Extensible Access Control with Authorization Contracts

Scott Moore[*] Christos Dimoulas[*] Robert Bruce Findler[†] Matthew Flatt[‡] Stephen Chong[*]
[*]Harvard University [†]Northwestern University [‡]University of Utah
{sdmoore,chrdimo,chong}@seas.harvard.edu robby@eecs.northwestern.edu mflatt@cs.utah.edu

Abstract

Existing programming language access control frameworks do not meet the needs of all software components. We propose an expressive framework for implementing access control monitors for components. The basis of the framework is a novel concept: the *authority environment*. An authority environment associates rights with an execution context. The building blocks of access control monitors in our framework are *authorization contracts*: software contracts that manage authority environments. We demonstrate the expressiveness of our framework by implementing a diverse set of existing access control mechanisms and writing custom access control monitors for three realistic case studies.

1. Introduction

An access control monitor mediates requests to call sensitive operations and allows each call if and only if the request possesses the necessary rights to call the operation. Broadly speaking, when an access control mechanism is presented with a call to a sensitive operation, it must be able to answer two questions. First, which rights are required for the call? And second, which rights does the request possess? The design of an access control mechanism specifies, implicitly or explicitly, the answers to these questions.

For example, Unix file permissions describe which users are allowed to call which operations on a file. The access control mechanism uses file permissions to determine what rights are necessary to call different sensitive operations. Each Unix process executes on behalf of a specific user, and a request to call an operation possesses the same rights as the user of the process that issues the request. Thus, file permissions answer the first question, and the rights of the user associated with a process answer the second question. Importantly, Unix associates users and processes in two different ways. By default, a new process runs on behalf of the same user as the process that spawned it. But a process can run on behalf of a different user if it runs an executable that has the `setuid` bit set. When a process invokes a `setuid` executable, the operating system launches a new process to run the executable and associates the new process with the user that owns the executable, rather than the user that invoked it. Hence, this feature creates services that provide restricted access to resources that an invoking user could not otherwise access.

Similar to operating systems, software components also need access control mechanisms to prevent unauthorized clients from calling sensitive operations while allowing authorized ones to do so. Thus, when responding to a request to call a sensitive operation, access control mechanisms for components must be able to answer the same two questions: which rights are necessary for the call and which rights the request possesses.

However, access control needs of components vary, and it is impossible to choose a single answer to these questions that satisfies all component authors. To make things worse, access control mechanisms for general purpose programming languages have made design choices that are not suitable for all application domains and are typically mutually incompatible. For example, Java stack inspection [42] determines the rights associated with a call site by walking the stack from the current stack frame. In contrast, object-capability languages (e.g., E [27] and Caja [28]) determine rights by the lexical structure of the program: a code may call operations on exactly those resources that are reachable from variables in the code's text.

In this paper we propose a new, extensible access control framework that allows component authors to design access control monitors that suit their needs. The framework supports the design and implementation of many different novel and existing access control monitors for software components. Moreover, because different monitors are implemented using a common framework, different software components within the same application can use different access control mechanisms.

The framework builds on a novel concept: the *authority environment*. Just as each execution context has a variable environment that maps variable identifiers to values, each execution context has an authority environment that associates the context with its rights to call operations. The rights that a call to a sensitive operation possesses are those possessed by the authority environment of the call's execution context.

This technical report expands [30] with additional examples and code listings, along with a formal definition and proof of complete monitoring for the contract system described in Section 3.1.

By analogy with dynamic and lexical scoping of variable environments, we identify two ways in which an execution context can receive authority:

1. *dynamically,* by inheriting the authority environment of the surrounding execution context, and
2. *lexically,* by capturing the authority environment of the execution context where it is defined.

Returning to the Unix file system example, a process receives authority dynamically when it inherits the user of the process that launched it. A process receives authority "lexically" when it runs a `setuid` executable.

Based on the correspondence with variable scoping, we define a framework for designing access control monitors as sets of monitor actions that manipulate authority environments (§3). We implement our framework as a library for Racket [17] without changes to the language's runtime. We use *higher-order contracts* [16] to specify where an access control monitor should interpose on a program and how it should manage authority environments. Contracts are executable specifications attached to software components that support separation of concerns by removing defensive checks from code implementing functionality [24–26]. In the same way, our *authorization contracts* separate the task of access control from the program's functionality.

The design of this framework presents four major contributions:

1. the introduction of *authority environments* as a unifying concept for access control mechanisms (§2),
2. the introduction of *context contracts* to check and enforce properties of execution contexts (§3.1),
3. a novel *authorization logic* for representing and querying authority in authority environments (§3.2), and
4. *authorization contracts* that specialize context contracts for managing authority environments and enforcing access control policies expressed in the logic (§3.3).

We have used the framework to implement diverse access control mechanisms: discretionary access control, stack inspection, history-based access control, and object-capabilities (§4). We demonstrate the practicality of our approach with three realistic case studies (§5).

2. Authority Environments

In this section, we introduce *authority environments* as a unifying concept for access control. First, we review the differences between lexical and dynamic scoping (§2.1). Then we describe the connection between lexical and dynamic scoping and access control (§2.2) and show how we can use scoping in the design of a framework for writing access control monitors (§2.3). Throughout, we use small examples in the Racket programming language [17].

2.1 Lexical and Dynamic Scoping

The scope of a variable binding is the spatial and temporal part of the program in which it is visible. A common way to categorize strategies for assigning scopes to bindings is as either *lexical* or *dynamic*. Earlier work distinguishes between the *scope* of a binding, which describes where the binding is visible in the program text, and the *extent* of a binding, which describes when the binding is visible during execution. Dynamic scope often refers to bindings that have dynamic extent and "indefinite" scope. Here, we use dynamic scope to refer to bindings that have dynamic extent and lexical scope, also called "fluid" scope [18, 36, 37].

Under lexical scoping, a variable refers to the binding from its closest binder in the textual structure of the program. For example, in the Racket expression below, the variable `x` in function `f` refers to the binding in the outer-most `let` statement. The evaluation of this expression returns `0` since the inner-most `let` statement has no effect on the value `x` binds within `f`.

```
(let ([x 0])
  (let ([f (lambda () x)])
    (let ([x 42])
      (f))))
```

In a programming language with fluid scoping, programmers can instead associate a binding with the dynamic extent of an expression. That binding is visible to any code that runs in the dynamic extent of the expression. For example, the following Racket expression defines a new fluidly-scoped variable `x` with default value `0`. The `parameterize` expression binds `x` to the value `42` in the dynamic extent of its body. The variable `x` in the body of `f` refers to the most recent binding rather than the closest one in the program text. Since `f` is invoked within the `parameterize` expression, the program evaluates to `42` instead of `0`.

```
(let ([x (make-parameter 0)])
  (let ([f (lambda () (x))])
    (parameterize ([x 42])
      (f))))
```

Fluid scoping is a useful programming construct because it allows the context of an expression to communicate with its callees without explicitly threading arguments through the program. For example, a library function for printing may offer a parameter that determines the standard output file. Instead of threading that file as an argument through every function call leading to the `printf` routine, a client program can instead set the parameter once and all calls to `printf` in the body of the program see the client-specified file.

2.2 Scoping for Access Control

This ability to pass contextual information from an execution context to an eventual callee closely matches the problem of correctly determining the authority of a request to call a sensitive operation. To demonstrate this relationship, consider the design of a web application with multiple users. A key component of this application is a login function that authenticates users and executes code on their behalf:

```
(define (login user guess onSuccess)
  (if (check-password? user guess)
      (run-as-user user onSuccess)
      (error "Wrong password!")))
```

This `login` function takes three arguments: the `user` attempting to authenticate, the password `guess`, and a callback `onSuccess` to invoke with the user's rights if the password is correct. After checking the password, the login function changes the state of the program to indicate that the current user is now `user` and then calls `onSuccess`.

The body of `onSuccess` may attempt to access sensitive resources. For example, it may try to update a user's profile. To avoid an unauthorized update, the `update-profile` function checks whether the current user has sufficient rights:

```
(define (update-profile profileUser text)
  (if (can-update? currentUser profileUser)
      ...
      (error "Unauthorized!")))
```

Function `can-update?` compares the current user with the user who owns the profile to determine whether the update is authorized. This code thus implicitly uses the authority of its context, i.e., the current user, in much the same way that code accesses the dynamically scoped bindings from its context. By managing authority as an implicit context in this way, we can avoid modifying the code between the decision to run a computation with particular authority and the call to the sensitive operation. This has two advantages. First, threading authority explicitly through the program reduces extensibility, since third party code would need to be aware of and correctly handle authority explicitly. Second, if the code is untrustworthy, it might attempt to subvert the access control checks that protect the sensitive operation by fabricating its own authority.

Another requirement of the security of this application is that only code running with the authority of the main loop is allowed to switch users. According to the Principle of Least Privilege [33], we should further limit the code that is allowed to switch users to just the `login` function, and switch to an unprivileged user for the rest the program. Crucially, the body of the `login` function must still use the authority that was in its environment when it was created, i.e., the authority of the main loop. In a sense, for `login`, we wish to close over the authority of the main loop, in the same way that closures capture lexically scoped bindings.

