Information-flow control for programming on encrypted data*

J.C. Mitchell, R. Sharma, D. Stefan and J. Zimmerman

April 13, 2012

Abstract

Using homomorphic encryption and secure multiparty computation, cloud servers may perform regularly structured computation on encrypted data, without access to decryption keys. However, prior approaches for programming on encrypted data involve restrictive models such as boolean circuits, or standard languages that do not guarantee secure execution of all expressible programs. We present an expressive core language for secure cloud computing, with primitive types, conditionals, standard functional features, mutable state, and a secrecy preserving form of general recursion. This language, which uses an augmented information-flow type system to prevent control-flow leakage, allows programs to be developed and tested using conventional means, then exported to a variety of secure cloud execution platforms, dramatically reducing the amount of specialized knowledge needed to write secure code. We present a Haskell-based implementation and prove that cloud implementations based on secret sharing, homomorphic encryption, or other alternatives satisfying our general definition meet precise security requirements.

1 Introduction

Homomorphic encryption [1, 2, 3] and secure multiparty computation [4, 5, 6, 7] open new opportunities for secure cloud computing on encrypted data. For example, cloud servers could examine encrypted email for spam, without decrypting the email. A cloud server could also potentially compute a route between two endpoints on a public map, and return the encrypted path to a client. This paradigm provides cryptographic confidentiality, because the cloud server never has the keys needed to decrypt or recover private data.

Our goal is to provide a language and programming environment that would allow developers to produce secure cloud applications, without sophisticated knowledge of the cryptographic constructions used. Theoretical approaches for programming on encrypted data involve restrictive models such as boolean circuits, which are not conventional programming models. Programming using a conventional language and a cryptographic library, on the other hand, may allow programmers to write programs that cannot execute, because control flow depends on encrypted values that are not available to the cloud execution platform. Due to the performance costs of computing on encrypted data, realistic computation must involve mixtures of secret (encrypted) and public data. Without information flow restrictions, programs could inadvertently specify leakage of secret data into public variables.

We present an expressive core language for secure cloud computing, with primitive types, conditionals, standard functional features, mutable state, and a secrecy preserving form of general recursion. This language uses an augmented information-flow type system to impose conventional information-flow control and addresses a new problem for information-flow type systems: prevent control-flow leakage to the cloud platform. The language allows programs to be developed and tested using conventional means, then exported to a variety of secure cloud execution platforms, dramatically reducing the amount of specialized knowledge needed

*This document is the extended version of a article scheduled to appear in the proceedings of the 25th IEEE Computer Security Foundations Symposium (CSF 2012).

to write secure code. Past efforts have produced generally less expressive programming languages (SMC [8], Fairplay [9], SIMAP [10, 11], VIFF [12]) with weaker security proofs (see Section 7).

A core problem is that if an untrusted cloud server is given a program to execute, the server can observe control flow through the program. Therefore, if any conditional branch depends on a secret (encrypted) value, the server must execute both paths and combine results using operations on encrypted data. For example, consider `if` x `then` $y := 4$ `else` $y := 5$, where $x : \mathrm{bool}$ and $y : \mathrm{int}$. If x is secret and y is public, then this statement cannot be executed because secret information flows to a public reference; we use conventional information-flow control to prevent this. If both x and y are secret, then this is executed by storing the *secretly computed* value $x \cdot 4 + (1 - x) \cdot 5$ in y. While computing both branches works for a simple if-then-else, this cannot be done for recursive functions, because the set of possible execution paths may be infinite. Therefore, we augment our type system with additional information labels to prevent unbounded recursion on secret values, without restricting computation when control flow relies on public data.

While homomorphic encryption and secure multiparty computation are based on different cryptographic insights and constructions, there is a surprising structural similarity between them. This similarity is also shared by so-called *partially homomorphic* encryption, in which the homomorphism property holds only for certain operations. We capture this similarity in our definition of *secure execution platform*. Figure 1 shows how our separation of programming environment from cryptographically secure execution platforms can be used to delay deployment decisions or run the same code on different platforms. One advantage of this approach is that a developer may write conventional code and debug it using standard tools, without committing to a specific form of execution platform security. Another advantage is that as cryptographic constructions for various forms of homomorphic encryption improve, the same code can be retargeted because our language makes correctness independent of choice of secure execution platform.

Our formal definition of secure execution platform allows us to develop a single set of definitions, theorems, and proofs that are applicable to many cryptographic systems. The two main theoretical results are theorems that guarantee the correctness and security of program execution. We state correctness using a reference semantics that expresses the standard meaning of programs, with encryption and decryption as the identity function. The correctness theorem states the cloud execution of a program on encrypted data produces the same output as the conventional execution without encryption. Our security theorem depends on the threat model, because homomorphic encryption and secret sharing are secure in different ways. The security theorem states that no adversary learns the initial secret client values, beyond what is revealed by the program output, because the probability distributions of program behaviors (on different secret inputs) are indistinguishable. We also show that fully homomorphic encryption and a specific secret-sharing scheme meet the definition of secure execution platform, as do somewhat homomorphic schemes when they support the operations actually used in the program.

We develop our results using the honest-but-curious adversary model, commonly used in practical applications (e.g., [13], [10]). However, there are established methods for assuring integrity, using commitments and zero-knowledge techniques [14], as well as for reducing communication overhead [7]. In addition, while we focus on data confidentiality, our current system can also protect confidential algorithms, in principle, by considering code as input data to an interpreter (or "universal Turing machine").

2 Implementation and motivating example

The core of our domain-specific language (DSL) is implemented as a Haskell library, an embedded domain-specific language (EDSL). Our implementation includes Shamir secret sharing and fully homomorphic encryption; both use SSL network communication between clients and any number of servers. A preliminary design without recursion, references, or conditionals, and with a different implementation was described in [15].

As highlighted in Figure 1, we provide a Template Haskell compiler, which translates a subset of Haskell syntax to our EDSL, at compile-time. The Template Haskell extension provides syntactic sugar for EDSL

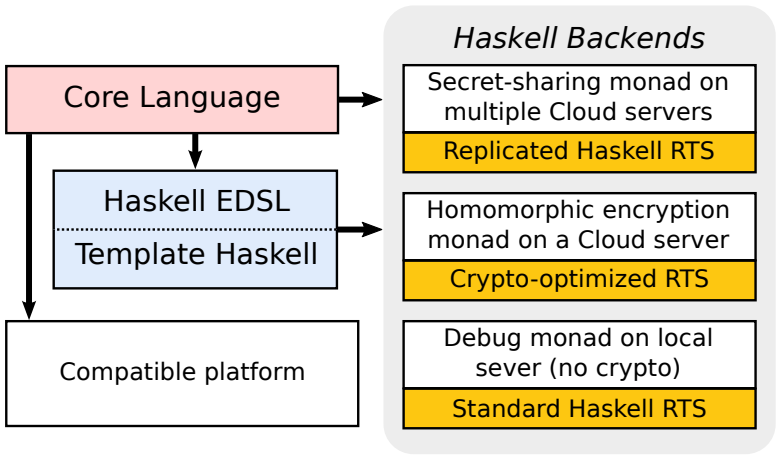

Figure 1: Multiple deployment options using different runtime systems. The EDSL (with the Template Haskell compiler) can be executed on various Haskell-backends. The core language compiler can additionally be compiled to other systems such as SMC and VIFF.

combinators, enabling the programmer to use already-familiar syntax when writing code that operates on secret data. We also provide a core-language compiler front-end that directly implements the information-flow type system of this paper; the front-end exports abstract syntax trees (ASTs) of well-typed, labeled expressions, facilitating code generation for various back-ends, including our Haskell EDSL.

As a working example, we consider the case for email filtering in a cloud setting. More specifically, we consider RE: Reliable Email [16], as shown in Figure 2. With Reliable Email, email from trustworthy senders (as deemed by the receiver) is forwarded directly to the receiver's inbox, bypassing the spam filter. This guarantees that a message from a trustworthy source, which may have been labeled "spam" because of its content, will be received—hence, making email a *reliable* communication medium between trustworthy entities.

A key component of (a simplified) Reliable Email system is a whitelist, containing the email addresses of all senders considered trustworthy by the receiver. For every incoming message, the authentication of the `From` address is verified[1] and checked against the whitelist. If the address is in the whitelist, the mail is forwarded directly to the inbox, otherwise it is forwarded to the spam filter.

It is important that the sender's whitelist, which is essentially an address book, remain confidential if Reliable Email and the spam filter are executed in the cloud. Hence, in our setting, the hashed email addresses are encrypted (or split into shares, in the case of secret sharing) and the check against the whitelist is done homomorphically. Figure 3 shows a component of the whitelist check in our Haskell DSL: computing the Hamming distance between the hashes of two email addresses. A Hamming distance of zero denotes a match, i.e., the email address is that of a trustworthy entity.

This example highlights several key aspects of our Haskell EDSL. First, our language provides various primitives such as `fix` and `toSecret`, that are respectively used for implementing (safe) recursion, and lifting public values to their secret equivalents. Second, the DSL embedding allows the programmer to use existing Haskell features including higher-order functions, abstract data types, lists, etc. Third, the Template Haskell compiler (`compileTHtoEDSL`) allows the programmer to use standard Haskell syntax. Finally, compared to languages with similar goals (e.g., SMCL [11]), where a programmer is required to write separate client and server code, using our EDSL, a programmer needs to only write a single program; we eliminate the client/server code separation by providing a simple runtime system that directs all parties.

Developers who write in the EDSL can take advantage of existing, carefully-engineered Haskell develop-

[1]Messages must be signed by the sender since forging the `From` address is trivial.

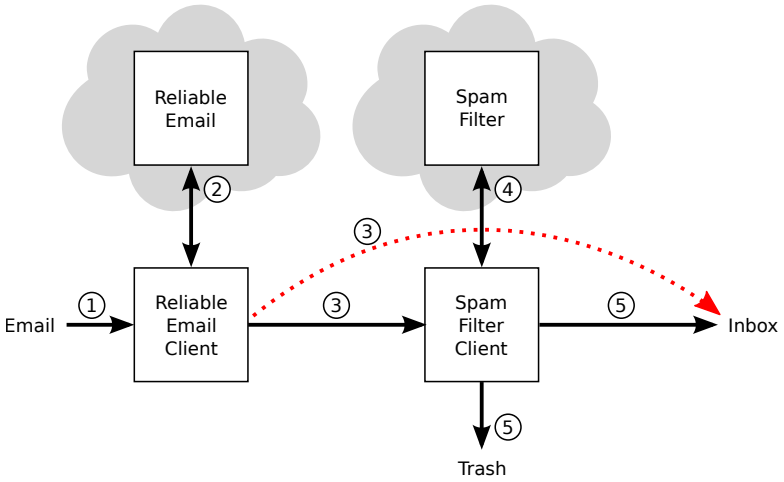

Figure 2: RE: Reliable Email in a cloud setting. Incoming email is forwarded by clients to cloud-based filters (steps 2 and 4). If email is from a trustworthy source as determined by Reliable Email (step 4), it is forwarded directly to the inbox (step 3), otherwise it takes the default path through the spam filter (steps 3-5).

```
-- | Given two secret, hashed email addresses,
-- compute the Hamming distance between them.
hammingDist :: SHA1EmailAddr
            → SHA1EmailAddr → SecIO SecretInt
hammingDist e1 e2 = hammingDist' e1 e2 0

-- | Number of bits in SHA1 digest
sha1DigestLen = 160

-- | Actual distance computation.
-- A SHA1EmailAddr is a list of bits.
hammingDist' e1 e2 = $(compileTHtoEDSL [|
  fix ( λf λi →
          -- Iterate over all bits of an email
      let k = if i < sha1DigestLen -- done?
                then f (i+1)       -- no, next
                else toSecret 0    -- yes
          -- Compute difference between i'th bits
          x = xor (e1 !! i) (e2 !! i)
      in x + k -- Sum of all bit differences
      )
  |])
```

Figure 3: Recursively computing the Hamming distance of two SHA1-hashed email addresses. Recursion with `fix` is used to iterate over all the bits of the hashed email addresses, which are XORed.

ment tools, compilers, and runtime systems. Programmers also have the benefit of sophisticated type-checking and general programming features of Haskell; we use the Haskell type and module system to enforce the correct usage of our secure execution platform. Because our EDSL and compilers are packaged in the form of Haskell libraries, other researchers could use our libraries to implement different programming paradigms over the same forms of cryptographic primitives, or compile our core language to other runtime systems.

3 Background

3.1 Haskell and EDSLs

There is a broad research community with extensive experience developing domain-specific languages (DSLs), represented by the USENIX conferences on Domain-Specific Languages and documented in an annotated bibliography [17]. Many DSLs are *embedded* into an existing language, which leverages the substantial development effort of any practical language and also has the benefit of allowing DSLs embedded in the same host language to interact and share infrastructure. Haskell is a purely functional language that is widely used for such DSL embeddings [18]. Haskell's type system, lazy evaluation strategy (expressions are evaluated only when their values are needed), and support for monads makes it easy to define new data structures, syntactic extensions, and control structures—features commonly desired when embedding languages.

The main Haskell constructs used in embedding our DSL are monads and type classes. A monad M provides a type constructor and related operations that obey several laws. Specifically, if M is a monad and α is an arbitrary type, then $M\alpha$ is a type with operations `return`, and $\gg\!=$ (pronounced "bind"), whose types are shown in Figure 4. In Haskell, all computations that produce side-effects are distinct, at the type-level, from pure functions. For instance, `getChar`, which reads a character from standard input, has type `IO Char`, indicating that it is a (input/output, `IO`) monadic computation. This type-level distinction allows for local-reasoning of an implementation's security properties (and information leaks).

```
class Monad M where
  return :: α → Mα
  (>>=)  :: Mα → (α → Mβ) → Mβ
```

Figure 4: Monad operations

The bind operator, which is used to combine multiple actions (which are similar to commands of imperative language), is typically invoked through syntactic sugar. Specifically, the `do` keyword introduces a series of actions in which each semicolon or newline is an invocation of bind. For example, a simple echo program can be written as: `do { c ←getChar ; putChar c }`. Here, the result of executing action `getChar` is bound to symbol `c` (hence the name "bind") which is then used to compute and execute the action `putChar c`. Observe that this code quite similar to mainstream imperative languages, e.g., `c = getchar(); putchar(c);` in C.

As further highlighted in Figure 4, Haskell provides support for monads through the `Monad` *type class*. Type classes provide a method of associating a collection of operations with a type or type constructor. Programmers, then, declare *instances* of a given type class by naming the type or type constructor and providing implementations of all required operations. As a consequence, programmers can declare a type, such as `SecureIO`, which may internally perform encryptions and network communication, to be an instance of `Monad` and directly leverage Haskell's `do`-notation. As explained later, type classes are also useful in 'overloading' arithmetic operations over secret and public data.

Haskell also supports compile-time meta-programming with Template Haskell [19]. Template Haskell provides a method for reifying Haskell source: converting concrete Haskell syntax to a Haskell data type representing the source abstract syntax tree (AST); and, dually, a method for splicing an AST into Haskell source. Figure 3 shows an example use case of compile-time meta-programming with Template Haskell. The recursive function is reified by enclosing it in '`[|`' brackets: `[| fix ( λf λi →... ) |]`. We use the function `compileTHtoEDSL` to analyze the corresponding AST and generate a new AST composed of EDSL primitives. Finally, the generated AST is spliced in as Haskell source by enclosing it with `$(...)`, providing a definition for `hammingDist'`. Our use of Template Haskell is limited to adding syntactic sugar to our EDSL, so that programmers can work with familiar constructs. An alternative extension, *QuasiQuotes*, can be used to reify arbitrary

syntax; in conjunction with Template Haskell, we can use QuasiQuotes to compile from the core language to our EDSL.

3.2 Homomorphic encryption

A *homomorphic encryption scheme* $\langle \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval} \rangle$ consists of a key generation algorithm, encryption and decryption algorithms, and an evaluation function that evaluates a function $f \in \mathcal{F}$ on encrypted data. More specifically, for a public/secret key pair $\langle \mathrm{pk}, \mathrm{sk} \rangle$ generated by KeyGen, and a plaintext m, if $c = \mathsf{Enc}(\mathrm{pk}, m)$, then $\mathsf{Dec}(\mathrm{sk}, \mathsf{Eval}(\mathrm{pk}, c, f)) = f(m)$ for every $f \in \mathcal{F}$, where $\mathcal{F}$ is some set of functions on plaintexts. We say that the encryption scheme is *homomorphic with respect to the set $\mathcal{F}$ of functions.*

While some homomorphic encryption schemes [20, 21, 22] are *partially homomorphic* – i.e., homomorphic with respect to a restricted class of functions, such as the set of quadratic multivariate polynomials or the set of shallow branching programs – recent research has produced constructions that are *fully homomorphic*, i.e., homomorphic with respect to all functions of polynomial complexity [1, 2, 23, 3]. Since this work has generated substantial interest, there is a rapidly growing set of fully homomorphic constructions. However, for efficiency reasons we remain interested in partially homomorphic schemes as well.

3.3 Secure multiparty computation

Another approach to computing on ciphertexts makes use of generic two-party or multi-party secure computation [24, 25, 4, 26, 6, 7, 27, 28, 29], in which the client, who has the plaintext m, communicates with the server(s), who have the function f to be computed on m. The standard conditions for secure multiparty computation guarantee that the client learns $f(m)$, while the server (or an adversary compromising some restricted set of servers) learns nothing about m.

In this work, we specifically consider Shamir secret sharing, and the multiparty computation protocol based on it [5]. According to this protocol, a client C shares a secret value a_0 from a prime-order finite field $\mathbb{F}_p$ among N servers. In an (N, k) secret sharing scheme, N servers can jointly perform computations on m and other shared secrets, such that at least k of the N servers must collude to learn anything about m. Letting $a_0 = m$, in Shamir secret sharing, the client C shares a_0 by choosing values $a_1, \ldots, a_{k-1}$ uniformly at random from F, and forms the polynomial $p(x) = \sum_{i=0}^{k-1} a_i x^i$. Then, C computes and distributes the *shares* $s_1 = p(1), \ldots, s_N = p(N)$ to servers $S_1, \ldots, S_N$, respectively.

Addition is easy for the servers to compute, since they can simply add their shares of two values pointwise: if the values s_i form a sharing of a_0 via p, and t_i form a sharing of b_0 via q, then $s_i + t_i$ form a sharing of $a_0 + b_0$ via $p + q$. Similarly, if the values s_i form a sharing of a_0 via p, then, for a constant c, $c \cdot s_i$ form a sharing of $c \cdot a_0$ via $c \cdot p$. Multiplication of two secret values is more complicated, because multiplication of polynomials increases their degree. The solution involves computing and communicating a new sharing among the servers.

4 Definitions and assumptions

Before presenting our language design, we detail the semantic structure used in our analysis. As shown below, our semantic structure is sufficient to prove correctness and security theorems for the language we develop, and general enough to encompass secret sharing, homomorphic encryption, and other platforms.

4.0.1 Primitive operations

We assume a family $\mathcal{Y}$ of sets, where each $Y \in \mathcal{Y}$ represents a set of primitive values. While different platforms may provide different primitive values, we assume that at least $\{\mathrm{bool}, \mathrm{unit}\} \subseteq \mathcal{Y}$, where $\mathrm{bool} = \{\texttt{true}, \texttt{false}\}$ and $\mathrm{unit} = \{()\}$. Although we do not require it, platforms often will also provide a set

$\text{int} \in \mathcal{Y}$, representing bounded integers (e.g., 64-bit integer values, or values over a finite field $\mathbb{F}_p$ in some implementations of secret sharing).

We also assume primitive operations $\text{op}_1, \ldots, \text{op}_r$ on these values, where each operation has its own type: i.e., $\text{op}_i : \text{dom}(\text{op}_i) \to \text{cod}(\text{op}_i)$. (For instance, if op_1 is addition modulo p, then $\text{dom}(\text{op}_1) = (\text{int}, \text{int})$, and $\text{cod}(\text{op}_1) = \text{int}$.) To provide additional flexibility for richer language features, we also assume that for each primitive type Y, we have a primitive branching operator: i.e., there is some $\text{op}_{\text{Br}(Y)}$ such that $\text{op}_{\text{Br}(Y)}(\texttt{true}, y_1, y_2) = y_1$ and $\text{op}_{\text{Br}(Y)}(\texttt{false}, y_1, y_2) = y_2$. (In platforms with addition and multiplication, for instance, we might have $\text{op}_{\text{Br(int)}}(b, z_1, z_2) = b \cdot z_1 + (1 - b) \cdot z_2$.) In reality, some platforms might only support branching operators for a subset of primitive types, or none at all (e.g., cryptosystems that are only additively or multiplicatively, rather than fully, homomorphic). However, to simplify the formalism, we assume branching operators for all primitive types; the development proceeds in a similar fashion for simpler platforms, and only requires that additional restrictions be imposed on branching constructs (`if-then-else`).

4.1 Distributed computing infrastructure

We assume N servers, $S_1, \ldots, S_N$, execute the secure computation on behalf of one client, C.[2] (In many natural cases, such as homomorphic encryption, $N = 1$). The $(N + 1)$ parties will communicate by sending messages via secure party-to-party channels; we denote by M the set of possible message values that may be sent. A *communication round* is a set $\{(P_1^{(i)}, P_2^{(i)}, m^{(i)})\}_{1 \le i \le r}$ of triples, each indicating a sending party, a receiving party, and a message $m \in M$. A *communication trace* is a sequence of communication rounds, possibly empty, and $\mathcal{T}$ is the set of communication traces.

If $A \subseteq \{S_1, \ldots, S_N\}$ is any subset of the servers, the *projection of trace T onto A*, written $\Pi_A(T)$, is the portion of the trace visible to the servers in A, i.e., $\Pi_A(\varepsilon) = \varepsilon$ and:

$$\Pi_A(\{(S_1^{(i)}, S_2^{(i)}, m^{(i)})\} \| T) =$$
$$\{(S_1^{(i)}, S_2^{(i)}, m^{(i)}) \mid \{S_1^{(i)}, S_2^{(i)}\} \cap A \neq \emptyset\} \| \Pi_A(T)$$

General form of cryptographic primitives We work with a two-element security lattice, $\text{P} \sqsubseteq \text{S}$, representing (respectively) "public" values, which are transmitted in the clear and may be revealed to any party; and "secret" values, which are encrypted or otherwise hidden, and must remain completely unknown to the adversary. For each primitive type $Y \in \mathcal{Y}$, we assume a set $\mathcal{E}_{\text{S}}(Y)$, holding "secret equivalents" of base values in Y; for notational uniformity, we also define $\mathcal{E}_{\text{P}}(Y) = Y$, signifying that the "public equivalent" of a value is just the value itself. Similarly, we assume, for any $y \in Y$, a set $\mathcal{E}_{\text{S}}(y) \subset \mathcal{E}_{\text{S}}(Y)$, holding the "secret equivalents" of y (with $\mathcal{E}_{\text{P}}(y) = \{y\}$); we assume that the sets $\{\mathcal{E}_\alpha(y) : y \in Y\}$ form a partition of $\mathcal{E}_\alpha(Y)$. We recall that for any two elements (or *labels*) of a lattice, we have a well-defined *join* ($\sqcup$), which corresponds to the *least upper bound* of the two elements (e.g., $\text{P} \sqcup \text{S} = \text{S}$).

We also assume a few standard cryptographic primitives, expressed as *protocol operations* that may operate on initial parameters $\iota \in \mathcal{I}$, generate communication traces among the parties, and/or consume bits from a source of randomness. For clarity, we leave this randomness source implicit, instead considering each operation to produce a distribution over the values in its range (and implicitly lifting the operations to act on distributions over their domains). We regard predicates over these distributions to be true if they hold with probability 1.

The operations we assume are as follows (overloaded for all primitive types Y):

- $\text{Enc}_{\text{S}} : Y \times \mathcal{I} \to \mathcal{E}_{\text{S}}(Y) \times \mathcal{T}$, "hiding" $y \in Y$.
- $\text{Dec}_{\text{S}} : \mathcal{E}_{\text{S}}(Y) \times \mathcal{I} \to Y \times \mathcal{T}$, "unhiding" $\tilde{y} \in \mathcal{E}_{\text{S}}(Y)$.

[2] The restriction to a single client is for clarity of presentation. The generalization to multiple clients is straightforward, and, indeed, it is common in practice to have N parties in total executing a multiparty computation, each one serving as both a client and a server.

- $\mathrm{Enc}_{\ell_1,\ldots,\ell_r}(\mathrm{op}_i) : \prod_j \mathcal{E}_{\ell_j}(\mathrm{dom}(\mathrm{op}_i)_j) \times \mathcal{I} \to \mathcal{E}_{\bigsqcup_j \ell_j}(\mathrm{cod}(\mathrm{op}_i)) \times \mathcal{T}$ (when at least one ℓ_j is S), evaluating a primitive operation.

We also assume that Init describes the generation of initial parameters according to some distribution $\mathcal{I}$ (for example, public and secret keys in the case of homomorphic encryption). For notational uniformity, as above, we also define the corresponding operations in the degenerate case of "hiding" public values (operating as the identity on the plaintext values, and yielding empty traces):

- $\mathrm{Enc}_{\mathrm{P},\ldots,\mathrm{P}}(\mathrm{op}_i)(y_1, \ldots, y_r, \iota) = (\mathrm{op}_i(y_1, \ldots, y_r), \varepsilon)$
- $\mathrm{Enc}_{\mathrm{P}}(y, \iota) = (y, \varepsilon)$
- $\mathrm{Dec}_{\mathrm{P}}(y, \iota) = (y, \varepsilon)$

We also write Dec as shorthand for $\mathrm{Dec}_{\mathrm{P}}$ or $\mathrm{Dec}_{\mathrm{S}}$, as appropriate based on the domain (i.e., Dec acts as $\mathrm{Dec}_{\mathrm{P}}$ on Y, and acts as $\mathrm{Dec}_{\mathrm{S}}$ on $\mathcal{E}_{\mathrm{S}}(Y)$). In addition, we assume a projection operator from the initial parameters onto any server or set of servers, writing:

$$\Pi_A(\iota) = (\Pi_{\{S_{a_1}\}}(\iota), \ldots, \Pi_{\{S_{a_k}\}}(\iota))$$

(where $A = \{S_{a_1}, \ldots, S_{a_k}\}$) to mean, intuitively, the portion of the initial parameters $\iota \in \mathcal{I}$ that servers in A should receive.

In addition, we assume that equality is efficiently decidable on any universe Y of primitive values; that the label ℓ and universe Y of a value in $\mathcal{E}_\ell(Y)$ are efficiently computable from the value itself (e.g., by tagging, when the underlying sets are the same); and that there is some canonical ordering on the universes.

Cryptographic correctness assumptions We assume the usual encryption and homomorphism conditions, augmented for cryptographic primitives that depend on randomness and that may communicate among servers to produce their result. For every element y of a primitive type Y, and every choice of initial parameters $\iota \in \mathcal{I}$, we assume a family of *safe distributions* $\hat{\mathcal{E}}^\iota_\ell(y)$ over $\mathcal{E}_\ell(y)$: intuitively, any distribution $l \in \hat{\mathcal{E}}^\iota_\ell(y)$ can safely serve as the "hiding" of y under the initial parameters ι (at secrecy level $\ell \in \{\mathrm{P}, \mathrm{S}\}$). We require that "hiding" a base value must yield a safe distribution:

- $\pi_1(\mathrm{Enc}_\ell(y, \iota)) \in \hat{\mathcal{E}}^\iota_\ell(y)$

We also require that unhiding ("decryption") is the left-inverse of hiding ("encryption"), and hiding commutes homomorphically with the primitive operations:

- $\pi_1(\mathrm{Dec}_\ell(\pi_1(\mathrm{Enc}_\ell(y, \iota)), \iota)) = y$
- $\pi_1(\mathrm{Enc}_{\ell_1,\ldots,\ell_r}(\mathrm{op}_i)(l_1, \ldots, l_r, \iota)) \in$
  $\hat{\mathcal{E}}_{\bigsqcup_j \ell_j}(\mathrm{op}_i(y_1, \ldots, y_r), \iota)$ whenever $l_j \in \hat{\mathcal{E}}^\iota_{\ell_j}(y_j)$

Indistinguishability conditions In general, the distributed threat model may involve any set of possible combinations of colluding servers. We formalize this by assuming a family $\mathcal{A}$ of sets that we refer to as valid sets of untrusted servers. Intuitively, for any set of servers $A \in \mathcal{A}$, we assume the cryptographic primitives are intended to provide security even if an adversary has access to all information possessed by all servers in A.

Different platforms may provide different security guarantees of their primitives. For example, protocols may specify that distributions are *computationally indistinguishable* (i.e., indistinguishable to a probabilistic polynomial-time adversary), or *information-theoretically indistinguishable* (i.e., identical). For the purposes of this development, we will use the term *indistinguishable* to refer to whichever of the above notions is specified by the secure execution platform. Using this terminology, we require that any two sequences of partial traces

are indistinguishable if each pair of corresponding partial traces describes either 1.) a "hiding" operation; 2.) a primitive operation whose public arguments agree (and whose hidden arguments are safely-distributed); or 3.) an "unhiding" operation on values that turn out to be equal. More precisely, we say that the pair of communication rounds $T_j(\iota), T'_j(\iota)$ is *safe*, denoted $\textsc{Safe}(\iota, T_j(\iota), T'_j(\iota))$, if it satisfies any of the following conditions:

1. $T_j(\iota) = \pi_2(\mathrm{Enc}_{\mathrm{S}}(y_j, \iota))$, and $T'_j(\iota) = \pi_2(\mathrm{Enc}_{\mathrm{S}}(y'_j, \iota))$ (for some Y, and $y_j, y'_j \in Y$). In this case, we say that $T_j(\iota), T'_j(\iota)$ constitute a "safe hiding" (denoted $\textsc{SafeEnc}(\iota, T_j(\iota), T'_j(\iota))$).

2. $T_j(\iota) = \pi_2(\mathrm{Enc}_{\ell_1,\ldots,\ell_r}(\mathrm{op}_i)(\tilde{y}_1, \ldots, \tilde{y}_r, \iota))$ and $T'_j(\iota) = \pi_2(\mathrm{Enc}_{\ell_1,\ldots,\ell_r}(\mathrm{op}_i)(\tilde{y}'_1, \ldots, \tilde{y}'_r, \iota))$ where for each k, either:

   - $\ell_k = \mathrm{S}$, and for some Y_k, we have $\tilde{y}_k \in \hat{\mathcal{E}}^{\iota}_{\mathrm{S}}(y_k)$ and $\tilde{y}'_k \in \hat{\mathcal{E}}^{\iota}_{\mathrm{S}}(y'_k)$, with $y_k, y'_k \in Y_k$.
   - $\ell_k = \mathrm{P}$, and for some Y_k, we have $\tilde{y}_k, \tilde{y}'_k \in Y_k$, and $\tilde{y}_k = \tilde{y}'_k$.

   (and the analogous conditions for $T'_j(\iota)$). In this case, we say that $T_j(\iota), T'_j(\iota)$ constitute a "safe primitive operation" (denoted $\textsc{SafeOp}(\iota, T_j(\iota), T'_j(\iota))$).

3. $T_j(\iota) = \pi_2(\mathrm{Dec}_{\mathrm{S}}(c_j, \iota)), T'_j(\iota) = \pi_2(\mathrm{Dec}_{\mathrm{S}}(c'_j, \iota))$ and $\pi_1(\mathrm{Dec}_{\mathrm{S}}(c_j, \iota)) = \pi_1(\mathrm{Dec}_{\mathrm{S}}(c'_j, \iota))$. In this case, we say that $T_j(\iota), T'_j(\iota)$ constitute a "safe unhiding" (denoted $\textsc{SafeDec}(\iota, T_j(\iota), T'_j(\iota))$).

We extend the predicate $\textsc{Safe}$ to pairs of entire traces if each component is safe: i.e., $\textsc{Safe}(\iota, T_1, T_2)$ if $|T_1| = |T_2|$ and $\textsc{Safe}(\iota, T_1(j), T_2(j))$ for all j. Finally, we consider partial traces $T(\iota) = (T_1(\iota), \ldots, T_m(\iota))$ and $T'(\iota) = (T'_1(\iota), \ldots, T'_m(\iota))$, and the corresponding adversarial views:

- $O(\iota) = (\Pi_A(\iota), \Pi_A(T_1(\iota)), \ldots, \Pi_A(T_k(\iota)))$
- $O'(\iota) = (\Pi_A(\iota), \Pi_A(T'_1(\iota)), \ldots, \Pi_A(T'_k(\iota)))$

We require the following indistinguishability condition: if $\textsc{Safe}(\iota, T_j(\iota), T'_j(\iota))$ for every j, then the distributions $O(\mathrm{Init}())$ and $O'(\mathrm{Init}())$ are indistinguishable.

Definition 1. We say that the system $(N, \mathcal{I}, \mathrm{Init}, \mathcal{E}, \hat{\mathcal{E}}, M, \mathrm{Enc}, \mathrm{Dec}, \mathcal{A})$ is a *secure execution platform* for (op_i) if it satisfies all of the conditions specified above.

5 Language design

We present a functional core language, $\lambda^{\rightarrow}_{\mathrm{P,S}}$, whose definition is parameterized by a set of given operations over primitive types. This language is an extension of the simply-typed lambda calculus, with labeled types as used in information flow languages (see, e.g., [30]). Our language design and analysis are valid for any secure execution platform, and are thus parameterized over implementation details such as the number of servers, the form of cryptography used, and the form and extent of communication in the system. From the programmer's standpoint, different cryptographic backends that support the same operations provide the same programming experience.

In order to prove desired correctness and security properties, we formulate both a standard *reference semantics* for $\lambda^{\rightarrow}_{\mathrm{P,S}}$ and a *distributed semantics* that allows an arbitrary number of servers to communicate with the client and with each other in order to complete a computation. Correctness of the distributed semantics is then proved by showing an equivalence with the reference semantics, while security properties are proved by analyzing the information available to the servers throughout the program execution.

5.1 Syntax

Our core language, $\lambda^{\rightarrow}_{\mathrm{P,S}}$, extends the simply-typed lambda calculus with primitive values, conditionals, mutable stores, and a fixpoint operator. Figure 5 describes the language syntax. Throughout this section, we assume primitive operations (op_i) and a secure execution platform, as specified in Section 4.

Figure 5 $\lambda^{\rightarrow}_{\mathrm{P,S}}$ syntax.

$$
\begin{array}{rl}
\text{Types} & t ::= Y \mid (\tau \to \tau) \mid Y^{\ell}\ \mathrm{ref} \\
\text{Labeled types} & \tau ::= t^{\ell} \\
\text{Values} & v ::= y \mid a \mid \lambda x.e \mid \texttt{fix}\ f.\lambda x.e \\
\text{Expressions} & e ::= v \mid x \mid X \mid e\ e \\
 & \quad \mid \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \\
 & \quad \mid \texttt{op}^{t}_{i}(e, \ldots, e) \mid \texttt{ref}\ e \mid\ !e \mid e := e \\
 & \quad \mid \texttt{reveal}\ e \\
\text{Programs} & p ::= \texttt{read}(X_1 : Y_1, \ldots, X_n : Y_n); e
\end{array}
$$

In addition to standard constructs, expressions in $\lambda^{\rightarrow}_{\mathrm{P,S}}$ may include variables bound at the program level by the `read` construct, representing secret values input by the clients before the body of the program is evaluated; these input variables are represented by capital letters X (in contrast to lambda-bound variables, which use lowercase letters x), to emphasize the phase distinction between input processing and evaluation of the program body[3]. Programs in $\lambda^{\rightarrow}_{\mathrm{P,S}}$ may also include `reveal` operations, which specify that the value in question need not be kept secret during the remainder of the computation. In addition to decrypting final result values (if they are intended to be public), the `reveal` construct also enables declassification of intermediate results, giving programmers fine-grained control over the tradeoff between performance and total secrecy. We note that references in $\lambda^{\rightarrow}_{\mathrm{P,S}}$ are limited to primitive types, since we later depend on some restricted termination results (and termination need not hold if references of arbitrary type are permitted).

Figure 6 $\lambda^{\rightarrow}_{\mathrm{P,S}}$ syntax (extended).

$$
\begin{array}{l}
\tilde{v} ::= v \mid \tilde{y} \mid \lambda x.\tilde{e} \mid \texttt{fix}\ f.\lambda x.\tilde{e} \mid a \mid \varphi(\tilde{v}_1, \tilde{v}_2, \tilde{v}_3) \\
\tilde{e} ::= e \mid \tilde{v} \mid x \mid X \mid \tilde{e}\ \tilde{e} \mid \texttt{if}\ \tilde{e}\ \texttt{then}\ \tilde{e}\ \texttt{else}\ \tilde{e} \\
\quad \mid \texttt{op}^{t}_{i}(\tilde{e}, \ldots \tilde{e}) \mid \texttt{ref}\ \tilde{e} \mid\ !\tilde{e} \mid \tilde{e} := \tilde{e} \\
\quad \mid \texttt{reveal}\ \tilde{e}
\end{array}
$$

In order to reason about the evaluation of programs we extend the language syntax as shown in Figure 6. The previous syntax (Figure 5), is a subset of the extended syntax, and encompasses all expressions that the programmer can write (which we refer to as the "surface syntax"), as well as values a, which range over a countably infinite set of abstract memory locations (as in standard presentations of lambda calculus with mutable references). However, the extensions are necessary to describe values that may result from evaluations in the distributed semantics (described below), despite not being present at the surface level.

