsf{Mem} & \text{Storage labels} & s ::= (a,p) \\
\hline
\text{Labeled record types} & S & ::= & \{\overrightarrow{x_i : \tau_i}\}_s & \text{Reference labels} & r ::= (a^+, a^-, p) \\
\text{Labeled reference types} & R & ::= & \{\overrightarrow{x_i : \tau_i}\}_r & \text{Persistence failure handlers} & \mathcal{H} ::= \overrightarrow{p_i} \\
\hline
\text{Base types} & b & ::= & \mathsf{bool} \mid \tau_1 \xrightarrow{pc,\mathcal{H}} \tau_2 \mid R \mid \mathsf{soft}\ R & \text{Types} & \tau ::= b_w \mid \mathbf{1} \\
\hline
\text{Values} & v,u & ::= & \multicolumn{3}{l}{x \mid \mathsf{true} \mid \mathsf{false} \mid * \mid m^S \mid \mathsf{soft}\ m^S \mid \lambda(x:\tau)[pc;\mathcal{H}].e\ \ (\mid \bot_p)} \\
\text{Terms} & e & ::= & \multicolumn{3}{l}{v \mid v_1\ v_2 \mid \mathsf{if}\ v_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \mid \{\overrightarrow{x_i = v_i}\}^S \mid v.x \mid v_1.x := v_2} \\
 & & & \multicolumn{3}{l}{\mid \mathsf{soft}\ e \mid e_1 \parallel e_2 \mid \mathsf{exists}\ v\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2 \mid \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2} \\
 & & & \multicolumn{3}{l}{\mid \mathsf{try}\ e_1\ \mathsf{catch}\ p\colon e_2}
\end{array}
$$

A.2 Full small-step operational semantics for nonadversarial execution of $\lambda_{persist}$

$$[\textsc{Apply}]\quad \langle (\lambda(x{:}\tau)[pc;\mathcal{H}].e)\ v, M\rangle \overset{\mathsf{e}}{\to} \langle e\{v/x\}, M\rangle \qquad [\textsc{Let}]\quad \frac{\forall p.\ v \neq \bot_p}{\langle \mathsf{let}\ x = v\ \mathsf{in}\ e, M\rangle \overset{\mathsf{e}}{\to} \langle e\{v/x\}, M\rangle}$$

$$[\textsc{If-True}]\quad \langle \mathsf{if\ true\ then}\ e_1\ \mathsf{else}\ e_2, M\rangle \overset{\mathsf{e}}{\to} \langle e_1, M\rangle \qquad [\textsc{If-False}]\quad \langle \mathsf{if\ false\ then}\ e_1\ \mathsf{else}\ e_2, M\rangle \overset{\mathsf{e}}{\to} \langle e_2, M\rangle$$

$$[\textsc{Create}]\quad \frac{m = \mathsf{newloc}(M)}{\left\langle \{\overrightarrow{x_i = v_i}\}^S, M\right\rangle \overset{\mathsf{e}}{\to} \left\langle m^S, M[m^S \mapsto \{\overrightarrow{x_i = v_i}\}]\right\rangle}$$

$$\begin{bmatrix}\textsc{Parallel-}\\\textsc{Result}\end{bmatrix}\quad \langle v_1 \parallel v_2, M\rangle \overset{\mathsf{e}}{\to} \langle *, M\rangle \qquad [\textsc{Select}]\quad \frac{M(m^S) = \{\overrightarrow{x_i = v_i}\}}{\left\langle m^S.x_c, M\right\rangle \overset{\mathsf{e}}{\to} \langle v_c, M\rangle}$$

$$[\textsc{Assign}]\quad \frac{M(m^S) \neq \bot \qquad \forall p.\ v \neq \bot_p}{\left\langle m^S.x_c := v, M\right\rangle \overset{\mathsf{e}}{\to} \left\langle *, M[m^S.x_c \mapsto v]\right\rangle}$$