To achieve this, we build on the analogy between scoping and access control and introduce the concept of an *authority environment*. An authority environment associates rights with an execution context, just as a variable environment associates bindings with an execution context. Just like variable environments, authority environments can be captured and associated with code, updated, and extended with new bindings for the dynamic extent of a computation. In this application, the authority environment of an execution context records the

```
(define-monitor users
 (monitor-interface
   setuid/c chuser/c checkuser/c)
 (action
  [chuser/c (user)
  #:on-create (do-create)
  #:on-apply  (do-apply
    #:check (≽ @ current-principal user
                user)
    #:set-principal user)]
  [checkuser/c (user)
  #:on-create (do-create)
  #:on-apply  (do-apply
    #:check (≽ @ current-principal user
                user))]
  [setuid/c
  #:on-create (do-create)
  #:on-apply  (do-apply
    #:set-principal closure-principal)]))
```

Figure 1. Defining a simple access control monitor

user on whose behalf the code executes. Section 2.3 shows how authority environments help enforce access control in our running example, including how to create a secure `login` function. Section 3 generalizes authority environments so that we can express a wide variety of access control mechanisms.

2.3 From Access Control to Authorization Contracts

Using the concept of an authority environment, we build an access control monitor that manipulates and inspects the authority environments of the example web application. The monitor consists of *actions* that describe how events in the execution of the application interact with its authority environment. We describe our framework for defining monitors in detail in Section 4. Here we explain only the features relevant to the example.

Figure 1 shows our example monitor. The monitor specifies three actions: `setuid/c`, `chuser/c`, and `checkuser/c`. Each action defines a higher-order function contract [16]. When one of these contracts is attached to a function, the contract captures the current authority environment and associates it with the function. When the function is called, the contract has access to both the authority environment at the call site and the authority environment that it has captured. The monitor configures each action-contract with two hooks: `#:on-create` and `#:on-apply`. By changing these hooks, monitor designers can implement actions that implement different forms of "lexically" and dynamically scoped authority environments.

Action `chuser/c` is parameterized with an argument `user` that identifies the user whose authority should be used during the execution of the body of a contracted function. The `#:on-apply` hook for `chuser/c` ignores the authority it has closed over and sets the active principal to `user` for the dynamic extent of the body of the contracted function, but only

if the `#:check` holds, that is, the `current-principal` has authority over user `user`. Otherwise, it raises a security violation as a contract violation. Monitor action `checkuser/c` is also parameterized with a `user`. Upon a call of its contracted function, it checks that the `current-principal` has authority over `user`. If the check succeeds, the action does not change the authority environment. If the check fails, the action raises a security violation. The final monitor action, `setuid/c`, creates an authority closure: calling a function with this contract changes the current principal in the authority environment to the closed-over principal `closure-principal` for the dynamic extent of the function's body.

Using the monitor, we can now reimplement the web application. First, we can replace code that defensively performs authorization checks with contracts that enforce authority requirements:

```
(define/contract
  (update-profile someUser text)
  (->a ([user principal?] [text string?])
       #:auth (user) (checkuser/c user)
       any))
  ...)
```

This revised implementation of `update-profile` uses the Racket form `define/contract` to attach a contract to the `update-profile` function. This contract is a *dependent contract* [16] for a function. It says that `update-profile` takes two arguments: `user`, which must be a `principal?`, and `text`, which must be a `string?`. The keyword argument[1] `#:auth (user)(checkuser/c user)` says that the authorization contract for this function depends on the `user` argument and attaches the contract `(checkuser/c user)` to the function. Finally, the range of this contract is `any`, making no requirements on the return values. The definition of the function can now elide the authorization check.

We also revise the implementation of `login`:

```
(define/contract
  (login user guess onSuccess)
  (->a ([user principal?] [guess string?]
        [onSuccess (user) (chuser/c user)])
       #:auth () setuid/c any)
  (if (check-password? user guess)
      (onSuccess)
      (error "Wrong password")))
```

The keyword argument `#:auth () setuid/c` attaches the `setuid/c` action to the login function, capturing the authority of the program context where it is created. This allows the `login` function to execute with the captured authority and thus allows the application to switch to a less privileged principal without losing the ability to safely authenticate as a different user. In the original implementation, `login` uses a hand-rolled function `run-as-user` to confine `onSuccess` within the authority of `user`. In the revised code, `login` can invoke `onSuccess` directly. The contract on the `onSuccess` argument attaches the action `(chuser/c user)` to the function. This ensures that any call to `onSuccess` has the correct authority.

Rewriting this application to use authorization contracts makes the authorization requirements of each function clear, while also simplifying its implementation by removing authorization code that was spread throughout the program.

[1] In Racket, a keyword argument is a (possibly optional) argument passed by keyword instead of position. A keyword is a symbol starting with `#:`.

3. A Framework for Access Control

In this section, we present the general design of our framework with a formal model. First, we show how we extend existing higher-order function contracts to *context contracts* that check and modify information about their execution context (§3.1). Context contracts are expressive enough to enforce a wide range of properties. However, this flexibility makes it difficult to use them to implement and reason about access control. To free users from this burden, our framework provides a specialized interface for defining *authorization contracts*. The interface simplifies the definition of context contracts for access control in two ways. First, it specifies a common representation for authorization environments (§3.2). At the core of this representation is a novel authorization logic that describes how authority captured in a closure may be used. Second, it defines combinators for building authorization contracts (§3.3). Authorization contracts are specializations of context contracts that use the authorization logic to succinctly describe how they manage authority environments.

3.1 A Contract System with Context Contracts

We model higher-order contracts and context contracts as extensions to an applied lambda calculus with modules and parameters, which implement dynamic binding. Figure 2 describes the syntax of our model. Figure 3 gives the reduction semantics of the model. Evaluation contexts that are not presented in Figure 2 are standard and enforce call-by-value, left-to-right evaluation. The type system and the definition of evaluation contexts are given in Appendix A. Though we omit the extension for clarity, we have also extended the model with first class continuations, following Takikawa et al. [39], because our implementation language supports them and they interact in interesting ways with our framework.

The semantics of common language features is standard. Below, we explain the semantics of modules, parameters, higher-order contracts, and context contracts.

3.1.1 Modules

A program p is a sequence of modules followed by a top-level expression. A module simultaneously defines a collection of values owned by a single component and a set of contracts for those values. Each module

$$\text{module } \ell \text{ exports } x_{v_1} \text{ with } x_{c_1}, \ldots \text{ where } y_1 = e_1, \ldots$$

Surface syntax

$$
\begin{array}{rcl}
p & ::= & m\,;p \mid e \\
m & ::= & \mathsf{module}\,\ell\,\mathsf{exports}\,x\,\mathsf{with}\,x,\ldots\,\mathsf{where}\,x=e,\ldots \\
e & ::= & v \mid e\;e \mid \mu\,x:\tau.\,e \mid \mathsf{let}\,x=e\,\mathsf{in}\,e \mid e\oplus e \\
 & \mid & e\le e \mid \mathsf{if}\,e\,\mathsf{then}\,e\,\mathsf{else}\,e \mid \mathsf{make\text{-}parameter}\,e \\
 & \mid & \mathsf{parameterize}\,e=e\,\mathsf{in}\,e \mid ?e \mid e:=e \\
 & \mid & \mathsf{flat/c}(e) \mid \mathsf{param/c}(e) \mid e:\tau\to(e)\,e \\
 & \mid & e:\tau\to_{\mathsf{a}}(\lambda\,x:\tau.\,e)\,e \\
 & \mid & \mathsf{ctx/c}(e,(e\Rightarrow e\leftarrow e),\ldots,e,(e\Rightarrow e\leftarrow e),\ldots) \\
v & ::= & () \mid n \mid \#\mathsf{t} \mid \#\mathsf{f} \mid \lambda\,x:\tau.\,e \mid c \\
c & ::= & \mathsf{flat/c}(v) \mid \mathsf{param/c}(c) \mid c:\tau\to(c)\,c \\
 & \mid & c:\tau\to_{\mathsf{a}}(\lambda\,x:\tau.\,e)\,c \\
 & \mid & \mathsf{ctx/c}(v,(v\Rightarrow v\leftarrow v)\ldots,v,(v\Rightarrow v\leftarrow v)\ldots) \\
\tau & ::= & \beta \mid \tau\to\tau \mid \tau\;\mathsf{param} \mid \tau\;\mathsf{ctc} \mid \mathsf{ctx\;ctc} \\
\beta & ::= & \mathsf{Unit} \mid \mathsf{Int} \mid \mathsf{Bool}
\end{array}
$$

Expanded syntax

$$
\begin{array}{rcl}
e & ::= & \ldots \mid {}^{\ell}\mathsf{mon}^{k}_{j}(e,e) \mid \mathsf{check}^{k}_{j}(e,e) \mid \mathsf{error}^{k}_{j} \\
 & \mid & \mathsf{guard}_{j}\,(e,v,v,e) \mid \mathsf{install/p}_{j}\,(v,e,e) \\
v & ::= & \ldots \mid \mathsf{p}(r) \mid {}^{\ell}\mathsf{param/p}^{k}_{j}(c,v) \\
 & \mid & {}^{\ell}\mathsf{ctx/p}^{k}_{j}\,(v,(v\Rightarrow v\leftarrow v),\ldots,v) \\
 & \mid & \mathsf{install/p}_{j}\,(v,v,v)
\end{array}
$$

Selected evaluation contexts

$$
\begin{array}{rcl}
E & ::= & \ldots \mid \mathsf{guard}_{j}\,(E,v,v,e) \\
 & \mid & \mathsf{install/p}_{j}\,(v,e,E) \mid \mathsf{install/p}_{j}\,(v,E,v)
\end{array}
$$

Figure 2. Syntax

has a label ℓ and defines a set of values $y_1,\ldots$. Values within the module are visible only to subsequent modules in the program if they are exported. An export declaration $x_{v_i}\,\mathsf{with}\,x_{c_i}$ binds x_{v_i} in the rest of the program to the value defined as x_{v_i} in the where clause, but only after attaching to it the contract defined as x_{c_i} in the where clause. Meta-function import, shown in Figure 3, substitutes occurrences of x_{v_i} in the rest of the program with monitored values ${}^{\ell}\mathsf{mon}^{k}_{j}(c,v)$ that enforce the contract.