In particular, we have a case for possibly-hidden primitive values $\tilde{y} \in \mathcal{E}_{\ell}(Y)$. As described in Section 4, $\tilde{y} \in \mathcal{E}_{\mathrm{S}}(Y)$ is a hidden value, while $\tilde{y} \in \mathcal{E}_{\mathrm{P}}(Y) = Y$ is a publicly-visible value but, for notational uniformity, may be regarded as hidden at the public confidentiality level. The value $\varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3)$, where $\tilde{b} \in \mathcal{E}_{\mathrm{S}}(\mathrm{bool})$,

[3]While we enforce this phase distinction to simplify the formalism, it is a straightforward extension to allow input and output operations throughout the program, rather than restricting inputs to the beginning and outputs to the final value.

and $\tilde{v}_2$, $\tilde{v}_3$ are (extended) values intuitively represents a "deferred decision". Because the boolean value $\tilde{b}$ was secret, the system could not decide which of two values was the result of a conditional expression, and thus had to propagate both. For example, the following expression:

$$\varphi(\tilde{b}, \lambda x.0, \lambda x.1)$$

might be the result of a conditional in which the condition evaluates to the secret boolean $\tilde{b}$. Note, however, that a value such as $\varphi(\tilde{b}, 17, 42)$ would never occur, since (as detailed below) the system would be able to evaluate the condition homomorphically on hidden primitive values and produce a hidden primitive value.

5.2 Static semantics

The static semantics rules are shown in Figure 7. In the typing judgment, Γ, as usual, represents the typing context for lambda-bound variables, while Σ represents the store typing (i.e., if $\Sigma(a) = Y^\ell$, then the (extended) expression $\tilde{e}$ should be evaluated in a store that maps a to an element of type Y^ℓ).

The key feature of the static semantics is the presence of value labels, $\ell \in \{\mathrm{P}, \mathrm{S}\}$, as well as the context label, $C \in \{\mathrm{P}, \mathrm{S}\}$. The intuitive meaning of these labels is similar to their meaning in standard information flow systems [30, 31]. A value labeled t^ℓ is a value of type t at confidentiality level ℓ. Our `reveal` operator, like traditional declassification operators, indicates that a particular value is allowed to be leaked; statically, it acts as a cast from S to P. A context label C signifies that an expression can be typed in a context in which control flow may depend on values of confidentiality level C. In our language, as in standard information flow systems, this context restriction is used to prevent implicit flows, such as an update to a public memory location inside a secret conditional. However, in our model, we must also regard any deviation in control flow as a publicly visible effect. Thus, not only updates to public memory locations, but also side-effects such as `reveal`, as well as unbounded iteration (or potential nontermination), must be independent of secret values. This makes our system, by necessity, strictly more restrictive than standard termination-insensitive information-flow systems such as JFlow/Jif [32].

One can view these additional restrictions as specifying that the control flow of the program is visible to the adversary, as in an oblivious memory model [33]; but, while intuitively helpful, this analogy does not present a complete picture. In a sense, the purpose of information flow control in our system is dual to its purpose in traditional language-based security: in a traditional system, the machine model permits the implementation of arbitrary operations, and the information flow system must statically rule out code that would leak information; while in our system, the machine model permits only operations that would not leak information (since secret values are encrypted or otherwise hidden), and thus our system must statically rule out code that would not be implementable in such a model.

In light of these objectives, we include additional restrictions in the static semantics, similar in flavor to some type-and-effect systems [34, 35], in addition to standard information flow control constructs. For example, in the conditional rule, we require that each branch is well-typed at a confidentiality level at least as high as that of the condition. As in other information flow systems, this restriction rules out the canonical example of implicit flow (where s is a secret boolean, and p is a reference to a public integer):

$$\texttt{if } s \texttt{ then } p := 0 \texttt{ else } p := 1$$

Since s is secret, the branches must type-check in a secret context; but the rule for assignment specifies that the label of the reference receiving the assignment is at least as high as that of the surrounding context, which cannot be satisfied by this expression. Our system also rules out the following expression:

$$\texttt{if } s_1 \texttt{ then } (\texttt{reveal } s_2) \texttt{ else } 17$$

The `reveal` operation would cause a publicly-observable side-effect (namely, causing a value to be "unhidden"), and thus it cannot be typed in a secret context. Similarly, our restrictions also rule out the following

Figure 7 Static semantics.

$$\frac{\tilde{y} \in \mathcal{E}_{\ell'}(Y) \quad \ell' \sqsubseteq \ell}{\Gamma, \Sigma, C \vdash \tilde{y} : Y^{\ell}} \qquad \frac{}{\Gamma, \Sigma, C \vdash x : \Gamma(x)} \qquad \frac{\Sigma(a) = Y^{\ell'}}{\Gamma, \Sigma, C \vdash a : (Y^{\ell'}\ \mathrm{ref})^{\ell}}$$

$$\frac{\Gamma, \Sigma, C \vdash \tilde{b} : \mathrm{bool}^{\mathrm{S}} \qquad \Gamma, \Sigma, C \vdash \tilde{v}_2 : (\tau_1 \xrightarrow{C} \tau_2)^{\ell_2} \qquad \Gamma, \Sigma, C \vdash \tilde{v}_3 : (\tau_1 \xrightarrow{C} \tau_2)^{\ell_3}}{\Gamma, \Sigma, C \vdash \varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3) : (\tau_1 \xrightarrow{C} \tau_2)^{\mathrm{S}}}$$

$$\frac{\Gamma, \Sigma, C \vdash \tilde{b} : \mathrm{bool}^{\mathrm{S}} \qquad \Gamma, \Sigma, C \vdash \tilde{v}_2 : (\tau\ \mathrm{ref})^{\ell_2} \qquad \Gamma, \Sigma, C \vdash \tilde{v}_3 : (\tau\ \mathrm{ref})^{\ell_3}}{\Gamma, \Sigma, C \vdash \varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3) : (\tau\ \mathrm{ref})^{\mathrm{S}}}$$

$$\frac{\Gamma[x \mapsto \tau_1], \Sigma, C' \vdash e : \tau}{\Gamma, \Sigma, C \vdash \lambda x.\tilde{e} : (\tau_1 \xrightarrow{C'} \tau)^{\ell}}$$

$$\frac{\Gamma[f \mapsto (\tau_1 \xrightarrow{\mathrm{P}} \tau)^{\ell}, x \mapsto \tau_1], \Sigma, C' \vdash \tilde{e} : \tau}{\Gamma, \Sigma, C \vdash \mathtt{fix}\ f.\lambda x.\tilde{e} : (\tau_1 \xrightarrow{\mathrm{P}} \tau)^{\ell}}$$

$$\frac{\forall j \in \{1, \ldots, r\}\,.\,\Gamma, \Sigma, C \vdash \tilde{e}_j : ((\mathrm{dom}\,\mathrm{op}_i)_j)^{\ell_j} \wedge \ell_j \sqsubseteq \ell}{\Gamma, \Sigma, C \vdash \mathtt{op}_i(\tilde{e}_1, \ldots, \tilde{e}_r) : (\mathrm{cod}\,\mathrm{op}_i)^{\ell}}$$

$$\frac{\Gamma, \Sigma, C \vdash \tilde{e} : (\tau_1 \xrightarrow{C'} t^{\ell'})^{\ell} \qquad \Gamma, \Sigma, C \vdash \tilde{e}_1 : \tau_1 \qquad \ell \sqsubseteq C' \qquad C \sqsubseteq C' \qquad \ell \sqcup \ell' \sqsubseteq \ell''}{\Gamma, \Sigma, C \vdash \tilde{e}\,\tilde{e}_1 : t^{\ell''}}$$

$$\frac{\Gamma, \Sigma, C \vdash \tilde{e}_1 : \mathrm{bool}^{\ell'} \qquad \Gamma, \Sigma, C \sqcup \ell' \vdash \tilde{e}_2 : t^{\ell''} \qquad \Gamma, \Sigma, C \sqcup \ell' \vdash \tilde{e}_3 : t^{\ell''} \qquad \ell' \sqcup \ell'' \sqsubseteq \ell}{\Gamma, \Sigma, C \vdash \mathtt{if}\ \tilde{e}_1\ \mathtt{then}\ \tilde{e}_2\ \mathtt{else}\ \tilde{e}_3 : t^{\ell}}$$

$$\frac{\Gamma, \Sigma, C \vdash \tilde{e}_1 : (t^{\ell'}\ \mathrm{ref})^{\ell} \qquad \Gamma, \Sigma, C \vdash \tilde{e}_2 : t^{\ell'} \qquad C \sqsubseteq \ell' \qquad \ell \sqsubseteq \ell'}{\Gamma, \Sigma, C \vdash \tilde{e}_1 := \tilde{e}_2 : \mathrm{unit}^{\ell''}}$$

$$\frac{\Gamma, \Sigma, C \vdash \tilde{e} : Y^{\mathrm{S}}}{\Gamma, \Sigma, \mathrm{P} \vdash \mathtt{reveal}\ \tilde{e} : Y^{\ell}}$$

$$\frac{\Gamma, \Sigma \vdash \tilde{e} : Y^{\ell'}}{\Gamma, \Sigma, C \vdash \mathtt{ref}\ \tilde{e} : (Y^{\ell'}\mathrm{ref})^{\ell}}$$

$$\frac{\Gamma, \Sigma, C \vdash \tilde{e} : (Y^{\ell'}\ \mathrm{ref})^{\ell''} \qquad \ell' \sqsubseteq \ell \qquad \ell'' \sqsubseteq \ell}{\Gamma, \Sigma, C \vdash !\tilde{e} : Y^{\ell}}$$

$$\frac{\{X_1 \mapsto Y_1 \ldots X_n \mapsto Y_n\}, \emptyset, C \vdash e : \tau}{\vdash \mathtt{read}(X_1 : Y_1 \ldots X_n : Y_n); e : \tau}$$

expression, although the reasoning is more involved:

$$\mathtt{fix}\ f\,.\,\lambda s\,.\,(\mathtt{if}\ s \leq 0\ \mathtt{then}\ 1\ \mathtt{else}\ s * f(s-1))$$

Notably, in the rules for lambda abstraction and recursive function definition, we add a context label above the arrow $(\tau_1 \xrightarrow{C'} \tau)$, signifying that the resulting function, when applied, may generate effects that must be

confined to a context C' (independent of the context C in which the term itself is evaluated).[4] In the case of general recursion, this effect label is assumed to be P, since we conservatively assume that any application of a function defined by general recursion (either within its own definition, or as a standalone expression) may generate unpredictable, publicly-observable control flow behavior, including divergence. It is also worth noting that a function's effect context label, C', is only taken into account when the function is applied; for instance, the following expression is well-typed:

$$\texttt{if}\ s\ \texttt{then}\ (\texttt{fix}\ f\,.\,\lambda x\,.\,f\ x)\ \texttt{else}\ (\lambda x\,.\,x)$$

The dependence of confidentiality on control flow also raises more subtle issues that necessitate additional typing restrictions. For instance, the following expression clearly does not preserve confidentiality (and thus cannot be implemented on a secure execution platform):

$$(\texttt{if}\ s\ \texttt{then}\ p_1\ \texttt{else}\ p_2) := 42$$

The underlying principle in this example is that values escaping a secret context may carry secret information with them, and thus computations that depend on them must be well-typed in a secret context. Indeed, this restriction is reflected in the static semantics. In the assignment rule, the restriction $\ell \sqsubseteq \ell'$ expresses that information could flow from the confidentiality label of the reference itself to the label of its contents: i.e., one can never write to a secret reference cell (a memory location whose identity is secret) if its contents may be public. A similar situation arises with lambda abstractions, as in the following (ill-typed) example:

$$(\texttt{if}\ s_1\ \texttt{then}\ (\lambda x\,.\,\texttt{reveal}\ s_2)\ \texttt{else}\ (\lambda x\,.\,17))\ ()$$

In the application rule, as above, the restriction $\ell \sqsubseteq C'$ expresses that information could flow from the identity of the function to the context in which it executes: i.e., one can never apply a function whose identity is secret if it might have publicly-observable effects.

5.3 Dynamic semantics

We now give a standard dynamic semantics for $\lambda^{\rightarrow}_{\mathrm{P,S}}$, the "reference semantics" (Figure 8), extending the evaluation rules for the simply-typed lambda calculus. As described above, the reference semantics gives a precise specification for the evaluation of a program, against which the actual secure implementation (the "distributed semantics") can be compared for correctness.

In the reference semantics, the mutable store μ maps addresses (generated by `ref`) to the values they contain, while the environment κ represents the initial (secret) values supplied by the client. The judgment $(\mu, e) \downarrow (v, \mu', \mathcal{O})$ indicates that the expression e, when evaluated in an initial store μ, produces a value v, a final store μ', and a sequence $\mathcal{O}$ of "observations", holding all values ever supplied to `reveal` throughout the evaluation (where the operator $\|$ indicates concatenation of observation sequences). The observation sequences are important in proving security properties (Theorem 2, below), as we will show that an appropriately constrained adversary learns nothing except what is logically entailed by these observations. The `read` construct only serves to bind the initial client-input variables, and is essentially a no-op; for clarity, however, we retain it in the syntax, since in the distributed semantics (Figure 10), it will represent the initial "hiding" operation (potentially including communication between the client and servers).

We give an additional dynamic semantics, the "distributed semantics" (Figure 10), that reflects the actual steps taken by an implementation in terms of a secure execution platform. Values in the distributed semantics may be "hidden" from the server(s) computing on them – shared or otherwise encrypted, according to the

[4] For simplicity of presentation, lambda abstractions in our syntax do not carry the traditional type annotations, $\lambda(x : t).e$. In any concrete implementation, we assume that terms are appropriately annotated with (unlabeled) types, so that type-checking becomes tractable while preserving label subtype polymorphism. Since such restrictions permit the typing of strictly fewer terms, our theoretical results still hold.

Figure 8 Reference semantics for $\lambda^{\rightarrow}_{\mathrm{P,S}}$.

$$\frac{}{(v,\mu)\downarrow(v,\mu,\varepsilon)}$$

$$\frac{\forall j\in\{1,\ldots,r\}\,.\,(e_j,\mu_{j-1})\downarrow(y_j,\mu_j,\mathcal{O}_j)}{(\mathrm{op}_i^t(e_1,\ldots,e_r),\mu_0)\downarrow(\mathrm{op}_i^t(y_1,\ldots,y_r),\mu',\mathcal{O}_1\|\ldots\|\mathcal{O}_r)}$$

$$\frac{(e_0,\mu)\downarrow(\texttt{fix}\ f.\lambda(x:t).e_0',\mu_0,\mathcal{O}_0)\qquad(e_1,\mu_0)\downarrow(v_1,\mu_1,\mathcal{O}_1)\qquad(e_0'[\texttt{fix}\ f.\lambda x.e_0')/f,v_1/x],\mu_1)\downarrow(v',\mu',\mathcal{O}_2)}{(e_0\,e_1,\mu)\downarrow(v',\mu',\mathcal{O}_0\|\mathcal{O}_1\|\mathcal{O}_2)}$$

$$\frac{(e_0,\mu)\downarrow(\lambda x.e_0',\mu_0,\mathcal{O}_0)\qquad(e_1,\mu_0)\downarrow(v_1,\mu_1,\mathcal{O}_1)\qquad(e[v_1/x],\mu_1)\downarrow(v',\mu',\mathcal{O}_2)}{(e_0\,e_1,\mu)\downarrow(v',\mu',\mathcal{O}_0\|\mathcal{O}_1\|\mathcal{O}_2)}$$

$$\frac{(e_1,\mu)\downarrow(\texttt{true},\mu_1,\mathcal{O}_1)\qquad(e_2,\mu_1)\downarrow(v',\mu',\mathcal{O}_2)}{(\texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3,\mu)\downarrow(v',\mu',\mathcal{O}_1\|\mathcal{O}_2)}$$

$$\frac{(e_1,\mu)\downarrow(\texttt{false},\mu_1,\mathcal{O}_1)\qquad(e_3,\mu_1)\downarrow(v',\mu',\mathcal{O}_3)}{(\texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3,\mu)\downarrow(v',\mu',\mathcal{O}_1\|\mathcal{O}_3)}$$

$$\frac{(e_1,\mu)\downarrow(a,\mu_1,\mathcal{O}_1)\qquad(e_2,\mu_1)\downarrow(v_2,\mu_2,\mathcal{O}_2)}{(e_1:=e_2,\mu)\downarrow((),\mu_2[a\mapsto v_2],\mathcal{O}_1\|\mathcal{O}_2)}$$

$$\frac{(e,\mu)\downarrow(y,\mu',\mathcal{O})}{(\texttt{reveal}\ e,\mu)\downarrow(y,\mu',\mathcal{O}\|y)}$$

$$\frac{(e,\mu)\downarrow(v_1,\mu_1,\mathcal{O})\qquad a\notin\mathrm{dom}\,\mu_1}{(\texttt{ref}\ e,\mu)\downarrow(a,\mu_1[a\mapsto v_1],\mathcal{O})}$$

$$\frac{(e,\mu)\downarrow(a,\mu',\mathcal{O})\qquad a\in\mathrm{dom}\,\mu'}{(!e,\mu)\downarrow(\mu'(a),\mu',\mathcal{O})}$$

$$\frac{(e[\kappa(X_1)/X_1\ldots\kappa(X_n)/X_n],\emptyset)\downarrow(v,\mu,\mathcal{O})}{(\kappa,\texttt{read}(X_1:Y_1,\ldots,X_n:Y_n);e)\downarrow(v,\mu,\mathcal{O})}$$

primitives of the secure execution platform. Thus, the distributed semantics makes central use of the lifted, or homomorphic, operations of the platform. In particular, we use the lifted primitives, $\mathrm{Enc}_{\ell_1,\ldots,\ell_r}(\mathrm{op}_i)$, to match the execution of the ordinary primitive operations op_i in the reference semantics. It is also worth noting that the distributed semantics implicitly act on probability distributions. As described above, the output of an operation in the secure execution platform is a distribution, and thus, when an operation $f(\tilde{x})$ appears in a semantic rule, it should be considered lifted to act on distributions (i.e., $f(\tilde{x})$ represents the sum of distributions $\sum_{x\in\mathrm{dom}\,\tilde{x}} p_{\tilde{x}}(x)f(x)$, where $p_{\tilde{x}}$ is a probability mass function of the distribution $\tilde{x}$).

Before presenting the distributed semantics, we need a few auxiliary definitions. First, we define a join operator, Φ^{ι}, that recursively joins two values according to a secret boolean (applying a primitive branching operator, if the values are primitive, and otherwise "deferring" the join by wrapping the boolean and operands in a φ symbol):

$$\Phi^\iota(\tilde{b}, \tilde{v}_2, \tilde{v}_3) = \begin{cases} (\tilde{v}_2, \varepsilon) \text{ if } \tilde{v}_2 \in \mathcal{E}_{\mathrm{P}}(Y) \text{ and } \tilde{v}_2 = \tilde{v}_3 \\ \mathrm{Enc}_{\ell_1,\ell_2,\ell_3}(\mathrm{op}_{\mathrm{Br}(Y)})(\tilde{b}, \tilde{v}_2, \tilde{v}_3, \iota) \\ \qquad \text{if } \tilde{b} \in \mathcal{E}_{\ell_1}(\mathrm{bool}), \tilde{v}_2 \in \mathcal{E}_{\ell_2}(Y), \tilde{v}_3 \in \mathcal{E}_{\ell_3}(Y) \\ (\varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3), \varepsilon) \text{ otherwise} \end{cases}$$

We extend Φ^ι to stores as follows. First, assume the addresses $(a_1, \ldots, a_n)$ in $\mathrm{dom}\,\tilde{\mu}_2 \cup \mathrm{dom}\,\tilde{\mu}_3$ are ordered lexicographically by the pair of type signatures of $\tilde{\mu}_2(a_i)$ and $\tilde{\mu}_3(a_i)$: i.e., by $(\mathrm{sig}_{\tilde{\mu}_2}(a_i), \mathrm{sig}_{\tilde{\mu}_3}(a_i))$, where:

$$\mathrm{sig}_{\tilde{\mu}}(a_i) = \begin{cases} (Y, \ell) & \text{if } \tilde{\mu}(a_i) \in \mathcal{E}_\ell(Y) \\ * & \text{otherwise} \end{cases}$$

Then we define:

$$\Phi^\iota(\tilde{b}, \tilde{\mu}_2, \tilde{\mu}_3) = \\ (\{a_i \mapsto \pi_1(Z(a_i)) : i \in \{1, \ldots, n\}\}, \pi_2(Z(a_1)) \| \cdots \| \pi_2(Z(a_n)))$$

where:

$$Z(a) = \begin{cases} \Phi^\iota(\tilde{b}, \tilde{\mu}_2(a), \tilde{\mu}_3(a)) & \text{if } a \in \mathrm{dom}\,\tilde{\mu}_2 \cap \mathrm{dom}\,\tilde{\mu}_3 \\ (\tilde{\mu}_2(a), \varepsilon) & \text{if } a \in \mathrm{dom}\,\tilde{\mu}_2 \setminus \mathrm{dom}\,\tilde{\mu}_3 \\ (\tilde{\mu}_3(a), \varepsilon) & \text{if } a \in \mathrm{dom}\,\tilde{\mu}_3 \setminus \mathrm{dom}\,\tilde{\mu}_2 \end{cases}$$

In addition, we use the join operator Φ^ι to define the result of updating a reference (consisting of either an address, or a φ symbol of references), as well as the result of retrieving a reference's value:

Definition 2 (Update Operator)**.**

$$\begin{aligned} \mathrm{update}^\iota(\tilde{\mu}, a, \tilde{v}) &= (\tilde{\mu}[a \mapsto \tilde{v}], \varepsilon) \\ \mathrm{update}^\iota(\tilde{\mu}, \varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3), \tilde{v}) &= (\pi_1(Z), \pi_2(Z_2) \| \pi_2(Z_3) \| \pi_2(Z)) \\ \text{where } Z_2 &= \mathrm{update}^\iota(\tilde{\mu}, \tilde{v}_2, \tilde{v}) \\ Z_3 &= \mathrm{update}^\iota(\tilde{\mu}, \tilde{v}_3, \tilde{v}) \\ Z &= \Phi^\iota(\tilde{b}, \pi_1(Z_2), \pi_1(Z_3)) \end{aligned}$$

Definition 3 (Select Operator)**.**

$$\begin{aligned} \mathrm{select}^\iota(\tilde{\mu}, a) &= (\tilde{\mu}(a), \varepsilon) \\ \mathrm{select}^\iota(\tilde{\mu}, \varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3), \tilde{v}) &= (\pi_1(Z), \pi_2(Z_2) \| \pi_2(Z_3) \| \pi_2(Z)) \\ \text{where } Z_2 &= \mathrm{select}^\iota(\tilde{\mu}, \tilde{v}_2) \\ Z_3 &= \mathrm{select}^\iota(\tilde{\mu}, \tilde{v}_3) \\ Z &= \Phi^\iota(\tilde{b}, \pi_1(Z_2), \pi_1(Z_3)) \end{aligned}$$

We note that all of the above definitions may execute branching operators in order to join primitive values, and thus may produce communication traces. For convenience, we will also use subscripts to indicate projections of the above operators (e.g., $\Phi^\iota_1(\tilde{v}_1, \tilde{v}_2, \tilde{v}_3) = \pi_1(\Phi^\iota(\tilde{v}_1, \tilde{v}_2, \tilde{v}_3))$).

We now present the distributed semantics (Figures 9 and 10). In this semantics, we use the extended form of the expression syntax ($\tilde{e}$), signifying that expressions may contain possibly-hidden primitive values $\tilde{y} \in \mathcal{E}_\ell(Y)$, as well as φ symbols $\varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3)$ (discussed below). The evaluation judgment $(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \mathcal{O})$ signifies that an expression $\tilde{e}$, when evaluated in an initial store $\tilde{\mu}$ and initialized with platform parameters ι, evaluates to the value $\tilde{v}'$, producing a final store $\tilde{\mu}'$, a communication trace T, and observations $\mathcal{O}$ (again containing all values *ever* supplied to the `reveal` operator). In contrast to the reference semantics, the distributed semantics

Figure 9 Distributed dynamic semantics for $\lambda^{\rightarrow}_{\mathrm{P,S}}$ (part 1 of 2).

$$\frac{}{(\tilde{v}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}, \tilde{\mu}, \varepsilon, \varepsilon)}$$

$$\frac{\begin{array}{c}\forall j \in \{1, \ldots, r\} \,.\, (e_j, \tilde{\mu}_{j-1}, \iota) \Downarrow (\tilde{y}_j, \tilde{\mu}_j, T_j, \mathcal{O}_j) \qquad \forall j \in \{1, \ldots, r\} \,.\, \tilde{y}_j \in \mathcal{E}_{\ell_j}(Y_j) \\ \mathrm{Enc}_{\ell_1,\ldots,\ell_r}(\mathrm{op}_i)(\tilde{y}_1, \ldots, \tilde{y}_r, \iota) = (\tilde{y}', T) \qquad T' = T_1 \| \cdots \| T_r \| T \qquad \mathcal{O}' = \mathcal{O}_1 \| \cdots \| \mathcal{O}_r\end{array}}{(\mathtt{op}_i(e_1, \ldots, e_r), \tilde{\mu}_0, \iota) \Downarrow (\tilde{y}', \tilde{\mu}_r, T', \mathcal{O}')}$$

$$\frac{\begin{array}{c}(\tilde{e}_0, \tilde{\mu}, \iota) \Downarrow (\lambda x.\tilde{e}'_0, \tilde{\mu}_0, T_0, \mathcal{O}_0) \qquad (\tilde{e}_1, \tilde{\mu}_0, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T_1, \mathcal{O}_1) \\ (\tilde{e}'_0[\tilde{v}_1/x], \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T_2, \mathcal{O}_2)\end{array}}{(\tilde{e}_0\ \tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', \mathcal{O}_0 \| \mathcal{O}_1 \| \mathcal{O}_2, \mathcal{O}_0 \| \mathcal{O}_1 \| \mathcal{O}_2)}$$

$$\frac{\begin{array}{c}(\tilde{e}_0, \tilde{\mu}, \iota) \Downarrow (\mathtt{fix}\ f.\lambda x.\tilde{e}'_0, \tilde{\mu}_0, T_0, \mathcal{O}_0) \qquad (\tilde{e}_1, \tilde{\mu}_0, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T_1, \mathcal{O}_1) \\ (\tilde{e}'_0[(\mathtt{fix}\ f.\lambda x.\tilde{e}'_0)/f, \tilde{v}_1/x], \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T_2, \mathcal{O}_2)\end{array}}{(\tilde{e}_0\ \tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T_0 \| T_1 \| T_2, \mathcal{O}_0 \| \mathcal{O}_1 \| \mathcal{O}_2)}$$

$$\frac{\begin{array}{c}(\tilde{e}_0, \tilde{\mu}, \iota) \Downarrow (\varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3), \tilde{\mu}_0, T_0, \mathcal{O}_0) \\ \tilde{b} \in \mathcal{E}_{\mathrm{S}}(\mathrm{bool}) \qquad (\tilde{e}_1, \tilde{\mu}_0, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T_1, \mathcal{O}_1) \qquad (\tilde{v}_2\ \tilde{v}_1, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}'_2, \tilde{\mu}_2, T_2, \mathcal{O}_2) \\ (\tilde{v}_3\ \tilde{v}_1, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}'_3, \tilde{\mu}_3, T_3, \mathcal{O}_3) \qquad (\tilde{v}', T_4) = \Phi^{\iota}(\tilde{b}, \tilde{v}'_2, \tilde{v}'_3) \qquad (\tilde{\mu}', T_5) = \Phi^{\iota}(\tilde{b}, \tilde{\mu}_2, \tilde{\mu}_3)\end{array}}{(\tilde{e}_0\ \tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T_0 \| \cdots \| T_5, \mathcal{O}_0 \| \cdots \| \mathcal{O}_3)}$$

$$\frac{(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{y}, \tilde{\mu}', T_1, \mathcal{O}) \qquad (\tilde{y}', T_2) = \mathrm{Dec}_{\mathrm{S}}(\tilde{y}, \iota)}{(\mathtt{reveal}\ \tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{y}', \tilde{\mu}', T_1 \| T_2, \mathcal{O} \| \tilde{y}')}$$

$$\frac{(\tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\mathtt{true}, \tilde{\mu}_1, T_1, \mathcal{O}_1) \qquad (\tilde{e}_2, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T_2, \mathcal{O}_2) \qquad T = T_1 \| T_2 \qquad \mathcal{O} = \mathcal{O}_1 \| \mathcal{O}_2}{(\mathtt{if}\ \tilde{e}_1\ \mathtt{then}\ \tilde{e}_2\ \mathtt{else}\ \tilde{e}_3, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \mathcal{O})}$$

$$\frac{(\tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\mathtt{false}, \tilde{\mu}_1, T_1, \mathcal{O}_1) \qquad (\tilde{e}_3, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T_3, \mathcal{O}_3) \qquad T = T_1 \| T_3 \qquad \mathcal{O} = \mathcal{O}_1 \| \mathcal{O}_3}{(\mathtt{if}\ \tilde{e}_1\ \mathtt{then}\ \tilde{e}_2\ \mathtt{else}\ \tilde{e}_3, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \mathcal{O})}$$

specifies that `reveal` and `read` operations result in communication between the parties according to the parameters of the secure execution platform: the `reveal` rule specifies that the parties execute an "unhiding", or decryption, $\mathsf{Dec}_{\mathrm{S}}$, of a secret value, while the `read` rule specifies that the client initializes the secure execution platform, distributes to each server its view of the initial parameters ($\Pi_{\{S_i\}}(\iota)$), and executes the "hiding", $\mathsf{Enc}_{\mathrm{S}}$, of the secret values in the client's initial environment κ.

For most of the other rules, we can intuitively regard evaluation in the distributed semantics as proceeding in lock-step with the reference semantics, executing the corresponding operations as provided by the secure execution platform, as long as control flow proceeds independently of secret values. For example, the distributed semantics provides two distinct rules for conditionals (`if-then-else`). When the condition evaluates to a public boolean (i.e., an element of $\mathcal{E}_{\mathrm{P}}(\mathrm{bool}) = \mathrm{bool}$), the evaluation precisely mirrors the reference semantics; observations and communication traces are propagated unchanged.

When control flow does depend on a secret value, however, the distributed semantics yields different behavior. For instance, when a condition evaluates to a secret boolean $\tilde{b}$, the distributed semantics specifies that both branches should be evaluated in the same store $\tilde{\mu}_1$, each generating its own value and resulting store $(\tilde{v}_2, \tilde{\mu}_2)$ and $(\tilde{v}_3, \tilde{\mu}_3)$. The distributed semantics then merges these values and stores, according to the secret boolean $\tilde{b}$, using the Φ^{ι} function. For example, if $\tilde{v}_2$ and $\tilde{v}_3$ are (hidden) primitive values representing, respectively, 17 and

Figure 10 Distributed dynamic semantics for $\lambda^{\rightarrow}_{\mathrm{P,S}}$ (part 2 of 2).

$$\frac{\begin{array}{c}\tilde{b} \in \mathcal{E}^{\iota}_{\mathrm{S}}(\mathrm{bool}) \\ (\tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{b}, \tilde{\mu}_1, T_1, \mathcal{O}_1) \\ (\tilde{e}_2, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}_2, \tilde{\mu}_2, T_2, \mathcal{O}_2) \qquad (\tilde{e}_3, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}_3, \tilde{\mu}_3, T_3, \mathcal{O}_3) \qquad \mathrm{dom}\,\tilde{\mu}_2 \cap \mathrm{dom}\,\tilde{\mu}_3 = \mathrm{dom}\,\tilde{\mu}_1 \\ (\tilde{v}', T_4) = \Phi^{\iota}(\tilde{b}, \tilde{v}_2, \tilde{v}_3) \qquad (\tilde{\mu}', T_5) = \Phi^{\iota}(\tilde{b}, \tilde{\mu}_2, \tilde{\mu}_3) \qquad T = T_1 \| \cdots \| T_5 \qquad \mathcal{O} = \mathcal{O}_1 \| \mathcal{O}_2 \| \mathcal{O}_3\end{array}}{(\texttt{if}\ \tilde{e}_1\ \texttt{then}\ \tilde{e}_2\ \texttt{else}\ \tilde{e}_3, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \mathcal{O})}$$

$$\frac{\begin{array}{c}(\tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T_1, \mathcal{O}_1) \qquad (\tilde{e}_2, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}, \tilde{\mu}_2, T_2, \mathcal{O}_2) \\ (\tilde{\mu}', T') = \mathrm{update}^{\iota}(\tilde{\mu}_2, \tilde{v}_1, \tilde{v}_2)\end{array}}{(\tilde{e}_1 := \tilde{e}_2, \tilde{\mu}, \iota) \Downarrow ((), \tilde{\mu}', T_1 \| T_2 \| T', \mathcal{O}_1 \| \mathcal{O}_2)}$$

$$\frac{(\tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T, \mathcal{O}) \qquad a \notin \mathrm{dom}\,\tilde{\mu}}{(\texttt{ref}\ \tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (a, \tilde{\mu}_1[a \mapsto \tilde{v}], T, \mathcal{O})}$$

$$\frac{(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}'_1, \tilde{\mu}', T_1, \mathcal{O}) \qquad (\tilde{v}', T') = \mathrm{select}^{\iota}(\tilde{\mu}', \tilde{v}_1)}{(!\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T_1 \| T', \mathcal{O})}$$

$$\frac{\begin{array}{c}\iota = \mathrm{Init}() \\ T_0 = \{(C, S_i, \Pi_{\{S_i\}}(\iota)) : 1 \leq i \leq N\} \qquad \forall j \in \{1, \ldots, r\}.(\tilde{v}_j, T'_j) = \mathrm{Enc}_{\mathrm{S}}(\kappa(X_j), \iota) \\ (e[\tilde{v}_1/X_1 \ldots \tilde{v}_r/X_r], \emptyset, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T', \mathcal{O}) \qquad T = T_0 \| T'_1 \| \ldots \| T'_r \| T'\end{array}}{(\kappa, \texttt{read}(X_1 : Y_1, \ldots X_r : Y_r); e) \Downarrow (\tilde{v}', \tilde{\mu}', T, \mathcal{O})}$$

42, and the boolean $\tilde{b}$ is a (hidden) representation of `true`, then the result:

$$\Phi^{\iota}(\tilde{b}, \tilde{v}_2, \tilde{v}_3) = \mathrm{Enc}_{\mathrm{S,S,S}}(\mathrm{op}_{\mathrm{Br(int)}})(\tilde{b}, \tilde{v}_2, \tilde{v}_3, \iota)$$

will be a hidden representation of 17, along with whatever traces were produced by the execution of the protocol operation $\mathrm{Enc}_{\mathrm{S,S,S}}(\mathrm{op}_{\mathrm{Br(int)}})$. On the other hand, if $\tilde{v}_2$ and $\tilde{v}_3$ are non-primitive values – e.g., lambda abstractions – then the join function, Φ^{ι}, is unable to arithmetize the branch immediately, since the arguments to which the abstractions will be applied are not available. Thus, Φ^{ι} wraps these operands in the special "deferred decision" symbol φ:

$$\Phi^{\iota}(\tilde{b}, \tilde{v}_2, \tilde{v}_3) = \varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3)$$

The contents of each memory address in the pair of stores are merged by Φ^{ι} in the same fashion (arithmetization, for primitive values; and wrapping by φ symbols, for non-primitive values). It is worth noting that since stores contain only primitive values, this process can never cause a memory location to contain a φ symbol.

Conversely, when values wrapped in φ symbols appear as results of evaluated subexpressions (e.g., in the rule for application of a φ symbol), the distributed semantics takes the value from each branch, uses it to evaluate the entire expression (inductively), and merges the results (again using the Φ^{ι} function). For example, if $\tilde{e}_1$ evaluates to $\varphi(\tilde{b}, \lambda x.0, \lambda x.1)$, and $\tilde{e}_2$ evaluates to 17, then the application $\tilde{e}_1\ \tilde{e}_2$ causes the subexpressions $(\lambda x.0)\ 17$ and $(\lambda x.1)\ 17$ to be evaluated (returning 0 and 1), and finally executes the merge $\Phi^{\iota}(\tilde{b}, 0, 1)$ (which, if $\tilde{b}$ is a hidden representation of `true`, will produce a hidden primitive value representing 0).

The case of assignment to a secret reference is similar. Since it is secret, the reference might evaluate to a tree of nested φ symbols (e.g., $\varphi(\tilde{b}_1, \varphi(\tilde{b}_2, a_1, a_2), a_3)$), rather than a single address. In this case, using the update operator, we perform the update recursively on the references from the left and right branches of the φ symbol, then join the resulting (updated) stores as above.

5.4 Theoretical results

Before presenting our main results, we will need a series of auxiliary definitions and lemmas. For continuity, we defer the proofs of these lemmas to Appendix A. We begin with some standard results, and proceed to note some immediate consequences of our definition of $\lambda^{\rightarrow}_{\mathrm{P,S}}$ and its semantics.

Lemma 1 (Weakening). *If* $\Gamma, \Sigma, C \vdash \tilde{e} : \tau$*,* $\Gamma' \supseteq \Gamma$*, and* $\Sigma' \supseteq \Sigma$*, then* $\Gamma', \Sigma', C \vdash \tilde{e} : \tau$*.*

For an environment ρ mapping variables to terms, we extend ρ to expressions in the usual way, defining $\rho(\tilde{e})$ to be $\tilde{e}$ with any free variable x replaced by $\rho(x)$.

Lemma 2 (Substitution). *If* $\Gamma, \Sigma, C \vdash \tilde{e} : \tau$*, and for all* $x \in \operatorname{dom}\Gamma$*, we have* $\Gamma, \Sigma, C \vdash \rho(x) : \Gamma(x)$*, then* $\emptyset, \Sigma, C \vdash \rho(\tilde{e}) : \tau$*.*

Lemma 3 (Output of Evaluation is a Value).

- *If* $(e, \mu) \downarrow (v, \mu', \mathcal{O})$*, then for all* μ_1*,* $(v, \mu_1) \downarrow (v, \mu_1, \varepsilon)$*.*
- *If* $(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}, \tilde{\mu}', \mathcal{O})$*, then for all* $\tilde{\mu}_1$*,* $(\tilde{v}, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}, \tilde{\mu}_1, \varepsilon, \varepsilon)$*.*
- $\tilde{v}$ *is an (extended) value in the syntax if and only if* $(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}, \tilde{\mu}', \mathcal{O})$ *for some* $\tilde{e}, \tilde{\mu}$*.*

Henceforth, we will assume Lemma 3 implicitly, using the term "value" to refer to any of the above definitions.