$$\begin{bmatrix}\textsc{Dangle-}\\\textsc{Select}\end{bmatrix}\quad \frac{M(m^S) = \bot \qquad p = \mathsf{persist}(m^S)}{\left\langle m^S.x_c, M\right\rangle \overset{\mathsf{e}}{\to} \langle \bot_p, M\rangle} \qquad \begin{bmatrix}\textsc{Dangle-}\\\textsc{Assign}\end{bmatrix}\quad \frac{M(m^S) = \bot \qquad p = \mathsf{persist}(m^S)}{\left\langle m^S.x_c := v, M\right\rangle \overset{\mathsf{e}}{\to} \langle \bot_p, M\rangle}$$

$$\begin{bmatrix}\textsc{Exists-}\\\textsc{True}\end{bmatrix}\quad \frac{M(m^S) \neq \bot}{\left\langle \mathsf{exists\ soft}\ m^S\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2, M\right\rangle \overset{\mathsf{e}}{\to} \left\langle e_1\{m^S/x\}, M\right\rangle}$$

$$\begin{bmatrix}\textsc{Exists-}\\\textsc{False}\end{bmatrix}\quad \frac{M(m^S) = \bot}{\left\langle \mathsf{exists\ soft}\ m^S\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2, M\right\rangle \overset{\mathsf{e}}{\to} \langle e_2, M\rangle}$$

$$\begin{bmatrix}\textsc{Eval-}\\\textsc{Context}\end{bmatrix}\quad \frac{\langle e, M\rangle \overset{\mathsf{e}}{\to} \left\langle e', M'\right\rangle}{\langle E[e], M\rangle \overset{\mathsf{e}}{\to} \left\langle E[e'], M'\right\rangle} \qquad \begin{bmatrix}\textsc{Fail-}\\\textsc{Prop}\end{bmatrix}\quad \langle F[\bot_p], M\rangle \overset{\mathsf{e}}{\to} \langle \bot_p, M\rangle$$

$$\begin{aligned}
E &::= \mathsf{soft}\ [\cdot] \mid \mathsf{let}\ x = [\cdot]\ \mathsf{in}\ e \mid [\cdot] \parallel e \mid e \parallel [\cdot] \mid \mathsf{try}\ [\cdot]\ \mathsf{catch}\ p\colon e \\
F &::= \mathsf{soft}\ [\cdot] \mid \mathsf{let}\ x = [\cdot]\ \mathsf{in}\ e
\end{aligned}$$

$$\begin{bmatrix}\textsc{Soft-}\\\textsc{Select}\end{bmatrix}\quad \frac{\left\langle m^S.x_c, M\right\rangle \overset{\mathsf{e}}{\to} \langle v, M\rangle}{\left\langle (\mathsf{soft}\ m^S).x_c, M\right\rangle \overset{\mathsf{e}}{\to} \langle v, M\rangle} \qquad \begin{bmatrix}\textsc{Soft-}\\\textsc{Assign}\end{bmatrix}\quad \frac{\left\langle m^S.x_c := v, M\right\rangle \overset{\mathsf{e}}{\to} \left\langle v', M'\right\rangle}{\left\langle (\mathsf{soft}\ m^S).x_c := v, M\right\rangle \overset{\mathsf{e}}{\to} \left\langle v', M'\right\rangle}$$

$$[\textsc{Try-Val}]\quad \frac{\forall p'.\ v \neq \bot_{p'}}{\langle \mathsf{try}\ v\ \mathsf{catch}\ p\colon e, M\rangle \overset{\mathsf{e}}{\to} \langle v, M\rangle} \qquad \begin{bmatrix}\textsc{Try-}\\\textsc{Catch}\end{bmatrix}\quad \frac{p \preccurlyeq p'}{\left\langle \mathsf{try}\ \bot_{p'}\ \mathsf{catch}\ p\colon e, M\right\rangle \overset{\mathsf{e}}{\to} \langle e, M\rangle}$$

$$[\textsc{Try-Esc}]\quad \frac{p \not\preccurlyeq p'}{\left\langle \mathsf{try}\ \bot_{p'}\ \mathsf{catch}\ p\colon e, M\right\rangle \overset{\mathsf{e}}{\to} \left\langle \bot_{p'}, M\right\rangle}$$

$$[\textsc{Prog-Step}]\quad \frac{\langle e, M\rangle \overset{\mathsf{e}}{\to} \left\langle e', M'\right\rangle}{\langle e, M\rangle \to \left\langle e', M'\right\rangle} \qquad [\textsc{GC}]\quad \frac{\mathsf{gc}(G, \langle e, M\rangle)}{\langle e, M\rangle \to \langle e, M[G \mapsto \bot]\rangle}$$