3.1.2 Higher-order Contracts

Term ${}^{\ell}\mathsf{mon}^{k}_{j}(c,v)$ attaches contract c to value v and monitors whether uses of v satisfy the contract. Labels j, k, and ℓ, identify, respectively, the module that imposed the contract, the module that provided the value, and the module that is the client of the value. The top-level expression of a program is identified by the distinguished label ℓ_0. These labels are used to assign *blame* when a contract is violated. The simplest contract is a flat contract $\mathsf{flat/c}(v_c)$ that takes a predicate v_c as an argument. Flat contracts can be applied only to values of base types Int, Bool, and Unit. When the contract is attached to a value, the predicate is applied to the value. If the predicate returns true, the value passes to its context. Otherwise, the contract system stops the program and raises an error blaming the provider of the value.

Contract $c_d:\tau\to(c_c)\;c_r$ is a higher-order function contract. It specifies a contract c_d for the domain of the function and a contract c_r for the range of the function. In addition, it specifies a *context contract* c_c for the function. Context contracts, novel to this work, are higher-order contracts that enforce restrictions on the execution context of function calls. They are described in detail below. Attaching contract $c_d:\tau\to(c_c)\;c_r$ to a function returns a new value that enforces contracts on the argument and results of the function and applies the context contract c_c.

Contract $c_d:\tau\to_{\mathsf{a}}(\lambda\,x:\tau.\,e_c)\;v_r$ is an indy-dependent[2] contract for higher-order functions. This contract corresponds to the `->a` contracts from Section 2. The contract is *dependent* since the contract uses the argument of a contracted function to choose a context contract for the function. Applying a function v with this contract has four steps. First, the argument is wrapped with the contract for the domain, c_d. Second, the wrapped argument is passed to the function $\lambda\,x:\tau.\,e_c$ to construct a context contract. Third, the resulting context contract is attached to v, which is applied to the wrapped argument. Finally, the contract for the range, c_r, is attached to the result of the application.

3.1.3 Parameters

Parameters are first-class values that can be used to access and install dynamic bindings. Parameters implement fluid scope because access to their dynamic bindings is controlled lexically by access to the parameter itself. The expression make-parameter e creates a new parameter $\mathsf{p}(r)$ with default value the result of e, where r is a fresh tag uniquely identifying the parameter. The default value is recorded in the store σ. Term $\mathsf{parameterize}\,\mathsf{p}(r)=e_1\,\mathsf{in}\,e_2$ installs the result of e_1 as the new value of the parameter $\mathsf{p}(r)$ for the dynamic extent of e_2. Accessing the value of a parameter with term $?\mathsf{p}(r)$ returns the value of the closest enclosing parameterize for $\mathsf{p}(r)$ in the current evaluation context. If there is no such term, it returns the current value for r in the store. Similarly, term $\mathsf{p}(r):=v$ mutates the current binding for the parameter, updating the parameter associated with either the closest enclosing parameterize for $\mathsf{p}(r)$ in the current evaluation context or the value for r in the store, if there is no such expression.

The parameter contract $\mathsf{param/c}(c)$ is a higher-order contract that restricts uses of a parameter. A contracted parameter v reduces to a proxy ${}^{\ell}\mathsf{param/p}^{k}_{j}(c,v)$ that records the labels of the contract, provider, and client modules and intercepts uses of the parameter to enforce that values bound to the parameter meet contract c.

[2] An indy-dependent contract is a dependent function contract that uses the "indy" strategy for blame assignment [9].

$\langle \text{module } \ell \text{ exports } x_1 \text{ with } x_{c_1},\ldots \text{ where } y_1 = v_1,\ldots,y_n = v_n,y_{e_1} = e_1,\ldots y_{e_m} = e_m;\ p,\sigma\rangle$
$\quad\to\quad \langle \text{module } \ell \text{ exports } x_1 \text{ with } x_{c_1},\ldots \text{ where } y_1 = v_1,\ldots,y_n = v_n,y_{e_1} = \{^{v_i}/_{y_i}\}e_1,y_{e_2} = e_2,\ldots y_{e_m} = e_m;\ p,\sigma\rangle$

$\langle \text{module } \ell \text{ exports } x_1 \text{ with } x_{c_1},\ldots \text{ where } \ldots,x_1 = v_1,\ldots,x_{c_1} = c_1,\ldots;\ p,\sigma\rangle$
$\quad\to\quad \langle \text{import}[\![\ell,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),p]\!],\sigma\rangle$

$\langle E[(\lambda\, x : \tau.\ e)\ v],\sigma\rangle$	$\to$	$\langle E[\{^v/_x\}e],\sigma\rangle$
$\langle E[\mu\, x : \tau.\ e],\sigma\rangle$	$\to$	$\langle E[\{^{\mu\, x:\tau.\ e}/_x\}e],\sigma\rangle$
$\langle E[\text{let } x = v \text{ in } e],\sigma\rangle$	$\to$	$\langle E[\{^v/_x\}e],\sigma\rangle$
$\langle E[\text{if \#t then } e_1 \text{ else } e_2],\sigma\rangle$	$\to$	$\langle E[e_1],\sigma\rangle$
$\langle E[\text{if \#f then } e_1 \text{ else } e_2],\sigma\rangle$	$\to$	$\langle E[e_2],\sigma\rangle$
$\langle E[v_1 \oplus v_2],\sigma\rangle$	$\to$	$\langle E[v],\sigma\rangle$ where $v = v_1 \oplus v_2$
$\langle E[v_1 \le v_2],\sigma\rangle$	$\to$	$\langle E[v],\sigma\rangle$ where $v = v_1 \le v_2$
$\langle E[\text{make-parameter } v],\sigma\rangle$	$\to$	$\langle E[\mathsf{p}(r)],\sigma[r \mapsto v]\rangle$ where r is fresh
$\langle E[?\mathsf{p}(r)],\sigma\rangle$	$\to$	$\langle v,\sigma\rangle$ where $\sigma(r) = v$ and E does not contain parameterize $\mathsf{p}(r) = v'$ in E'
$\langle E[\text{parameterize } \mathsf{p}(r) = v \text{ in } E'[?r]],\sigma\rangle$	$\to$	$\langle E[\text{parameterize } \mathsf{p}(r) = v \text{ in } E'[v]],\sigma\rangle$ where E' does not contain parameterize $\mathsf{p}(r) = v'$ in E''
$\langle E[\mathsf{p}(r) := v],\sigma\rangle$	$\to$	$\langle E[()],\sigma[r \mapsto v]\rangle$ where E does not contain parameterize $\mathsf{p}(r) = v'$ in E'
$\langle E[\text{parameterize } \mathsf{p}(r) = v \text{ in } E'[\mathsf{p}(r) := v']],\sigma\rangle$	$\to$	$\langle E[\text{parameterize } \mathsf{p}(r) = v' \text{ in } E'[v']],\sigma\rangle$ where E' does not contain parameterize $\mathsf{p}(r) = v''$ in E''
$\langle E[\text{parameterize } \mathsf{p}(r) = v \text{ in } v'],\sigma\rangle$	$\to$	$\langle E[v'],\sigma\rangle$
$\langle E[{}^{\ell}\mathsf{mon}^k_j(\mathsf{flat/c}(v_c),v)],\sigma\rangle$	$\to$	$\langle E[\mathsf{check}^k_{j,k}((v_c\ v),v)],\sigma\rangle$
$\langle E[{}^{\ell}\mathsf{mon}^k_j(\mathsf{param/c}(c),v)],\sigma\rangle$	$\to$	$\langle E[{}^{\ell}\mathsf{param/p}^k_j(c,v)],\sigma\rangle$
$\langle E[{}^{\ell}\mathsf{mon}^k_j(c_d : \tau \to (c_c)\ c_r,v)],\sigma\rangle$	$\to$	$\langle E[({}^{\ell}\mathsf{mon}^k_j(c_c,\lambda\, x : \tau_d.\ {}^{\ell}\mathsf{mon}^k_j(c_r,v_f\ {}^k\mathsf{mon}^{\ell}_j(c_d,x)))\ v)],\sigma\rangle$ where x is fresh
$\langle E[{}^{\ell}\mathsf{mon}^k_j(c_d : \tau \to_{\mathsf{a}} (\lambda\, x : \tau.\ e_c)\ c_r,v)],\sigma\rangle$	$\to$	$\langle E[\lambda\, y : \tau_d.\ {}^{\ell}\mathsf{mon}^k_j(\{^{{}^k\mathsf{mon}^{\ell}_j(c_d,y)}/_x\}e_c,{}^{\ell}\mathsf{mon}^k_j(c_r,v\ {}^k\mathsf{mon}^{\ell}_j(c_d,y)))],\sigma\rangle$ where y is fresh

$\langle E[{}^{\ell}\mathsf{mon}^k_j(\mathsf{ctx/c}(v_c,(v_{c_{g_1}} \Rightarrow v_{c_{p_1}} \leftarrow v_{c_{v_1}}),\ldots,v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\ldots),v)],\sigma\rangle \quad\to\quad \langle E[\mathsf{check}^k_j((v_c\ ()),e)],\sigma\rangle$
where $e = \mathsf{guard}_j\ ((v_{c_{g_1}}\ ()),v_{c_{p_1}},v_{c_{v_1}},\ldots\ \mathsf{guard}_j\ ((v_{c_{g_n}}\ ()),v_{c_{p_n}},v_{c_{v_n}},{}^{\ell}\mathsf{ctx/p}^k_j\ (v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\ldots,v)))$