Lemma 4 (Values are Closed). *If* $\tilde{v}$ *is a value, then* $\tilde{v}$ *contains no free variables.*

Lemma 5 (Well-Typed Values are Well-Typed in Secret Contexts). *If* $\Gamma, \Sigma, C \vdash \tilde{v} : \tau$*, and* $\tilde{v}$ *is a value, then* $\Gamma, \Sigma, \mathrm{S} \vdash \tilde{v} : \tau$*.*

Lemma 6 (Context Subtyping). *If* $\Gamma, \Sigma, C \vdash \tilde{e} : \tau$*, and* $C' \sqsubseteq C$*, then* $\Gamma, \Sigma, C' \vdash \tilde{e} : \tau$*,*

In what follows, we assume Ψ is a bijective mapping between sets of memory locations. We will write $\Psi : A \leftrightarrow B$ to signify that Ψ has domain A and codomain B. We also denote by $\operatorname{Addrs}(\tilde{e})$ the set of memory locations contained by an expression $\tilde{e}$, and we extend mappings Ψ to expressions $\tilde{e}$ inductively, applying Ψ to each address contained in $\tilde{e}$, requiring that $\operatorname{Addrs}(\tilde{e}) \subseteq \operatorname{dom}\Psi$.

Lemma 7 (Extension of Permutations). *If* $\Psi' \supseteq \Psi$ *and* $\operatorname{Addrs}(\tilde{e}) \subseteq \operatorname{dom}\Psi$ *then* $\Psi(\tilde{e}) = \Psi'(\tilde{e})$*.*

Lemma 8 (Well-Typed Expressions are Address-Closed). *If* $\emptyset, \Sigma, C \vdash \tilde{e} : \tau$*, then* $\operatorname{Addrs}(\tilde{e}) \subseteq \operatorname{dom}\Sigma$*.*

Lemma 9 (Distributed Semantics Preserves Address-Closed Expressions). *If* $(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}, \tilde{\mu}', T, \mathcal{O})$*, and* $\operatorname{Addrs}(\tilde{e}) \subseteq \tilde{\mu}$*, then* $\operatorname{Addrs}(\tilde{v}) \subseteq \tilde{\mu}'$*.*

Lemma 10 (Permutation Invariance of Static Semantics). *If* $\Gamma, \Sigma, C \vdash \tilde{e} : \tau$ *then* $\Gamma, \Psi(\Sigma), C \vdash \Psi(\tilde{e}) : \tau$*.*

Lemma 11 (Permutation Invariance of Distributed Semantics). *If* $\operatorname{Addrs}(\tilde{e}) \subseteq \operatorname{dom}\tilde{\mu}$*,* $(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}, \tilde{\mu}_1, T, \mathcal{O})$*, then* $(\Psi(\tilde{e}), \Psi(\tilde{\mu}), \iota) \Downarrow (\Psi(\tilde{v}), \Psi(\tilde{\mu}_1), T, \mathcal{O})$*.*

We also require some basic properties of the join and update function, Φ^{ι} and $\operatorname{update}^{\iota}$, defined above.

Lemma 12 (Phi Takes the Union of Domains). *If* $\tilde{\mu} = \Phi^{\iota}_1(\tilde{b}, \tilde{\mu}_2, \tilde{\mu}_3)$*, then* $\operatorname{dom}\tilde{\mu} = \operatorname{dom}\tilde{\mu}_2 \cup \operatorname{dom}\tilde{\mu}_3$*.*

Lemma 13 (Store Update Preserves Domain). *If* $\tilde{\mu}' = \operatorname{update}^{\iota}_1(\tilde{\mu}, \tilde{v}, \tilde{v}')$ *and* $\operatorname{Addrs}(\tilde{v}) \subseteq \operatorname{dom}\tilde{\mu}$ *then* $\operatorname{dom}\tilde{\mu}' = \operatorname{dom}\tilde{\mu}$*.*

Lemma 14 (Substitution Commutes With Permutation). *If* $\tilde{e}_2 = \Psi(\tilde{e}_1)$ *and* $\tilde{v}_2 = \Psi(\tilde{v}_1)$*, then* $\tilde{e}_2[\tilde{v}_2/x] = \Psi(\tilde{e}_1[\tilde{v}_1/x])$*.*

Lemma 15 (Permutation Invariance of Join of Values)**.** *If* $(\tilde{v}_1)_b = \Psi((\tilde{v}_1)_a)$*,* $(\tilde{v}_2)_b = \Psi((\tilde{v}_2)_a)$*,* $(\tilde{v}_3)_b = \Psi((\tilde{v}_3)_a)$*,* $\tilde{v}'_a = \Phi^\iota_1((\tilde{v}_1)_a, (\tilde{v}_2)_a), (\tilde{v}_3)_a)$*, and* $\tilde{v}'_b = \Phi^\iota_1((\tilde{v}_1)_b, (\tilde{v}_2)_b), (\tilde{v}_3)_b)$*, then* $\tilde{v}'_b = \Psi(\tilde{v}'_a)$*.*

Lemma 16 (Permutation Invariance of Join of Stores)**.** *If* $(\tilde{v}_1)_b = \Psi((\tilde{v}_1)_a)$*,* $(\tilde{\mu}_2)_b = \Psi((\tilde{\mu}_2)_a)$*,* $(\tilde{\mu}_3)_b = \Psi((\tilde{\mu}_3)_a)$*,* $\tilde{\mu}'_a = \Phi^\iota_1((\tilde{v}_1)_a, (\tilde{\mu}_2)_a), (\tilde{\mu}_3)_a)$*, and* $\tilde{\mu}'_b = \Phi^\iota_1((\tilde{v}_1)_b, (\tilde{\mu}_2)_b), (\tilde{\mu}_3)_b)$*, then* $\tilde{\mu}'_b = \Psi(\tilde{\mu}'_a)$*.*

Lemma 17 (Permutation Invariance of Store Update)**.** *If* $\tilde{\mu}_b = \Psi(\tilde{\mu}_a)$*,* $\tilde{v}_b = \Psi(\tilde{v}_a)$*, and* $\tilde{v}'_b = \Psi(\tilde{v}'_a)$*,* $\tilde{\mu}'_a = \mathrm{update}^\iota_1(\tilde{\mu}_a, \tilde{v}_a, \tilde{v}'_a)$*, and* $\tilde{\mu}'_b = \mathrm{update}^\iota_1(\tilde{\mu}_b, \tilde{v}_b, \tilde{v}'_b)$*, then* $\tilde{\mu}'_b = \Psi(\tilde{\mu}'_a)$*.*

Lemma 18 (Permutation Invariance of Store Selection)**.** *If* $\tilde{\mu}_b = \Psi(\tilde{\mu}_a)$*,* $\tilde{v}_b = \Psi(\tilde{v}_a)$*,* $\tilde{v}'_a = \mathrm{select}^\iota_2(\tilde{\mu}_a, \tilde{v}_a)$*, and* $\tilde{v}'_b = \mathrm{select}^\iota_2(\tilde{\mu}_b, \tilde{v}_b)$*, then* $\tilde{v}'_b = \Psi(\tilde{v}'_a)$*.*

In many of the following lemmas, we will need the property that the distributed semantics is essentially deterministic: i.e., evaluations of two identical expression/store pairs (up to permutation of memory locations) produces identical results (again up to permutation).

Lemma 19 (Determinism of Distributed Semantics)**.** *If* $(\tilde{e}_a, \tilde{\mu}_a, \iota) \Downarrow (\tilde{v}'_a, \tilde{\mu}'_a, T_a, \mathcal{O}_a)$*,* $\tilde{e}_b = \Psi(\tilde{e}_a)$*,* $\tilde{\mu}_b = \Psi(\tilde{\mu}_a)$*,* $(\tilde{e}_b, \tilde{\mu}_b, \iota) \Downarrow (\tilde{v}'_b, \tilde{\mu}'_b, T_b, \mathcal{O}_b)$*, and* $\Psi : \mathrm{dom}\,\tilde{\mu}_a \leftrightarrow \mathrm{dom}\,\tilde{\mu}_b$*, then* $\mathcal{O}_a = \mathcal{O}_b$ *and for some* $\Psi' \supseteq \Psi$*,* $\Psi' : \mathrm{dom}\,\tilde{\mu}'_a \leftrightarrow \mathrm{dom}\,\tilde{\mu}'_b$*,* $\tilde{v}'_b = \Psi'(\tilde{v}'_a)$ *and* $\tilde{\mu}'_b = \Psi'(\tilde{\mu}'_a)$*.*

We will also need a series of definitions and results concerning typing of stores and their operations.

Definition 4 (Store Typing)**.** We define $\vdash \tilde{\mu} : \Sigma$ if for all $a \in \mathrm{dom}\,\Sigma$, whenever $\Sigma(a) = Y^\ell$, we have $\tilde{\mu}(a) \in \mathcal{E}_{\ell'}(Y)$.

To simplify notation, we also define a notion of well-formed store typing extension.

Definition 5 (Type Extension)**.** We define $(\tilde{\mu}', \Sigma') \succ^\iota (\tilde{\mu}, \Sigma)$ if $\vdash \tilde{\mu} : \Sigma$, $\vdash \tilde{\mu}' : \Sigma'$, and $\Sigma' \supseteq \Sigma$.

Lemma 20 (Phi Preserves Value Typing)**.** *If* $\tilde{b} \in \mathcal{E}_\ell(\mathrm{bool})$*;* $\Sigma_2|_{\mathrm{dom}\,\Sigma_3} = \Sigma_3|_{\mathrm{dom}\,\Sigma_2}$*;* $\Sigma' = \Sigma_2 \cup \Sigma_3$*;* $\tilde{v}' = \Phi^\iota_1(\tilde{b}, \tilde{v}_2, \tilde{v}_3)$*;* $\emptyset, \Sigma_2, \mathrm{S} \vdash \tilde{v}_2 : \tau$*; and* $\emptyset, \Sigma_3, \mathrm{S} \vdash \tilde{v}_3 : \tau$*, then* $\emptyset, \Sigma', \mathrm{S} \vdash \tilde{v}' : \tau$*.*

Lemma 21 (Phi Preserves Store Typing)**.** *If* $\tilde{b} \in \mathcal{E}_\ell(\mathrm{bool})$*,* $\vdash \tilde{\mu}_2 : \Sigma_2$*,* $\vdash \tilde{\mu}_3 : \Sigma_3$*,* $\Sigma_2|_{\mathrm{dom}\,\Sigma_3} = \Sigma_3|_{\mathrm{dom}\,\Sigma_2}$*, and* $\tilde{\mu}' = \Phi^\iota_1(\tilde{b}, \tilde{\mu}_2, \tilde{\mu}_3)$*, then* $\vdash \tilde{\mu}' : \Sigma_2 \cup \Sigma_3$*.*

Lemma 22 (Store Update Preserves Store Typing)**.** *If* $\emptyset, \Sigma, C \vdash \tilde{v} : (Y^\ell \ \mathrm{ref})^{\ell'}$*;* $\emptyset, \Sigma, C \vdash \tilde{v}' : Y^\ell$*;* $\ell' \sqsubseteq \ell$*;* $\vdash \tilde{\mu} : \Sigma$*; and* $\tilde{\mu}' = \mathrm{update}^\iota_1(\tilde{\mu}, \tilde{v}, \tilde{v}')$*, then* $\vdash \tilde{\mu}' : \Sigma$*.*

Lemma 23 (Store Selection Preserves Typing)**.** *If* $\emptyset, \Sigma, \mathrm{S} \vdash \tilde{v} : (Y^{\ell'} \ \mathrm{ref})^{\ell''}$*;* $\ell', \ell'' \sqsubseteq \ell$*;* $\vdash \tilde{\mu} : \Sigma$*; and* $\tilde{v}' = \mathrm{select}^\iota_1(\tilde{\mu}, \tilde{v})$*, then* $\emptyset, \Sigma, \mathrm{S} \vdash \tilde{v}' : Y^\ell$*.*

Given these results, we can now prove a standard type preservation lemma for the distributed semantics.

Lemma 24 (Type Preservation for Distributed Semantics)**.** *If* $\emptyset, \Sigma, C \vdash \tilde{e} : \tau$*,* $\vdash \tilde{\mu} : \Sigma$*, and* $(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \mathcal{O})$*, then for some* Σ'*,* $(\tilde{\mu}', \Sigma') \succ^\iota (\tilde{\mu}, \Sigma)$*, and* $\emptyset, \Sigma', C' \vdash \tilde{v}' : \tau$ *for any* C'*.*

In order to ensure that joins of stores are well-defined, we will often need stores to agree on any values in locations typed as public. Thus, we state the following definition, which expresses that one store is a "public extension" of another – i.e., each is well-typed (the latter according to a superset of the type environment of the former); and the latter differs only in locations that were not typed as public.

Definition 6 (Public Extension)**.** We define $\tilde{\mu}' \succsim^\iota_\Sigma \tilde{\mu}$ if for any address $a \in \mathrm{dom}\,\tilde{\mu}$, if $\Sigma(a) = Y^\mathrm{P}$ then $\tilde{\mu}'(a) = \tilde{\mu}(a)$.

For convenience, we also give the following notation for stores that are each well-typed and also form a public extension.

Definition 7 (Full Public Extension)**.** We define $(\tilde{\mu}', \Sigma') \succsim^{\iota} (\tilde{\mu}, \Sigma)$ if $\vdash \tilde{\mu} : \Sigma$, $\vdash \tilde{\mu}' : \Sigma'$, $\Sigma' \supseteq \Sigma$, and $\tilde{\mu}' \succsim^{\iota}_{\Sigma} \tilde{\mu}$.

We also state some facts about public extensions that follow from this definition.

Lemma 25 (Permutation Invariance of Public Extension)**.** *If* $\tilde{\mu}' \succsim^{\iota}_{\Sigma} \tilde{\mu}$, $\vdash \tilde{\mu} : \Sigma$, *and* $\Psi|_{\mathrm{dom}\,\Sigma} = \mathrm{id}$, *then* $\Psi(\tilde{\mu}') \succsim^{\iota}_{\Sigma} \tilde{\mu}$.

Lemma 26 (Permutation Invariance of Full Public Extension)**.** *If* $(\tilde{\mu}', \Sigma') \succsim^{\iota} (\tilde{\mu}, \Sigma)$ *and* $\Psi|_{\mathrm{dom}\,\Sigma} = \mathrm{id}$, *then* $(\Psi(\tilde{\mu}'), \Psi(\Sigma')) \succsim^{\iota} (\tilde{\mu}, \Sigma)$.

Lemma 27 (Phi is a Public Extension)**.** *If* $\tilde{b} \in \mathcal{E}_{\ell}(\mathrm{bool})$, $(\tilde{\mu}_2, \Sigma_2) \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1)$, $(\tilde{\mu}_3, \Sigma_3) \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1)$, $\mathrm{dom}\,\Sigma_2 \cap \mathrm{dom}\,\Sigma_3 \subseteq \mathrm{dom}\,\Sigma_1$, *then* $(\Phi^{\iota}_1(\tilde{b}, \tilde{\mu}_2, \tilde{\mu}_3), \Sigma_2 \cup \Sigma_3) \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1)$.

Now, we give the key definitions for one of the main lemmas: Lemma 35, *conditional purity* for the distributed semantics, which states that if an expression is well-typed in a secret context, then its evaluation terminates, yields no observations, and produces no publicly-observable effects on the mutable store. We will prove this result using the method of reducibility; we now define the corresponding reducibility predicate (Figure 11, notated as $\tilde{v} \in \mathcal{V}(\tau, \Sigma)$ (the value $\tilde{v}$ is reducible at type τ, under store typing context Σ).

Figure 11 Reducibility predicate for distributed semantics.

$$\frac{\begin{array}{c}\forall \tilde{v}_1 \in \mathcal{V}(\tau_1, \Sigma_1)\,.\,\Sigma_1 \supseteq \Sigma \wedge \vdash \tilde{\mu}_1 : \Sigma_1 \implies \\ (\tilde{e}[\tilde{v}_1/x], \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \varepsilon) \wedge \tilde{v}' \in \mathcal{V}(\tau, \Sigma') \wedge (\tilde{\mu}', \Sigma') \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1)\end{array}}{\lambda x.\tilde{e} \in \mathcal{V}((\tau_1 \xrightarrow{\mathrm{S}} \tau)^{\ell}, \Sigma)}$$

$$\frac{\ell' \sqsubseteq \ell \qquad \tilde{y} \in \mathcal{E}^{\iota}_{\ell'}(Y)}{\tilde{y} \in \mathcal{V}(Y^{\ell}, \Sigma)} \qquad \frac{\Sigma(a) = Y^{\ell}}{a \in \mathcal{V}((Y^{\ell}\ \mathrm{ref})^{\ell'}, \Sigma)}$$

$$\frac{\tilde{b} \in \mathcal{E}_{\ell_1}(\mathrm{bool}) \qquad \tilde{v}_2, \tilde{v}_3 \in \mathcal{V}((\tau\ \mathrm{ref})^{\ell}, \Sigma)}{\varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3) \in \mathcal{V}((\tau\ \mathrm{ref})^{\ell}, \Sigma)} \qquad \frac{\tilde{b} \in \mathcal{E}_{\ell_1}(\mathrm{bool}) \qquad \tilde{v}_2, \tilde{v}_3 \in \mathcal{V}((\tau_1 \xrightarrow{\mathrm{S}} \tau)^{\ell}, \Sigma)}{\varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3) \in \mathcal{V}((\tau_1 \xrightarrow{\mathrm{S}} \tau)^{\ell}, \Sigma)}$$

$$\frac{\begin{array}{c}\tilde{b} \in \mathcal{E}_{\ell_1}(\mathrm{bool}) \qquad \forall \tilde{v}_1 \in \mathcal{V}(\tau_1, \Sigma_1)\,.\,\Sigma_1 \supseteq \Sigma \wedge \vdash \tilde{\mu}_1 : \Sigma_1 \implies \\ (\tilde{v}_2\ \tilde{v}_1, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}'_2, \tilde{\mu}'_2, T, \varepsilon) \wedge \tilde{v}'_2 \in \mathcal{V}(\tau, \Sigma'_2) \wedge (\tilde{\mu}'_2, \Sigma'_2) \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1) \wedge \\ (\tilde{v}_3\ \tilde{v}_1, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}'_3, \tilde{\mu}'_3, T, \varepsilon) \wedge \tilde{v}'_3 \in \mathcal{V}(\tau, \Sigma'_3) \wedge (\tilde{\mu}'_3, \Sigma'_3) \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1)\end{array}}{\varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3) \in \mathcal{V}((\tau_1 \xrightarrow{\mathrm{S}} \tau)^{\ell}, \Sigma)} \qquad \frac{(-, -, \iota) \Downarrow (\tilde{v}, -, -, -)}{\tilde{v} \in \mathcal{V}((\tau_1 \xrightarrow{\mathrm{P}} \tau)^{-}, \Sigma)}$$

We note that we make no requirement of values reducible at public arrow types (i.e., types with publicly-observable effects), as they can never be applied in a secret context (and thus Lemma 35 is not concerned with their execution). Before proceeding, we also note a few facts about the reducibility predicate:

Lemma 28 (Reducible Values are Values)**.** *If* $\tilde{v} \in \mathcal{V}(\tau, \Sigma)$, *then* $\tilde{v}$ *is a value.*

Lemma 29 (Permutation Invariance of Reducibility Predicate)**.** *If* $\tilde{v} \in \mathcal{V}(\tau, \Sigma)$, *then* $\Psi(\tilde{v}) \in \mathcal{V}(\tau, \Psi(\Sigma))$.

Lemma 30 (Store Weakening for Reducibility Predicate)**.** *If* $\tilde{v} \in \mathcal{V}(\tau, \Sigma)$, *and* $\Sigma' \supseteq \Sigma$, *then* $\tilde{v} \in \mathcal{V}(\tau, \Sigma')$.

Lemma 31 (Phi of Reducible Values is Reducible)**.** *If* $\tilde{b} \in \mathcal{E}_{\ell}(\mathrm{bool})$, $\tilde{v}_2 \in \mathcal{V}(\tau, \Sigma_2)$, $\tilde{v}_3 \in \mathcal{V}(\tau, \Sigma_3)$, *and* $\Sigma_{2,3} \supseteq \Sigma_2, \Sigma_3$, *then* $\Phi^{\iota}_1(\tilde{b}, \tilde{v}_2, \tilde{v}_3) \in \mathcal{V}(\tau, \Sigma_{2,3})$.

Lemma 32 (Update to Reducible Secret Reference is a Public Extension)**.** *If* $\tilde{v} \in \mathcal{V}((Y^{\mathrm{S}}\ \mathrm{ref})^{\ell'}, \Sigma)$, $\vdash \tilde{\mu} : \Sigma$, *and* $\tilde{v}' \in \mathcal{E}_{\ell}(Y)$, *then* $(\mathrm{update}^{\iota}_1(\tilde{\mu}, \tilde{v}, \tilde{v}'), \Sigma) \succsim^{\iota} (\tilde{\mu}, \Sigma)$.

Lemma 33 (Select on Reducible Secret Reference is Reducible)**.** *If* $\tilde{v} \in \mathcal{V}((Y^{\ell'})^{\ell''}\ \mathrm{ref}, \Sigma)$*,* $\vdash \tilde{\mu} : \Sigma$*,* $\ell' \sqsubseteq \ell$*, and* $\ell'' \sqsubseteq \ell$*, then* $\mathrm{select}_1^\iota(\tilde{\mu}, \tilde{v}) \in \mathcal{V}(Y^\ell, \Sigma)$*.*

For convenience, we now prove a generalization of the application case of conditional purity (Lemma 35) as a separate lemma. Ordinarily, this case would be expressed as an inner induction on the structure of an expression with nested phi symbols. However, we separate it from Lemma 35 both for clarity and because it is relevant to later proofs.

Lemma 34 (Purity for Applications of Pure Values)**.** *If* $\tilde{v} \in \mathcal{V}(\tau_1 \xrightarrow{\mathrm{S}} \tau, \Sigma_0)$*,* $(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}, \tilde{\mu}_0, T_0, \mathcal{O}_0)$*,* $(\tilde{e}_1, \tilde{\mu}_0, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T_1, \mathcal{O}_1)$*,* $\tilde{v}_1 \in \mathcal{V}(\tau_1, \Sigma_1)$*, and* $(\tilde{\mu}_1, \Sigma_1) \succ^\iota (\tilde{\mu}_0, \Sigma_0) \succ^\iota (\tilde{\mu}, \Sigma)$*, then* $(\tilde{e}\ \tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T', \mathcal{O}_0 \| \mathcal{O}_1)$*,* $(\tilde{\mu}', \Sigma') \precsim^\iota (\tilde{\mu}_1, \Sigma_1)$*, and* $\tilde{v}' \in \mathcal{V}(\tau, \Sigma')$*.*

We can now prove conditional purity of the distributed semantics. The proof proceeds via a reducibility argument, as discussed above, and makes extensive use of the control flow restrictions of the static semantics.

Lemma 35 (Conditional Purity for Distributed Semantics)**.** *If* $\Gamma, \Sigma, \mathrm{S} \vdash \tilde{e} : \tau$*, and for all* $x \in \mathrm{dom}\,\Gamma$*,* $\rho(x) \in \mathcal{V}(\Gamma(x), \Sigma)$ *and* $\mathrm{Addrs}(\rho(x)) \subseteq \mathrm{dom}\,\Sigma$*, then for some* $\tilde{v}$*,* T*, and* $(\tilde{\mu}', \Sigma') \precsim^\iota (\tilde{\mu}, \Sigma)$*,* $(\rho(\tilde{e}), \tilde{\mu}, \iota) \Downarrow (\tilde{v}, \tilde{\mu}', T, \varepsilon)$ *and* $\tilde{v} \in \mathcal{V}(\tau, \Sigma')$*.*

Having established Lemma 35, we are now ready to approach one of our two main results: correctness of the distributed semantics. In particular, we will be able to show that any well-typed source program, when it evaluates according to the reference semantics, will evaluate "similarly" according to the distributed semantics. Before proving this result, we establish what is meant by similarity of expressions (Figure 12) and similarity of stores (Definition 8), and note a few consequences of these definitions.

Figure 12 Similar expressions.

$$\frac{\tilde{y} \in \hat{\mathcal{E}}_\ell^\iota(y)}{y \sim_\Psi^\iota \tilde{y}} \qquad \frac{}{a \sim_\Psi^\iota \Psi(a)} \qquad \frac{}{x \sim_\Psi^\iota x} \qquad \frac{e \sim_\Psi^\iota \tilde{e}}{\lambda x.e \sim_\Psi^\iota \lambda x.\tilde{e}} \qquad \frac{e \sim_\Psi^\iota \tilde{e}}{\mathtt{fix}\ f.\lambda x.\tilde{e} \sim_\Psi^\iota \mathtt{fix}\ f.\lambda x.\tilde{e}}$$

$$\frac{e_0 \sim_\Psi^\iota \tilde{e}_0 \qquad e_1 \sim_\Psi^\iota \tilde{e}_1}{e_0\ e_1 \sim_\Psi^\iota \tilde{e}_0\ \tilde{e}_1} \qquad \frac{e \sim_\Psi^\iota \tilde{v}_2 \qquad \tilde{b} \in \hat{\mathcal{E}}_\ell^\iota(\mathtt{true})}{e \sim_\Psi^\iota \varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3)} \qquad \frac{e \sim_\Psi^\iota \tilde{v}_3 \qquad \tilde{b} \in \hat{\mathcal{E}}_\ell^\iota(\mathtt{false})}{e \sim_\Psi^\iota \varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3)}$$

$$\frac{e_1 \sim_\Psi^\iota \tilde{e}_1 \qquad e_2 \sim_\Psi^\iota \tilde{e}_2 \qquad e_3 \sim_\Psi^\iota \tilde{e}_3}{\mathtt{if}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3 \sim_\Psi^\iota \mathtt{if}\ \tilde{e}_1\ \mathtt{then}\ \tilde{e}_2\ \mathtt{else}\ \tilde{e}_3} \qquad \frac{\forall j \in \{1, \ldots, r\}\,.\, e_j \sim_\Psi^\iota \tilde{e}_j}{\mathtt{op}_i^t(e_1, \ldots, e_r) \sim_\Psi^\iota \mathtt{op}_i^t(\tilde{e}_1, \ldots, \tilde{e}_r)}$$

$$\frac{e_1 \sim_\Psi^\iota \tilde{e}_1}{\mathtt{ref}\ e_1 \sim_\Psi^\iota \mathtt{ref}\ \tilde{e}_1} \qquad \frac{e_1 \sim_\Psi^\iota \tilde{e}_1}{!e_1 \sim_\Psi^\iota !\tilde{e}_1} \qquad \frac{e_1 \sim_\Psi^\iota \tilde{e}_1 \qquad e_2 \sim_\Psi^\iota \tilde{e}_2}{e_1 := e_2 \sim_\Psi^\iota \tilde{e}_1 := \tilde{e}_2} \qquad \frac{e_1 \sim_\Psi^\iota \tilde{e}_1}{\mathtt{reveal}\ e_1 \sim_\Psi^\iota \mathtt{reveal}\ \tilde{e}_1}$$

Definition 8 (Similarity of Stores)**.** We define $\mu \sim_\Psi^\iota \tilde{\mu}$ if for all $a \in \mathrm{dom}\,\Psi$, $\mu(a) \sim_\Psi^\iota \tilde{\mu}(\Psi(a))$.

Lemma 36 (Similarity is Reflexive for Surface Expressions)**.** *For any expression* e *(in the surface language),* $(e, \{\}) \sim_\emptyset^\iota (e, \{\})$*.*

Lemma 37 (Permutation Weakening for Similarity Relation)**.**

- *If* $e \sim_\Psi^\iota \tilde{e}$ *and* $\Psi' \supseteq \Psi$ *then* $e \sim_{\Psi'}^\iota \tilde{e}$*.*
- *If* $\mu \sim_\Psi^\iota \tilde{\mu}$ *and* $\Psi' \supseteq \Psi$ *then* $\mu \sim_{\Psi'}^\iota \tilde{\mu}$*.*

Lemma 38 (Substitution for Similarity Relation)**.** *If* $e \sim_\Psi^\iota \tilde{e}$ *and* $v \sim_\Psi^\iota \tilde{v}$*, then* $e[v/x] \sim_\Psi^\iota \tilde{e}[\tilde{v}/x]$*.*

Lemma 39 (Join of Values Preserves Equivalence)**.**

- *If* $\tilde{b} \in \hat{\mathcal{E}}^{\iota}_{\ell}(\texttt{true})$ *and* $v \sim^{\iota}_{\Psi} \tilde{v}_2$*, then* $v \sim^{\iota}_{\Psi} \Phi^{\iota}_1(\tilde{b}, \tilde{v}_2, \tilde{v}_3)$*.*
- *If* $\tilde{b} \in \hat{\mathcal{E}}^{\iota}_{\ell}(\texttt{false})$ *and* $v \sim^{\iota}_{\Psi} \tilde{v}_3$*, then* $v \sim^{\iota}_{\Psi} \Phi^{\iota}_1(\tilde{b}, \tilde{v}_2, \tilde{v}_3)$*.*

Lemma 40 (Join of Stores Preserves Equivalence)**.**

- *If* $\mu \sim^{\iota}_{\Psi} \tilde{\mu}_2$*,* $\tilde{\mu}' = \Phi^{\iota}_1(\tilde{b}, \tilde{\mu}_2, \tilde{\mu}_3)$*, and* $\tilde{b} \in \mathcal{E}_{\ell_1}(\texttt{true})$*, then* $\mu \sim^{\iota}_{\Psi} \tilde{\mu}'$*.*
- *If* $\mu \sim^{\iota}_{\Psi} \tilde{\mu}_3$*,* $\tilde{\mu}' = \Phi^{\iota}_1(\tilde{b}, \tilde{\mu}_2, \tilde{\mu}_3)$*, and* $\tilde{b} \in \mathcal{E}_{\ell_1}(\texttt{false})$*, then* $\mu \sim^{\iota}_{\Psi} \tilde{\mu}'$*.*

We now proceed to introduce the main correctness result (Theorem 1, generalized by Lemma 44). As with Lemma 35, we first prove separate lemmas that generalize a few of the cases, rather than relying on inner inductions.

Lemma 41 (Correctness for Applications of Related Values)**.** *If* $\lambda x.e'_0 \sim^{\iota}_{\Psi_0} \tilde{v}_0$*, then for all* $\tilde{e}_0$ *and* $\tilde{e}_1$*, if the following conditions are satisfied:*

- $(\tilde{e}_0, \tilde{\mu}, \iota) \Downarrow (\tilde{v}_0, \tilde{\mu}_0, T_0, \mathcal{O}_0)$
- $(\tilde{e}_1, \tilde{\mu}_0, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T_1, \mathcal{O}_1)$
- $\emptyset, \Sigma_0, C \vdash \tilde{v}_0 : (\tau_1 \xrightarrow{C} \tau)^{\ell}$
- $\emptyset, \Sigma_1, C \vdash \tilde{v}_1 : \tau_1$
- $v_1 \sim^{\iota}_{\Psi_1} \tilde{v}_1$
- *For any* $\tilde{e}$*, if* $e'_0[v_1/x] \sim^{\iota}_{\Psi_1} \tilde{e}$ *and* $\emptyset, \Sigma_1, C \vdash \tilde{e} : \tau$*, then* $(\tilde{e}, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \mathcal{O}_2)$*,* $v' \sim^{\iota}_{\Psi'} \tilde{v}'$*,* $\mu' \sim^{\iota}_{\Psi'} \tilde{\mu}'$*, and* $\Psi' \supseteq \Psi_1$*.*

then $(\tilde{e}_0\ \tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T', \mathcal{O}_0 \| \mathcal{O}_1 \| \mathcal{O}_2)$*,* $v' \sim^{\iota}_{\Psi'} \tilde{v}'$*,* $\mu' \sim^{\iota}_{\Psi'} \tilde{\mu}'$*,* $\Psi' \supseteq \Psi_1$*; and whenever* $\emptyset, \Sigma_0, C \vdash \tilde{v}_0 : (\tau_1 \xrightarrow{\mathrm{S}} \tau)^{\ell}$*,* $\mathcal{O}_2 = \varepsilon$ *and* $\tilde{\mu}' \succsim^{\iota}_{\Sigma_1} \tilde{\mu}_1$*.*

Lemma 42 (Correctness of Distributed Store Update)**.** *If* $a \sim^{\iota}_{\Psi} \tilde{v}$*,* $\mu \sim^{\iota}_{\Psi} \tilde{\mu}$*,* $v' \in Y$*,* $\tilde{v}' \in \hat{\mathcal{E}}^{\iota}_{\ell}(v')$*, and* $\tilde{\mu}' = \mathrm{update}^{\iota}_1(\tilde{\mu}, \tilde{v}, \tilde{v}')$*, then* $\mu[a \mapsto v'] \sim^{\iota}_{\Psi} \tilde{\mu}'$*.*

Lemma 43 (Correctness of Distributed Store Select)**.** *If* $a \sim^{\iota}_{\Psi} \tilde{v}$*,* $\mu \sim^{\iota}_{\Psi} \tilde{\mu}$*, and* $\tilde{v}' = \mathrm{select}^{\iota}_1(\tilde{\mu}, \tilde{v})$*, then* $\mu(a) \sim^{\iota}_{\Psi} \tilde{v}'$*.*

Lemma 44 (Correctness (Generalized))**.** *If* $(e, \mu) \downarrow (v', \mu', \mathcal{O})$*,* $(e, \mu) \sim^{\iota}_{\Psi} (\tilde{e}, \tilde{\mu})$*,* $\vdash \tilde{\mu} : \Sigma$*, and* $\emptyset, \Sigma, C \vdash \tilde{e} : \tau$*, then* $(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \mathcal{O})$*,* $(v', \mu') \sim^{\iota}_{\Psi'} (\tilde{v}', \tilde{\mu}')$*, and* $\Psi' \supseteq \Psi$*.*

Finally, we turn to the question of security (Theorem 2). For this, we define a notion of "publicly-equivalent" expressions, or expressions "equivalent to a public observer" (i.e., two expressions that are structurally equal, except possibly differing in corresponding secret values of the same type). We define this relation formally in Figure 13, and proceed to show a few consequences of our definition.

Lemma 45 (Public Similarity is Reflexive for Surface Expressions)**.** *For any expression* e *(in the surface language),* $(e, \{\}) \approx^{\iota}_{\emptyset} (e, \{\})$*.*

Lemma 46 (Permutation Weakening for Public Similarity Relation)**.**

- *If* $\tilde{e}_1 \approx^{\iota}_{\Psi} \tilde{e}_2$ *and* $\Psi' \supseteq \Psi$ *then* $\tilde{e}_1 \approx^{\iota}_{\Psi'} \tilde{e}_2$*.*

Figure 13 Publicly equivalent expressions ("equivalent to a public observer").

$$\frac{}{y \approx^{\iota}_{\Psi} y} \qquad \frac{\tilde{y}, \tilde{y}' \in \hat{\mathcal{E}}^{\iota}_{\mathrm{S}}(Y)}{\tilde{y} \approx^{\iota}_{\Psi} \tilde{y}'} \qquad \frac{}{a \approx^{\iota}_{\Psi} \Psi(a)} \qquad \frac{}{x \approx^{\iota}_{\Psi} x} \qquad \frac{\tilde{e} \approx^{\iota}_{\Psi} \tilde{e}'}{\lambda x.\tilde{e} \approx^{\iota}_{\Psi} \lambda x.\tilde{e}'}$$

$$\frac{\tilde{e} \approx^{\iota}_{\Psi} \tilde{e}'}{\texttt{fix}\ f.\lambda x.\tilde{e} \approx^{\iota}_{\Psi} \texttt{fix}\ f.\lambda x.\tilde{e}'} \qquad \frac{\tilde{v}_1 \approx^{\iota}_{\Psi} \tilde{v}_1 \qquad \tilde{v}_2 \approx^{\iota}_{\Psi} \tilde{v}_2 \qquad \tilde{v}_3 \approx^{\iota}_{\Psi} \tilde{v}_3}{\varphi(\tilde{v}_1, \tilde{v}_2, \tilde{v}_3) \approx^{\iota}_{\Psi} \varphi(\tilde{v}'_1, \tilde{v}'_2, \tilde{v}'_3)} \qquad \frac{\tilde{e}_1 \approx^{\iota}_{\Psi} \tilde{e}'_1 \qquad \tilde{e}_2 \approx^{\iota}_{\Psi} \tilde{e}'_2}{\tilde{e}_1\ \tilde{e}_2 \approx^{\iota}_{\Psi} \tilde{e}'_1\ \tilde{e}'_2}$$

$$\frac{\tilde{e}_1 \approx^{\iota}_{\Psi} \tilde{e}'_1 \qquad \tilde{e}_2 \approx^{\iota}_{\Psi} \tilde{e}'_2 \qquad \tilde{e}_3 \approx^{\iota}_{\Psi} \tilde{e}'_3}{\texttt{if}\ \tilde{e}_1\ \texttt{then}\ \tilde{e}_2\ \texttt{else}\ \tilde{e}_3 \approx^{\iota}_{\Psi} \texttt{if}\ \tilde{e}'_1\ \texttt{then}\ \tilde{e}'_2\ \texttt{else}\ \tilde{e}'_3} \qquad \frac{\forall j \in \{1, \ldots, r\}\,.\, \tilde{e}_j \approx^{\iota}_{\Psi} \tilde{e}'_j}{\mathtt{op}^t_i(\tilde{e}_1, \ldots, \tilde{e}_r) \approx^{\iota}_{\Psi} \mathtt{op}^t_i(\tilde{e}'_1, \ldots, \tilde{e}'_r)}$$

$$\frac{\tilde{e} \approx^{\iota}_{\Psi} \tilde{e}'}{\texttt{ref}\ \tilde{e} \approx^{\iota}_{\Psi} \texttt{ref}\ \tilde{e}'} \qquad \frac{\tilde{e} \approx^{\iota}_{\Psi} \tilde{e}'}{!\tilde{e} \approx^{\iota}_{\Psi} !\tilde{e}'} \qquad \frac{\tilde{e}_1 \approx^{\iota}_{\Psi} \tilde{e}'_1 \qquad \tilde{e}_2 \approx^{\iota}_{\Psi} \tilde{e}'_2}{\tilde{e}_1 := \tilde{e}_2 \approx^{\iota}_{\Psi} \tilde{e}'_1 := \tilde{e}'_2} \qquad \frac{\tilde{e}_1 \approx^{\iota}_{\Psi} \tilde{e}'}{\texttt{reveal}\ \tilde{e} \approx^{\iota}_{\Psi} \texttt{reveal}\ \tilde{e}'}$$

- *If* $\tilde{\mu}_1 \approx^{\iota}_{\Psi} \tilde{\mu}_2$ *and* $\Psi' \supseteq \Psi$ *then* $\tilde{\mu}_1 \approx^{\iota}_{\Psi'} \tilde{\mu}_2$.