A.3 Full subtyping rules for $\lambda_{persist}$

$$[\mathrm{S1}]\quad \frac{n > m}{\vdash \{x_1 : \tau_1, \ldots, x_n : \tau_n\}_r \leq \{x_1 : \tau_1, \ldots, x_m : \tau_m\}_r} \qquad [\mathrm{S2}]\quad \frac{\vdash R_1 \leq R_2}{\vdash \mathsf{soft}\ R_1 \leq \mathsf{soft}\ R_2} \qquad [\mathrm{S3}]\quad \frac{\begin{array}{c}\vdash b_1 \leq b_2 \\ \vdash w_2 \preccurlyeq w_1\end{array}}{\vdash (b_1)_{w_1} \leq (b_2)_{w_2}}$$

$$[\mathrm{S4}]\quad \frac{\begin{array}{cc}\vdash \tau_2 \leq \tau_1 & \vdash \tau_1' \leq \tau_2' \\ \vdash pc_1 \preccurlyeq pc_2 & \vdash \mathcal{H}_2 \preccurlyeq \mathcal{H}_1\end{array}}{\vdash \tau_1 \xrightarrow{pc_1,,\mathcal{H}_1} \tau_1' \leq \tau_2 \xrightarrow{pc_2,,\mathcal{H}_2} \tau_2'} \qquad [\mathrm{S5}]\quad \frac{\vdash a_1^+ \preccurlyeq a_2^+ \qquad \vdash a_2^- \preccurlyeq a_1^- \qquad \vdash p_2 \preccurlyeq p_1}{\vdash \{\overrightarrow{x_i : \tau_i}\}_{(a_1^+, a_1^-, p_1)} \leq \{\overrightarrow{x_i : \tau_i}\}_{(a_2^+, a_2^-, p_2)}}$$

A.4 Full typing rules for $\lambda_{persist}$

$$[\textsc{T-Bool}]\quad \frac{b \in \{\mathsf{true}, \mathsf{false}\}}{\Gamma; pc; \mathcal{H} \vdash b : \mathsf{bool}_\top, \top} \qquad [\textsc{T-Unit}]\quad \Gamma; pc; \mathcal{H} \vdash * : \mathbf{1}, \top \qquad [\textsc{T-Var}]\quad \frac{\Gamma(x) = \tau}{\Gamma; pc; \mathcal{H} \vdash x : \tau, \top}$$

$$[\textsc{T-Bottom}]\quad \frac{p \neq \top \qquad \vdash \mathcal{H} \preccurlyeq p}{\Gamma; pc; \mathcal{H} \vdash \bot_p : \tau, p} \qquad [\textsc{T-Loc}]\quad \frac{\vdash_{wf} S : \mathsf{rectype} \qquad S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)}}{\Gamma; pc; \mathcal{H} \vdash m^S : (\{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top, \top}$$

$$[\textsc{T-Parallel}]\quad \frac{\Gamma; pc; \top \vdash e_i : \tau_i, \top\ ^{(\forall i)}}{\Gamma; pc; \mathcal{H} \vdash e_1 \parallel e_2 : \mathbf{1}, \top} \qquad [\textsc{T-Soft}]\quad \frac{\Gamma; pc; \mathcal{H} \vdash e : R_w, \mathcal{X}}{\Gamma; pc; \mathcal{H} \vdash \mathsf{soft}\ e : (\mathsf{soft}\ R)_w, \mathcal{X}}$$

$$[\textsc{T-If}]\quad \frac{\begin{array}{c}\Gamma; pc; \mathcal{H} \vdash v : \mathsf{bool}_w, \top \\ \Gamma; pc \sqcap w; \mathcal{H} \vdash e_i : \tau, \mathcal{X}_i\ ^{(\forall i)} \qquad \vdash \mathsf{auth}^+(\tau) \preccurlyeq pc \sqcap w\end{array}}{\Gamma; pc; \mathcal{H} \vdash \mathsf{if}\ v\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau \sqcap w, \mathcal{X}_1 \sqcap \mathcal{X}_2}$$

