Flow-Limited Authorization for consensus, replication, and secret sharing

Priyanka Mondal [a], Maximilian Algehed [b] and Owen Arden [c]
[a] *University of California, Santa Cruz, CA, USA*
E-mail: pmondal@ucsc.edu
[b] *Chalmers University of Technology, Gothenburg, Sweden*
E-mail: algehed@chalmers.se
[c] *University of California, Santa Cruz, CA, USA*
E-mail: owen@soe.ucsc.edu

Abstract. Availability is crucial to the security of distributed systems, but guaranteeing availability is hard, especially when participants in the system may act maliciously. Quorum replication protocols provide both integrity and availability: data and computation is replicated at multiple independent hosts, and a quorum of these hosts must agree on the output of all operations applied to the data. Unfortunately, these protocols have high overhead and can be difficult to calibrate for a specific application's needs. Ideally, developers could use high-level abstractions for consensus and replication to write fault-tolerant code that is secure by construction.

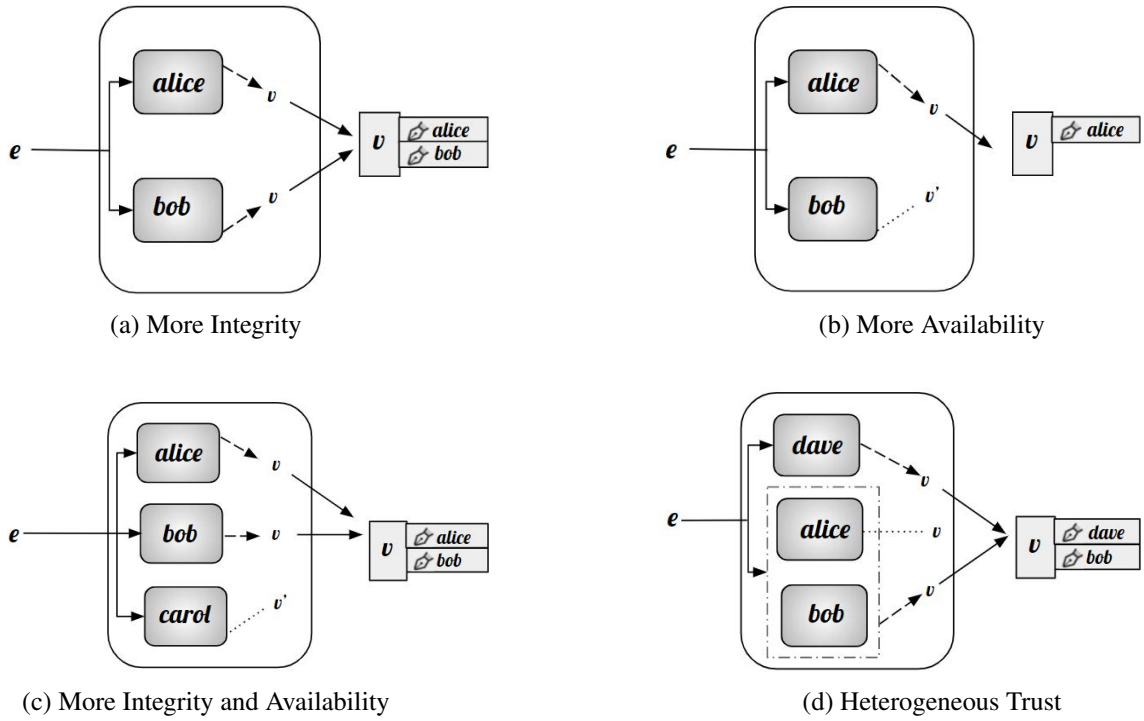
This paper presents Flow-Limited Authorization for Quorum Replication (FLAQR), a core calculus for building distributed applications with heterogeneous quorum replication protocols while enforcing end-to-end information security. Our type system ensures that well-typed FLAQR programs cannot *fail* (experience an unrecoverable error) in ways that violate their type-level specifications. We present noninterference theorems that characterize FLAQR's confidentiality, integrity, and availability in the presence of consensus, replication, and failures, as well as a liveness theorem for the class of majority quorum protocols under a bounded number of faults. Additionally, we present an extension to FLAQR that supports secret sharing as a form of declassification and prove it preserves integrity and availability security properties.

1. Introduction

Failure is inevitable in distributed systems, but its consequences may vary. The consequences of failure are particularly severe in centralized system designs, where single points-of-failure can render the entire system inoperable. Even distributed systems are sometimes built using a single, centralized authority to execute security-critical tasks. If this trusted entity is compromised, the security of the entire system may be compromised as well.

Building reliable *decentralized systems*, which have no single point-of-failure, is a complex task. Quorum replication protocols such as Paxos [1] and PBFT [2], and blockchains such as Bitcoin [3] replicate state and computation at independent hosts and use consensus protocols to ensure the integrity and availability of operations on system state. In these protocols, there is neither centralization of function nor centralization of trust: all honest hosts work to replicate the same computation on the same data, and this redundancy helps the system tolerate a bounded number of node failures and corruptions.

Within a single trust domain such as a corporate data center, replicas likely have uniform trust relationships and may be treated interchangeably. However, many large-scale systems depend on services



(a) More Integrity (b) More Availability

(c) More Integrity and Availability (d) Heterogeneous Trust

Fig. 1. Integrity-Availability Trade-off

hosted by multiple external services. Even when a service's internal components are replicated, developers must take into account the failure properties of external dependencies when considering their own robustness.

Information flow control (IFC) has been used to enforce decentralized security in distributed systems for confidentiality and integrity (e.g., Fabric [4] and DStar [5]). Less attention has been paid to enforcing decentralized availability policies with IFC. In particular, no language (or protocol) we are aware of addresses systems that compose multiple quorums or consider quorum participants with arbitrary trust relationships.

To build a formal foundation for such languages, we present FLAQR, a core calculus for Flow-Limited Authorization [6] for Quorum Replication. FLAQR uses high-level abstractions for replication and consensus that help manage tradeoffs between the availability and integrity of computation and data.

Consider the scenarios in Figure 1. Shaded boxes represent hosts in a distributed system. Dashed lines denote outputs that contribute to the final result, a value v. Dotted lines denote ignored outputs and solid lines indicate the flow of data from an initial expression e distributed to hosts to the collected result. Results are accompanied by labels that indicate which hosts influenced the final result.

In Figure 1b, e is distributed to hosts `alice` and `bob`. The hosts' results are compared and, if they match, the result is produced. Since a value is output only if the values match, we can treat the output of this protocol as having *more integrity* than just `alice` or `bob`. While both `alice` and `bob` technically influence the output, neither host can unilaterally control its value. However, either host can cause the protocol to fail.

By contrast, the protocol in Figure 1a prioritizes availability over integrity: if either alice or bob produce a value, the protocol outputs a value—in this case alice's. Here, neither host can unilaterally cause a failure; the protocol only fails if both alice and bob fail. Either alice or bob (but not both) has complete control over the result in the event of the other's failure, so we should treat the output as having *less integrity* than just alice or bob.

With an adequate number of hosts, we can combine these two techniques to form the essential components of a quorum system. In Figure 1c, e is replicated to alice, bob, and carol. This protocol outputs a value if any two hosts have matching outputs. Since alice and bob both output v, the protocol outputs v and attaches alice and bob's signatures. The non-matching value v' from carol is ignored. Hence, this protocol prevents any single host from unilaterally controlling the failure of the protocol or its output.

Figure 1c is similar in spirit to consensus protocols such as Paxos or PBFT where quorums of independent replicas are used to tolerate a bounded number of failures. FLAQR also permits us to write protocols where principals have differing trust relationships. Figure 1d illustrates a protocol that tolerates failure (or corruption) of either alice or bob, but requires dave's output to be part of any quorum. This protocol will fail if both alice and bob fail to produce matching outputs, but can also fail if dave fails to produce a matching output. This example illustrates the distributed systems where the hosts do not have homogeneous trust.

The main contributions of this paper are:

- An extension of the static fragment of the Flow Limited Authorization Model (FLAM) [6] with availability policies and algebraic operators representing the effective authority of consensus and replication protocols (§3-§5).
- A formalization of the FLAQR language (§4) and accompanying results:
  - A liveness theorem for majority-quorum FLAQR protocols (§7.1) which experience a bounded number of faults using a novel proof technique: a *blame semantics* that associates failing executions of a FLAQR program with a set of principals who may have caused the failure.
  - Noninterference theorems for confidentiality, integrity, and availability (§7.2).
  - An extension to FLAQR adding support for simple secret sharing, and results demonstrating it preserves integrity and availability noninterference as well as our liveness theorem, despite introducing an additional source of failure due to mismatched shares (§9).

This paper is an expanded and updated version of an article previously published in the proceedings of the 35^{th} Computer Security Foundations Symposium [7]. This version adds support for secret sharing (Section 9) and extensions of our previous results that demonstrate these new terms neither impact integrity and availability noninterference, nor majority liveness, despite introducing an additional source of failure. In addition, we corrected a minor issue in the original blame semantics, and include complete rule sets and proofs for our formalization and theoretical results.

Non-goals. The design of FLAQR is motivated by application-agnostic consensus protocols such as Paxos [1] and PBFT [2], but our present goal is not to develop a framework for *verifying* implementations of such protocols (although it would be interesting future work). Rather, the goal is to develop security abstractions that make it easier to create components with application-specific integrity and availability guarantees, and compose them in a secure and principled way.

In particular, the FLAQR system model lacks some features that a protocol verification model would require, most notably a concurrent semantics, asynchronous message delivery, and arbitrary communication patterns. Although this simplifies some aspects of consensus protocols, our model retains many

```
1  getBalance(acct):
2      bal_a = fetch bal(acct) @ alice;
3      bal_b = fetch bal(acct) @ bob;
4      bal_c = fetch bal(acct) @ carol;
5
6      if (bal_a==bal_b && bal_a != fail)
7         return bal_a;
8      else if (bal_b==bal_c && bal_b != fail)
9         return bal_b;
10     else if (bal_c==bal_a && bal_c != fail)
11        return bal_c;
12     else return fail;
```

Fig. 2. Majority quorum

of the core challenges present in fault tolerance models. For example, perfect fault detection is impossible and faulty hosts can manipulate data to cause failures to manifest at other hosts. We argue that even in a synchronous, deterministic model with RPC-style communication, the challenges of specifying and enforcing policies remain quite difficult to solve, and are among the primary security concerns of high-level application developers.

2. Motivating examples.

In this section we present two motivating examples. The first example highlights the trade-off between integrity and availability. The second example highlights the need for availability policies in distributed systems.

2.1. Tolerating failure and corruption

If a bank's deposit records are stored in a single node, then customers will be unable to access their accounts if that node is unavailable or is compromised. To eliminate this single point-of-failure, banks can replicate their records on multiple hosts as illustrated in Figure 1c. If a majority of hosts agree on an account balance, then the system can tolerate the remaining minority of hosts failing or returning corrupted results.

Consider a quorum system with three hosts: `alice`, `bob`, and `carol`. To tolerate the failure of a single node, balance queries attempt to contact all three hosts and compare the responses. As long as the client receives two responses with the same balance, the client can be confident the balance is correct even if one node is compromised or has failed.

Figure 2 illustrates a pseudocode implementation of `getBalance` in this system. The code fetches balances from the three hosts (lines 2-4). The function returns the balance if each fetched value matches, otherwise the function returns `fail` (lines 6-12).

The downside of this approach is that it is quite verbose and repetitive compared to a single-line fetch without any fault tolerance. Small mistakes in any of these lines could have significant consequences. For example, suppose a programmer typed `bal_b` instead of `bal_c` on line 8. This small change gives

```
1  highestBalance(acct_1, acct_2):
2    bal_1:= fetch getBalance(acct_1) @ b;
3    bal_2:= fetch getBalance(acct_2) @ b';
4
5    if (bal_1==fail) && (bal_2==fail) then
6       return fail;
7    else if (bal_1==fail) then
8       return acct_2;
9    else if (bal_2==fail) then
10      return acct_1;
11
12   if (bal_1 > bal_2) then
13       return acct_1;
14   else
15       return acct_2;
```

Fig. 3. Available largest balance

bob (or an attacker in control of bob's node) the ability to unilaterally choose the return value of the function, even when `alice` and `carol` agree on a different value.

2.2. *Using best available services*

Real world applications often consist of communication between entities with mutual distrust. The pseudocode in Figure 3 communicates with two banks, represented by `b` and `b'`, during a distributed computation. A user has two accounts `acc_1`, and `acc_2` with `b` and `b'` respectively. The user has linked both accounts to a service and specifies the bill should be paid

(1) as long as at least one account is available
(2) using the highest-balance account, if available

Lines 7-10 take care of point(1), ensuring the comparison on line 12 does not get stuck if a `fetch` returns `fail`. Lines 12-15 cover point (2), returning the account with the highest balance when both balances are available.

This example shows how availability of data can effect the final result of an application and thus highlights the importance of enforcing availability in distributed computations. As in the previous example, the programmer must reason about failures due to unavailable hosts and make the correct comparisons to implement the (implicitly) desired policy. Furthermore, the programmer may be unaware of the availability guarantees offered by `b` and `b'`. For example, if `b` and `b'` rely on the same replicas to implement `getBalance`, the availability of `highestBalance` may be lower than expected.

Finally, in both of the above examples, an attacker should not be able to read an account balance, or infer which account balance was greater. With the FLAQR type-system, programmers can not only specify and enforce availability and integrity, but also confidentiality—crucial for dealing with sensitive information. Moreover, FLAQR enables programmers to write fault-tolerant code concisely, with explicit primitives for consensus and replication operations that clarify the programmer's intentions.

3. Specifying FLAQR policies

FLAQR policies are specified using an extension of the FLAM [6, 8] principal algebra that includes availability policies.[1] FLAM principals represent both the *authority* of entities in a system as well as bounds on the *information flow policies* that authority entails. For example, Alice's authority is represented by the principal $\texttt{alice}$. *Authority projections* allow us to refer to specific categories of Alice's authority. The principal $\texttt{alice}^{\texttt{c}}$ refers to Alice's confidentiality authority: what Alice may read. Principal $\texttt{alice}^{\texttt{i}}$ refers to Alice's integrity authority: what Alice may write or influence.[2] Principal $\texttt{alice}^{\texttt{a}}$ refers to her availability authority: what Alice may cause to *fail*. A principal always acts for any projection of its authority, so for example $\texttt{alice} \succcurlyeq \texttt{alice}^{\texttt{a}}$. We refer to the set of all *primitive principals* such as $\texttt{alice}$ and $\texttt{bob}$ as $\mathcal{N}$.

We can write the conjunction of two principals with the Boolean connective $\wedge$ as $\texttt{alice} \wedge \texttt{bob}$, denoting the combined authority of Alice and Bob. Put another way, $\texttt{alice} \wedge \texttt{bob}$ is a principal both Alice and Bob *trust*. The disjunction of two principals' authority is written using the connective $\vee$ as $\texttt{alice} \vee \texttt{bob}$. This is a principal whose authority is less than both Alice and Bob; either Alice or Bob can act on behalf of the principal $\texttt{alice} \vee \texttt{bob}$. Put another way $\texttt{alice} \vee \texttt{bob}$ is a principal that trusts both Alice and Bob. Authority projections distribute over $\wedge$ and $\vee$, so for example $(\texttt{alice} \wedge \texttt{bob})^{\texttt{i}} \equiv \texttt{alice}^{\texttt{i}} \wedge \texttt{bob}^{\texttt{i}}$.

The confidentiality, integrity, and availability authorities make up the totality of a principal's authority, so writing $\texttt{alice}^{\texttt{c}} \wedge \texttt{alice}^{\texttt{i}} \wedge \texttt{alice}^{\texttt{a}}$ is equivalent to writing $\texttt{alice}$. For brevity, we sometimes write $\texttt{alice}^{\texttt{ci}}$ as a shorthand for $\texttt{alice}^{\texttt{c}} \wedge \texttt{alice}^{\texttt{i}}$ when we wish to include all but one kind of authority. Our complete formalization of the acts-for relation is presented in Figures 39 and 40 in Appendix A.

In addition to conjunctions and disjunctions of authority, FLAQR also introduce two new operators: *partial conjunction* ($\boxplus$), and *partial disjunction* ($\boxminus$). These operations are necessary to represent the tradeoffs between integrity and availability mediated by consensus and replication. Consider the "more integrity" protocol from Figure 1b. It is reasonable to think of the consensus value v as having more integrity than (or at least, "not less integrity than") Alice or Bob alone, but it turns out to be useful to distinguish between this authority and the combined integrity authority of Alice and Bob, $(\texttt{alice}^{\texttt{i}} \wedge \texttt{bob}^{\texttt{i}})$. A principal with integrity authority $(\texttt{alice}^{\texttt{i}} \wedge \texttt{bob}^{\texttt{i}})$ may act arbitrarily on behalf of both Alice and Bob since it is trusted by them. In contrast, the integrity of the value produced in Figure 1b is *not* fully trusted by Alice and Bob. Instead, Alice and Bob only trust the value when Alice and Bob agree on it. If they do not agree, that trust is revoked and no value is produced. For this reason, we describe the integrity of consensus values such as v as the *partial conjunction* of Alice and Bob, written $\texttt{alice}^{\texttt{i}} \boxplus \texttt{bob}^{\texttt{i}}$.

Similarly, for replication protocols like that in Figure 1a, we want to distinguish the integrity of values that may have been received from either Alice or Bob due to failure, from the integrity of values that may have been influenced by both Alice and Bob: $\texttt{alice}^{\texttt{i}} \vee \texttt{bob}^{\texttt{i}}$. The integrity of a value produced by either Alice or Bob is written as the *partial disjunction* $\texttt{alice}^{\texttt{i}} \boxminus \texttt{bob}^{\texttt{i}}$. This principal does not have the same integrity authority as Alice or Bob alone since we cannot guarantee which host's value will be used in the event of a failure. However, the value does have more integrity than $\texttt{alice}^{\texttt{i}} \vee \texttt{bob}^{\texttt{i}}$, since *only* Alice or Bob (and not both) may have influenced it.

We compare the authority of principals using the *acts-for* relation $\succcurlyeq$, which partially orders (equivalence classes of) principals by increasing authority. We form the set of all principals $\mathcal{P}$ as the closure

[1] Specifically, we extend the static fragment of FLAM's principal algebra defined by FLAC [8].

[2] Prior FLAM-based formalizations have used $\rightarrow$ and $\leftarrow$ for confidentiality and integrity, respectively.

$$[\textsc{PAndL}]\ \frac{\Pi \Vdash p_i \succcurlyeq p \quad k \in \{1,2\}}{\Pi \Vdash p_1 \boxplus p_2 \succcurlyeq p} \qquad [\textsc{PAndR}]\ \frac{\Pi \Vdash p \succcurlyeq p_1 \quad \Pi \Vdash p \succcurlyeq p_2}{\Pi \Vdash p \succcurlyeq p_1 \boxplus p_2} \qquad [\textsc{AndPAnd}]\ \Pi \Vdash p \wedge q \succcurlyeq p \boxplus q$$

$$[\textsc{PAndPOr}]\ \Pi \Vdash p \boxplus q \succcurlyeq p \boxminus q \qquad [\textsc{ProjPAndL}]\ \Pi \Vdash p^\pi \boxplus q^\pi \succcurlyeq (p \boxplus q)^\pi$$

$$[\textsc{ProjPAndR}]\ \Pi \Vdash (p \boxplus q)^\pi \succcurlyeq p^\pi \boxplus q^\pi \qquad [\textsc{ProjPOrL}]\ \Pi \Vdash p^\pi \boxminus q^\pi \succcurlyeq (p \boxminus q)^\pi$$

$$[\textsc{ProjPOrR}]\ \Pi \Vdash (p \boxminus q)^\pi \succcurlyeq p^\pi \boxminus q^\pi \qquad [\textsc{POrOr}]\ \Pi \Vdash p \boxminus q \succcurlyeq p \vee q$$

Fig. 4. Selected acts-for rules for partial conjunction and disjunction.

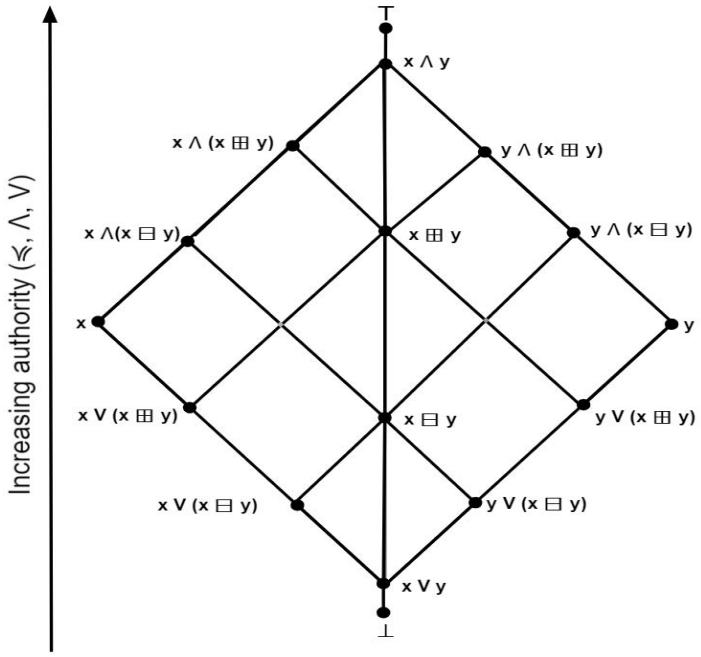


Fig. 5. The FLAQR authority lattice for the principal set $\{\bot, x, y, \top\}$.

of the set $\mathcal{N} \cup \{\top, \bot\}$ over the operations $\wedge$, $\vee$, $\boxplus$, $\boxminus$, and authority projections `c`, `i`, and `a`. We say Alice acts for Bob (or equivalently, Bob trusts Alice) and write `alice` $\succcurlyeq$ `bob` when Alice has at least as much authority as Bob. The $\succcurlyeq$ relation forms a lattice with join $\wedge$, meet $\vee$, greatest element $\top$, and least element $\bot$.

In addition to the trust relationships such as $p \wedge q \succcurlyeq p$ and $p \succcurlyeq p^{\mathrm{i}}$ implied by the principal algebra, explicit delegations of trust such as $p \succcurlyeq q$ (for any p, q in $\mathcal{P}$) may expressed using a *delegation context* Π. An acts-for judgment has the form $\Pi \Vdash p \succcurlyeq q$ and means that p acts for q in context Π. While FLAC has a feature that allows dynamic extensions of Π, for simplicity we fix Π to a static set of delegations in FLAQR.

We extend the acts-for relation defined by Arden et al. [8] with new rules for availability authority and partial conjunction and disjunction. Figure 4 presents a selection of these rules—we have omitted the distributivity rules for brevity. In Figure 4, an *acts for* judgement of form $\Pi \Vdash p \succcurlyeq q$, states that p has at least as much authority as q in delegation context Π. Figures 39 and 40 in Appendix A together present the complete formalization of the $\succcurlyeq$ relation. In Figure 40 we present only the extensions to this relation introduced by FLAQR.

As a consequence of these new acts-for rules we have additional distinct points in the authority lattice. Figure 5 illustrates the authority sublattice over elements $\{\bot, x, y, \top\}$. Figure 5 shows the trust ordering of all possible distinct combinations of elements that can be formed on the set $\{\bot, x, y, \top\}$ with operations $\wedge$, $\vee$, $\boxplus$ and $\boxminus$ over them. The relationship between principals $\bot, x, y, x \wedge y, x \vee y$, and $\top$ is the same as in FLAM, but Figure 5 also includes principals constructed using partial conjunctions and disjunctions. For example, $x \wedge (x \boxplus y)$ is the least upper bound of $x \wedge (x \boxminus y)$ and $x \boxplus y$. This is due to rule PANDPOR in Figure 4, which lets us simplify $x \wedge (x \boxminus y) \wedge x \boxplus y$ to $x \wedge (x \boxplus y)$.

To compare the restrictiveness of information flow policies, we use the *flows-to* relation $\sqsubseteq$, which partially orders principals by increasing policy restrictiveness, rather than by authority. For example, we say Alice's integrity flows to Bob's integrity and write $\mathtt{alice}^{\mathsf{i}} \sqsubseteq \mathtt{bob}^{\mathsf{i}}$ if Bob trusts information influenced by Alice at least as much as information he influenced himself. Likewise, we write $\mathtt{alice}^{\mathsf{c}} \sqsubseteq \mathtt{bob}^{\mathsf{c}}$ if Alice trusts Bob to protect the confidentiality of her information, and $\mathtt{alice}^{\mathsf{a}} \sqsubseteq \mathtt{bob}^{\mathsf{a}}$ if Bob is trusted to keep Alice's data available. The flows-to relation behaves similarly to a sub-typing relation. Treating information labeled $\mathtt{alice}^{\mathsf{cia}}$ (i.e. alice) as though it was labeled $\mathtt{bob}^{\mathsf{cia}}$ (i.e. bob) is only safe (doesn't violate anyone's policies) if $\mathtt{alice}^{\mathsf{cia}} \sqsubseteq \mathtt{bob}^{\mathsf{cia}}$ (i.e. $\mathtt{alice} \sqsubseteq \mathtt{bob}$).

One advantage of the FLAM principal algebra is that we can define the flows-to relation, as well as the upper and lower bounds of information flow policies, in terms of the acts-for relation, simplifying our formalism.

$$p \sqsubseteq q \;\Leftrightarrow q^{\mathsf{c}} \succcurlyeq p^{\mathsf{c}} \text{ and } p^{\mathsf{i}} \succcurlyeq q^{\mathsf{i}} \text{ and } p^{\mathsf{a}} \succcurlyeq q^{\mathsf{a}}$$

$$p \sqcup q \triangleq (p^{\mathsf{c}} \wedge q^{\mathsf{c}}) \wedge (p^{\mathsf{i}} \vee q^{\mathsf{i}}) \wedge (p^{\mathsf{a}} \vee q^{\mathsf{a}})$$

$$p \sqcap q \triangleq (p^{\mathsf{c}} \vee q^{\mathsf{c}}) \wedge (p^{\mathsf{i}} \wedge q^{\mathsf{i}}) \wedge (p^{\mathsf{a}} \wedge q^{\mathsf{a}})$$

Based on this, the equivalence classes of $\succcurlyeq$ and $\sqsubseteq$ are identical, meaning that the lattice formed by $\sqsubseteq$ with joins $\sqcup$ and meets $\sqcap$ has the same elements as the acts-for lattice. A flow from p to q is secure only when q^{c} is at least as confidential as p^{c}, q^{i} trusts information influenced by p^{i}, and q^{a} cannot cause failures that p^{a} cannot.

4. FLAQR syntax and semantics

Figures 6 and 7 present the FLAQR syntax and selected evaluation rules. For space and exposition purposes, we omit some term annotations and standard lambda calculus rules in order to focus on FLAQR's contributions, but the complete, annotated FLAQR syntax and semantics can be found in Figures 29 and 31 in the Appendix.

FLAQR is based on FLAC [8, 9], a monadic calculus in the style of Abadi's Polymorphic DCC [10]. In addition to standard extensions to System F [11–13] such as pairs and tagged unions, an Abadi-style calculus supports monadic operations on values in a monad indexed by a lattice of security labels. Such

$\pi \in \{\mathsf{c}, \mathsf{i}, \mathsf{a}\}$ (projections)
$n \in \mathcal{N}$ (primitive principals)
$x \in \mathcal{V}$ (variable names)

$$
\begin{aligned}
p, \ell, pc ::= {} & n \mid \top \mid \bot \mid p^{\pi} \mid p \wedge p \mid p \vee p \\
& \mid p \sqcup p \mid p \sqcap p \mid p \boxminus p \mid p \boxplus p \\
\tau ::= {} & \mathsf{unit} \mid X \mid (\tau + \tau) \mid (\tau \times \tau) \\
& \mid \tau \xrightarrow{pc} \tau \mid \forall X[pc].\, \tau \mid \ell \ \mathsf{says}\ \tau \\
v ::= {} & () \mid \underline{(\overline{\eta}_{\ell}\ v)} \mid \mathsf{inj}_i^{(\tau+\tau)}\ v \mid \langle v, v \rangle^{\tau} \\
& \mid \lambda(x{:}\tau)[pc].\, e \mid \Lambda X[pc].\, e \\
f ::= {} & v \mid \underline{\mathtt{fail}^{\tau}} \\
e ::= {} & f \mid x \mid e\ e \mid e\ \tau \mid \eta_{\ell}\ e \mid \langle e, e \rangle^{\tau} \\
& \mid \mathsf{proj}_i\ e \mid \mathsf{inj}_i^{(\tau+\tau)}\ e \mid \mathsf{bind}\ x = e\ \mathsf{in}\ e \\
& \mid \mathsf{case}^{\tau}\ e\ \mathsf{of}\ \mathsf{inj}_1^{\tau}(x).e \mid \mathsf{inj}_2^{\tau}(x).e \\
& \mid \mathtt{run}^{\tau}\ e@p \mid \underline{\mathtt{ret}\ e@p} \mid \underline{\mathtt{expect}^{\tau}} \\
& \mid \mathtt{select}^{\tau}\ e\ \mathsf{or}\ e \mid \mathtt{compare}^{\tau}\ e\ \mathsf{and}\ e
\end{aligned}
$$

Fig. 6. FLAQR Syntax. Shaded terms are new to FLAQR. Underlined terms are used during evaluation and not available at the source level.

a value has a type of the form ℓ says τ, meaning that it is a value of type τ, *protected* at level ℓ, where ℓ is an element of the security lattice. Here we focus on FLAQR's additions to FLAC and DCC, and refer the readers to Figures 33 and 34 in the Appendix for our complete formalization.

FLAQR builds on FLAC's expressive principal algebra and type system to model distributed security policies for applications that use replication and consensus. FLAC supports arbitrary policy downgrades through dynamic delegations of authority, but for simplicity we omit these features in FLAQR.

The monadic unit or return term $\eta_{\ell}\ e$ protects the value that e evaluates to at level ℓ (E-SEALED).[3] Protected values, $(\overline{\eta}_{\ell}\ v)$ cannot be operated on directly. Instead, a bind expression must be used to bind the protected value to a variable whose scope is limited to the body of the bind term (E-BINDM). The body performs the desired computation and "returns" the result to the monad, ensuring the result is protected. These rules (E-SEALED and E-BINDM) FLAQR inherits from FLAC. The remaining rules of Figure 7 are specific to FLAQR.

The primary novelty in the FLAQR calculus is the introduction of `compare` and `select` terms for expressing consensus and replication operations. We represent the consensus problem as a comparison of two values with the same underlying type but distinct outer security labels. In other words, we want to check the equality of values produced by two different principals. If the values match, we can treat

[3]Polymorphic DCC does not define a term similar to $(\overline{\eta}_{\ell}\ v)$ and thus does not have an rule equivalent to E-SEALED. This approach enables us to distinguish where a value may be created (e.g., on a host authorized to read and create values protected at ℓ) and use more permissive rules to control where a sealed value may flow.

[E-SEALED] $\eta_\ell\ v \longrightarrow (\overline{\eta}_\ell\ v)$ [E-BINDM] $\mathsf{bind}\ x = (\overline{\eta}_\ell\ v)\ \mathsf{in}\ e \longrightarrow e[x \mapsto v]$

[E-COMPARE] $$\mathsf{compare}\ (\overline{\eta}_{\ell_1}\ v)\ \mathsf{and}\ (\overline{\eta}_{\ell_2}\ v) \longrightarrow (\overline{\eta}_{\ell_1 \oplus \ell_2}\ v)$$

[E-COMPAREFAIL] $$\frac{v_1 \neq v_2 \qquad \tau = (\ell_1 \oplus \ell_2)\ \mathsf{says}\ \tau'}{\mathsf{compare}\ (\overline{\eta}_{\ell_1}\ v_1)\ \mathsf{and}\ (\overline{\eta}_{\ell_2}\ v_2) \longrightarrow \mathsf{fail}^{\tau}}$$

[E-COMPAREFAILL] $$\frac{\tau_1 = \ell_1\ \mathsf{says}\ \tau \qquad \tau' = (\ell_1 \oplus \ell_2)\ \mathsf{says}\ \tau \qquad f_2 = \begin{cases} (\overline{\eta}_{\ell_2}\ v) \\ \mathsf{fail}^{\ell_2\ \mathsf{says}\ \tau} \end{cases}}{\mathsf{compare}\ (\mathsf{fail}^{\tau_1})\ \mathsf{and}\ f_2 \longrightarrow \mathsf{fail}^{\tau'}}$$

[E-COMPAREFAILR] $$\frac{\tau_2 = \ell_2\ \mathsf{says}\ \tau \qquad \tau' = (\ell_1 \oplus \ell_2)\ \mathsf{says}\ \tau}{\mathsf{compare}\ (\overline{\eta}_{\ell_1}\ v)\ \mathsf{and}\ \mathsf{fail}^{\tau_2} \longrightarrow \mathsf{fail}^{\tau'}}$$

[E-SELECT] $$\mathsf{select}\ (\overline{\eta}_{\ell_1}\ v_1)\ \mathsf{or}\ (\overline{\eta}_{\ell_2}\ v_2) \longrightarrow (\overline{\eta}_{\ell_1 \ominus \ell_2}\ v_1)$$

[E-SELECTL] $$\mathsf{select}\ (\overline{\eta}_{\ell_1}\ v)\ \mathsf{or}\ (\mathsf{fail}^{\ell_1\ \mathsf{says}\ \tau}) \longrightarrow (\overline{\eta}_{\ell_1 \ominus \ell_2}\ v)$$

[E-SELECTFAIL] $$\frac{\forall i \in \{1,2\} \qquad \tau_i = \ell_i\ \mathsf{says}\ \tau \qquad \tau' = (\ell_1 \ominus \ell_2)\ \mathsf{says}\ \tau}{\mathsf{select}\ (\mathsf{fail}^{\tau_1})\ \mathsf{or}\ (\mathsf{fail}^{\tau_2}) \longrightarrow \mathsf{fail}^{\tau'}}$$

[E-RETSTEP] $$\frac{e \longrightarrow e'}{\mathsf{ret}\ e@c \longrightarrow \mathsf{ret}\ e'@c}$$ [E-STEP] $$\frac{e \longrightarrow e'}{E[e] \longrightarrow E[e']}$$

Fig. 7. FLAQR local semantics

$$\begin{aligned} E ::= {} & [\cdot] \mid E\ e \mid v\ E \mid \eta_\ell\ E \mid \mathsf{bind}\ x = E\ \mathsf{in}\ e \\ & \mid \mathsf{ret}\ E@p \mid \mathsf{select}\ E\ \mathsf{or}\ e \mid \mathsf{select}\ f\ \mathsf{or}\ E \\ & \mid \mathsf{compare}\ E\ \mathsf{and}\ e \mid \mathsf{compare}\ f\ \mathsf{and}\ E \end{aligned}$$

Fig. 8. FLAQR evaluation context

them as having the (partially) combined integrity of the principals. If not, then the principals failed to reach consensus.

Rule E-COMPARE defines the former case: two syntactically equal values protected at different labels evaluate to a value that combines labels using the *compare action* on labels $\oplus$. Intuitively, $\ell_1 \oplus \ell_2$ determines the increase in integrity and the corresponding decrease in availability inherent in requiring a consensus. We define $\oplus$ formally in Definition 1.

[E-AppFail] $\lambda(x{:}\tau)[pc].\, e\ \texttt{fail}^{\tau} \longrightarrow e[x \mapsto \texttt{fail}^{\tau}]$ [E-SealedFail] $\eta_{\ell}\ \texttt{fail}^{\tau} \longrightarrow \texttt{fail}^{\ell\ \texttt{says}\ \tau}$

[E-InjFail] $\texttt{inj}_i^{(\tau_1+\tau_2)}\ \texttt{fail}^{\tau_i} \longrightarrow \texttt{fail}^{(\tau_1+\tau_2)}$ [E-ProjFail] $\texttt{proj}_i\ \texttt{fail}^{(\tau_1\times\tau_2)} \longrightarrow \texttt{fail}^{\tau_i}$

Fig. 9. `fail` propagation rules.