Lemma 47 (Substitution for Public Similarity Relation)**.** *If* $\tilde{e}_1 \approx^{\iota}_{\Psi} \tilde{e}_2$ *and* $\tilde{v}_1 \approx^{\iota}_{\Psi} \tilde{v}_2$*, then* $\tilde{e}_1[\tilde{v}_1/x] \approx^{\iota}_{\Psi} \tilde{e}_2[\tilde{v}_2/x]$.

Lemma 48 (Public Similarity Relation Contains All Addresses)**.** *If* $\tilde{e}_1 \approx^{\iota}_{\Psi} \tilde{e}_2$*, then* $\mathrm{Addrs}(\tilde{e}_1) \subseteq \mathrm{dom}\,\Psi$ *and* $\mathrm{Addrs}(\tilde{e}_2) \subseteq \mathrm{cod}\,\Psi$.

Lemma 49 (Phi Preserves Public Equivalence of Values)**.** *If* $(\tilde{v}_1)_a \approx^{\iota}_{\Psi} (\tilde{v}_1)_b$*,* $(\tilde{v}_2)_a \approx^{\iota}_{\Psi} (\tilde{v}_2)_b$*,* $(\tilde{v}_3)_a \approx^{\iota}_{\Psi} (\tilde{v}_3)_b$*,* $\tilde{v}'_a = \Phi^{\iota}_1((\tilde{v}_1)_a, (\tilde{v}_2)_a), (\tilde{v}_3)_a)$*, and* $\tilde{v}'_b = \Phi^{\iota}_1((\tilde{v}_1)_b, (\tilde{v}_2)_b), (\tilde{v}_3)_b)$*, then* $\tilde{v}'_a \approx^{\iota}_{\Psi} \tilde{v}'_b$.

Lemma 50 (Phi Preserves Public Equivalence of Stores)**.** *If* $(\tilde{v}_1)_a \approx^{\iota}_{\Psi} (\tilde{v}_1)_b$*,* $(\tilde{\mu}_2)_a \approx^{\iota}_{\Psi} (\tilde{\mu}_2)_b$*,* $(\tilde{\mu}_3)_a \approx^{\iota}_{\Psi} (\tilde{\mu}_3)_b$*,* $\tilde{\mu}'_a = \Phi^{\iota}_1((\tilde{v}_1)_a, (\tilde{\mu}_2)_a), (\tilde{\mu}_3)_a)$*, and* $\tilde{\mu}'_b = \Phi^{\iota}_1((\tilde{v}_1)_b, (\tilde{\mu}_2)_b), (\tilde{\mu}_3)_b)$*, then* $\tilde{\mu}'_a \approx^{\iota}_{\Psi} \tilde{\mu}'_b$.

Lemma 51 (Store Update Preserves Public Equivalence)**.** *If* $\tilde{\mu}_a \approx^{\iota}_{\Psi} \tilde{\mu}_b$*,* $\tilde{v}_a \approx^{\iota}_{\Psi} \tilde{v}_b$*,* $\tilde{v}'_a \approx^{\iota}_{\Psi} \tilde{v}'_b$*,* $\tilde{\mu}'_a = \mathrm{update}^{\iota}_1(\tilde{\mu}_a, \tilde{v}_a, \tilde{v}'_a)$*, and* $\tilde{\mu}'_b = \mathrm{update}^{\iota}_1(\tilde{\mu}_b, \tilde{v}_b, \tilde{v}'_b)$*, then* $\tilde{\mu}'_a \approx^{\iota}_{\Psi} \tilde{\mu}'_b$.

Lemma 52 (Store Selection Preserves Public Equivalence)**.** *If* $\tilde{\mu}_a \approx^{\iota}_{\Psi} \tilde{\mu}_b$*, and* $\tilde{v}_a \approx^{\iota}_{\Psi} \tilde{v}_b$*,* $\tilde{v}'_a = \mathrm{select}^{\iota}_2(\tilde{\mu}_a, \tilde{v}_a)$*, and* $\tilde{v}'_b = \mathrm{select}^{\iota}_2(\tilde{\mu}_b, \tilde{v}_b)$*, then* $\tilde{v}'_a \approx^{\iota}_{\Psi} \tilde{v}'_b$.

Lemma 53 (Safety of Phi of Values)**.** *If* $(\tilde{v}_1)_a \approx^{\iota}_{\Psi} (\tilde{v}_1)_b$*;* $(\tilde{v}_2)_a \approx^{\iota}_{\Psi} (\tilde{v}_2)_b$*;* $(\tilde{v}_3)_a \approx^{\iota}_{\Psi} (\tilde{v}_3)_b$*;* $T_a = \Phi^{\iota}_2((\tilde{v}_1)_a, (\tilde{v}_2)_a), (\tilde{v}_3)_a)$*; and* $T_b = \Phi^{\iota}_2((\tilde{v}_1)_b, (\tilde{v}_2)_b), (\tilde{v}_3)_b)$*, then* $\textsc{Safe}(\iota, T_a, T_b)$.

Lemma 54 (Safety of Phi of Stores)**.** *If* $(\tilde{v}_1)_a \approx^{\iota}_{\Psi} (\tilde{v}_1)_b$*;* $(\tilde{\mu}_2)_a \approx^{\iota}_{\Psi} (\tilde{\mu}_2)_b$*;* $(\tilde{\mu}_3)_a \approx^{\iota}_{\Psi} (\tilde{\mu}_3)_b$*;* $T_a = \Phi^{\iota}_2((\tilde{v}_1)_a, (\tilde{\mu}_2)_a), (\tilde{\mu}_3)_a)$*; and* $T_b = \Phi^{\iota}_2((\tilde{v}_1)_b, (\tilde{\mu}_2)_b), (\tilde{\mu}_3)_b)$*, then* $\textsc{Safe}(\iota, T_a, T_b)$.

Lemma 55 (Safety of Store Update)**.** *If* $\tilde{\mu}_a \approx^{\iota}_{\Psi} \tilde{\mu}_b$*;* $\tilde{v}_a \approx^{\iota}_{\Psi} \tilde{v}_b$*;* $\tilde{v}'_a \approx^{\iota}_{\Psi} \tilde{v}'_b$*;* $T_a = \mathrm{update}^{\iota}_2(\tilde{\mu}_a, \tilde{v}_a, \tilde{v}'_a)$*; and* $T_b = \mathrm{update}^{\iota}_2(\tilde{\mu}_b, \tilde{v}_b, \tilde{v}'_b)$*, then* $\textsc{Safe}(\iota, T_a, T_b)$.

Lemma 56 (Safety of Store Selection)**.** *If* $\tilde{\mu}_a \approx^{\iota}_{\Psi} \tilde{\mu}_b$*;* $\tilde{v}_a \approx^{\iota}_{\Psi} \tilde{v}_b$*;* $T_a = \mathrm{select}^{\iota}_2(\tilde{\mu}_a, \tilde{v}_a)$*; and* $T_b = \mathrm{select}^{\iota}_2(\tilde{\mu}_b, \tilde{v}_b)$*, then* $\textsc{Safe}(\iota, T_a, T_b)$.

We now present the main lemma required for the security theorem: *safety of traces*. It states that if two expression/store pairs are publicly-equivalent, and evaluate to two corresponding traces and observation sequences, then neither observation sequence is a proper prefix of the other; and, if the observation sequences are identical, then the resulting values are equivalent, and the pair of traces is "safe" (in the sense of the secure execution platform).

Lemma 57 (Safety of Traces)**.** *If* $(\tilde{e}_a, \tilde{\mu}_a, \iota) \Downarrow (\tilde{v}'_a, \tilde{\mu}'_a, T_a, \mathcal{O}_a)$*,* $(\tilde{e}_b, \tilde{\mu}_b, \iota) \Downarrow (\tilde{v}'_b, \tilde{\mu}'_b, T_b, \mathcal{O}_b)$*,* $(\tilde{e}_a, \tilde{\mu}_a) \approx^{\iota}_{\Psi} (\tilde{e}_b, \tilde{\mu}_b)$*, and* $\Psi : \operatorname{dom} \tilde{\mu}_a \leftrightarrow \operatorname{dom} \tilde{\mu}_b$*, then neither of* $\mathcal{O}_a$ *and* $\mathcal{O}_b$ *is a proper prefix of the other; and, if* $\mathcal{O}_a = \mathcal{O}_b$*, then for some* $\Psi' \supseteq \Psi$*,* $(\tilde{v}'_a, \tilde{\mu}'_a) \approx^{\iota}_{\Psi'} (\tilde{v}'_b, \tilde{\mu}'_b)$*,* $\Psi' : \operatorname{dom} \tilde{\mu}'_a \leftrightarrow \operatorname{dom} \tilde{\mu}'_b$*, and* $\textsc{Safe}(\iota, T_a, T_b)$.

Finally, we present the two main results of our system: theorems that guarantee the correctness and security of an execution.

Theorem 1 (Correctness)**.** *For any program* p *and initial input value environment* κ*, if* $\vdash p : \tau$ *and* $(\kappa, p) \downarrow (y, \mu', \mathcal{O})$ *for some* $y \in Y$*, then for some mutable store* $\tilde{\mu}'$ *and communication trace* T*,* $(\kappa, p) \Downarrow (\tilde{y}, \tilde{\mu}', T, \mathcal{O})$*, with* $\tilde{y} \in \hat{\mathcal{E}}^{\iota}_{\ell}(y)$ *for some* ℓ.

Proof. Follows immediately from Lemmas 36 and 44. □

The correctness theorem expresses that any well-typed source program, when it evaluates according to the reference semantics to some primitive value, will also evaluate according to the distributed semantics to the same primitive value (possibly hidden), yielding identical observations. Since the reference semantics expresses the standard meaning of programs in $\lambda^{\rightarrow}_{\mathrm{P,S}}$, this theorem guarantees that the distributed semantics give a correct implementation for any well-typed source program.

Theorem 2 (Security)**.** *If* $(\kappa_a, p) \Downarrow (\tilde{v}'_a, \tilde{\mu}'_a, T_a, \mathcal{O})$ *and* $(\kappa_b, p) \Downarrow (\tilde{v}'_b, \tilde{\mu}'_b, T_b, \mathcal{O})$*, then for all valid adversarial views* $A \in \mathcal{A}$*, the distributions* $\Pi_A(T_a)$ *and* $\Pi_A(T_b)$ *are indistinguishable*[5].

Proof. Let $p = \texttt{read}(X_1 : Y_1, \ldots, X_r : Y_r); e$. Then by definition, we have:

- $\iota = \mathrm{Init}()$
- $T_0 = \{(C, S_i, \Pi_{\{S_i\}}(\iota)) : 1 \leq i \leq N\}$
- $\forall j \in \{1, \ldots, r\}.((\tilde{v}_j)_a, (T'_j)_a) = \mathrm{Enc}_{\mathrm{S}}(\kappa_a(X_j), \iota)$
- $\forall j \in \{1, \ldots, r\}.((\tilde{v}_j)_b, (T'_j)_b) = \mathrm{Enc}_{\mathrm{S}}(\kappa_b(X_j), \iota)$
- $(e[(\tilde{v}_1)_a/X_1 \ldots (\tilde{v}_r)_a/X_r], \{\}, \iota) \Downarrow (\tilde{v}'_a, \tilde{\mu}'_a, T'_a, \mathcal{O})$
- $(e[(\tilde{v}_1)_b/X_1 \ldots (\tilde{v}_r)_b/X_r], \{\}, \iota) \Downarrow (\tilde{v}'_b, \tilde{\mu}'_b, T'_b, \mathcal{O})$
- $T_a = T_0 \| (T'_1)_a \| \ldots \| (T'_r)_a \| T'_a$
- $T_b = T_0 \| (T'_1)_b \| \ldots \| (T'_r)_b \| T'_b$

Now, we note that $\Pi_A(T_0) = \Pi_A(\iota)$; by definition, we have $\textsc{SafeEnc}(\iota, (T'_j)_a, (T'_j)_b)$; and by Lemmas 45 and 57, we have $\textsc{Safe}(\iota, T'_a, T'_b)$. This precisely satisfies the indistinguishability assumption of the secure execution platform. □

The security theorem expresses that if two executions of the same program, each with its own secret initial inputs, yield the same observations, then their communication traces (as distributions) are indistinguishable to any adversary that receives only the information available to servers in A. In other words, the theorem guarantees that any suitably constrained adversary learns nothing about the initial secret client values that was not already logically entailed by the observations from `reveal`. It is impossible for the programmer to unintentionally leak information due to the distributed implementation.

[5] In the sense specified by the secure execution platform in Section 4.

5.5 Examples of secure execution platforms

Since we have defined our language in terms of the assumptions of a secure execution platform, it is also important to note that the examples we describe (Shamir secret sharing and fully homomorphic encryption) indeed satisfy these assumptions. We now give an overview of the arguments that establish these facts. Since the arguments proceed from known properties of Shamir secret sharing and fully homomorphic encryption, and are not central to our presentation, we defer the full treatment to Appendix B for continuity.

In Shamir secret sharing, N servers execute the computation in a distributed manner, using an (N, k) sharing scheme. The primitive values are expressed as elements of a finite field $\mathbb{F}_p$. "Hidden" values are represented by N-tuples of field elements, representing each server's share of the value, and the primitive arithmetic and logical operations (including branching), as well as the initial sharing and decryptions. Intermediate values and communication traces are produced according to the rules of Shamir secret sharing. Against a valid adversarial set of fewer than k servers, the primitives of Shamir secret sharing provide information-theoretic security.

On the other hand, in fully homomorphic encryption, the execution proceeds on a single server, $N = 1$. Except for the initial "hiding" (encryption and sending to the server), and "unhiding" (returning values to the client for decryption) upon `reveal` operations, the operations of the platform are executed on the server according to the definition of the cryptosystem, and produce no communication traces. In this case, the trace for the entire evaluation consists of the encrypted values and the revealed plaintexts, and computational indistinguishability (up to revealed values) follows from the security of the cryptosystem.

6 Implementation

Our implementation consists of three parts. First, we implemented the constructs of the secure execution platform as an embedded domain-specific language in Haskell. Specifically, our EDSL framework consists of a module defining these constructs as a set of combinators, as well as secure multi-party computation (SMC) and fully homomorphic encryption (FHE) libraries implementing the combinators. Second, we implemented a lightweight compiler in Template Haskell, which removes syntactic sugar and translates a subset of Haskell to our EDSL.[6] Finally, we implemented a compiler front-end for $\lambda^{\rightarrow}_{\mathrm{P,S}}$, producing both type and label information for core language programs. In this section, we present these implementations, evaluate their performance, and describe our experience in writing applications.

6.1 A Haskell EDSL for secure execution platforms

Our EDSL is composed of several type classes [36], which define primitives of a secure execution platform (e.g., `(.+)` for addition) in terms of actions in a *secure* I/O monad. In a debug setting, this "secure" monad is simply Haskell's `IO` monad. However, the secure monad used in implementing a real secure execution platform is more complex. We use monad transformers [37] to stack additional functionalities, in a modular fashion, on top of Haskell's `IO`. Specifically, we implement the following functionalities:

- *RNG:* used for implementing random number generation. Specifically, the transformer provides access to the cryptographically strong, deterministic random bit generator (DRBG) of the Haskell crypto-api library.
- *State:* used for threading library-specific state through computations (including the initial parameters ι). For example, the FHE library uses this state to store the public and private keys used in the secure computation.

[6] Notably, the implementation adds constructs for arrays of public length, product and sum types, and bounded iteration primitives. While these features are very useful in practice, we omit them in our present theoretical analysis, as they would further complicate the formalism without providing additional insight.

- *MPI/RPC* (message passing interface and remote procedure calls): used to enable communication among the client and servers. We use the SSL protocol to provide secure, authenticated party-to-party channels.

The constructs for both of our examples of secure execution platforms, SMC and FHE, can be built using this secure monad. For example, the SMC multiplication combinator (`.*`) uses the RNG and MPI functionalities to generate and communicate a new secret sharing among the servers. On the other hand, the FHE multiplication combinator relies on the State functionality to store the key needed by the homomorphic evaluation function (from the Gentry-Halevi implementation of FHE [38, 39]) that performs the actual multiplication. Furthermore, we note that although our implementation currently treats only SMC and FHE, the secure monad can easily be extended with other monad transformers to add features required by other platforms. The flexibility and modularity of monads makes this embedding approach especially attractive.

For both secure execution platforms, SMC and FHE, our EDSL implements secure addition (`.+`), subtraction (`.-`) and multiplication (`.*`), bitwise and logical operators (and, or, and exclusive-or), and comparison operations (equality (`.==`) and inequality (`./=`) testing, less-than (`.<`), greater-than (`.>`), and so on). We also implement branching operators, `sif-sthen-selse`, in terms of arithmetization, as described above: `sif b sthen x selse y` becomes `b .* x + (1 .- b) .* y`.

For Shamir secret sharing, our implementation of the above primitives is essentially a direct translation of the protocol into a Haskell implementation. In the case of fully homomorphic encryption, our library extends the Gentry-Halevi implementation, presented in [38, 39]. Their C++ implementation provides several functions, including a public/private key pair generation function, encryption/decryption functions, a recrypt (ciphertext refreshing) function, and simple single-bit homomorphic arithmetic operators. We extend their implementation by providing support for k-bit homomorphic addition, multiplication, comparison and equality testing functions. In integrating the extended C++ FHE library with our Haskell framework, we implemented C wrappers for the basic FHE operations, and various library functions. The EDSL primitives are implemented as foreign calls to the corresponding C functions, using Haskell's Foreign Function Interface (FFI). Our design hides the low-level C details from the programmer, in addition to adding garbage collection support to the underlying FHE library.

6.2 Extending the EDSL with Template Haskell

Our Template Haskell compiler takes a Haskell AST, enclosed in Template Haskell quotes `[| ... |]`, and outputs a transformed AST, which is spliced into the surrounding code using Template Haskell's `$(...)`. The compiler removes syntactic sugar, performs label inference and static label checks, and translates Haskell library operators, such as `<=`, to our EDSL operators (`.<=`), in addition to inserting type annotations and explicit conversions from public to secret values. An example use of this compiler is shown in Figure 3. In our implementation efforts, this compiler serves as an intermediate step between the EDSL (i.e., using the secure execution platform primitives directly) and the full core language $\lambda^{\rightarrow}_{\mathrm{P,S}}$.

6.3 Core language

We also implement a compiler front-end for our core language, $\lambda^{\rightarrow}_{\mathrm{P,S}}$, adapting standard approaches [40] to perform type inference and type checking according to the rules of Section 5. In ongoing work, we are continuing to improve the compiler front-end, and extending our development to a full $\lambda^{\rightarrow}_{\mathrm{P,S}}$ compiler using the Template Haskell and QuasiQuotes extensions.

6.4 Evaluation

Our experimental setup consists of 6 machines, interconnected on a local Gig-E network, with each machine containing two Intel Xeon E5620 (2.4GHz) processors, and 48GB of RAM. Our SMC implementation uses arithmetic modulo the largest 32-bit prime.

	Addition	Multiplication	Comparison	Assignment
Public	0.003 sec	0.006 sec	0.004 sec	0.003 sec
Secret	0.006 sec	1.71 sec	406.8 sec	3.28 sec

Table 1: Micro-benchmarks for SMC ($N = 5, k = 2$). We measure 1000 operations on randomly generated values.

We measured the performance of several SMC core primitives. In particular, we measure the cost of addition (`.+`), multiplication (`.*`), and comparison (`.>=`) of two secret integers and compare them to the corresponding public operations. Similarly, we compare the cost of assignment to a memory location in secret and public conditionals. Table 1 summarizes these results. We observe that SMC additions, multiplications, and comparisons on secret operands are roughly $1.8\times$, $300\times$ and $91000\times$ slower than the corresponding operation on public values[7]. We note that the time difference in the additions of public and secret values (since addition does not involve network communication) serves as a measurement for the performance overhead incurred by our secret monad, i.e., our library imposes a $1.8\times$ overhead. Assignments are roughly $1000\times$ slower, reflecting the fact that each assignment in a secret context consists of an application of a branching operator $\mathrm{op}_{\mathrm{Br(int)}}$ (which, in the case of SMC, consists of two multiplications, a subtraction, and an addition).

We also measure the cost of using our framework to implement a portion of Reliable Email. Specifically, we measure the cost of checking if an email address is present in a whitelist of 100 random entries. Our results indicate that the average time of checking such a secret list is about 2.4 minutes.

Although the overhead for secret computations seems formidable, we note that our prototype implementation uses a very simple multi-party computation protocol, which incurs a round of communication for each multiplication operation. There exist more efficient protocols that use a constant number of rounds to execute an arbitrary (though pre-specified) sequence of operations [41]. Since the overhead from network latency is significant, the efficiency of our system should improve substantially upon adapting our implementation to use a more round-efficient scheme for sub-computations whose operations are known in advance (i.e., do not depend on values from `reveal` operations). Likewise, we anticipate a significant speedup when independent computations (such as the 100 email address comparisons above) are performed in parallel. Moreover, given the wide interest in secure cloud computing, we expect the performance of the underlying cryptographic primitives, and thereby our EDSL, to improve in the near future.

7 Related work

Our static semantics is similar to many standard type systems for information flow control (for example, the system described in the work of Pottier and Simonet [31]). However, as described in Section 5.2, our system is designed for a dual purpose: rather than ensuring that any expressible program is free of information leaks, our system ensures that any expressible program (which, by definition, cannot leak information on a secure execution platform) is nevertheless implementable in our model. In addition, we need substantially different restrictions to deal with the problem of control flow leakage.

Li and Zdancewic's seminal work [42, 43] presents the first implementation of information flow control as a library, in Haskell. They enforce information flow dynamically, and consider only pure computations; our system relies on strong static guarantees, and addresses a language with side effects. Subsequently [44], Tsai et al. addressed the issue of internal timing for a multi-threaded language; since in our model the servers can only observe their own execution, timing attacks are not relevant. More closely related, Russo et al. [45] present

[7]We omit performance evaluations for fully homomorphic encryption, as past results have shown that the time complexity of existing schemes renders experimental results of limited value [15].

a static information flow library, SecIO, that is statically enforced using Haskell type classes. They prove *termination-insensitive* non-interference for a call-by-name λ-calculus. Our type system enforces substantially stricter requirements on control flow. Nevertheless, their system is complementary to ours, and SecIO could be used to implement some of our static restrictions.

Vaughan presents an extension to Aura, called AuraConf [46], which provides an information flow language with cryptographic support. AuraConf allows programmers with knowledge of access control to implement general decentralized information flow control. The AuraConf system also builds on earlier work by Vaughan and Zdancewic, in which they develop a decentralized label model for cryptographic operations [47]. In addition, Fournet et al., in work on the CFlow system [48], analyze information flow annotations on cryptographic code. In contrast to these approaches, our system abstracts away the cryptography primitives from the language, allowing programmers without specialized knowledge to write secure applications for the cloud.

Systems for secure computations include SCET [49], with focus on economic applications and secure double auctions; FairplayMP [50], a specification language SFDL that is converted to primitive operations on bits; Sharemind [51], for multiparty computations on large datasets; VIFF [12], a basic language embedded in Python and API to cryptographic primitives. These systems implement cryptographic protocols, without proving the more comprehensive correctness and security properties.

The closest work to ours is SMCL [11], an imperative-style DSL with similar goals (notably, SMCL aims to enable programmers to use secure multiparty computation without detailed knowledge of the underlying protocol), as well as some similar constructs (notably, the behavior of SMCL's `open` construct is very similar to our `reveal`). However, our work improves on existing efforts in several respects. While the papers on SMCL do exhibit correctness and security properties, but they do not formally define some crucial aspects: notably, the execution model of the platform, and the security properties required of its primitives so that security for the entire system can be guaranteed. Unlike SMCL, our system also generalizes to other platforms beyond SMC. In addition, our system provides significantly richer language constructs (encompassing both imperative and functional features).

In a previous paper [15], we described a restricted language without recursion, mutable stores, and conditionals. As in this paper, we proved correctness and security for programs written in the language executing on a secure execution platform. Drawing on our experience of working with the EDSL in our previous work, we implemented the Template Haskell compiler and the compiler frontend for our core language. As far as we know, we are the first to formalize and prove correctness and security properties for a unified language framework, providing rich language features such as recursion, mutable stores, and conditionals, and encompassing a wide range of cryptographic schemes for computation on encrypted data.

8 Conclusions

We present an expressive core language for secure cloud computing, with primitive types, conditionals, standard functional features, mutable state, and a secrecy preserving form of general recursion. This language uses an augmented information-flow type system to impose conventional information-flow control and prevent previously unexplored forms of control-flow leakage that may occur when the execution platform is untrusted. The language allows programs to be developed and tested using conventional means, then exported to a variety of secure cloud execution platforms, dramatically reducing the amount of specialized knowledge needed to write secure code. We prove correctness and confidentiality for any platform meeting our definitions, and note two examples of such platforms: fully homomorphic encryption, as well as a multi-party computation protocol based on Shamir secret sharing.

The implementation of our language as a Haskell library allows developers to use standard Haskell software development environments. Programmers also have the benefit of sophisticated type-checking and general programming features of Haskell. On the other hand, implementation in Haskell is not an inherent feature of our core language; other languages with functional features, such as Scala or F# (or even object-oriented languages

such as Java, with some additional implementation effort) would also be reasonable choices. Our core language could also be used to develop secure libraries that can be safely called from other languages, providing the strong security guarantees of our DSL in an unrestricted multi-language programming environment.

In future work, we plan to extend our theoretical framework to other secure execution platforms that can provide stronger guarantees, such as security against active adversaries. We will also explore the possibility of mechanically verifying that a particular implementation realizes our distributed semantics. Finally, we plan to develop more sophisticated implementation techniques, possibly leveraging Template Haskell metaprogramming, such as automatically producing code that is optimized for particular forms of more efficient partially homomorphic encryption schemes.

Acknowledgements

We thank the anonymous reviewers, as well as Amit Levy and Jérémy Planul, for helpful feedback on earlier versions of this paper. This work was supported by DARPA PROCEED, under contract #N00014-11-1-0276-P00002, the National Science Foundation, and the Air Force Office of Scientific Research. Deian Stefan and Joe Zimmerman are further supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program.

References

[1] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC*, 2009, pp. 169–178.

[2] C. Gentry, "Computing arbitrary functions of encrypted data," *Commun. ACM*, vol. 53, no. 3, pp. 97–105, 2010.

[3] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *EUROCRYPT*, 2010, pp. 24–43.

[4] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract)," in *STOC*, 1988, pp. 1–10.

[5] R. Gennaro, M. O. Rabin, and T. Rabin, "Simplified VSS and fact-track multiparty computations with applications to threshold cryptography," in *PODC*, 1998, pp. 101–111.

[6] R. Cramer, I. Damgård, and U. M. Maurer, "General secure multi-party computation from any linear secret-sharing scheme," in *EUROCRYPT*, 2000, pp. 316–334.

[7] M. Naor and K. Nissim, "Communication preserving protocols for secure function evaluation," in *STOC*, 2001, pp. 590–599.

[8] M. C. Silaghi, "SMC: Secure multiparty computation language," http://www.cs.fit.edu/~msilaghi/SMC/tutorial.html, 2004.

[9] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay - secure two-party computation system," in *USENIX Security Symposium*, 2004, pp. 287–302.

[10] P. Bogetoft, D. L. Christensen, I. Damgard, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft, "Multiparty computation goes live," Cryptology ePrint Archive, Report 2008/068, 2008, http://eprint.iacr.org/.

[11] J. D. Nielsen and M. I. Schwartzbach, "A domain-specific programming language for secure multiparty computation," in *PLAS*, 2007, pp. 21–30.

[12] I. Damgård, M. Geisler, M. Krøigaard, and J. B. Nielsen, "Asynchronous multiparty computation: Theory and implementation," in *Public Key Cryptography*, 2009, pp. 160–179.

[13] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: protecting confidentiality with encrypted query processing," in *SOSP*, 2011, pp. 85–100.

[14] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or a completeness theorem for protocols with honest majority," in *STOC*, 1987, pp. 218–229.

[15] A. Bain, J. C. Mitchell, R. Sharma, D. Stefan, and J. Zimmerman, "A domain-specific language for computing on encrypted data (invited talk)," in *FSTTCS*, 2011, pp. 6–24.

[16] S. Garriss, M. Kaminsky, M. J. Freedman, B. Karp, D. Mazières, and H. Yu, "Re: Reliable email," in *NSDI*, 2006.

[17] A. van Deursen, P. Klint, and J. Visser, "Domain-specific languages: An annotated bibliography," http://homepages.cwi.nl/~arie/papers/dslbib.

[18] "Embedded domain-specific languages in haskell," http://www.haskell.org/haskellwiki/Research_papers/Domain_specific_languages.

[19] T. Sheard and S. L. P. Jones, "Template meta-programming for Haskell," *SIGPLAN Notices*, vol. 37, no. 12, pp. 60–75, 2002.

[20] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-DNF formulas on ciphertexts," in *TCC*, 2005, pp. 325–341.

[21] C. Gentry, S. Halevi, and V. Vaikuntanathan, "A simple BGN-type cryptosystem from LWE," in *EUROCRYPT*, 2010, pp. 506–522.

[22] Y. Ishai and A. Paskin, "Evaluating branching programs on encrypted data," in *TCC*, 2007, pp. 575–594.

[23] N. P. Smart and F. Vercauteren, "Fully homomorphic encryption with relatively small key and ciphertext sizes," in *Public Key Cryptography*, 2010, pp. 420–443.

[24] A. C.-C. Yao, "Protocols for secure computations (extended abstract)," in *FOCS*, 1982, pp. 160–164.

[25] Y. Lindell and B. Pinkas, "A proof of security of Yao's protocol for two-party computation," *J. Cryptology*, vol. 22, no. 2, pp. 161–188, 2009.

[26] Y. Ishai and E. Kushilevitz, "Randomizing polynomials: A new representation with applications to round-efficient secure computation," in *FOCS*, 2000, pp. 294–304.

[27] Y. Ishai, E. Kushilevitz, and A. Paskin, "Secure multiparty computation with minimal interaction," in *CRYPTO*, 2010, pp. 577–594.

[28] I. Damgård, Y. Ishai, and M. Krøigaard, "Perfectly secure multiparty computation and the computational overhead of cryptography," in *EUROCRYPT*, 2010, pp. 445–465.

[29] B. Applebaum, Y. Ishai, and E. Kushilevitz, "From secrecy to soundness: Efficient verification via secure computation," in *ICALP (1)*, 2010, pp. 152–163.

[30] A. Sabelfeld and A. C. Myers, "Language-based information-flow security," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 1, pp. 5–19, 2003.

[31] F. Pottier and V. Simonet, "Information flow inference for ML," in *POPL*, 2002, pp. 319–330.

[32] A. C. Myers, "JFlow: Practical mostly-static information flow control," in *POPL*, 1999, pp. 228–241.

[33] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *J. ACM*, vol. 43, no. 3, 1996.

[34] J. M. Lucassen and D. K. Gifford, "Polymorphic effect systems," in *POPL*, 1988, pp. 47–57.

[35] D. Marino and T. D. Millstein, "A generic type-and-effect system," in *TLDI*, 2009, pp. 39–50.

[36] C. V. Hall, K. Hammond, S. L. P. Jones, and P. Wadler, "Type classes in Haskell," in *ESOP*, 1994, pp. 241–256.

[37] S. Liang, P. Hudak, and M. P. Jones, "Monad transformers and modular interpreters," in *POPL*, 1995, pp. 333–343.

[38] C. Gentry and S. Halevi, "Implementing Gentry's fully-homomorphic encryption scheme," Manuscript; see https://researcher.ibm.com/researcher/view_page.php?id=1579, 2010.

[39] C. Gentry and S. Halevi, "Gentry-Halevi implementation of a fully-homomorphic encryption scheme," https://researcher.ibm.com/researcher/files/us-shaih/fhe-code.zip.

[40] L. Damas and R. Milner, "Principal type-schemes for functional programs," in *POPL*, 1982, pp. 207–212.

[41] D. Beaver, S. Micali, and P. Rogaway, "The round complexity of secure protocols (extended abstract)," in *STOC*, 1990, pp. 503–513.

[42] P. Li and S. Zdancewic, "Encoding information flow in Haskell," in *CSFW*, 2006, p. 16.

[43] P. Li and S. Zdancewic, "Arrows for secure information flow," *Theor. Comput. Sci.*, vol. 411, no. 19, pp. 1974–1994, 2010.

[44] T.-C. Tsai, A. Russo, and J. Hughes, "A library for secure multi-threaded information flow in Haskell," in *CSF*, 2007, pp. 187–202.

[45] A. Russo, K. Claessen, and J. Hughes, "A library for light-weight information-flow security in Haskell," in *Haskell*, 2008, pp. 13–24.

[46] J. A. Vaughan, "Auraconf: a unified approach to authorization and confidentiality," in *TLDI*, 2011, pp. 45–58.

[47] J. A. Vaughan and S. Zdancewic, "A cryptographic decentralized label model," in *IEEE Symposium on Security and Privacy*, 2007, pp. 192–206.

[48] C. Fournet, J. Planul, and T. Rezk, "Information-flow types for homomorphic encryptions," in *ACM Conference on Computer and Communications Security*, 2011, pp. 351–360.

[49] P. Bogetoft, I. B. Damgard, T. Jakobsen, K. Nielsen, J. Pagter, and T. Toft, "Secure computing, economy, and trust: A generic solution for secure auctions with real-world applications," BRICS, Tech. Rep. RS-05-18, 2005.

[50] A. Ben-David, N. Nisan, and B. Pinkas, "FairplayMP: a system for secure multi-party computation," in *ACM Conference on Computer and Communications Security*, 2008, pp. 257–266.

[51] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *ESORICS*, 2008, pp. 192–206.

A Proofs of auxiliary lemmas

In this appendix, we give proofs of the auxiliary lemmas in Section 5.4 (omitted above, for continuity).

Lemma 1 (Weakening). *If* $\Gamma, \Sigma, C \vdash \tilde{e} : \tau$*,* $\Gamma' \supseteq \Gamma$*, and* $\Sigma' \supseteq \Sigma$*, then* $\Gamma', \Sigma', C \vdash \tilde{e} : \tau$*.*

Proof. By induction on the structure of $\tilde{e}$. □

Lemma 2 (Substitution). *If* $\Gamma, \Sigma, C \vdash \tilde{e} : \tau$*, and for all* $x \in \operatorname{dom}\Gamma$*, we have* $\Gamma, \Sigma, C \vdash \rho(x) : \Gamma(x)$*, then* $\emptyset, \Sigma, C \vdash \rho(\tilde{e}) : \tau$*.*

Proof. By induction on the structure of $\tilde{e}$. □

Lemma 3 (Output of Evaluation is a Value)**.**

- *If* $(e, \mu) \downarrow (v, \mu', \mathcal{O})$*, then for all* μ_1*,* $(v, \mu_1) \downarrow (v, \mu_1, \varepsilon)$*.*
- *If* $(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}, \tilde{\mu}', \mathcal{O})$*, then for all* $\tilde{\mu}_1$*,* $(\tilde{v}, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}, \tilde{\mu}_1, \varepsilon, \varepsilon)$*.*
- $\tilde{v}$ *is an (extended) value in the syntax if and only if* $(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}, \tilde{\mu}', \mathcal{O})$ *for some* $\tilde{e}, \tilde{\mu}$*.*

Proof. By inspection of the reference and distributed semantics (with the last claim proceeding by induction on the evaluation derivation). □

Lemma 4 (Values are Closed). *If* $\tilde{v}$ *is a value, then* $\tilde{v}$ *contains no free variables.*

Proof. By induction on the structure of $\tilde{v}$. □

Lemma 5 (Well-Typed Values are Well-Typed in Secret Contexts)**.** *If* $\Gamma, \Sigma, C \vdash \tilde{v} : \tau$*, and* $\tilde{v}$ *is a value, then* $\Gamma, \Sigma, \mathrm{S} \vdash \tilde{v} : \tau$*.*

Proof. By inspection of the static semantics. □

Lemma 6 (Context Subtyping)**.** *If* $\Gamma, \Sigma, C \vdash \tilde{e} : \tau$*, and* $C' \sqsubseteq C$*, then* $\Gamma, \Sigma, C' \vdash \tilde{e} : \tau$*,*

Proof. By inspection of the static semantics. □

Lemma 7 (Extension of Permutations)**.** *If* $\Psi' \supseteq \Psi$ *and* $\operatorname{Addrs}(\tilde{e}) \subseteq \operatorname{dom}\Psi$ *then* $\Psi(\tilde{e}) = \Psi'(\tilde{e})$*.*

Proof. By induction on the structure of the expression $\tilde{e}$. □

Lemma 8 (Well-Typed Expressions are Address-Closed)**.** *If* $\emptyset, \Sigma, C \vdash \tilde{e} : \tau$*, then* $\operatorname{Addrs}(\tilde{e}) \subseteq \operatorname{dom}\Sigma$*.*

Proof. By induction on the typing judgment $\emptyset, \Sigma, C \vdash \tilde{e} : \tau$. □

Lemma 9 (Distributed Semantics Preserves Address-Closed Expressions)**.** *If* $(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}, \tilde{\mu}', T, \mathcal{O})$*, and* $\operatorname{Addrs}(\tilde{e}) \subseteq \tilde{\mu}$*, then* $\operatorname{Addrs}(\tilde{v}) \subseteq \tilde{\mu}'$*.*

Proof. By induction on the evaluation derivation $(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}, \tilde{\mu}', T, \mathcal{O})$. □

Lemma 10 (Permutation Invariance of Static Semantics)**.** *If* $\Gamma, \Sigma, C \vdash \tilde{e} : \tau$ *then* $\Gamma, \Psi(\Sigma), C \vdash \Psi(\tilde{e}) : \tau$*.*

Proof. By induction on the typing judgment $\Gamma, \Sigma, C \vdash \tilde{e} : \tau$. □

Lemma 11 (Permutation Invariance of Distributed Semantics)**.** *If* $\operatorname{Addrs}(\tilde{e}) \subseteq \operatorname{dom}\tilde{\mu}$*,* $(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}, \tilde{\mu}_1, T, \mathcal{O})$*, then* $(\Psi(\tilde{e}), \Psi(\tilde{\mu}), \iota) \Downarrow (\Psi(\tilde{v}), \Psi(\tilde{\mu}_1), T, \mathcal{O})$*.*

Proof. By induction on the evaluation derivation $(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}, \tilde{\mu}_1, T, \mathcal{O})$. □

We also require some basic properties of the join and update function, Φ^ι and update^ι, defined above.