$\langle E[({}^{\ell}\mathsf{ctx/p}^k_j\ (v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\ldots,v_f)\ v)],\sigma\rangle$
$\quad\to\quad \langle E[(\mathsf{check}^{\ell}_j((v_a\ ()),e)\ v)],\sigma\rangle$
where $e = \mathsf{guard}_j\ ((v_{a_{g_1}}\ ()),v_{a_{p_1}},v_{a_{v_1}},\ldots\ \mathsf{guard}_j\ ((v_{a_{g_n}}\ ()),v_{a_{p_n}},v_{a_{v_n}},v_f))$

$\langle E[\mathsf{guard}_j\ (\#\mathsf{f},v_p,v_v,e_f)],\sigma\rangle$	$\to$	$\langle E[e_f],\sigma\rangle$
$\langle E[\mathsf{guard}_j\ (\#\mathsf{t},v_p,v_v,e_f)],\sigma\rangle$	$\to$	$\langle E[\mathsf{install/p}_j\ (v_p,(v_v\ ()),e_f)],\sigma\rangle$
$\langle E[(\mathsf{install/p}_j\ (v_p,v_v,v_f)\ v)],\sigma\rangle$	$\to$	$\langle E[\text{parameterize } v_p = v_v \text{ in } (v_f\ v)],\sigma\rangle$
$\langle E[?{}^{\ell}\mathsf{param/p}^k_j(c,v_p)],\sigma\rangle$	$\to$	$\langle E[{}^{\ell}\mathsf{mon}^k_j(c,?v_p)],\sigma\rangle$
$\langle E[\text{parameterize } {}^{\ell}\mathsf{param/p}^k_j(c,v_p) = v \text{ in } e],\sigma\rangle$	$\to$	$\langle E[\text{parameterize } v_p = {}^k\mathsf{mon}^{\ell}_j(c,v) \text{ in } e],\sigma\rangle$
$\langle E[{}^{\ell}\mathsf{param/p}^k_j(c,v_p) := v],\sigma\rangle$	$\to$	$\langle E[v_p := {}^k\mathsf{mon}^{\ell}_j(c,v)],\sigma\rangle$
$\langle E[\mathsf{check}^k_j(\#\mathsf{t},v)],\sigma\rangle$	$\to$	$\langle E[v],\sigma\rangle$
$\langle E[\mathsf{check}^k_j(\#\mathsf{f},v)],\sigma\rangle$	$\to$	$\langle \mathsf{error}^k_j,\sigma\rangle$

$\text{import}[\![k,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),m_1;\ldots;m_n;e]\!] =$
$\quad \text{import}[\![k,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),m_1]\!];$
$\quad \ldots;$
$\quad \text{import}[\![k,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),m_n]\!];$
$\quad \text{import}[\![k,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),e]\!]$

$\text{import}[\![k,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),\text{module } \ell \text{ exports } x_{v_1} \text{ with } x_{c_1},\ldots \text{ where } y_1 = e_1,\ldots,y_n = e_n]\!] =$
$\quad \text{module } \ell \text{ exports } x_{v_1} \text{ with } x_{c_1},\ldots \text{ where } y_1 = \{^{{}^{\ell}\mathsf{mon}^k_k(c_i,v_i)}/_{x_i}\}e_1,\ldots,y_n = \{^{{}^{\ell}\mathsf{mon}^k_k(c_i,v_i)}/_{x_i}\}e_n$

$\text{import}[\![k,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),e]\!] \quad = \quad \{^{\ell_0\,{}^{\ell}\mathsf{mon}^k_k(c_i,v_i)}/_{x_i}\}e$

Figure 3. Reduction semantics

3.1.4 Context Contracts

To track properties of execution contexts, context contracts use parameters to install and access relevant state. A context contract interposes on programs at two key times: when the contract is attached to a function and when the contracted function is applied. At both times, the contract can inspect the current values of parameters to check that the current environment is satisfactory, capture the current value for later use, or change the parameterization of a call to the contracted function.

A context contract

$$\begin{array}{l}\mathsf{ctx/c}(v_c,(v_{g_c} \Rightarrow v_{p_c} \leftarrow v_{v_c}),\ldots,\\ \qquad v_a,(v_{g_a} \Rightarrow v_{p_a} \leftarrow v_{v_a}),\ldots)\end{array}$$

has four parts:

1. v_c, a predicate that checks whether the context is appropriate when the contract is attached,
2. $(v_{g_c} \Rightarrow v_{p_c} \leftarrow v_{v_c}),\ldots$, a list of *guarded parameterizations*, described below, to close over when the contract is attached,
3. v_a, a predicate that checks whether the context is appropriate when the contract function is called, and
4. $(v_{g_a} \Rightarrow v_{p_a} \leftarrow v_{v_a}),\ldots$, a list of guarded parameterizations to be installed around the body of the contracted function if the contract check succeeds.

The first two parts are evaluated when the contract is attached to a value. First, the predicate v_c is executed to allow the contract to check the current context. If the predicate returns #f, a contract error is raised blaming the client of the contract. Otherwise, each guarded parameterization $(v_{g_c} \Rightarrow v_{p_c} \leftarrow v_{v_c})$ from part 2 is evaluated in turn. Each guarded parameterization specifies a guard function v_{g_c}, a parameter v_{p_c}, and a value function v_{v_c}. If invoking the guard thunk v_{g_c} returns #t, the corresponding value thunk v_{v_c} is executed to produce a new value. This value is "closed over" and re-installed for parameter v_{p_c} when the contracted function is applied. The predicate v_a and the remaining parameterizations are recorded in a proxy ${}^{\ell}\mathsf{ctx/p}_j^k\,(v_a,(v_{g_a} \Rightarrow v_{p_a} \leftarrow v_{v_a}),\ldots,v)$.

The proxy enforces additional checks and parameterizations when the contracted function is called. First, the parameter values captured when the contract was attached are reinstalled. This gives the evaluation of the proxy and the function call access to some bindings from when the contract was attached, in addition to any bindings that are present in the current evaluation context. With these captured bindings in place, the proxy first evaluates the predicate v_a, which checks whether the current context is satisfactory. If the predicate returns false, a contract error is raised blaming the client ℓ. Otherwise, the guarded parameterizations of the proxy are evaluated in a similar fashion as before. However, any new bindings are installed just for the dynamic extent of the contracted function's call.

Figure 4 demonstrates context contracts with a small example. The example involves two context contracts, $outer/ctx$ and $inner/ctx$, that communicate via parameter p. The contracts ensure that function $inner$ can be applied only in the dynamic extent of the function returned by $outer$. Evaluating $(inner\ 42)$ results in a contract error $\mathsf{error}_{\ell}^{top}$ blaming the context that applied $inner$. Replacing this expression with $((outer\ inner)\ 42)$ evaluates to 42.

```
module ℓ exports inner with inner/c,
                 outer with outer/c
where inner = λ x : Int. x,
      outer = λ f : (Int → Int). λ x : Int. (f x),
      true = λ _ : Unit. #t,
      int/c = flat/c(λ _ : Int. #t),
      fun/c = λ ctx : ctx ctc. (int/c : Int → (ctx) int/c),
      any/ctx = ctx/c(true,true),
      any/c = (fun/c any/ctx),
      p = make-parameter #f,
      check/ctx = ctx/c(true,λ _ : Unit. ?p),
      enable/ctx = ctx/c(true,true,((true ⇒ p ← true))),
      enable/c = (fun/c enable/ctx),
      inner/c = (fun/c check/ctx),
      outer/c = (any/c : (Int→Int) → (any/ctx) enable/c);
(inner 42)
```

Figure 4. Context contracts enforcing nested applications.

```
module ℓ exports ...
where ...,
      cp = make-parameter #f,
      capture/ctx = ctx/c(true,
                          ((true ⇒ cp ← λ _ : Unit. ?p)),
                          true,
                          ((true ⇒ p ← λ _ : Unit. ?cp)))
      ...;
...
```

Figure 5. A context contract that closes over parameter p.

The ability to close over an environment is a key feature of authorization contracts. To see that context contracts can close over some part of the environment when a contract is applied, consider extending the example in Figure 4 with the contract $capture/c$ from Figure 5. This contract captures the value of parameter p when the contract is applied, and reinstates that value for the dynamic extent of subsequent applications of the contracted value.

3.1.5 Complete Monitoring

Our contract system satisfies *complete monitoring* [9], an important correctness criterion for contract systems. Complete monitoring guarantees that a contract system correctly assigns blame to components that violate their contracts and, crucially, that the contract system can interpose on all uses of a value in a component that did not create that value. This property makes contracts suitable for interposing on programs to enforce access control policies. Moreover, because the interposition is local to individual components, an access control monitor can be installed around a component without a global enforcement mechanism or the cooperation of other components.

Put differently, complete monitoring guarantees that contracts can enforce the same set of properties as reference monitors: an arbitrary prefix-closed property of a sequence of events. For contracts, these events are the attachment of contracts to values and the use of contracted values. In contrast, for inlined reference monitors built with aspects, this set of events is determined by the point-cuts selected by the policy. In either case, the programmer must correctly identify relevant events and specify the policy, but can assume the policy is enforced.

The formal definition and proof of complete monitoring for our contract system are given in Appendix B.

3.2 Representing Authority

In principle, a programmer can use context contracts to enforce arbitrary properties of execution contexts such as access control, but in practice this requires the careful design of an appropriate representation of the relevant information as an environment, i.e., a set of parameters. In particular, for access control this requires a representation of the authority of an execution context.