Definition 1 (Compare action on principals)**.**

$$\ell_1 \oplus \ell_2 \triangleq (\ell_1^{\mathsf{c}} \wedge \ell_2^{\mathsf{c}}) \wedge (\ell_1^{\mathsf{i}} \boxplus \ell_2^{\mathsf{i}}) \wedge (\ell_1^{\mathsf{a}} \vee \ell_2^{\mathsf{a}})$$

We also lift this notation to `says` types by defining

$$\ell_1\ \texttt{says}\ \tau \oplus \ell_2\ \texttt{says}\ \tau \triangleq (\ell_1 \oplus \ell_2)\ \texttt{says}\ \tau$$

As discussed in Section 3, the integrity authority of `compare` is not as trusted as the conjunction of ℓ_1 and ℓ_2's integrity. Instead, we represent the limited "increase" in integrity authority[4] using a partial conjunction in Definition 1. In contrast, the decrease in availability is represented by a (full) $\ell_1^{\mathsf{a}} \vee \ell_2^{\mathsf{a}}$ since either ℓ_1 or ℓ_2 could unilaterally cause the compare expression to fail.

The decreased availability resulting from applying `compare` is more apparent in rules E-CompareFail, E-CompareFailL and E-CompareFailR. In E-CompareFail, two unequal values are compared, resulting in a failure. Failure is represented syntactically using a $\texttt{fail}^{\tau}$ term. We use a type annotation τ on many terms in our formal definitions so that our semantics is well defined with respect to failure terms, but we omit most of these annotations in Figure 7. These annotations are only necessary for our formalization and would be unnecessary in a FLAQR implementation.

A `compare` term may also result in failure if either subexpression fails. Rule E-CompareFailL and E-CompareFailR, defines how failure of an input propagates to the output. In fact, most FLAQR terms result in failure when a subexpression fails. Figure 9 presents selected failure propagation rules (complete failure propagation rules are presented in Figure 32). Note that `fail` terms are treated similarly to values, but are distinct from them. For example, in E-AppFail, applying a lambda term to a fail term substitutes the failure as it would a value, but in E-SealedFail the failure is propagated beyond the monadic unit term. This latter behavior captures the idea that failures cannot be hidden or isolated in the same way as secrets or untrusted data.

Failures are tolerated using replication. A `select` term will evaluate to a value as long as at least one of its subexpressions does not fail. For example, rule E-SelectL returns its left subexpression when the right subexpression fails. In contrast to `compare`, applying `select` increases availability since either subexpression can be used, but reduces integrity since influencing only one of the subexpressions is potentially sufficient to influence the result of evaluating `select`. The effect of a `select` statement on the labels of its sub-expressions is captured with the *select action* $\ominus$.

Definition 2 (Select action on principals)**.**

$$\ell_1 \ominus \ell_2 \triangleq (\ell_1^{\mathsf{c}} \wedge \ell_2^{\mathsf{c}}) \wedge (\ell_1^{\mathsf{i}} \boxminus \ell_2^{\mathsf{i}}) \wedge (\ell_1^{\mathsf{a}} \wedge \ell_2^{\mathsf{a}})$$

[4] Strictly speaking, $x \boxplus y$ is not an increase in integrity over x (or y); $x \boxplus y$ and x are incomparable.

$$[\text{E-DSTEP}]\ \frac{e \longrightarrow e'}{\langle E[e], c\rangle \,\&\, t \Longrightarrow \langle E[e'], c\rangle \,\&\, t}$$

$$[\text{E-RUN}]\ \langle E[\mathtt{run}^{\tau}\ e@c'], c\rangle \,\&\, t \Longrightarrow \langle \mathtt{ret}\ e@c, c'\rangle \,\&\, \langle E[\mathtt{expect}^{\tau}], c\rangle :: t$$

$$[\text{E-RETV}]\ \langle \mathtt{ret}\ v@c, c'\rangle \,\&\, \langle E[\mathtt{expect}^{pc^{\mathrm{ia}}}\ \mathtt{says}\ \tau'], c\rangle :: t \Longrightarrow \langle E[(\overline{\eta}_{pc^{\mathrm{ia}}}\ v)], c\rangle \,\&\, t$$

$$[\text{E-RETFAIL}]\ \langle \mathtt{ret}\ (\mathtt{fail}^{\tau'})@c, c'\rangle \,\&\, \langle E[\mathtt{expect}^{pc^{\mathrm{ia}}}\ \mathtt{says}\ \tau'], c\rangle :: t \Longrightarrow \langle E[\mathtt{fail}^{pc^{\mathrm{ia}}}\ \mathtt{says}\ \tau'], c\rangle \,\&\, t$$

Fig. 10. Global semantics

$$s ::= \langle e, c\rangle \,\&\, t$$
$$t ::= \mathtt{empty} \mid \langle E[\mathtt{expect}^{\tau}], c\rangle :: t$$

Fig. 11. Global configuration stack

We define the select action on types similarly to compare:

$$\ell_1\ \mathtt{says}\ \tau \ominus \ell_2\ \mathtt{says}\ \tau = (\ell_1 \ominus \ell_2)\ \mathtt{says}\ \tau$$

The end result of a *select* statement, $\mathtt{select}\ (\overline{\eta}_{\ell_1}\ v)\ \mathtt{or}\ (\overline{\eta}_{\ell_2}\ v)$, will have integrity of either ℓ_1^{i} or ℓ_2^{i} since only one of the two possible values will be used. We use a partial disjunction to represent this integrity since the result does not have the same integrity as ℓ_1 or ℓ_2, but does have more integrity than $\ell_1 \vee \ell_2$ since it is never the case that *both* principals influence the output.

4.1. Global semantics

We capture the distributed nature of quorum replication by embedding the local semantic rules within a global distributed semantics defined in Figure 10. This semantics uses a *configuration stack* $s = \langle e, c\rangle \,\&\, t$ (Figure 11) to keep track of the currently executing expression e, the host on which it is executing c, and the remainder of the stack t. We also make explicit use of the evaluation contexts from Figure 7 to identify the reducible subterms across stack elements.

The core operation for distributed computation is $\mathtt{run}^{\tau}\ e@p$ which runs the computation e of type τ on node p. Local evaluation steps are captured in the global semantics via rule E-DSTEP. This rule says that if e steps to e' locally, then $E[e]$ steps to $E[e']$ globally.

Rule E-RUN takes an expression e at host c, pushes a new configuration on the stack containing e at host c' and places an `expect` term at c as a place holder for the return value.

Once the remote expression is fully evaluated, rule E-RETV pops the top configuration off the stack and replaces the `expect` term at c with the protected value $(\overline{\eta}_{pc^{\mathrm{ia}}}\ v)$. Rule E-RETFAIL serves the same purpose for `fail` terms, but is necessary since `fail` terms are not considered values (see Figure 6). The label pc^{ia} reflects both the integrity and availability context of the caller (c) as well as the integrity and availability of the remote host (c'). We discuss this aspect of remote execution in more detail in Section 5.

$\boxed{\Pi; \Gamma; pc; c \vdash e : \tau}$

$$[\textsc{Unit}]\ \frac{\Pi \Vdash c \succcurlyeq pc}{\Pi; \Gamma; pc; c \vdash () : \mathtt{unit}} \qquad [\textsc{Fail}]\ \frac{\Pi \Vdash c \succcurlyeq pc}{\Pi; \Gamma; pc; c \vdash \mathtt{fail}^{\tau} : \tau} \qquad [\textsc{Expect}]\ \frac{\Pi \Vdash c \succcurlyeq pc}{\Pi; \Gamma; pc; c \vdash \mathtt{expect}^{\tau} : \tau}$$

$$[\textsc{Lam}]\ \frac{\Pi; \Gamma, x{:}\tau_1; pc'; u \vdash e : \tau_2 \qquad \Pi \Vdash c \succcurlyeq pc \qquad u = \mathtt{C}(\tau_1 \xrightarrow{pc'} \tau_2) \qquad \Pi \Vdash c \succcurlyeq u}{\Pi; \Gamma; pc; c \vdash \lambda(x{:}\tau_1)[pc'].\, e : \tau_1 \xrightarrow{pc'} \tau_2}$$

$$[\textsc{App}]\ \frac{\Pi; \Gamma; pc; c \vdash e_1 : \tau' \xrightarrow{pc'} \tau \qquad \Pi; \Gamma; pc; c \vdash e_2 : \tau' \qquad \Pi \Vdash pc \sqsubseteq pc' \qquad \Pi \Vdash c \succcurlyeq pc}{\Pi; \Gamma; pc; c \vdash e_1\ e_2 : \tau}$$

$$[\textsc{UnitM}]\ \frac{\Pi; \Gamma; pc; c \vdash e : \tau \qquad \Pi \Vdash pc \sqsubseteq \ell \qquad \Pi \Vdash c \succcurlyeq pc}{\Pi; \Gamma; pc; c \vdash \eta_{\ell}\ e : \ell\ \mathtt{says}\ \tau}$$

$$[\textsc{Sealed}]\ \frac{\Pi; \Gamma; pc; c \vdash v : \tau \qquad \Pi \Vdash c \succcurlyeq pc}{\Pi; \Gamma; pc; c \vdash (\overline{\eta}_{\ell}\ v) : \ell\ \mathtt{says}\ \tau}$$

$$[\textsc{BindM}]\ \frac{\Pi; \Gamma; pc; c \vdash e' : \ell\ \mathtt{says}\ \tau' \qquad \Pi \Vdash \ell \sqcup pc \sqsubseteq \tau \qquad \Pi; \Gamma, x{:}\tau'; \ell \sqcup pc; c \vdash e : \tau \qquad \Pi \Vdash c \succcurlyeq pc}{\Pi; \Gamma; pc; c \vdash \mathtt{bind}\ x = e'\ \mathtt{in}\ e : \tau}$$

$$[\textsc{Run}]\ \frac{\Pi; \Gamma; pc'; c' \vdash e : \tau' \qquad \Pi \Vdash pc \sqsubseteq pc' \qquad \Pi \Vdash c \succcurlyeq pc \qquad \Pi \Vdash c \succcurlyeq \mathtt{C}(\tau') \qquad \tau = pc'^{\mathtt{ia}}\ \mathtt{says}\ \tau'}{\Pi; \Gamma; pc; c \vdash \mathtt{run}^{\tau}\ e@c' : \tau}$$

$$[\textsc{Ret}]\ \frac{\Pi; \Gamma; pc; c \vdash e : \tau \qquad \Pi \Vdash c' \succcurlyeq \mathtt{C}(\tau) \qquad \Pi \Vdash c \succcurlyeq pc}{\Pi; \Gamma; pc; c \vdash \mathtt{ret}\ e@c' : pc^{\mathtt{ia}}\ \mathtt{says}\ \tau}$$

$$[\textsc{Compare}]\ \frac{\forall i \in \{1,2\}.\Pi; \Gamma; pc; c \vdash e_i : \ell_i\ \mathtt{says}\ \tau \qquad \Pi \Vdash c \rhd \ell_i\ \mathtt{says}\ \tau \qquad \Pi \Vdash c \succcurlyeq pc}{\Pi; \Gamma; pc; c \vdash \mathtt{compare}\ e_1\ \mathtt{and}\ e_2 : (\ell_1 \oplus \ell_2)\ \mathtt{says}\ \tau}$$

$$[\textsc{Select}]\ \frac{\forall i \in \{1,2\}.\Pi; \Gamma; pc; c \vdash e_i : \ell_i\ \mathtt{says}\ \tau \qquad \Pi \Vdash c \succcurlyeq pc}{\Pi; \Gamma; pc; c \vdash \mathtt{select}\ e_1\ \mathtt{or}\ e_2 : (\ell_1 \ominus \ell_2)\ \mathtt{says}\ \tau}$$

Fig. 12. Typing rules for expressions

5. FLAQR typing rules

As we have a local and global semantics, we have two corresponding forms of typing judgements: local typing judgments for expressions and global typing judgments for the stack. Local typing judgments have the form $\Pi; \Gamma; pc; c \vdash e : \tau$. Π is the program's delegation context and is used to derive acts-for relationships with the rules in Figures 4 and 40. Γ is the typing context containing in-scope variable names and their types. The *pc* label tracks the information flow policy on the program counter (due to control flow) and on unsealed protected values such as in the body of a `bind`.

Figure 12 presents a selection of local typing rules. Each typing rule includes an acts-for premise of the form $\Pi \Vdash c \succcurlyeq pc$. This enforces the invariant that each host principal c has control of the program it executes locally. Thus for any judgment $\Pi; \Gamma; pc; c \vdash e : \tau$ *pc* should never exceed the authority of c, the principal executing the expression. Rules FAIL and EXPECT type `fail` and `expect` terms according

to their type annotation τ. Rule LAM types lambda abstractions. Since functions are first-class values, we have to ensure that the *pc* annotation on the lambda term preserves the invariant $\Pi \Vdash c \geqslant pc$. The *clearance* of a type τ, written $\mathtt{C}(\tau)$, is an upper bound on the *pc* annotations of the function types in τ. By checking that $\Pi \Vdash c \geqslant \mathtt{C}(\tau_1 \xrightarrow{pc'} \tau_2)$ holds (along with similar checks in RUN and RET), we ensure the contents of the lambda term is protected when sending or receiving lambda expressions, and that hosts never receive a function they cannot securely execute. Due to space constraints, the definition of $\mathtt{C}(\cdot)$ is presented in Appendix A, in Figure 35. The APP rule requires the *pc* label at any function application to flow to the function's *pc* label annotation. Hence the premise $\Pi \Vdash pc \sqsubseteq pc'$.

Protected terms $\eta_\ell\ e$ are typed by rule UNITM as ℓ `says` τ where τ is the type of e. Additionally, it requires that $\Pi \Vdash pc \sqsubseteq \ell$. This ensures that any unsealed values in the context are adequately protected by policy ℓ if they are used by e. The SEALED rule types protected values $(\overline{\eta}_\ell\ v)$. These values are well-typed at any host, and does not require $\Pi \Vdash pc \sqsubseteq \ell$ since no unsealed values in the context could be captured by the (closed) value v.

Computation on protected values occurs in bind terms `bind` $x = e'$ `in` e. The policy protecting e must be at least as restrictive as the policy on e' so that the occurrences of x in e are adequately protected. Thus, rule BINDM requires $\Pi \Vdash \ell \sqcup pc \sqsubseteq \tau$, and furthermore e is typed at a more restrictive program counter label $\ell \sqcup pc$ to reflect the dependency of e on the value bound to x.

Rule RUN requires that the *pc* at the local host flow to the pc' of the remote host, and that e be well-typed at c', which implies that c' acts for pc'. Additionally, c must act for the clearance of the remote return type τ' to ensure c is authorized to receive the return value. The type of the run expression is $pc'^{\mathtt{ia}}$ `says` τ', which reflects the fact that c' controls the availability of the return value and also has some influence on which value of type τ' is returned. Although c' may not be able to *create* a value of type τ' unless $pc'^{\mathtt{ia}}$ flows to τ', if c' has *access* to more than one value of type τ', it could choose which one to return. Rule RET requires that expression e is welltyped at c and that c' is authorized to receive the return value based on the clearance of τ.

The COMPARE rule gives type $(\ell_1 \oplus \ell_2)$ `says` τ to the expression `compare` e_1 `and` e_2 where e_1 and e_2 have types ℓ_1 `says` τ and ℓ_2 `says` τ respectively. Additionally, it requires that c, the host executing the `compare`, is authorized to fully examine the results of evaluating e_1 and e_2 so that they may be checked for equality.[5] This requirement is captured by the premise $\Pi \Vdash c \rhd \ell_i$ `says` τ, pronounced "*c reads* ℓ_i `says` τ". The inference rules for the reads judgment are found in Figure 37 in Appendix A.

Finally, the SELECT rule gives type $(\ell_1 \ominus \ell_2)$ `says` τ to the expression `select` e_1 `or` e_2 where e_1 and e_2 have types ℓ_1 `says` τ and ℓ_2 `says` τ respectively.

The typing judgment for the global configuration is presented in Figure 13 and consists of three rules. Rule HEAD shows that the global configuration $\langle e, c\rangle$ & t, is well-typed if the expression e is well-typed at host c with program counter pc' where $\Pi \Vdash pc \sqsubseteq pc'$ and the tail t is well-typed. $[\tau']\tau$ means that the tail of the stack is of type τ while the expression in the head of the configuration is of type τ'. We introduced rules TAIL(when $t \neq$ `empty`) and EMP(when $t =$ `empty`) to typecheck the tail t.

$\langle E[\mathtt{expect}^{\tau'}], c\rangle :: t$ is well-typed with type $[\tau']\tau$, if expression $E[\mathtt{expect}^{\tau'}]$ is well-typed with type $\hat{\tau}$ at host c. And, the rest of the stack t needs to be well-typed with type $[\hat{\tau}]\tau$. Rule EMP says the tail is empty and the type of the expression in the head of the configuration is τ, in which case the type of the whole stack is $[\tau]\tau$.

[5] Assuming a more sophisticated mechanism for checking equality that reveals less information to the host such as zero-knowledge proofs or a trusted execution environment could justify relaxing this constraint.

$\boxed{\Pi; \Gamma; pc \vdash \langle e, c\rangle \,\&\, t : \tau}$

$$[\textsc{Head}]\ \frac{\Pi; \Gamma; pc'; c \vdash e : \tau' \qquad \Pi; \Gamma; pc \vdash t : [\tau']\tau \qquad \Pi \Vdash pc \sqsubseteq pc' \qquad \Pi \Vdash c \geqslant pc}{\Pi; \Gamma; pc \vdash \langle e, c\rangle \,\&\, s : \tau}$$

$\boxed{\Pi; \Gamma; pc \vdash \langle e, c\rangle :: t : [\tau']\tau}$

$$[\textsc{Tail}]\ \frac{\Pi; \Gamma; pc'; c \vdash E[\texttt{expect}^{\tau'}] : \hat{\tau} \qquad \Pi; \Gamma; pc \vdash t : [\hat{\tau}]\tau \qquad \Pi \Vdash pc \sqsubseteq pc' \qquad \Pi \Vdash c \geqslant pc}{\Pi; \Gamma; pc \vdash \langle E[\texttt{expect}^{\tau'}], c\rangle :: t : [\tau']\tau}$$

$$[\textsc{Emp}]\ \Pi; \Gamma; pc \vdash \texttt{empty} : [\tau]\tau$$

Fig. 13. Typing rules for configuration stack

6. Availability Attackers

Availability attackers are different from traditional integrity and confidentiality attackers. While an integrity attacker's goal is to manipulate data and a confidentiality attacker's goal is to learn secrets, an availability attacker's goal is to cause failures. In our model, an availability attacker can substitute a value only with a `fail` term. Integrity attackers may also cause failures in consensus based protocols when consensus is not reached because of data manipulation. In FLAQR this scenario is relevant during executing a `compare` statement: if one of the values in the `compare` statement is substituted with a wrong (mismatching) value then a `fail` term is returned. Thus we need to consider an availability attacker's integrity authority when reasoning about its power to `fail` a program. Specifically, the authority of principal ℓ as an availability attacker is ℓ^{ia}.

We consider a static but active attacker model similar to those used in Byzantine consensus protocols. By static we mean which principal or collection of principals can act maliciously is fixed prior to program execution. By active we mean that the attackers may manipulate inputs (including higher-order functions) during run time. We formally define the power of an availability attacker with respect to quorum systems.

Availability attackers in FLAQR are somewhat different than integrity and confidentiality attackers because we want to represent multiple possible attackers but limit which attackers are active for a particular execution. This goal supports the bounded fault assumptions found in consensus protocols where system configurations assume an upper bound on the number of faults possible.

A quorum system $\mathcal{Q}$ is represented as set of sets of hosts (or principals) e.g. $\mathcal{Q} = \{q_1, q_2, \ldots, q_n\}$. Here each q_i represents a set of principals whose consensus is adequate for the system to make progress. We define availability attackers in terms of the *toleration set* $[\![\mathcal{Q}]\!]$ of a quorum system $\mathcal{Q}$. The toleration set is a set of principals where each principal represents an upper bound on the authority of an attacker the quorum can tolerate without failing.

Example.

(1) The toleration set for quorum $\mathcal{Q}_1 = \{q_1 := \{a, b\}; q_2 := \{b, c\}; q_3 := \{a, c\}\}$ is $[\![\mathcal{Q}_1]\!] = \{a^{\mathsf{ia}}, b^{\mathsf{ia}}, c^{\mathsf{ia}}\}$,
(2) For heterogeneous quorum system $\mathcal{Q}_2 = \{q_1 := \{p, q\}; q_2 := \{r\}\}$ the toleration set is $[\![\mathcal{Q}_2]\!] = \{p^{\mathsf{ia}} \wedge q^{\mathsf{ia}}, r^{\mathsf{ia}}\}$
(3) For $\mathcal{Q}_3 = \{q := \{alice\}\}$ the toleration set is $[\![\mathcal{Q}_3]\!] = \{\}$, i.e. $\mathcal{Q}_3$ can not tolerate any fault.

$$[\text{A-PAIR}]\ \frac{\Pi \Vdash \ell \succcurlyeq \tau_i \quad i \in \{1,2\}}{\Pi \Vdash \ell \succcurlyeq (\tau_1 \times \tau_2)} \qquad [\text{A-SUM}]\ \frac{\Pi \Vdash \ell \succcurlyeq \tau_i \quad i \in \{1,2\}}{\Pi \Vdash \ell \succcurlyeq (\tau_1 + \tau_2)} \qquad [\text{A-FUN}]\ \frac{\Pi \Vdash \ell \succcurlyeq \tau_2}{\Pi \Vdash \ell \succcurlyeq \tau_1 \xrightarrow{pc'} \tau_2}$$

$$[\text{A-TYPE}]\ \frac{\Pi \Vdash \ell \succcurlyeq \tau}{\Pi \Vdash \ell \succcurlyeq \ell' \ \mathsf{says}\ \tau} \qquad [\text{A-AVAIL}]\ \frac{\Pi \Vdash \ell^{\mathsf{a}} \succcurlyeq \ell'^{\mathsf{a}}}{\Pi \Vdash \ell \succcurlyeq \ell' \ \mathsf{says}\ \tau} \qquad [\text{A-INTEGCOM}]\ \frac{\Pi \Vdash \ell^{\mathsf{i}} \succcurlyeq {\ell_j}^{\mathsf{i}}, j \in \{1,2\}}{\Pi \Vdash \ell \succcurlyeq (\ell_1 \oplus \ell_2)\ \mathsf{says}\ \tau}$$

Fig. 14. fails judgments.

An availability attacker's authority is at most equivalent to a (single) principal's authority in the toleration set. We define the set of all such attackers for a quorum $\mathcal{Q}$ as

$$\mathcal{A}_{[\![\mathcal{Q}]\!]} = \{\ell \mid \exists \ell' \in [\![\mathcal{Q}]\!].\Pi \Vdash \ell' \succcurlyeq \ell\}.$$

which includes weaker attackers who a principal in the toleration set may act on behalf of.

The fails relation ($\succcurlyeq$) determines whether a principal can cause a program of a particular type to evaluate to `fail`. Similar to the reads judgment, the fails judgment not only considers the outermost `says` principal, but also any nested `says` principals whose propagated failures could cause the whole term to fail. Figure 14 defines the fails judgment, written $\Pi \Vdash l \succcurlyeq \tau$, which describes when a principal l can fail an expression of type τ in delegation context Π.

Consider an expression $\eta_\ell\ (\eta_{\ell'}\ e)$ and an attacker principal l_a. If $\Pi \Vdash l_a^{\mathsf{c}} \succcurlyeq \ell'^{\mathsf{c}}$, and $\Pi \not\Vdash l_a^{\mathsf{c}} \succcurlyeq \ell^{\mathsf{c}}$, then the attacker learns nothing by evaluating $\eta_\ell\ (\eta_{\ell'}\ e)$. Similarly, if $\Pi \Vdash l_a^{\mathsf{i}} \succcurlyeq \ell'^{\mathsf{i}}$ and $\Pi \not\Vdash l_a^{\mathsf{i}} \succcurlyeq \ell^{\mathsf{i}}$, then the attacker cannot influence the value $\eta_\ell\ (\eta_{\ell'}\ e)$.

In contrast, if $\Pi \Vdash {l_a}^{\mathsf{a}} \succcurlyeq \ell'^{\mathsf{a}}$, and $\Pi \not\Vdash l_a^{\mathsf{a}} \succcurlyeq \ell^{\mathsf{a}}$, an availability attacker may cause $\eta_{\ell'}\ e$ to evaluate to $\mathtt{fail}^{\ell'\ \mathsf{says}\ \tau}$, which steps to $\mathtt{fail}^{\ell\ \mathsf{says}\ (\ell'\ \mathsf{says}\ \tau)}$ by E-SEALEDFAIL. The fails relation reflects this possibility. Using A-TYPE and A-AVAIL (or A-INTEGCOM if ℓ' was of form $(\ell_1 \oplus \ell_2)$) we get $\Pi \Vdash l_a \succcurlyeq \ell\ \mathsf{says}\ (\ell'\ \mathsf{says}\ \tau)$.

We use the fails relation and the attacker set to define which availability policies a particular quorum system is capable of enforcing. We say $\mathcal{Q}$ *guards* τ if the following rule applies:

$$[\text{Q-GUARD}]\ \frac{\forall \ell \in \mathcal{A}_{[\![\mathcal{Q}]\!]}.\Pi \not\Vdash \ell \succcurlyeq \tau}{\Pi \Vdash \mathcal{Q}\ \mathit{guards}\ \tau}$$

Definition 3 (Valid quorum type)**.** A type τ is a *valid quorum type* with respect to quorum system $\mathcal{Q}$ and delegation set Π if the condition $\Pi \Vdash \mathcal{Q}\ \mathit{guards}\ \tau$ is satisfied.

Example. If $\mathcal{Q} = \{q_1 := \{a,b\}; q_2 := \{b,c\}; q_3 := \{a,c\}\}$ and $\ell_{\mathcal{Q}} = (a \oplus b) \ominus (b \oplus c) \ominus (a \oplus c)$ then $\ell_{\mathcal{Q}}\ \mathsf{says}\ (a\ \mathsf{says}\ \tau)$ is not a valid quorum type because $\Pi \not\Vdash \mathcal{Q}\ \mathit{guards}\ (\ell_{\mathcal{Q}}\ \mathsf{says}\ (a\ \mathsf{says}\ \tau))$ as $\Pi \Vdash a^{\mathsf{ia}} \succcurlyeq \ell_{\mathcal{Q}}\ \mathsf{says}\ (a\ \mathsf{says}\ \tau)$ and $a^{\mathsf{ia}} \in \mathcal{A}_{[\![\mathcal{Q}]\!]}$. But it is a valid quorum type for heterogeneous quorum system $\mathcal{Q}' = \{q_1 := \{a,b\}; q_2 := \{a,c\}\}$ as $a^{\mathsf{ia}} \notin \mathcal{A}_{[\![\mathcal{Q}']\!]}$.

7. Security Properties

To evaluate the formal properties of FLAQR, we prove that FLAQR preserves noninterference for confidentiality, integrity, and availability (section 7.2). These theorems state that attackers cannot learn

$$\mathcal{C} ::= \mathcal{F} = \varnothing \mid \mathcal{B}$$
$$\mathcal{B} ::= \ell \in \mathcal{F} \mid \mathcal{B}_1 \texttt{ OR } \mathcal{B}_2 \mid \mathcal{B}_1 \texttt{ AND } \mathcal{B}_2$$

Fig. 15. Blame constraint syntax

$$\text{[C-Conj]}\ \frac{\mathcal{C} \vDash \ell_1 \in \mathcal{F} \qquad \mathcal{C} \vDash \ell_2 \in \mathcal{F}}{\mathcal{C} \vDash \ell_1 \wedge \ell_2 \in \mathcal{F}} \qquad \text{[C-Disj]}\ \frac{\exists i \in \{1,2\}.\, \mathcal{C} \vDash \ell_i \in \mathcal{F}}{\mathcal{C} \vDash \ell_1 \vee \ell_2 \in \mathcal{F}} \qquad \text{[C-ParAnd]}\ \frac{\exists i \in \{1,2\}.\, \mathcal{C} \vDash \ell_i \in \mathcal{F}}{\mathcal{C} \vDash \ell_1 \boxplus \ell_2 \in \mathcal{F}}$$

$$\text{[C-ParOr]}\ \frac{\mathcal{C} \vDash \ell_1 \in \mathcal{F} \qquad \mathcal{C} \vDash \ell_2 \in \mathcal{F}}{\mathcal{C} \vDash \ell_1 \boxminus \ell_2 \in \mathcal{F}} \qquad \text{[C-In]}\ \frac{\Pi \Vdash \ell' \succcurlyeq p^{\pi} \quad \pi \in \{\mathtt{i}, \mathtt{a}\}}{\ell' \in \mathcal{F} \vDash p^{\pi} \in \mathcal{F}}$$

$$\text{[C-Or]}\ \frac{\mathcal{C}_1 \vDash p^{\pi} \in \mathcal{F} \qquad \mathcal{C}_2 \vDash p^{\pi} \in \mathcal{F} \qquad \pi \in \{\mathtt{i}, \mathtt{a}\}}{\mathcal{C}_1 \texttt{ OR } \mathcal{C}_2 \vDash p^{\pi} \in \mathcal{F}} \qquad \text{[C-And]}\ \frac{\exists i \in \{1,2\}.\, \mathcal{C}_i \vDash p^{\pi} \in \mathcal{F} \qquad \pi \in \{\mathtt{i}, \mathtt{a}\}}{\mathcal{C}_1 \texttt{ AND } \mathcal{C}_2 \vDash p^{\pi} \in \mathcal{F}}$$

Fig. 16. Blame membership: To apply C-In, C-Or and C-And the label p needs to be a primitive principal in $\mathcal{N} \cup \{\bot, \top\}$. The blame semantics rules ensure all statements added to the blame set only refer to primitive principals. This rule set differs from the originally published one [7], which didn't correctly handle compound principals such as $p \wedge q$.

secret inputs, influence trusted outputs, or control the failure behavior of well-typed FLAQR programs. In addition, we also prove additional theorems that formalize the soundness of our type system with respect to a program's failure behavior.

7.1. Soundness of failure

FLAQR's semantics uses the `compare` and `select` security abstractions and the failure propagation rules to model failure and failure-tolerance in distributed programs, and FLAQR's type system lets us reason statically about this failure behavior. To verify that such reasoning is *sound*, we prove two related theorems regarding the type of a program and the causes of potential failures.

In pursuit of this goal, this section introduces our *blame semantics* which reasons about failure-causing (faulty) principals during program execution. The goal is to record the set of principals which may cause run-time failures as a constraint on the set of faulty hosts $\mathcal{F}$. Figure 15 presents the syntax of *blame constraints*, which are boolean formulas representing a lower bound on the contents of $\mathcal{F}$. Atomic constraints $\ell \in \mathcal{F}$ denote that label ℓ is in faulty set $\mathcal{F}$. This initial blame constraint ($\mathcal{C}_{init}$) is represented using the toleration set of the implied quorum system.

Definition 4 (Initial blame constraint)**.** For toleration set $[\![\mathcal{Q}]\!]$ of the form $\{(p_1^1 \wedge ... \wedge p_{m_1}^1)^{\mathtt{ia}}, ..., (p_1^k \wedge ... \wedge p_{m_k}^k)^{\mathtt{ia}}\}$ the initial blame constraint $\mathcal{C}_{init}$ is defined as a (logical) disjunction of conjunctions:

$$\mathcal{C}_{init} \triangleq (p_1^1 \in \mathcal{F} \texttt{ AND } ... \texttt{ AND } p_{m_1}^1 \in \mathcal{F}) \texttt{ OR } \quad ... \quad \texttt{ OR } (p_1^k \in \mathcal{F} \texttt{ AND } ... \texttt{ AND } p_{m_k}^k \in \mathcal{F})$$

Each disjunction represents a minimal subset of a possible satisfying assignment for the faulty set $\mathcal{F}$. For brevity, we will refer to these subsets as the *possible faulty sets* implied by a particular blame constraint. Observe that for quorum system $\mathcal{Q}$, there is a one-to-one correspondence between every $t_i \in [\![\mathcal{Q}]\!]$ and every possible faulty set $\mathcal{F}_1, ..., \mathcal{F}_k$ in $\mathcal{C}_{init}$ where $\mathcal{F}_i$ is the set implied by the i^{th} disjunction in $\mathcal{C}_{init}$ such that $t_i = b_i{}^{\mathtt{ia}}$, where $b_i = \bigwedge_{p \in \mathcal{F}_i} p$.

$$[\text{C-CompareFail}]\ \frac{v_1 \neq v_2 \quad \mathcal{C}' := \mathcal{L}(v_1, v_2, \mathcal{C}, \ell_1, \ell_2)}{\langle\langle \texttt{compare}\ (\overline{\eta}_{\ell_1}\ v_1)\ \texttt{and}\ (\overline{\eta}_{\ell_2}\ v_2), c\rangle\ \&\ s\rangle^{\mathcal{C}} \Longrightarrow \langle\langle \texttt{fail}^{(\ell_1 \oplus \ell_2)}\ \texttt{says}\ \tau, c\rangle\ \&\ s\rangle^{\mathcal{C}'}}$$

Fig. 17. E-CompareFail with Blame Semantics.

Evaluation rule C-CompareFail, in Figure 17, shows how function $\mathcal{L}$ (discussed below) updates the blame constraint from $\mathcal{C}$ to $\mathcal{C}'$. We omit the blame-enabled versions of other evaluation rules since they simply propagate the blame constraint without modification.

Example.

(1) Quorum system $\mathcal{Q}_1 = \{q_1 = \{a,b\}; q_2 = \{b,c\}; q_3 = \{a,c\}\}$ has toleration set $[\![\mathcal{Q}_1]\!] = \{a^{\texttt{ia}}, b^{\texttt{ia}}, c^{\texttt{ia}}\}$ and three possible faulty sets in $\mathcal{C}_{init}$: $\mathcal{F} = \{a\}$ or $\mathcal{F} = \{b\}$ or $\mathcal{F} = \{c\}$

(2) Quorum system $\mathcal{Q}_2 = \{q_1 := \{p,q\}; q_2 := \{r\}\}$ has toleration set $[\![\mathcal{Q}_2]\!] = \{p^{\texttt{ia}} \wedge q^{\texttt{ia}}, r^{\texttt{ia}}\}$ and two possible faulty sets in $\mathcal{C}_{init}$: $\mathcal{F} = \{p,q\}$ or $\mathcal{F} = \{r\}$.

While $\mathcal{C}_{init}$ is defined statically according to the type of the program, rule C-CompareFail updates these constraints according to actual failures that occur during the program's execution. This approach identifies "unexpected" failures not implied by $\mathcal{C}_{init}$.

For example, $\mathcal{Q}_2 = \{q_1 := \{p,q\}; q_2 := \{r\}\}$ has two possible faulty sets $\mathcal{F} = \{p,q\}$ or $\mathcal{F} = \{r\}$. The initial blame constraint is $\mathcal{C}_{init} ::= (p \in \mathcal{F}\ \texttt{AND}\ q \in \mathcal{F})\ \texttt{OR}\ (r \in \mathcal{F})$

Placing blame for a specific failure in a distributed system is challenging, (and often impossible!). For example, when a comparison of values signed by ℓ_1 and ℓ_2 fails, it is unclear who to blame since either principal (or a principal acting on their behalf) could have influenced the values that led to the failure. We do know, however, that at least one of them is faulty; recording this information helps constrain the contents of possible faulty sets.

We can reason about principals that *must* be in $\mathcal{F}$ by considering all possible faulty sets implied by the blame constraints. We write $\mathcal{C} \vDash \ell \in \mathcal{F}$ (read as $\mathcal{C}$ *entails* $\ell \in \mathcal{F}$), when every possible faulty set in $\mathcal{C}$, has the $\ell \in \mathcal{F}$ clause. Figure 16 presents inference rules for the $\vDash$ relation.

The rules C-In, C-Or and C-And are defined for a primitive principal p^{π} in $\mathcal{N} \cup \{\bot, \top\}$, where $\pi \in \{\texttt{i}, \texttt{a}\}$. Whereas rules C-Conj, C-Disj, C-ParAnd and C-ParOr are defined for compound principals such as $p \wedge q$, $p \boxminus q$ etc. The blame semantics rules (particularly, the $\mathcal{L}$ and NORM functions) ensure all statementss added to the blame set only refer to primitive principals. This rule set differs from the rule set presented in the originally published one [7], which didn't correctly handle compound principals such as $p \wedge q$. For example, if $\mathcal{C} \vDash p$ and $\mathcal{C} \vDash q$, then with the old ruleset from [7], we can not prove $\mathcal{C} \vDash p \wedge q$, because the blame semantics did not add the compound principal $p \wedge q$ to $\mathcal{F}$. Instead the blame semantics add $p \in \mathcal{F}$ and $q \in \mathcal{F}$ to the blame set as two different statements. But with our corrected ruleset we can prove $\mathcal{C} \vDash p \wedge q$, given $\mathcal{C} \vDash p$ and $\mathcal{C} \vDash q$ (using C-Conj and C-In).