Lemma 12 (Phi Takes the Union of Domains)**.** *If* $\tilde{\mu} = \Phi_1^\iota(\tilde{b}, \tilde{\mu}_2, \tilde{\mu}_3)$*, then* $\mathrm{dom}\,\tilde{\mu} = \mathrm{dom}\,\tilde{\mu}_2 \cup \mathrm{dom}\,\tilde{\mu}_3$.

Proof. By inspection of the definition of Φ. □

Lemma 13 (Store Update Preserves Domain)**.** *If* $\tilde{\mu}' = \mathrm{update}_1^\iota(\tilde{\mu}, \tilde{v}, \tilde{v}')$ *and* $\mathrm{Addrs}(\tilde{v}) \subseteq \mathrm{dom}\,\tilde{\mu}$ *then* $\mathrm{dom}\,\tilde{\mu}' = \mathrm{dom}\,\tilde{\mu}$.

Proof. By induction on the structure of $\tilde{v}$. The base case is immediate. If $\tilde{v} = \varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3)$, then the inductive hypothesis gives $\mathrm{dom}\,\mathrm{update}_1^\iota(\tilde{\mu}, \tilde{v}_2, \tilde{v}') = \mathrm{dom}\,\tilde{\mu}$ and $\mathrm{dom}\,\mathrm{update}_1^\iota(\tilde{\mu}, \tilde{v}_3, \tilde{v}') = \mathrm{dom}\,\tilde{\mu}$, and the claim then follows by Lemma 12. □

Lemma 14 (Substitution Commutes With Permutation)**.** *If* $\tilde{e}_2 = \Psi(\tilde{e}_1)$ *and* $\tilde{v}_2 = \Psi(\tilde{v}_1)$*, then* $\tilde{e}_2[\tilde{v}_2/x] = \Psi(\tilde{e}_1[\tilde{v}_1/x])$.

Proof. By induction on the structure of $\tilde{e}_1$. □

Lemma 15 (Permutation Invariance of Join of Values)**.** *If* $(\tilde{v}_1)_b = \Psi((\tilde{v}_1)_a)$*,* $(\tilde{v}_2)_b = \Psi((\tilde{v}_2)_a)$*,* $(\tilde{v}_3)_b = \Psi((\tilde{v}_3)_a)$*,* $\tilde{v}'_a = \Phi_1^\iota((\tilde{v}_1)_a, (\tilde{v}_2)_a), (\tilde{v}_3)_a)$*, and* $\tilde{v}'_b = \Phi_1^\iota((\tilde{v}_1)_b, (\tilde{v}_2)_b), (\tilde{v}_3)_b)$*, then* $\tilde{v}'_b = \Psi(\tilde{v}'_a)$.

Proof. By inspection of the definition of Φ. □

Lemma 16 (Permutation Invariance of Join of Stores)**.** *If* $(\tilde{v}_1)_b = \Psi((\tilde{v}_1)_a)$*,* $(\tilde{\mu}_2)_b = \Psi((\tilde{\mu}_2)_a)$*,* $(\tilde{\mu}_3)_b = \Psi((\tilde{\mu}_3)_a)$*,* $\tilde{\mu}'_a = \Phi_1^\iota((\tilde{v}_1)_a, (\tilde{\mu}_2)_a), (\tilde{\mu}_3)_a)$*, and* $\tilde{\mu}'_b = \Phi_1^\iota((\tilde{v}_1)_b, (\tilde{\mu}_2)_b), (\tilde{\mu}_3)_b)$*, then* $\tilde{\mu}'_b = \Psi(\tilde{\mu}'_a)$.

Proof. The result follows directly by applying Lemma 15 to the contents of each address. □

Lemma 17 (Permutation Invariance of Store Update)**.** *If* $\tilde{\mu}_b = \Psi(\tilde{\mu}_a)$*,* $\tilde{v}_b = \Psi(\tilde{v}_a)$*, and* $\tilde{v}'_b = \Psi(\tilde{v}'_a)$*,* $\tilde{\mu}'_a = \mathrm{update}_1^\iota(\tilde{\mu}_a, \tilde{v}_a, \tilde{v}'_a)$*, and* $\tilde{\mu}'_b = \mathrm{update}_1^\iota(\tilde{\mu}_b, \tilde{v}_b, \tilde{v}'_b)$*, then* $\tilde{\mu}'_b = \Psi(\tilde{\mu}'_a)$.

Proof. By induction on the structure of $\tilde{v}_a$. □

Lemma 18 (Permutation Invariance of Store Selection)**.** *If* $\tilde{\mu}_b = \Psi(\tilde{\mu}_a)$*,* $\tilde{v}_b = \Psi(\tilde{v}_a)$*,* $\tilde{v}'_a = \mathrm{select}_2^\iota(\tilde{\mu}_a, \tilde{v}_a)$*, and* $\tilde{v}'_b = \mathrm{select}_2^\iota(\tilde{\mu}_b, \tilde{v}_b)$*, then* $\tilde{v}'_b = \Psi(\tilde{v}'_a)$.

Proof. By induction on the structure of $\tilde{v}_a$. □

Lemma 19 (Determinism of Distributed Semantics)**.** *If* $(\tilde{e}_a, \tilde{\mu}_a, \iota) \Downarrow (\tilde{v}'_a, \tilde{\mu}'_a, T_a, \mathcal{O}_a)$*,* $\tilde{e}_b = \Psi(\tilde{e}_a)$*,* $\tilde{\mu}_b = \Psi(\tilde{\mu}_a)$*,* $(\tilde{e}_b, \tilde{\mu}_b, \iota) \Downarrow (\tilde{v}'_b, \tilde{\mu}'_b, T_b, \mathcal{O}_b)$*, and* $\Psi : \mathrm{dom}\,\tilde{\mu}_a \leftrightarrow \mathrm{dom}\,\tilde{\mu}_b$*, then* $\mathcal{O}_a = \mathcal{O}_b$ *and for some* $\Psi' \supseteq \Psi$*,* $\Psi' : \mathrm{dom}\,\tilde{\mu}'_a \leftrightarrow \mathrm{dom}\,\tilde{\mu}'_b$*,* $\tilde{v}'_b = \Psi'(\tilde{v}'_a)$ *and* $\tilde{\mu}'_b = \Psi'(\tilde{\mu}'_a)$.

Proof. By induction on the first evaluation derivation $(\tilde{e}_a, \tilde{\mu}_a, \iota) \Downarrow (\tilde{v}'_a, \tilde{\mu}'_a, T_a, \mathcal{O}_a)$. The argument exactly matches that of Lemma 57, with invocations of Lemmas 46–52 replaced by invocations of Lemmas 14–18. □

Lemma 20 (Phi Preserves Value Typing)**.** *If* $\tilde{b} \in \mathcal{E}_\ell(\mathrm{bool})$*;* $\Sigma_2|_{\mathrm{dom}\,\Sigma_3} = \Sigma_3|_{\mathrm{dom}\,\Sigma_2}$*;* $\Sigma' = \Sigma_2 \cup \Sigma_3$*;* $\tilde{v}' = \Phi_1^\iota(\tilde{b}, \tilde{v}_2, \tilde{v}_3)$*;* $\emptyset, \Sigma_2, \mathrm{S} \vdash \tilde{v}_2 : \tau$*; and* $\emptyset, \Sigma_3, \mathrm{S} \vdash \tilde{v}_3 : \tau$*, then* $\emptyset, \Sigma', \mathrm{S} \vdash \tilde{v}' : \tau$.

Proof. By weakening, we have $\emptyset, \Sigma', \mathrm{S} \vdash \tilde{v}_2 : \tau$ and $\emptyset, \Sigma', \mathrm{S} \vdash \tilde{v}_3 : \tau$. The result then follows by inspection of the static semantics and the definition of Φ. □

Lemma 21 (Phi Preserves Store Typing)**.** *If* $\tilde{b} \in \mathcal{E}_\ell(\mathrm{bool})$*,* $\vdash \tilde{\mu}_2 : \Sigma_2$*,* $\vdash \tilde{\mu}_3 : \Sigma_3$*,* $\Sigma_2|_{\mathrm{dom}\,\Sigma_3} = \Sigma_3|_{\mathrm{dom}\,\Sigma_2}$*, and* $\tilde{\mu}' = \Phi_1^\iota(\tilde{b}, \tilde{\mu}_2, \tilde{\mu}_3)$*, then* $\vdash \tilde{\mu}' : \Sigma_2 \cup \Sigma_3$.

Proof. The result is immediate by Lemmas 12 and 20. □

Lemma 22 (Store Update Preserves Store Typing)**.** *If* $\emptyset, \Sigma, C \vdash \tilde{v} : (Y^{\ell}\ \mathrm{ref})^{\ell'}$*;* $\emptyset, \Sigma, C \vdash \tilde{v}' : Y^{\ell}$*;* $\ell' \sqsubseteq \ell$*;* $\vdash \tilde{\mu} : \Sigma$*; and* $\tilde{\mu}' = \mathrm{update}_1^{\iota}(\tilde{\mu}, \tilde{v}, \tilde{v}')$*, then* $\vdash \tilde{\mu}' : \Sigma$*.*

Proof. By induction on the structure of $\tilde{v}$. □

Lemma 23 (Store Selection Preserves Typing)**.** *If* $\emptyset, \Sigma, \mathrm{S} \vdash \tilde{v} : (Y^{\ell'}\ \mathrm{ref})^{\ell''}$*;* $\ell', \ell'' \sqsubseteq \ell$*;* $\vdash \tilde{\mu} : \Sigma$*; and* $\tilde{v}' = \mathrm{select}_1^{\iota}(\tilde{\mu}, \tilde{v})$*, then* $\emptyset, \Sigma, \mathrm{S} \vdash \tilde{v}' : Y^{\ell}$*.*

Proof. By induction on the structure of $\tilde{v}$. □

Lemma 24 (Type Preservation for Distributed Semantics)**.** *If* $\emptyset, \Sigma, C \vdash \tilde{e} : \tau$*,* $\vdash \tilde{\mu} : \Sigma$*, and* $(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \mathcal{O})$*, then for some* Σ'*,* $(\tilde{\mu}', \Sigma') \succ^{\iota} (\tilde{\mu}, \Sigma)$*, and* $\emptyset, \Sigma', C' \vdash \tilde{v}' : \tau$ *for any* C'*.*

Proof. First, we note that by Lemma 6, it suffices to show the conclusion for the particular context $C' = \mathrm{S}$. Now, we proceed by induction on the evaluation derivation $(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \mathcal{O})$.

- Suppose $(\texttt{if}\ \tilde{e}_1\ \texttt{then}\ \tilde{e}_2\ \texttt{else}\ \tilde{e}_3, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \mathcal{O})$. Then $(\tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T_1, \mathcal{O}_1)$. Inversion on the typing judgment gives $\emptyset, \Sigma, C \vdash \tilde{e}_1 : \tau_1$, $\emptyset, \Sigma, C \vdash \tilde{e}_2 : \tau$, and $\emptyset, \Sigma, C \vdash \tilde{e}_3 : \tau$, with $\tau_1 = \mathrm{bool}^{\ell}$. So, by the inductive hypothesis, we have $(\tilde{\mu}_1, \Sigma_1) \succ^{\iota} (\tilde{\mu}, \Sigma)$, and $\emptyset, \Sigma_1, \mathrm{S} \vdash \tilde{v}_1 : \mathrm{bool}^{\ell}$. In addition, $(\tilde{e}_2, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}_2, \tilde{\mu}_2, T_2, \mathcal{O}_2)$; by weakening (Lemma 1), we have $\emptyset, \Sigma_1, C \vdash \tilde{e}_2 : \tau$; and thus $(\tilde{\mu}_2, \Sigma_2) \succ^{\iota} (\tilde{\mu}_1, \Sigma_1) \succ^{\iota} (\tilde{\mu}, \Sigma)$ and $\Gamma, \Sigma_2, \mathrm{S} \vdash \tilde{v}_2 : \tau$. Now, either $\tilde{v}_1 \in \mathcal{E}_{\ell'}(\texttt{true})$ or $\tilde{v}_1 \in \mathcal{E}_{\ell'}(\texttt{false})$; without loss of generality we assume $\tilde{v}_1 \in \mathcal{E}_{\ell'}(\texttt{true})$. Then we have the following cases:
  - $\tilde{v}_1 \in \mathcal{E}_{\mathrm{P}}(\texttt{true})$. In this case, $\tilde{v}' = \tilde{v}_2$, and setting $\Sigma' = \Sigma_2$, the claim follows immediately from the above.
  - $\tilde{v}_1 \in \mathcal{E}_{\mathrm{S}}(\texttt{true})$. In this case, $(\tilde{e}_3, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}_3, \tilde{\mu}_3, T_3, \mathcal{O}_3)$, $\tilde{v}' = \Phi_1^{\iota}(\tilde{v}_1, \tilde{v}_2, \tilde{v}_3)$, and $\tilde{\mu}' = \Phi_1^{\iota}(\tilde{v}_1, \tilde{\mu}_2, \tilde{\mu}_3)$. Now, we must have $\tau_1 = \mathrm{bool}^{\mathrm{S}}$, and so $\emptyset, \Sigma, \mathrm{S} \vdash \tilde{e}_2 : \tau$ and $\emptyset, \Sigma, \mathrm{S} \vdash \tilde{e}_3 : \tau$ (and thus, by weakening (Lemma 1), $\emptyset, \Sigma_1, \mathrm{S} \vdash \tilde{e}_2 : \tau$ and $\emptyset, \Sigma_1, \mathrm{S} \vdash \tilde{e}_3 : \tau$. So the inductive hypothesis gives $\emptyset, \Sigma_3, \mathrm{S} \vdash \tilde{v}_3 : \tau$ and $(\tilde{\mu}_3, \Sigma_3) \succ^{\iota} (\tilde{\mu}_1, \Sigma_1) \succ^{\iota} (\tilde{\mu}, \Sigma)$. Finally, setting $\Sigma' = \Sigma_2 \cup \Sigma_3$, and invoking Lemma 20, we conclude that $\emptyset, \Sigma', \mathrm{S} \vdash \tilde{v}' : \tau$; and invoking Lemma 21, we conclude that $(\tilde{\mu}', \Sigma') \succ^{\iota} (\tilde{\mu}_1, \Sigma_1) \succ^{\iota} (\tilde{\mu}, \Sigma)$, as desired.
- Suppose $(\tilde{e}_0\ \tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \mathcal{O})$, so that $(\tilde{e}_0, \tilde{\mu}, \iota) \Downarrow (\tilde{v}_0, \tilde{\mu}_0, T_0, \mathcal{O}_0)$ and $(\tilde{e}_1, \tilde{\mu}_0, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T_1, \mathcal{O}_1)$. Then the typing judgment gives $\emptyset, \Sigma, C \vdash \tilde{e}_0 : (\tau_1 \xrightarrow{C} \tau)^{\ell}$ and $\emptyset, \Sigma, C \vdash \tilde{e}_1 : \tau_1$, where $\ell \sqsubseteq C$, and thus by the inductive hypothesis, $\emptyset, \Sigma_0, C \vdash \tilde{v}_0 : (\tau_1 \xrightarrow{C} \tau)^{\ell}$ with $(\tilde{\mu}_0, \Sigma_0) \succ^{\iota} (\tilde{\mu}, \Sigma)$. In addition, by weakening (Lemma 1), $\emptyset, \Sigma_0, C \vdash \tilde{e}_1 : \tau_1$, and thus the inductive hypothesis also gives $\emptyset, \Sigma_1, C \vdash \tilde{v}_1 : \tau_1$ with $(\tilde{\mu}_1, \Sigma_1) \succ^{\iota} (\tilde{\mu}_0, \Sigma_0)$. Now, we have the following cases:
  - Suppose $\tilde{v}_0 = \lambda x.\tilde{e}_0'$. Then, by definition, $\{x \mapsto \tau_1\}, \Sigma_0, C \vdash \tilde{e}_0' : \tau$, so that substitution (Lemma 2) gives $\emptyset, \Sigma_0, \mathrm{S} \vdash \tilde{e}_0'[\tilde{v}_1/x] : \tau$ (and, by weakening (Lemma 1), $\emptyset, \Sigma_1, C \vdash \tilde{e}_0'[\tilde{v}_1/x] : \tau$). Finally, since $(\tilde{e}_0'[\tilde{v}_1/x], \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T_3, \mathcal{O}_3)$, the inductive hypothesis gives $(\tilde{\mu}', \Sigma') \succ^{\iota} (\tilde{\mu}_1, \Sigma_1) \succ^{\iota} (\tilde{\mu}, \Sigma)$ and $\emptyset, \Sigma_1, \mathrm{S} \vdash \tilde{v}' : \tau$, as desired.
  - Suppose $\tilde{v}_0 = (\texttt{fix}\ f.\lambda x.\tilde{e}_0')$. Then, by definition, $\{f \mapsto (\tau_1 \xrightarrow{C} \tau)^{\ell}, x \mapsto \tau_1\}, \Sigma_0, C \vdash \tilde{e}_0' : \tau$, so that substitution (Lemma 2) gives $\emptyset, \Sigma_0, \mathrm{S} \vdash \tilde{e}_0'[\tilde{v}_0/f, \tilde{v}_1/x] : \tau$ (and, by weakening (Lemma 1), $\emptyset, \Sigma_1, C \vdash \tilde{e}_0'[\tilde{v}_0/f, \tilde{v}_1/x] : \tau$). Finally, since $(\tilde{e}_0'[\tilde{v}_0/f, \tilde{v}_1/x], \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T_3, \mathcal{O}_3)$, the inductive hypothesis gives $(\tilde{\mu}', \Sigma') \succ^{\iota} (\tilde{\mu}_1, \Sigma_1) \succ^{\iota} (\tilde{\mu}, \Sigma)$ and $\emptyset, \Sigma_1, \mathrm{S} \vdash \tilde{v}' : \tau$, as desired.
  - Suppose $\tilde{v}_0 = \varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3)$, In this case, $\tilde{b} \in \mathcal{E}_{\ell_1}(\mathrm{bool})$, $(\tilde{v}_2\ \tilde{v}_1, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}_2', \tilde{\mu}_2', T_2, \mathcal{O}_2)$, $(\tilde{v}_3\ \tilde{v}_1, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}_3', \tilde{\mu}_3', T_3, \mathcal{O}_3)$, $\tilde{v}' = \varphi(\tilde{b}, \tilde{v}_2', \tilde{v}_3')$, and $\tilde{\mu}' = \varphi(\tilde{b}, \tilde{\mu}_2', \tilde{\mu}_3')$. Now, we have $\emptyset, \Sigma_0, \mathrm{S} \vdash \tilde{v}_2 : (\tau_1 \xrightarrow{C} \tau)^{\ell}$ and $\emptyset, \Sigma_0, \mathrm{S} \vdash \tilde{v}_3 : (\tau_1 \xrightarrow{C} \tau)^{\ell}$ (and, by weakening (Lemma 1), $\emptyset, \Sigma_1, \mathrm{S} \vdash \tilde{v}_2 : (\tau_1 \xrightarrow{C} \tau)^{\ell}$ and $\emptyset, \Sigma_1, \mathrm{S} \vdash \tilde{v}_3 : (\tau_1 \xrightarrow{C} \tau)^{\ell}$). Thus, the inductive hypothesis gives $\emptyset, \Sigma_2, \mathrm{S} \vdash \tilde{v}_2' : \tau$ with

$(\tilde{\mu}_2', \Sigma_2) \succ^\iota (\tilde{\mu}_1, \Sigma_1)$; and, similarly, $\emptyset, \Sigma_3, \mathrm{S} \vdash \tilde{v}_3' : \tau$ with $(\tilde{\mu}_3', \Sigma_3) \succ^\iota (\tilde{\mu}_1, \Sigma_1)$; Finally, setting $\Sigma' = \Sigma_2 \cup \Sigma_3$, and invoking Lemma 20, we conclude that $\emptyset, \Sigma', \mathrm{S} \vdash \tilde{v}' : \tau$; and invoking Lemma 21, we conclude that $(\tilde{\mu}', \Sigma') \succ^\iota (\tilde{\mu}_1, \Sigma_1) \succ^\iota (\tilde{\mu}, \Sigma)$, as desired.

- Suppose $(\tilde{e}_1 := \tilde{e}_2, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \mathcal{O})$, so that $(\tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T_1, \mathcal{O}_1)$, $(\tilde{e}_2, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}_2, \tilde{\mu}_2, T_2, \mathcal{O}_2)$, $\tilde{\mu}' = \mathrm{update}_1^\iota(\tilde{\mu}_2, \tilde{v}_1, \tilde{v}_2)$, and $\tilde{v}' = ()$. Then the typing judgment gives $\tau = \mathrm{unit}^{\ell''}$, $\emptyset, \Sigma, C \vdash \tilde{e}_1 : (Y^\ell \ \mathrm{ref})^{\ell'}$, with $\ell' \sqsubseteq \ell$, and $\emptyset, \Sigma, C \vdash \tilde{e}_2 : Y^\ell$. Thus, the inductive hypothesis gives $\emptyset, \Sigma_1, C \vdash \tilde{v}' : (Y^\ell \ \mathrm{ref})^{\ell'}$, with $(\tilde{\mu}_1, \Sigma_1) \succ^\iota (\tilde{\mu}, \Sigma)$. By weakening (Lemma 1), we also have $\emptyset, \Sigma_1, C \vdash \tilde{e}_2 : Y^\ell$, so that the inductive hypothesis gives $\emptyset, \Sigma_1, C \vdash \tilde{v}_2 : Y^\ell$ with $(\tilde{\mu}_2, \Sigma_2) \succ^\iota (\tilde{\mu}_1, \Sigma_1)$. Now, evidently $\emptyset, \Sigma_2, C \vdash () : \mathrm{unit}^{\ell''}$; and, by Lemma 22, we have $(\tilde{\mu}', \Sigma_2) \succ^\iota (\tilde{\mu}_2, \Sigma_2)$, as desired.

- Suppose $(!\tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \mathcal{O})$, so that $(\tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}', T, \mathcal{O})$ and $\tilde{v}' = \mathrm{select}_1^\iota(\tilde{\mu}', \tilde{v}_1)$. Then the typing judgment gives $\tau = Y^\ell$ and $\emptyset, \Sigma, C \vdash \tilde{e}_1 : (Y^{\ell'} \ \mathrm{ref})^{\ell''}$, with $\ell', \ell'' \sqsubseteq \ell$. Thus, the inductive hypothesis gives $\emptyset, \Sigma_1, \mathrm{S} \vdash \tilde{v}_1 : (Y^{\ell'} \ \mathrm{ref})^{\ell''}$, with $(\tilde{\mu}_1, \Sigma_1) \succ^\iota (\tilde{\mu}, \Sigma)$. The claim then follows from Lemma 23.

- Suppose $(\texttt{ref}\ \tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \mathcal{O})$, so that $(\tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T, \mathcal{O})$, $\tilde{v}' = a \notin \mathrm{dom}\, \tilde{\mu}_1$, and $\tilde{\mu}' = \tilde{\mu}_1[a \mapsto \tilde{v}_1]$. Then the typing judgment gives $\tau = (Y^{\ell'} \ \mathrm{ref})^\ell$ and $\emptyset, \Sigma, C \vdash \tilde{e} : Y^{\ell'}$, so that the inductive hypothesis gives $\emptyset, \Sigma_1, \mathrm{S} \vdash \tilde{v}_1 : Y^{\ell'}$ with $(\tilde{\mu}_1, \Sigma_1) \succ^\iota (\tilde{\mu}, \Sigma)$, and thus $\tilde{v}_1 \in \mathcal{E}_{\ell''}(Y)$ for some $\ell'' \sqsubseteq \ell'$. Setting $\Sigma' = \Sigma_1[a \mapsto Y^{\ell'}]$, we have $(\tilde{\mu}', \Sigma') \succ^\iota (\tilde{\mu}_1, \Sigma_1) \succ^\iota (\tilde{\mu}, \Sigma)$, as desired.

- Suppose $(\texttt{reveal}\ \tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \mathcal{O})$, so that $(\tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}, T_1, \mathcal{O}_1)$, and $\tilde{v}' = \pi_1(\mathrm{Dec}(\tilde{v}_1))$. Then the typing judgment gives $\tau = Y^\ell$ and $\emptyset, \Sigma, C \vdash \tilde{e}_1 : Y^{\mathrm{S}}$, so that the inductive hypothesis gives $\emptyset, \Sigma', \mathrm{S} \vdash \tilde{v}_1 : Y^{\mathrm{S}}$, with $(\tilde{\mu}', \Sigma') \succ^\iota (\tilde{\mu}, \Sigma)$. Hence $\tilde{v}_1 \in \mathcal{E}_{\ell'}(Y)$, and thus by definition of Dec, $\tilde{v}' \in Y$. So $\emptyset, \Sigma', \mathrm{S} \vdash \tilde{v}' : Y^\ell$, as desired.

- Suppose $(\mathtt{op}_i(\tilde{e}_1, \ldots, \tilde{e}_r), \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \mathcal{O})$, so that for all $j \in \{1, \ldots, r\}$, $(\tilde{e}_j, \tilde{\mu}_{j-1}, \iota) \Downarrow (\tilde{v}_j, \tilde{\mu}_j, T_j, \mathcal{O}_j)$ (where $\tilde{\mu}_0 = \tilde{\mu}$ and $\tilde{\mu}' = \tilde{\mu}_r$), $\tilde{v}_j \in \mathcal{E}_{\ell_j'}(Y_j)$, $\tilde{v}' = \mathrm{Enc}_{\ell_1', \ldots, \ell_r'}(\mathrm{op}_i)(\tilde{v}_1, \ldots, \tilde{v}_r) \in \mathcal{E}_{\ell'}(Y)$ (where $\ell' = \sqcup_j \ell_j'$), and $\mathrm{op}_i : \prod_j Y_j \to Y$. Then the typing judgment gives $\tau = Y^\ell$ and $\emptyset, \Sigma, C \vdash \tilde{e}_j : Y_j^{\ell_j}$, where $\ell = \sqcup_j \ell_j$. After r successive applications of the inductive hypothesis, along with weakening (Lemma 1), we obtain $\emptyset, \Sigma_r, \mathrm{S} \vdash \tilde{v}_j : Y_j^{\ell_j}$, with $(\tilde{\mu}_r, \Sigma_r) \succ^\iota \ldots \succ^\iota (\tilde{\mu}_0, \Sigma_0)$ (where $\Sigma_0 = \Sigma$). Thus $\ell_j' \sqsubseteq \ell_j$ for all j, and hence $\ell' \sqsubseteq \ell$, so that $\emptyset, \Sigma', \mathrm{S} \vdash \tilde{v}' : Y^\ell$, as desired.

□

Lemma 25 (Permutation Invariance of Public Extension)**.** *If* $\tilde{\mu}' \succsim_\Sigma^\iota \tilde{\mu}$, $\vdash \tilde{\mu} : \Sigma$, *and* $\Psi|_{\mathrm{dom}\,\Sigma} = \mathrm{id}$, *then* $\Psi(\tilde{\mu}') \succsim_\Sigma^\iota \tilde{\mu}$.

Proof. Follows directly from the definition of public extension. □

Lemma 26 (Permutation Invariance of Full Public Extension)**.** *If* $(\tilde{\mu}', \Sigma') \succsim^\iota (\tilde{\mu}, \Sigma)$ *and* $\Psi|_{\mathrm{dom}\,\Sigma} = \mathrm{id}$, *then* $(\Psi(\tilde{\mu}'), \Psi(\Sigma')) \succsim^\iota (\tilde{\mu}, \Sigma)$.

Proof. Immediate from Lemma 25. □

Lemma 27 (Phi is a Public Extension)**.** *If* $\tilde{b} \in \mathcal{E}_\ell(\mathrm{bool})$, $(\tilde{\mu}_2, \Sigma_2) \succsim^\iota (\tilde{\mu}_1, \Sigma_1)$, $(\tilde{\mu}_3, \Sigma_3) \succsim^\iota (\tilde{\mu}_1, \Sigma_1)$, $\mathrm{dom}\,\Sigma_2 \cap \mathrm{dom}\,\Sigma_3 \subseteq \mathrm{dom}\,\Sigma_1$, *then* $(\Phi_1^\iota(\tilde{b}, \tilde{\mu}_2, \tilde{\mu}_3), \Sigma_2 \cup \Sigma_3) \succsim^\iota (\tilde{\mu}_1, \Sigma_1)$.

Proof. By Lemma 21, we must only show that $\Phi_1^\iota(\tilde{b}, \tilde{\mu}_2, \tilde{\mu}_3)$ is well-defined and $\Phi_1^\iota(\tilde{b}, \tilde{\mu}_2, \tilde{\mu}_3) \succsim_{\Sigma_1}^\iota \tilde{\mu}_1$. Fix $a \in \mathrm{dom}\,\Sigma_2 \cup \mathrm{dom}\,\Sigma_3$. If $a \in \mathrm{dom}\,\Sigma_1$ and $\Sigma_1(a) = \mathrm{P}$, then $\Sigma_2(a) = \Sigma_3(a)$, so that $\Phi_1^\iota(\tilde{b}, \tilde{\mu}_2, \tilde{\mu}_3)(a) = \tilde{\mu}_2(a) = \tilde{\mu}_3(a)$; otherwise, $\Phi_1^\iota(\tilde{b}, \tilde{\mu}_2, \tilde{\mu}_3)(a)$ is a secret value, and the result follows by inspection of the definition of Φ. □

Lemma 28 (Reducible Values are Values)**.** *If* $\tilde{v} \in \mathcal{V}(\tau, \Sigma)$*, then* $\tilde{v}$ *is a value.*

Proof. By inspection of the reducibility judgment. □

Lemma 29 (Permutation Invariance of Reducibility Predicate)**.** *If* $\tilde{v} \in \mathcal{V}(\tau, \Sigma)$*, then* $\Psi(\tilde{v}) \in \mathcal{V}(\tau, \Psi(\Sigma))$.

Proof. By induction on the type τ. In the cases for lambda abstraction and phi of lambda abstractions, the result follows from Lemmas 11 and 14; in other cases, the result is immediate. □

Lemma 30 (Store Weakening for Reducibility Predicate)**.** *If* $\tilde{v} \in \mathcal{V}(\tau, \Sigma)$*, and* $\Sigma' \supseteq \Sigma$*, then* $\tilde{v} \in \mathcal{V}(\tau, \Sigma')$.

Proof. By induction on the judgment $\tilde{v} \in \mathcal{V}(\tau, \Sigma)$. The cases for abstraction and phi follow since the premise is immediately generalized to $\Sigma_1 \supseteq \Sigma$, while the cases for base value, nested phi of locations, and nested phi of arrows are immediate. On the other hand, if $\tilde{v}$ is a location, $\tilde{v} = a \in \mathcal{V}((Y^{\ell'}\ \mathrm{ref})^{\ell}, \Sigma)$, then $\Sigma'(a) = \Sigma(a) = Y^{\ell'}$, so again immediately $a \in \mathcal{V}((Y^{\ell'}\ \mathrm{ref})^{\ell}, \Sigma')$, as desired. □

Lemma 31 (Phi of Reducible Values is Reducible)**.** *If* $\tilde{b} \in \mathcal{E}_{\ell}(\mathrm{bool})$*,* $\tilde{v}_2 \in \mathcal{V}(\tau, \Sigma_2)$*,* $\tilde{v}_3 \in \mathcal{V}(\tau, \Sigma_3)$*, and* $\Sigma_{2,3} \supseteq \Sigma_2, \Sigma_3$*, then* $\Phi_1^{\iota}(\tilde{b}, \tilde{v}_2, \tilde{v}_3) \in \mathcal{V}(\tau, \Sigma_{2,3})$.

Proof. By Lemma 30, we have $\tilde{v}_2, \tilde{v}_3 \in \mathcal{V}(\tau, \Sigma_{2,3})$. The result then follows by case analysis on the reducibility judgments. □

Lemma 32 (Update to Reducible Secret Reference is a Public Extension)**.** *If* $\tilde{v} \in \mathcal{V}((Y^{\mathrm{S}}\ \mathrm{ref})^{\ell'}, \Sigma)$*,* $\vdash \tilde{\mu} : \Sigma$*, and* $\tilde{v}' \in \mathcal{E}_{\ell}(Y)$*, then* $(\mathrm{update}_1^{\iota}(\tilde{\mu}, \tilde{v}, \tilde{v}'), \Sigma) \succsim^{\iota} (\tilde{\mu}, \Sigma)$.

Proof. By induction on the reducibility predicate $\tilde{v} \in \mathcal{V}((Y^{\mathrm{S}}\ \mathrm{ref})^{\ell'}, \Sigma)$. □

Lemma 33 (Select on Reducible Secret Reference is Reducible)**.** *If* $\tilde{v} \in \mathcal{V}((Y^{\ell'})^{\ell''}\ \mathrm{ref}, \Sigma)$*,* $\vdash \tilde{\mu} : \Sigma$*,* $\ell' \sqsubseteq \ell$*, and* $\ell'' \sqsubseteq \ell$*, then* $\mathrm{select}_1^{\iota}(\tilde{\mu}, \tilde{v}) \in \mathcal{V}(Y^{\ell}, \Sigma)$.

Proof. By induction on the reducibility predicate $\tilde{v} \in \mathcal{V}((Y^{\ell'}\ \mathrm{ref})^{\ell''}, \Sigma)$. □

Lemma 34 (Purity for Applications of Pure Values)**.** *If* $\tilde{v} \in \mathcal{V}(\tau_1 \xrightarrow{\mathrm{S}} \tau, \Sigma_0)$*,* $(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}, \tilde{\mu}_0, T_0, \mathcal{O}_0)$*,* $(\tilde{e}_1, \tilde{\mu}_0, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T_1, \mathcal{O}_1)$*,* $\tilde{v}_1 \in \mathcal{V}(\tau_1, \Sigma_1)$*, and* $(\tilde{\mu}_1, \Sigma_1) \succ^{\iota} (\tilde{\mu}_0, \Sigma_0) \succ^{\iota} (\tilde{\mu}, \Sigma)$*, then* $(\tilde{e}\ \tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T', \mathcal{O}_0 \| \mathcal{O}_1)$*,* $(\tilde{\mu}', \Sigma') \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1)$*, and* $\tilde{v}' \in \mathcal{V}(\tau, \Sigma')$.

Proof. By induction on the reducibility judgment $\tilde{v} \in \mathcal{V}(\tau_1 \xrightarrow{\mathrm{S}} \tau, \Sigma_1)$.

- Suppose $\tilde{v} = \lambda x.\tilde{e}_0$, reducible by the abstraction rule. The premise of that rule then gives that $(\tilde{e}_0[\tilde{v}_1/x], \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T', \varepsilon)$ with $(\tilde{\mu}', \Sigma') \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1)$ and $\tilde{v}' \in \mathcal{V}(\tau, \Sigma')$. Further, by Lemma 28, $\tilde{v}'$ and $\tilde{v}_1$ are values, so that $(\tilde{v}', \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}', \tilde{\mu}_1, \varepsilon, \varepsilon)$ and $(\tilde{v}_1, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, \varepsilon, \varepsilon)$. Thus, the claim follows immediately by the rule for application of lambda abstraction.

- Suppose $\tilde{v} = \varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3)$, reducible by the phi rule. The premises of that rule then yield $\tilde{b} \in \mathcal{E}_{\ell_1}(\mathrm{bool})$; $(\tilde{v}_2\ \tilde{v}_1, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}_2', \tilde{\mu}_2, T_2, \varepsilon)$ with $(\tilde{\mu}_2, \Sigma_2) \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1)$ and $\tilde{v}_2' \in \mathcal{V}(\tau, \Sigma_2)$; and $(\tilde{v}_3\ \tilde{v}_1, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}_3', \tilde{\mu}_3, T_3, \varepsilon)$ with $(\tilde{\mu}_3, \Sigma_3) \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1)$ and $\tilde{v}_3' \in \mathcal{V}(\tau, \Sigma_3)$. Let Ψ be a permutation that extends $\mathrm{id}_{\mathrm{dom}\,\Sigma_1}$ so that $\Psi(a) \notin \mathrm{dom}\,\Sigma_2$ for any $a \in \mathrm{dom}\,\Sigma_3$ (i.e., $\mathrm{dom}\,\Sigma_2 \cap \mathrm{dom}\,\Psi(\Sigma_3) = \mathrm{dom}\,\Sigma_1$). Then by Lemma 26, $(\Psi(\tilde{\mu}_3), \Psi(\Sigma_3)) \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1)$, and by Lemma 29, $\Psi(\tilde{v}_3') \in \mathcal{V}(\tau, \Psi(\Sigma_3))$. We now define $\tilde{v}' = \Phi_1^{\iota}(\tilde{b}, \tilde{v}_2', \Psi(\tilde{v}_3'))$; $\tilde{\mu}' = \Phi_1^{\iota}(\tilde{b}, \tilde{\mu}_2, \Psi(\tilde{\mu}_3))$; and $\Sigma' = \Sigma_2 \cup \Psi(\Sigma_3)$ (well-defined since $\Sigma_2, \Psi(\Sigma_3) \supseteq \Sigma_1$ and $\mathrm{dom}\,\Sigma_2 \cap \mathrm{dom}\,\Psi(\Sigma_3) \subseteq \mathrm{dom}\,\Sigma_1$). In addition, Lemma 31 gives $\tilde{v}' \in \mathcal{V}(\tau, \Sigma')$. Thus, by the rule for application of phi, we conclude $(\tilde{e}\ \tilde{e}_1, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \mathcal{O}_0 \| \mathcal{O}_1 \| \varepsilon)$ (for some trace T produced by concatenating traces above). Finally, Lemma 27 gives $(\tilde{\mu}', \Sigma') \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1)$, as desired.