The authority of an execution context describes the rights it has to perform sensitive operations. In different access control mechanisms, the form and organization of these rights varies. For example, in a web application, a session executes on behalf of a particular user whose rights may change over time in accordance with the access control policies attached to the application's resources. In the Java stack inspection framework, rights are sets of "permissions" possessed by activation records that can be queried with the `checkPermission` operation.

A common way to describe the structure of authority in an access control system involves a mapping from *subjects* (users, processes, or security domains) to access rights for objects (resources that require the protection of the access control system) [22]. In practice, subjects and objects may comprise the same entities, so we refer to both as *security principals* (or, simply, *principals*).

To build a framework that supports many different access control mechanisms, we need a general way to express and reason about principals, the authority of principals, and how principals delegate and restrict their authority. For this purpose, we use an authorization logic [2] based on the Flow-Limited Authorization Model (FLAM) [5]. We briefly describe our logic, highlighting where it differs from FLAM. In Section 3.3, we employ this logic to represent authority as a set of parameters that are managed by specialized context contracts, dubbed authorization contracts.

Figure 6 presents the syntax of our logic. We assume an enumerable set of *primitive principals* $\mathcal{P}$. Primitive principals represent program entities that possess rights, such as users, modules, or activation records. We assume a most trusted principal $\top$ and a least trusted principal $\bot$. For principals p and q, the conjunctive principal $p \wedge q$ is a principal with the authority of *both* p and q. Similarly, the disjunctive principal $p \vee q$ has the authority of *either* p or q.

Primitives	$\mathcal{P}$	::=	$a \mid b \mid \ldots$
Projection dimensions	d	::=	$\alpha \mid \beta \mid \ldots$
Principals	p, q, r, s	::=	$\mathcal{P} \mid \top \mid \bot$
		$\mid$	$p \wedge p \mid p \vee p$
		$\mid$	$p \triangleright d \mid \overleftarrow{}_D p \mid \overrightarrow{}_D p$
Delegations	A	::=	$p \succeq p \mathbin{@} p$
Delegation sets	D	::=	$\{A, A, \ldots\}$

Figure 6. Syntax of principals, delegations, and worlds

If principal p trusts principal q, we write $q \succeq p$, and say that q *acts for* p. The acts-for relation is reflexive and transitive, and induces a lattice structure over the set of principals, with conjunction as join, disjunction as meet, and $\top$ and $\bot$ as the top and bottom elements of the lattice.

Principals may assert the existence of trust relationships. A *delegation* $p \succeq q \mathbin{@} r$ means that principal r asserts that p acts for q (or, equivalently, that q delegates its authority to p). Of course, whether a principal s believes the assertion depends on whether s trusts principal r. (We differ from FLAM in that we describe only the integrity of delegations, not their confidentiality.)

Judgment $D ; r \vdash p \succeq q$ denotes that given the set of delegations D, principal r believes that principal p acts for principal q. Intuitively, r believes that p acts for q if that trust relationship can be derived using only delegations asserted by principals that r trusts.

Figure 7 presents the inference rules for the judgment $D ; r \vdash p \succeq q$. Rules BOT, TOP, REFL, TRANS, CONJ-LEFT, CONJ-RIGHT, DISJ-LEFT, and DISJ-RIGHT are standard and provide the underlying lattice structure for the acts-for relation. Rule DEL captures the intuition that principal r trusts only delegations asserted by principals that it trusts, that is delegations $p \succeq q \mathbin{@} s$ where $D ; r \vdash s \succeq r$

We have three additional principal constructors. Principal $p \triangleright \alpha$ is the *projection* of the authority of principal p on dimension α[3]. We use projections to limit or attenuate the authority of a principal, and to identify access rights. For example, $p \triangleright files$ may refer to principal p's authority restricted to p's rights to access the file system. Similarly, principal $p \triangleright obj \triangleright invoke$ (equivalently $p \triangleright invoke \triangleright obj$) might refer to the right to invoke a particular object belonging to principal p. Principal p can grant this right to another principal q by asserting a delegation: $q \succeq p \triangleright obj \triangleright invoke \mathbin{@} p$.

We leave projection dimensions underspecified, and access control mechanisms can define their own dimensions. For any projection dimension α, principal p acts for principal $p \triangleright \alpha$, as captured in Rule PROJ. Typically the converse does not hold, and so $p \triangleright \alpha$ has strictly less authority than p.

Novel to this work, we introduce *closure principals* $\overleftarrow{}_D p$ and $\overrightarrow{}_D p$. Given a set of delegations D and principal p,

[3] FLAM considers *basis projections* and *ownership projections*. The projections we use here are more general and have less structure than either. In addition to preserving the acts-for lattice, the only structure we impose is that projections are commutative: $p \triangleright \alpha \triangleright \beta = p \triangleright \beta \triangleright \alpha$.

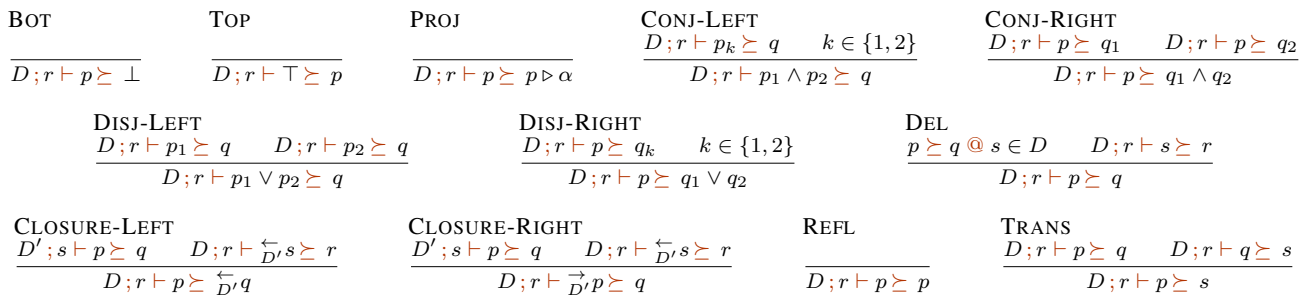

Figure 7. Inference rules for judgment $D ; l \vdash p \succeq q$

the *left-closure principal* $\overleftarrow{}_D p$ represents p with all of the trust relationships derivable from D where p delegates its authority to other principals. The *right-closure principal* $\overrightarrow{}_D p$ represents p with all of the trust relationships derivable from D where p acts for other principals. In our framework, delegations may change over time. Closure principals are useful because they allow us to capture trust relationships as they exist at particular moments in time. In particular, closure principals are a principled mechanism to describe how authority captured by a context contract should be combined with the current authority environment based on which parts of the closed over authority environment are trusted by principals in the current authority environment.

Rule CLOSURE-LEFT shows that $D ; r \vdash p \succeq \overleftarrow{}_{D'} q$ holds when there is some principal s such that at the time of closure creation (i.e., with delegation set D'), s believed that p acted for q (premise $D' ; s \vdash p \succeq q$), and moreover, right now (i.e., with delegation set D) principal r trusts principal $\overleftarrow{}_{D'} s$ (premise $D ; r \vdash \overleftarrow{}_{D'} s \succeq r$). Typically, s and r are the same principal, meaning that r-at-time-D trusts the decisions made by r-at-time-D'. Rule CLOSURE-RIGHT is similar and $D ; r \vdash \overrightarrow{}_{D'} p \succeq q$ holds when there is some principal s such that at the time the closure was taken s believed that p acted for q (premise $D' ; s \vdash p \succeq q$), and principal r trusts s-at-time-D' (premise $D ; r \vdash \overleftarrow{}_{D'} s \succeq r$).

To query whether a particular set of delegations satisfies an acts-for relation, we use a proof search algorithm adapted from FLAM [5]. We give examples of using delegations to implement different authorization mechanisms in Section 4.

Based on this logic, we represent an authority environment as:

1. a *principal*, who is responsible for the current execution context, and
2. a *delegation set*, which records the current trust relationships between principals.

The latter has two sub-parts: a global, mutable delegation set, and a set of delegations that are in place only for a currently executing context.

3.3 Authorization Contracts

Using authority environments, we can now introduce authorization contracts. Authorization contracts specialize context contracts in two ways. First, they prevent interference from untrustworthy code by using parameters that the rest of the program does not have access to. Second, they use a high-level representation of authority environments rather than directly manipulating parameters. Authorization contracts provide a structured way to describe how the underlying context contracts should manipulate authority environments.

Authorization contracts are defined as monitor actions using the `define-monitor` form (§2). In this section, we model a pared-down version of `define-monitor` as an extension to the language model from Section 3.1. Figure 8 displays the syntax of the extension. The extension introduces new types, constructors, and operations for principals, delegations, and delegations sets, including an expression that evaluates an acts-for judgment: $e ; e \vdash e \succeq e$. The `define-monitor` form corresponds to the monitor $(a \ldots)$ form that can appear in the where clause of a module definition in the extended model. Each a in monitor $(a \ldots)$ is an action specification. An action specification action $x\,(y : \tau, \ldots)$ (ce,ae) has a name (x), a set of arguments $(y : \tau, \ldots)$, and two terms (ce and ae) that define the action's `#:on-create` and `#:on-apply` hooks.