$$[\textsc{T-Abs}]\quad \frac{\Gamma, x{:}\tau'; pc'; \mathcal{H}' \vdash e : \tau, \mathcal{H}' \qquad \vdash_{wf} (\tau' \xrightarrow{pc', \mathcal{H}'} \tau)_\top : \mathsf{type} \qquad \vdash pc' \preccurlyeq pc}{\Gamma; pc; \mathcal{H} \vdash \lambda(x{:}\tau')[pc'; \mathcal{H}'].e : (\tau' \xrightarrow{pc', \mathcal{H}'} \tau)_\top, \top}$$

$$[\textsc{T-App}]\quad \frac{\begin{array}{c}\Gamma; pc; \mathcal{H} \vdash v_1 : (\tau' \xrightarrow{pc', \mathcal{H}'} \tau)_w, \top \qquad \Gamma; pc; \mathcal{H} \vdash v_2 : \tau', \top \\ \vdash pc' \preccurlyeq pc \sqcap w \qquad \vdash \mathcal{H} \preccurlyeq \mathcal{H}'\end{array}}{\Gamma; pc; \mathcal{H} \vdash v_1\ v_2 : \tau \sqcap w, \mathcal{H}'}$$

$$[\textsc{T-Record}]\quad \frac{\begin{array}{c}\vdash_{wf} S : \mathsf{rectype} \qquad S = \{\overrightarrow{x_i : \tau_i}\}_{(a,p)} \qquad \Gamma; pc; \mathcal{H} \vdash v_i : \tau_i', \top\ ^{(\forall i)} \\ \vdash \tau_i' \leq \tau_i\ ^{(\forall i)} \qquad \vdash \mathsf{auth}^+(\tau_i') \preccurlyeq pc\ ^{(\forall i)} \qquad \vdash \mathsf{integ}(\tau_i) \preccurlyeq pc\ ^{(\forall i)} \qquad \vdash p \preccurlyeq pc\end{array}}{\Gamma; pc; \mathcal{H} \vdash \{\overrightarrow{x_i = v_i}\}^S : (\{\overrightarrow{x_i : \tau_i}\}_{(a,a,p)})_\top, \top}$$

$$[\textsc{T-Select}]\quad \frac{\Gamma; pc; \mathcal{H} \vdash v : (\{\overrightarrow{x_i : \tau_i}\}_{(a^+, a^-, p)})_w, \top \qquad \vdash a^+ \preccurlyeq pc \qquad w' = w \sqcap p \qquad \vdash \mathcal{H} \preccurlyeq p}{\Gamma; pc; \mathcal{H} \vdash v.x_c : \tau_c \sqcap w', p}$$

$$[\textsc{T-Assign}]\quad \frac{\begin{array}{c}\Gamma; pc; \mathcal{H} \vdash v_1 : (\{\overrightarrow{x_i : \tau_i}\}_{(a^+, a^-, p)})_w, \top \qquad \vdash a^+ \preccurlyeq pc \qquad \Gamma; pc; \mathcal{H} \vdash v_2 : \tau, \top \\ \vdash \tau \sqcap pc \sqcap w \leq \tau_c \qquad \vdash \mathsf{auth}^+(\tau) \preccurlyeq pc \sqcap w \qquad \vdash \mathcal{H} \preccurlyeq p\end{array}}{\Gamma; pc; \mathcal{H} \vdash v_1.x_c := v_2 : \mathbf{1}, p}$$

$$\left[\begin{array}{c}\textsc{T-Soft-} \\ \textsc{Select}\end{array}\right]\quad \frac{\begin{array}{c}\Gamma; pc; \mathcal{H} \vdash v : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_{(a^+, a^-, p)})_w, \top \qquad \vdash \mathsf{auth}^+(\tau_c) \preccurlyeq pc \\ w' = w \sqcap a^- \sqcap p \qquad \vdash \mathcal{H} \preccurlyeq p\end{array}}{\Gamma; pc; \mathcal{H} \vdash v.x_c : \tau_c \sqcap w', p}$$