Let us see another example. Since ℓ_1 is included in all satisfying choices of $\mathcal{F}$ below, we can say $\mathcal{C} \vDash \ell_1 \in \mathcal{F}$(using C-Conj,C-In, and possibly C-In).

$$\begin{aligned}\mathcal{C} =& (\ell_1 \in \mathcal{F}\ \texttt{AND}\ \ell_2 \in \mathcal{F})\ \texttt{OR}\ (\ell_1 \in \mathcal{F}\ \texttt{AND}\ \ell_3 \in \mathcal{F}) \\ & \texttt{OR}\ (\ell_1 \in \mathcal{F}\ \texttt{AND}\ \ell_4 \in \mathcal{F})\ \texttt{OR}\ (\ell_1 \in \mathcal{F}\ \texttt{AND}\ \ell_5 \in \mathcal{F})\end{aligned}$$

The $\mathcal{L}$ function (full definition in Figure 45) is used by rule C-CompareFail to update $\mathcal{C}$. For an expression:

$$\texttt{compare}\ (\overline{\eta}_{\ell_1}\ v_1)\ \texttt{and}\ (\overline{\eta}_{\ell_2}\ v_2)$$

with $v_1 \neq v_2$, $\mathcal{L}(v_1, v_2, \mathcal{C}, \ell_1, \ell_2)$ updates the formulas in $\mathcal{C}$ to reflect that either ℓ_1 or ℓ_2 is faulty. If ℓ_1 or ℓ_2 already *must* be faulty, specifically if $\mathcal{C} \vDash \ell_1 \in \mathcal{F}$ or $\mathcal{C} \vDash \ell_2 \in \mathcal{F}$, then the function does not update any formulas. This approach avoids blaming honest principals when the other principal is already known to be faulty.

If neither ℓ_1 nor ℓ_2 are known to be faulty. then function $\mathcal{L}$ is called recursively on inner layers (i.e., nested $(\overline{\eta}\ )$ expressions) of v_1 and v_2 until a subexpression protected by a known-faulty principal is found. If no such layer is present, then the principal protecting the innermost layer is added to $\mathcal{C}$ (or the outer principals if there are no inner layers). Only this principal has seen the unprotected value and thus could have knowingly protected the wrong value. Observe that for well-typed `compare` expressions, only the outer layer of `compared` terms may differ in protection level, so there is less ambiguity when blaming an inner principal.

Updated constraints are kept in disjunctive normal form. Specifically, for compared terms $(\overline{\eta}_{\ell_1}\ v_1)$ and $(\overline{\eta}_{\ell_2}\ v_2)$, with $v_1 \neq v_2$, with initial constraint: $\mathcal{C}_{init} ::= (p \in \mathcal{F}\ \texttt{AND}\ q \in \mathcal{F})\ \texttt{OR}\ (r \in \mathcal{F})$, then $\mathcal{L}(v_1, v_2, \mathcal{C}_{init}, \ell_1, \ell_2)$ returns

$$\begin{aligned}\mathcal{C}' = &(p \in \mathcal{F}\ \texttt{AND}\ q \in \mathcal{F}\ \texttt{AND}\ \ell_1 \in \mathcal{F})\\ &\texttt{OR}\ (p \in \mathcal{F}\ \texttt{AND}\ q \in \mathcal{F}\ \texttt{AND}\ \ell_2 \in \mathcal{F})\\ &\texttt{OR}\ (r \in \mathcal{F}\ \texttt{AND}\ \ell_1 \in \mathcal{F})\ \texttt{OR}\ (r \in \mathcal{F}\ \texttt{AND}\ \ell_2 \in \mathcal{F})\end{aligned}$$

We can now state the soundness theorem for our blame semantics, and apply it to prove a liveness result. Theorem 1 states that for any well-typed FLAQR program with a failing execution, and the faulty sets $\mathcal{F}_i$ implied by $\mathcal{C}'$ (the final constraint computed by the blame semantics), it must be the case that the program's type τ reflects the ability of the (possibly colluding) principals in $\mathcal{F}_i$ to fail the program.

Theorem 1 (Sound blame)**.** *Given,*

(1) $\Pi; \Gamma; pc; c \vdash \langle\langle e, c\rangle \,\&\, \texttt{empty}\rangle^{\mathcal{C}_{init}} : \tau$

(2) $\langle\langle e, c\rangle \,\&\, \texttt{empty}\rangle^{\mathcal{C}_{init}} \longrightarrow^* \langle\langle \texttt{fail}^{\tau}, c\rangle \,\&\, \texttt{empty}\rangle^{\mathcal{C}'}$

where e is a source-level expression,[6]

then for each possible faulty set $\mathcal{F}_i$ implied by $\mathcal{C}'$, there is a principal $b_i = \bigwedge_{p \in \mathcal{F}_i} p$ such that $\Pi \Vdash {b_i}^{\texttt{ia}} \succcurlyeq \tau$.

Proof. Either e takes single step or multiple steps to produce the $\texttt{fail}^{\tau}$ term as the end result. For both the cases we prove it by induction over structure of e. See Appendix B for full proof. □

While Theorem 1 characterizes the relationship between a program's type and the possible faulty sets for a failing execution, it does not explicitly tell us anything about the fault-tolerance of a particular

[6] In other words, e does not contain any `fail` terms.

program. Since the type of a FLAQR program specifies its availability policy (in addition to its confidentiality and integrity), different FLAQR types will be tolerant of different failures. Below, we prove a liveness result for a common case, majority quorum protocols.

Definition 5 (Majority quorum system)**.** An m/n majority quorum system is a quorum system that always requires at least m of its hosts to reach consensus, where $m > n - m$.

Theorem 2 (Majority Liveness)**.** *If e is a source-level expression and:*

(1) $\Pi; \Gamma; pc; c \vdash \langle\langle e, c\rangle \,\&\, \texttt{empty}\rangle^{\mathcal{C}_{init}} : \tau$
(2) $\Pi \Vdash \mathcal{Q}$ *guards* τ
(3) $\mathcal{Q}$ *is a* m/n *majority quorum system*
(4) $\langle\langle e, c\rangle \,\&\, \texttt{empty}\rangle^{\mathcal{C}_{init}} \longrightarrow^* \langle\langle \texttt{fail}^{\tau}, c\rangle \,\&\, \texttt{empty}\rangle^{\mathcal{C}'}$

then for every possible faulty set $\mathcal{F}'$ *implied by* $\mathcal{C}'$, $|\mathcal{F}'| > (n - m)$.

Proof. From (2), we know τ is a valid quorum type for $\mathcal{Q}$ so $\forall \ell \in \mathcal{A}_{[\![\mathcal{Q}]\!]}.\Pi \not\Vdash \ell \succcurlyeq \tau$. Since $\mathcal{A}_{[\![\mathcal{Q}]\!]}$ is a superset of $[\![\mathcal{Q}]\!]$, we also have $\forall t \in [\![\mathcal{Q}]\!].\Pi \not\Vdash t \succcurlyeq \tau$. Furthermore, from Definition 4, for each possible faulty set $\mathcal{F}_i$ implied by $\mathcal{C}_{init}$, we know there is a principal $t_i \in [\![\mathcal{Q}]\!]$ such that $t_i = {b_i}^{\mathsf{ia}}$, where $b_i = \bigwedge_{p \in \mathcal{F}_i} p$. Therefore, for each such b_i, we know $\Pi \not\Vdash {b_i}^{\mathsf{ia}} \succcurlyeq \tau$.

Since $\mathcal{Q}$ is an m/n majority quorum system, every quorum is of size m and every faulty set in $\mathcal{C}_{init}$ is of size $(n - m)$. For contradiction, assume there exists a faulty set $\mathcal{F}'$ satisfying $\mathcal{C}'$ that has size $(n-m)$. Then by the definition of $\mathcal{L}$, all possible faulty sets implied by $\mathcal{C}'$ also have size $(n-m)$ since $\mathcal{L}$ monotonically increases the size of all possible faulty sets or none of them. Furthermore, each possible faulty set implied by $\mathcal{C}_{init}$ is a subset (or equal to) a possible faulty set implied by $\mathcal{C}'$, so $|\mathcal{F}'| = (n - m)$ implies $\mathcal{C}_{init} = \mathcal{C}'$.

From Theorem 1 we know for every possible faulty set $\mathcal{F}'_i$ implied by $\mathcal{C}'$, it must be the case that $\Pi \Vdash {b'_i}^{\mathsf{ia}} \succcurlyeq \tau$, where $\bigwedge_{p \in \mathcal{F}'_i} p$. However, since $\mathcal{C}_{init} = \mathcal{C}'$, we have a contradiction since (2) implies $\Pi \not\Vdash {b'_i}^{\mathsf{ia}} \succcurlyeq \tau$. Thus there cannot exist a possible faulty set of size (at least) $(n - m)$ implied by $\mathcal{C}'$, and all possible faulty sets must have size greater than $(n - m)$. □

7.2. Noninterference

We prove noninterference by extending the FLAQR syntax with bracketed expressions in the style of Pottier and Simonet [14]. Figure 43 shows selected bracketed evaluation rules and Figure 42a and 42b show the typing rules for bracketed terms. The soundness and completeness of the bracketed semantics are proved the Appendix A (Lemmata 16 - 21).

Noninterference often is expressed with a distinct attacker label. We use H to denote the attacker. This means the attacker can read data with label ℓ if $\Pi \Vdash \ell^{\mathsf{c}} \sqsubseteq H^{\mathsf{c}}$ and can forge or influence it if $\Pi \Vdash H^{\mathsf{i}} \sqsubseteq \ell^{\mathsf{i}}$ and can make it unavailable if $\Pi \Vdash H^{\mathsf{a}} \sqsubseteq \ell^{\mathsf{a}}$

An issue in typing brackets is how to deal with `fail` terms. Our confidentiality and integrity results are *failure-insensitive* in the sense that they only apply to terminating executions. This is similar to how termination-insensitive noninterference is typically characterized for potentially non-terminating programs.

Traditionally, bracketed typing rules require that bracketed terms have a restrictive type, ensuring that only values derived from secret (or untrusted) inputs are bracketed. In FLAQR, there are several scenarios

where a bracketed value may not have a restrictive type. For example, when a `run` expression is evaluated within a bracket, it pushes an element onto the configuration stack, but only in one of the executions. Another example is when a bracketed value occurs in a `compare` expression, but the result is no longer influenceable by the attacker H. For these scenarios, several of the typing rules in Figure 42a permit bracketed values to have less restrictive types. Because of these rules, subject reduction does not directly imply noninterference as it does in most bracketed approaches, but the additional proof obligations are relatively easy to discharge.

	Can have less restrictive type	
Term	$\pi = \mathtt{i}$	$\pi = \mathtt{a}$
$(v \mid v')$	No	Yes
$(v \mid \mathtt{fail}^{\tau})$	Yes	No
$(v \mid v)$	Yes	Yes
$(\mathtt{fail}^{\tau} \mid \mathtt{fail}^{\tau})$	Yes	Yes

The table above summarizes how bracketed terms are typed depending on whether we are concerned with integrity or availability. For integrity, unequal bracketed values must have a restrictive type (i.e., one that protects H), but equal bracketed values may have a less restrictive type. For availability, only bracketed terms where one side contains a value and the other a failure must have a restrictive type.

7.2.1. Confidentiality and Integrity Noninterference

To prove confidentiality (integrity) noninterference we need to show that given two different secret (untrusted) inputs to an expression e the evaluated public (trusted) outputs are equivalent. Equivalence is defined in terms of an observation function $\mathcal{O}$ adapted from FLAC [8] in Appendix A, Figure 44.

Theorem 3 (`c-i` Noninterference)**.** *If* $\Pi; \Gamma, x : \ell' \ \mathtt{says}\ \tau' \vdash \langle e, c\rangle \ \&\ \mathtt{empty} : \ell \ \mathtt{says}\ \tau$ *where*

(1) $\Pi; \Gamma; pc; c \vdash v_i : \ell' \ \mathtt{says}\ \tau'$*,* $i \in \{1, 2\}$
(2) $\langle e[x \mapsto (v_1 \mid v_2)], c\rangle \ \&\ \mathtt{empty} \longrightarrow^* \langle v, c\rangle \ \&\ \mathtt{empty}$
(3) $\Pi \Vdash H^{\pi} \sqsubseteq \ell'$ *and* $\Pi \not\Vdash H^{\pi} \sqsubseteq \ell$*,* $\pi \in \{\mathtt{c}, \mathtt{i}\}$.

then, $\mathcal{O}(\lfloor v \rfloor_1, \Pi, \ell, \pi) = \mathcal{O}(\lfloor v \rfloor_2, \Pi, \ell, \pi)$

Proof. From subject reduction we can prove that $\lfloor v \rfloor_1$ and $\lfloor v \rfloor_2$ have same type. By induction over the structure of projected values, $\lfloor v \rfloor_i$, we can show $\mathcal{O}(\lfloor v \rfloor_1, \Pi, \ell, \pi) = \mathcal{O}(\lfloor v \rfloor_2, \Pi, \ell, \pi)$ Please refer to the Appendix A for full proof. □

7.2.2. Availability Noninterference

Similar to [15] our end-to-end availability guarantee is also expressed as noninterference property. Specifically, if one run of a well-typed FLAQR program running on a quorum system terminates successfully (does not fail), then all other runs of the program also terminate.

This approach treats "buggy" programs where every execution returns `fail` regardless of the choice of inputs as noninterfering. This behavior is desirable because here we are concerned with proving the absence of failures that attackers can *control*. For structured quorum systems with a liveness result such as Theorem 2 for m/n majority quorums, we can further constrain when failures may occur. For example, Theorem 2 proves failures can only occur when more than $(n - m)$ principals are faulty. In contrast, Theorem 4 applies to arbitrary quorum systems provided they guard the program's type, but cannot distinguish programs where all executions fail.

```
(λ(x:τ_a)[pc]. λ(y:τ_b)[pc]. λ(z:τ_c)[pc].
  (select
    (compare x and y)
   or
    (select
      (compare y and z)
     or
      (compare x and z))))
 (run^{τ_a} e_a@a) (run^{τ_b} e_b@b) (run^{τ_c} e_c@c)
```

Fig. 18. FLAQR implementation of majority quorum example

Theorem 4 (Availability Noninterference)**.** *If*
$\Pi; \Gamma, x : \ell \ \mathtt{says}\ \tau' \vdash \langle e, c\rangle \ \&\ \mathtt{empty} : \ell_{\mathcal{Q}} \ \mathtt{says}\ \tau$ *where*

(1) $\Pi; \Gamma; pc; c \vdash f_i : \ell \ \mathtt{says}\ \tau', i \in \{1, 2\}$
(2) $\langle e[x \mapsto (f_1 \mid f_2)], c\rangle \ \&\ \mathtt{empty} \longrightarrow^* \langle f, c\rangle \ \&\ \mathtt{empty}$
(3) $\Pi \Vdash H \gg \ell \ \mathtt{says}\ \tau'$ *and* $H^{\mathtt{ia}} \in \mathcal{A}_{[\![\mathcal{Q}]\!]}$ *and*
$\Pi \Vdash \mathcal{Q}$ *guards* $(\ell_{\mathcal{Q}} \ \mathtt{says}\ \tau)$

then $\lfloor f \rfloor_1 \neq \mathtt{fail}^{\ell_{\mathcal{Q}} \ \mathtt{says}\ \tau} \iff \lfloor f \rfloor_2 \neq \mathtt{fail}^{\ell_{\mathcal{Q}} \ \mathtt{says}\ \tau}$

Proof. From subject reduction (see Lemma 25 in the Appendix) we know, $\Pi; \Gamma; pc; c \vdash \lfloor f \rfloor_i : \ell_{\mathcal{Q}} \ \mathtt{says}\ \tau$. Because $\Pi \Vdash \mathcal{Q}$ *guards* $(\ell_{\mathcal{Q}} \ \mathtt{says}\ \tau)$ and $H^{\mathtt{ia}} \in \mathcal{A}_{[\![\mathcal{Q}]\!]}$ we can write $\Pi \not\Vdash H^{\mathtt{ia}} \gg \ell_{\mathcal{Q}} \ \mathtt{says}\ \tau$ from rule Q-GUARD . This ensures if $\lfloor f \rfloor_1 \neq \mathtt{fail}^{\ell_{\mathcal{Q}} \ \mathtt{says}\ \tau}$, then $\lfloor f \rfloor_2 \neq \mathtt{fail}^{\ell_{\mathcal{Q}} \ \mathtt{says}\ \tau}$, and vice-versa. □

8. Examples revisited

We are now ready to implement the examples from section 2 with FLAQR semantics. To make these implementations intuitive we assume that our language supports integer (*int*) types, a mathematical operator > (greater than), and ternary operator `:?`. Beacuse `int` is a base type `C(int)` returns $\bot$. The examples also read from the local state of the participating principals. Which is fine because there are standard ways to encode memory (reads/writes) into lambda-calculus.

8.1. Tolerating failure and corruption

In this FLAQR implementation (Figure 18) of $2/3$ majority quorum example of section 2.1, we refer principals representing `alice`, `bob` and `carol` as a, b and c respectively. The program is executed at host c' with program counter pc. Which means condition $\Pi \Vdash c' \gg pc$ holds. The program body consists of a function of type $\tau_f = (\tau_a \xrightarrow{pc} \tau_b \xrightarrow{pc} \tau_c \xrightarrow{pc} (((a^{\mathtt{ia}} \oplus b^{\mathtt{ia}}) \ominus (b^{\mathtt{ia}} \oplus c^{\mathtt{ia}}) \ominus (a^{\mathtt{ia}} \oplus c^{\mathtt{ia}})) \ \mathtt{says}\ \tau))$ and the three arguments to the function are `run` statements. Here τ is $(a \wedge b \wedge c)^{\mathtt{c}}$ `says int`. Which means $\mathtt{C}(\tau_f) = pc$. The function body can be evaluated at c', as condition $\Pi \Vdash c' \gg pc$ is true.

Here e_a, e_b and e_c are the expressions that read the balances for account *acct* from the local states of a, b and c respectively. The program counter at a, b, and c are a, b and c respectively. The data returned

```
1 (λ(arg1 : τb)[pc]. (λ(arg2 : τb′)[pc].
2   (select
3     (bind x = arg1 in (bind y = arg2 in
4       (bind x′ = x in (bind y′ = y in
5         (η̄d (η̄(bc∧b′c)) (x′ > y′ ? x′ : y′)))))))
6    or
7   (select (arg1) or (arg2)))))(run^τb′ e′@b′))(run^τb e@b)
```

Fig. 19. FLAQR implementation of available largest balance example

from a has type τ_a, which is basically a^{ia} says τ. Similarly τ_b is b^{ia} says τ and τ_c is c^{ia} says τ. Because each run returns a balance, the base type of τ is an *int* type, and it is protected with confidentiality label $(a \wedge b \wedge c)^{\mathsf{c}}$, meaning anyone who can read all the three labels (a, b and c), can read the returned balances.

In order to typecheck the run statements the conditions $\Pi \Vdash pc \sqsubseteq a$, $\Pi \Vdash pc \sqsubseteq b$, and $\Pi \Vdash pc \sqsubseteq c$ need to hold. The condition $\Pi \Vdash c' \geqslant \mathsf{C}(\tau_a)$ is trivially true as $\mathsf{C}(\tau_a) = \bot$. Similarly $\mathsf{C}(\tau_b) = \bot$ and $\mathsf{C}(\tau_c) = \bot$ as well.

The host executing the code need to be able to read the return values from the three hosts. This means conditions $\Pi \Vdash c' \rhd a^{\mathsf{ia}}$ says τ $\Pi \Vdash c' \rhd b^{\mathsf{ia}}$ says τ and $\Pi \Vdash c' \rhd c^{\mathsf{ia}}$ says τ need to hold in order to typecheck the compare statements. The type of the whole program is $(((a^{\mathsf{ia}} \oplus b^{\mathsf{ia}}) \ominus (b^{\mathsf{ia}} \oplus c^{\mathsf{ia}}) \ominus (a^{\mathsf{ia}} \oplus c^{\mathsf{ia}}))$ says $\tau)$, which is a valid quorum type for $\mathcal{Q} = \{q_1 := \{a, b\}; q_2 := \{b, c\}; q_3 := \{a, c\}\}$.

Based on the security properties defined in section 7 this program offers the confidentiality, integrity and availability guaranteed by quorum system $\mathcal{Q}$. Therefore, the result cannot be learned or influenced by unauthorized principals, and will be available as long as two hosts out of a, b, and c are non-faulty.

The toleration set here is $[\![\mathcal{Q}]\!] = \{a^{\mathsf{ia}}, b^{\mathsf{ia}}, c^{\mathsf{ia}}\}$. So, the program is not safe against an attacker with label $l_a = a^{\mathsf{ia}} \wedge b^{\mathsf{ia}}$ (or, $a^{\mathsf{i}} \wedge b^{\mathsf{a}}$), for example. This is because $\nexists t \in [\![\mathcal{Q}]\!].\Pi \Vdash t \geqslant l_a$. Since $\Pi \Vdash l_a \geqslant a^{\mathsf{ia}}$, principal l_a can fail two compare statements on lines 3 and 8. And, because $\Pi \Vdash l_a \geqslant b^{\mathsf{ia}}$, l_a can also fail another two compare statements (one overlapping compare statment) on lines 3 and 6. Thus the whole program evaluates to fail. This FLAQR code also helps prevent incorrect comparisons. For instance, replacing z with y on line 8 will not typecheck.

8.2. Using best available services

The code in Figure 19 is the FLAQR implementation of Figure 3. The program runs at a host c with program counter pc. The expressions e and e' read account balances from principals b and b', representing the banks. The values returned from b and b' have types $\tau_b = (b^{\mathsf{ia}}$ says $(b^{\mathsf{c}} \wedge b'^{\mathsf{c}})$ says int$)$ and $\tau_{b'} = (b'^{\mathsf{ia}}$ says $(b^{\mathsf{c}} \wedge b'^{\mathsf{c}})$ says int$)$ respectively.

The type of the whole program is $((d \ominus b^{\mathsf{ia}} \ominus b'^{\mathsf{ia}})$ says $(b^{\mathsf{c}} \wedge b'^{\mathsf{c}})$ says *int*$)$. Here $d = pc \sqcup b \sqcup b'$. In order to typecheck the run statements, the conditions $\Pi \Vdash pc \sqsubseteq b$ and $\Pi \Vdash pc \sqsubseteq b'$ need to hold. The program counter at b is b and b' is b'. The bind statements (lines 3-4) typecheck because conditions $\Pi \Vdash pc \sqcup b^{\mathsf{ia}} \sqsubseteq d$, $\Pi \Vdash pc \sqcup b^{\mathsf{ia}} \sqcup b'^{\mathsf{ia}} \sqsubseteq d$, $\Pi \Vdash pc \sqcup b^{\mathsf{ia}} \sqcup b'^{\mathsf{ia}} \sqcup b^{\mathsf{c}} \sqsubseteq d$, and $\Pi \Vdash pc \sqcup b^{\mathsf{ia}} \sqcup b'^{\mathsf{ia}} \sqcup b^{\mathsf{c}} \sqcup b'^{\mathsf{c}} \sqsubseteq d$ hold, because of our choice of d.

9. Secret sharing with FLAQR

Secret sharing is a cryptographic mechanism used in several distributed systems protocols such as oblivious transfer, multiparty communication, byzantine agreement, etc. In this section we extend FLAQR with two new language constructs to support secret sharing. We call this extended version of our programming model FLAQR$^+$. Secret sharing is a process of splitting a secret into n shares and distributing the shares among n hosts (or principals in our setting). When an adequate number of hosts, say t, combine their respective shares, the secret is reconstructed [16, 17]. Sometimes this process is referred as *(t,n)-threshold* secret sharing scheme [16, 18], i.e. a quorum of t hosts (where $1 < t \leqslant n$) out of those n hosts need to combine their shares to retrieve the initial secret value. With $(t-1)$ or fewer shares, adversaries learn nothing about the secret.

Secret sharing is most commonly described in terms of a mathematical polynomial, say $p(x)$, of degree $(t-1)$, $p(0)$ being the secret value [16]. The polynomial's values at n different co-ordinates are distributed as the secret shares, and by knowing t of these values one can reconstruct[7] the polynomial, and hence they can find the secret value $p(0)$.

For simplicity, we extend FLAQR to only supports *(2,2)-threshold* secret sharing, but later we explain that extending this framework to support *(t,n)-threshold* secret sharing for $2 < n$ and $t < n$ would be straightforward. We model secret shares abstractly using a value sealed by new kinds of principals $k.L$ and $k.R$. We call k a *key principal* because, unlike the principals in $\mathcal{P}$, a new, unique key principal k is created each time secret shares are created. In contrast, the principals in $\mathcal{P}$ are statically known. The principals $k.L$ and $k.R$ represent the two associated shares of the key principal k. We will be refering to *(2,2)-threshold* secret sharing simply as *(2,2)* secret sharing in the following sections.

9.1. Motivating example of secret sharing : password splitting.

Figure 20 presents a simple *(2,2)* secret sharing example and the corresponding pseudocode is shown in Figure 21. A (secret) password `s` belongs to `bob` who wants keep a backup of it. Bob creates two secret-shares of `s` as `s_1` and `s_2` and sends them to `alice` and `carol` respectively. That is, anyone who wants to get access to `s` has to either get it directly from `bob`, or needs to get both `s_1` and `s_2` from `alice` and `carol` and reproduce it. Later, `dave` fetches the secret shares from `alice` and `carol` and combines them to produce the password `s`.

An advantage of secret sharing is that it permits the secure transmission of secrets without requiring key distribution or public-key infrastructure (PKI). Instead, any party who possesses t shares may recover the secret. This can also be a liability, however, since an adversary needs to only obtain the shares to access the secret, too. Thus, considering the flow of shares between principals is central to the security of a secret sharing scheme. In the example in Figure 21, `alice` and `carol` are unable to access the secret only because they possess a single share; a coding error can transmit both shares to either `alice` or `carol` obviating the cryptographic protection. For this reason, embedding secret sharing in a language like FLAQR makes sense because the type system ensures the code only permits authorized flows.

9.2. $(2,2)$ secret sharing in FLAQR$^+$

Our abstractions for secret sharing in FLAQR$^+$make use of the $\overline{\eta}_\ell$ term to represent sealed secret shares. However, aspects of secret sharing schemes differ from the use of $\overline{\eta}_\ell$ in prior FLAC-based

[7] Typically using Lagrange interpolation [17].

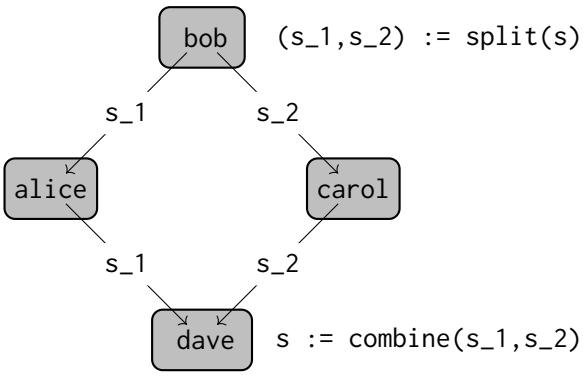


Fig. 20. Overview of $(2, 2)$ secret sharing: bob shares his secret shares with `alice` and `carol`. Later, `alice` and `carol` forward their respective shares to dave. Finally, dave reproduces the initial secret `s` with the shares he received from `alice` and `carol`.

```
1  splitCombinePassword():
2    (s_1,s_2) := split(s) @ bob;
3    send s_1 to alice;
4    send s_2 to carol;
5    con := func(); // func() returns a bool
6    if(con)
7      fetch s_1 from alice;
8      fetch s_2 from carol;
9      s' := combine(s_1,s_2) @ dave;
10   else return;
```

Fig. 21. Creating two secret shares from a secret and then reconstructing the secret from the two secret shares using functions of a $(2, 2)$ secret sharing protocol.

languages [8, 19, 20]. Here, in addition to the sealed values generated by the η_ℓ term, sealed values may also be created when splitting a secret into shares. In the previous approaches, a value sealed by η_ℓ serves as a reasonable model for signed and encrypted values. Specifically, the confidentiality component ℓ^{c} behaves like a public-key encrypted value: anyone can encrypt values at ℓ^{c}, but only authorized parties (which possess the associated private key) can distinguish the values protected at ℓ^{c}. Since η_ℓ can only be applied in contexts where $pc \succcurlyeq \ell^{\mathsf{i}}$ (see UNITM), the integrity component behaves like a digitally signed value: only authorized principals can cause a value to be signed with ℓ^{i} integrity, but anyone can use high-integrity values.[8] Obviously then, enforcing these policies cryptographically would require public-key infrastructure.

Secret sharing behaves differently from the above interpretations: rather than authorization being based on possession of a long-lived private key (a reasonable proxy for identity), secret sharing implicitly authorizes *any party* possessing t shares. Therefore using identity-based principals such as `alice` or bob is inappropriate since, even if a shared secret is intended for `alice`, anyone with t shares will be able to distinguish the secret. Even `alice` must have t shares to access the secret. An abstraction for secret sharing should capture this behavior, but doing so in FLAQR$^+$requires new concepts.

[8] There doesn't appear to be a natural cryptographic analog for the availability component.

$$[\text{E-SPLIT}] \qquad \frac{k \text{ is fresh}}{\mathtt{split}_\ell\ v \longrightarrow \langle(\overline{\eta}_{k.L^{\mathsf{c}} \wedge \ell}\ v), (\overline{\eta}_{k.R^{\mathsf{c}} \wedge \ell}\ v)\rangle}$$

$$[\text{E-COMBINE}] \qquad \mathtt{combine}\ x = \langle(\overline{\eta}_{k.L^{\mathsf{c}} \wedge \ell}\ v), (\overline{\eta}_{k.R^{\mathsf{c}} \wedge \ell}\ v)\rangle @ pc\ \mathtt{in}\ e \longrightarrow e[x \mapsto v]$$

Fig. 22. FLAQR$^+$ semantics for secret sharing (splitting secrets and combining shares).

$$[\text{SPLIT}] \frac{\Pi; \Gamma; pc; c \vdash e : \tau \qquad \Pi \Vdash c \geqslant pc \qquad \Pi \Vdash pc \sqsubseteq \ell \sqcup \triangle(pc^{\mathsf{i}})}{\Pi; \Gamma; pc; c \vdash \mathtt{split}_\ell\ e : (L^{\mathsf{c}} \wedge \ell\ \mathtt{says}\ \tau \times R^{\mathsf{c}} \wedge \ell\ \mathtt{says}\ \tau)}$$

$$[\text{COMBINE}] \frac{\Pi; \Gamma; pc; c \vdash e : (L^{\mathsf{c}} \wedge \ell\ \mathtt{says}\ \tau \times R^{\mathsf{c}} \wedge \ell\ \mathtt{says}\ \tau) \qquad \Pi \Vdash c \geqslant pc \qquad \Pi; \Gamma, x{:}\tau; \ell \sqcup pc; c \vdash e' : \ell'\ \mathtt{says}\ \tau \qquad \Pi \Vdash \ell \sqcup pc \sqsubseteq \ell'\ \mathtt{says}\ \tau}{\Pi; \Gamma; pc; c \vdash \mathtt{combine}\ x = e @ pc\ \mathtt{in}\ e' : \ell'\ \mathtt{says}\ \tau}$$

$$[\text{SEALEDK}] \frac{\Pi; \Gamma; pc; c \vdash v : \tau \qquad \Pi \Vdash c \geqslant pc \qquad K \in \{L^{\mathsf{c}}, R^{\mathsf{c}}\}}{\Pi; \Gamma; pc; c \vdash (\overline{\eta}_{k.K \wedge \ell}\ v) : K \wedge \ell\ \mathtt{says}\ \tau}$$

Fig. 23. FLAQR$^+$ typing rules for secret sharing.

We extend the set of principals $\mathcal{P}$ with new primitive principals L and R representing the left and right shares of our $(2, 2)$ secret sharing scheme. The set of all principals $\mathcal{P}$ for FLAQR$^+$is thus the closure of the set $\mathcal{N} \cup \{\top, \bot, L, R\}$ over the same operations as FLAQR. In the following we are only interested in the confidentiality projections L^{c} and R^{c}, since secret sharing only concerns enforcing the confidentiality of the secret.

Another aspect of secret sharing that departs from prior uses of FLAC principals is that each time shares are created, they are protected by a different secret. Consequently, shares created from different invocations cannot be mixed, even when the underlying value is the same. For this reason, we define *key principals*, a new type of principal generated dynamically at runtime. For our purposes, each $k \in \mathcal{K}$, where $\mathcal{K}$ is the set of all key principals, is equipped with a left and right principal, $k.L^{\mathsf{c}}$ and $k.R^{\mathsf{c}}$. Importantly, since key principals are generated dynamically, they are not directly representable statically. The principals L^{c} and R^{c} are the static representation for the left and right principals of any key principal, but the shares of different key principals cannot be distinguished at the type level.

9.3. Semantics and types for secret sharing

Figure 22 presents the semantic rules added to FLAQR$^+$. Expression $\mathtt{split}_\ell\ v$ produces two secret shares, sealed with principals $k.L^{\mathsf{c}} \wedge \ell$ and $k.R^{\mathsf{c}} \wedge \ell$ from the secret value v (rule E-SPLIT) using a fresh key principal k. The ℓ annotation specifies an additional policy to seal the secret with an addition to

the key principal. Primarily ℓ is used for integrity and availability components since $k.L^{\mathsf{c}}$ and $k.R^{\mathsf{c}}$ are confidentiality projections.

Two shares are combined with expression

$$\texttt{combine}\ x = \langle (\overline{\eta}_{k.L^{\mathsf{c}} \wedge \ell}\ v), (\overline{\eta}_{k.R^{\mathsf{c}} \wedge \ell}\ v) \rangle @ pc\ \texttt{in}\ e$$

Rule E-COMBINE evaluates these terms to $e[x \mapsto v]$ revealing the secret v and substituting it for x in the body e. Notice that the key principal is the same on both sides of the pair. As discussed below in Section 9.4, mismatched key principals result in failure. The additional *pc* annotation on combine terms is used by the extended blame semantics, discussed in Section 9.4.

For simplicity, our extension only supports *(2,2)-threshold* secret sharing, but we believe extending this framework to support *(t,n)-threshold* secret sharing for $2 < n$ and $t < n$ would be straightforward. For example, given some t and n, we could redefine E-SPLIT to generate a tuple containing n shares sealed by principals $k.S^{c}_{1}, k.S^{c}_{2}, ..., k.S^{c}_{n}$. E-COMBINE would be replaced by $\binom{n}{t}$ rules: one for each valid t-sized subset of shares.

Figure 23 presents the FLAQR^{+} typing rules for `split` and `combine`. The last premise in the SPLIT rule involves the *view* [20] of the *pc*'s integrity, $\triangle(pc^{\mathsf{i}})$. The view of a principal was introduced by Ceccetti et al [20] to specify an upper bound on the confidentiality that may be *robustly declassified* [21] based on the integrity of the context performing the declassification and the data itself. These restrictions ensure an attacker cannot influence what (or whether) information is declassified. Below, we extend the definition of *view* with the principals' availability projection counterpart as well.

Definition 6 (*view* of a principal)**.** Let $\ell = p^{\mathsf{c}} \wedge q^{\mathsf{i}} \wedge r^{\mathsf{a}}$ be a FLAM [6] label (principal) expressed in normal form. The view of ℓ, written as $\triangle(\ell)$, is defined as $\triangle(p^{\mathsf{c}} \wedge q^{\mathsf{i}} \wedge r^{\mathsf{a}}) \triangleq q^{\mathsf{c}}$.

The premise $\Pi \Vdash pc \sqsubseteq \ell \sqcup \triangle(pc^{\mathsf{i}})$ in SPLIT serves two purposes. First, it ensures the confidentiality of control flow and the unsealed values in the context, represented by *pc*, are no more restrictive than the upper bound on declassification $\triangle(pc^{\mathsf{i}})$ (or the confidentiality of ℓ if no declassification takes place). Second, it ensures the label ℓ protects the availability and integrity of the context; only confidentiality may be downgraded by `split` terms.