The term for the `#:on-create` hook has the form

check: *cee* add: *cee* remove: *cee* set!-principal: *cee*
closure-principal: *cee* closure-delegations: *cee*

and specifies what the authorization contract should do when the contract is applied to a value. In particular it describes how to modify each part of the authority environment. Its field check accepts an acts-for judgment. If this judgement does not hold, a contract error is raised blaming the client of the contract. Field add accepts a set of delegations to add to the global delegation set. Field remove accepts a set of delegations to remove from the global delegation set, if present. Field set!-principal changes the current principal to the given principal. Fields closure-principal and closure-delegations accept a principal and a set of delegations, respectively, and record the principal and delegations for use upon a call to the contracted function. Terms in each of these six fields can access the pieces of the current authority environment using current-principal and current-delegations.

$$
\begin{array}{rcl}
m & ::= & \text{module } \ell \text{ exports } x \text{ with } x,\ldots \text{ where } x = e,\ldots,\text{monitor } (a,\ldots),x = e,\ldots \\
v & ::= & \ldots \mid \top \mid \bot \mid \mathcal{P} \mid \mathcal{D} \mid v \succeq v \mathbin{@} v \mid \{v,\ldots\} \\
e & ::= & \ldots \mid \text{new-principal} \mid \text{new-dimension} \mid e \triangleright e \mid e\,;e \vdash e \succeq e \mid e \succeq e \mathbin{@} e \mid \text{let } x \succeq x \mathbin{@} x = e \text{ in } e \\
 & \mid & \{\} \mid \{e\} \mid e \cup e \mid e \setminus e \mid (\text{fold } e\ e\ e) \\
\beta & ::= & \ldots \mid \text{Prin} \mid \text{Dim} \mid \text{Del} \mid \text{DelSet} \\
a & ::= & \text{action } x\ (y : \tau,\ldots)\ (ce,ae) \\
ce & ::= & \text{check: } cee \text{ add: } cee \text{ remove: } cee \text{ set!-principal: } cee \text{ closure-principal: } cee \text{ closure-delegations: } cee \\
ae & ::= & \text{check: } aee \text{ add: } aee \text{ remove: } aee \text{ scope: } aee \text{ set-principal?: } aee \text{ principal: } aee \text{ set!-principal: } aee \\
cee & ::= & e \mid \text{let } x = cee \text{ in } cee \mid \text{current-principal} \mid \text{current-delegations} \\
aee & ::= & e \mid \text{let } x = aee \text{ in } aee \mid \text{current-principal} \mid \text{current-delegations} \mid \text{closure-principal} \mid \text{closure-delegations}
\end{array}
$$

Figure 8. Syntax extensions for authorization contracts.

The second term corresponds to the `#:on-apply` hook, which specifies what the authorization contract should do upon a call of the contracted function. It has the form

check: cee add: cee remove: cee scope: cee
set-principal?: cee principal: cee set!-principal: cee.

Similar to a ce term, it allows the configuration of the contract's behavior. As before, check accepts an acts-for judgment and raises a contract error if it does not hold. Likewise, fields add and remove mutate the global delegation set, and field set!-principal changes the current principal. The scope field accepts a set of delegations, but this set is installed only for the dynamic extent of the current function call, rather than added to the global delegation set. The set-principal? field requires a boolean value. If that value is #t, the current authorization environment is extended with a principal for the dynamic extent of the function call. This (1) allows changing the principal visible within the extent of the function call and (2) prevents contracts that change the principal during the extent of the function call from modifying the principal of the enclosing context. In addition to accessing the current principal and delegations from the authority environment, the seven fields of an ae term can access the principal and delegations closed over by the contract with terms closure-principal and closure-delegations.

To give a detailed semantics for monitor terms, we use a compilation function that replaces monitor expressions with terms that explicitly construct context contracts. The compilation uses five parameters: one each for the current principal, global delegation set, and scoped delegation set, plus a pair to record the closed-over principal and delegations. Each monitor term generates a fresh set of parameters, preventing separately defined monitors from interfering with each other. Each action term compiles to a single context contract that closes over the fresh parameters. The full compilation function is listed in Appendix A, along with typing judgments for authorization contracts and the semantics of expressions that operate on principals, dimensions, delegations, and delegation sets.

The hooks for defining actions are sufficiently flexible to implement a variety of access control mechanisms (§4). Here, we briefly describe some of the ways programmers can configure authorization contracts.

Mutable Authority Many access control mechanisms have a global policy that changes over time. For example, in discretionary access control, users can grant or revoke access to their resources. We can implement this with a contract that adds or removes (global) delegations.

Dynamically-scoped Authority An authority closure can inherit the authority environment from its calling context by ignoring the authority environment it closes over.

"Lexically"-scoped Authority An authority closure can isolate itself from the authority of its calling context by replacing the authority environment at a call site with the authority that it closes over.

Moreover, different access control mechanisms may require authorization contracts that blend these different strategies. For example, implementing `setuid`-like authority closures requires capturing the principal but not the delegations the closures close over. Otherwise, updates to the global, mutable discretionary access control policy would be forgotten when a `setuid` function runs.

4. Putting Authorization Contracts to Work

As evidence of the usefulness and expressiveness of the framework, we implemented a variety of existing access control mechanisms including discretionary access control, stack inspection [42], history-based access control [1], and object capabilities [27]. Before delving into the mechanisms, we further explain **`define-monitor`**, the main linguistic tool that our framework provides.

4.1 The `define-monitor` Form

Figure 9 shows the complete syntax of **`define-monitor`**. It has two sections in addition to the **`action`** section we have seen before: **`extra`** and **`syntax`**. The first defines extra functions and contracts that the programmer wants to include in the interface of a monitor. These are usually contracts that combine two or more actions together or contracts that fix the arguments of an action. The **`syntax`** section defines macros that

```
(define-monitor monitor-name
  (monitor-interface
    action-name ... extra-name ...)
  (monitor-syntax-interface
    syntax-name ...)
  (action ; definitions of monitor actions
    [action-name (action-var ...)
      #:on-create on-create-hook
      #:on-apply  on-apply-hook]
      ... )
  (extra  ; additional monitor abstractions
    (define extra-name extra-body)
    ... )
  (syntax ; syntactic monitor abstractions
    (define-syntax syntax-name syntax-body)
    ... ))
```

Figure 9. The `define-monitor` form

serve as syntactic abstractions over the monitor's interface, for example, to automate the placement of authority contracts when defining a function. We give examples of definitions in the `extra` and `syntax` sections later. The `monitor-interface` and `monitor-syntax-interface` clauses specify which elements are available to users of the monitor. After defining a monitor `monitor-name`, a client can instantiate it with `(run monitor-name)`. This creates a fresh monitor, i.e., one with a fresh authority environment and contracts.

The most complicated part of defining a monitor is writing the two hooks for each monitor action. To facilitate this, we provide two functions, `do-create` and `do-apply`, that simplify this process. Each function has optional keyword arguments corresponding to the fields of an action form in Section 3.3. The functions provide default values for any argument not specified. Thus, the programmer need only specify the results of the hooks they care about. For instance, the default value for the argument with keyword `#:check` seen in Figure 1 is an acts-for relation that is always true.

4.2 A Discretionary Access Control Monitor

In a discretionary access control system, individual users can choose what policy to enforce on objects they control. We extended the basic discretionary access control monitor from Figure 1 with additional contracts for marking functions as objects owned by individual users, and for granting or revoking the right to invoke these functions. The resulting monitor is shown in Figure 11 in Appendix C. It defines four actions: `make-user/c`, `make-object/c`, `grant/c`, and `revoke/c`. Action `make-user/c` creates an authority closure that captures a fresh principal representing a new user. Invoking the wrapped function requires that the current principal at the call site is authorized for the `invoke` projection of the new user's authority, and switches to the new user's principal without modifying any delegations. Action `make-object/c` creates a new projection of the current user's authority, and requires that any context invoking the wrapped function has authority to use this projection of the user's authority. Action `grant/c` takes two functions as arguments, one wrapped with a `make-user/c` authorization contract and one wrapped with a `make-object/c` contract. When a function wrapped with this contract is applied, the contract adds a new delegation that says the principal of the `make-user/c` closure acts for the principal of the `make-object/c` closure. This delegation is ignored if the current context has insufficient authority to delegate the object. Similarly, `revoke/c` takes a user and an object as arguments, and removes any delegation from the global delegation environment that grants the user right to invoke the object where the current principal acts for the principal that asserted the delegation.

4.3 A Stack Inspection Monitor

In stack inspection [42], code obtains permissions based on static properties such as the package it belongs to. At run time, code can choose to enable its static permissions making them eligible for satisfying an access control check. Before a sensitive operation, stack inspection checks for the presence of a particular permission by walking the run-time call stack until a frame from code that has enabled the permission is found. To prevent *luring attacks* [42], stack inspection additionally requires that all execution contexts between the enabled permission and the authorization check have the required static permission. Despite this protection, untrusted code may be able to influence the program even if its frames are no longer on the stack. As a result, modern adaptations of stack inspection provide additional support for capturing the permissions of the stack at some point in an execution and reinstating them for a later check.

Implementations of stack inspection provide the following primitives: `checkPermission`, which checks that a frame on the stack has the required permission enabled and that all intervening frames have the required static permission; `doPrivileged`, which enables the static permissions of the current code for its dynamic extent, possibly using captured permissions instead of the current permissions; and `getContext`, which captures the permissions of the stack at some point in execution. In addition, the implementation must provide a mechanism to associate static permissions with code.

To realize stack inspection using authorization contracts, a monitor must provide (1) actions that implement these primitives and (2) a way to grant static permissions to code. In our monitor, the actions for (1) are `check-permission/c`, `do-privileged/c`, and `context/c`. To track which permissions are held by code on the stack, we use the authority environment to grant permissions to individual frames, each represented by a distinct principal. Each stack frame has three projections that are used to manage its authority. The `static` projection indicates the permissions granted to the code statically. The `enable` projection has the authority of the permissions enabled for this frame. The `active` projection represents permissions that would satisfy a privilege check. We say a

principal has a particular permission if it acts for the corresponding projection of the $\top$ principal.