$$\left[\begin{array}{c}\textsc{T-Soft-} \\ \textsc{Assign}\end{array}\right]\quad \frac{\begin{array}{c}\Gamma; pc; \mathcal{H} \vdash v_1 : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_{(a^+, a^-, p)})_w, \top \qquad \Gamma; pc; \mathcal{H} \vdash v_2 : \tau, \top \\ \vdash \tau \sqcap pc \sqcap w \leq \tau_c \qquad \vdash \mathsf{auth}^+(\tau) \preccurlyeq pc \sqcap w \qquad \vdash \mathcal{H} \preccurlyeq p\end{array}}{\Gamma; pc; \mathcal{H} \vdash v_1.x_c := v_2 : \mathbf{1}, p}$$

$$[\textsc{T-Exists}]\quad \frac{\begin{array}{c}\Gamma; pc; \mathcal{H} \vdash v : (\mathsf{soft}\ \{\overrightarrow{x_i : \tau_i}\}_r)_w, \top \qquad \vdash \mathsf{auth}^+(r) \preccurlyeq pc \sqcap w \\ w' = \mathsf{auth}^-(r) \sqcap \mathsf{persist}(r) \sqcap w \qquad \Gamma, x{:}(\{\overrightarrow{x_i : \tau_i}\}_r)_w; pc \sqcap w'; \mathcal{H} \vdash e_1 : \tau, \mathcal{X}_1 \\ \Gamma; pc \sqcap w'; \mathcal{H} \vdash e_2 : \tau, \mathcal{X}_2 \qquad \vdash \mathsf{auth}^+(\tau) \preccurlyeq pc \sqcap w'\end{array}}{\Gamma; pc; \mathcal{H} \vdash \mathsf{exists}\ v\ \mathsf{as}\ x : e_1\ \mathsf{else}\ e_2 : \tau \sqcap w', \mathcal{X}_1 \sqcap \mathcal{X}_2}$$

$$[\textsc{T-Try}]\quad \frac{\begin{array}{c}\Gamma; pc; \mathcal{H}, p \vdash e_1 : \tau, \mathcal{X}_1 \qquad w = \bigsqcap_{p' \in \mathcal{X}_1} (p \sqcup p') \\ \Gamma; pc \sqcap w \sqcap \mathsf{integ}(\tau); \mathcal{H} \vdash e_2 : \tau, \mathcal{X}_2 \qquad \vdash \mathsf{auth}^+(\tau) \preccurlyeq pc\end{array}}{\Gamma; pc; \mathcal{H} \vdash \mathsf{try}\ e_1\ \mathsf{catch}\ p\text{:}\ e_2 : \tau \sqcap w, (\mathcal{X}_1 / p) \sqcap \mathcal{X}_2}$$

$$[\textsc{T-Let}]\quad \frac{\begin{array}{c}\Gamma; pc; \mathcal{H} \vdash e_1 : \tau', \mathcal{X}_1 \qquad \vdash \mathsf{auth}^+(\tau') \preccurlyeq pc \qquad w = (\bigsqcap \mathcal{X}_1) \sqcap \mathsf{integ}(\tau') \\ pc' = pc \sqcap w \qquad \Gamma, x{:}\tau'; pc'; \mathcal{H} \vdash e_2 : \tau, \mathcal{X}_2 \qquad \vdash \mathsf{auth}^+(\tau) \preccurlyeq pc'\end{array}}{\Gamma; pc; \mathcal{H} \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau \sqcap w, \mathcal{X}_1 \sqcap \mathcal{X}_2}$$

$$[\textsc{T-Subsume}]\quad \frac{\Gamma; pc; \mathcal{H} \vdash e : \tau', \mathcal{X}' \qquad \vdash \tau' \leq \tau \qquad \vdash \mathcal{H} \preccurlyeq \mathcal{X} \qquad \vdash \mathcal{X} \preccurlyeq \mathcal{X}'}{\Gamma; pc; \mathcal{H} \vdash e : \tau, \mathcal{X}}$$