When shares are combined to reveal the secret, the rule COMBINE ensures the combined pair contains a left and right share (although not which key principal they are associated with), and that the body of the `combine` term protects the result with a principal at least as restrictive as the upper bound of ℓ and the context *pc* the `combine` occurs in.

In some sense, $\texttt{split}_{\ell}$ and `combine` function as an alternative to η_{ℓ} and `bind`. The difference is that `split` seals values using a key principal in addition to a label ℓ, and permits secrets more restrictive than ℓ to be sealed. Combine is similar to a `bind` that declassifies its bound value, dropping the key principals $k.L$ and $k.R$ from the protection requirements on the body of the `combine`. Note that it is not possible for a type-safe program to `bind` a secret share. Since a premise of bind would require the body accessing the unsealed share on host c to typecheck at $pc \sqcup (L^{\mathsf{c}} \wedge \ell)$, this would violate the invariant that c must act for the *pc* of the programs it executes.[9]

We need one more typing rule, SEALEDK, to preserve the types of the values sealed with labels $k.L^{\mathsf{c}}$ and $k.R^{\mathsf{c}}$, for any freshly generated key $k \in \mathcal{K}$. The existing SEALED rule is not enough as it does not handle the values protected with these new *key* principals. Although, the consequence of having this

[9] Recall this invariant is enforced by the $\Pi \Vdash c \succcurlyeq pc$ premise included in all typing rules.

$$[\text{E-SplitFail}] \quad \frac{k \text{ is fresh}}{\texttt{split}_\ell\ \texttt{fail}^\tau \longrightarrow \texttt{fail}^{(L^c \wedge \ell\ \texttt{says}\ \tau \times R^c \wedge \ell\ \texttt{says}\ \tau)}}$$

$$[\text{E-CombineFail}] \quad \frac{k_1 \neq k_2}{\texttt{combine}\ x = \langle(\overline{\eta}_{k_1.L^c \wedge \ell}\ v), (\overline{\eta}_{k_2.R^c \wedge \ell}\ v)\rangle @pc\ \texttt{in}\ e \longrightarrow e[x \mapsto \texttt{fail}^\tau]}$$

$$[\text{E-CombineFailL}] \quad \texttt{combine}\ x = \langle \texttt{fail}^{L^c \wedge \ell\ \texttt{says}\ \tau}, f\rangle @pc\ \texttt{in}\ e \longrightarrow e[x \mapsto \texttt{fail}^\tau]$$

$$[\text{E-CombineFailR}] \quad \texttt{combine}\ x = \langle v, \texttt{fail}^{R^c \wedge \ell\ \texttt{says}\ \tau}\rangle @pc\ \texttt{in}\ e \longrightarrow e[x \mapsto \texttt{fail}^\tau]$$

Fig. 24. `fail` propagation rules in FLAQR$^+$

$$[\text{C-CombineFail}] \quad \frac{k_1 \neq k_2 \quad \mathcal{C}' := \texttt{NORM}(pc, \mathcal{C})}{\langle\langle \texttt{combine}\ x = \langle(\overline{\eta}_{k_1.L^c \wedge \ell}\ v_1), (\overline{\eta}_{k_2.R^c \wedge \ell}\ v_2)\rangle @pc\ \texttt{in}\ (\overline{\eta}_\ell\ x), c\rangle\ \&\ s\rangle^{\mathcal{C}} \Longrightarrow \langle\langle \texttt{fail}^{\ell\ \texttt{says}\ \tau}, c\rangle\ \&\ s\rangle^{\mathcal{C}'}}$$

Fig. 25. E-CombineFail with Blame Semantics.

rule is that, inspite of being well-typed, a FLAQR$^+$ program can produce mismatching shares with two different keys (say k_1 and k_2) during run-time. We handle such cases with mismatching shares with our extended blame semantics in section 9.4.

9.4. Extending the blame semantics

As with other FLAQR terms, `fail` values propagate through `split` and `combine`. Figure 24 presents fail propagation rules for `split` and `combine` statements. These rules are straightforward propagation rules except for E-CombineFail, which evaluates to `fail` if the key principals sealing the shares are mismatched.

The introduction of E-CombineFail rule creates an additional source of failure besides `compare` terms with mismatched values. Rule C-CombineFail extends FLAQR's blame semantics to account for this. Unlike the case for `compare`, we cannot blame the failure on the principal that sealed the mismatched values given to `combine`. The failure in this case is due to pairing together shares generated by different `split` evaluations. Rather than blaming the creators of the sealed value, we instead want to blame the principals that influenced this pairing. This influence is represented by the label of the *pc*. Hence, when k_1 and k_2 do not match, C-CombineFail adds *pc* to the blame set. The function $\mathcal{L}$ used in C-CompareFail is unnecessary here because it is unnecessary to examine any subterms of the combined pair—only the outer key principals contribute to a `combine` failure.

The `NORM` function[10] in the premise of C-CombineFail is used to add the new (potentially malicious) pricipal *pc* in the exisiting blame set $\mathcal{C}$, in a normalized form. For example, if $pc = (a \wedge b) \vee c$ and if

[10]This normalization function was not present in the original FLAQR publication [7], which is an error. Normalization of the statements added to the blame set is required to ensure compound principals are correctly handled.

$\mathcal{C} := (\ell_1 \in \mathcal{F})$ OR $(\ell_2 \in \mathcal{F})$, then calling NORM$(pc, \mathcal{C})$ will return a blame set

$$\mathcal{C}' := (a \in \mathcal{F} \text{ AND } b \in \mathcal{F} \text{ AND } \ell_1 \in \mathcal{F}) \text{ OR } (a \in \mathcal{F} \text{ AND } b \in \mathcal{F} \text{ AND } \ell_2 \in \mathcal{F})$$
$$\text{OR } (c \in \mathcal{F} \text{ AND } \ell_1 \in \mathcal{F}) \text{ OR } (c \in \mathcal{F} \text{ AND } \ell_1 \in \mathcal{F})$$

In case of C-COMPAREFAIL (Figure 17), the NORM function was called from within the $\mathcal{L}$ function (see Figure 45). Figure 46 contains the complete definition of the NORM function.

9.5. Security properties

By design, split and combine are interfering with respect to confidentiality: they can cause secret values to be declassified. However, we would like to ensure that integrity and availability noninterference are unaffected.

In order to prove integrity and availability noninterference for FLAQR$^+$ programs we extend the bracketed semantics (Figure 26) and observation function (Figure 27) with rules for split and combine, and add the corresponding cases for split and combine terms to the proofs for the lemmas and theorems of FLAQR$^+$. The noninterference theorem statements for FLAQR$^+$ are identical to Theorems 3 and 4, though Theorem 3 only holds for $\pi = \mathtt{i}$ in FLAQR$^+$. Since the new static principals L and R are only used in confidentiality projections, rules such as Q-GUARD and the *fails* are unaffected by the new terms for secret sharing, the proofs of these theorems is largely unchanged from those for FLAQR. However, ensuring the new terms did not break an essential lemma such as subject reduction (Lemma 24) required careful design of the new evaluation, failure propagation, bracketed semantics, and typing rules.

Although we protect the robustness (in theory) of what values may be declassified via split and combine, secret sharing is inherently non-robust since the party possessing the shares decides whether to reveal the secret. To formalize the protections that split and combine do offer, a weaker form of robust declassification would be required that permits secure uses of split and prohibits insecure ones (such as those violating the premise $\Pi \Vdash pc \sqsubseteq \ell \sqcup \triangle(pc^{\mathtt{i}})$). Such a definition is not immediately clear to us, and we leave further investigation to future work. Since split and combine permit non-robust declassification, they could potentially permit malleability attacks [20]. Since secret shares cannot be unsealed via bind, the possibility for such attacks is limited, but we leave formalization of these limitations to future work.

For compare statements failures are generated because of two mismatching values. For combine statements the contents of the secret shares are irrelevant. Instead the failures happen due to mismatching keys of the secret shares. Hence, we can only blame the control flow of the program for putting the two mismatching secret shares together. Because, the program counter tracks the control flow of the program, we dynamically blame the program counter *pc* in the C-COMBINEFAIL rule by adding it to the blame set when a fail term is returned while combining two shares. Our Theorem 1 (Sound blame) still holds, even though combine statement we incorporates a new source of failure. This is because of the premise $\Pi \Vdash pc \sqsubseteq \ell' \texttt{ says } \tau$ in COMBINE typing rule allows us to show $\Pi \Vdash pc \succ \ell' \texttt{ says } \tau$ (using P-LBL and A-AVAIL). Since, Theorem 1 holds Theorem 2 (Majority liveness) holds as well, as it depends on Theorem 1.

9.6. Password splitting example with FLAQR$^+$.

Figure 28 presents the FLAQR$^+$implementation of the example we discussed in Section 9.1. The program executes at host c' with program counter *pc*, such that $\Pi \Vdash c' \succcurlyeq pc$. The host a has program

$$[\text{B-SPLIT}] \qquad \mathtt{split}_\ell\ (v_1 \mid v_2) \longrightarrow \langle (\overline{\eta}_{k.L^{\mathsf{c}} \wedge \ell}\ (v_1 \mid v_2)), (\overline{\eta}_{k.R^{\mathsf{c}} \wedge \ell}\ (v_1 \mid v_2)) \rangle$$

$$[\text{B-COMBINE}]\ \mathtt{combine}\ x = (v_1 \mid v_2)@pc\ \mathtt{in}\ e \longrightarrow (\mathtt{combine}\ x = v_1@pc\ \mathtt{in}\ e \mid \mathtt{combine}\ x = v_2@pc\ \mathtt{in}\ e)$$

Fig. 26. Bracketed semantics for FLAQR$^+$ terms.

$$\mathcal{O}(\mathtt{split}_l\ e, \Pi, \ell, \pi) = \mathtt{split}_l\ \mathcal{O}(e, \Pi, \ell, \pi)$$

$$\mathcal{O}(\mathtt{combine}\ x = \langle e_1, e_2 \rangle @pc\ \mathtt{in}\ e, \Pi, \ell, \pi) = \mathtt{combine}\ x = \langle \mathcal{O}(e_1, \Pi, \ell, \pi), \mathcal{O}(e_2, \Pi, \ell, \pi) \rangle @pc\ \mathtt{in}\ \mathcal{O}(e, \Pi, \ell, \pi)$$

Fig. 27. Observation function for intermediate FLAQR$^+$ terms (extended from FLAC [8]).

counter a, host b has program counter b and host c has program counter c. The program basically consists of a function body (lines 1-5) and an argument to it (line 6). Particularly, the function body is of type $\tau_b \xrightarrow{pc} \ell'$ says int, where $\tau_b = b^{\mathsf{ia}}$ says $(L^{\mathsf{c}} \wedge b$ says int $\times R^{\mathsf{c}} \wedge b$ says int$)$. The function body takes the value of running a split statement at host b (i.e. $\mathsf{run}^{\tau_b}\ (\mathtt{split}_b\ v)@b$), which splits b's secret v. The argument type is τ_b. This means the pair of the secret shares created at and returned by b is tainted with b's integrity and availability. In order to typecheck the run statement pc needs to flow to b, i.e. the condition $\Pi \Vdash pc \sqsubseteq b$ needs to hold. The condition $\Pi \Vdash c \succcurlyeq \mathsf{C}((L^{\mathsf{c}} \wedge b$ says int $\times R^{\mathsf{c}} \wedge b$ says int$))$ satisfies trivially, as $\mathsf{C}((L^{\mathsf{c}} \wedge b$ says int $\times R^{\mathsf{c}} \wedge b$ says int$)) = \bot$. The function body can be executed at c' as $\mathsf{C}(\tau_b \xrightarrow{pc} \ell'$ says int$) = pc$ and we mentioned earlier that the condition $\Pi \Vdash c' \succcurlyeq pc$ is true. The run statements on line 3 and 4 indicates that the left share is tainted by a's and the right share is tainted by c's integrity and availability. Which means a and c have seen and approved on the secret shares created by b. To make the run statements on lines 3 and 4 well-typed, the conditions $\Pi \Vdash pc \sqsubseteq a$ $\Pi \Vdash pc \sqsubseteq c$ should satisfy. We choose label ℓ such that $\Pi \Vdash pc \sqcup a^{\mathsf{ia}} \sqcup b^{\mathsf{ia}} \sqcup c^{\mathsf{ia}} \sqsubseteq \ell$. The bind statements (lines 2-4) typecheck because the conditions $\Pi \Vdash pc \sqcup b^{\mathsf{ia}} \sqsubseteq \ell$, $\Pi \Vdash pc \sqcup b^{\mathsf{ia}} \sqcup a^{\mathsf{ia}} \sqsubseteq \ell$ and $\Pi \Vdash pc \sqcup b^{\mathsf{ia}} \sqcup a^{\mathsf{ia}} \sqcup c^{\mathsf{ia}} \sqsubseteq \ell$ hold due to our choice of ℓ.

10. Related work

FLAM [6, 22] offers an algebra to integrate authorization logics and information flow control policies. FLAM also introduces a security condition, robust authorization, that is useful to ensure security when delegations and revocations change the meaning of confidentiality and integrity policies. In FLAQR we extend FLAM algebra with availability policies, and new binary operations to represent integrity and availability policies of the output of quorum based protocols. FLAC [9][8] embeds its types with FLAM information flow policies. FLAC supports dynamic delegation of authority, but this feature is omitted in FLAQR.

A limited number of previous approaches [15, 23] combine availability with more common confidentiality and integrity policies in distributed systems. Zheng and Myers [23] extend the Decentralized

```
1 λ(arg:b^ia says (L^c ∧ b^ia says int × R^c ∧ b^ia says int))[pc].
2   (bind s = arg in
3     (bind s1 = (run^τa (proj1 s)@a) in
4       (bind s2 = (run^τc (proj2 s)@c) in
5         (combine sec = ⟨s1, s2⟩@pc in (η̄ℓ sec)))))
6 (run^τb (split_b^ia v)@b)
```

where

$$\tau_a = a^{\mathsf{ia}}\ \mathtt{says}\ (L^{\mathsf{c}} \wedge b\ \mathtt{says}\ \mathtt{int})$$

$$\tau_b = b^{\mathsf{ia}}\ \mathtt{says}\ (L^{\mathsf{c}} \wedge b\ \mathtt{says}\ \mathtt{int} \times R^{\mathsf{c}} \wedge b\ \mathtt{says}\ \mathtt{int})$$

$$\tau_c = c^{\mathsf{ia}}\ \mathtt{says}\ (R^{\mathsf{c}} \wedge b\ \mathtt{says}\ \mathtt{int})$$

Fig. 28. A simple example of secret sharing in FLAQR$^+$.

Label Model [24] with availability policies, but focus primarily tracking dependencies rather than applying mechanisms such as consensus and replication to improve availability and integrity. Zheng and Myers later introduce the language Qimp [15] with a type system explicitly parameterized on a quorum system for offloading computation while enforcing availability policies. Instead of treating quorums specially, FLAQR quorums emerge naturally using `compare` and `select` and enable application-specific integrity and availability policies that are secure by construction.

Hunt and Sands [25] present a novel generalisation of information flow lattices that captures disjunctive flows similar to the influence of replicas in FLAQR on a `select` result. Our partial-or operation was inspired by their treatment of disjunctive dependencies.

Models of distributed system protocols are often verified with model checking approaches such as TLA+ [26]. Model checking programs is typically undecidable, making it ill-suited to integrate directly into a programming model in the same manner as a (decidable) type system. To make verification tractable, TLA+ models are often simplified versions of the implementations they represent, potentially leading to discrepancies. FLAQR is designed as a core calculus for a distributed programming model, making direct verification of implementations more feasible.

BFT protocols [2, 27] use consensus and replication to protect the integrity and availability of operations on a system's state. Each instance of a BFT protocol essentially enforces a single availability policy and a single integrity policy. While composing multiple instances is possible, doing so provides no end-to-end availability or integrity guarantees for the system as a whole. FLAQR programs, by constrast, routinely compose consensus and replication primitives to enforce multiple policies while also providing end-to-end system guarantees.

Our blame semantics presented in Section 7.1 has some resemblance to the idea of blame used to detect contract violations [28] and applied to gradual typing [29]. In our system, blame is necessarily ambiguous since perfect fault detection is not possible. Hence, rather than identifying a single program point responsible for a contract or type violation, our semantics builds constraints that specify a set of principals that may be responsible for a given failure.

In [30] Clarkson and Schneider talks about integrity measures such as contamination and suppression. The idea of suppression can be equivalent to the idea of unavailability. Although in [30] suppression happens due to untrusted input making trusted output unavailable. In FLAQR, unavailibity is caused by both unavailable and untrusted (via `compare` statement) inputs.

11. Conclusion

In this work, we extend Flow Limited Authorization Model [6] with availability policies. We introduce a core calculus and type-system, FLAQR, for building decentralized applications that are secure by construction. We identify a trade-off relation between integrity and availability, and introduce two binary operations *partial-and* and *partial-or*, specifically to express integrities of quorum based replicated programs. We define $fails$ relation and judgments that help us reason about a principal's authority over availability of a type. We introduce blame semantics that associate failures with malicious hosts of a quorum system to ensure that quorums can not exceed a bounded number of failures without causing the whole system to fail. FLAQR ensures end-to-end information security with noninterference for confidentiality, integrity and availability. Finally we present FLAQR$^+$, which is an extension of FLAQR with language constructs that support secret sharing between hosts with mutual distrust. We extend our failure propagation rules and blame semantics to assign blame to appropriate principals when a secret sharing round fails.

12. Acknowledgements

Funding for this work was provided in part by NSF CAREER CNS-1750060 and IARPA HECTOR CW3002436.

References

[1] L. Lamport, The Part-time Parliament, *ACM Trans. on Computer Systems* **16**(2) (1998), 133–169. doi:10.1145/279227.279229.
[2] M. Castro and B. Liskov, Practical Byzantine fault tolerance and proactive recovery, *ACM Trans. on Computer Systems* **20** (2002), 2002.
[3] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, *Consulted* **1**(2012) (2008), 28.
[4] J. Liu, O. Arden, M.D. George and A.C. Myers, Fabric: Building Open Distributed Systems Securely by Construction, *J. Computer Security* **25**(4–5) (2017), 319–321. doi:10.3233/JCS-0559.
[5] N. Zeldovich, S. Boyd-Wickizer and D. Mazières, Securing distributed systems with information flow control, in: *5th USENIX Symp. on Networked Systems Design and Implementation (NSDI)*, 2008, pp. 293–308.
[6] O. Arden, J. Liu and A.C. Myers, Flow-Limited Authorization, in: *28th IEEE Computer Security Foundations Symp. (CSF)*, 2015, pp. 569–583.
[7] P. Mondal, M. Algehed and O. Arden, Applying consensus and replication securely with FLAQR, in: *IEEE Computer Security Foundations Symp (CSF)*, 2022, pp. 163–178. doi:10.1109/CSF54842.2022.9919637.
[8] O. Arden, A. Gollamudi, E. Cecchetti, S. Chong and A.C. Myers, A Calculus for Flow-Limited Authorization: Technical Report, 2021. doi:10.48550/ARXIV.2104.10379.
[9] O. Arden and A.C. Myers, A Calculus for Flow-Limited Authorization, in: *29th IEEE Computer Security Foundations Symp. (CSF)*, 2016, pp. 135–147.
[10] M. Abadi, Access Control in a Core Calculus of Dependency, in: *11th ACM SIGPLAN Int'l Conf. on Functional Programming*, ACM, New York, NY, USA, 2006, pp. 263–273. doi:10.1145/1159803.1159839.
[11] J.-Y. Girard, Une extension de L'interpretation de gödel a L'analyse, et son application a L'elimination des coupures dans L'analyse et la theorie des types, in: *Studies in Logic and the Foundations of Mathematics*, Vol. 63, Elsevier, 1971, pp. 63–92.
[12] J.-Y. Girard, Interpretation fonctionelle et elimination des coupure dans l'arithmetic d'ordre superieur, *Ph. D. Thesis, L'universite Paris VII* (1972).
[13] J.C. Reynolds, Towards a theory of type structure, in: *Programming Symposium*, Springer, 1974, pp. 408–425.
[14] F. Pottier and V. Simonet, Information flow inference for ML, in: *29th ACM Symp. on Principles of Programming Languages (POPL)*, 2002, pp. 319–330.

[15] L. Zheng and A.C. Myers, A Language-Based Approach to Secure Quorum Replication, in: *9th ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS)*, 2014.
[16] A. Shamir, How to Share a Secret, *Communications of the ACM* **22**(11) (1979), 612–613.
[17] E. Dawson and D. Donovan, The breadth of Shamir's secret-sharing scheme, Computer & Security, 1994.
[18] C.-C. Yang, T.-Y. Chang and M.-S. Hwang, A (t,n) multi-secret sharing scheme, in: *Applied Mathematics and Computation*, 2004, pp. 483–490.
[19] A. Gollamudi, S. Chong and O. Arden, Information flow control for distributed trusted execution environments, in: *32nd IEEE Computer Security Foundations Symp. (CSF)*, 2019.
[20] E. Cecchetti, A.C. Myers and O. Arden, Nonmalleable Information Flow Control, in: *24th ACM Conf. on Computer and Communications Security (CCS)*, 2017, pp. 1875–1891.
[21] S. Zdancewic and A.C. Myers, Robust Declassification, in: *14th IEEE Computer Security Foundations Workshop (CSFW)*, 2001, pp. 15–23. ISSN 1063-6900. doi:10.1109/CSFW.2001.930133.
[22] O. Arden, J. Liu and A.C. Myers, Flow-Limited Authorization: Technical Report, Technical Report, 1813–40138, Cornell University Computing and Information Science, 2015.
[23] L. Zheng and A.C. Myers, End-to-End Availability Policies and Noninterference, in: *18th IEEE Computer Security Foundations Workshop (CSFW)*, 2005, pp. 272–286.
[24] A.C. Myers and B. Liskov, Protecting Privacy using the Decentralized Label Model, *ACM Transactions on Software Engineering and Methodology* **9**(4) (2000), 410–442.
[25] S. Hunt and D. Sands, A Quantale of Information, in: *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*, 2021. doi:10.1109/CSF51468.2021.00031.
[26] L. Lamport, The PlusCal Algorithm Language, in: *Theoretical Aspects of Computing - ICTAC 2009*, M. Leucker and C. Morgan, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 36–60. ISBN 978-3-642-03466-4.
[27] A. Bessani, J. Sousa and E.E. Alchieri, State machine replication for the masses with BFT-SMaRt, in: *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, IEEE, 2014, pp. 355–362.
[28] R.B. Findler and M. Felleisen, Contracts for Higher-Order Functions, *SIGPLAN Not.* **37**(9) (2002), 48–59–. doi:10.1145/583852.581484.
[29] P. Wadler and R.B. Findler, Well-Typed Programs Can't Be Blamed, in: *Programming Languages and Systems*, G. Castagna, ed., Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 1–16. ISBN 978-3-642-00590-9.
[30] M.R. Clarkson and F.B. Schneider, Quantification of Integrity, in: *Proc. IEEE Computer Security Foundations Symposium*, 2010, pp. 28–43.

Appendix A. Complete FLAQR rule set and noninterference proofs

In the following proofs we use the simple and the annotated FLAQR syntax interchageably.

Lemma 1 (UniqueType)**.** *If* $\Pi;\Gamma;pc;c \vdash e : \tau$ *and* $\Pi;\Gamma;pc;c \vdash e : \mathring{\tau}$ *then* $\tau = \mathring{\tau}$

Proof. Straightforward by induction on typing derivation of e. □

Lemma 2 (WaitUniqueT)**.** *If* $\Pi;\Gamma;pc;c \vdash E[\texttt{expect}^{\hat{\tau}}] : \tau$ *and* $\Pi;\Gamma;pc;c \vdash E[\texttt{expect}^{\hat{\tau}}] : \tau'$ *then* $\tau = \tau'$.

Proof. Straightforward using induction over structure of E. □

Lemma 3 (stackUniqueT)**.** *If type of the tail* $\Pi;\Gamma;pc \vdash t : [\hat{\tau}]\tau$ *and* $\Pi;\Gamma;pc \vdash t : [\hat{\tau}]\tau'$ *then* $\tau = \tau'$.

Proof. The proof is by induction over typing derivation of s. □

Lemma 4 (distUniqueT)**.** *If* $\Pi;\Gamma;pc \vdash \langle e,c\rangle \,\&\, s : \tau$ *and* $\Pi;\Gamma;pc \vdash \langle e,c\rangle \,\&\, s : \tau'$ *then* $\tau = \tau'$.

Proof. Straightforward proof using lemmas UniqueType 1 and 2. □

$\pi \in \{\mathsf{c}, \mathsf{i}, \mathsf{a}\}$ (projections)
$n \in \mathcal{N}$ (primitive principals)
$x \in \mathcal{V}$ (variable names)

$$
\begin{aligned}
p, \ell, pc ::= {} & n \mid \top \mid \bot \mid p^{\pi} \mid p \wedge p \mid p \vee p \\
& \mid p \sqcup p \mid p \sqcap p \mid p \boxminus p \mid p \boxplus p \\
\tau ::= {} & \mathsf{unit} \mid X \mid (\tau + \tau) \mid (\tau \times \tau) \\
& \mid \tau \xrightarrow{pc} \tau \mid \forall X[pc].\, \tau \mid \ell \ \mathsf{says}\ \tau \\
v ::= {} & () \mid (\overline{\eta}_{\ell}\ v) \mid \mathsf{inj}_i^{(\tau+\tau)}\ v \mid \langle v, v \rangle^{\tau} \\
& \mid \lambda(x{:}\tau)[pc].\, e \mid \Lambda X[pc].\, e \\
f ::= {} & v \mid \mathsf{fail}^{\tau} \\
e ::= {} & f \mid x \mid e\ e \mid e\ \tau \mid \eta_{\ell}\ e \mid \langle e, e \rangle^{\tau} \\
& \mid \mathsf{proj}_i\ e \mid \mathsf{inj}_i^{(\tau+\tau)}\ e \mid \mathsf{bind}\ x = e\ \mathsf{in}\ e \\
& \mid \mathsf{case}^{\tau}\ e\ \mathsf{of}\ \mathsf{inj}_1^{\tau}(x).e \mid \mathsf{inj}_2^{\tau}(x).e \\
& \mid \mathsf{run}^{\tau}\ e@p \mid \mathsf{ret}\ e@p \\
& \mid \mathsf{select}^{\tau}\ e\ \mathsf{or}\ e \mid \mathsf{compare}^{\tau}\ e\ \mathsf{and}\ e \mid \mathsf{expect}^{\tau} \\
& \mid \mathsf{split}_{\ell}^{\tau}\ e \mid \mathsf{combine}^{\tau} x = e_1@pc\ \mathsf{in}\ e_2
\end{aligned}
$$

Fig. 29. Type annotated FLAQR Syntax (Full version).

$$
\begin{aligned}
E ::= {} & [\cdot] \mid E\ e \mid v\ E \mid E\ \tau \mid \langle E, e \rangle^{\tau} \mid \langle f, E \rangle^{\tau} \mid \eta_{\ell}\ E \\
& \mid \mathsf{proj}_i\ E \mid \mathsf{inj}_i^{\tau}\ E \mid \mathsf{bind}\ x = E\ \mathsf{in}\ e \\
& \mid \mathsf{case}^{\tau}\ E\ \mathsf{of}\ \mathsf{inj}_1^{\tau}(x).e \mid \mathsf{inj}_2^{\tau}(x).e \\
& \mid \mathsf{ret}\ E@p \\
& \mid \mathsf{select}^{\tau}\ E\ \mathsf{or}\ e \mid \mathsf{select}^{\tau}\ f\ \mathsf{or}\ E \\
& \mid \mathsf{compare}^{\tau}\ E\ \mathsf{and}\ e \mid \mathsf{compare}^{\tau}\ f\ \mathsf{and}\ E \\
& \mid \mathsf{split}_{\ell}^{\tau}\ E \mid \mathsf{combine}^{\tau} x = E@pc\ \mathsf{in}\ e \\
& \mid \mathsf{combine}^{\tau} x = v@pc\ \mathsf{in}\ E
\end{aligned}
$$

Fig. 30. Evaluation context.

Lemma 5 (Γ-Weakening)**.** *If* $\Pi; \Gamma; pc; c \vdash e : \tau$ *and for all* τ' *and* $x \notin dom(\Gamma)$, $\Pi; \Gamma, x{:}\tau'; pc; c \vdash e : \tau$

Proof. By Induction on structure of e. $\square$

Lemma 6 (CTXif)**.** *If* $\Pi; \Gamma; pc; c \vdash E[v] : \tau$ *and* $x \notin dom(\Gamma)$ *then* $\exists \tau'$*, such that* $\Pi; \Gamma, x{:}\tau', pc; c \vdash E[x] : \tau$ *and* $\Pi; \Gamma; pc; c \vdash v : \tau'$.

$$\boxed{e \longrightarrow e'}$$

[E-App] $(\lambda(x{:}\tau)[pc].\, e)\ v \longrightarrow e[x \mapsto v]$ [E-TApp] $(\Lambda X[pc].\, e)\ \tau \longrightarrow e[X \mapsto \tau]$

[E-UnPair] $\mathtt{proj}_i \langle v_1, v_2 \rangle^{\tau} \longrightarrow v_i$ [E-Sealed] $\eta_\ell\ v \longrightarrow (\overline{\eta}_\ell\ v)$

[E-BindM] $\mathtt{bind}\ x = (\overline{\eta}_\ell\ v)\ \mathtt{in}\ e \longrightarrow e[x \mapsto v]$

[E-Case] $(\mathtt{case}^{\tau}\ (\mathtt{inj}_i^{\tau}\ v)\ \mathtt{of}\ \mathtt{inj}_1^{\tau}(x).e_1 \mid \mathtt{inj}_2^{\tau}(x).e_2) \longrightarrow e_i[x \mapsto v]$

[E-Compare] $$\frac{v_1 = v_2}{\mathtt{compare}^{(\ell_1 \oplus \ell_2)\ \mathtt{says}\ \tau}\ (\overline{\eta}_{\ell_1}\ v_1)\ \mathtt{and}\ (\overline{\eta}_{\ell_2}\ v_2) \longrightarrow (\overline{\eta}_{\ell_1 \oplus \ell_2}\ v_1)}$$

[E-CompareFail] $$\frac{v_1 \neq v_2}{\mathtt{compare}^{(\ell_1 \oplus \ell_2)\ \mathtt{says}\ \tau}\ (\overline{\eta}_{\ell_1}\ v_1)\ \mathtt{and}\ (\overline{\eta}_{\ell_2}\ v_2) \longrightarrow \mathtt{fail}^{(\ell_1 \oplus \ell_2)\ \mathtt{says}\ \tau}}$$

[E-CompareFailL] $\mathtt{compare}^{(\ell_1 \oplus \ell_2)\ \mathtt{says}\ \tau}\ \mathtt{fail}^{\ell_1\ \mathtt{says}\ \tau}\ \mathtt{and}\ f_2 \longrightarrow \mathtt{fail}^{(\ell_1 \oplus \ell_2)\ \mathtt{says}\ \tau}$

[E-CompareFailR] $\mathtt{compare}^{(\ell_1 \oplus \ell_2)\ \mathtt{says}\ \tau}\ f_1\ \mathtt{and}\ \mathtt{fail}^{\ell_2\ \mathtt{says}\ \tau} \longrightarrow \mathtt{fail}^{(\ell_1 \oplus \ell_2)\ \mathtt{says}\ \tau}$

[E-Select] $$\frac{f_i = (\overline{\eta}_{\ell_i}\ w_i) \qquad f_j \in \{(\overline{\eta}_{\ell_j}\ v_j),\ \mathtt{fail}^{\ell_j\ \mathtt{says}\ \tau}\}}{\mathtt{select}^{(\ell_1 \ominus \ell_2)\ \mathtt{says}\ \tau}\ f_1\ \mathtt{or}\ f_2 \longrightarrow (\overline{\eta}_{\ell_1 \ominus \ell_2}\ v_i)}$$

[E-SelectFail] $\mathtt{select}^{(\ell_1 \ominus \ell_2)\ \mathtt{says}\ \tau}\ (\mathtt{fail}^{\ell_1\ \mathtt{says}\ \tau})\ \mathtt{or}\ (\mathtt{fail}^{\ell_2\ \mathtt{says}\ \tau}) \longrightarrow \mathtt{fail}^{(\ell_1 \ominus \ell_2)\ \mathtt{says}\ \tau}$

[E-RetStep] $$\frac{e \longrightarrow e'}{\mathtt{ret}\ e@c \longrightarrow \mathtt{ret}\ e'@c}$$ [E-Split] $$\frac{k \text{ is fresh}}{\mathtt{split}_\ell^{\tau}\ v \longrightarrow \langle (\overline{\eta}_{k.L^{\mathsf{c}} \wedge \ell}\ v), (\overline{\eta}_{k.R^{\mathsf{c}} \wedge \ell}\ v) \rangle}$$

[E-Combine] $\mathtt{combine}^{\tau} x = \langle (\overline{\eta}_{k.L \wedge \ell}\ v), (\overline{\eta}_{k.R \wedge \ell}\ v) \rangle @ pc\ \mathtt{in}\ e \longrightarrow e[x \mapsto v]$ [E-Step] $$\frac{e \longrightarrow e'}{E[e] \longrightarrow E[e']}$$

Fig. 31. Full FLAQR local semantics

Proof. By induction on structure of E. □

Lemma 7 (CTXonlyif)**.** $\Pi; \Gamma; pc; c \vdash v : \tau'$ *and* $\Pi; \Gamma, x{:}\tau', pc; c \vdash E[x] : \tau$ *then* $\Pi; \Gamma; pc; c \vdash E[v] : \tau$*, when* $x \notin FV(E)$*(FV returns the free variables).*

Proof. Proof by induction over structure of E. □

Lemma 8 (CTXiff)**.** $\Pi; \Gamma; pc; c \vdash E[v] : \tau$ *iff* $\exists \tau'$*, such that* $\Pi; \Gamma, x{:}\tau', pc; c \vdash E[x] : \tau$ *and* $\Pi; \Gamma; pc; c \vdash v : \tau'$ *when* $x \notin dom(\Gamma)$.

Proof. Straightforward from lemma 6 and 7. □

Lemma 9 (Expect)**.** *If* $\Pi; \Gamma; pc; c \vdash E[v] : \tau$ *and* $\Pi; \Gamma; pc; c \vdash v : \tau'$ *then* $\Pi; \Gamma; pc; c \vdash E[\mathtt{expect}^{\tau'}] : \tau$.

$\boxed{e \longrightarrow \mathtt{fail}^{\tau}}$

[E-AppFailL] $\lambda(x{:}\tau)[pc].\,\mathtt{fail}^{\tau \xrightarrow{pc} \tau'}\ e \longrightarrow \mathtt{fail}^{\tau'}$

[E-AppFail] $\lambda(x{:}\tau)[pc].\,e\ \mathtt{fail}^{\tau} \longrightarrow e[x \mapsto \mathtt{fail}^{\tau}]$ [E-TAppFail] $\mathtt{fail}^{\forall X[pc].\,\tau}\ \tau' \longrightarrow \mathtt{fail}^{\tau[X \mapsto \tau']}$

[E-SealedFail] $\eta_{\ell}\ \mathtt{fail}^{\tau} \longrightarrow \mathtt{fail}^{\ell\ \mathtt{says}\ \tau}$ [E-InjFail] $\mathtt{inj}_i^{(\tau_1+\tau_2)}\ \mathtt{fail}^{\tau_i} \longrightarrow \mathtt{fail}^{(\tau_1+\tau_2)}$

[E-CaseFail] $\mathtt{case}^{\tau}\ \mathtt{fail}^{\tau'}\ \mathtt{of}\ \mathtt{inj}_1(x).e_1 \mid \mathtt{inj}_2(x).e_2 \longrightarrow \mathtt{fail}^{\tau}$

[E-PairFailL] $\langle \mathtt{fail}^{\tau_1}, f_2 \rangle^{(\tau_1 \times \tau_2)} \longrightarrow \mathtt{fail}^{(\tau_1 \times \tau_2)}$ [E-PairFailR] $\langle f_1, \mathtt{fail}^{\tau_2} \rangle^{(\tau_1 \times \tau_2)} \longrightarrow \mathtt{fail}^{(\tau_1 \times \tau_2)}$

[E-ProjFail] $\mathtt{proj}_i\ \mathtt{fail}^{(\tau_1 \times \tau_2)} \longrightarrow \mathtt{fail}^{\tau_i}$

[E-SplitFail] $$\frac{k \text{ is fresh}}{\mathtt{split}_{\ell}\ \mathtt{fail}^{\tau} \longrightarrow \mathtt{fail}^{(L^{\mathsf{c}} \wedge \ell\ \mathtt{says}\ \tau \times R^{\mathsf{c}} \wedge \ell\ \mathtt{says}\ \tau)}}$$

[E-CombineFail] $$\frac{k_1 \neq k_2}{\mathtt{combine}\ x = \langle (\overline{\eta}_{k_1.L^{\mathsf{c}} \wedge \ell}\ v), (\overline{\eta}_{k_2.R^{\mathsf{c}} \wedge \ell}\ v) \rangle @ pc\ \mathtt{in}\ e \longrightarrow e[x \mapsto \mathtt{fail}^{\tau}]}$$

[E-CombineFailL] $\mathtt{combine}\ x = \langle \mathtt{fail}^{L^{\mathsf{c}} \wedge \ell\ \mathtt{says}\ \tau}, f \rangle @ pc\ \mathtt{in}\ e \longrightarrow e[x \mapsto \mathtt{fail}^{\tau}]$

[E-CombineFailR] $\mathtt{combine}\ x = \langle v, \mathtt{fail}^{R^{\mathsf{c}} \wedge \ell\ \mathtt{says}\ \tau} \rangle @ pc\ \mathtt{in}\ e \longrightarrow e[x \mapsto \mathtt{fail}^{\tau}]$

Fig. 32. All propagation of fail terms.