- Suppose $\tilde{v} = \varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3)$, reducible by the rule for nested phi. In this case, $\tilde{b} \in \mathcal{E}_{\ell_1}(\mathrm{bool})$ and $\tilde{v}_2, \tilde{v}_3 \in \mathcal{V}((\tau_1 \xrightarrow{\mathrm{S}} \tau)^{\ell}, \Sigma)$. Thus, the inductive hypothesis gives $(\tilde{v}_2\ \tilde{v}_1, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}_2', \tilde{\mu}_2, T_2, \varepsilon)$ with $(\tilde{\mu}_2, \Sigma_2) \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1)$ and $\tilde{v}_2' \in \mathcal{V}(\tau, \Sigma_2)$; and $(\tilde{v}_3\ \tilde{v}_1, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}_3', \tilde{\mu}_3, T_3, \varepsilon)$ with $(\tilde{\mu}_3, \Sigma_3) \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1)$ and $\tilde{v}_3' \in \mathcal{V}(\tau, \Sigma_3)$, and the proof proceeds as in the previous case.

□

Lemma 35 (Conditional Purity for Distributed Semantics)**.** *If* $\Gamma, \Sigma, \mathrm{S} \vdash \tilde{e} : \tau$*, and for all* $x \in \mathrm{dom}\,\Gamma$*,* $\rho(x) \in \mathcal{V}(\Gamma(x), \Sigma)$ *and* $\mathrm{Addrs}(\rho(x)) \subseteq \mathrm{dom}\,\Sigma$*, then for some* $\tilde{v}$*,* T*, and* $(\tilde{\mu}', \Sigma') \succsim^{\iota} (\tilde{\mu}, \Sigma)$*,* $(\rho(\tilde{e}), \tilde{\mu}, \iota) \Downarrow (\tilde{v}, \tilde{\mu}', T, \varepsilon)$ *and* $\tilde{v} \in \mathcal{V}(\tau, \Sigma')$*.*

Proof. By induction on the typing judgment $\Gamma, \Sigma, \mathrm{S} \vdash \tilde{e} : \tau$. Some cases are immediately ruled out by the context component, leaving only the following:

- Suppose $\Gamma, \Sigma, \mathrm{S} \vdash \texttt{if}\ \tilde{e}_1\ \texttt{then}\ \tilde{e}_2\ \texttt{else}\ \tilde{e}_3 : \tau$. Then we have $\Gamma, \Sigma, \mathrm{S} \vdash \tilde{e}_1 : \tau_1$, $\Gamma, \Sigma, \mathrm{S} \vdash \tilde{e}_2 : \tau$, and $\Gamma, \Sigma, \mathrm{S} \vdash \tilde{e}_3 : \tau$, with $\tau_1 = \mathrm{bool}^{-}$. Thus, the inductive hypothesis gives $(\rho(\tilde{e}_1), \tilde{\mu}, \iota) \Downarrow (\tilde{b}, \tilde{\mu}_1, T_1, \varepsilon)$ with $(\tilde{\mu}_1, \Sigma_1) \succsim^{\iota} (\tilde{\mu}, \Sigma)$ and $(\tilde{b}, \tilde{\mu}_1) \in \mathcal{V}(\mathrm{bool}^{\langle -, \mathrm{S} \rangle}, \Sigma_1)$. In addition, by weakening of typing (Lemma 1), $\Gamma, \Sigma_1, \mathrm{S} \vdash \tilde{e}_2 : \tau$ and $\Gamma, \Sigma_1, \mathrm{S} \vdash \tilde{e}_3 : \tau$; and by weakening of the reducibility predicate (Lemma 30), $(\rho(x), \tilde{\mu}_1) \in \mathcal{V}(\Gamma(x), \Sigma_1)$ for all $x \in \mathrm{dom}\,\Gamma$. Inversion on the reducibility judgment $(\tilde{b}, \tilde{\mu}_1) \in \mathcal{V}(\mathrm{bool}^{-}, \Sigma_1)$ now gives the following cases:
  - Suppose $\tilde{b} \in \mathcal{E}^{\iota}_{\mathrm{P}}(\mathrm{bool}) = \mathrm{bool}$. In this case, $\tilde{b} = \texttt{true}$ or $\tilde{b} = \texttt{false}$; without loss of generality we assume $\tilde{b} = \texttt{true}$. Then the inductive hypothesis immediately gives $(\rho(\tilde{e}_2), \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T', \varepsilon)$, with $(\tilde{\mu}', \Sigma') \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1) \succsim^{\iota} (\tilde{\mu}, \Sigma)$ and $(\tilde{v}', \tilde{\mu}') \in \mathcal{V}(\tau, \Sigma')$, as desired.
  - Suppose $\tilde{b} \in \mathcal{E}^{\iota}_{\mathrm{S}}(\mathrm{bool})$, so that $\tau = t^{\mathrm{S}}$; assume without loss of generality that $\tilde{b} \in \mathcal{E}^{\iota}_{\mathrm{S}}(\texttt{true})$. Then the inductive hypothesis gives $(\rho(\tilde{e}_2), \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}_2, \tilde{\mu}_2, T_2, \varepsilon)$ with $(\tilde{v}_2, \tilde{\mu}_2) \in \mathcal{V}(\tau, \Sigma_2)$ and $(\tilde{\mu}_2, \Sigma_2) \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1)$; and, similarly, $(\rho(\tilde{e}_3), \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}_3, \tilde{\mu}_3, T_3, \varepsilon)$, with $(\tilde{v}_3, \tilde{\mu}_3) \in \mathcal{V}(\tau, \Sigma_3)$ and $(\tilde{\mu}_3, \Sigma_3) \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1)$. Let Ψ be a permutation that extends $\mathrm{id}_{\mathrm{dom}\,\Sigma_1}$ so that $\mathrm{dom}\,\Sigma_2 \cap \mathrm{dom}\,\Psi(\Sigma_3) = \mathrm{dom}\,\Sigma_1$. Then by Lemma 26, $(\Psi(\tilde{\mu}_3), \Psi(\Sigma_3)) \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1)$; and by Lemmas 8 and 11, $(\rho(\tilde{e}_3), \tilde{\mu}_1, \iota) \Downarrow (\Psi(\tilde{v}_3), \Psi(\tilde{\mu}_3), T_3, \varepsilon)$. We now define $\tilde{v}' = \Phi^{\iota}_1(\tilde{b}, \tilde{v}_2, \Psi(\tilde{v}_3))$; $\tilde{\mu}' = \Phi^{\iota}_1(\tilde{b}, \tilde{\mu}_2, \Psi(\tilde{\mu}_3))$; and $\Sigma' = \Sigma_2 \cup \Psi(\Sigma_3)$ (well-defined since $\Sigma_2, \Psi(\Sigma_3) \supseteq \Sigma_1$ and $\mathrm{dom}\,\Sigma_2 \cap \mathrm{dom}\,\Psi(\Sigma_3) \subseteq \mathrm{dom}\,\Sigma_1$). In addition, Lemma 31 gives $\tilde{v}' \in \mathcal{V}(\tau, \Sigma')$, and Lemma 27 gives $(\tilde{\mu}', \Sigma') \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1)$, as desired. Thus, the claim follows immediately from the conditional rule in the distributed semantics.

- Suppose $\Gamma, \Sigma, \mathrm{S} \vdash \tilde{e}_0\ \tilde{e}_1 : \tau$. Then $\Gamma, \Sigma, \mathrm{S} \vdash \tilde{e}_0 : (\tau_1 \xrightarrow{\mathrm{S}} \tau)^{\ell}$ and $\Gamma, \Sigma, \mathrm{S} \vdash \tilde{e}_1 : \tau_1$. Thus, the inductive hypothesis gives $(\rho(\tilde{e}_0), \tilde{\mu}, \iota) \Downarrow (\tilde{v}_0, \tilde{\mu}_0, T_0, \varepsilon)$, with $\tilde{v}_0 \in \mathcal{V}((\tau_1 \xrightarrow{\mathrm{S}} \tau)^{\ell}, \Sigma_0)$ and $(\tilde{\mu}_0, \Sigma_0) \succsim^{\iota} (\tilde{\mu}, \Sigma)$. Since $\Sigma_0 \supseteq \Sigma$, $\vdash \tilde{\mu}_0 : \Sigma$, and thus the inductive hypothesis also gives $(\rho(\tilde{e}_1), \tilde{\mu}_0, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, \tau_1, \varepsilon)$ with $\tilde{v}_1 \in \mathcal{V}(\tau_1, \Sigma_1)$ and $(\tilde{\mu}_1, \Sigma_1) \succsim^{\iota} (\tilde{\mu}_0, \Sigma_0) \succsim^{\iota} (\tilde{\mu}, \Sigma)$. The claim then follows directly from Lemma 34.

- Suppose $\Gamma, \Sigma, \mathrm{S} \vdash (\tilde{e}_1 := \tilde{e}_2) : \tau$, so that $\tau = \mathrm{unit}^{-}$. Then $\Gamma, \Sigma, \mathrm{S} \vdash \tilde{e}_1 : (Y^{\mathrm{S}}\ \mathrm{ref})^{\ell'}$, so the inductive hypothesis gives $(\rho(\tilde{e}_1), \tilde{\mu}, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T_1, \varepsilon)$ with $(\tilde{\mu}_1, \Sigma_1) \succsim^{\iota} (\tilde{\mu}, \Sigma)$ and $\tilde{v}_1 \in \mathcal{V}((Y^{\mathrm{S}}\ \mathrm{ref})^{\ell'}, \Sigma_1)$. In addition, we conclude by weakening (Lemma 1) that $\Gamma, \Sigma_1, \mathrm{S} \vdash \tilde{e}_2 : Y^{\ell}$, and thus the inductive hypothesis also gives $(\rho(\tilde{e}_2), \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}_2, \tilde{\mu}_2, T_2, \varepsilon)$ with $(\tilde{\mu}_2, \Sigma_2) \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1) \succsim^{\iota} (\tilde{\mu}, \Sigma)$ and $\tilde{v}_2 \in \mathcal{V}(Y^{\ell}, \Sigma_2)$. The claim then follows directly from Lemma 32 and the assignment rule in the distributed semantics.

- Suppose $\Gamma, \Sigma, \mathrm{S} \vdash !\tilde{e}_1 : \tau$. Then we have $\tau = Y^{\ell}$, and $\Gamma, \Sigma, \mathrm{S} \vdash \tilde{e} : (Y^{\ell'}\ \mathrm{ref})^{\ell''}$, with $\ell' \sqsubseteq \ell$ and $\ell'' \sqsubseteq \ell$. Thus, the inductive hypothesis gives $(\rho(\tilde{e}_1), \tilde{\mu}, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T_1, \mathcal{O}_1)$, with $(\tilde{\mu}_1, \Sigma_1) \succsim^{\iota} (\tilde{\mu}, \Sigma)$, and $\tilde{v}_1 \in \mathcal{V}((Y^{\ell'}\ \mathrm{ref})^{\ell''}, \Sigma_1)$. The claim then follows directly from Lemma 33 and the dereference rule in the distributed semantics.

- Suppose $\Gamma, \Sigma, \mathrm{S} \vdash \texttt{ref}\ \tilde{e}_1 : \tau$. Then $\tau = ((Y^{\ell'})\ \mathrm{ref})^{\ell}$ and $\Gamma, \Sigma, \mathrm{S} \vdash \tilde{e}_1 : Y^{\ell'}$. Thus, the inductive hypothesis gives $(\rho(\tilde{e}_1), \tilde{\mu}, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T_1, \mathcal{O}_1)$, with $(\tilde{\mu}_1, \Sigma_1) \succsim^{\iota} (\tilde{\mu}, \Sigma)$ and $\tilde{v}_1 \in \mathcal{V}(Y^{\ell'}, \Sigma_1)$. Pick $\tilde{a} \notin \mathrm{dom}\, \tilde{\mu}_1$, and set $\tilde{\mu}' = \tilde{\mu}_1[\tilde{a} \mapsto \tilde{v}_1]$, $\Sigma' = \Sigma[\tilde{a} \mapsto Y^{\ell'}$. Then by definition, $\tilde{a} \in \mathcal{V}((Y^{\ell'}\ \mathrm{ref})^{\ell}, \Sigma')$. Now, since $\tilde{\mu}'|_{\mathrm{dom}\, \Sigma_1} = \mathrm{id}$, we have $\tilde{\mu}' \succsim^{\iota}_{\Sigma_1} \tilde{\mu}_1$; and since $\tilde{v}_1 \in \mathcal{E}_{\ell'}(Y)$, we have $\vdash \tilde{\mu}' : \Sigma'$. Hence $(\tilde{\mu}', \Sigma') \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1) \succsim^{\iota} (\tilde{\mu}, \Sigma)$, and the claim follows directly from the reference rule in the distributed semantics.

- Suppose $\Gamma, \Sigma, \mathrm{S} \vdash \mathtt{op}_i^t : \tau$, so that $\tau = Y^{\ell}$, and for all $j \in \{1, \ldots, r\}$, we have $\Gamma, \Sigma, \mathrm{S} \vdash \tilde{e}_j : Y_j^{\ell_j}$, with $\ell_j \sqsubseteq \ell$ (and $\mathrm{op}_i : \prod_j Y_j \to Y$). Then after r successive applications of the inductive hypothesis, along with weakening (Lemmas 1 and 30), we obtain, for each j: $(\tilde{e}_j, \tilde{\mu}_{j-1}, \iota) \Downarrow (\tilde{v}_j, \tilde{\mu}_j, T_j, \mathcal{O}_j)$, with $(\tilde{\mu}_r, \Sigma_r) \succsim^{\iota} \ldots \succsim^{\iota} (\tilde{\mu}_0, \Sigma_0)$, and $\tilde{v}_j \in \mathcal{V}(Y_j^{\ell_j}, \Sigma_r)$ (where $\tilde{\mu}_0 = \tilde{\mu}$, $\Sigma_0 = \Sigma$). Setting $\tilde{\mu}' = \tilde{\mu}_r$ and $\Sigma' = \Sigma_r$, we note that by definition, $\tilde{v}_j \in \mathcal{E}_{\ell'_j}(Y_j)$ for some $\ell'_j \sqsubseteq \ell_j$, and thus, setting $\tilde{v}' = \mathrm{Enc}_{\ell'_1, \ldots, \ell'_r}(\mathrm{op}_i)(\tilde{v}_1, \ldots, \tilde{v}_r)$, we have $\tilde{v}' \in \mathcal{E}_{\sqcup_j \ell'_j}(Y)$. But $\sqcup_j \ell'_j \sqsubseteq \sqcup_j \ell_j$, so by definition, $\tilde{v}' \in \mathcal{V}(Y^{\ell}, \Sigma')$, and the claim follows directly from the primitive operation rule in the distributed semantics.

- Suppose $\Gamma, \Sigma, \mathrm{S} \vdash \lambda x.\tilde{e}' : (\tau_1 \xrightarrow{C} \tau_2)^{\mathrm{P}}$. We have $\rho(\lambda(x : \tau_1).\tilde{e}') = \lambda(x : \tau_1).\rho(\tilde{e}')$ (since we are reasoning with capture-avoiding substitution). So $(\lambda(x : \tau_1).\rho(\tilde{e}'), \tilde{\mu}, \iota) \Downarrow (\lambda(x : \tau_1).\rho(\tilde{e}'), \tilde{\mu}, \varepsilon, \varepsilon)$ and $(\tilde{\mu}, \Sigma) \succsim^{\iota} (\tilde{\mu}, \Sigma)$. Thus, it only remains to show that $\lambda(x : \tau_1).\rho(\tilde{e}') \in \mathcal{V}((\tau_1 \xrightarrow{C} \tau_2)^{\mathrm{P}})$. If $C = \mathrm{P}$, this condition is vacuous, so we assume $C = \mathrm{S}$, so that $\Gamma[x \mapsto \tau_1], \Sigma, \mathrm{S} \vdash \tilde{e}' : \tau_2$. Fix $\tilde{v}_1 \in \mathcal{V}(\tau_1, \Sigma_1)$, $(\tilde{\mu}_1, \Sigma_1) \succsim^{\iota} (\tilde{\mu}, \Sigma)$, and let $\rho' = \rho[x \mapsto \tilde{v}_1]$, $\Gamma' = \Gamma[x \mapsto \tau_1]$. Thus $\rho'(\tilde{e}') = \rho(\tilde{e}')[\tilde{v}_1/x]$, and $\rho'(x) \in \mathcal{V}(\Gamma'(x), \Sigma_1)$. In addition, by Lemma 9, $\mathrm{Addrs}(\rho'(x)) \subseteq \mathrm{dom}\, \Sigma_1$; and, by Lemma 30, $\rho'(y) \in \mathcal{V}(\Gamma'(y), \Sigma_1)$ for all $y \in \mathrm{dom}\, \Gamma$ (and, trivially, $\mathrm{Addrs}(\rho'(y)) \subseteq \mathrm{dom}\, \Sigma \subseteq \mathrm{dom}\, \Sigma_1$). Hence, by the inductive hypothesis, $(\rho(\tilde{e}')[\tilde{v}_1/x], \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \varepsilon)$, $\tilde{v}' \in \mathcal{V}(\tau_2, \Sigma')$, and $(\tilde{\mu}', \Sigma') \succsim^{\iota} (\tilde{\mu}_1, \Sigma_1)$, as desired.

- Suppose $\Gamma, \Sigma, \mathrm{S} \vdash \texttt{fix}\ f.\lambda x.\tilde{e}' : (\tau_1 \xrightarrow{\mathrm{P}} \tau_2)^{\mathrm{P}}$. We have $\rho(\texttt{fix}\ f.\lambda x.\tilde{e}') = \texttt{fix}\ f.\lambda x.\rho(\tilde{e}')$ (since we are reasoning with capture-avoiding substitution). But then $(\texttt{fix}\ f.\lambda x.\rho(\tilde{e}'), \tilde{\mu}, \iota) \Downarrow (\texttt{fix}\ f.\lambda x.\rho(\tilde{e}'), \tilde{\mu}, \varepsilon, \varepsilon)$, and trivially $\texttt{fix}\ f.\lambda x.\rho(\tilde{e}') \in \mathcal{V}((\tau_1 \xrightarrow{\mathrm{P}} \tau_2)^{\mathrm{P}}, \Sigma)$, as desired.

- Suppose $\Gamma, \Sigma, \mathrm{S} \vdash \varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3) : \tau$, so that $\Gamma, \Sigma, \mathrm{S} \vdash \tilde{b} : \mathrm{bool}^{\mathrm{S}}$, $\Gamma, \Sigma, \mathrm{S} \vdash \tilde{v}_2 : t^{\ell_2}$, and $\Gamma, \Sigma, \mathrm{S} \vdash \tilde{v}_3 : t^{\ell_3}$, with $\tau = t^{\ell}$; $\ell_2, \ell_3 \sqsubseteq \ell$; and either $t = (\tau_1 \to \tau_2)$ or $t = \tau_1\ \mathrm{ref}$. By definition, $\tilde{b}$, $\tilde{v}_2$, and $\tilde{v}_3$ are values, so by Lemma 4, $\rho(\varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3)) = \varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3)$; and, in addition, $(\tilde{v}_2, \tilde{\mu}, \iota) \Downarrow (\tilde{v}_2, \tilde{\mu}, \varepsilon, \varepsilon)$; $(\tilde{v}_3, \tilde{\mu}, \iota) \Downarrow (\tilde{v}_3, \tilde{\mu}, \varepsilon, \varepsilon)$; and $(\varphi(\tilde{b}, \tilde{v}_2, tv_3), \tilde{\mu}, \iota) \Downarrow (\varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3), \tilde{\mu}, \varepsilon, \varepsilon)$. Thus, applying the rule for phi in the distributed semantics, it only remains to show that $\varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3) \in \mathcal{V}(\tau, \Sigma)$. Now, the inductive hypothesis gives $(\tilde{v}_2, \tilde{\mu}, \iota) \Downarrow (\tilde{v}'_2, \tilde{\mu}_2, T_2, \mathcal{O}_2)$, with $\tilde{v}'_2 \in \mathcal{V}(t^{\ell_2}, \Sigma_2)$, $\Sigma_2 \supseteq \Sigma$, $\tilde{v}'_3 \in \mathcal{V}(t_3^{\ell_3}, \Sigma_3)$, and $\Sigma_3 \supseteq \Sigma$. But determinism (Lemma 19) gives that $\tilde{v}'_2 = \Psi_2(\tilde{v}_2)$ and $\tilde{v}'_3 = \Psi_3(\tilde{v}_3)$, with $\Psi_2|_{\mathrm{dom}\, \tilde{\mu}} = \mathrm{id}$ and $\Psi_3|_{\mathrm{dom}\, \tilde{\mu}} = \mathrm{id}$. In addition, Lemma 8 gives $\mathrm{Addrs}(\tilde{v}_2) \subseteq \mathrm{dom}\, \Sigma \subseteq \mathrm{dom}\, \tilde{\mu}$ and $\mathrm{Addrs}(\tilde{v}_3) \subseteq \mathrm{dom}\, \Sigma \subseteq \mathrm{dom}\, \tilde{\mu}$, and thus $\tilde{v}'_2 = \tilde{v}_2 \in \mathcal{V}(t^{\ell_2}, \Sigma)$, $\tilde{v}'_3 = \tilde{v}_3 \in \mathcal{V}(t_3^{\ell_3}, \Sigma)$. The claim now follows by definition of the reducibility predicate.

- Suppose $\Gamma, \Sigma, \mathrm{S} \vdash a : \tau$, so that $\tau = (Y^{\ell'}\ \mathrm{ref})^{\ell}$ and $\Sigma(a) = Y^{\ell'}$. Then $(a, \tilde{\mu}, \iota) \Downarrow (a, \tilde{\mu}, \varepsilon, \varepsilon)$, so the claim follows by definition of the reducibility predicate.

- Suppose $\Gamma, \Sigma, \mathrm{S} \vdash \tilde{y} : \tau$, so that $\tau = Y^{\ell}$ and $\tilde{y} \in \mathcal{E}_{\ell'}(Y)$ with $\ell' \sqsubseteq \ell$. Then $(\tilde{y}, \tilde{\mu}, \iota) \Downarrow (\tilde{y}, \tilde{\mu}, \varepsilon, \varepsilon)$, so the claim follows by definition of the reducibility predicate.

- Suppose $\Gamma, \Sigma, \mathrm{S} \vdash x : \tau$. Then $\Gamma(x) = \tau$, and hence $\rho(x) \in \mathcal{V}(\tau, \Sigma)$. By inspection, all cases for $\mathcal{V}(\tau, \Sigma)$ are values, so that $(\rho(x), \tilde{\mu}, \iota) \Downarrow (\rho(x), \tilde{\mu}, \varepsilon, \varepsilon)$, and the claim is immediate.

□

Lemma 36 (Similarity is Reflexive for Surface Expressions)**.** *For any expression e (in the surface language),* $(e, \{\}) \sim^{\iota}_{\emptyset} (e, \{\})$.

Proof. By induction on the structure of the expression e. □

Lemma 37 (Permutation Weakening for Similarity Relation)**.**

- *If* $e \sim^{\iota}_{\Psi} \tilde{e}$ *and* $\Psi' \supseteq \Psi$ *then* $e \sim^{\iota}_{\Psi'} \tilde{e}$.
- *If* $\mu \sim^{\iota}_{\Psi} \tilde{\mu}$ *and* $\Psi' \supseteq \Psi$ *then* $\mu \sim^{\iota}_{\Psi'} \tilde{\mu}$.

Proof. The first claim follows by induction on the similarity relation $e \sim^{\iota}_{\Psi} \tilde{e}$; the second claim then follows immediately by applying the first claim for each address $a \in \operatorname{dom} \mu$. □

Lemma 38 (Substitution for Similarity Relation)**.** *If* $e \sim^{\iota}_{\Psi} \tilde{e}$ *and* $v \sim^{\iota}_{\Psi} \tilde{v}$*, then* $e[v/x] \sim^{\iota}_{\Psi} \tilde{e}[\tilde{v}/x]$.

Proof. By induction on the similarity relation $e \sim^{\iota}_{\Psi} \tilde{e}$. □

Lemma 39 (Join of Values Preserves Equivalence)**.**

- *If* $\tilde{b} \in \hat{\mathcal{E}}^{\iota}_{\ell}(\texttt{true})$ *and* $v \sim^{\iota}_{\Psi} \tilde{v}_2$*, then* $v \sim^{\iota}_{\Psi} \Phi^{\iota}_1(\tilde{b}, \tilde{v}_2, \tilde{v}_3)$.
- *If* $\tilde{b} \in \hat{\mathcal{E}}^{\iota}_{\ell}(\texttt{false})$ *and* $v \sim^{\iota}_{\Psi} \tilde{v}_3$*, then* $v \sim^{\iota}_{\Psi} \Phi^{\iota}_1(\tilde{b}, \tilde{v}_2, \tilde{v}_3)$.

Proof. By case analysis on the similarity relation $v \sim^{\iota}_{\Psi} \tilde{v}_2$ (resp., $v \sim^{\iota}_{\Psi} \tilde{v}_3$). □

Lemma 40 (Join of Stores Preserves Equivalence)**.**

- *If* $\mu \sim^{\iota}_{\Psi} \tilde{\mu}_2$*,* $\tilde{\mu}' = \Phi^{\iota}_1(\tilde{b}, \tilde{\mu}_2, \tilde{\mu}_3)$*, and* $\tilde{b} \in \mathcal{E}_{\ell_1}(\texttt{true})$*, then* $\mu \sim^{\iota}_{\Psi} \tilde{\mu}'$.
- *If* $\mu \sim^{\iota}_{\Psi} \tilde{\mu}_3$*,* $\tilde{\mu}' = \Phi^{\iota}_1(\tilde{b}, \tilde{\mu}_2, \tilde{\mu}_3)$*, and* $\tilde{b} \in \mathcal{E}_{\ell_1}(\texttt{false})$*, then* $\mu \sim^{\iota}_{\Psi} \tilde{\mu}'$.

Proof. Follows by applying Lemma 39 for each address in $\operatorname{dom} \tilde{\mu}_2 \cap \operatorname{dom} \tilde{\mu}_3$ (and follows by definition for other addresses). □

Lemma 41 (Correctness for Applications of Related Values)**.** *If* $\lambda x.e'_0 \sim^{\iota}_{\Psi_0} \tilde{v}_0$*, then for all* $\tilde{e}_0$ *and* $\tilde{e}_1$*, if the following conditions are satisfied:*

- $(\tilde{e}_0, \tilde{\mu}, \iota) \Downarrow (\tilde{v}_0, \tilde{\mu}_0, T_0, \mathcal{O}_0)$
- $(\tilde{e}_1, \tilde{\mu}_0, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T_1, \mathcal{O}_1)$
- $\emptyset, \Sigma_0, C \vdash \tilde{v}_0 : (\tau_1 \xrightarrow{C} \tau)^{\ell}$
- $\emptyset, \Sigma_1, C \vdash \tilde{v}_1 : \tau_1$
- $v_1 \sim^{\iota}_{\Psi_1} \tilde{v}_1$
- *For any* $\tilde{e}$*, if* $e'_0[v_1/x] \sim^{\iota}_{\Psi_1} \tilde{e}$ *and* $\emptyset, \Sigma_1, C \vdash \tilde{e} : \tau$*, then* $(\tilde{e}, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \mathcal{O}_2)$*,* $v' \sim^{\iota}_{\Psi'} \tilde{v}'$*,* $\mu' \sim^{\iota}_{\Psi'} \tilde{\mu}'$*, and* $\Psi' \supseteq \Psi_1$.

then $(\tilde{e}_0\ \tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T', \mathcal{O}_0 \| \mathcal{O}_1 \| \mathcal{O}_2)$*,* $v' \sim^{\iota}_{\Psi'} \tilde{v}'$*,* $\mu' \sim^{\iota}_{\Psi'} \tilde{\mu}'$*,* $\Psi' \supseteq \Psi_1$*; and whenever* $\emptyset, \Sigma_0, C \vdash \tilde{v}_0 : (\tau_1 \xrightarrow{\mathrm{S}} \tau)^{\ell}$*,* $\mathcal{O}_2 = \varepsilon$ *and* $\tilde{\mu}' \succsim^{\iota}_{\Sigma_1} \tilde{\mu}_1$.

Proof. By induction on the similarity relation $\lambda x.e'_0 \sim^{\iota}_{\Psi_0} \tilde{v}_0$.

- Suppose $\tilde{v}_0 = \varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3)$. Then $\tilde{b} \in \hat{\mathcal{E}}^\iota_{\ell_1}(\texttt{true})$ or $\tilde{b} \in \hat{\mathcal{E}}^\iota_{\ell_1}(\texttt{false})$; without loss of generality we assume the former, so that $\lambda x.e'_0 \sim^\iota_{\Psi_0} \tilde{v}_2$. Since $\emptyset, \Sigma_0, C \vdash \tilde{v}_0 : (\tau_1 \xrightarrow{C} \tau)^\ell$, and $\tilde{v}_0 = \varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3)$, we must have $C = \mathrm{S}$, and also $\emptyset, \Sigma_0, \mathrm{S} \vdash \tilde{v}_2 : (\tau_1 \xrightarrow{\mathrm{S}} \tau)^\ell$ and $\emptyset, \Sigma_0, \mathrm{S} \vdash \tilde{v}_3 : (\tau_1 \xrightarrow{\mathrm{S}} \tau)^\ell$. We now invoke the inductive hypothesis on the expressions $\tilde{v}_2$ and $\tilde{v}_1$ (by inspection, the first two conditions are satisfied, since $\tilde{v}_2$ and $\tilde{v}_1$ are values; the third was shown above; and the remaining conditions are unchanged from the premises). Thus, we conclude that $(\tilde{v}_2\ \tilde{v}_1, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}'_2, \tilde{\mu}'_2, T'_2, \varepsilon)$, with $v' \sim^\iota_{\Psi_2} \tilde{v}'_2$, $\mu' \sim^\iota_{\Psi_2} \tilde{\mu}'_2$, $\Psi_2 \supseteq \Psi_1$, and $\tilde{\mu}'_2 \succsim^\iota_{\Sigma_1} \tilde{\mu}_1$. On the other hand, we note that by Lemma 35, we have $\tilde{v}_3 \in \mathcal{V}(\tau_1 \xrightarrow{\mathrm{S}} \tau, \Sigma_0)$; and by Lemmas 35 and 5, $\tilde{v}_1 \in \mathcal{V}(\tau_1, \Sigma_1)$ (since $\tilde{v}_1$ is a value). Thus, we can also invoke Lemma 34 to conclude that $(\tilde{v}_3\ \tilde{v}_1, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}'_3, \tilde{\mu}'_3, T'_3, \varepsilon)$, with $(\tilde{\mu}_3, \Sigma_3) \succsim^\iota (\tilde{\mu}_1, \Sigma_1)$. Let Ψ'_3 be a permutation that extends $\mathrm{id}_{\mathrm{dom}\,\Sigma_1}$ so that $\Psi'_3(a) \notin \mathrm{dom}\,\Sigma_2$ for any $a \in \mathrm{dom}\,\Sigma_3$ (i.e., $\mathrm{dom}\,\Sigma_2 \cap \mathrm{dom}\,\Psi'_3(\Sigma_3) = \mathrm{dom}\,\Sigma_1$). Then by Lemma 26, $(\Psi'_3(\tilde{\mu}_3), \Psi'_3(\Sigma_3)) \succsim^\iota (\tilde{\mu}_1, \Sigma_1)$; and by Lemmas 8 and 11, $(\tilde{v}_3\ \tilde{v}_1, \tilde{\mu}_1, \iota) \Downarrow (\Psi'_3(\tilde{v}'_3), \Psi'_3(\tilde{\mu}'_3), T'_3, \varepsilon)$. Let $\Psi' = \Psi_2$, $\tilde{v}' = \Phi^\iota_1(\tilde{b}, \tilde{v}'_2, \tilde{v}'_3)$, and $\tilde{\mu}' = \Phi^\iota_1(\tilde{b}, \tilde{\mu}'_2, \tilde{\mu}'_3)$. Lemma 39 then gives $v' \sim^\iota_{\Psi'} \tilde{v}'$, while Lemmas 27 and 40 give $\mu' \sim^\iota_{\Psi'} \tilde{\mu}'$. The claim then follows immediately from the distributed semantics rule for application of phi.

- Suppose $\tilde{v}_0 = \lambda x.\tilde{e}'_0$. Then by definition, $e'_0 \sim^\iota_{\Psi_0} \tilde{e}'_0$, so that Lemma 38 gives $e_0[v_1/x] \sim^\iota_{\Psi_0} \tilde{e}'_0[\tilde{v}_1/x]$. In addition, by substitution (Lemma 2) and weakening (Lemma 1), we have $\emptyset, \Sigma_1, C \vdash \tilde{e}'_0[\tilde{v}_1/x] : \tau$. Thus, using the final condition given in the hypothesis, and instantiating $\tilde{e}$ with $\tilde{e}'_0[\tilde{v}_1/x]$, we conclude that $(\tilde{e}'_0[\tilde{v}_1/x], \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}', \tilde{\mu}, T, \mathcal{O}_2)$, with $v' \sim^\iota_{\Psi'} \tilde{v}'$, $\mu' \sim^\iota_{\Psi'} \tilde{\mu}'$, and $\Psi' \supseteq \Psi_1$. So it only remains to show the final clause of the conclusion. Suppose $\emptyset, \Sigma_0, C \vdash \tilde{v}_0 : (\tau_1 \xrightarrow{\mathrm{S}} \tau)^\ell$. Then by definition, $\{x \mapsto \tau_1\}, \Sigma_0, \mathrm{S} \vdash \tilde{e}'_0 : \tau$. But by Lemma 5, we also have $\emptyset, \Sigma_1, \mathrm{S} \vdash \tilde{v}_1 : \tau_1$, and hence by substitution (Lemma 2) and weakening (Lemma 1), $\emptyset, \Sigma_1, \mathrm{S} \vdash \tilde{e}'_0[\tilde{v}_1/x] : \tau$. Thus, purity (Lemma 35) along with determinism (Lemma 19) gives $\mathcal{O}_2 = \varepsilon$ and $\Psi'_2(\tilde{\mu}') \succsim^\iota \tilde{\mu}_1$ for some Ψ'_2 such that $\Psi'_2|_{\mathrm{dom}\,\tilde{\mu}_1} = \mathrm{id}$. The claim then follows directly from Lemma 26.

□

Lemma 42 (Correctness of Distributed Store Update)**.** *If $a \sim^\iota_\Psi \tilde{v}$, $\mu \sim^\iota_\Psi \tilde{\mu}$, $v' \in Y$, $\tilde{v}' \in \hat{\mathcal{E}}^\iota_\ell(v')$, and $\tilde{\mu}' = \mathrm{update}^\iota_1(\tilde{\mu}, \tilde{v}, \tilde{v}')$, then $\mu[a \mapsto v'] \sim^\iota_\Psi \tilde{\mu}'$.*

Proof. By induction on the similarity relation $a \sim^\iota_\Psi \tilde{v}$.

- Suppose $\tilde{v} = \varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3)$ with $\tilde{b} \in \hat{\mathcal{E}}^\iota_{\ell_1}(\mathrm{bool})$. Without loss of generality, we assume $\tilde{b} \in \hat{\mathcal{E}}^\iota_{\ell_1}(\texttt{true})$, so that $a \sim^\iota_\Psi \tilde{v}_2$. Then, by the inductive hypothesis, we have $\mu[a \mapsto v'] \sim^\iota_\Psi \mathrm{update}^\iota_1(\tilde{\mu}, \tilde{v}_2, \tilde{v}')$. But by definition, $\mathrm{update}^\iota_1(\tilde{\mu}, \tilde{v}, \tilde{v}') = \Phi^\iota_1(\tilde{b}, \mathrm{update}^\iota_1(\tilde{\mu}, \tilde{v}_2, \tilde{v}'), \mathrm{update}^\iota_1(\tilde{\mu}, \tilde{v}_3, \tilde{v}'))$, so the claim follows immediately from Lemma 40.

- Suppose $\tilde{v} = \Psi(a)$. Then $\mathrm{update}^\iota_1(\tilde{\mu}, \tilde{v}, \tilde{v}') = \tilde{\mu}[\Psi(a) \mapsto \tilde{v}']$, so the claim follows by definition of similarity of stores.

□

Lemma 43 (Correctness of Distributed Store Select)**.** *If $a \sim^\iota_\Psi \tilde{v}$, $\mu \sim^\iota_\Psi \tilde{\mu}$, and $\tilde{v}' = \mathrm{select}^\iota_1(\tilde{\mu}, \tilde{v})$, then $\mu(a) \sim^\iota_\Psi \tilde{v}'$.*

Proof. By induction on the similarity relation $a \sim^\iota_\Psi \tilde{v}$.