We use one additional monitor action, `privileged/c`, to indicate the static permissions a piece of code possesses and to enforce that a stack frame's `active` projection acts for exactly those permissions for which `checkPermission` should succeed. Action `privileged/c` takes a list of permissions (each of which is a projection of the $\top$ principal). On an `#:on-apply` event, it creates a new principal `callee` to represent the new stack frame and adds delegations initializing these projections for the dynamic extent of the function:

```
(≽ @ (▷ callee static) permissions ⊤)
(≽ @ (▷ callee enable)
     (▷ current-principal active)
      current-principal)
(≽ @ (▷ callee active)
     (∨ (▷ callee enable) (▷ callee static))
     callee).
```

These delegations give `callee` the specified static permissions (by asserting that the callee's `static` dimension acts for the conjunctive principle `permissions`), assert that the new frame inherits the `active` permissions from the previous frame, and require that the `callee` has both static and enabled permissions to make them active.

Tracking the authority of each frame in this way makes walking the stack unnecessary. Action `check-permission/c` only checks that the `active` projection of the current principal acts for all of the requested permissions.

Action `do-privileged/c` enables the current frame's static permissions by adding a delegation from the frame's `static` projection to its `enable` projection for the dynamic extent of the wrapped function.

Action `context/c` is used to capture the permissions of the current stack for future permission checks. It captures the current authorization environment when it is attached to a function. When it is invoked, it installs the same set of delegations as `privileged/c`, except that the first delegation that grants static permissions gets replaced with a delegation that derives permissions from the active permissions of the captured frame at the time they were captured:

```
(≽ @ (▷ callee static)
   (▷ (→ closure-principal
         closure-delegations)
   active) ⊤).
```

The right-closure principal on the right hand side of this delegation acts for all of the principals that `closure-principal` acted for when the closure was created.

The monitor must also provide (2) a way to grant static permissions to code. Because Racket does not have class-loading facilities that would allow permissions to be granted to code at load-time, we use macros to attach authorization contracts to code that should have static permissions. In particular, the monitor provides a new definition form `define/rights` in its **`syntax`** section. This form works like the **`define`** form,

```
(define/rights (read-file file) (filesys)
  (check-permission/c filesys)
  ...)

(define/rights (read-privileged file)
    (filesys)
  do-privileged/c
  (if (safe? file) (read-file file) #f))

(define/rights (malicious) (net)
  any/c
  (read-file "/etc/passwd"))

> (malicious)
; read-file: contract violation;
;  (▷ frame10247 active) ⋡ (▷ ⊤ filesys)
;    @ (▷ ⊤ filesys)
;   contract from: (definition read-file)
;   blaming: top-level
```

Figure 10. Using the stack inspection monitor

but takes two additional arguments: a set of permissions and a contract to apply to the definition. It defines a function wrapped with the given contract and a `privileged/c` contract. In addition, the macro `define/rights` coerces any function arguments or free-variables appearing in the body of the function to authority closures by applying an additional contract `unprivileged/c`, which is defined in the **`extra`** section of the monitor. Action `unprivileged/c` switches to the $\bot$ principal for the dynamic extent of the closure it wraps, preventing any `check-permission/c` actions from succeeding. Thus, these contracts prevent functions that were not defined with `define/rights` from using code that requires permissions.

The complete implementation of the monitor is given in Figure 12 in Appendix C, and Figure 10 shows an example program using the stack inspection monitor. There are three functions defined using `define/rights`. Two of these functions are trusted to access the filesystem: `read-file` and `read-privileged`. However, `read-file` should not be used directly, so it checks that the `filesys` permission has been enabled by one of its callers. Function `read-privileged` enables the `filesys` permission, but only calls `read-file` if the `file` is safe to read. Function `malicious` does not have the `filesys` permission but attempts to read `"/etc/passwd"` anyway, so invoking this function results in a contract violation. The contract violation says that the stack frame corresponding to the call to `read-file` does not have the necessary permission `filesys`.

4.4 A History-Based Access Control Monitor

Abadi and Fournet [1] observe that stack inspection fails to protect against attacks where the influence of untrusted code is no longer apparent from the call stack. As a remedy, they propose history-based access control (HBAC). In HBAC, the rights of an execution context depend not just on the

rights of code currently on the execution stack, but also on the rights of all code that has previously been executed. HBAC has two primitives: `grant` and `accept`. `grant` has the same behavior as the `do-privileged` operator we implement for stack inspection. `accept` allows a component to take responsibility for code in its dynamic extent. After `accept` returns, it restores any privileges that were present before it was invoked.

Our implementation of a monitor for HBAC is very similar to the monitor for stack inspection, and is shown in Figure 13 in Appendix C. The primary difference between the two monitors is the definition of action `privileged/c`. In the HBAC monitor, in addition to initializing the three projections for the new frame, its `#:on-apply` event walks the call stack by reading the current delegations. For every frame on the call stack, it adds a disjunct with the callee's static permissions to the delegation that granted permissions from that frame's caller. For example, the delegation:

```
(≽ @ (▷ parent enable) (▷ grandparent
    active)
     grandparent)
```

is replaced with the delegation:

```
(≽ @ (▷ parent enable)
    (∨ (▷ grandparent active) (▷ callee
        static))
    grandparent)
```

This means that future attempts to use the `parent` frame's permissions will be restricted to whatever rights the `callee` had, unless the `parent` frame specifically vouches for an action by enabling its own permissions.

We define `accept/c` in the **`extra`** section. `accept/c` uses the `accept-context/c` action to create an authority closure around its continuation. When the function `accept/c` wraps returns, `accept/c` invokes this continuation, restoring the authority environment before the wrapped function was called.

4.5 An Object Capability Monitor

Authority closures are closely related to the idea of capabilities. A capability both designates a resource and confers the authority necessary to use it [8]. An authority closure designates resources (those accessed by the wrapped function) and captures the authority that should be used to access those resources. Any code that can invoke an authority closure can exercise the authority of the closure, though that authority is attenuated by the functionality of the closure itself. For example, in our web application example, `login` is a capability that allows any code that invokes it to use the "root" user's authority; however, the implementation of the `login` function ensures that this authority can be used only to switch to a different user after supplying the correct password.

In a memory-safe programming language, any reference to a value can be considered a capability that grants restricted access to some set of resources that can be accessed through it (by invoking or otherwise inspecting it). Object-capability languages like E [27], Joe-E [23], or Caja [28], embrace this fact as the basis of their security architecture. In an object-capability language, all sensitive resources are represented as objects. Access to these objects is controlled by structuring the language to limit the ways that objects acquire references to other objects [27]:

1. by initial conditions: two objects may reference each other before a computation begins,
2. by parenthood: the creator of an object is initially the only object with a reference to it,
3. by endowment: an object can close over references to objects available in its parent's environment, and
4. by introduction: an object can receive a references to other objects passed as arguments to its methods or returned from methods it invokes.

These restrictions are sometimes referred to as "capability safety" and are designed to support modular reasoning about security. For example, the restrictions above are sufficient to enforce the property that "only connectivity begets connectivity;" that is, two components may communicate or share references only via a capability shared by both components [27].

We built a monitor to enforce capability safety. This monitor, shown in Figure 14 in Appendix C, defines two actions, `capability/c` and `unprivileged-capability/c`, which is the same as the `capability/c`, but does not grant any initial authority. These actions enforce capability safety by creating a new principal for each capability and using delegations to specify when one capability has the authority to invoke another. Their `#:on-create` hooks handle parenthood and endowment and their `#:on-apply` hooks handle introduction. Parenthood amounts to a delegation:

```
(≽ @ (▷ parent caps) (▷ child invoke) child)
```

Similarly, endowment closes over the current authority of the parent and grants it to the child:

```
(≽ @ (▷ child caps)
     (→ (▷ parent caps)
        current-delegations)
     parent)
```

We must also assert that the parent authorizes the use of its closed-over delegations for the rest of the execution:

```
(≽ @ (← parent current-delegations) parent
     parent)
```

Introduction is implemented by adding additional delegations granting callees authority over arguments they are passed, and vice versa for return values.

5. Case Studies

To evaluate the use of our framework in practical applications, we developed three case studies. The first adds simple

authorization contracts to the implementation of a card game to ensure that player's moves affect only the parts of the game state they control. The second secures a plugin interface of the DrRacket development environment and demonstrates how the flexibility of the framework can support complex security mechanisms. The third, which mirrors the example from Section 2.2, replaces authorization checks in a web application with authorization contracts.

We evaluated the performance of our framework on each case study. The experiments were conducted on a MacBook Pro with a 2.6 GHz Intel Core i5 and 16GB of RAM running Mac OS X 10.11 and Racket 6.4.0.9. In the first two case studies, authorization contracts have significant impact on the performance of the benchmarks. However, both case studies are worst case scenarios: they have no existing code implementing access control (and so we are strictly adding functionality), and after adding contracts, they invoke many access control checks (tens of thousands in the case of the card game) while performing cheap operations. Moreover, in the DrRacket case study, the absolute overhead for each benchmark due to authorization contracts is less than 45ms, but the relative overhead is high since the baseline running time is less than 15ms. The third case study replaces existing access control checks with authorization contracts, with negligible impact on performance. Our implementation is a prototype, and we anticipate that optimizations in the implementation of our contracts can further reduce their overhead.

Preventing Cheating in a Card Game We have used authorization contracts to enforce a security policy for a functional implementation of the card game Dominion[4]. The exact rules of Dominion do not matter for our purpose, except that each player collects cards in a local deck and attempts to outscore the rest of the players by playing cards from their deck. During each turn, players can play cards from their deck to either purchase additional cards or attack other players, forcing them to discard some of their cards.