Proof. Straightforward using lemma CTXonlyif 7. □

Lemma 10 (RExpect). *$\Pi; \Gamma; pc; c \vdash v : \tau'$ and $\Pi; \Gamma; pc; c \vdash E[\mathtt{expect}^{\tau'}] : \tau$ then $\Pi; \Gamma; pc; c \vdash E[v] : \tau$.*

Proof. Straightforward from lemma CTXonlyif 7. □

Lemma 11 (Values PC). *Let $\Pi; \Gamma; pc; c \vdash v : \tau$. If $\Pi \Vdash c' \geqslant pc'$ and $\Pi \Vdash c' \geqslant \mathsf{C}(\tau)$ then $\Pi; \Gamma; pc'; c' \vdash v : \tau$.*

Proof. Given that,

$$\Pi \Vdash c' \geqslant pc' \tag{1}$$

$$\Pi \Vdash c' \geqslant \mathsf{C}(\tau) \tag{2}$$

Using induction over values.

Case UnitM. Using Unit typing rule and (1) we have $\Pi; \Gamma; pc'; c' \vdash () : \mathtt{unit}$

$\boxed{\Pi; \Gamma; pc; c \vdash e : \tau}$

$$[\textsc{Var}]\ \frac{\Gamma(x) = \tau \qquad \Pi \Vdash c \succcurlyeq pc}{\Pi; \Gamma; pc; c \vdash x : \tau} \qquad [\textsc{Unit}]\ \frac{\Pi \Vdash c \succcurlyeq pc}{\Pi; \Gamma; pc; c \vdash () : \mathtt{unit}}$$

$$[\textsc{Fail}]\ \frac{\Pi \Vdash c \succcurlyeq pc}{\Pi; \Gamma; pc; c \vdash \mathtt{fail}^{\tau} : \tau} \qquad [\textsc{Lam}]\ \frac{\begin{array}{c}\Pi; \Gamma, x{:}\tau_1; pc'; u \vdash e : \tau_2 \qquad \Pi \Vdash c \succcurlyeq pc \\ u = \mathtt{C}(\tau_1 \xrightarrow{pc'} \tau_2) \qquad \Pi \Vdash c \succcurlyeq u\end{array}}{\Pi; \Gamma; pc; c \vdash \lambda(x{:}\tau_1)[pc'].\, e : \tau_1 \xrightarrow{pc'} \tau_2}$$

$$[\textsc{App}]\ \frac{\begin{array}{c}\Pi; \Gamma; pc; c \vdash e_1 : \tau' \xrightarrow{pc'} \tau \\ \Pi; \Gamma; pc; c \vdash e_2 : \tau' \qquad \Pi \Vdash pc \sqsubseteq pc' \qquad \Pi \Vdash c \succcurlyeq pc\end{array}}{\Pi; \Gamma; pc; c \vdash e_1\ e_2 : \tau}$$

$$[\textsc{TLam}]\ \frac{\begin{array}{c}\Pi; \Gamma, X; pc'; u \vdash e : \tau \qquad \Pi \Vdash c \succcurlyeq pc \\ u = \mathtt{C}(\tau) \qquad \Pi \Vdash c \succcurlyeq u\end{array}}{\Pi; \Gamma; pc; c \vdash \Lambda X[pc'].\, e : \forall X[pc'].\, \tau}$$

$$[\textsc{TApp}]\ \frac{\begin{array}{c}\Pi; \Gamma; pc; c \vdash e : \forall X[pc'].\, \tau \\ \Pi \Vdash pc \sqsubseteq pc' \qquad \Pi \Vdash c \succcurlyeq pc\end{array}}{\Pi; \Gamma; pc; c \vdash (e\ \tau') : \tau[X \mapsto \tau']}\ \tau' \text{ is well-formed in } \Gamma$$

$$[\textsc{Pair}]\ \frac{\begin{array}{c}\Pi; \Gamma; pc; c \vdash e_1 : \tau_1 \qquad \Pi; \Gamma; pc; c \vdash e_2 : \tau_2 \\ \Pi \Vdash c \succcurlyeq pc\end{array}}{\Pi; \Gamma; pc; c \vdash \langle e_1, e_2 \rangle^{(\tau_1 \times \tau_2)} : (\tau_1 \times \tau_2)}$$

$$[\textsc{UnPair}]\ \frac{\Pi; \Gamma; pc; c \vdash e : (\tau_1 \times \tau_2) \qquad \Pi \Vdash c \succcurlyeq pc}{\Pi; \Gamma; pc; c \vdash \mathtt{proj}_i\ e : \tau_i}$$

$$[\textsc{Inj}]\ \frac{\Pi; \Gamma; pc; c \vdash e : \tau_i \qquad \Pi \Vdash c \succcurlyeq pc}{\Pi; \Gamma; pc; c \vdash \mathtt{inj}_i^{(\tau_1+\tau_2)}\ e : (\tau_1 + \tau_2)}$$

$$[\textsc{Case}]\ \frac{\begin{array}{c}\Pi; \Gamma; pc; c \vdash e : (\tau_1 + \tau_2) \qquad \Pi \Vdash pc \sqsubseteq \tau \\ \Pi \Vdash c \succcurlyeq pc \qquad \Pi \Vdash {\tau_i}^a \succcurlyeq \tau^a \\ \Pi; \Gamma, x{:}\tau_1; pc; c \vdash e_1 : \tau \qquad \Pi; \Gamma, x{:}\tau_2; pc; c \vdash e_2 : \tau\end{array}}{\Pi; \Gamma; pc; c \vdash \mathtt{case}^{\tau}\ e\ \mathtt{of}\ \mathtt{inj}_1^{(\tau_1+\tau_2)}(x).e_1 \mid \mathtt{inj}_2^{(\tau_1+\tau_2)}(x).e_2 : \tau}$$

$$[\textsc{UnitM}]\ \frac{\begin{array}{c}\Pi; \Gamma; pc; c \vdash e : \tau \qquad \Pi \Vdash pc \sqsubseteq \ell \\ \Pi \Vdash c \succcurlyeq pc\end{array}}{\Pi; \Gamma; pc; c \vdash \eta_\ell\ e : \ell\ \mathtt{says}\ \tau} \qquad [\textsc{Sealed}]\ \frac{\Pi; \Gamma; pc; c \vdash v : \tau \qquad \Pi \Vdash c \succcurlyeq pc}{\Pi; \Gamma; pc; c \vdash (\overline{\eta}_\ell\ v) : \ell\ \mathtt{says}\ \tau}$$

Fig. 33. Typing rules for expressions (Full version) Part 1/2.

$\boxed{\Pi;\Gamma;pc;c \vdash e : \tau}$

$$[\textsc{BindM}]\ \frac{\Pi;\Gamma;pc;c \vdash e' : \ell\ \mathsf{says}\ \tau' \qquad \Pi \Vdash \ell \sqcup pc \sqsubseteq \tau \qquad \Pi;\Gamma, x{:}\tau';\ell \sqcup pc;c \vdash e : \tau \qquad \Pi \Vdash c \geqslant pc}{\Pi;\Gamma;pc;c \vdash \mathsf{bind}\ x = e'\ \mathsf{in}\ e : \tau}$$

$$[\textsc{Run}]\ \frac{\Pi;\Gamma;pc';c' \vdash e : \tau' \qquad \Pi \Vdash pc \sqsubseteq pc' \qquad \Pi \Vdash c \geqslant pc \qquad \Pi \Vdash c \geqslant \mathsf{C}(\tau') \qquad \tau = pc'^{\mathrm{ia}}\ \mathsf{says}\ \tau'}{\Pi;\Gamma;pc;c \vdash \mathsf{run}^{\tau}\ e@c' : \tau}$$

$$[\textsc{Ret}]\ \frac{\Pi;\Gamma;pc;c \vdash e : \tau \qquad \Pi \Vdash c' \geqslant \mathsf{C}(\tau) \qquad \Pi \Vdash c \geqslant pc}{\Pi;\Gamma;pc;c \vdash \mathsf{ret}\ e@c' : pc^{\mathrm{ia}}\ \mathsf{says}\ \tau}$$

$$[\textsc{Compare}]\ \frac{\forall i \in \{1,2\}.\Pi;\Gamma;pc;c \vdash e_i : \ell_i\ \mathsf{says}\ \tau \qquad \Pi \Vdash c \rhd \ell_i\ \mathsf{says}\ \tau \qquad \Pi \Vdash c \geqslant pc}{\Pi;\Gamma;pc;c \vdash \mathsf{compare}^{(\ell_1 \oplus \ell_2)\ \mathsf{says}\ \tau}\ e_1\ \mathsf{and}\ e_2 : (\ell_1 \oplus \ell_2)\ \mathsf{says}\ \tau}$$

$$[\textsc{Select}]\ \frac{\forall i \in \{1,2\}.\Pi;\Gamma;pc;c \vdash e_i : \ell_i\ \mathsf{says}\ \tau \qquad \Pi \Vdash c \geqslant pc}{\Pi;\Gamma;pc;c \vdash \mathsf{select}^{(\ell_1 \ominus \ell_2)\ \mathsf{says}\ \tau}\ e_1\ \mathsf{or}\ e_2 : (\ell_1 \ominus \ell_2)\ \mathsf{says}\ \tau}$$

$$[\textsc{Expect}]\ \frac{\Pi \Vdash c \geqslant pc}{\Pi;\Gamma;pc;c \vdash \mathsf{expect}^{\tau} : \tau}$$

$$[\textsc{Split}]\ \frac{\Pi;\Gamma;pc;c \vdash e : \tau \qquad \Pi \Vdash c \geqslant pc \qquad \Pi \Vdash pc \sqsubseteq \ell \sqcup \triangle(pc^{\mathrm{i}}) \qquad \tau' = (L^{\mathrm{c}} \wedge \ell\ \mathsf{says}\ \tau \times R^{\mathrm{c}} \wedge \ell\ \mathsf{says}\ \tau)}{\Pi;\Gamma;pc;c \vdash \mathsf{split}_{\ell}^{\tau'}\ e : (L^{\mathrm{c}} \wedge \ell\ \mathsf{says}\ \tau \times R^{\mathrm{c}} \wedge \ell\ \mathsf{says}\ \tau)}$$

$$[\textsc{Combine}]\ \frac{\Pi;\Gamma;pc;c \vdash e : (L^{\mathrm{c}} \wedge \ell\ \mathsf{says}\ \tau \times R^{\mathrm{c}} \wedge \ell\ \mathsf{says}\ \tau) \qquad \Pi \Vdash c \geqslant pc \qquad \Pi;\Gamma, x{:}\tau;\ell \sqcup pc;c \vdash e' : \ell'\ \mathsf{says}\ \tau \qquad \Pi \Vdash \ell \sqcup pc \sqsubseteq \ell'\ \mathsf{says}\ \tau}{\Pi;\Gamma;pc;c \vdash \mathsf{combine}^{\ell'\ \mathsf{says}\ \tau}\ x = e@pc\ \mathsf{in}\ e' : \ell'\ \mathsf{says}\ \tau}$$

$$[\textsc{SealedK}]\ \frac{\Pi;\Gamma;pc;c \vdash v : \tau \qquad \Pi \Vdash c \geqslant pc \qquad K \in \{L^{\mathrm{c}}, R^{\mathrm{c}}\}}{\Pi;\Gamma;pc;c \vdash (\overline{\eta}_{k.K \wedge \ell}\ v) : K \wedge \ell\ \mathsf{says}\ \tau}$$

Fig. 34. Typing rules for expressions (Full version). Part 2/2

Case PAIR. Given

$$\Pi;\Gamma;pc;c \vdash \langle v_1, v_2 \rangle^{(\tau_1 \times \tau_2)} : (\tau_1 \times \tau_2) \tag{3}$$

$$\mathsf{C}(\tau_1 \xrightarrow{pc} \tau_2) = \mathsf{C}(\tau_1) \sqcup pc \sqcup \mathsf{C}(\tau_2)$$
$$\mathsf{C}(\forall X[pc].\,\tau) = pc \sqcup \mathsf{C}(\tau)$$
$$\mathsf{C}(\ell \ \mathsf{says}\ \tau) = \mathsf{C}(\tau)$$
$$\mathsf{C}((\tau_1 + \tau_2)) = \mathsf{C}(\tau_1) \sqcup \mathsf{C}(\tau_2)$$
$$\mathsf{C}((\tau_1 \times \tau_2)) = \mathsf{C}(\tau_1) \sqcup \mathsf{C}(\tau_2)$$
$$\mathsf{C}(\mathsf{unit}) = \bot$$

Fig. 35. Clearance function

$$\mathscr{C}(\mathsf{unit}) = \mathsf{unit}$$
$$\mathscr{C}((\tau_1 + \tau_2)) = (\mathscr{C}(\tau_1) + \mathscr{C}(\tau_2))$$
$$\mathscr{C}((\tau_1 \times \tau_2)) = (\mathscr{C}(\tau_1) \times \mathscr{C}(\tau_2))$$
$$\mathscr{C}(\tau_1 \xrightarrow{pc} \tau_2) = \mathscr{C}(\tau_1) \xrightarrow{pc} \mathscr{C}(\tau_2)$$
$$\mathscr{C}(\Lambda X[pc].\,\tau) = \Lambda X[pc].\,\mathscr{C}(\tau)$$
$$\mathscr{C}((\ell_1 \boxminus \ell_2) \ \mathsf{says}\ \tau) = (\ell_1 \vee \ell_2) \ \mathsf{says}\ \mathscr{C}(\tau)$$
$$\mathscr{C}((\ell_1 \boxplus \ell_2) \ \mathsf{says}\ \tau) = (\ell_1 \wedge \ell_2) \ \mathsf{says}\ \mathscr{C}(\tau)$$
$$\text{(otherwise)}\ \mathscr{C}(\ell \ \mathsf{says}\ \tau) = \ell \ \mathsf{says}\ \mathscr{C}(\tau)$$

Fig. 36. $\mathscr{C}$ function on types.

$$[\text{R-UNIT}]\ \Pi \Vdash p \rhd () \qquad [\text{R-TFUN}]\ \frac{\Pi \Vdash p \rhd \tau}{\Pi \Vdash p \rhd \forall X[pc].\,\tau} \qquad [\text{R-SUM}]\ \frac{\Pi \Vdash p \rhd \tau_1 \quad \Pi \Vdash p \rhd \tau_2}{\Pi \Vdash p \rhd (\tau_1 + \tau_2)}$$

$$[\text{R-PROD}]\ \frac{\Pi \Vdash p \rhd \tau_1 \quad \Pi \Vdash p \rhd \tau_2}{\Pi \Vdash p \rhd (\tau_1 \times \tau_2)} \qquad [\text{R-LBL}]\ \frac{\Pi \Vdash p^{\mathsf{c}} \geqslant \ell^{\mathsf{c}} \quad \Pi \Vdash p \rhd \tau}{\Pi \Vdash p \rhd \ell \ \mathsf{says}\ \tau} \qquad [\text{R-FUN}]\ \frac{\Pi \Vdash p \rhd \tau_1 \quad \Pi \Vdash p \rhd \tau_2}{\Pi \Vdash p \rhd \tau_1 \xrightarrow{pc} \tau_2}$$

Fig. 37. Reads judgments.

Inverting (3) we get

$$\Pi; \Gamma; pc; c \vdash v_1 : \tau_1 \tag{4}$$

and

$$\Pi; \Gamma; pc; c \vdash v_2 : \tau_2 \tag{5}$$

$\boxed{\vdash \ell \sqsubseteq \tau}$

$$[\text{P-UNIT}]\ \Pi \vdash \ell \sqsubseteq \mathtt{unit} \qquad [\text{P-PAIR}]\ \frac{\Pi \vdash \ell \sqsubseteq \tau_1 \qquad \Pi \vdash \ell \sqsubseteq \tau_2}{\Pi \vdash \ell \sqsubseteq (\tau_1 \times \tau_2)}$$

$$[\text{P-FUN}]\ \frac{\Pi \vdash \ell \sqsubseteq \tau_2 \qquad \Pi \vdash \ell \sqsubseteq pc'}{\Pi \vdash \ell \sqsubseteq \tau_1 \xrightarrow{pc'} \tau_2} \qquad [\text{P-TFUN}]\ \frac{\Pi \vdash \ell \sqsubseteq \tau \qquad \Pi \vdash \ell \sqsubseteq pc'}{\Pi \vdash \ell \sqsubseteq \forall X[pc'].\,\tau}$$

$$[\text{P-LBL}]\ \frac{\Pi \Vdash \ell \sqsubseteq \ell'}{\Pi \vdash \ell \sqsubseteq \ell'\ \mathtt{says}\ \tau}$$

Fig. 38. Type protection levels

By applying induction hypothesis on (4) and (5), we get

$$\Pi; \Gamma; pc'; c' \vdash v_1 : \tau_1 \tag{6}$$

and

$$\Pi; \Gamma; pc'; c' \vdash v_2 : \tau_2 \tag{7}$$

From rule PAIR, (6), (7), and (1) we get $\Pi; \Gamma; pc'; c' \vdash \langle v_1, v_2 \rangle^{(\tau_1 \times \tau_2)} : (\tau_1 \times \tau_2)$

Case INJ. Similar to **case PAIR**.

Case SEALED. Given

$$\Pi; \Gamma; pc; c \vdash (\overline{\eta}_\ell\ v) : \ell\ \mathtt{says}\ \tau \tag{8}$$

Inverting (8) we get

$$\Pi; \Gamma; pc; c \vdash v : \tau \tag{9}$$

By applying induction hypothesis on (9) we get

$$\Pi; \Gamma; pc'; c' \vdash v : \tau \tag{10}$$

Thus from rule [SEAL], (1) and (10) we get $\Pi; \Gamma; pc'; c' \vdash (\overline{\eta}_\ell\ v) : \ell\ \mathtt{says}\ \tau$

Case LAM. We have

$$\Pi; \Gamma; pc; c \vdash \lambda(x{:}\tau_1)[pc''].\, e : \tau_1 \xrightarrow{pc''} \tau_2 \tag{11}$$

$\boxed{\Pi \Vdash p \succcurlyeq q}$

$[\textsc{Bot}]\Pi \Vdash p \succcurlyeq \bot \qquad [\textsc{Top}]\Pi \Vdash \top \succcurlyeq p \qquad [\textsc{Refl}]\Pi \Vdash p \succcurlyeq p \qquad [\textsc{Trans}]\dfrac{\Pi \Vdash p \succcurlyeq q \qquad \Pi \Vdash q \succcurlyeq r}{\Pi \Vdash p \succcurlyeq r}$

$[\textsc{Proj}] \quad \dfrac{\Pi \Vdash p \succcurlyeq q}{\Pi \Vdash p^\pi \succcurlyeq q^\pi} \qquad [\textsc{ProjR}] \quad \Pi \Vdash p \succcurlyeq p^\pi$

$[\textsc{ProjIdemp}] \quad \Pi \Vdash (p^\pi)^\pi \succcurlyeq p^\pi \qquad [\textsc{ProjBasis}] \quad \dfrac{\pi \neq \pi'}{\Pi \Vdash \bot \succcurlyeq (p^\pi)^{\pi'}}$

$[\textsc{ProjDistConj}] \quad \Pi \Vdash p^\pi \wedge q^\pi \succcurlyeq (p \wedge q)^\pi \qquad [\textsc{ProjDistDisj}] \quad \Pi \Vdash (p \vee q)^\pi \succcurlyeq p^\pi \vee q^\pi$

$[\textsc{ConjL}]\dfrac{\Pi \Vdash p_k \succcurlyeq p \quad k \in \{1,2\}}{\Pi \Vdash p_1 \wedge p_2 \succcurlyeq p} \qquad [\textsc{ConjR}]\dfrac{\Pi \Vdash p \succcurlyeq p_1 \quad \Pi \Vdash p \succcurlyeq p_2}{\Pi \Vdash p \succcurlyeq p_1 \wedge p_2} \qquad [\textsc{ConjBasis}]\Pi \Vdash p^{\mathsf{c}} \wedge p^{\mathsf{i}} \wedge p^{\mathsf{a}} \succcurlyeq p$

$[\textsc{DisjL}]\dfrac{\Pi \Vdash p_1 \succcurlyeq p \quad \Pi \Vdash p_2 \succcurlyeq p}{\Pi \Vdash p_1 \vee p_2 \succcurlyeq p} \qquad [\textsc{DisjR}]\dfrac{\Pi \Vdash p \succcurlyeq p_k \quad k \in \{1,2\}}{\Pi \Vdash p \succcurlyeq p_1 \vee p_2} \qquad [\textsc{DisjBasis}]\dfrac{\pi \neq \pi'}{\Pi \Vdash \bot \succcurlyeq p^\pi \vee q^{\pi'}}$

$[\textsc{ConjDistDisjL}] \quad \Pi \Vdash (p \wedge q) \vee (p \wedge r) \succcurlyeq p \wedge (q \vee r)$

$[\textsc{ConjDistDisjR}] \quad \Pi \Vdash p \wedge (q \vee r) \succcurlyeq (p \wedge q) \vee (p \wedge r)$

$[\textsc{DisjDistConjL}] \quad \Pi \Vdash (p \vee q) \wedge (p \vee r) \succcurlyeq p \vee (q \wedge r)$

$[\textsc{DisjDistConjR}] \quad \Pi \Vdash p \vee (q \wedge r) \succcurlyeq (p \vee q) \wedge (p \vee r)$

Fig. 39. Static principal lattice rules, adapted from FLAC [8].

Inverting (11) we get

$$\Pi; \Gamma, x{:}\tau_1; pc''; u \vdash e : \tau_2 \tag{12}$$

$$[\textsc{PAndL}]\ \frac{\Pi \Vdash p_i \succcurlyeq p \quad k \in \{1,2\}}{\Pi \Vdash p_1 \boxplus p_2 \succcurlyeq p} \qquad [\textsc{PAndR}]\ \frac{\Pi \Vdash p \succcurlyeq p_1 \quad \Pi \Vdash p \succcurlyeq p_2}{\Pi \Vdash p \succcurlyeq p_1 \boxplus p_2} \qquad [\textsc{AndPAnd}]\ \Pi \Vdash p \wedge q \succcurlyeq p \boxplus q$$

$$[\textsc{PAndPOr}]\ \Pi \Vdash p \boxplus q \succcurlyeq p \boxminus q \qquad [\textsc{ProjPAndL}]\ \Pi \Vdash p^\pi \boxplus q^\pi \succcurlyeq (p \boxplus q)^\pi$$

$$[\textsc{ProjPAndR}]\ \Pi \Vdash (p \boxplus q)^\pi \succcurlyeq p^\pi \boxplus q^\pi \qquad [\textsc{ProjPOrL}]\ \Pi \Vdash p^\pi \boxminus q^\pi \succcurlyeq (p \boxminus q)^\pi$$

$$[\textsc{ProjPOrR}]\ \Pi \Vdash (p \boxminus q)^\pi \succcurlyeq p^\pi \boxminus q^\pi \qquad [\textsc{POrOr}]\ \Pi \Vdash p \boxminus q \succcurlyeq p \vee q$$

$$[\textsc{AndDistPOrR}]\ \Pi \Vdash p \wedge (q \boxminus r) \succcurlyeq (p \wedge q) \boxminus (p \wedge r)$$

$$[\textsc{POrDistAndR}]\ \Pi \Vdash p \boxminus (q \wedge r) \succcurlyeq (p \boxminus q) \wedge (p \boxminus r)$$

$$[\textsc{AndDistPOrL}]\ \Pi \Vdash (p \wedge q) \boxminus (p \wedge r) \succcurlyeq p \wedge (q \boxminus r)$$

$$[\textsc{POrDistAndL}]\ \Pi \Vdash (p \boxminus q) \wedge (p \boxminus r) \succcurlyeq p \boxminus (q \wedge r)$$

$$[\textsc{OrDistPOrR}]\ \Pi \Vdash p \vee (q \boxminus r) \succcurlyeq (p \vee q) \boxminus (p \vee r)$$

$$[\textsc{OrDistPOrL}]\ \Pi \Vdash (p \vee q) \boxminus (p \vee r) \succcurlyeq p \vee (q \boxminus r)$$

$$[\textsc{POrDistOrR}]\ \Pi \Vdash p \boxminus (q \vee r) \succcurlyeq (p \boxminus q) \vee (p \boxminus r)$$

$$[\textsc{POrDistOrL}]\ \Pi \Vdash (p \boxminus q) \vee (p \boxminus r) \succcurlyeq p \boxminus (q \vee r)$$

$$[\textsc{AndDistPAndR}]\ \Pi \Vdash p \wedge (q \boxplus r) \succcurlyeq (p \wedge q) \boxplus (p \wedge r)$$

$$[\textsc{PAndDistAndR}]\ \Pi \Vdash p \boxplus (q \wedge r) \succcurlyeq (p \boxplus q) \wedge (p \boxplus r)$$

$$[\textsc{AndDistPAndL}]\ \Pi \Vdash (p \wedge q) \boxplus (p \wedge r) \succcurlyeq p \wedge (q \boxplus r)$$

$$[\textsc{PAndDistAndL}]\ \Pi \Vdash (p \boxplus q) \wedge (p \boxplus r) \succcurlyeq p \boxplus (q \wedge r)$$

$$[\textsc{OrDistPAndR}]\ \Pi \Vdash p \vee (q \boxplus r) \succcurlyeq (p \vee q) \boxplus (p \vee r)$$

$$[\textsc{OrDistPAndL}]\ \Pi \Vdash (p \vee q) \boxplus (p \vee r) \succcurlyeq p \vee (q \boxplus r)$$

$$[\textsc{PAndDistOrR}]\ \Pi \Vdash p \boxplus (q \vee r) \succcurlyeq (p \boxplus q) \vee (p \boxplus r)$$

$$[\textsc{PAndDistOrL}]\ \Pi \Vdash (p \boxplus q) \vee (p \boxplus r) \succcurlyeq p \boxplus (q \vee r)$$

Fig. 40. FLAQR Partial conjunction and disjunction acts-for rules.

Syntax

$$v ::= \ldots \mid (v \mid v)$$
$$f ::= \ldots \mid (f \mid f)$$
$$e ::= \ldots \mid (e \mid e)$$

$$\begin{aligned}
\lfloor \mathtt{empty} \rfloor_k &= \mathtt{empty} \\
\lfloor \langle e, c\rangle :: s \rfloor_k &= \langle \lfloor e \rfloor_k, c\rangle :: \lfloor s \rfloor_k \\
\lfloor \langle e, c\rangle \,\&\, s \rfloor_k &= \langle \lfloor e \rfloor_k, c\rangle \,\&\, \lfloor s \rfloor_k \\
\lfloor E[\mathtt{expect}^{\tau}] \rfloor_k &= \lfloor E \rfloor_k [\mathtt{expect}^{\tau}] \\
\lfloor E[(e_1 \mid e_2)] \rfloor_k &= \lfloor E \rfloor_k [e_k] \\
\lfloor \mathtt{split}_{\ell}\ e \rfloor_k &= \mathtt{split}_{\ell}\ \lfloor e \rfloor_k \\
\lfloor \mathtt{combine}\ x = e@pc\ \mathtt{in}\ e' \rfloor_k &= \mathtt{combine}\ k = \lfloor e \rfloor_k @pc\ \mathtt{in}\ \lfloor e' \rfloor_k \\
\lfloor \mathtt{compare}^{(\ell_1 \oplus \ell_2)}\ \mathtt{says}^{\tau}\ f_1\ \mathtt{and}\ f_2 \rfloor_k &= \mathtt{compare}^{(\ell_1 \oplus \ell_2)}\ \mathtt{says}^{\tau}\ \lfloor f_1 \rfloor_k\ \mathtt{and}\ \lfloor f_2 \rfloor_k \\
\lfloor \mathtt{select}^{(\ell_1 \ominus \ell_2)}\ \mathtt{says}^{\tau}\ f_1\ \mathtt{or}\ f_2 \rfloor_k &= \mathtt{select}^{(\ell_1 \ominus \ell_2)}\ \mathtt{says}^{\tau}\ \lfloor f_1 \rfloor_k\ \mathtt{or}\ \lfloor f_2 \rfloor_k \\
\lfloor \mathtt{ret}\ e@c \rfloor_k &= \mathtt{ret}\ \lfloor e \rfloor_k @c \\
\lfloor \mathtt{run}^{\tau}\ e@c \rfloor_k &= \mathtt{run}^{\tau}\ \lfloor e \rfloor_k @c \\
\lfloor \mathtt{proj}_i\ e \rfloor_k &= \mathtt{proj}_i\ \lfloor e \rfloor_k \\
\lfloor \mathtt{case}\ e_1\ \mathtt{of}\ \mathtt{inj}_1(x).e_2 \mid \mathtt{inj}_2(x).e_3 \rfloor_k &= \mathtt{case}\ \lfloor e_1 \rfloor_k\ \mathtt{of}\ \mathtt{inj}_1(x).\lfloor e_2 \rfloor_k \mid \mathtt{inj}_2(x).\lfloor e_3 \rfloor_k \\
\lfloor (e \mid \bullet) \rfloor_2 &= \bullet
\end{aligned}$$

Fig. 41. Projection for bracketed expressions.

$$\Pi \Vdash c \geqslant pc \tag{13}$$

$$u = \mathtt{C}(\tau_1 \xrightarrow{pc''} \tau_2) \tag{14}$$

$$\Pi \Vdash c \geqslant \mathtt{C}(\tau_1 \xrightarrow{pc''} \tau_2) \tag{15}$$

Applying IH on 12

$$\Pi; \Gamma, x{:}\tau_1; pc''; u \vdash e : \tau_2 \tag{16}$$

given in lemma statement

$$\Pi \Vdash c' \geqslant pc' \tag{17}$$

$$\Pi \Vdash c' \geqslant \mathtt{C}(\tau_1 \xrightarrow{pc''} \tau_2) \tag{18}$$

$$[\textsc{Bracket}]\ \frac{\begin{array}{c}\Pi \Vdash (H^\pi \sqcup pc) \sqsubseteq pc' \qquad e_1 = v_1 \iff e_2 \neq v_2 \\ \Pi;\Gamma;pc';c \vdash e_1 : \tau \qquad \Pi;\Gamma;pc';c \vdash e_2 : \tau \\ \Pi \vdash H^\pi \sqsubseteq \mathscr{C}(\tau) \qquad \Pi \Vdash c \geqslant pc\end{array}}{\Pi;\Gamma;pc;c \vdash (e_1 \mid e_2) : \tau}$$

$$[\textsc{Bracket-Values}]\ \frac{\begin{array}{c}\Pi;\Gamma;pc;c \vdash v_1 : \tau \qquad \Pi;\Gamma;pc;c \vdash v_2 : \tau \\ \Pi \vdash H^\pi \sqsubseteq \mathscr{C}(\tau) \qquad \Pi \Vdash c \geqslant pc\end{array}}{\Pi;\Gamma;pc;c \vdash (v_1 \mid v_2) : \tau}$$

$$[\textsc{BullR}]\ \frac{\Pi;\Gamma;pc;c \vdash e : \tau}{\Pi;\Gamma;pc;c \vdash (e \mid \bullet) : \tau}\ [\textsc{BullL}]\ \frac{\Pi;\Gamma;pc;c \vdash e : \tau}{\Pi;\Gamma;pc;c \vdash (\bullet \mid e) : \tau}$$

$$[\textsc{Bracket-Fail-L}]\ \frac{\Pi;\Gamma;pc;c \vdash e : \tau}{\Pi;\Gamma;pc;c \vdash (e \mid \mathtt{fail}^\tau) : \tau}$$

$$[\textsc{Bracket-Fail-R}]\ \frac{\Pi;\Gamma;pc;c \vdash e : \tau}{\Pi;\Gamma;pc;c \vdash (\mathtt{fail}^\tau \mid e) : \tau}$$

$$[\textsc{Bracket-Fail-A}]\ \frac{\Pi;\Gamma;pc;c \vdash e_i : \tau \qquad e_i \neq \mathtt{fail}^\tau \qquad \pi = \mathsf{a}}{\Pi;\Gamma;pc;c \vdash (e_1 \mid e_2) : \tau}$$

$$[\textsc{Bracket-Same}]\ \frac{\Pi;\Gamma;pc;c \vdash v : \tau}{\Pi;\Gamma;pc;c \vdash (v \mid v) : \tau}$$

(a) Typing rules for bracketed expressions.

$$[\textsc{Bracket-Stack}]\ \frac{\begin{array}{c}\Pi;\Gamma;pc';c \vdash e : \tau' \qquad \Pi \Vdash pc \sqsubseteq pc' \\ \forall i \in \{1,2\}.\Pi;\Gamma;pc \vdash s_i : [\tau']\tau\end{array}}{\Pi;\Gamma;pc \vdash \langle e, c\rangle \,\&\, (s_1 \mid s_2) : \tau}$$

$$[\textsc{Bracket-Head}]\ \frac{\begin{array}{c}\Pi;\Gamma;pc';c \vdash (e_1 \mid e_2) : \tau' \qquad \Pi \Vdash pc \sqsubseteq pc' \\ \Pi;\Gamma;pc \vdash s : [\tau']\tau\end{array}}{\Pi;\Gamma;pc \vdash \langle (e_1 \mid e_2), c\rangle \,\&\, s : \tau}$$

(b) Typing rules for bracketed configuration stack.