- Suppose $\tilde{v} = \varphi(\tilde{b}, \tilde{v}_2, \tilde{v}_3)$ with $\tilde{b} \in \hat{\mathcal{E}}^\iota_{\ell_1}(\mathrm{bool})$. Without loss of generality, we assume $\tilde{b} \in \hat{\mathcal{E}}^\iota_{\ell_1}(\texttt{true})$, so that $a \sim^\iota_\Psi \tilde{v}_2$. Then, by the inductive hypothesis, we have $\mu(a) \sim^\iota_\Psi \mathrm{select}^\iota_1(\tilde{\mu}, \tilde{v}_2)$. But by definition, $\mathrm{select}^\iota_1(\tilde{\mu}, \tilde{v}) = \Phi^\iota_1(\tilde{b}, \mathrm{select}^\iota_1(\tilde{\mu}, \tilde{v}_2), \mathrm{select}^\iota_1(\tilde{\mu}, \tilde{v}_3))$, so the claim follows immediately from Lemma 39.

- Suppose $\tilde{v} = \Psi(a)$. Then $\mathrm{select}_1^\iota(\tilde{\mu}, \tilde{v}) = \tilde{\mu}(\Psi(a))$, so the claim follows by definition of similarity of addresses.

□

Lemma 44 (Correctness (Generalized))**.** *If* $(e, \mu) \downarrow (v', \mu', \mathcal{O})$*,* $(e, \mu) \sim^\iota_\Psi (\tilde{e}, \tilde{\mu})$*,* $\vdash \tilde{\mu} : \Sigma$*, and* $\emptyset, \Sigma, C \vdash \tilde{e} : \tau$*, then* $(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}', \tilde{\mu}', T, \mathcal{O})$*,* $(v', \mu') \sim^\iota_{\Psi'} (\tilde{v}', \tilde{\mu}')$*, and* $\Psi' \supseteq \Psi$*.*

Proof. By induction on the evaluation derivation $(e, \mu) \downarrow (v', \mu', \mathcal{O})$.

- Suppose $(\texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3, \mu) \downarrow (v', \mu', \mathcal{O})$, so that $\tilde{e} = \texttt{if}\ \tilde{e}_1\ \texttt{then}\ \tilde{e}_2\ \texttt{else}\ \tilde{e}_3$, with $e_1 \sim^\iota_\Psi \tilde{e}_1$, $e_2 \sim^\iota_\Psi \tilde{e}_2$, and $e_3 \sim^\iota_\Psi \tilde{e}_3$. Then $(e_1, \mu) \downarrow (v_1, \mu_1, \mathcal{O}_1)$, and $\emptyset, \Sigma, C \vdash \tilde{e}_1 : \mathrm{bool}^\ell$. Thus, by the inductive hypothesis, $(\tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T_1, \mathcal{O}_1)$, with $(v_1, \mu_1) \sim^\iota_{\Psi_1} (\tilde{v}_1, \tilde{\mu}_1)$ and $\Psi_1 \supseteq \Psi$. In addition, by preservation (Lemma 24), we have $\vdash \tilde{\mu}_1 : \Sigma_1$, $\Sigma_1 \supseteq \Sigma$, and $\emptyset, \Sigma_1, C \vdash \tilde{v}_1 : \mathrm{bool}^\ell$. So for some $\ell' \sqsubseteq \ell$, either $\tilde{v}_1 \in \mathcal{E}_{\ell'}(\texttt{true})$ or $\tilde{v}_1 \in \mathcal{E}_{\ell'}(\texttt{false})$; without loss of generality we assume the former. Since $v_1 \sim^\iota_{\Psi_1} \tilde{v}_1$, $v_1 = \texttt{true}$ also. So $(e_2, \mu_1) \downarrow (v', \mu', \mathcal{O}_2)$ and $\mathcal{O} = \mathcal{O}_1 \| \mathcal{O}_2$. By weakening of typing (Lemma 1), $\Gamma, \Sigma_1, C \vdash \tilde{e}_2 : \tau$; and by weakening of the similarity relation (Lemma 37), $e_2 \sim^\iota_{\Psi_1} \tilde{e}_2$. Thus, the inductive hypothesis gives $(\tilde{e}_2, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}_2, \tilde{\mu}_2, T_2, \mathcal{O}_2)$ with $(v', \mu') \sim^\iota_{\Psi_2} (\tilde{v}_2, \tilde{\mu}_2)$ and $\Psi_2 \supseteq \Psi_1 \supseteq \Psi$. Now, we have the following cases:

  - Suppose $\tilde{v}_1 \in \mathcal{E}_{\mathrm{P}}(\texttt{true})$. Setting $\Psi' = \Psi_2$, $(\tilde{v}', \tilde{\mu}') = (\tilde{v}_2, \tilde{\mu}_2)$, the claim follows immediately from the public conditional rule for the distributed semantics.
  - Suppose $\tilde{v}_1 \in \mathcal{E}_{\mathrm{S}}(\texttt{true})$, so that $\ell = \mathrm{S}$. By preservation (Lemma 24), $\vdash \tilde{\mu}_2 : \Sigma_2$, $\Sigma_2 \supseteq \Sigma_1$, and $\emptyset, \Sigma_2, C \vdash \tilde{v}_2 : \tau$. Since $\emptyset, \Sigma, C \vdash \tilde{v}_1 : \mathrm{bool}^{\mathrm{S}}$, we have $\emptyset, \Sigma, \mathrm{S} \vdash \tilde{e}_2 : \tau$ and $\emptyset, \Sigma, \mathrm{S} \vdash \tilde{e}_3 : \tau$ (and thus, again by weakening (Lemma 1), we have $\emptyset, \Sigma_1, \mathrm{S} \vdash \tilde{e}_2 : \tau$ and $\emptyset, \Sigma_1, \mathrm{S} \vdash \tilde{e}_3 : \tau$). Now, by purity (Lemma 35), $(\tilde{e}_2, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}_2', \tilde{\mu}_2', T_2', \varepsilon)$, $(\tilde{\mu}_2', \Sigma_2') \precsim^\iota (\tilde{\mu}_1, \Sigma_1)$. But by determinism (Lemma 19), there is some permutation Ψ_2' with $\Psi_2'|_{\mathrm{dom}\,\Sigma_1} = \mathrm{id}$ and $\tilde{\mu}_2' = \Psi_2'(\tilde{\mu}_2)$; and, invoking Lemma 25 with the permutation $(\Psi_2')^{-1}$ (also the identity on $\mathrm{dom}\,\Sigma_1$), we obtain $\tilde{\mu}_2 \precsim^\iota_{\Sigma_1} \tilde{\mu}_1$ (and thus, from the above, $(\tilde{\mu}_2, \Sigma_2) \precsim^\iota (\tilde{\mu}_1, \Sigma_1)$). In addition, again by purity (Lemma 35), we have $(\tilde{e}_3, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}_3, \tilde{\mu}_3, T_3, \varepsilon)$ with $(\tilde{\mu}_3, \Sigma_3) \precsim^\iota (\tilde{\mu}_1, \Sigma_1)$; and, by preservation (Lemma 24), $\emptyset, \Sigma_3, C \vdash \tilde{v}_3 : \tau$, $\vdash \tilde{\mu}_3 : \Sigma_3$, and $\Sigma_3 \supseteq \Sigma_1$. Let Ψ_3 be a permutation that extends $\mathrm{id}_{\mathrm{dom}\,\Sigma_1}$ so that $\Psi_3(a) \notin \mathrm{dom}\,\Sigma_2$ for any $a \in \mathrm{dom}\,\Sigma_3$ (i.e., $\mathrm{dom}\,\Sigma_2 \cap \mathrm{dom}\,\Psi_3(\Sigma_3) = \mathrm{dom}\,\Sigma_1$). Then by Lemma 26, $(\Psi_3(\tilde{\mu}_3), \Psi_3(\Sigma_3)) \precsim^\iota (\tilde{\mu}_1, \Sigma_1)$; by Lemma 10, $\emptyset, \Psi_3(\Sigma_3), C \vdash \Psi_3(\tilde{v}_3) : \tau$; and by Lemma 11, $(\tilde{e}_3, \tilde{\mu}_1, \iota) \Downarrow (\Psi_3(\tilde{v}_3), \Psi_3(\tilde{\mu}_3), T_3, \varepsilon)$. Let $\Psi' = \Psi_2$, $\tilde{v}' = \Phi_1^\iota(\tilde{v}_1, \tilde{v}_2, \tilde{v}_3)$, and $\tilde{\mu}' = \Phi_1^\iota(\tilde{v}_1, \tilde{\mu}_2, \tilde{\mu}_3)$. Lemma 39 then gives $v' \sim^\iota_{\Psi'} \tilde{v}'$, while Lemmas 27 and 40 gives $\mu' \sim^\iota_{\Psi'} \tilde{\mu}'$. The claim then follows from the secret conditional rule for the distributed semantics.

- Suppose $(e_0\ e_1, \mu) \downarrow (v', \mu', \mathcal{O})$. Then $\tilde{e} = \tilde{e}_0\ \tilde{e}_1$, with $e_0 \sim^\iota_\Psi \tilde{e}_0$ and $e_1 \sim^\iota_\Psi \tilde{e}_1$. Since $\emptyset, \Sigma, C \vdash \tilde{e} : \tau$, we have $\emptyset, \Sigma, C \vdash \tilde{e}_0 : (\tau_1 \xrightarrow{C} \tau)^\ell$, with $\ell \sqsubseteq C$, and $\emptyset, \Sigma, C \vdash \tilde{e}_1 : \tau_1$. Now, $(e_0, \mu) \downarrow (v_0, \mu_0, \mathcal{O}_0)$ so that by the inductive hypothesis, $(\tilde{e}_0, \tilde{\mu}, \iota) \Downarrow (\tilde{v}_0, \tilde{\mu}_0, T_0, \mathcal{O}_0)$, with $v_0 \sim^\iota_{\Psi_0} \tilde{v}_0$; $\mu_0 \sim^\iota_{\Psi_0} \tilde{\mu}_0$; $\vdash \tilde{\mu}_0 : \Sigma_0$; and $\Psi_0 \supseteq \Psi$. By weakening of typing (Lemma 1), $\emptyset, \Sigma_1, C \vdash \tilde{e}_1 : \tau_1$; and by weakening of the similarity relation (Lemma 37), $e_1 \sim^\iota_{\Psi_0} \tilde{v}_1$. Thus, since $(e_1, \mu_0) \downarrow (v_1, \mu_1, \mathcal{O}_1)$, the inductive hypothesis gives $(\tilde{e}_1, \tilde{\mu}_0, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T_1, \mathcal{O}_1)$ with $v_1 \sim^\iota_{\Psi_1} \tilde{v}_1$; $\mu_1 \sim^\iota_{\Psi_1} \tilde{\mu}_1$; $\vdash \tilde{\mu}_1 : \Sigma_1$; and $\Psi_1 \supseteq \Psi_0 \supseteq \Psi$. In addition, preservation (Lemma 24) gives $\emptyset, \Sigma_0, C \vdash \tilde{v}_0 : (\tau_1 \xrightarrow{C} \tau)^\ell$ and $\emptyset, \Sigma_1, C \vdash \tilde{v}_1 : \tau_1$. We now consider the following cases:

  - $v_0 = \lambda x.e_0'$. We then have $(e_0'[v_1/x], \mu_1) \downarrow (v', \mu', \mathcal{O}_2)$ and $\mathcal{O} = \mathcal{O}_0 \| \mathcal{O}_1 \| \mathcal{O}_2$. Thus, the claim follows directly from Lemma 41 (supplying its final condition with the inductive hypothesis applied to $e_0'[v_1/x]$).

– $v_0 = \texttt{fix}\ f.\lambda x.e_0'$. In this case, $(e_0'[(\texttt{fix}\ f.\lambda x.e_0')/f, v_1/x], \mu_1) \downarrow (v', \mu', \mathcal{O}_2)$ and $\mathcal{O} = \mathcal{O}_0 \| \mathcal{O}_1 \| \mathcal{O}_2$. Now, by definition, we have $e_0' \sim^{\iota}_{\Psi_0} \tilde{e}_0'$, so that Lemma 38 gives $e_0[(\texttt{fix}\ f.\lambda x.e_0')/f, v_1/x] \sim^{\iota}_{\Psi_0} \tilde{e}_0'[(\texttt{fix}\ f.\lambda x.\tilde{e}_0')/f, \tilde{v}_1/x]$. In addition, substitution (Lemma 2) and weakening (Lemma 1) give $\emptyset, \Sigma_1, C \vdash \tilde{e}_0'[(\texttt{fix}\ f.\lambda x.\tilde{e}_0')/f, \tilde{v}_1/x] : \tau$. The claim then follows immediately by the inductive hypothesis.

- Suppose $(e_1 := e_2, \mu) \downarrow (v', \mu', \mathcal{O})$. Then $(e_1, \mu) \downarrow (a, \mu_1, \mathcal{O}_2)$, $(e_2, \mu_1) \downarrow (v_2, \mu_2, \mathcal{O}_2)$, $\tilde{\mu}' = \mu_2[a \mapsto v_2]$, $v' = ()$, and $\mathcal{O} = \mathcal{O}_1 \| \mathcal{O}_2$. We also have $\tilde{e} = (\tilde{e}_1 := \tilde{e}_2)$, with $e_1 \sim^{\iota}_{\Psi} \tilde{e}_1$ and $e_2 \sim^{\iota}_{\Psi} \tilde{e}_2$. Inversion on the typing judgment gives $\emptyset, \Sigma, C \vdash \tilde{e}_1 : (Y^{\ell}\ \mathrm{ref})^{\ell'}$ and $\emptyset, \Sigma, C \vdash \tilde{e}_2 : Y^{\ell}$, so that by the inductive hypothesis, $(\tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T_1, \mathcal{O}_1)$ with $a \sim^{\iota}_{\Psi_1} \tilde{v}_1$, $\mu_1 \sim^{\iota}_{\Psi_1} \tilde{\mu}_1$, and $\Psi_1 \supseteq \Psi$. In addition, preservation (Lemma 24) gives $\vdash \tilde{\mu}_1 : \Sigma_1$, $\Sigma_1 \supseteq \Sigma$, and $\emptyset, \Sigma_1, C \vdash \tilde{v}_1 : (Y^{\ell}\ \mathrm{ref})^{\ell'}$. By weakening of typing (Lemma 1), we have $\emptyset, \Sigma_1, C \vdash \tilde{e}_2 : Y^{\ell}$, and by weakening of the similarity relation (Lemma 37), we have $e_1 \sim^{\iota}_{\Psi_1} \tilde{e}_1$ and $\mu_1 \sim^{\iota}_{\Psi_1} \tilde{\mu}_1$. Thus, the inductive hypothesis also gives $(\tilde{e}_2, \tilde{\mu}_1, \iota) \Downarrow (\tilde{v}_2, \tilde{\mu}_2, T_2, \mathcal{O}_2)$ with $v_2 \sim^{\iota}_{\Psi_2} \tilde{v}_2$, $\mu_2 \sim^{\iota}_{\Psi_2} \tilde{\mu}_2$, and $\Psi_2 \supseteq \Psi_1 \supseteq \Psi$. Further, by preservation (Lemma 24), $\vdash \tilde{\mu}_2 : \Sigma_2$, $\Sigma_2 \supseteq \Sigma$, and $\emptyset, \Sigma_2, C \vdash \tilde{v}_2 : (Y^{\ell}\ \mathrm{ref})^{\ell'}$. Hence by definition, $v_2 \in Y$ and $\tilde{v}_2 \in \hat{\mathcal{E}}^{\iota}_{\ell''}(v_2)$ for $\ell'' \sqsubseteq \ell$. In addition, again by weakening (Lemmas 1 and 37), we have $a \sim^{\iota}_{\Psi_1} \tilde{v}_2$, and $\emptyset, \Sigma_2, C \vdash \tilde{v}_1 : (Y^{\ell}\ \mathrm{ref})^{\ell'}$. Now, we consider the following cases:

  – Suppose $\ell' = \mathrm{P}$. In this case, the typing judgment gives that $\tilde{v}_1 = \tilde{a}$ for some address $\tilde{a}$; and the equivalence judgment gives that $\tilde{a} = \Psi_1(a) = \Psi_2(a)$. Thus, $\mathrm{update}^{\iota}(\tilde{\mu}_2, \tilde{v}_1, \tilde{v}_2) = (\tilde{\mu}_2[\tilde{a} \mapsto \tilde{v}_2], \varepsilon)$, and the claim follows immediately from the assignment rule in the distributed semantics.
  – Suppose $\ell' = \mathrm{S}$. Then $C = \mathrm{S}$, and so $\ell = \mathrm{S}$, so that $\Sigma_2(\tilde{a}) = \mathrm{S}$, and $\tilde{v}_2 \in \hat{\mathcal{E}}^{\iota}_{\mathrm{S}}(v_2)$. The claim now follows immediately from Lemma 42 and the assignment rule in the distributed semantics.

- Suppose $(!e, \mu) \downarrow (v', \mu', \mathcal{O})$, so that $(e, \mu) \downarrow (a, \mu', \mathcal{O})$ and $v' = \mu'(a)$. The equivalence judgment gives $\tilde{e}' = !\tilde{e}$, with $e \sim^{\iota}_{\Psi} \tilde{e}$. In addition, the typing judgment gives $\emptyset, \Sigma, C \vdash \tilde{e} : (Y^{\ell}\ \mathrm{ref})^{\ell'}$, so that by the inductive hypothesis, $(\tilde{e}, \tilde{\mu}, \iota) \Downarrow (\tilde{v}, \tilde{\mu}', T_1, \mathcal{O})$ with $a \sim^{\iota}_{\Psi'} \tilde{v}$, $\mu' \sim^{\iota}_{\Psi'} \tilde{\mu}'$, and $\Psi' \supseteq \Psi$. In addition, preservation (Lemma 24) gives $\vdash \tilde{\mu}' : \Sigma'$, $\Sigma' \supseteq \Sigma$, and $\emptyset, \Sigma', C \vdash \tilde{v} : (Y^{\ell}\ \mathrm{ref})^{\ell'}$. Thus, setting $(\tilde{v}', T') = \mathrm{select}^{\iota}(\tilde{\mu}', \tilde{v})$, the claim follows immediately from Lemma 43 and the dereference rule in the distributed semantics.

- Suppose $(\texttt{ref}\ e, \mu) \downarrow (a, \mu', \mathcal{O})$, so that $(e_1, \mu) \downarrow (v_1, \mu_1, \mathcal{O})$, $a \notin \mathrm{dom}\, \mu$, and $\mu' = \mu[a \mapsto v_1]$. Then we have $\tilde{e} = \texttt{ref}\ \tilde{e}_1$, with $e_1 \sim^{\iota}_{\Psi} \tilde{e}_1$, and $\emptyset, \Sigma, C \vdash \tilde{e}_1 : Y^{\ell'}$. Thus, the inductive hypothesis gives $(\tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T_1, \mathcal{O})$, with $(v_1, \mu_1) \sim^{\iota}_{\Psi_1} (\tilde{v}_1, \tilde{\mu}_1)$, and $\Psi_1 \supseteq \Psi$. Pick any $\tilde{a} \notin \mathrm{dom}\, \tilde{\mu}_1$, and let $\Psi' = \Psi_1[a \mapsto \tilde{a}]$, $\tilde{\mu}' = \tilde{\mu}[\tilde{a} \mapsto \tilde{v}]$. Then by weakening of the similarity relation (Lemma 37), $v_1 \sim^{\iota}_{\Psi'} \tilde{v}_1$; so, by definition, $a \sim^{\iota}_{\Psi'} \tilde{a}$ and $\mu' \sim^{\iota}_{\Psi'} \tilde{\mu}'$; and thus the claim follows immediately from the reference rule in the distributed semantics.

- Suppose $(\texttt{reveal}\ e, \mu) \downarrow (v', \mu', \mathcal{O})$, so that $(e_1, \mu) \downarrow (y, \mu', \mathcal{O}_1)$ and $\mathcal{O} = \mathcal{O}_1 \| y$. Then we have $\tilde{e} = \texttt{reveal}\ \tilde{e}_1$, with $e_1 \sim^{\iota}_{\Psi} \tilde{e}_1$ and $\emptyset, \Sigma, C \vdash \tilde{e}_1 : Y^{\mathrm{S}}$. Thus, the inductive hypothesis gives $(\tilde{e}_1, \tilde{\mu}, \iota) \Downarrow (\tilde{v}_1, \tilde{\mu}_1, T_1, \mathcal{O}_1)$, with $(y, \mu_1) \sim^{\iota}_{\Psi_1} (\tilde{v}_1, \tilde{\mu}_1)$ and $\Psi_1 \supseteq \Psi$. Then by preservation (Lemma 24), we have $\emptyset, \Sigma_1, C \vdash \tilde{v}_1 : Y^{\mathrm{S}}$. Hence $\tilde{v}_1 \in \hat{\mathcal{E}}^{\iota}_{\ell}(y)$ for some ℓ, and so by correctness of Dec, we have $\pi_1(\mathrm{Dec}(\tilde{v}_1)) = y$, and the claim follows from the reveal rule in the distributed semantics.

- Suppose $(\mathtt{op}^t_i(e_1, \ldots, e_r), \mu) \downarrow (v', \mu', \mathcal{O})$, so that for all $j \in \{1, \ldots, r\}$, $(e_j, \mu_{j-1}) \downarrow (y_j, \mu_j, \mathcal{O}_j)$, where $\mu = \mu_0$, $\mu' = \mu_r$, and $v' = \mathrm{op}^t_i(y_1, \ldots, y_r)$. Then we have $\tilde{e} = \mathrm{op}^t_i(\tilde{e}_1, \ldots, \tilde{e}_r)$, and for all j, $e_j \sim^{\iota}_{\Psi} \tilde{e}_j$ and $\emptyset, \Sigma, C \vdash \tilde{e}_j : Y_j^{\ell_j}$. Now, after r successive applications of the inductive hypothesis, preservation (Lemma 24), and weakening (Lemmas 1 and 37), we obtain $(\tilde{e}_j, \tilde{\mu}_{j-1}, \iota) \Downarrow (\tilde{y}_j, \tilde{\mu}_j, T_j, \mathcal{O}_j)$, with $(v_j, \mu_j) \sim^{\iota}_{\Psi_j} (\tilde{v}_j, \tilde{\mu}_j)$, $\emptyset, \Sigma_j, C \vdash \tilde{v}_j : Y_j^{\ell_j}$, and $\vdash \tilde{\mu}_j : \Sigma_j$. Thus, $\tilde{v}_j \in \hat{\mathcal{E}}^{\iota}_{\ell'_j}(v_j)$, so that by correctness of Enc, we have $\pi_1(\mathrm{Enc}_{\ell'_1, \ldots, \ell'_r}(\mathrm{op}^t_i)(\tilde{v}_1, \ldots, \tilde{v}_r, \iota)) \in \hat{\mathcal{E}}^{\iota}_{\ell'}(\mathrm{op}^t_i(v_1, \ldots, v_r)) = \hat{\mathcal{E}}^{\iota}_{\ell'}(v')$, where $\ell' = \sqcup_j \ell'_j$.

The claim then follows from the primitive operation rule in the distributed semantics and the definition of similarity.

□

Lemma 45 (Public Similarity is Reflexive for Surface Expressions)**.** *For any expression e (in the surface language),* $(e, \{\}) \approx^{\iota}_{\emptyset} (e, \{\})$.

Proof. By induction on the structure of the expression e. □

Lemma 46 (Permutation Weakening for Public Similarity Relation)**.**

- *If* $\tilde{e}_1 \approx^{\iota}_{\Psi} \tilde{e}_2$ *and* $\Psi' \supseteq \Psi$ *then* $\tilde{e}_1 \approx^{\iota}_{\Psi'} \tilde{e}_2$.
- *If* $\tilde{\mu}_1 \approx^{\iota}_{\Psi} \tilde{\mu}_2$ *and* $\Psi' \supseteq \Psi$ *then* $\tilde{\mu}_1 \approx^{\iota}_{\Psi'} \tilde{\mu}_2$.

Proof. The first claim follows by induction on the similarity relation $\tilde{e}_1 \approx^{\iota}_{\Psi} \tilde{e}_1$; the second claim then follows immediately by applying the first claim for each address $a \in \operatorname{dom} \tilde{\mu}_1$. □

Lemma 47 (Substitution for Public Similarity Relation)**.** *If* $\tilde{e}_1 \approx^{\iota}_{\Psi} \tilde{e}_2$ *and* $\tilde{v}_1 \approx^{\iota}_{\Psi} \tilde{v}_2$*, then* $\tilde{e}_1[\tilde{v}_1/x] \approx^{\iota}_{\Psi} \tilde{e}_2[\tilde{v}_2/x]$.

Proof. By induction on the similarity relation $\tilde{e}_1 \approx^{\iota}_{\Psi} \tilde{e}_2$. □

Lemma 48 (Public Similarity Relation Contains All Addresses)**.** *If* $\tilde{e}_1 \approx^{\iota}_{\Psi} \tilde{e}_2$*, then* $\operatorname{Addrs}(\tilde{e}_1) \subseteq \operatorname{dom} \Psi$ *and* $\operatorname{Addrs}(\tilde{e}_2) \subseteq \operatorname{cod} \Psi$.

Proof. By induction on the similarity relation $\tilde{e}_1 \approx^{\iota}_{\Psi} \tilde{e}_2$. □

Lemma 49 (Phi Preserves Public Equivalence of Values)**.** *If* $(\tilde{v}_1)_a \approx^{\iota}_{\Psi} (\tilde{v}_1)_b$*,* $(\tilde{v}_2)_a \approx^{\iota}_{\Psi} (\tilde{v}_2)_b$*,* $(\tilde{v}_3)_a \approx^{\iota}_{\Psi} (\tilde{v}_3)_b$*,* $\tilde{v}'_a = \Phi^{\iota}_1((\tilde{v}_1)_a, (\tilde{v}_2)_a), (\tilde{v}_3)_a)$*, and* $\tilde{v}'_b = \Phi^{\iota}_1((\tilde{v}_1)_b, (\tilde{v}_2)_b), (\tilde{v}_3)_b)$*, then* $\tilde{v}'_a \approx^{\iota}_{\Psi} \tilde{v}'_b$.

Proof. By case analysis on the statement $\tilde{v}'_a = \Phi^{\iota}_1((\tilde{v}_1)_a, (\tilde{v}_2)_a), (\tilde{v}_3)_a)$. □

Lemma 50 (Phi Preserves Public Equivalence of Stores)**.** *If* $(\tilde{v}_1)_a \approx^{\iota}_{\Psi} (\tilde{v}_1)_b$*,* $(\tilde{\mu}_2)_a \approx^{\iota}_{\Psi} (\tilde{\mu}_2)_b$*,* $(\tilde{\mu}_3)_a \approx^{\iota}_{\Psi} (\tilde{\mu}_3)_b$*,* $\tilde{\mu}'_a = \Phi^{\iota}_1((\tilde{v}_1)_a, (\tilde{\mu}_2)_a), (\tilde{\mu}_3)_a)$*, and* $\tilde{\mu}'_b = \Phi^{\iota}_1((\tilde{v}_1)_b, (\tilde{\mu}_2)_b), (\tilde{\mu}_3)_b)$*, then* $\tilde{\mu}'_a \approx^{\iota}_{\Psi} \tilde{\mu}'_b$.

Proof. Follows by applying Lemma 49 for each address in $\operatorname{dom} \tilde{\mu}_2 \cap \operatorname{dom} \tilde{\mu}_3$ (and follows by definition for other addresses). □

Lemma 51 (Store Update Preserves Public Equivalence)**.** *If* $\tilde{\mu}_a \approx^{\iota}_{\Psi} \tilde{\mu}_b$*,* $\tilde{v}_a \approx^{\iota}_{\Psi} \tilde{v}_b$*,* $\tilde{v}'_a \approx^{\iota}_{\Psi} \tilde{v}'_b$*,* $\tilde{\mu}'_a = \operatorname{update}^{\iota}_1(\tilde{\mu}_a, \tilde{v}_a, \tilde{v}'_a)$*, and* $\tilde{\mu}'_b = \operatorname{update}^{\iota}_1(\tilde{\mu}_b, \tilde{v}_b, \tilde{v}'_b)$*, then* $\tilde{\mu}'_a \approx^{\iota}_{\Psi} \tilde{\mu}'_b$.

Proof. By induction on the statement $\tilde{\mu}'_a = \operatorname{update}^{\iota}_1(\tilde{\mu}_a, \tilde{v}_a, \tilde{v}'_a)$, applying Lemma 50 in the inductive case. □

Lemma 52 (Store Selection Preserves Public Equivalence)**.** *If* $\tilde{\mu}_a \approx^{\iota}_{\Psi} \tilde{\mu}_b$*, and* $\tilde{v}_a \approx^{\iota}_{\Psi} \tilde{v}_b$*,* $\tilde{v}'_a = \operatorname{select}^{\iota}_2(\tilde{\mu}_a, \tilde{v}_a)$*, and* $\tilde{v}'_b = \operatorname{select}^{\iota}_2(\tilde{\mu}_b, \tilde{v}_b)$*, then* $\tilde{v}'_a \approx^{\iota}_{\Psi} \tilde{v}'_b$.

Proof. By induction on the statement $\tilde{v}'_a = \operatorname{select}^{\iota}_2(\tilde{\mu}_a, \tilde{v}_a)$, applying Lemma 49 in the inductive case. □

Lemma 53 (Safety of Phi of Values)**.** *If* $(\tilde{v}_1)_a \approx^{\iota}_{\Psi} (\tilde{v}_1)_b$*;* $(\tilde{v}_2)_a \approx^{\iota}_{\Psi} (\tilde{v}_2)_b$*;* $(\tilde{v}_3)_a \approx^{\iota}_{\Psi} (\tilde{v}_3)_b$*;* $T_a = \Phi^{\iota}_2((\tilde{v}_1)_a, (\tilde{v}_2)_a), (\tilde{v}_3)_a)$*; and* $T_b = \Phi^{\iota}_2((\tilde{v}_1)_b, (\tilde{v}_2)_b), (\tilde{v}_3)_b)$*, then* $\textsc{Safe}(\iota, T_a, T_b)$.

Proof. By case analysis on the statement $T_a = \Phi^{\iota}_2((\tilde{v}_1)_a, (\tilde{v}_2)_a), (\tilde{v}_3)_a)$. If $(\tilde{v}_2)_a$, $(\tilde{v}_3)_a$ are not primitive values, then the traces are empty and the result is immediate. On the other hand, if $(\tilde{v}_2)_a$, $(\tilde{v}_3)_a$ are primitive values, then the conditions of public similarity immediately give $\textsc{SafeOp}(\iota, T_a, T_b)$, as desired. □

Lemma 54 (Safety of Phi of Stores)**.** *If* $(\tilde{v}_1)_a \approx^{\iota}_{\Psi} (\tilde{v}_1)_b$*;* $(\tilde{\mu}_2)_a \approx^{\iota}_{\Psi} (\tilde{\mu}_2)_b$*;* $(\tilde{\mu}_3)_a \approx^{\iota}_{\Psi} (\tilde{\mu}_3)_b$*;* $T_a = \Phi^{\iota}_2((\tilde{v}_1)_a, (\tilde{\mu}_2)_a), (\tilde{\mu}_3)_a)$*; and* $T_b = \Phi^{\iota}_2((\tilde{v}_1)_b, (\tilde{\mu}_2)_b), (\tilde{\mu}_3)_b)$*, then* $\textsc{Safe}(\iota, T_a, T_b)$.

Proof. We note that the definition of Φ on stores proceeds in lexicographical order by type signature. Since $(\tilde{\mu}_2)_a \approx^{\iota}_{\Psi} (\tilde{\mu}_2)_b$ and $(\tilde{\mu}_3)_a \approx^{\iota}_{\Psi} (\tilde{\mu}_3)_b$, for each address $\tilde{a} \in \operatorname{dom} \tilde{\mu}_2 \cap \operatorname{dom} \tilde{\mu}_3$, we have $(\operatorname{sig}_{(\tilde{\mu}_2)_a}(\tilde{a}), \operatorname{sig}_{(\tilde{\mu}_3)_a}(\tilde{a})) = (\operatorname{sig}_{(\tilde{\mu}_2)_b}(\Psi(\tilde{a})), \operatorname{sig}_{(\tilde{\mu}_3)_b}(\Psi(\tilde{a})))$. Thus, the components of the traces for each individual address are safe, by Lemma 53, and hence their concatenation is also safe. □

Lemma 55 (Safety of Store Update)**.** *If* $\tilde{\mu}_a \approx^{\iota}_{\Psi} \tilde{\mu}_b$*;* $\tilde{v}_a \approx^{\iota}_{\Psi} \tilde{v}_b$*;* $\tilde{v}'_a \approx^{\iota}_{\Psi} \tilde{v}'_b$*;* $T_a = \operatorname{update}^{\iota}_2(\tilde{\mu}_a, \tilde{v}_a, \tilde{v}'_a)$*; and* $T_b = \operatorname{update}^{\iota}_2(\tilde{\mu}_b, \tilde{v}_b, \tilde{v}'_b)$*, then* $\textsc{Safe}(\iota, T_a, T_b)$.

Proof. By induction on the statement $T_a = \operatorname{update}^{\iota}_2(\tilde{\mu}_a, \tilde{v}_a, \tilde{v}'_a)$, applying Lemma 54 in the inductive case. □

Lemma 56 (Safety of Store Selection)**.** *If* $\tilde{\mu}_a \approx^{\iota}_{\Psi} \tilde{\mu}_b$*;* $\tilde{v}_a \approx^{\iota}_{\Psi} \tilde{v}_b$*;* $T_a = \operatorname{select}^{\iota}_2(\tilde{\mu}_a, \tilde{v}_a)$*; and* $T_b = \operatorname{select}^{\iota}_2(\tilde{\mu}_b, \tilde{v}_b)$*, then* $\textsc{Safe}(\iota, T_a, T_b)$.

Proof. By induction on the statement $T_a = \operatorname{select}^{\iota}_2(\tilde{\mu}_a, \tilde{v}_a)$, applying Lemma 53 in the inductive case. □

Lemma 57 (Safety of Traces)**.** *If* $(\tilde{e}_a, \tilde{\mu}_a, \iota) \Downarrow (\tilde{v}'_a, \tilde{\mu}'_a, T_a, \mathcal{O}_a)$*,* $(\tilde{e}_b, \tilde{\mu}_b, \iota) \Downarrow (\tilde{v}'_b, \tilde{\mu}'_b, T_b, \mathcal{O}_b)$*,* $(\tilde{e}_a, \tilde{\mu}_a) \approx^{\iota}_{\Psi} (\tilde{e}_b, \tilde{\mu}_b)$*, and* $\Psi : \operatorname{dom} \tilde{\mu}_a \leftrightarrow \operatorname{dom} \tilde{\mu}_b$*, then neither of* $\mathcal{O}_a$ *and* $\mathcal{O}_b$ *is a proper prefix of the other; and, if* $\mathcal{O}_a = \mathcal{O}_b$*, then for some* $\Psi' \supseteq \Psi$*,* $(\tilde{v}'_a, \tilde{\mu}'_a) \approx^{\iota}_{\Psi'} (\tilde{v}'_b, \tilde{\mu}'_b)$*,* $\Psi' : \operatorname{dom} \tilde{\mu}'_a \leftrightarrow \operatorname{dom} \tilde{\mu}'_b$*, and* $\textsc{Safe}(\iota, T_a, T_b)$.

Proof. By induction on the first evaluation derivation $(\tilde{e}_a, \tilde{\mu}_a, \iota) \Downarrow (\tilde{v}'_a, \tilde{\mu}'_a, T_a, \mathcal{O}_a)$. If $\tilde{e}_a$ is a value, then by inspection $\tilde{e}_b$ is a value, so the claim follows trivially; thus, we only treat the cases in which $\tilde{e}_a$ is not a value.