Fig. 42. Bracketed typing rules.

from 17, 18, 16 and LAM rule we get

$$\Pi;\Gamma;pc';c' \vdash \lambda(x{:}\tau_1)[pc''].\, e : \tau_1 \xrightarrow{pc''} \tau_2 \tag{19}$$

(20)

(21)

$$[\text{B-Step}]\quad \frac{e_i \longrightarrow e_i' \qquad e_j' = e_j \qquad \{i,j\} = \{1,2\}}{(e_1 \mid e_2) \longrightarrow (e_1' \mid e_2')} \qquad [\text{B-App}]\quad (v_1 \mid v_2)\ v \longrightarrow (v_1\ \lfloor v \rfloor_1 \mid v_2\ \lfloor v \rfloor_2)$$

$$[\text{B-TApp}]\quad (v \mid v')\ \tau \longrightarrow (v\ \tau \mid v'\ \tau)$$

$$[\text{B-BindM}]\quad \mathsf{bind}\ x = (v \mid v')\ \mathsf{in}\ e \longrightarrow (\mathsf{bind}\ x = v\ \mathsf{in}\ \lfloor e \rfloor_1 \mid \mathsf{bind}\ x = v'\ \mathsf{in}\ \lfloor e \rfloor_2)$$

$$[\text{B-CompareCommon}]\quad \frac{\lfloor \mathsf{compare}^{\ell_1 \oplus \ell_2}\ \mathsf{says}^{\tau}\ (f_{11} \mid f_{12})\ \mathsf{and}\ (f_{21} \mid f_{22}) \rfloor_i \longrightarrow f_i \qquad \forall i \in \{1,2\}}{\mathsf{compare}^{\ell_1 \oplus \ell_2}\ \mathsf{says}^{\tau}\ (f_{11} \mid f_{12})\ \mathsf{and}\ (f_{21} \mid f_{22}) \longrightarrow (f_1 \mid f_2)}$$

$$[\text{B-CompareCommonRight}]\quad \frac{\lfloor \mathsf{compare}^{\ell_1 \oplus \ell_2}\ \mathsf{says}^{\tau}\ (f_{11} \mid f_{12})\ \mathsf{and}\ f \rfloor_i \longrightarrow f_i \qquad \forall i \in \{1,2\}}{\mathsf{compare}^{\ell_1 \oplus \ell_2}\ \mathsf{says}^{\tau}\ (f_{11} \mid f_{12})\ \mathsf{and}\ f \longrightarrow (f_1 \mid f_2)}$$

$$[\text{B-CompareCommonLeft}]\quad \frac{\lfloor \mathsf{compare}^{\ell_1 \oplus \ell_2}\ \mathsf{says}^{\tau}\ f\ \mathsf{and}\ (f_{21} \mid f_{22}) \rfloor_i \longrightarrow f_i \qquad \forall i \in \{1,2\}}{\mathsf{compare}^{\ell_1 \oplus \ell_2}\ \mathsf{says}^{\tau}\ f\ \mathsf{and}\ (f_{21} \mid f_{22}) \longrightarrow (f_1 \mid f_2)}$$

$$[\text{B-SelectCommon}]\quad \frac{\lfloor \mathsf{select}^{\ell_1 \ominus \ell_2}\ \mathsf{says}^{\tau}\ (f_{11} \mid f_{12})\ \mathsf{or}\ (f_{21} \mid f_{22}) \rfloor_i \longrightarrow f_i \qquad \forall i \in \{1,2\}}{\mathsf{select}^{\ell_1 \ominus \ell_2}\ \mathsf{says}^{\tau}\ (f_{11} \mid f_{12})\ \mathsf{or}\ (f_{21} \mid f_{22}) \longrightarrow (f_1 \mid f_2)}$$

$$[\text{B-SelectCommonLeft}]\quad \frac{\lfloor \mathsf{select}^{\ell_1 \ominus \ell_2}\ \mathsf{says}^{\tau}\ (f_{11} \mid f_{12})\ \mathsf{or}\ f \rfloor_i \longrightarrow f_i \qquad \forall i \in \{1,2\}}{\mathsf{select}^{\ell_1 \ominus \ell_2}\ \mathsf{says}^{\tau}\ (f_{11} \mid f_{12})\ \mathsf{or}\ f \longrightarrow (f_1 \mid f_2)}$$

$$[\text{B-SelectCommonRight}]\quad \frac{\lfloor \mathsf{select}^{\ell_1 \ominus \ell_2}\ \mathsf{says}^{\tau}\ f\ \mathsf{or}\ (f_{21} \mid f_{22}) \rfloor_i \longrightarrow f_i \qquad \forall i \in \{1,2\}}{\mathsf{select}^{\ell_1 \ominus \ell_2}\ \mathsf{says}^{\tau}\ f\ \mathsf{or}\ (f_{21} \mid f_{22}) \longrightarrow (f_1 \mid f_2)}$$

$$[\text{B-Fail1}]\ \eta_\ell\ (v \mid \mathsf{fail}^{\tau}) \longrightarrow ((\overline{\eta}_\ell\ v) \mid \mathsf{fail}^{\ell\ \mathsf{says}\ \tau}) \qquad [\text{B-Fail2}]\ \eta_\ell\ (\mathsf{fail}^{\tau} \mid v) \longrightarrow (\mathsf{fail}^{\ell\ \mathsf{says}\ \tau} \mid (\overline{\eta}_\ell\ v))$$

$$[\text{B-Fail}]\quad \eta_\ell\ (\mathsf{fail}^{\tau} \mid \mathsf{fail}^{\tau}) \longrightarrow \mathsf{fail}^{\tau}$$

$$[\text{B-RunLeft}]\ \langle (E[\mathsf{run}^{\tau}\ e_1 @ c'] \mid e_2), c\rangle\ \&\ s \Longrightarrow \langle (\mathsf{ret}\ e_1 @ c \mid \bullet), c'\rangle\ \&\ \langle (E[\mathsf{expect}^{\tau}] \mid e_2), c\rangle :: s$$

$$[\text{B-RunRight}]\ \langle (e_1 \mid E[\mathsf{run}^{\tau}\ e_2 @ c']), c\rangle\ \&\ s \Longrightarrow \langle (\bullet \mid \mathsf{ret}\ e_2 @ c), c'\rangle\ \&\ \langle (e_1 \mid E[\mathsf{expect}^{\tau}]), c\rangle :: s$$

$$[\text{B-RetLeft}]\ \frac{f' = \begin{cases} (\overline{\eta}_\ell\ v) & \text{if } f = v \\ \mathsf{fail}^{\ell\ \mathsf{says}\ \tau} & \text{if } f = \mathsf{fail}^{\tau} \end{cases}}{\langle (\mathsf{ret}\ f @ c \mid \bullet), c'\rangle\ \&\ \langle (E[\mathsf{expect}^{\ell\ \mathsf{says}\ \tau}] \mid e_2), c\rangle :: s \Longrightarrow \langle (E[f'] \mid e_2), c\rangle\ \&\ s}$$

$$[\text{B-RetRight}]\ \frac{f' = \begin{cases} (\overline{\eta}_\ell\ v) & \text{if } f = v \\ \mathsf{fail}^{\ell\ \mathsf{says}\ \tau} & \text{if } f = \mathsf{fail}^{\tau} \end{cases}}{\langle (\bullet \mid \mathsf{ret}\ f @ c), c'\rangle\ \&\ \langle (e_2 \mid E[\mathsf{expect}^{\ell\ \mathsf{says}\ \tau}]), c\rangle :: s \Longrightarrow \langle (e_2 \mid E[f']), c\rangle\ \&\ s}$$

$$[\text{B-RetV}]\ \frac{f_i' = \begin{cases} (\overline{\eta}_\ell\ v) & \text{if } f_i = v \\ \mathsf{fail}^{\ell\ \mathsf{says}\ \tau} & \text{if } f_i = \mathsf{fail}^{\tau} \end{cases}}{\langle \mathsf{ret}\ (f_1 \mid f_2) @ c, c'\rangle\ \&\ \langle E[\mathsf{expect}^{\ell\ \mathsf{says}\ \tau}], c\rangle :: s \Longrightarrow \langle E[(f_1' \mid f_2')], c\rangle\ \&\ s}$$

Fig. 43. Selected bracketed Evaluation Rules.

Case TLAM. Similar to **case LAM**.

Case BRACKET. Given,

$$\Pi; \Gamma; pc; c \vdash (v_1 \mid v_2) : \tau \tag{22}$$

inverting 22 we get

$$\Pi \vdash H^{\pi} \sqsubseteq \mathscr{C}(\tau) \tag{23}$$
$$\Pi; \Gamma; pc; c \vdash v_1 : \tau \tag{24}$$
$$\Pi; \Gamma; pc; c \vdash v_2 : \tau \tag{25}$$

IH on 24 and 25

$$\Pi; \Gamma; pc'; c' \vdash v_1 : \tau \tag{26}$$
$$\Pi; \Gamma; pc'; c' \vdash v_2 : \tau \tag{27}$$

and given in lemma statement,

$$\Pi \Vdash c' \geqslant pc' \tag{28}$$

from 26,27, 28 and 23 we get

$$\Pi; \Gamma; pc'; c' \vdash (v_1 \mid v_2) : \tau \tag{29}$$

□

Lemma 12 (*pc* reduction)**.** *Let* $\Pi; \Gamma; pc; c \vdash e : \tau$. *For all* pc, pc', *such that* $\Pi \Vdash pc' \sqsubseteq pc$ *and* $\Pi \Vdash c \geqslant pc'$ *then* $\Pi; \Gamma; pc'; c \vdash e : \tau$ *holds.*

Proof. Proof is by induction on the derivation of the typing judgment. Given that,

$$\Pi \Vdash pc' \sqsubseteq pc \tag{30}$$
$$\Pi \Vdash c \geqslant pc' \tag{31}$$

Case RUN. From the premises of RUN typing rule
$\Pi; \Gamma; pc; c \vdash \mathsf{run}^{\tau}\ e@c' : \tau$
we get

$$\Pi; \Gamma; pc''; c' \vdash e : \tau' \tag{32}$$
$$\Pi \Vdash pc \sqsubseteq pc'' \tag{33}$$
$$\Pi \Vdash c \geqslant pc \tag{34}$$
$$\Pi \Vdash c \geqslant \mathsf{C}(\tau') \tag{35}$$

where $\tau' = pc''^{\mathsf{ia}}\ \mathtt{says}\ \tau'$.

From 36 and 33 we have

$$\Pi \Vdash pc' \sqsubseteq pc'' \tag{36}$$

From 32, 31 35, 36 we get

$$\tag{37}$$

$\Pi; \Gamma; pc'; c \vdash \mathtt{run}^{\tau}\ e@c' : \tau$
Case RET. From the premises of RET typing rule
$\Pi; \Gamma; pc; c \vdash \mathtt{ret}\ e@c' : pc^{\mathsf{ia}}\ \mathtt{says}\ \tau$
we get

$$\Pi; \Gamma; pc; c \vdash e : \tau \tag{38}$$
$$\Pi \Vdash c' \geqslant (\mathtt{C}(\tau)) \tag{39}$$
$$\Pi \Vdash c \geqslant pc \tag{40}$$

Applying Induction hypothesis to 38 we get

$$\Pi; \Gamma; pc'; c \vdash e : \tau \tag{41}$$

From 41, 39 and 31 we get

$$\tag{42}$$

$\Pi; \Gamma; pc'; c \vdash \mathtt{ret}\ e@c' : pc^{\mathsf{ia}}\ \mathtt{says}\ \tau$
Case COMPARE: Straightforward using IH and 31.
Case SELECT: Straightforward using IH and 31.
Case BRACKET: From the premises Bracket typing rule
$\Pi; \Gamma; pc; c \vdash (e_1 \mid e_2) : \tau$,
we get,

$$\Pi \Vdash (H^{\pi} \sqcup pc) \sqsubseteq pc'' \tag{43}$$
$$e_1 = v_1 \iff e_2 \neq v_2 \tag{44}$$
$$\Pi; \Gamma; pc''; c \vdash e_1 : \tau \tag{45}$$
$$\Pi; \Gamma; pc''; c \vdash e_2 : \tau \tag{46}$$
$$\Pi \vdash H^{\pi} \sqsubseteq \mathscr{C}(\tau) \tag{47}$$
$$\Pi \Vdash c \geqslant pc \tag{48}$$

From and 30 we can write

$$\Pi \Vdash (H^{\pi} \sqcup pc') \sqsubseteq (H^{\pi} \sqcup pc) \tag{49}$$

From 49 and 43 we get

$$\Pi \Vdash (H^{\pi} \sqcup pc') \sqsubseteq pc'' \tag{50}$$

Thus from 50,45,46,47 and 31 we can write

(51)

$\Pi; \Gamma; pc'; c \vdash (e_1 \mid e_2) : \tau$
Case BRACKET-VALUES: Straightforward using IH and 31.
Case BRACKET-* : Straightforward using IH and 31.
Other Cases : Straightforward from 31 and *pc reduction* lemma in [8]. □

Lemma 13 (Clearance)**.** *If* $\Pi; \Gamma; pc; c \vdash e : \tau$ *then* $\Pi \Vdash c \geqslant pc$

Proof. Proof is straightforward by induction on the typing judgments. □

Lemma 14 (Variable substitution)**.** *If* $\Pi; \Gamma, x : \tau'; pc; c \vdash e : \tau$ *and* $\Pi; \Gamma; pc; c \vdash v : \tau'$*, then* $\Pi; \Gamma; pc; c \vdash e[x \mapsto v] : \tau$.

Proof. Proof is by induction on the typing derivation of e.
Case LAM Given,

$$\Pi; \Gamma; pc; c \vdash v : \tau' \tag{52}$$

$$\Pi; \Gamma, x{:}\tau'; pc; c \vdash \lambda(y{:}\tau_1)[pc']. e : \tau_1 \xrightarrow{pc'} \tau_2 \tag{53}$$

inverting 53

$$\Pi; \Gamma, x{:}\tau', y{:}\tau_1; pc'; c \vdash e : \tau_2 \tag{54}$$

$$\Pi \Vdash c \geqslant pc \tag{55}$$

$$\Pi \Vdash c \geqslant \mathsf{C}(\tau_1 \xrightarrow{pc'} \tau_2) \tag{56}$$

Applying lemma values host pc and Weakening lemma in 52

$$\Pi; \Gamma, y{:}\tau_1; pc'; c \vdash v : \tau' \tag{57}$$

IH, 54, 57

$$\Pi; \Gamma, y{:}\tau_1; pc'; c \vdash e[x \mapsto v] : \tau_2 \tag{58}$$

from 58, 55, 56 and LAM rule we get

$$\Pi; \Gamma; pc; c \vdash \lambda(y:\tau_1)[pc'].\, e[x \mapsto v] : \tau_2 \tag{59}$$

$$\tag{60}$$

Case BRACKET Given, $\Pi; \Gamma, x:\tau_1; pc; c \vdash (e_1 \mid e_2) : \tau$ We have to prove t

$$\Pi; \Gamma, x:\tau_1; pc; c \vdash (e_1[x \mapsto \lfloor v \rfloor_1] \mid e_2[x \mapsto \lfloor v \rfloor_2]) : \tau$$

We first describe the case BRACKET. The proof for the case BRACKET-VALUES is analogous. From BRACKET, we have

$$\Pi; \Gamma, x:\tau_1; pc'; c \vdash e_1 : \tau \tag{61}$$

$$\Pi; \Gamma, x:\tau_1; pc'; c \vdash e_2 : \tau \tag{62}$$

$$\Pi \Vdash (H^\pi \sqcup pc) \sqsubseteq pc' \tag{63}$$

$$\Pi \vdash H^\pi \sqsubseteq \mathscr{C}(\tau) \tag{64}$$

Applying clearance (Lemma 13), we have $\Pi \Vdash c \geqslant pc'$. Depending on whether v is a bracket value, we have two cases

Case $v = (v_1 \mid v_2)$:
From BRACKET-VALUES, we have

$$\Pi; \Gamma, x:\tau_1; pc; c \vdash v_1 : \tau' \tag{65}$$

$$\Pi; \Gamma, x:\tau_1; pc; c \vdash v_2 : \tau' \tag{66}$$

$$\Pi \vdash H^\pi \sqsubseteq \mathscr{C}(\tau') \tag{67}$$

Since values can be typed under any pc which acts for the host under which value is typed (Lemma 11), we have $\Pi; \Gamma, x:\tau_1; pc'; c \vdash v_i : \tau'$ for $i = \{1, 2\}$. Applying induction to the premises (61) and (62), we get

$$\Pi; \Gamma, x:\tau_1; pc'; c \vdash e_1[x \mapsto v_1] : \tau \tag{68}$$

$$\Pi; \Gamma, x:\tau_1; pc'; c \vdash e_2[x \mapsto v_2] : \tau \tag{69}$$

and thus from [Bracket] we have

$$\Pi; \Gamma, x:\tau_1; pc; c \vdash (e_1[x \mapsto v_1] \mid e_2[x \mapsto v_2]) : \tau$$

Case $v \neq (v_1 \mid v_2)$: Since values can be typed under any pc which acts for the place under which value is typed (Lemma 11), we have $\Pi; \Gamma, x:\tau_1; pc'; c \vdash v : \tau'$. Applying induction to the premises (61) and (62), we get

$$\Pi; \Gamma, x:\tau_1; pc'; c \vdash e_1[x \mapsto v] : \tau \tag{70}$$

$$\Pi; \Gamma, x:\tau_1; pc'; c \vdash e_2[x \mapsto v] : \tau \tag{71}$$

and thus from [Bracket] rule

$$\Pi; \Gamma, x{:}\tau_1; pc; c \vdash (e_1[x \mapsto v] \mid e_2[x \mapsto v]) : \tau$$

Case COMPARE: Straightforward using IH.
Case SELECT: Straightforward using IH.
Case RUN: Straightforward using IH and values pc lemma. (Same as LAM case.)
Case RET: Straightforward using IH. □

Lemma 15 (Type Substitution)**.** *Let τ' be well-formed in Γ, X, Γ'. If $\Pi; \Gamma, X, \Gamma'; pc; c \vdash e : \tau$ then $\Pi; \Gamma, \Gamma'[X \mapsto \tau']; pc; c \vdash e[X \mapsto \tau'] : \tau[X \mapsto \tau']$.*

Proof. Proof is by the induction on the typing derivation of $\Pi; \Gamma, X, \Gamma'; pc; c \vdash e : \tau$. □

Lemma 16 (Soundness)**.** *If $e \longrightarrow e'$ then $\lfloor e \rfloor_k \longrightarrow^* \lfloor e' \rfloor_k$ for $k \in \{1, 2\}$.*

Proof. By induction on the evaluation of e.
Case B-SPLIT: From rule B-SPLIT $\texttt{split}_\ell\ (f_1 \mid f_2) \longrightarrow f$, then for $k \in \{1, 2\}$ $\lfloor \texttt{split}_\ell\ (f_1 \mid f_2) \rfloor_k \longrightarrow \lfloor f \rfloor_k$

Case B-COMBINE: From rule B-COMBINE $\texttt{combine}\ x = (f_1 \mid f_2)@pc\ \texttt{in}\ e \longrightarrow f$, then for $k \in \{1, 2\}$ $\lfloor \texttt{combine}\ x = (f_1 \mid f_2)@pc\ \texttt{in}\ e \rfloor_k \longrightarrow \lfloor f \rfloor_k$
Case B-COMPARECOMMON: From rule B-COMPARECOMMON we can say if
$\texttt{compare}^{(\ell_1 \oplus \ell_2)}\ \texttt{says}\ \tau\ (f_{11} \mid f_{12})\ \texttt{and}\ (f_{21} \mid f_{22}) \longrightarrow f$
then for $k \in \{1, 2\}$
$\lfloor \texttt{compare}^{(\ell_1 \oplus \ell_2)}\ \texttt{says}\ \tau\ (f_{11} \mid f_{12})\ \texttt{and}\ (f_{21} \mid f_{22}) \rfloor_k$
$\longrightarrow \lfloor f \rfloor_k$

Case B-Compare*: Similar to the case above.
Case B-Select*: Similar to the case above.
Case B-STEP: $\lfloor e \rfloor_i \longrightarrow \lfloor e' \rfloor_i$ and $\lfloor e \rfloor_j = \lfloor e' \rfloor_j$.
Other Cases All other cases in Figure 43 only expand brackets So, $\lfloor e \rfloor_k = \lfloor e' \rfloor_k$ for $k \in \{1, 2\}$. □

Lemma 17 (Stuck expressions)**.** *If e gets stuck then $\lfloor e \rfloor_i$ is stuck for some $i \in \{1, 2\}$.*

Proof. By induction on the structure of e.

Case: $\texttt{split}_\ell\ e$: E-SPLIT can not be applied, i.e. e is not of the form v.That means E-SPLIT can not applied to $\lfloor \texttt{split}_\ell\ e \rfloor_i$. From I.H. $\lfloor e \rfloor_i$ is stuck.
Case: $\texttt{combine}\ x = e_1@pc\ \texttt{in}\ e_2$: E-COMBINE can not be applied, i.e. e is not of the form $\langle (\overline{\eta}_{k.L \wedge \ell}\ v), (\overline{\eta}_{k.R \wedge \ell}\ v) \rangle$.That means E-COMBINE can not be applied to $\lfloor \texttt{combine}\ x = e_1@pc\ \texttt{in}\ e_2 \rfloor_k$. From I.H. $\lfloor e_1 \rfloor_i$ is stuck.
Case: $\texttt{compare}^\tau\ e_1\ \texttt{and}\ e_2$: E-COMPARE can not be applied, i.e. e_1 and/or e_2 are/is not of the form $(\overline{\eta}_\ell\ w)$.That means E-COMPARE can not applied to $\lfloor \texttt{compare}^\tau\ e_1\ \texttt{and}\ e_2 \rfloor_i$. From I.H. either $\lfloor e_1 \rfloor_i$ is stuck or $\lfloor e_2 \rfloor_i$ is stuck for $i \in \{1, 2\}$.

[E-CompareFail*] can not be applied, i.e. e_1 and/or e_2 are/is not of the form $(\overline{\eta}_\ell\ w)$ or $\texttt{fail}^{\tau'}$.That means [E-COMPARE*] can not applied to $\lfloor \texttt{compare}^{\tau}\ e_1\ \texttt{and}\ e_2 \rfloor_i$. From I.H. either $\lfloor e_1 \rfloor_i$ is stuck or $\lfloor e_2 \rfloor_i$ is stuck for $i \in \{1,2\}$.

[B-COMPARE*] can not be applied, i.e. [E-COMPARE*] not be applied to $\lfloor \texttt{compare}^{\tau}\ e_1\ \texttt{and}\ e_2 \rfloor_i$ for $i \in \{1,2\}$. This means $\lfloor \texttt{compare}^{\tau}\ e_1\ \texttt{and}\ e_2 \rfloor_i$ is stuck for $i \in \{1,2\}$. From I.H. we can say either $\lfloor e_1 \rfloor_i$ is stuck or $\lfloor e_2 \rfloor_i$ is stuck for $i \in \{1,2\}$.

Case: $\texttt{select}^{\tau}\ e_1\ \texttt{or}\ e_2$: Same as compare.

Case: $\texttt{ret}\ e\texttt{@}c$: From E-RetStep rule we can say if $\texttt{ret}\ e\texttt{@}c$ is stuck then e is stuck. From I.H. we can say e stuck only when $\lfloor e \rfloor_i$ is stuck for $i \in \{1,2\}$.

Case: $\eta_\ell\ e$: E-Sealed step can not be taken. So E-Sealed step can not be taken for $\lfloor \eta_\ell\ e \rfloor_i$. Which means $\lfloor e \rfloor_i$ is stuck for $i \in \{1,2\}$.
BFail2 and B-Fail1 steps can not be taken. Which means e is not of the forms $(v \mid \texttt{fail}^{\tau})$ or $(\texttt{fail}^{\tau} \mid v)$. Which again, from I.H., implies either $\lfloor e \rfloor_1$ is stuck or $\lfloor e \rfloor_2$ is stuck.

Case $\texttt{proj}_j\ e$: Similar to the above case.

Case $\texttt{inj}_j\ e$: Similar to the above case.

Case $\langle e, e \rangle$: Similar to the above case.

Case $\texttt{case}\ e\ \texttt{of}\ \texttt{inj}_1(x).e_1 \mid \texttt{inj}_2(x).e_2$: Since B-Case , and E-Case are not applicable, it follows that e is not of the form $(v \mid v')$, or $\texttt{inj}_j\ v$. It follows that $\lfloor \texttt{case}\ e\ \texttt{of}\ \texttt{inj}_1(x).e_1 \mid \texttt{inj}_2(x).e_2 \rfloor_i$ is also stuck.

Case $\texttt{bind}\ x = v\ \texttt{in}\ e'$: Similar to the above case.

□

Lemma 18 (Completeness)**.** *If $\lfloor e \rfloor_1 \longrightarrow^* v_1$ and $\lfloor e \rfloor_2 \longrightarrow^* v_2$, then there exists some v such that $e \longrightarrow^* v$.*

Proof. The rules in Figure 43 move brackets out of subterms, and therefore can only be applied a finite number of times. Therefore, by Lemma 16, if e diverges, either $\lfloor e \rfloor_1$ or $\lfloor e \rfloor_2$ diverge; this contradicts our assumption.

Furthermore, by Lemma 17, if the evaluation of e gets stuck, either $\lfloor e \rfloor_1$ or $\lfloor e \rfloor_2$ gets stuck. Therefore, since we assumed $\lfloor e \rfloor_i \longrightarrow^* v_i$, then e must terminate. Thus, there exists some v such that $e \longrightarrow^* v$. □

Lemma 19 (Soundness for global bracketed semantics)**.** *If $\langle e, c \rangle\ \&\ t \implies \langle e', c' \rangle\ \&\ t'$ then $\lfloor \langle e, c \rangle\ \&\ t \rfloor_k \Longrightarrow^* \lfloor \langle e', c' \rangle\ \&\ t' \rfloor_k$ for $k \in \{1,2\}$.*

Proof. If $c = c'$ and $t = t'$ then the lemma statement holds following lemma 16. The cases where the stack and the head of the configuration changes are the interesting ones.

Case B-RetV: Straightforward from [B-RetV] evaluation rule. The premise of the rule says $\forall k \in \{1,2\}$, the kth projection of the expression should take a step.

Case B-Ret* : Same as the above case.

Case B-RunLeft: The lemma trivially holds for $k = 1$.

Case B-RunRight: The lemma trivially holds for $k = 2$.

□

Lemma 20 (Dist Stuck expressions)**.** *If* $\langle e, c\rangle$ & t *gets stuck then* $\lfloor\langle e, c\rangle$ & $t\rfloor_k$ *is stuck for some* $k \in \{1, 2\}$.

Proof. Induction over structure of e.

Case $\langle$`ret` e`@`$c', c\rangle$ & $\langle E[$`expect`$^\tau], c'\rangle$ `::` t**:** This means [E-RetV] and [B-RET*] steps can not be taken. So, [E-RetV] and [B-RetV] steps can not be taken for $\lfloor\langle$`ret` e`@`$c', c\rangle$ & $\langle E[$`expect`$^\tau], c'\rangle$ `::` $t\rfloor_i$. This means e is not of the form v or $(f_1 \mid f_2)$. From I.H. it is clear that $\lfloor v\rfloor_i$ or f_i is stuck for $i \in \{1, 2\}$.

Case $\langle E[$`run`$^\tau$ e`@`$c']$ $, c\rangle$ & t This can always take a E-RUN step and then can get stuck. In that case the argument is same as case `ret` e`@`c as in lemma 17.

Case $\langle(E[$`run`$^\tau$ e`@`$c'] \mid e'), c\rangle$ & t Can not get stuck as $E[$`run`$^\tau$ e`@`$c']$ run can always take a step.

Case $\langle(e' \mid E[$`run`$^\tau$ e`@`$c'])$ $, c\rangle$ & t Can not get stuck as $E[$`run`$^\tau$ e`@`$c']$ run can always take a step.

Other Cases: In all other cases the active configuration and the stack does not change. So lemma statement holds following lemma 17.

□

Lemma 21 (Completeness)**.** *If* $\lfloor\langle e, c\rangle$ & $t\rfloor_1 \longrightarrow^* \langle v_1, c\rangle$ & `empty` *and* $\lfloor\langle e, c\rangle$ & $t\rfloor_2 \longrightarrow^* \langle v_2, c\rangle$ & `empty`*, then there exists some* v *such that* $\langle e, c\rangle$ & $t \longrightarrow^* \langle v, c\rangle$ & `empty`.

Proof. Similar argument as 18 □

Lemma 22 (Label Flowsto SelCmp)**.** *If* $\Pi \Vdash \ell \sqsubseteq \ell_1$ *and* $\Pi \Vdash \ell \sqsubseteq \ell_2$ *then* $\Pi \Vdash \ell \sqsubseteq (\ell_1 \ominus \ell_2)$ *and* $\Pi \Vdash \ell \sqsubseteq (\ell_1 \oplus \ell_2)$

Proof. Given,

$$\Pi \Vdash \ell \sqsubseteq \ell_1 \tag{72}$$

$$\Pi \Vdash \ell \sqsubseteq \ell_2 \tag{73}$$

which implies

$$\Pi \Vdash \ell^{\mathsf{i}} \succcurlyeq \ell_1^{\mathsf{i}} \tag{74}$$

$$\Pi \Vdash \ell^{\mathsf{i}} \succcurlyeq \ell_2^{\mathsf{i}} \tag{75}$$

$$\Pi \Vdash \ell^{\mathsf{a}} \succcurlyeq \ell_1^{\mathsf{a}} \tag{76}$$

$$\Pi \Vdash \ell^{\mathsf{a}} \succcurlyeq \ell_2^{\mathsf{a}} \tag{77}$$

$$\Pi \Vdash \ell_1^{\mathsf{c}} \succcurlyeq \ell^{\mathsf{c}} \tag{78}$$

$$\Pi \Vdash \ell_2^{\mathsf{c}} \succcurlyeq \ell^{\mathsf{c}} \tag{79}$$

74, 75 and R-CONJR implies

$$\Pi \Vdash \ell^{\mathsf{i}} \succcurlyeq \ell_1^{\mathsf{i}} \wedge \ell_2^{\mathsf{i}} \tag{80}$$

from 74, 75 and PAANDR

$$\Pi \Vdash \ell^{\mathsf{i}} \succcurlyeq \ell_1^{\mathsf{i}} \boxplus \ell_2^{\mathsf{i}} \tag{81}$$

78 and CONJL implies

$$\Pi \Vdash \ell_1^{\mathsf{c}} \wedge \ell_2^{\mathsf{c}} \succcurlyeq \ell^{\mathsf{c}} \tag{82}$$

76 and DISJR implies

$$\Pi \Vdash \ell^{\mathsf{a}} \succcurlyeq \ell_1^{\mathsf{a}} \vee \ell_2^{\mathsf{a}} \tag{83}$$

80, 82 and 83 together proves $\Pi \Vdash \ell \sqsubseteq (\ell_1 \oplus \ell_2)$
Similarly we can prove $\Pi \Vdash \ell \sqsubseteq (\ell_1 \ominus \ell_2)$
□

Lemma 23 (Projection Preserves Types)**.** *If* $\Gamma; pc \vdash e : \tau$*, then* $\Gamma; pc \vdash \lfloor e \rfloor_i : \tau$ *for* $i = \{1, 2\}$.

Proof. Proof is by induction on the typing derivation of $\Gamma; pc \vdash e : \tau$. The interesting case is $e = (e_1 \mid e_2)$. By BRACKET, we have $\Pi; \Gamma; pc' \vdash e_i : \tau$ for some pc' such that $\Pi \Vdash (H^{\pi} \sqcup pc^{\pi}) \sqsubseteq pc'^{\pi}$. Therefore, by Lemma 12(*pc* reduction), we have $\Pi; \Gamma; pc \vdash e_i : \tau$. □

Lemma 24 (Subject Reduction(within a host))**.** *Let* $\Pi; \Gamma; pc; c \vdash e : \tau$ *and* $\Pi \Vdash c \succcurlyeq \mathtt{C}(\tau)$*. If* $e \longrightarrow e'$ *then* $\Pi; \Gamma; pc; c \vdash e' : \tau$.

Proof. Case E-APP Given $e = (\lambda(x : \tau_1)[pc'].\, e)\ v$ and $e' = e[x \mapsto v]$. Also $\Pi; \Gamma; pc; c \vdash \lambda(x : \tau_1)[pc'].\, e\ v : \tau_2$. From the premises of APP, we have:

$$\Pi; \Gamma; pc; c \vdash \lambda(x{:}\tau_1)[pc'].\, e : \tau_1 \xrightarrow{pc'} \tau_2 \tag{84}$$

$$\Pi; \Gamma; pc; c \vdash v : \tau_1 \tag{85}$$

$$\Pi \Vdash pc \sqsubseteq pc' \tag{86}$$

From (84), we further have that $\Pi; \Gamma, x : \tau_1; pc'; c \vdash e : \tau_2$. Since (86) holds, we can now apply PC reduction to get $\Pi; \Gamma, x : \tau_1; pc \vdash e : \tau_2$. Applying substitution preservation using (85), we have $\Pi; \Gamma; pc; c \vdash e[x \mapsto v] : \tau_2$.

Case E-BINDM. Given $e = \mathtt{bind}\ x = (\overline{\eta}_{\ell}\ v)\ \mathtt{in}\ e_1$ and $e' = e_1[x \mapsto v]$ and also

$$\Pi; \Gamma; pc; c \vdash \mathtt{bind}\ x = (\overline{\eta}_{\ell}\ v)\ \mathtt{in}\ e_1 : \tau \tag{87}$$

From the premises of (87) we have

$$\Pi; \Gamma; pc; c \vdash (\overline{\eta}_{\ell}\ v) : \ell\ \mathtt{says}\ \tau' \tag{88}$$

$$\Pi; \Gamma; pc; c \vdash v : \tau' \tag{89}$$

$$\Pi; \Gamma, x : \tau'; pc \sqcup \ell; c \vdash e_1 : \tau \tag{90}$$

$$\Pi \vdash pc \sqcup \ell \sqsubseteq \tau \tag{91}$$

$$\Pi \Vdash c \succcurlyeq pc \tag{92}$$

(Since $\Pi \Vdash c \succcurlyeq pc$) applying pc reduction lemma in (90) we get $\Pi; \Gamma, x : \tau'; pc \vdash e_1 : \tau$. Invoking variable substitution lemma, we thus have $\Pi; \Gamma; pc; c \vdash e_1[x \mapsto v] : \tau$.

Case E-RETSTEP

$$\texttt{ret}\ e_1 @ c' \longrightarrow \texttt{ret}\ e_1' @ c' \tag{93}$$

Given, $e = \texttt{ret}\ e_1 @ c'$ and $e' = \texttt{ret}\ e_1' @ c'$ and also

$$\Pi; \Gamma; pc; c \vdash \texttt{ret}\ e_1 @ c' : \tau \tag{94}$$

From the premises of (94) we get

$$\Pi; \Gamma; pc; c \vdash e_1 : \tau' \tag{95}$$

where $\tau = pc^{\text{ia}}\ \texttt{says}\ \tau'$

$$\Pi \Vdash c \succcurlyeq pc \tag{96}$$

$$\Pi \Vdash c' \succcurlyeq \texttt{C}(\tau') \tag{97}$$

and applying induction hypothesis on the premise of (93) we get

$$\Pi; \Gamma; pc; c \vdash e_1' : \tau' \tag{98}$$

From (98), (96) and (97) we have $\Pi; \Gamma; pc; c \vdash \texttt{ret}\ e_1' @ c' : \tau$

E-COMPARE Given, $e = (\texttt{compare}^{\ell_1 \oplus \ell_2}\ \texttt{says}\ ^{\tau}\ (\overline{\eta}_{\ell_1}\ v)\ \texttt{and}\ (\overline{\eta}_{\ell_2}\ v))$ and $e' = (\overline{\eta}_{\ell_1 \oplus \ell_2}\ v)$ and also,

$$\Pi; \Gamma; pc; c \vdash \texttt{compare}^{(\ell_1 \oplus \ell_2)}\ \texttt{says}\ ^{\tau}\ (\overline{\eta}_{\ell_1}\ v)\ \texttt{and}\ (\overline{\eta}_{\ell_2}\ v) : (\ell_1 \oplus \ell_2)\ \texttt{says}\ \tau \tag{99}$$

Inverting (99)

$$\Pi; \Gamma; pc; c \vdash (\overline{\eta}_{\ell_1}\ v) : \ell_1\ \texttt{says}\ \tau \tag{100}$$

$$\Pi; \Gamma; pc; c \vdash (\overline{\eta}_{\ell_2}\ v) : \ell_2\ \texttt{says}\ \tau \tag{101}$$

$$c^{\text{c}} \rhd \ell_1\ \texttt{says}\ \tau \tag{102}$$

$$c^{\text{c}} \rhd \ell_2\ \texttt{says}\ \tau \tag{103}$$

$$\Pi \Vdash c \succcurlyeq pc \tag{104}$$

$$\tag{105}$$

Inverting (100) or (101) further, we get

$$\Pi; \Gamma; pc; c \vdash v : \tau \tag{106}$$

From rule SEALED, (106) and (104) we can say $\Pi; \Gamma; pc; c \vdash (\overline{\eta}_{\ell_1 \oplus \ell_2}\ v) : (\ell_1 \oplus \ell_2)\ \texttt{says}\ \tau$

Case E-COMPAREFAIL*. Trivial, as $\texttt{fail}^{\tau}$ typechecks with any protected type, and based on our type-system τ is always a protected type.

Case E-SELECT. Given, $e = \texttt{select}^{\ell\ \texttt{says}\ \tau}\ (\overline{\eta}_{\ell_1}\ v_1)\ \texttt{or}\ (\overline{\eta}_{\ell_2}\ v_2)$ and $e' = (\overline{\eta}_{\ell}\ v_1)$ where $\ell = \ell_1 \ominus \ell_2$. The following is also given.

$$\Pi; \Gamma; pc; c \vdash \texttt{select}^{\ell\ \texttt{says}\ \tau}\ (\overline{\eta}_{\ell_1}\ v_1)\ \texttt{or}\ (\overline{\eta}_{\ell_2}\ v_2) : \ell\ \texttt{says}\ \tau \tag{107}$$

Inverting (107) we get the following,

$$\Pi; \Gamma; pc; c \vdash (\overline{\eta}_{\ell_1}\ v_1) : \ell_1\ \texttt{says}\ \tau \tag{108}$$

$$\Pi; \Gamma; pc; c \vdash (\overline{\eta}_{\ell_2}\ v_2) : \ell_2\ \texttt{says}\ \tau \tag{109}$$

$$\Pi \Vdash c \geqslant pc \tag{110}$$

$$\tag{111}$$

Further inverting (108) we get,

$$\Pi; \Gamma; pc; c \vdash v_1 : \tau \tag{112}$$

Thus from rule SEALED, (112) and (115) we can argue, $\Pi; \Gamma; pc; c \vdash (\overline{\eta}_{\ell}\ v_1) : \ell\ \texttt{says}\ \tau$

Case E-SELECTL,E-SELECTR . Similar to above. The only diffrence is we need to invert both (108) and (109) and argue both $\Pi; \Gamma; pc; c \vdash (\overline{\eta}_{\ell}\ v_1) : \tau$ and $\Pi; \Gamma; pc; c \vdash (\overline{\eta}_{\ell}\ v_2) : \tau$ holds.

Case E-SELECTFAIL. Trivial, as $\texttt{fail}^{\tau}$ typechecks for any protected type, and based on our type-system τ is always a protected type.

Case B-SELECTCOMMON. This case has a number of sub-cases based on whether f_{ij} is a value or a `fail` term. We will show few cases. Proof of the rest of the combinations will be similar.
Given, $e = \texttt{select}\ ((\overline{\eta}_{\ell_1}\ v_1) \mid (\overline{\eta}_{\ell_1}\ v_1))\ \texttt{or}\ ((\overline{\eta}_{\ell_2}\ v_1') \mid (\overline{\eta}_{\ell_2}\ v_2'))$
and $e' = ((\overline{\eta}_{(\ell_1 \ominus \ell_2)}\ v_1) \mid (\overline{\eta}_{(\ell_1 \ominus \ell_2)}\ v_2))$

$\Pi; \Gamma; pc; c \vdash e : (\ell_1 \ominus \ell_2)\ \texttt{says}\ \tau$
Inverting the above we get,

$$\Pi; \Gamma; pc; c \vdash ((\overline{\eta}_{\ell_1}\ v_1) \mid (\overline{\eta}_{\ell_1}\ v_2)) : \ell_1\ \texttt{says}\ \tau \tag{113}$$

$$\Pi; \Gamma; pc; c \vdash ((\overline{\eta}_{\ell_2}\ v_1') \mid (\overline{\eta}_{\ell_2}\ v_2')) : \ell_2\ \mathsf{says}\ \tau \tag{114}$$

$$\Pi \Vdash c \geqslant pc \tag{115}$$

inverting 113 and 114 we get

$$\Pi \vdash H^\pi \sqsubseteq \mathscr{C}(\ell_1\ \mathsf{says}\ \tau) \tag{116}$$

$$\Pi \vdash H^\pi \sqsubseteq \mathscr{C}(\ell_2\ \mathsf{says}\ \tau) \tag{117}$$

$$\Pi \Vdash (H^\pi \sqcup pc) \sqsubseteq pc' \tag{118}$$

$$\Pi; \Gamma; pc'; c \vdash (\overline{\eta}_{\ell_1}\ v_1) : \ell_1\ \mathsf{says}\ \tau \tag{119}$$

$$\Pi; \Gamma; pc'; c \vdash (\overline{\eta}_{\ell_1}\ v_2) : \ell_1\ \mathsf{says}\ \tau \tag{120}$$

$$\Pi; \Gamma; pc'; c \vdash (\overline{\eta}_{\ell_2}\ v_1') : \ell_2\ \mathsf{says}\ \tau \tag{121}$$

$$\Pi; \Gamma; pc'; c \vdash (\overline{\eta}_{\ell_2}\ v_2') : \ell_2\ \mathsf{says}\ \tau \tag{122}$$

From 116 and 117 and lemma 22 we get

$$\Pi \vdash H^\pi \sqsubseteq \mathscr{C}((\ell_1 \ominus \ell_2)\ \mathsf{says}\ \tau) \tag{123}$$

applying UNITM in 120 120, 121 , 122 we get:

$$\Pi; \Gamma; pc'; c \vdash v_1 : \tau \tag{124}$$

$$\Pi; \Gamma; pc'; c \vdash v_2 : \tau \tag{125}$$

applying pc-reduction lemma , SEALED in 124 and 125 we get

$$\Pi; \Gamma; pc'; c \vdash (\overline{\eta}_{(\ell_1 \ominus \ell_2)}\ v_1) : (\ell_1 \ominus \ell_2)\ \mathsf{says}\ \tau \tag{126}$$

$$\Pi; \Gamma; pc'; c \vdash (\overline{\eta}_{(\ell_1 \ominus \ell_2)}\ v_2) : (\ell_1 \ominus \ell_2)\ \mathsf{says}\ \tau \tag{127}$$

from 126, 127 , 123 and BRACKET-VALUES we get $\Pi; \Gamma; pc; c \vdash ((\overline{\eta}_{\ell_1 \ominus \ell_2}\ v_1) \mid (\overline{\eta}_{(\ell_1 \ominus \ell_2)}\ v_2)) : (\ell_1 \ominus \ell_2)\ \mathsf{says}\ \tau$ Let us do another case, where
$e = \mathtt{select}\ (\mathtt{fail}^{\ell_1\ \mathsf{says}\ \tau} \mid (\overline{\eta}_{\ell_1}\ v'))\ \mathtt{or}\ ((\overline{\eta}_{\ell_2}\ v) \mid \mathtt{fail}^{\ell_2\ \mathsf{says}\ \tau})$
and
$e' = ((\overline{\eta}_{\ell_1 \ominus \ell_2}\ v) \mid (\overline{\eta}_{\ell_1 \ominus \ell_2}\ v'))$
Similar proof as above by inverting using BRACKET rule on
$(\mathtt{fail}^{\ell_1\ \mathsf{says}\ \tau} \mid (\overline{\eta}_{\ell_1}\ v'))$ and $((\overline{\eta}_{\ell_2}\ v) \mid \mathtt{fail}^{\ell_2\ \mathsf{says}\ \tau})$

Case B-SELECTCOMMONLEFT.
$e = \mathtt{select}\ (\mathtt{fail}^{\ell_1\ \mathsf{says}\ \tau} \mid (\overline{\eta}_{\ell_1}\ v))\ \mathtt{or}\ \mathtt{fail}^{\ell_2\ \mathsf{says}\ \tau}$
and
$e' = (\mathtt{fail}^{\ell_1 \ominus \ell_2\ \mathsf{says}\ \tau} \mid (\overline{\eta}_{\ell_1 \ominus \ell_2}\ v))$
Proof is straightforward using BRACKET to invert $(\mathtt{fail}^{\ell_1\ \mathsf{says}\ \tau} \mid (\overline{\eta}_{\ell_1}\ v))$ and then using BRACKET-FAIL-L rule to prove the conclusion.

Let us prove another case where
$e = \texttt{select}\ ((\overline{\eta}_{\ell_1}\ v_1) \mid (\overline{\eta}_{\ell_1}\ v_2))\ \texttt{or}\ (\overline{\eta}_{\ell_2}\ v_3)$
and
$e' = ((\overline{\eta}_{\ell_1 \ominus \ell_2}\ v_1) \mid (\overline{\eta}_{\ell_1 \ominus \ell_2}\ v_2))$
and $\pi = \mathsf{a}$
Proof is straightforward using BRACKET-VALUES to invert
$((\overline{\eta}_{\ell_1}\ v_1) \mid (\overline{\eta}_{\ell_1}\ v_2))$ and then using BRACKET-FAIL-A rule to prove the conclusion.

Case B-COMPARECOMMON:
Similar to B-SELECTCOMMON.

Case B-COMPARECOMMONLEFT
Similar to B-SELECTCOMMONLEFT.

Case B-FAIL1 and B-FAIL2
Straightforward using BRACKET-FAIL-L BRACKET-FAIL-R rules.
Case B-STEP Straightforward using Induction Hypothesis.

Case B-APP: Given $e = (v_1 \mid v_2)\ v'$ and $e' = (v_1\ \lfloor v' \rfloor_1 \mid v_2\ \lfloor v \rfloor_2)$. Also given that $\Gamma; pc \vdash (v_1 \mid v_2)\ v' : \tau_2$ is well-typed, from APP, we have the following:

$$\Gamma; pc \vdash (v_1 \mid v_2) : \tau_1 \xrightarrow{pc''} \tau_2 \tag{128}$$
$$\Gamma; pc \vdash v' : \tau_1 \tag{129}$$
$$\Pi \Vdash pc \sqsubseteq pc'' \tag{130}$$

Thus from BRACKET-VALUES, we have $\Pi \Vdash H^\pi \sqsubseteq \mathscr{C}((\tau_1 \xrightarrow{pc''} \tau_2)^\pi)$. That is, from the definition of type protection (Figure 38), we have $\Pi \Vdash H^\pi \sqsubseteq \mathscr{C}(\tau_1 \xrightarrow{pc''^\pi} \tau_2^\pi)$. From P-FUN, we thus have

$$\Pi \Vdash H^\pi \sqsubseteq \tau_2^\pi \tag{131}$$
$$\Pi \Vdash H^\pi \sqsubseteq pc''^\pi \tag{132}$$

We need to prove

$$\Gamma; pc \vdash (v_1\ \lfloor v' \rfloor_1 \mid v_2\ \lfloor v' \rfloor_2) : \tau_2$$

That is we need the following premises of BRACKET.

$$\Pi; \Gamma; pc' \vdash v_1\ \lfloor v' \rfloor_1 : \tau_2 \tag{133}$$
$$\Pi; \Gamma; pc' \vdash v_2\ \lfloor v' \rfloor_2 : \tau_2 \tag{134}$$
$$\Pi \Vdash H^\pi \sqcup pc^\pi \sqsubseteq pc'^\pi \tag{135}$$
$$\Pi \Vdash H^\pi \sqsubseteq \mathscr{C}(\tau_2^\pi) \tag{136}$$

Let $pc' = pc''$. We have (135) from (130) and (132). We already have (136) from (131). To prove (133), we need the following premises:

$$\Pi; \Gamma; pc' \vdash v_1 : \tau_1 \xrightarrow{pc''} \tau_2 \tag{137}$$

$$\Pi; \Gamma; pc' \vdash \lfloor v' \rfloor_1 : \tau_2 \tag{138}$$

$$\Pi \Vdash pc' \sqsubseteq pc'' \tag{139}$$

The last premise (139) holds trivially (from reflexivity). Applying Lemma 11 (values can be typed under any pc) to (128) we have (137). Applying Lemma 11 (values can be typed under any pc) and Lemma (23) (projection preserves typing) to (129) we have (138). Thus from APP, we have (133). Similarly, (134) holds. Hence proved.

Case B-TAPP Similar to B-APP.

Case B-BINDM Given $e = \texttt{bind}\ x = (\overline{\eta}_\ell\ v_1 \mid \overline{\eta}_\ell\ v_2)\ \texttt{in}\ e$. We have that:

$$e' = (\texttt{bind}\ x = \overline{\eta}_\ell\ v_1\ \texttt{in}\ \lfloor e \rfloor_1 \mid \texttt{bind}\ x = \overline{\eta}_\ell\ v_2\ \texttt{in}\ \lfloor e \rfloor_2)$$

Also $\Gamma; pc \vdash \texttt{bind}\ x = (\overline{\eta}_\ell\ v_1 \mid \overline{\eta}_\ell\ v_2)\ \texttt{in}\ e : \tau$. From BINDM, we have

$$\Gamma; pc \vdash (\overline{\eta}_\ell\ v_1 \mid \overline{\eta}_\ell\ v_2) : \ell\ \texttt{says}\ \tau' \tag{140}$$

$$\Pi; \Gamma, x : \tau'; pc \sqcup \ell \vdash e : \tau \tag{141}$$

$$\Pi \vdash pc \sqcup \ell \sqsubseteq \tau \tag{142}$$

From (140) and BRACKET-VALUES, we have

$$\Gamma; pc \vdash \overline{\eta}_\ell\ v_1 : \ell\ \texttt{says}\ \tau' \tag{143}$$

$$\Gamma; pc \vdash \overline{\eta}_\ell\ v_2 : \ell\ \texttt{says}\ \tau' \tag{144}$$

$$\Pi \Vdash H^\pi \sqsubseteq \mathscr{C}(\ell\ \texttt{says}\ \tau') \tag{145}$$

Inverting 142 we get

$$\Pi \Vdash c \gtrdot pc \sqcup \ell \tag{146}$$

We have to prove that

$$\Gamma; pc \vdash (\texttt{bind}\ x = \overline{\eta}_\ell\ v_1\ \texttt{in}\ \lfloor e \rfloor_1 \mid \texttt{bind}\ x = \overline{\eta}_\ell\ v_2\ \texttt{in}\ \lfloor e \rfloor_2) : \tau$$

For some $\widehat{pc}$ we need the following premises to satisfy BRACKET:

$$\Pi; \Gamma; \widehat{pc} \vdash \texttt{bind}\ x = \overline{\eta}_\ell\ v_1\ \texttt{in}\ \lfloor e \rfloor_1 : \tau \tag{147}$$

$$\Pi; \Gamma; \widehat{pc} \vdash \texttt{bind}\ x = \overline{\eta}_\ell\ v_2\ \texttt{in}\ \lfloor e \rfloor_2 : \tau \tag{148}$$

$$\Pi \Vdash (H^\pi \sqcup pc^\pi) \sqsubseteq \widehat{pc}^\pi \tag{149}$$

$$\Pi \vdash H^\pi \sqsubseteq \mathscr{C}(\tau^\pi) \tag{150}$$

A natual choice for $\widehat{pc}$ is $pc \sqcup \ell$. From Lemma 11 (values can be typed under any pc) and 146, we have

$$\Pi; \Gamma; \widehat{pc} \vdash \overline{\eta}_{\ell}\ v_i : \tau'$$

Applying Lemma 23 (bracket projection preserves typing) to (141), we have

$$\Pi; \Gamma, x : \tau'; \widehat{pc} \vdash \lfloor e \rfloor_i : \tau$$

From BINDM, we therefore have (147) and (148). Applying TRANS to (145) and (142), we have (150). Thus we have all required premises.

Case B-SPLIT

Given $e = \mathtt{split}_{\ell}\ (v_1 \mid v_2)$ and $e' = \langle (\overline{\eta}_{k.L^{\mathsf{c}} \wedge \ell}\ (v_1 \mid v_2)), (\overline{\eta}_{k.R^{\mathsf{c}} \wedge \ell}\ (v_1 \mid v_2)) \rangle$
From applying the SPLIT typying rule on $\mathtt{split}_{\ell}\ (v_1 \mid v_2)$ we get

$$\Gamma; pc \vdash (v_1 \mid v_2) : \tau \tag{151}$$

$$\Pi \Vdash c \geqslant pc \tag{152}$$

$$\Pi \Vdash pc \sqsubseteq \ell \sqcup \triangle(pc^{\mathsf{i}}) \tag{153}$$

Inverting 151 with BRACKET-VALUES we get

$$\Pi; \Gamma; pc; c \vdash v_1 : \tau \tag{154}$$

$$\Pi; \Gamma; pc; c \vdash v_2 : \tau \tag{155}$$

$$\Pi \Vdash H^{\pi} \sqsubseteq \mathscr{C}(\tau) \tag{156}$$

$$\Pi \Vdash c \geqslant pc \tag{157}$$

From 157,151, SEALEDK , and for any k, we get the following:

$$\Pi; \Gamma; pc; c \vdash (\overline{\eta}_{k.L^{\mathsf{c}} \wedge \ell}\ (v_1 \mid v_2)) : L^{\mathsf{c}} \wedge \ell\ \mathtt{says}\ \tau \tag{158}$$

$$\Pi; \Gamma; pc; c \vdash (\overline{\eta}_{k.R^{\mathsf{c}} \wedge \ell}\ (v_1 \mid v_2)) : R^{\mathsf{c}} \wedge \ell\ \mathtt{says}\ \tau \tag{159}$$

From 158, 159, and 157 and PAIR we get

$$\Pi; \Gamma; pc; c \vdash \langle (\overline{\eta}_{k.L^{\mathsf{c}} \wedge \ell}\ (v_1 \mid v_2)), (\overline{\eta}_{k.R^{\mathsf{c}} \wedge \ell}\ (v_1 \mid v_2)) \rangle : (L^{\mathsf{c}} \wedge \ell\ \mathtt{says}\ \tau \times R^{\mathsf{c}} \wedge \ell\ \mathtt{says}\ \tau) \tag{160}$$

Case B-COMBINE

Given, $e = \mathtt{combine}\ x = \langle ((\overline{\eta}_{k.L^{\mathsf{c}} \wedge \ell}\ v_1) \mid (\overline{\eta}_{k.L^{\mathsf{c}} \wedge \ell}\ v_2)), ((\overline{\eta}_{k.R^{\mathsf{c}} \wedge \ell}\ v_1) \mid (\overline{\eta}_{k.R^{\mathsf{c}} \wedge \ell}\ v_2)) \rangle @pc\ \mathtt{in}\ e$ and $e' = (\mathtt{combine}\ x = \langle (\overline{\eta}_{k.L \wedge \ell}\ v_1), (\overline{\eta}_{k.R^{\mathsf{c}} \wedge \ell}\ v_1) \rangle @pc\ \mathtt{in}\ e \mid \mathtt{combine}\ x = \langle (\overline{\eta}_{k.L \wedge \ell}\ v_1), (\overline{\eta}_{k.R^{\mathsf{c}} \wedge \ell}\ v_1) \rangle @pc\ \mathtt{in}\ e)$// From applying the COMBINE typying rule we get

$$\Pi; \Gamma; pc; c \vdash \langle ((\overline{\eta}_{k.L^{\mathsf{c}} \wedge \ell}\ v_1) \mid (\overline{\eta}_{k.L^{\mathsf{c}} \wedge \ell}\ v_2)), ((\overline{\eta}_{k.R^{\mathsf{c}} \wedge \ell}\ v_1) \mid (\overline{\eta}_{k.R^{\mathsf{c}} \wedge \ell}\ v_2)) \rangle : (L^{\mathsf{c}} \wedge \ell\ \mathtt{says}\ \tau \times R^{\mathsf{c}} \wedge \ell\ \mathtt{says}\ \tau) \tag{161}$$

$$\Pi \Vdash c \geqslant pc \tag{162}$$

$$\Pi \Vdash \ell \sqcup pc \sqsubseteq \ell' \ \mathsf{says}\ \tau \tag{163}$$

$$\Pi; \Gamma; pc \sqcup \ell; c \vdash e : \ell' \ \mathsf{says}\ \tau \tag{164}$$

Inverting 161 with PAIR we get

$$\Pi; \Gamma; pc; c \vdash ((\overline{\eta}_{k.L^{\mathsf{c}} \wedge \ell}\ v_1) \mid (\overline{\eta}_{k.L^{\mathsf{c}} \wedge \ell}\ v_2)) : L^{\mathsf{c}} \wedge \ell \ \mathsf{says}\ \tau \tag{165}$$

$$\Pi; \Gamma; pc; c \vdash ((\overline{\eta}_{k.R^{\mathsf{c}} \wedge \ell}\ v_1) \mid (\overline{\eta}_{k.R^{\mathsf{c}} \wedge \ell}\ v_2)) : R^{\mathsf{c}} \wedge \ell \ \mathsf{says}\ \tau \tag{166}$$

Inverting 165 and 166 with BRACKETVALUES and then with SEALEDK we get

$$\Pi; \Gamma; pc; c \vdash v_1 : \tau \tag{167}$$

$$\Pi; \Gamma; pc; c \vdash v_2 : \tau \tag{168}$$

$$\Pi \Vdash H^{\pi} \sqsubseteq \mathscr{C}(L^{\mathsf{c}} \wedge \ell \ \mathsf{says}\ \tau) \tag{169}$$

$$\Pi \Vdash H^{\pi} \sqsubseteq \mathscr{C}(R^{\mathsf{c}} \wedge \ell \ \mathsf{says}\ \tau) \tag{170}$$

From which we can write for $\pi \in \{\mathtt{i}, \mathtt{a}\}$

$$\Pi \Vdash H^{\pi} \sqsubseteq \mathscr{C}(\ell \ \mathsf{says}\ \tau) \tag{171}$$

or, we can just write

$$\Pi \Vdash H^{\pi} \sqsubseteq \ell \tag{172}$$

and when $\pi \in \{\mathtt{i}, \mathtt{a}\}$ we can write

$$\Pi \Vdash H^{\pi} \sqsubseteq \ell \sqcup pc \tag{173}$$

because of 163 we get

$$\Pi \Vdash H^{\pi} \sqsubseteq \ell' \tag{174}$$

which implies

$$\Pi \Vdash H^{\pi} \sqsubseteq \mathscr{C}(\ell' \ \mathsf{says}\ \tau) \tag{175}$$

Inverting 164 we get

$$\Pi \Vdash c \geqslant pc \sqcup \ell \tag{176}$$

Hence a natural choice of the pc' is $pc \sqcup \ell$. From 172 we get

$$\Pi \Vdash H^{\pi} \sqcup pc \sqsubseteq \ell \sqcup pc \tag{177}$$

We already have from 163

$$\Gamma; \Pi; \Gamma; pc' \sqcup \ell; c \vdash e : \ell' \ \mathtt{says}\ pc \tag{178}$$

From 167, 168, 175, 177, 178,

$$\Pi; \Gamma; pc; c \vdash e' : \ell \ \mathtt{says}\ \tau$$

where, $e' = (\mathtt{combine}\ x = \langle(\overline{\eta}_{k.L \wedge \ell}\ v_1), (\overline{\eta}_{k.R^c \wedge \ell}\ v_1)\rangle @ pc\ \mathtt{in}\ e \mid \mathtt{combine}\ x = \langle(\overline{\eta}_{k.L \wedge \ell}\ v_1), (\overline{\eta}_{k.R^c \wedge \ell}\ v_1)\rangle @ pc\ \mathtt{in}\ e)$

□

Lemma 25 (Subject Reduction(inter-host)). *If* $\Pi; \Gamma; pc \vdash \langle e, c\rangle \,\&\, t : \tau$ *and* $\langle e, c\rangle \,\&\, t \Longrightarrow \langle e', c'\rangle \,\&\, t'$ *hold, then* $\Pi; \Gamma; pc \vdash \langle e', c'\rangle \,\&\, t' : \tau$.

Proof. Induction over typing derivation of $\langle e, c\rangle \,\&\, t$.
Case E-RUN. Given, $\langle E[\mathtt{run}^{\hat{\tau}}\ e @ c'], c\rangle \,\&\, t \Longrightarrow$
$\langle \mathtt{ret}\ e @ c, c'\rangle \,\&\, \langle E[\mathtt{expect}^{\hat{\tau}}], c\rangle :: t$

$$\Pi; \Gamma; pc \vdash \langle E[\mathtt{run}^{\hat{\tau}}\ e @ c'], c\rangle \,\&\, t : \tau \tag{179}$$

need to prove

$$\Pi; \Gamma; pc \vdash \langle \mathtt{ret}\ e @ c, c'\rangle \,\&\, \langle E[\mathtt{expect}^{\hat{\tau}}], c\rangle :: t : \tau \tag{180}$$

Inverting 179 we get

$$\Pi; \Gamma; pc'; c \vdash E[\mathtt{run}^{\hat{\tau}}\ e @ c'] : \tau' \tag{181}$$
$$\Pi; \Gamma; pc \vdash t : [\tau']\tau \tag{182}$$
$$\Pi \Vdash c \geqslant pc \tag{183}$$
$$pc \sqsubseteq pc' \tag{184}$$

applying lemma CTX to 181 we get

$$\Pi; \Gamma; pc'; c \vdash \mathtt{run}^{\hat{\tau}}\ e @ c' : \hat{\tau} \tag{185}$$

inverting 185 we get

$$\Pi; \Gamma, \hat{pc}; c' \vdash e : \hat{\tau'} \tag{186}$$

where

$$\hat{\tau'} = \hat{pc}^{\mathrm{ia}}\ \mathtt{says}\ \hat{\tau} \tag{187}$$
$$\Pi \Vdash c \geqslant pc' \tag{188}$$
$$pc' \sqsubseteq \hat{pc} \tag{189}$$
$$\Pi \Vdash c \geqslant \mathtt{C}(\hat{\tau}) \tag{190}$$

applying pc-reduction on 186 with pc we get

$$\Pi; \Gamma, pc; c' \vdash e : \hat{\tau'} \tag{191}$$

applying clearance (Lemma 13) in 191

$$\Pi \Vdash c' \succcurlyeq pc \tag{192}$$

applying clearance lemma in 186

$$\Pi \Vdash c' \succcurlyeq \hat{pc} \tag{193}$$

from 184 and 189 we get

$$pc \sqsubseteq \hat{pc} \tag{194}$$

from 193 and 186 190 and 193 and RET we get

$$\Pi; \Gamma, \hat{pc}; c' \vdash \texttt{ret}\ e@c : \hat{\tau} \tag{195}$$

from 181 and 185 and Expect 9 lemma we get

$$\Pi; \Gamma; pc'; c \vdash E[\texttt{expect}^{\hat{\tau}}] : \tau' \tag{196}$$

from 182, 183, 184,196 and TAIL rule we get

$$\Pi; \Gamma; pc \vdash \langle E[\texttt{expect}^{\hat{\tau}}], c\rangle :: t : [\hat{\tau}]\tau \tag{197}$$

from 194,195,197 183 and HEAD rule we get

$$\Pi; \Gamma; pc \vdash \langle \texttt{ret}\ e@c, c'\rangle \,\&\, \langle E[\texttt{expect}^{\hat{\tau}}], c\rangle :: t : \tau \tag{198}$$

Case E-RETV. Given,
$\langle \texttt{ret}\ v@c, c'\rangle \,\&\, \langle E[\texttt{expect}^{\tau'}], c\rangle :: t \Longrightarrow$
$\langle E[(\overline{\eta}_{pc'^{\texttt{ia}}}\ w)], c\rangle \,\&\, t$

(where $\tau' = pc'^{\texttt{ia}}\ \texttt{says}\ \tau''$)

$$\Pi; \Gamma; pc \vdash \langle \texttt{ret}\ v@c, c'\rangle \,\&\, \langle E[\texttt{expect}^{\tau'}], c\rangle :: t : \tau \tag{199}$$

$$\Pi \Vdash c \succcurlyeq pc \tag{200}$$

need to prove

$$\Pi; \Gamma; pc \vdash \langle E[(\overline{\eta}_{pc'^{\texttt{ia}}}\ w)], c\rangle \,\&\, t : \tau \tag{201}$$

inverting 199 we get

$$\Pi; \Gamma; pc'; c' \vdash \texttt{ret}\ v@c : \tau' \tag{202}$$

$$\Pi; \Gamma; pc \vdash \langle E[\texttt{expect}^{\tau'}], c\rangle :: t : [\tau']\tau \tag{203}$$

$$pc \sqsubseteq pc' \tag{204}$$

$$\Pi \Vdash c' \geqslant pc \tag{205}$$

inverting 202 we get

$$\Pi; \Gamma; pc'; c' \vdash v : \tau'' \tag{206}$$

$$\Pi \Vdash c \geqslant \texttt{C}(\tau'') \tag{207}$$

$$\Pi \Vdash c' \geqslant pc' \tag{208}$$

inverting 203

$$\Pi; \Gamma; \hat{pc}; c \vdash E[\texttt{expect}^{\tau'}] : \hat{\tau} \tag{209}$$

$$pc \sqsubseteq \hat{pc} \tag{210}$$

$$\Pi \Vdash c \geqslant pc \tag{211}$$

$$\Pi; \Gamma; pc \vdash t : [\hat{\tau}]\tau \tag{212}$$

applying clearance lemma on 209

$$\Pi \Vdash c \geqslant \hat{pc} \tag{213}$$

from 213, 206, 207 and ValuesHost lemma

$$\Pi; \Gamma; \hat{pc}; c \vdash v : \tau'' \tag{214}$$

pc reduction Lemma 12 in 214

$$\Pi; \Gamma; pc; c \vdash v : \tau'' \tag{215}$$

applying UNITM rule in 215

$$\Pi; \Gamma; pc; c \vdash (\overline{\eta}_{pc'\texttt{ia}}\ v) : \tau' \tag{216}$$

clearance lemma on 216

$$\Pi \Vdash c \geqslant pc \tag{217}$$

from 216, 209 and RExpect 10 lemma

$$\Pi; \Gamma; \hat{pc}; c \vdash E[(\overline{\eta}_{pc'\texttt{ia}}\ v)] : \hat{\tau} \tag{218}$$

from 218, 210, 212, 217 and HEAD rule we get

$$\Pi; \Gamma; pc \vdash \langle E[(\overline{\eta}_{pc'^{\mathsf{ia}}}\ v)], c\rangle \ \&\ t : \tau \tag{219}$$

Case E-DSTEP. Straightforward using Induction hypothesis
Case B-RUNLEFT. Given, $\langle (E[\mathsf{run}^{\hat{\tau}}\ e@c'] \mid e_2), c\rangle \ \&\ t$
$\Longrightarrow \langle (\mathsf{ret}\ e@c \mid \bullet), c'\rangle \ \&\ \langle (E[\mathsf{expect}^{\hat{\tau}}] \mid e_2), c\rangle :: t$
and

$$\Pi; \Gamma; pc \vdash \langle (E[\mathsf{run}^{\hat{\tau}}\ e@c'] \mid e_2), c\rangle \ \&\ t : \tau \tag{220}$$

need to prove
$\Pi; \Gamma; pc \vdash \langle (\mathsf{ret}\ e@c \mid \bullet), c'\rangle \ \&\ \langle (E[\mathsf{expect}^{\hat{\tau}}] \mid e_2), c\rangle :: t : \tau$

Inverting 220 we get

$$\Pi; \Gamma; pc'; c \vdash (E[\mathsf{run}^{\hat{\tau}}\ e@c'] \mid e_2) : \tau' \tag{221}$$
$$\Pi; \Gamma; pc \vdash t : [\tau']\tau \tag{222}$$
$$\Pi \Vdash c \geqslant pc \tag{223}$$
$$pc \sqsubseteq pc' \tag{224}$$

inverting 221 we get

$$\Pi; \Gamma; pc'', c \vdash E[\mathsf{run}^{\hat{\tau}}\ e@c'] : \tau' \tag{225}$$
$$\Pi; \Gamma; pc'', c \vdash e_2 : \tau' \tag{226}$$
$$\Pi \Vdash H^{\pi} \sqsubseteq \mathscr{C}(\tau') \tag{227}$$
$$\Pi \Vdash H^{\pi} \sqcup pc' \sqsubseteq pc'' \tag{228}$$

applying lemma CTX to 225 we get

$$\Pi; \Gamma; pc''; c \vdash \mathsf{run}^{\hat{\tau}}\ e@c' : \hat{\tau} \tag{229}$$

inverting 229 we get

$$\Pi; \Gamma, \hat{pc}; c' \vdash e : \hat{\tau'} \tag{230}$$

where

$$\hat{\tau'} = \hat{pc}^{\mathsf{ia}}\ \mathsf{says}\ \hat{\tau} \tag{231}$$
$$\Pi \Vdash c \geqslant pc'' \tag{232}$$
$$pc'' \sqsubseteq \hat{pc} \tag{233}$$
$$\Pi \Vdash c \geqslant \mathsf{C}(\hat{\tau}) \tag{234}$$

applying pc-reduction on 230 with pc we get

$$\Pi; \Gamma, pc'; c' \vdash e : \hat{\tau'} \tag{235}$$

applying clearance in 235

$$\Pi \Vdash c' \succcurlyeq pc' \tag{236}$$

applying clearance lemma in 230

$$\Pi \Vdash c' \succcurlyeq \hat{pc} \tag{237}$$

from 224, 225 and 233 we get

$$pc \sqsubseteq \hat{pc} \tag{238}$$

from 237 and 230 234 and 237 and RET we get

$$\Pi; \Gamma, \hat{pc}; c' \vdash \texttt{ret}\ e@c : \hat{\tau} \tag{239}$$

from 225 and 229 and Expect 9 lemma we get

$$\Pi; \Gamma; pc''; c \vdash E[\texttt{expect}^{\hat{\tau}}] : \tau' \tag{240}$$

from 226, 227, 228, 241 we get

$$\Pi; \Gamma; pc''; c \vdash (E[\texttt{expect}^{\hat{\tau}}] \mid e_2) : \tau' \tag{241}$$

from 242 and BULLL rule

$$\Pi; \Gamma, \hat{pc}; c' \vdash (\texttt{ret}\ e@c \mid \bullet) : \hat{\tau} \tag{242}$$

from 222, 223, 224,241 and TAIL rule we get

$$\Pi; \Gamma; pc \vdash \langle (E[\texttt{expect}^{\hat{\tau}}] \mid e_2), c \rangle :: t : [\hat{\tau}]\tau \tag{243}$$

from 238,242,243 223 and HEAD rule we get
$\Pi; \Gamma; pc \vdash \langle (\texttt{ret}\ e@c \mid \bullet), c' \rangle \,\&\, \langle (E[\texttt{expect}^{\hat{\tau}}] \mid e_2), c \rangle :: t : \tau$

Case B-RUNRIGHT. Same as above.
Case B-RETLEFT Same as above.
Case B-RETRIGHT Same as above.
Case B-RETV Same as above.