- Suppose $\tilde{e}_a = \texttt{if}\ (\tilde{e}_1)_a\ \texttt{then}\ (\tilde{e}_2)_a\ \texttt{else}\ (\tilde{e}_3)_a$, so that $\tilde{e}_b = \texttt{if}\ (\tilde{e}_1)_b\ \texttt{then}\ (\tilde{e}_2)_b\ \texttt{else}\ (\tilde{e}_3)_b$ and:
  - $((\tilde{e}_1)_a, \tilde{\mu}_a, \iota) \Downarrow ((\tilde{v}_1)_a, (\tilde{\mu}_1)_a, (T_1)_a, (\mathcal{O}_1)_a)$
  - $((\tilde{e}_2)_a, (\tilde{\mu}_1)_a, \iota) \Downarrow ((\tilde{v}_2)_a, (\tilde{\mu}_2)_a, (T_2)_a, (\mathcal{O}_2)_a)$
  - $((\tilde{e}_3)_a, (\tilde{\mu}_1)_a, \iota) \Downarrow ((\tilde{v}_3)_a, (\tilde{\mu}_3)_a, (T_3)_a, (\mathcal{O}_3)_a)$
  - $\operatorname{dom}(\tilde{\mu}_2)_a \cap \operatorname{dom}(\tilde{\mu}_3)_a = \operatorname{dom}(\tilde{\mu}_1)_a$
  - $\tilde{v}'_a = \Phi^{\iota}_1((\tilde{v}_1)_a, (\tilde{v}_2)_a, (\tilde{v}_3)_a)$
  - $\tilde{\mu}'_a = \Phi^{\iota}_1((\tilde{\mu}_1)_a, (\tilde{\mu}_2)_a, (\tilde{\mu}_3)_a)$
  - $\mathcal{O}_a = (\mathcal{O}_1)_a \| (\mathcal{O}_2)_a \| (\mathcal{O}_3)_a$
  - $T_a = (T_1)_a \| (T_2)_a \| (T_3)_a \| \Phi^{\iota}_2((\tilde{v}_1)_a, (\tilde{v}_2)_a, (\tilde{v}_3)_a) \| \Phi^{\iota}_2((\tilde{\mu}_1)_a, (\tilde{\mu}_2)_a, (\tilde{\mu}_3)_a)$

  By inversion on the second evaluation derivation, we obtain the analogous facts:

  - $((\tilde{e}_1)_b, \tilde{\mu}_b, \iota) \Downarrow ((\tilde{v}_1)_b, (\tilde{\mu}_1)_b, (T_1)_b, (\mathcal{O}_1)_b)$
  - $((\tilde{e}_2)_b, (\tilde{\mu}_1)_b, \iota) \Downarrow ((\tilde{v}_2)_b, (\tilde{\mu}_2)_b, (T_2)_b, (\mathcal{O}_2)_b)$
  - $((\tilde{e}_3)_b, (\tilde{\mu}_1)_b, \iota) \Downarrow ((\tilde{v}_3)_b, (\tilde{\mu}_3)_b, (T_3)_b, (\mathcal{O}_3)_b)$
  - $\operatorname{dom}(\tilde{\mu}_2)_b \cap \operatorname{dom}(\tilde{\mu}_3)_b = \operatorname{dom}(\tilde{\mu}_1)_b$
  - $\tilde{v}'_b = \Phi^{\iota}_1((\tilde{v}_1)_b, (\tilde{v}_2)_b, (\tilde{v}_3)_b)$
  - $\tilde{\mu}'_b = \Phi^{\iota}_1((\tilde{\mu}_1)_b, (\tilde{\mu}_2)_b, (\tilde{\mu}_3)_b)$
  - $\mathcal{O}_b = (\mathcal{O}_1)_b \| (\mathcal{O}_2)_b \| (\mathcal{O}_3)_b$
  - $T_b = (T_1)_b \| (T_2)_b \| (T_3)_b \| \Phi^{\iota}_2((\tilde{v}_1)_b, (\tilde{v}_2)_b, (\tilde{v}_3)_b) \| \Phi^{\iota}_2((\tilde{\mu}_1)_b, (\tilde{\mu}_2)_b, (\tilde{\mu}_3)_b)$

Now, by the inductive hypothesis, neither of $(\mathcal{O}_1)_a$ and $(\mathcal{O}_1)_b$ is a proper prefix of the other; thus, either $(\mathcal{O}_1)_a = (\mathcal{O}_1)_b$, or else $\mathcal{O}_a \neq \mathcal{O}_b$ and they differ on some position within the length of $\mathcal{O}_1$ (i.e., neither of $\mathcal{O}_a$ and $\mathcal{O}_b$ is a proper prefix of the other, satisfying the only required condition). Thus, we may assume without loss of generality that $(\mathcal{O}_1)_a = (\mathcal{O}_1)_b$, so that $((\tilde{v}_1)_a, (\tilde{\mu}_1)_a) \approx^{\iota}_{\Psi_1} ((\tilde{v}_1)_b, (\tilde{\mu}_1)_b)$ for some $\Psi_1 \supseteq \Psi$, $\textsc{Safe}(\iota, (T_1)_a, (T_1)_b)$, and $\Psi_1 : \mathrm{dom}(\tilde{\mu}_1)_a \leftrightarrow \mathrm{dom}(\tilde{\mu}_1)_b$. Proceeding in the same fashion with $(\mathcal{O}_2)_a$ and $(\mathcal{O}_3)_a$, we obtain the following:

- $(\mathcal{O}_2)_a = (\mathcal{O}_2)_b$
- $((\tilde{v}_2)_a, (\tilde{\mu}_2)_a) \approx^{\iota}_{\Psi_2} ((\tilde{v}_2)_b, (\tilde{\mu}_2)_b)$
- $\Psi_2 \supseteq \Psi_1 \supseteq \Psi$
- $\textsc{Safe}(\iota, (T_2)_a, (T_2)_b)$
- $\Psi_2 : \mathrm{dom}(\tilde{\mu}_2)_a \leftrightarrow \mathrm{dom}(\tilde{\mu}_2)_b$
- $(\mathcal{O}_3)_a = (\mathcal{O}_3)_b$
- $((\tilde{v}_3)_a, (\tilde{\mu}_3)_a) \approx^{\iota}_{\Psi_3} ((\tilde{v}_2)_b, (\tilde{\mu}_3)_b)$
- $\Psi_3 \supseteq \Psi_1 \supseteq \Psi$
- $\textsc{Safe}(\iota, (T_3)_a, (T_3)_b)$
- $\Psi_3 : \mathrm{dom}(\tilde{\mu}_3)_a \leftrightarrow \mathrm{dom}(\tilde{\mu}_3)_b$

In addition, setting $\Psi' = \Psi_2 \cup \Psi_3 \supseteq \Psi_1 \supseteq \Psi$ (well-defined since $\mathrm{dom}(\tilde{\mu}_2)_a \cap \mathrm{dom}(\tilde{\mu}_3)_a = \mathrm{dom}(\tilde{\mu}_1)_a$ and $\mathrm{dom}(\tilde{\mu}_2)_b \cap \mathrm{dom}(\tilde{\mu}_3)_b = \mathrm{dom}(\tilde{\mu}_1)_b$), we note that by Lemma 12, $\Psi' : \mathrm{dom}\,\tilde{\mu}'_a \leftrightarrow \mathrm{dom}\,\tilde{\mu}'_b$; and, by Lemma 46, the above public-equivalence judgments carry over to Ψ'. Thus, Lemmas 49 and 50 give $(\tilde{v}'_a, \tilde{\mu}'_a) \approx^{\iota}_{\Psi'} (\tilde{v}'_b, \tilde{\mu}'_b)$; Lemma 53 gives $\textsc{Safe}(\iota, \Phi^{\iota}_2((\tilde{v}_1)_a, (\tilde{v}_2)_a, (\tilde{v}_3)_a), \Phi^{\iota}_2((\tilde{v}_1)_b, (\tilde{v}_2)_b, (\tilde{v}_3)_b))$; and Lemma 54 gives $\textsc{Safe}(\iota, \Phi^{\iota}_2((\tilde{\mu}_1)_a, (\tilde{\mu}_2)_a, (\tilde{\mu}_3)_a), \Phi^{\iota}_2((\tilde{\mu}_1)_b, (\tilde{\mu}_2)_b, (\tilde{\mu}_3)_b))$, proving the claim.

- Suppose $\tilde{e}_a = (\tilde{e}_0)_a\ (\tilde{e}_1)_a$, so that $\tilde{e}_b = (\tilde{e}_0)_b\ (\tilde{e}_1)_b$. Then we have the following subcases.

  - $\tilde{e}_a$ evaluates to a lambda abstraction, i.e.:
    - $((\tilde{e}_0)_a, \tilde{\mu}_a, \iota) \Downarrow (\lambda x.(\tilde{e}'_0)_a, (\tilde{\mu}_0)_a, (T_0)_a, (\mathcal{O}_0)_a)$
    - $((\tilde{e}_1)_a, (\tilde{\mu}_0)_a, \iota) \Downarrow ((\tilde{v}_1)_a, (\tilde{\mu}_1)_a, (T_1)_a, (\mathcal{O}_1)_a)$
    - $((\tilde{e}'_0)_a[(\tilde{v}_1)_a/x], (\tilde{\mu}_1)_a, \iota) \Downarrow ((\tilde{v}')_a, \tilde{\mu}'_a, (T_2)_a, (\mathcal{O}_2)_a)$
    - $\mathcal{O}_a = (\mathcal{O}_0)_a \| (\mathcal{O}_1)_a \| (\mathcal{O}_2)_a$
    - $T_a = (T_0)_a \| (T_1)_a \| (T_2)_a$

    By inversion on the second evaluation derivation, we obtain the analogous facts with b in place of a. Further, by the same argument as in the conditional case, we have:

    - $(\mathcal{O}_0)_a = (\mathcal{O}_0)_b$, $(\mathcal{O}_1)_a = (\mathcal{O}_1)_b$
    - $\textsc{Safe}(\iota, (T_0)_a, (T_0)_b)$, $\textsc{Safe}(\iota, (T_1)_a, (T_1)_b)$
    - $((\lambda x.(\tilde{e}'_0)_a), (\tilde{\mu}_0)_a) \approx^{\iota}_{\Psi_0} ((\lambda x.(\tilde{e}'_0)_b), (\tilde{\mu}_0)_b)$
    - $((\tilde{v}_1)_a, (\tilde{\mu}_1)_a) \approx^{\iota}_{\Psi_1} ((\tilde{v}_1)_b, (\tilde{\mu}_1)_b)$
    - $\Psi_0 : \mathrm{dom}(\tilde{\mu}_0)_a \leftrightarrow \mathrm{dom}(\tilde{\mu}_0)_b$, $\Psi_1 : \mathrm{dom}(\tilde{\mu}_1)_a \leftrightarrow \mathrm{dom}(\tilde{\mu}_1)_b$
    - $\Psi_1 \supseteq \Psi_0 \supseteq \Psi$

    Now, by definition, $(\tilde{e}'_0)_a \approx^{\iota}_{\Psi_0} (\tilde{e}'_0)_b$, and thus Lemma 47 gives $(\tilde{e}'_0)_a[(\tilde{v}_1)_a/x] \approx^{\iota}_{\Psi_0} (\tilde{e}'_0)_b[(\tilde{v}_1)_b/x]$ (and, by Lemma 46, $(\tilde{e}'_0)_a[(\tilde{v}_1)_a/x] \approx^{\iota}_{\Psi_1} (\tilde{e}'_0)_b[(\tilde{v}_1)_b/x]$). So, again by the same argument as in the conditional case, we have:

    - $(\mathcal{O}_2)_a = (\mathcal{O}_2)_b$
    - $(\tilde{v}'_a, \tilde{\mu}'_a) \approx^{\iota}_{\Psi'} (\tilde{v}'_b, \tilde{\mu}'_b)$

* $\textsc{Safe}(\iota, (T_2)_a, (T_2)_b)$
* $\Psi' : \mathrm{dom}\,\tilde{\mu}'_a \leftrightarrow \mathrm{dom}\,\tilde{\mu}'_b$,

and the claim follows immediately.

– $\tilde{e}_a$ evaluates to a fixpoint definition. The proof is exactly the same as in the previous case, except that we perform the substitution $(\tilde{e}'_0)_a[(\texttt{fix}\ f.\lambda x.(\tilde{e}'_0)_a)/f, (\tilde{v}_1)_a/x]$ rather than $(\tilde{e}'_0)_a[(\tilde{v}_1)_a/x]$ (again using Lemma 46 to conclude that $(\texttt{fix}\ f.\lambda x.(\tilde{e}'_0)_a) \approx^{\iota}_{\Psi_1} (\texttt{fix}\ f.\lambda x.(\tilde{e}'_0)_b)$ follows from $(\texttt{fix}\ f.\lambda x.(\tilde{e}'_0)_a) \approx^{\iota}_{\Psi_0} (\texttt{fix}\ f.\lambda x.(\tilde{e}'_0)_b)$).

– $\tilde{e}_a$ evaluates to a phi symbol, i.e.:

* $\tilde{b}_a \in \mathcal{E}_{\ell_1}(\mathrm{bool})$
* $((\tilde{e}_0)_a, \tilde{\mu}_a, \iota) \Downarrow (\varphi(\tilde{b}_a, (\tilde{v}_2)_a, (\tilde{v}_3)_a), (\tilde{\mu}_0)_a, (T_0)_a, (\mathcal{O}_0)_a)$
* $((\tilde{e}_1)_a, (\tilde{\mu}_0)_a, \iota) \Downarrow ((\tilde{v}_1)_a, (\tilde{\mu}_1)_a, (T_1)_a, (\mathcal{O}_1)_a)$
* $((\tilde{v}_2)_a\ (\tilde{v}_1)_a, (\tilde{\mu}_1)_a, \iota) \Downarrow ((\tilde{v}'_2)_a, (\tilde{\mu}_2)_a, (T_2)_a, (\mathcal{O}_2)_a)$
* $((\tilde{v}_3)_a\ (\tilde{v}_1)_a, (\tilde{\mu}_1)_a, \iota) \Downarrow ((\tilde{v}'_3)_a, (\tilde{\mu}_3)_a, (T_3)_a, (\mathcal{O}_3)_a)$
* $\mathrm{dom}(\tilde{\mu}_2)_a \cap \mathrm{dom}(\tilde{\mu}_3)_a = \mathrm{dom}(\tilde{\mu}_1)_a$
* $(\tilde{v}'_a, (T_4)_a) = \Phi^{\iota}(\tilde{b}_a, (\tilde{v}'_2)_a, (\tilde{v}'_3)_a)$
* $(\tilde{\mu}'_a, (T_5)_a) = \Phi^{\iota}(\tilde{b}_a, (\tilde{\mu}_2)_a, (\tilde{\mu}_3)_a)$
* $\mathcal{O}_a = (\mathcal{O}_0)_a \| \ldots \| (\mathcal{O}_3)_a$
* $T_a = (T_0)_a \| \ldots \| (T_5)_a$

The proof now proceeds as in the conditional case.

- Suppose $\tilde{e}_a = ((\tilde{e}_1)_a := (\tilde{e}_2)_a)$, so that $\tilde{e}_b = ((\tilde{e}_1)_b := (\tilde{e}_2)_b)$, and:

– $((\tilde{e}_1)_a, \tilde{\mu}_a, \iota) \Downarrow ((\tilde{v}_1)_a, (\tilde{\mu}_1)_a, (T_1)_a, (\mathcal{O}_1)_a)$
– $((\tilde{e}_2)_a, (\tilde{\mu}_1)_a, \iota) \Downarrow ((\tilde{v}_2)_a, (\tilde{\mu}_2)_a, (T_2)_a, (\mathcal{O}_2)_a)$
– $\mathcal{O}_a = (\mathcal{O}_1)_a \| (\mathcal{O}_2)_a$
– $T_a = (T_1)_a \| (T_2)_a \| \mathrm{update}^{\iota}_2((\tilde{\mu}_2)_a, (\tilde{v}_1)_a, (\tilde{v}_2)_a)$
– $\tilde{\mu}'_a = \mathrm{update}^{\iota}_1((\tilde{\mu}_2)_a, (\tilde{v}_1)_a, (\tilde{v}_2)_a)$
– $\tilde{v}'_a = ()$

By inversion on the second evaluation derivation, we obtain the analogous facts with b in place of a. Further, by the same argument as in the conditional case, we have:

– $(\mathcal{O}_1)_a = (\mathcal{O}_1)_b$, $(\mathcal{O}_2)_a = (\mathcal{O}_2)_b$
– $((\tilde{v}_1)_a, (\tilde{\mu}_1)_a) \approx^{\iota}_{\Psi_1} ((\tilde{v}_1)_b, (\tilde{\mu}_1)_b)$
– $((\tilde{v}_2)_a, (\tilde{\mu}_2)_a) \approx^{\iota}_{\Psi_2} ((\tilde{v}_2)_b, (\tilde{\mu}_2)_b)$
– $\textsc{Safe}(\iota, (T_1)_a, (T_1)_b)$, $\textsc{Safe}(\iota, (T_2)_a, (T_2)_b)$
– $\Psi_1 : \mathrm{dom}(\tilde{\mu}_1)_a \leftrightarrow \mathrm{dom}(\tilde{\mu}_1)_b$, $\Psi_2 : \mathrm{dom}(\tilde{\mu}_2)_a \leftrightarrow \mathrm{dom}(\tilde{\mu}_2)_b$
– $\Psi_2 \supseteq \Psi_1 \supseteq \Psi$

By Lemma 46, $((\tilde{v}_1)_a, (\tilde{\mu}_1)_a) \approx^{\iota}_{\Psi_2} ((\tilde{v}_1)_b, (\tilde{\mu}_1)_b)$. Thus, setting $\Psi' = \Psi_2$, Lemma 51 gives $((), \tilde{\mu}'_a) \approx^{\iota}_{\Psi'} ((), \tilde{\mu}'_b)$; Lemma 55 gives $\textsc{Safe}(\iota, \mathrm{update}^{\iota}_2((\tilde{\mu}_2)_a, (\tilde{v}_1)_a, (\tilde{v}_2)_a), \mathrm{update}^{\iota}_2((\tilde{\mu}_2)_b, (\tilde{v}_1)_b, (\tilde{v}_2)_b))$; and Lemmas 13 and 48 gives $\Psi' : \mathrm{dom}\,\tilde{\mu}'_a \leftrightarrow \mathrm{dom}\,\tilde{\mu}'_b$, proving the claim.

- Suppose $\tilde{e}_a = !(\tilde{e}_1)_a$, so that $\tilde{e}_b = !(\tilde{e}_1)_b$, and:

– $((\tilde{e}_1)_a, \tilde{\mu}_a, \iota) \Downarrow ((\tilde{v}_1)_a, (\tilde{\mu}_1)_a, (T_1)_a, (\mathcal{O}_1)_a)$

- $\mathcal{O}_a = (\mathcal{O}_1)_a$
- $T_a = (T_1)_a \| \mathrm{select}_2^\iota((\tilde{\mu}_1)_a, (\tilde{v}_1)_a)$
- $\tilde{v}'_a = \mathrm{select}_1^\iota((\tilde{\mu}_1)_a, (\tilde{v}_1)_a)$
- $\tilde{\mu}'_a = (\tilde{\mu}_1)_a$

By inversion on the second evaluation derivation, we obtain the analogous facts with b in place of a. Further, by the same argument as in the conditional case, we have:

- $(\mathcal{O}_1)_a = (\mathcal{O}_1)_b$
- $\textsc{Safe}(\iota, (T_1)_a, (T_1)_b)$
- $((\tilde{v}_1)_a, (\tilde{\mu}_1)_a) \approx^\iota_{\Psi_1} ((\tilde{v}_1)_b, (\tilde{\mu}_1)_b)$
- $\Psi_0 : \mathrm{dom}(\tilde{\mu}_0)_a \leftrightarrow \mathrm{dom}(\tilde{\mu}_0)_b$, $\Psi_1 : \mathrm{dom}(\tilde{\mu}_1)_a \leftrightarrow \mathrm{dom}(\tilde{\mu}_1)_b$
- $\Psi_1 \supseteq \Psi_0 \supseteq \Psi$

Thus, setting $\Psi' = \Psi_1$, Lemma 52 gives $(\tilde{v}'_a, \tilde{\mu}'_a) \approx^\iota_{\Psi'} (\tilde{v}'_b, \tilde{\mu}'_b)$, and Lemma 56 gives $\textsc{Safe}(\iota, \mathrm{select}_2^\iota((\tilde{\mu}_2)_a, (\tilde{v}_1)_a, (\tilde{v}_2)_a)$ proving the claim.

- Suppose $\tilde{e}_a = \texttt{ref}\ (\tilde{e}_1)_a$, so that $\tilde{e}_b = \texttt{ref}\ (\tilde{e}_1)_b$, and:
  - $(\tilde{e}_a, \tilde{\mu}_a, \iota) \Downarrow (\tilde{v}_a, (\tilde{\mu}_1)_a, (T_1)_a, (\mathcal{O}_1)$
  - $l_a \notin \mathrm{dom}\, \tilde{\mu}_a$
  - $\tilde{\mu}'_a = (\tilde{\mu}_1)_a[l_a \mapsto \tilde{v}_a]$
  - $T'_a = (T_1)_a$
  - $\mathcal{O}_a = (\mathcal{O}_1)_a$

  By inversion on the second evaluation derivation, we obtain the analogous facts with b in place of a. Further, by the same argument as in the conditional case, we have:
  - $(\mathcal{O}_1)_a = (\mathcal{O}_1)_b$
  - $\textsc{Safe}(\iota, (T_1)_a, (T_1)_b)$
  - $((\tilde{v}_1)_a, (\tilde{\mu}_1)_a) \approx^\iota_{\Psi_1} ((\tilde{v}_1)_b, (\tilde{\mu}_1)_b)$
  - $\Psi_1 : \mathrm{dom}(\tilde{\mu}_1)_a \leftrightarrow \mathrm{dom}(\tilde{\mu}_1)_b$
  - $\Psi_1 \supseteq \Psi$

  Setting $\Psi' = \Psi_1[l_a \mapsto l_b]$, the claim is then immediate.

- Suppose $\tilde{e}_a = \texttt{reveal}\ (\tilde{e}_1)_a$, so that $\tilde{e}_b = \texttt{reveal}\ (\tilde{e}_1)_b$, and:
  - $((\tilde{e}_1)_a, \tilde{\mu}_a, \iota) \Downarrow ((\tilde{v}_1)_a, (\tilde{\mu}_1)_a, (T_1)_a, (\mathcal{O}_1)_a)$
  - $\mathcal{O}_a = (\mathcal{O}_1)_a \| \pi_1(\mathsf{Dec}((\tilde{v}_1)_a, \iota))$
  - $T_a = (T_1)_a \| \pi_2(\mathsf{Dec}((\tilde{v}_1)_a, \iota))$
  - $\tilde{\mu}'_a = (\tilde{\mu}_1)_a$
  - $\tilde{v}'_a = \pi_1(\mathsf{Dec}((\tilde{v}_1)_a, \iota))$

  By inversion on the second evaluation derivation, we obtain the analogous facts with b in place of a. Further, by the same argument as in the conditional case, we have:
  - $(\mathcal{O}_1)_a = (\mathcal{O}_1)_b$

- $((\tilde{v}_1)_a, (\tilde{\mu}_1)_a) \approx^{\iota}_{\Psi_1} ((\tilde{v}_1)_b, (\tilde{\mu}_1)_b)$
- $\textsc{Safe}(\iota, (T_1)_a, (T_1)_b)$
- $\Psi_1 : \mathrm{dom}(\tilde{\mu}_1)_a \leftrightarrow \mathrm{dom}(\tilde{\mu}_1)_b$
- $\Psi_1 \supseteq \Psi$

Again setting $\Psi' = \Psi_1$, we note that Now, since $(\mathcal{O}_1)_a = (\mathcal{O}_1)_b$, we have $\pi_1(\mathsf{Dec}((\tilde{v}_1)_a, \iota)) = \pi_1(\mathsf{Dec}((\tilde{v}_1)_b, \iota))$ (and thus also $\pi_1(\mathsf{Dec}((\tilde{v}_1)_a, \iota)) \approx^{\iota}_{\Psi'} \pi_1(\mathsf{Dec}((\tilde{v}_1)_b, \iota))$). The assumptions of the secure execution platform then give $\textsc{Safe}(\iota, \pi_2(\mathsf{Dec}((\tilde{v}_1)_a, \iota)), \pi_2(\mathsf{Dec}((\tilde{v}_1)_a, \iota)))$, as desired.

- Suppose $\tilde{e}_a = \mathsf{op}^t_i((e_1)_a, \ldots, (e_r)_a)$, so that $\tilde{e}_b = \mathsf{op}^t_i((e_1)_b, \ldots, (e_r)_b)$, and:
  - $\forall 1 \leq j \leq r.((e_j)_a, (\tilde{\mu}_{j-1})_a, \iota) \Downarrow ((\tilde{y}_j)_a, (\tilde{\mu}_j)_a, (T_j)_a, (\mathcal{O}_j)_a)$
  - $(\tilde{y}_j)_a \in \mathcal{E}_{(\ell_j)_a}((Y_j)_a)$
  - $\mathrm{Enc}_{(\ell_1)_a, \ldots, (\ell_j)_a}(\mathrm{op}^t_i)((\tilde{y}_1)_a, \ldots, (\tilde{y}_r)_a, \iota) = (\tilde{v}'_a, T'_a)$
  - $T_a = (T_1)_a \| \ldots \| (T_r)_a \| T'_a$
  - $\mathcal{O}_a = (\mathcal{O}_1)_a \| \ldots \| (\mathcal{O}_r)_a$

  By inversion on the second evaluation derivation, we obtain the analogous facts with b in place of a. In addition, we obtain $(Y_j)_a = (Y_j)_b = Y_j$, $(\ell_j)_a = (\ell_j)_b = \ell_j$, and $(\tilde{y}_j)_a, (\tilde{y}_j)_b \in \hat{\mathcal{E}}^{\iota}_{\ell_j}(Y_j)$. Further, by the same argument as in the conditional case, we have:
  - $(\mathcal{O}_j)_a = (\mathcal{O}_j)_b$
  - $((\tilde{v}_j)_a, (\tilde{\mu}_j)_a) \approx^{\iota}_{\Psi_j} ((\tilde{v}_j)_b, (\tilde{\mu}_j)_b)$
  - $\textsc{Safe}(\iota, (T_j)_a, (T_j)_b)$
  - $\Psi_j : \mathrm{dom}(\tilde{\mu}_j)_a \leftrightarrow \mathrm{dom}(\tilde{\mu}_j)_b$
  - $\Psi_r \supseteq \ldots \supseteq \Psi_1 \supseteq \Psi$

  Finally, setting $\Psi' = \Psi_r$, the cryptographic correctness assumptions of the secure execution platform give $\tilde{v}'_a \approx^{\iota}_{\Psi'} \tilde{v}'_b$; and, since each $(\tilde{v}_j)_a \approx^{\iota}_{\Psi_j} (\tilde{v}_j)_b$, the indistinguishability conditions give:

$$\textsc{Safe}(\iota, \mathrm{Enc}_{\ell_1, \ldots, \ell_j}(\mathrm{op}^t_i)((\tilde{y}_1)_a, \ldots, (\tilde{y}_r)_a, \iota), \mathrm{Enc}_{\ell_1, \ldots, \ell_j}(\mathrm{op}^t_i)((\tilde{y}_1)_b, \ldots, (\tilde{y}_r)_b, \iota)$$

  as desired.

□

B Examples of secure execution platforms

B.1 Shamir secret sharing

We now define Shamir secret sharing in the notation of our framework (Section 4), and show that it is a secure execution platform (Definition 1), thereby concluding all of the correctness and security results of Section 5 as applied to $\lambda^{\rightarrow}_{\mathrm{P,S}}$. In particular, we consider an (N, k) sharing scheme over the finite field $\mathbb{F}_p$, where we require $N \geq 2k$. We define $\mathcal{Y} = \{\mathrm{int}, \mathrm{bool}, \mathrm{unit}\}$, where each is concretely represented by $\mathbb{F}_p$ (and assumed to be tagged); values in bool will be restricted to $\{0, 1\} \subset \mathbb{F}_p$, and values in unit restricted to $\{0\}$. Our primitive operations are addition and multiplication modulo p, on int; and logical operators, on bool. (Without loss of generality, we describe only int and its operations.) In addition, for any $Y \in \mathcal{Y}$, we specify $\mathrm{op}_{\mathrm{Br}(Y)}(b, y_1, y_2) = by_1 + (1-b)y_2$; we define its secure implementation similarly (invoking the secure equivalents of addition and multiplication); so, without loss of generality, we will omit $\mathrm{op}_{\mathrm{Br}(Y)}$ from our discussion below (its correctness and security properties follow immediately from those of the constituent operations). We also define the set M of protocol messages to be $\mathbb{F}_p$. We define the set of "hidden equivalents" $\mathcal{E}_{\mathrm{S}}(\mathbb{F}_p)$ to be $\mathbb{F}_p^N$. During computations, we will be concerned specifically with inhabitants of $\mathcal{E}_{\mathrm{S}}(\mathbb{F}_p)$ that represent each of the N servers' shares of some base value. Thus, for $y \in \mathbb{F}_p$, we define:

$$\mathcal{E}_{\mathrm{S}}(y) = \{(p(1), \ldots, p(N)) : p \in \mathbb{F}_p[t], \deg p = k-1, p(0) = y\}$$

We also note that for any given $\tilde{y} = (y_1, \ldots, y_n)$, by polynomial interpolation, there is exactly one y with $\tilde{y} \in \mathcal{E}_{\mathrm{S}}(y)$; thus, $\{\mathcal{E}_{\mathrm{S}}(y) : y \in \mathrm{int}\}$ is a partition of $\mathcal{E}_{\mathrm{S}}(\mathrm{int})$. Apart from the initial secret sharing, there is no initialization phase, so we let $\mathcal{I}$ be the singleton set $\{()\}$.

The hiding, unhiding, and primitive operations are defined using the standard Shamir secret sharing constructions. In particular:

- $\mathrm{Enc}_{\mathrm{S}}(y, ()) = ((y_1, \ldots, y_N), \{(C, S_i, y_i) : i \in \{1, \ldots, N\}\})$, where $y_i = y + \sum_{j=1}^{k-1} r_j i^j$, with the r_j drawn uniformly at random from $\mathbb{F}_p$. (To share a value y, the client choose a degree-$(k-1)$ polynomial $p(t)$ uniformly at random such that $p(0) = y$, then sends $p(i)$ to each server i.)

- $\mathrm{Dec}_{\mathrm{S}}((y_1, \ldots, y_N), ()) = ((V_k^{-1})_1 \cdot (y_1, \ldots, y_k), \{(S_i, S_j, y_i)\}_{1 \leq i,j \leq N})$, where $(V_k^{-1})_1$ is the first row of the inverse Vandermonde matrix, $(V_k)_{i,j} = i^{j-1}$. (To recover a value y, the servers exchange all shares, then recompute the original value by polynomial interpolation.)

- $\mathrm{Enc}_{\mathrm{P,S}}(+)(y, (z_1, \ldots, z_N), ()) = ((y + z_1, \ldots, y + z_N), ())$. (To add a public constant to a shared value, each server simply adds that constant to its share.)

- $\mathrm{Enc}_{\mathrm{S,S}}(+)((y_1, \ldots, y_N), (z_1, \ldots, z_N), ()) = ((y_1 + z_1, \ldots, y_N + z_N), ())$. (To add two shared values together, each server simply adds that constant to its share.)

- $\mathrm{Enc}_{\mathrm{P,S}}(*)(y, (z_1, \ldots, z_N)) = ((y \cdot z_1, \ldots, y \cdot z_N), ())$. (To multiply a shared value by a public constant, each server simply multiplies its share by that constant.)

- $\mathrm{Enc}_{\mathrm{S,S}}(*)((y_1, \ldots, y_N), (z_1, \ldots, z_N), ()) = ((d_1, \ldots, d_N), \{(S_i, S_j, h_{i,j})\}_{1 \leq i \leq 2k, 1 \leq j \leq N})$, where $h_{i,j} = y_i z_i + \sum_{a=1}^{k-1} r_i^{(a)} j^a$ and $d_j = (V_{2k}^{-1})_1 \cdot (h_{1,j}, \ldots, h_{2k,j})$, with the $r_i^{(a)}$ drawn uniformly at random from $\mathbb{F}_p$. (To multiply two shared values together, each server i multiplies its shares pointwise, and communicates a new sharing of that product, $(h_{i,1}, \ldots, h_{i,N})$; then, the servers collectively perform polynomial interpolation on the resulting degree-$2k$ polynomial to obtain shares of the original product.)

Now, since any base value has only one distribution that can result from using uniform randomness (namely, the uniform distribution over all valid sharings), we define the set of safe distributions to contain only this one:

$$\hat{\mathcal{E}}^{\iota}_{\mathrm{S}}(y) = \{\mathcal{D}(y)\} = \{\pi_1(\mathrm{Enc}_{\mathrm{S}}(y))\}$$

We also define $\mathcal{A}$, the family of valid untrusted subsets of the servers, to include exactly those subsets with cardinality less than k, and we specify that the system should provide information-theoretic security.

Assuming this specification, we can now prove the required correctness and indistinguishability properties of the platform. In particular, the following properties are obtained by direct computation from the definitions above:

- $\mathrm{Enc}_{\mathrm{P,S}}(+)(c, \mathcal{D}(y), ()) = \mathcal{D}(c + y)$
- $\mathrm{Enc}_{\mathrm{S,S}}(+)(\mathcal{D}(y), \mathcal{D}(z), ()) = \mathcal{D}(y + z)$
- $\mathrm{Enc}_{\mathrm{P,S}}(*)(c, \mathcal{D}(y), ()) = \mathcal{D}(cy)$
- $\mathrm{Enc}_{\mathrm{S,S}}(*)(\mathcal{D}(y), \mathcal{D}(z), ()) = \mathcal{D}(yz)$

The correctness requirements are therefore immediate. In addition, it is known that for a uniformly random polynomial p of degree at least $k-1$ over a finite field, the sequence of values $(p(i_1), \ldots, p(i_m))$ for $m < k$ is also uniformly random. Thus:

- For any valid set $A = \{S_{i_1}, \ldots, S_{i_m}\} \in \mathcal{A}$ of untrusted servers, and for any $y, z \in \mathbb{F}_p$, $(\mathcal{D}(y)_{i_1}, \ldots, \mathcal{D}(y)_{i_m}) = (\mathcal{D}(z)_{i_1}, \ldots, \mathcal{D}(z)_{i_m})$, as joint random variables.

Given this fact, the indistinguishability requirements are now straightforward to show:

- For the initial sharing Enc, we have $\Pi_A(\pi_2(\mathrm{Enc}(y), ())) = \Pi_A(\pi_2(\mathrm{Enc}(z), ()))$ (as distributions), and thus by definition they are information-theoretically indistinguishable.
- For any Dec operation, the only values in the trace are shares of y, whose distribution is a function of $\mathcal{D}(y)$. Thus, if two traces invoke Dec on equal values y, the distributions are identical (and hence, as above, information-theoretically indistinguishable by definition).
- For the secure primitives, the only one that produces a nonempty trace is $\mathrm{Enc}_{\mathrm{S,S}}(*)$. In that case, $\Pi_A(\pi_2(\mathrm{Enc}_{\mathrm{S,S}}(*)(\mathcal{D}(y), \mathcal{D}(z), ())))$ is a function of $(h_{i,j})_{S_j \in A}$. But this is precisely the concatenation of $\Pi_A(\mathcal{D}(y_i z_i))$ for all i; and each $\Pi_A(\mathcal{D}(y_i z_i))$, as above, is uniformly random, independent of $y_i z_i$. Hence the traces again are information-theoretically indistinguishable.

Thus, we conclude that Shamir secret sharing is a secure execution platform.

B.2 Fully homomorphic encryption

In addition to secure multiparty computation, a variety of homomorphic encryption schemes can also serve as secure execution platforms for standard primitive operations. In particular, we will now show that any fully homomorphic encryption scheme, is a secure execution platform, achieving security against a computationally-bounded adversary. Although traditionally the operations provided under fully homomorphic encryption would be a complete set of circuit gates, in order to provide a better analogy with secret sharing, we assume the operations of the cryptosystem work over $\mathbb{Z}_{2^k}$; they may be implemented in terms of the underlying circuit operations. In fully homomorphic encryption, the number of servers, N, is 1; the client simply sends encrypted values to the server, and the server performs the computation homomorphically, returning the encrypted result. As above, we define $\mathcal{Y} = \{\mathrm{int}, \mathrm{bool}, \mathrm{unit}\}$, where each is concretely represented by $\mathbb{Z}_{2^k}$ (and assumed to be tagged); values in bool will be restricted to $\{0, 1\} \subset \mathbb{Z}_{2^k}$, and values in unit restricted to $\{0\}$. Our primitive operations are addition and multiplication in the ring, on int; and logical operators, on bool. (Without loss of generality, we describe only int and its operations.) In addition, we define the branching operators by arithmetization, as above: $\mathrm{op}_{\mathrm{Br}(Y)}(b, y_1, y_2) = by_1 + (1-b)y_2$.

The initialization step is just $\mathrm{Init}() = \mathsf{KeyGen}(\lambda) = (\mathrm{sk}, \mathrm{pk})$ generating the secret/public key pair, where λ is the security parameter to the system; the set M of messages in M consists of plaintexts and ciphertexts. To begin the computation, the client sends the public key to the server (i.e., $\Pi_{\{S_1\}}(\iota)$ here is $\Pi_{\{S_1\}}((\mathrm{sk}, \mathrm{pk})) = \mathrm{pk}$), then encrypts the initial values and and sends the corresponding ciphertexts to the server (i.e., $\mathrm{Enc}_{\mathrm{S}}(y, (\mathrm{sk}, \mathrm{pk})) = (\mathsf{Enc}(\mathrm{pk}, y), \{(C, S_1, \tilde{y})\})$). During the computation, the server itself performs additions and multiplications on the ciphertexts by homomorphically evaluating the corresponding circuits, producing no communication trace with the client (i.e., $\mathrm{Enc}_{\mathrm{S,S}}(\mathrm{op})(\tilde{y}_1, \tilde{y}_2, (\mathrm{sk}, \mathrm{pk})) = (\mathsf{Eval}(\mathrm{pk}, \mathrm{op}, \tilde{y}_1, \tilde{y}_2), \varepsilon)$); when one of the operands is a public value (i.e., $\mathrm{Enc}_{\mathrm{S,P}}$, $\mathrm{Enc}_{\mathrm{P,S}}$), the server simply "hides" it using $\mathsf{Enc}(\mathrm{pk}, \cdot)$, and then uses $\mathrm{Enc}_{\mathrm{S,S}}$. For `reveal` operations, the server sends back to the client a ciphertext to be decrypted, and the corresponding plaintext (some base value) is returned to the server (i.e., $\mathrm{Dec}_{\mathrm{S}}(\tilde{y}, (\mathrm{sk}, \mathrm{pk})) = (y, \{(S_1, C, \tilde{y}), (C, S_1, y)\})$ where $y = \mathsf{Dec}(\mathrm{sk}, \tilde{y}_i)$). Finally, given these operations, we define $\hat{\mathcal{E}}^{\iota}_{\mathrm{S}}(y)$ tautologically, as the set of all distributions that result when a computation (yielding y on the corresponding plaintexts) is performed on values originating from $\mathsf{Enc}(\mathrm{pk}, \cdot)$. We can then define $\mathcal{E}_{\mathrm{S}}(y)$ naturally as the union of the supports of $\hat{\mathcal{E}}^{\iota}_{\mathrm{S}}(y)$.

Correctness of the primitives now follows immediately from the homomorphic properties of the encryption scheme. Indistinguishability is immediate for partial traces derived from op_i, since these operations produce empty traces. For the other partial traces (i.e., the initial encryptions $\mathrm{Enc}_{\mathrm{S}}$), indistinguishability follows from CPA-security of the encryption scheme, since the only values in the traces are the encryptions of each of the bits of the secret client inputs. Thus, fully homomorphic encryption is a secure execution platform, and as above, all of the results of Section 5 hold of programs in $\lambda^{\rightarrow}_{\mathrm{P,S}}$ when it is given the semantics of fully homomorphic encryption (now obtaining security guarantees against a computationally bounded adversary, as determined by the semantic security properties of the cryptosystem).