□

Theorem 3 (c-i Noninterference)**.** *If* $\Pi; \Gamma, x : \ell'\ \texttt{says}\ \tau' \vdash \langle e, c \rangle \,\&\, \texttt{empty} : \ell\ \texttt{says}\ \tau$ *where*

$$
\begin{array}{ll}
\mathcal{O}(\mathtt{fail}^{\tau}, \Pi, p, \pi) & = \circ \\
\mathcal{O}(x, \Pi, p, \pi) & = x \\
\mathcal{O}((), \Pi, p, \pi) & = \circ \\
\mathcal{O}(\eta_{\ell}\ e, \Pi, p, \pi) & = \eta_{\ell}\ \mathcal{O}(e, \Pi, p, \pi) \\
\mathcal{O}(\overline{\eta}_{\ell}\ v, \Pi, p, \pi) & = \begin{cases} \overline{\eta}_{\ell}\ \mathcal{O}(v, \Pi, p, \pi) & \Pi \Vdash \ell^{\pi} \sqsubseteq p^{\pi} \\ \circ & \text{otherwise} \end{cases} \\
\mathcal{O}(\lambda(x{:}\tau)[pc].\, e, \Pi, p, \pi) & = \begin{cases} \lambda(x{:}\tau)[pc].\, \mathcal{O}(e, \Pi, p, \pi) & \Pi \Vdash pc^{\pi} \sqsubseteq p^{\pi} \\ \circ & \text{otherwise} \end{cases} \\
\mathcal{O}(\Lambda X[pc].\, e, \Pi, p, \pi) & = \begin{cases} \Lambda X[pc].\, \mathcal{O}(e, \Pi, p, \pi) & \Pi \Vdash pc^{\pi} \sqsubseteq p^{\pi} \\ \circ & \text{otherwise} \end{cases} \\
\mathcal{O}(e\ e', \Pi, p, \pi) & = \mathcal{O}(e, \Pi, p, \pi)\ \mathcal{O}(e', \Pi, p, \pi) \\
\mathcal{O}(\langle e_1, e_2 \rangle, \Pi, p, \pi) & = \begin{cases} \circ & \mathcal{O}(e_i, \Pi, p, \pi) = \circ \\ \langle \mathcal{O}(e_1, \Pi, p, \pi), \mathcal{O}(e_2, \Pi, p, \pi) \rangle & \text{otherwise} \end{cases} \\
\mathcal{O}(\mathtt{proj}_i\ e, \Pi, p, \pi) & = \mathtt{proj}_i\ \mathcal{O}(e, \Pi, p, \pi) \\
\mathcal{O}(\mathtt{inj}_i\ e, \Pi, p, \pi) & = \begin{cases} \circ & \mathcal{O}(e, \Pi, p, \pi) = \circ \\ \mathtt{inj}_i\ \mathcal{O}(e, \Pi, p, \pi) & \text{otherwise} \end{cases} \\
\mathcal{O}(\mathtt{case}\ e\ \mathtt{of}\ \mathtt{inj}_1(x).e_1 \mid \mathtt{inj}_2(x).e_2, \Pi, p, \pi) & = \mathtt{case}\ \mathcal{O}(e, \Pi, p, \pi)\ \mathtt{of} \\
 & \quad \mathtt{inj}_1\ .\ \mathcal{O}(e_1, \Pi, p, \pi) \\
 & \quad \mid \mathtt{inj}_2\ .\ \mathcal{O}(e_2, \Pi, p, \pi) \\
\mathcal{O}(\mathtt{bind}\ x = e\ \mathtt{in}\ e', \Pi, p, \pi) & = \mathtt{bind}\ x = \mathcal{O}(e, \Pi, p, \pi)\ \mathtt{in}\ \mathcal{O}(e', \Pi, p, \pi) \\
\mathcal{O}(\mathtt{select}\ e_1\ \mathtt{or}\ e_2, \Pi, p, \pi) & = \mathtt{select}\ \mathcal{O}(e_1, \Pi, p, \pi)\ \mathtt{or}\ \mathcal{O}(e_2, \Pi, p, \pi) \\
\mathcal{O}(\mathtt{compare}\ e_1\ \mathtt{and}\ e_2, \Pi, p, \pi) & = \mathtt{compare}\ \mathcal{O}(e_1, \Pi, p, \pi)\ \mathtt{and}\ \mathcal{O}(e_2, \Pi, p, \pi) \\
\mathcal{O}(\langle e, c \rangle\ \&\ s, \Pi, \ell, \pi) & = \begin{cases} \mathcal{O}(e, \Pi, \ell, \pi) & s = \mathtt{empty} \\ \mathcal{O}(e, \Pi, \ell, \pi) \& \mathcal{O}(s, \Pi, \ell, \pi) & \end{cases} \\
\mathcal{O}(\langle e, c \rangle :: s, \Pi, \ell, \pi) & = \begin{cases} \mathcal{O}(e, \Pi, \ell, \pi) & s = \mathtt{empty} \\ \mathcal{O}(e, \Pi, \ell, \pi) :: \mathcal{O}(s, \Pi, \ell, \pi) & \end{cases} \\
\mathcal{O}(E[\mathtt{run}^{\tau}\ e@c], \Pi, \ell, \pi) & = \mathcal{O}(E[e], \Pi, \ell, \pi) \\
\mathcal{O}(\mathtt{ret}\ e@c, \Pi, \ell, \pi) & = \mathcal{O}(e, \Pi, \ell, \pi) \\
\mathcal{O}(\mathtt{split}_l\ e, \Pi, \ell, \pi) & = \mathtt{split}_l\ \mathcal{O}(e, \Pi, \ell, \pi) \\
\mathcal{O}(\mathtt{combine}\ x = \langle e_1, e_2 \rangle @ pc\ \mathtt{in}\ e, \Pi, \ell, \pi) & = \mathtt{combine}\ x = \langle \mathcal{O}(e_1, \Pi, \ell, \pi), \mathcal{O}(e_2, \Pi, \ell, \pi) \rangle @ pc\ \mathtt{in}\ \mathcal{O}(e, \Pi, \ell, \pi)
\end{array}
$$

Fig. 44. Observation function for intermediate FLAQR terms.

(1) $\Pi; \Gamma; pc; c \vdash v_i : \ell'\ \mathtt{says}\ \tau'$, $i \in \{1, 2\}$

(2) $\langle e[x \mapsto (v_1 \mid v_2)], c \rangle\ \&\ \mathtt{empty} \longrightarrow^{*} \langle v, c \rangle\ \&\ \mathtt{empty}$

(3) $\Pi \Vdash H^{\pi} \sqsubseteq \ell'$ *and* $\Pi \not\Vdash H^{\pi} \sqsubseteq \ell$, $\pi \in \{\mathtt{c}, \mathtt{i}\}$.

then, $\mathcal{O}(\lfloor v \rfloor_1, \Pi, \ell, \pi) = \mathcal{O}(\lfloor v \rfloor_2, \Pi, \ell, \pi)$

Proof. From Subject reduction of bracketed FLAQR constructs we can write

$$\Pi; \Gamma; pc \vdash \langle v, c \rangle\ \&\ \mathtt{empty} : \ell\ \mathtt{says}\ \tau$$

We will write $\lfloor v \rfloor_i$ as v_i. We need to show $\mathcal{O}(v_1, \Pi, \ell, \pi) = \mathcal{O}(v_2, \Pi, \ell, \pi)$. Since v_i has protected type, we know that it is of form $(\overline{\eta}_\ell\ v_i')$ Since $\Pi \Vdash \ell^\pi \sqsubseteq \ell^\pi$ for $\pi \in \{\mathtt{c}, \mathtt{i}\}$, we just have to show
$(\overline{\eta}_\ell\ \mathcal{O}(v_1', \Pi, \ell, \pi)) = (\overline{\eta}_\ell\ \mathcal{O}(v_2', \Pi, \ell, \pi))$
Which is true if we can show
$\mathcal{O}(v_1', \Pi, \ell, \pi) = \mathcal{O}(v_2', \Pi, \ell, \pi)$.
Which can be easily shown by induction over structure of v_is. □

Theorem 4 (Availability Noninterference)**.** *If*
$\Pi; \Gamma, x : \ell\ \mathtt{says}\ \tau' \vdash \langle e, c\rangle\ \&\ \mathtt{empty} : \ell_{\mathcal{Q}}\ \mathtt{says}\ \tau$ *where*

(1) $\Pi; \Gamma; pc; c \vdash f_i : \ell\ \mathtt{says}\ \tau', i \in \{1, 2\}$
(2) $\langle e[x \mapsto (f_1 \mid f_2)], c\rangle\ \&\ \mathtt{empty} \longrightarrow^* \langle f, c\rangle\ \&\ \mathtt{empty}$
(3) $\Pi \Vdash H \succcurlyeq \ell\ \mathtt{says}\ \tau'$ *and* $H^{\mathtt{ia}} \in \mathcal{A}_{[\![\mathcal{Q}]\!]}$ *and*
$\Pi \Vdash \mathcal{Q}$ *guards* $(\ell_{\mathcal{Q}}\ \mathtt{says}\ \tau)$

then $\lfloor f \rfloor_1 \neq \mathtt{fail}^{\ell_{\mathcal{Q}}\ \mathtt{says}\ \tau} \iff \lfloor f \rfloor_2 \neq \mathtt{fail}^{\ell_{\mathcal{Q}}\ \mathtt{says}\ \tau}$

Proof. From subject reduction (24 and 25) we know, $\Pi; \Gamma; pc; c \vdash \lfloor f \rfloor_i : \ell_{\mathcal{Q}}\ \mathtt{says}\ \tau$. Because $\Pi \Vdash \mathcal{Q}$ *guards* $(\ell_{\mathcal{Q}}\ \mathtt{says}\ \tau)$ and $H^{\mathtt{ia}} \in \mathcal{A}_{[\![\mathcal{Q}]\!]}$ we can write $\Pi \not\Vdash H^{\mathtt{ia}} \succcurlyeq \ell_{\mathcal{Q}}\ \mathtt{says}\ \tau$ from rule Q-GUARD . This ensures if $\lfloor f \rfloor_1 \neq \mathtt{fail}^{\ell_{\mathcal{Q}}\ \mathtt{says}\ \tau}$, then $\lfloor f \rfloor_2 \neq \mathtt{fail}^{\ell_{\mathcal{Q}}\ \mathtt{says}\ \tau}$, and vice-versa. □

Appendix B. Correctness of Blame Semantics.

In the following proofs, a possible faulty set $\mathcal{F}$ is referred as a faulty set that satisfies the blame constraint, or $\mathcal{F}$ implied by $\mathcal{C}$. And $\Pi \Vdash b \succcurlyeq \tau$ is equivalent to saying $\Pi \Vdash b^{\mathtt{ia}} \succcurlyeq \tau$.

Lemma 26 (Reach ConjL)**.** *If* $\Pi \Vdash b^{\mathtt{ia}} \geqslant t$ *then* $\Pi \Vdash b \wedge p^{\mathtt{ia}} \geqslant t$

Proof. Given, $\Pi \Vdash b^{\mathtt{ia}} \geqslant t$, which means $t^{\mathtt{c}} = \bot$. So from CONJL we can write: $\Pi \Vdash b^{\mathtt{ia}} \wedge p^{\mathtt{ia}} \geqslant t$, which is same as writing $\Pi \Vdash b \wedge p^{\mathtt{ia}} \geqslant t$. □

Lemma 27 (Reach ConjR Type)**.** *If* $\Pi \Vdash b \succcurlyeq \ell_1\ \mathtt{says}\ \tau$ *and* $\Pi \Vdash b \succcurlyeq \ell_2\ \mathtt{says}\ \tau$ *then* $\Pi \Vdash b \succcurlyeq (\ell_1 \ominus \ell_2)\ \mathtt{says}\ \tau$

Proof. If $\Pi \Vdash b \succcurlyeq \tau$ then $\Pi \Vdash b \succcurlyeq (\ell_1 \ominus \ell_2)\ \mathtt{says}\ \tau$ is true as well from A-TYPE. If $\Pi \not\Vdash b \succcurlyeq \tau$ then it is obvious that $\Pi \Vdash b^{\mathtt{a}} \geqslant \ell_1^{\mathtt{a}}$ and $\Pi \Vdash b^{\mathtt{a}} \geqslant \ell_2^{\mathtt{a}}$. Which along with R-CONJR implies, $\Pi \Vdash b^{\mathtt{a}} \geqslant \ell_1^{\mathtt{a}} \wedge \ell_2^{\mathtt{a}}$. Thus from A-AVAIL we can write $\Pi \Vdash b \succcurlyeq (\ell_1 \ominus \ell_2)\ \mathtt{says}\ \tau$ □

Lemma 28 (FAIL RESULT ONESTEP TO FAIL)**.** *If* $\langle\langle e, c\rangle\ \&\ s\rangle^{\mathcal{C}} \longrightarrow \langle\langle \mathtt{fail}^{\tau}, c'\rangle\ \&\ s'\rangle^{\mathcal{C}'}$ *and e is a source level term, then e must be of the form* $\mathtt{compare}^{\tau}\ v_1\ \mathtt{and}\ v_2$ *and the evaluation step that has been taken to transition from e to* $\mathtt{fail}^{\tau}$ *is either* C-COMPAREFAIL *or* C-COMBINEFAIL.

Proof. Trivial(by inspection on evaluation rules). □

Lemma 29 (FAIL SUBEXP ONESTEP)**.** *If* $\langle\langle e, c\rangle\ \&\ s\rangle^{\mathcal{C}} \longrightarrow \langle\langle e', c'\rangle\ \&\ s'\rangle^{\mathcal{C}'}$ *and e is a source level term but* e' *has a* `fail` *term in it then the evaluation step that has been taken is either* C-COMPAREFAIL *or* C-COMBINEFAIL.

$$
\begin{array}{l}
\mathcal{L}(x, y, \mathcal{C}, \ell_1, \ell_2) = \ \texttt{match}\ (x, y)\ \texttt{with} \\
\quad \mid ((\overline{\eta}_\ell\ v_1), (\overline{\eta}_\ell\ v_2)) = \\
\qquad \texttt{if}\ \mathcal{C} \models \ell_1 \in \mathcal{F}\ \texttt{then}\ \mathcal{C} \\
\qquad \texttt{else if}\ \mathcal{C} \models \ell_2 \in \mathcal{F}\ \texttt{then}\ \mathcal{C} \\
\qquad \texttt{else if}\ \mathcal{C} \models \ell \in \mathcal{F}\ \texttt{then}\ \mathcal{C} \\
\qquad \texttt{else}\ \mathcal{L}(v_1, v_2, \mathcal{C}, \ell, \ell) \\
\quad \mid (\eta_\ell\ e_1, \eta_\ell\ e_2) = \mathcal{L}(e_1, e_2, \mathcal{C}, \ell_1, \ell_2) \\
\quad \mid (\texttt{inj}_i^\tau\ e_1, \texttt{inj}_i^\tau\ e_2) = \mathcal{L}(e_1, e_2, \mathcal{C}, \ell_1, \ell_2) \\
\quad \mid (\langle e_{11}, e_{12}\rangle^\tau, \langle e_{21}, e_{22}\rangle^\tau) = \\
\qquad \mathcal{L}(e_{11}, e_{21}, (\mathcal{L}(e_{12}, e_{22}, \mathcal{C}, \ell_1, \ell_2)), \ell_1, \ell_2) \\
\quad \mid (\texttt{run}^\tau\ e_1 @ p, \texttt{run}^\tau\ e_2 @ p) = \mathcal{L}(e_1, e_2, \mathcal{C}, \ell_1, \ell_2) \\
\quad \mid (\texttt{select}^\tau\ \ e_1\ \texttt{or}\ e_2, \texttt{select}^\tau\ \ e_1'\ \texttt{or}\ e_2') = \\
\qquad \mathcal{L}(e_1, e_1', (\mathcal{L}(e_2, e_2', \mathcal{C}, \ell_1, \ell_2)), \ell_1, \ell_2) \\
\quad \mid (\texttt{compare}^\tau\ e_1\ \texttt{and}\ e_2, \texttt{compare}^\tau\ e_1'\ \texttt{and}\ e_2') = \\
\qquad \mathcal{L}(e_1, e_1', (\mathcal{L}(e_2, e_2', \mathcal{C}, \ell_1, \ell_2)), \ell_1, \ell_2) \\
\quad \mid (\lambda(x{:}\tau)[pc].\, e_1, \lambda(x{:}\tau)[pc].\, e_2) = \mathcal{L}(e_1, e_2, \mathcal{C}, \ell_1, \ell_2) \\
\quad \mid (\Lambda X[pc].\, e_1, \Lambda X[pc].\, e_2) = \mathcal{L}(e_1, e_2, \mathcal{C}, \ell_1, \ell_2) \\
\quad \mid (\texttt{proj}_i\ e_1, \texttt{proj}_i\ e_2) = \mathcal{L}(e_1, e_2, \mathcal{C}, \ell_1, \ell_2) \\
\quad \mid (\texttt{bind}\ x_1 = e_1\ \texttt{in}\ e_1', \texttt{bind}\ x_2 = e_2\ \texttt{in}\ e_2') = \\
\qquad \mathcal{L}(e_1, e_2, \mathcal{L}(e_1', e_2', \mathcal{C}, \ell_1, \ell_2), \mathcal{C}, \ell_1, \ell_2) \\
\quad \mid (\texttt{case}^\tau\ e_1\ \texttt{of}\ \texttt{inj}_1^\tau(z).e_2 \mid \texttt{inj}_2^\tau(z).e_3, \\
\qquad \texttt{case}^\tau\ e_1'\ \texttt{of}\ \texttt{inj}_1^\tau(z).e_2' \mid \texttt{inj}_2^\tau(z).e_3') = \\
\qquad\quad \mathcal{L}(e_1, e_1', \mathcal{L}(e_2, e_2', \mathcal{L}(e_3, e_3', \mathcal{C}, \ell_1, \ell_2), \ell_1, \ell_2), \ell_1, \ell_2) \\
\quad \mid (f_1, f_2) = \\
\qquad \texttt{if}\ f_1 = f_2\ \texttt{then}\ \mathcal{C} \\
\qquad \texttt{else if}\ \mathcal{C} \models \ell_1 \in \mathcal{F}\ \texttt{then}\ \mathcal{C} \\
\qquad \texttt{else if}\ \mathcal{C} \models \ell_2 \in \mathcal{F}\ \texttt{then}\ \mathcal{C} \\
\qquad \texttt{else}\ \texttt{NORM}(\ell_1, \mathcal{C})\ \ \texttt{OR}\ \ \texttt{NORM}(\ell_2, \mathcal{C})
\end{array}
$$

Fig. 45. Function $\mathcal{L}$ to construct blame constraint $\mathcal{C}$.

$\mathsf{NORM}(\ell, \mathcal{C}) \Rightarrow \mathtt{match}\ \ell\ \mathtt{with}$

$\quad \mid \ell_1 \wedge \ell_2 \Rightarrow \mathsf{NORM}(\ell_1, \mathcal{C})\ \mathtt{AND}\ \mathsf{NORM}(\ell_2, \mathcal{C})$

$\quad \mid \ell_1 \vee \ell_2 \Rightarrow \mathsf{NORM}(\ell_1, \mathcal{C})\ \mathtt{OR}\ \mathsf{NORM}(\ell_2, \mathcal{C})$

$\quad \mid \ell_1 \boxplus \ell_2 \Rightarrow \mathsf{NORM}(\ell_1, \mathcal{C})\ \mathtt{OR}\ \mathsf{NORM}(\ell_2, \mathcal{C})$

$\quad \mid \ell_1 \boxminus \ell_2 \Rightarrow \mathsf{NORM}(\ell_1, \mathcal{C})\ \mathtt{AND}\ \mathsf{NORM}(\ell_2, \mathcal{C})$

$\quad \mid p \Rightarrow \mathsf{DNF}(p^{\mathsf{ia}} \in \mathcal{F}\ \mathtt{AND}\ \mathcal{C})$

$\mathsf{DNF}(\ell \in \mathcal{F}\ \mathtt{AND}\ \mathcal{C}) \Rightarrow \mathtt{match}\ \mathcal{C}\ \mathtt{with}$

$\quad \mid \mathcal{F} = \varnothing \Rightarrow \ell \in \mathcal{F}$

$\quad \mid \ell' \in \mathcal{F} \Rightarrow \ell \in \mathcal{F}\ \mathtt{AND}\ \ell' \in \mathcal{F}$

$\quad \mid \mathcal{C}_1\ \mathtt{OR}\ \mathcal{C}_2 \Rightarrow \mathsf{DNF}(\ell \in \mathcal{F}\ \mathtt{AND}\ \mathcal{C}_1)\ \mathtt{OR}\ \mathsf{DNF}(\ell \in \mathcal{F}\ \mathtt{AND}\ \mathcal{C}_2)$

$\quad \mid \mathcal{C}_1\ \mathtt{AND}\ \mathcal{C}_2 \Rightarrow \mathcal{C}_1\ \mathtt{AND}\ \mathcal{C}_2\ \mathtt{AND}\ \ell \in \mathcal{F}$

Fig. 46. Helper functions (NORM and DNF) to construct blame constraint $\mathcal{C}$: NORM and DNF.

Proof. Trivial(by inspection on evaluation rules). □

Lemma 30 (INTRO GTR). *If* $\Pi \Vdash \ell_1 \succcurlyeq \tau$ *then for any* ℓ_2 $\Pi \Vdash \ell_1 \wedge \ell_2 \succcurlyeq \tau$

Proof. Proof is straightforward from *fail* judgements $\succcurlyeq$ (Figure 14) rules and ConjL rule. □

Lemma 31 (FAIL ONESTEP NON-EMPTY BLAME SET). *If* $\langle\langle e, c\rangle \,\&\, s\rangle^{\mathcal{C}} \longrightarrow \langle\langle e', c'\rangle \,\&\, s'\rangle^{\mathcal{C}'}$, *e is a source level term, and* e' *contains* $\mathtt{fail}^{\tau}$, *then for any* $\mathcal{F}$ *that satisfies* $\mathcal{C}'$ *the following condition holds:*

$\Pi \Vdash b' \succcurlyeq \tau$, *where* $b' = \bigwedge_{\forall f \in \mathcal{F}} f$

Proof. Since transition from e to e' introduces a $\mathtt{fail}^{\tau}$ term, from lemma 29, we know that the evaluation step that was taken is either C-CompareFail or C-CombineFail. Without loss of generality we can state that $e = E[\mathtt{compare}^{\ell_1 \oplus \ell_2}\ \mathtt{says}^{\tau'}\ (\overline{\eta}_{\ell_1}\ v_1)\ \mathtt{and}\ (\overline{\eta}_{\ell_2}\ v_2)]$ or $e = E[\mathtt{combine}\ x = \langle(\overline{\eta}_{k_1.L \wedge \ell}\ v_1), (\overline{\eta}_{k_2.R \wedge \ell}\ v_2)\rangle @ \ell\ \mathtt{in}\ e']$ and $\mathcal{C}' = \mathcal{L}(v_1, v_2, \mathcal{C}, \ell_1, \ell_2)$, or $\mathcal{C}' = \mathsf{NORM}(\ell, \mathcal{C})$. In the following proof for $\mathcal{F}$ and $\mathcal{F}'$ that satisfies $\mathcal{C}$ and $\mathcal{C}'$ respectively, b and b' will mean $\bigwedge_{\forall f \in \mathcal{F}} f$ and $\bigwedge_{\forall f \in \mathcal{F}'} f$ respectively. There are three possibilities for $\mathcal{C}'$.

(1) $\boldsymbol{\mathcal{C}' = ((\ell_1 \in \mathcal{F})\ \mathtt{AND}\ \mathcal{C})\ \mathtt{OR}\ ((\ell_2 \in \mathcal{F})\ \mathtt{AND}\ \mathcal{C})}$ **:** This case happens only when C-CompareFail step is taken. It is straightforward to say that, for any $\mathcal{F}$ that satisfies $\mathcal{C}$, $\mathcal{F}' = \mathcal{F} \cup \{\ell_i\}$ satisfies $\mathcal{C}'$, for $i \in \{1, 2\}$. If b is conjunction of labels in $\mathcal{F}$ then $b \wedge \ell_i$, $i \in \{1, 2\}$ are conjunction of labels in $\mathcal{F}'$. We know, $\Pi \Vdash \ell_i \succcurlyeq (\ell_1 \oplus \ell_2)\ \mathtt{says}\ \tau'$. From lemma 30 (ITRO GTR) we can write for label b, $\Pi \Vdash (\ell_i \wedge b) \succcurlyeq (\ell_1 \oplus \ell_2)\ \mathtt{says}\ \tau'$. Beacuse conjunctions of labels in $\mathcal{F}'$ are of form $b' = (\ell_i \wedge b)$, we can say that for any $\mathcal{F}'$ that satisfies $\mathcal{C}'$ the following condition holds

$\Pi \Vdash b' \succcurlyeq (\ell_1 \oplus \ell_2)\ \mathtt{says}\ \tau'$, where $b' = \bigwedge_{\forall f \in \mathcal{F}'} f$.

(2) $\mathcal{C}' = (\ell \in \mathcal{F})$ AND $\mathcal{C}$ **:** Here either ℓ is some inner layer in τ, i.e. $\tau = (\ell_1 \oplus \ell_2)$ says $\tau' = (\ell_1 \oplus \ell_2)$ says $(...(\ell$ says $\tau'')...)$ when compare statement (i.e. C-CompareFail) got executed; or ℓ is some program counter label when combine statement (i.e. C-CombineFail) got executed. It is straightforward to say that, for any $\mathcal{F}$ that satisfies $\mathcal{C}$, $\mathcal{F}' = \{\ell\} \cup \mathcal{F}$ satisfies $\mathcal{C}'$. If b is conjunction of labels in $\mathcal{F}$ then $\ell \wedge b$ is conjunction of labels in $\mathcal{F}'$. We know $\Pi \Vdash \ell \succcurlyeq \tau$. From lemma ITRO GTR 30 we can write for any label b, $\Pi \Vdash (\ell \wedge b) \succcurlyeq \tau$. Beacuse conjunctions of labels in $\mathcal{F}'$ are of the form $b' = (\ell \wedge b)$, we say that for any $\mathcal{F}'$ that satisfies $\mathcal{C}'$ the following condition holds

$\Pi \Vdash b' \succcurlyeq \tau$, where $b' = \bigwedge_{\forall f \in \mathcal{F}'} f$.

(3) $\mathcal{C}' = \mathcal{C}$ **:** This case occurs when C-CompareFail or C-CombineFail rule does not update $\mathcal{C}$ beacuse the label ℓ that is responsible for generating fail is already included in all possible $\mathcal{F}$ in $\mathcal{C}$. That means $\mathcal{C} \models \ell \in \mathcal{F}$ and $\Pi \Vdash \ell \succcurlyeq \tau$ (since the end result is fail and ℓ is the responsible label). Thus it is straightforward from lemma 30 that for label b, where $b = b_1 \wedge ...\ell... \wedge b_k$ $\Pi \Vdash b \succcurlyeq \tau$. Thus we have, $\Pi \Vdash b \succcurlyeq \tau$, where $b = \bigwedge_{\forall f \in \mathcal{F}} f$.

□

Lemma 32 (GTRDOT OR)**.** *If* $\Pi \Vdash \ell \succcurlyeq \ell_1$ says τ *then* $\Pi \Vdash \ell \succcurlyeq (\ell_1 \vee \ell_2)$ says τ.

Proof. Proof is straightforward inspecting $\succcurlyeq$ rule (Figure 14) and using DisjR. □

Lemma 33 (FAIL RESULT ONESTEP)**.** *Given,*

(1) $\Pi; \Gamma; pc; c \vdash \langle\langle e, c\rangle \,\&\, s\rangle^{\mathcal{C}} : \tau'$
(2) $\langle\langle e, c\rangle \,\&\, s\rangle^{\mathcal{C}} \longrightarrow \langle\langle \mathtt{fail}^{\tau'}, c\rangle \,\&\, s\rangle^{\mathcal{C}'}$

then for every $\mathcal{F}$ *that satisfies* $\mathcal{C}'$ *it must be the case that* $\Pi \Vdash b' \succcurlyeq \tau'$ *, where* $b' = \bigwedge_{f \in \mathcal{F}} f$

Proof. Let us proof this by induction over typing derivation $\Pi; \Gamma; pc; c \vdash e : \tau$. In the proof b is always $\bigwedge_{f \in \mathcal{F}} f$.

- **Case E-AppFailL:** $e = (\lambda(x{:}\tau_1)[pc].\, \mathtt{fail}^{\tau_1 \xrightarrow{pc} \tau})e_1$ and $e' = \mathtt{fail}^{\tau}$. From IH we get, $\Pi \Vdash b \succcurlyeq \tau$, which is what we wanted.
- **Case E-SealedFail:** $e = \eta_\ell\ \mathtt{fail}^{\tau}$ and $e' = \mathtt{fail}^{\ell\ \mathtt{says}\ \tau}$
  $\langle\langle \eta_\ell\ \mathtt{fail}^{\tau}, c\rangle \,\&\, s\rangle^{\mathcal{C}} \longrightarrow \langle\langle \mathtt{fail}^{\ell\ \mathtt{says}\ \tau}, c\rangle \,\&\, s\rangle^{\mathcal{C}}$
  Given, $\Pi; \Gamma; pc; c \vdash \eta_\ell\ \mathtt{fail}^{\tau} : \ell$ says τ.
  Therefore from induction hypothesis we have, $\Pi \Vdash b \succcurlyeq \tau$, $\mathcal{C}$ does not get updated during the evaluation step. Thus from the rule A-Type we have, $\Pi \Vdash b \succcurlyeq \ell$ says τ,
- **Case E-InjFail:** $e = \mathtt{inj}_i^{(\tau_1+\tau_2)}\ \mathtt{fail}^{\tau_i}$ and $e' = \mathtt{fail}^{(\tau_1+\tau_2)}$
  $\mathtt{inj}_i^{(\tau_1+\tau_2)}\ \mathtt{fail}^{\tau_i} \longrightarrow \mathtt{fail}^{(\tau_1+\tau_2)}$
  Given, $\Pi; \Gamma; pc; c \vdash \mathtt{inj}_i^{(\tau_1+\tau_2)}\ \mathtt{fail}^{\tau_i} : (\tau_1 + \tau_2)$
  From IH we can say for every $\mathcal{F}$ that satisfies $\mathcal{C}$, $\Pi \Vdash b \succcurlyeq \tau_i$
  $\mathcal{C}$ does not get updated because of the evaluation step. Using rule A-Sum, we can write the following, $\Pi \Vdash b \succcurlyeq (\tau_1 + \tau_2)$

- **Case E-CaseFail:** From Case typing rule we know $\Pi \Vdash {\tau_i}^{\mathsf{a}} \succcurlyeq \tau^{\mathsf{a}}$ for $i \in \{1,2\}$. IH gives us $\Pi \Vdash b \succ \tau_i$. Thus we can write $\Pi \Vdash b \succ \tau$.
- **Case E-ProjFail:** Same as case E-AppFail as the type annotation of `fail` does not change.
- **Case E-SelectFail:**
  $e = \mathtt{select}^{(\ell_3 \oplus \ell_4)\ \mathsf{says}\ \tau}\ \mathtt{fail}^{\ell_3\ \mathsf{says}\ \tau}\ \mathtt{or}\ \mathtt{fail}^{\ell_4\ \mathsf{says}\ \tau}$
  $e' = \mathtt{fail}^{(\ell_3 \ominus \ell_4)\ \mathsf{says}\ \tau}$
  $\langle\langle e, c\rangle \,\&\, s\rangle^{\mathcal{C}} \longrightarrow \langle\langle \mathtt{fail}^{(\ell_3 \ominus \ell_4)\ \mathsf{says}\ \tau}, c\rangle \,\&\, s\rangle^{\mathcal{C}}$
  Given,

$$\Pi; \Gamma; pc; c \vdash \mathtt{select}^{(\ell_3 \ominus \ell_4)\ \mathsf{says}\ \tau}\ f_1\ \mathtt{or}\ f_2 : (\ell_3 \ominus \ell_4)\ \mathsf{says}\ \tau \tag{244}$$

  From IH,

$$\Pi \Vdash b \succ \ell_3\ \mathsf{says}\ \tau \tag{245}$$
$$\Pi \Vdash b \succ \ell_4\ \mathsf{says}\ \tau \tag{246}$$

  Applying Lemma 27 with 245 and 246

$$\Pi \Vdash b \succ (\ell_3 \ominus \ell_4)\ \mathsf{says}\ \tau \tag{247}$$
$$\tag{248}$$

- **Case E-CompareFailL:**
  $e = \mathtt{compare}^{\ell_3 \oplus \ell_4\ \mathsf{says}\ \tau}\ (\mathtt{fail}^{\ell_3\ \mathsf{says}\ \tau})\ \mathsf{and}\ (\overline{\eta}_{\ell_4}\ v_2)$
  $\langle\langle e, c\rangle \,\&\, s\rangle^{\mathcal{C}} \longrightarrow \langle\langle \mathtt{fail}^{\ell_3 \oplus \ell_4\ \mathsf{says}\ \tau}, c\rangle \,\&\, s\rangle^{\mathcal{C}}$
  From IH we get,

$$\Pi \Vdash b \succ \ell_3\ \mathsf{says}\ \tau \tag{249}$$
$$\Pi \Vdash b \succ \ell_4\ \mathsf{says}\ \tau \tag{250}$$

  using A-Type, or A-Avail or A-IntegCom we can write the following

$$\Pi \Vdash b \succ (\ell_3 \oplus \ell_4)\ \mathsf{says}\ \tau \tag{251}$$

- **Case E-CompareFailR:** same as E-CompareFailL
- **Case C-CompareFail:** Trivially true based on definition of $\mathcal{L}$. and lemma 31.
- **Case E-CombineFail:**
  $e = \mathtt{combine}\ x = \langle v_1, v_2\rangle @ pc\ \mathtt{in}\ e'$. Let us assume $\Pi; \Gamma; pc; c \vdash e' : \ell\ \mathsf{says}\ \tau$, which means the whole term has type $\ell\ \mathsf{says}\ \tau$. From IH $\Pi \Vdash b \succ \ell\ \mathsf{says}\ \tau$, which trivially proves this case.

- **Case E-TAppFail:**
  $e = \mathtt{fail}^{\forall X[pc].\,\tau}$ and $\langle\langle \mathtt{fail}^{\forall X[pc].\,\tau_1}\ \tau_2, c\rangle \,\&\, s\rangle^{\mathcal{C}} \longrightarrow \langle\langle \mathtt{fail}^{\tau_1[X \mapsto \tau_2]}, c\rangle \,\&\, s\rangle^{\mathcal{C}}$ Given, $\Pi \Vdash b \succ \forall X[pc].\,\tau$, which is same as saying $\Pi \Vdash b \succ \tau[X \mapsto \tau']$ where $b = \bigwedge_{f \in \mathcal{F}} f$

- **Case E-RETFAIL:** IH gives us $\Pi \Vdash b \succcurlyeq \tau$, Then from A-TYPE $\Pi \Vdash b \succcurlyeq pc^{\text{ia}}$ says τ.
- **Other cases:** neither fail propagates nor any change in $\mathcal{C}$.

□

Theorem 1 (Sound blame)**.** *Given,*

(1) $\Pi; \Gamma; pc; c \vdash \langle\langle e, c\rangle \,\&\, \texttt{empty}\rangle^{\mathcal{C}_{init}} : \tau$
(2) $\langle\langle e, c\rangle \,\&\, \texttt{empty}\rangle^{\mathcal{C}_{init}} \longrightarrow^* \langle\langle \texttt{fail}^{\tau}, c\rangle \,\&\, \texttt{empty}\rangle^{\mathcal{C}'}$

where e is a source-level expression,[11]
then for each possible faulty set $\mathcal{F}_i$ implied by $\mathcal{C}'$, there is a principal $b_i = \bigwedge_{p \in \mathcal{F}_i} p$ such that $\Pi \Vdash b_i^{\text{ia}} \succcurlyeq \tau$.

Proof. e does not have any `fail` terms in it and it steps to a `fail` term. From lemma 29 we know that there has to be at least one C-COMPAREFAIL or C-COMBINEFAIL step taken during the evaluation. Either e takes single step or multiple steps to produce the $\texttt{fail}^{\tau}$ term as the end result.

(1) $\texttt{fail}^{\tau}$ **is produced via single step:** If e takes a single step ,

- From lemma 29 we know e is either of the form $\texttt{compare}^{(\ell_3 \oplus \ell_4)}\ v_1$ and v_2, or $\texttt{combine}\ x = v@pc\ \texttt{in}\ e_2$ and from lemma 31 we know that $\Pi \Vdash b_i \succcurlyeq \tau$

- Another possibility is that, the last evaluation step produces the $\texttt{fail}^{\tau}$ result.
  $\langle\langle e, c\rangle \,\&\, \texttt{empty}\rangle^{\varnothing} \longrightarrow^* \langle\langle e_{n-1}, c_{n-1}\rangle \,\&\, s_{n-1}\rangle^{\mathcal{C}_{n-1}} \longrightarrow \langle\langle \texttt{fail}^{\tau}, c\rangle \,\&\, \texttt{empty}\rangle^{\mathcal{C}}$ From lemma 28 we know e_{n-1} is either $\texttt{compare}^{\tau}\ v_1$ and v_2 or $\texttt{combine}\ x = v@pc\ \texttt{in}\ e_2$ and then from lemma 31 we know $\Pi \Vdash b_i \succcurlyeq \tau$

(2) $\texttt{fail}^{\tau}$ **is produced by propagation of a `fail` term:** This means the evaluation takes more than one step. Let us prove it by induction over structure of e.
$\langle\langle e, c\rangle \,\&\, \texttt{empty}\rangle^{\varnothing} \longrightarrow^* \langle\langle e_i, c_i\rangle \,\&\, s_i\rangle^{\mathcal{C}_i} \longrightarrow^* \langle\langle \texttt{fail}^{\tau}, c_n\rangle \,\&\, s_n\rangle^{\mathcal{C}_n}$
Without loss of generality we can say that the `fail` term that propagates till the end is introduced first in expression e_i. That means there exists an expression e_{i-1} such that it takes C-COMPAREFAIL or C-COMBINEFAIL evaluation rule to step to e_i.
$\langle\langle e, c\rangle \,\&\, \texttt{empty}\rangle^{\varnothing} \longrightarrow^* \langle\langle e_{i-1}, c_{i-1}\rangle \,\&\, s_{i-1}\rangle^{\mathcal{C}_{i-1}} \longrightarrow \langle\langle e_i, c_i\rangle \,\&\, s_i\rangle^{\mathcal{C}_i}$
$\longrightarrow^* \langle\langle \texttt{fail}^{\tau}, c\rangle \,\&\, \texttt{empty}\rangle^{\mathcal{C}}$
(e_{i-1} can be e itself). From lemma 31, we know that because the step that is taken is C-COMPAREFAIL or C-COMBINEFAIL we have
for faulty set $\mathcal{F}$ satisfying $\mathcal{C}_i$ $\Pi \Vdash b_i \succcurlyeq \tau_i$ where $b_i = \bigwedge_{f \in \mathcal{F}} f$
From e_i onwards any step taken is either going to propagate `fail` or is going to not touch the `fail` term at all. For every evaluation step the invariant $\Pi \Vdash b_i \succcurlyeq \tau_i$ holds following lemma 33.

□

[11] In other words, e does not contain any `fail` terms.