In this implementation, each player is a program that runs in its own process and responds automatically to messages from a central broker. The broker maintains the shared inventory of cards and a mirror of each player's local deck. Players perform moves by sending messages to the broker describing the move.

To perform a move, the player sends a message to the broker identifying a card to play. In response, the broker updates its copy of the game state to reflect the move and, if the move involves an attack on another player, informs the other player of the attack. The other player then has an opportunity to defend by choosing which card to discard and the broker again updates the game state.

The broker represents the local deck of each player as an immutable record `player` and the state of the game as an immutable structure `game` that holds a list of `player` records. The first element in this list corresponds to the player who makes the next move. The broker is implemented as a core `drive` function that delegates to two functions: `move` and `defend`. Both functions perform functional updates to the relevant structures.

We enforce the policy that the broker only updates the current player's deck or a defending player's deck. The monitor that enforces this policy specifies three authorization contracts: `deprivilege/c`, which sets the principal for the dynamic extent of a function to $\bot$; `(switch-player/c name)`, which sets the principal for the dynamic extent of a function to the player with name `name`; and `(check-player/c name)`, which checks before calling a function if the current principal is the player with name `name`.

To enforce the policy, we attach contract `deprivilege/c` to the function `drive` so that only authorized code can modify the game state during the game. The contract for the `game` structure, `game/c`, gives the accessor functions of each field of the player records in the `game` the contract`(check-player/c name)`, where `name` is the name of the corresponding player. The contract for the `move` function is

```
(->a
  ([game game/c] [turn any/c] [play any/c])
  #:auth (game)
    (switch-player/c
      (player-name
        (first (game-players game))))
  (values [game-result-game game/c]
          [turn-result any/c]))
```

and it authorizes the `move` function to act on behalf of the current player, i.e., `(first (game-players game))`. The contract for `defend` is

```
(->a ([player player/c] [defense any/c])
     #:auth (player)
       (switch-player/c
         (player-name player))
     [result player/c])
```

which similarly allows the function to update the state of the player who was attacked.

We created 10 benchmarks for the Dominion case study that each consists of a simulated game with 2-7 players. Adding authorization contracts increases running time by 1.3–1.7× at both the median and 99$^{\text{th}}$ percentile.

Securing a Plugin Interface We wrote a monitor to protect DrRacket from malicious or buggy third-party key bindings. First, we explain aspects of DrRacket's design related to key bindings. Keystrokes sent to DrRacket are dispatched as method calls to a `text%` object which encapsulates the state of the editor. This object has methods that access and modify parts of DrRacket. For instance, the `get-text` method returns the content of the editor, while the `set-padding` method changes the inset padding used to display the editor's content. Each `text%` object has a `keymap%` object that stores

[4] The implementation is part of the teaching material of a long running undergraduate Functional Programming course.

registered key bindings and maps sequences of keystrokes to the action they trigger. A keybinding action is an arbitrary Racket function of two arguments: the current `text%` object and an `event%` object, which describes the event that triggered the action. On startup, DrRacket populates its `text%` object's key map with built-in key bindings. In addition, DrRacket registers user-defined key bindings from configuration files. Keybinding actions can inspect and modify almost any aspect of DrRacket through the `text%` object. This gives users a powerful interface for customizing DrRacket but makes key bindings a source of vulnerabilities. For instance, a key binding could accidentally erase the user's code or snoop on the editing session.

Our monitor restricts which `text%` object methods a keybinding action can invoke. We group methods of `text%` that can access or modify similar parts of DrRacket. For instance, methods that write to the clipboard (e.g. `cut` and `copy`) belong to the same group while methods that change how DrRacket displays content (e.g. `set-max-width` and `set-line-spacing`) belong to a second group. Each group has a corresponding privilege that is required to invoke the group's methods. For example, the privileges `ReadClipboard` and `ChangeEditorView` grant access to the methods mentioned above. Methods can belong to multiple groups. Access control checks around each method verify that the authority of a calling execution context has the necessary privileges.

In addition to methods that require specific privileges to invoke, `text%` has sensitive methods that should be invoked only by another method of the `text%` object. For example, the `on-delete` method should never be invoked directly as its correctness depends on DrRacket's state. Instead, key bindings should invoke the `delete` method that subsequently calls `on-delete`. To support this use case, we require an additional privilege to call `on-delete` that is granted during the dynamic extent of `delete`.

The stack-inspection-like access control mechanism we have described so far is not sufficient. Some methods of `text%` install callbacks that are triggered by subsequent events. For example, `add-undo` registers a callback that runs when the user wishes to undo the action of a key binding. This callback should not run with the authority of its calling context, but instead should use the privileges of the action that created it. To achieve this, we create authority closures around any callbacks registered by an action.

Our monitor represents each privilege as a unique principal and represents sets of principals as conjunctions and disjunctions of principals. It defines three actions: `check/c`, `enable/c`, and `closure/c`. Upon an `#:on-apply` event, the first action consumes principal `perms` and checks if the current principal has permissions that imply `perms`. Then the action switches the current principal to a principal that only has permissions `perms`. When a function wrapped with `enable/c` is applied, it switches the current principal to a principal that has the same permissions as the current principal augmented with `perms`. The `#:on-create` event of the third action creates an authority closure. When the authority closure is applied, it installs the closed-over principal.

We use the actions of the monitor to define an authorization contract for the keybinding interface:

```
(->a ([t text/c] [e (is-a?/c event%)])
     #:auth () (check/c perms) any)
```

where `perms` is a principal which encodes the privileges we grant to the key binding and `text/c` is the object contract we define for the editor's `text%` object. `text/c` applies a contract to each method of `text%` specifying whether the method enables some permission, requires some permissions, or creates an authority closure around one of its arguments. For example, `text/c` gives the method `blink-caret` the contract `(check/c ChangeEditorView)`. In essence, `text/c` defines a security policy for the editor.

To assess the monitor's performance, we ran a series of 30 benchmarks, adapted from DrRacket's test suite, that simulate a sequence of keystrokes that trigger built-in key bindings. We ran these benchmarks with the monitor off and on. When the monitor is on, the prototype grants the minimum set of privileges necessary for each key binding. For each benchmark, we measured the time required to retrieve and execute each key binding. Our measurements show that the authority monitor increases median response time by 3–7$\times$ and increases response time at the 99$^{\text{th}}$ percentile by 3–5$\times$. However, for an IDE, a response time fast enough for interactive use is more important. Our prototype achieves this goal with a maximum response time of 53ms.

Authentication in a Web Application The Racket package system allows users to discover and install packages from a public index service. Individual users can add new packages or update old ones by logging into the index service web application, which is implemented using the Racket web-server. Requests to add or modify packages are issued to the application as asynchronous http requests. The baseline implementation of the application uses macros to authenticate the user and perform any required access control checks before processing the request. For example, the `jsonp/pkg/modify` endpoint authenticates the current user and checks that they are an author of the package they are attempting to modify. This approach to access control is brittle, since it requires that the checks included for each endpoint accurately capture the privileges required when processing the response.

Using authorization contracts, we are able to separate the tasks of authentication and authorization in the index service web application. Rather than performing a different set of access control checks for each endpoint, all endpoints now simply invoke an `authenticate` function that checks whether the current session is valid and which user is logged in, then invokes a procedure to process the request, like the `login` function from Section 2.2. The access control policies for sensitive operations like updating a package are enforced by

adding authorization contracts that implement the necessary checks to the web application's data model. There are two types of checks: `(is-author/c pkg)` which checks that the logged in user is an author of package `pkg`, and `is-curator/c`, which checks whether the logged in user has "curator" status, which allows them to tag packages with information about their quality.

To evaluate the new implementation's performance, we measured the latency of 1,000 repeated requests to modify a package record. Replacing inline checks with authorization contracts has minimal impact on performance. Median latency was 283ms for the baseline implementation versus 281ms with authorization contracts. At the 99^{th} percentile, using authorization contracts latency was 338ms versus 330ms with the baseline implementation.

6. Related Work

The connection between scoping and access control has been implicit in prior work on security in programming languages but has never been a central concept for extensible access control. Morris's seminal paper "Protection in Programming Languages" [31] describes how lexical scope can be used to create security abstractions within a program. More recently, the object-capability paradigm has embraced lexical scope as an organizing security principle [27]. Wallach and Felten [41] note that "in some ways, [stack inspection] resembles dynamic variables (where free variables are resolved from the caller's environment rather than from the environment in which the function is defined)." Phung et al. [32] use dynamic and lexical scoping to associate principals with executing code in order to correctly enforce security policies on programs that mix JavaScript and ActionScript code.

Inlined Reference Monitors An alternative approach to language-level access control is inlined reference monitoring. Reference monitors observe the actions taken by a system and intercede to prevent violations of a security policy [3]. They can enforce a large class of policies [34]. Inlined reference monitoring (IRM) weaves the implementation of a reference monitor into the program being monitored [13]. Many implementations of inlined reference monitoring rely on aspects to identify security relevant actions during program execution [6, 7, 14, 15, 21]. Policies supported by these tools typically focus on access patterns for sensitive resources. While policies supported by our framework can be encoded this way, as in Erlingsson and Schneider's IRM implementation of Java stack inspection [14], policies where the authority of code depends on application state require duplicating code. A further disadvantage of IRMs is that they require a global transformation of the program to inline the security monitor. Because authorization contracts are applied at component boundaries, our framework requires only local modifications.

Authorization Logics Authorization logics give a formal language to express access control policies [1]. Authorization logics have been used to understand existing access control mechanisms, including Java stack inspection [41]. Aura [20] and Fine [38] implement access control using proof-carrying authentication, where proofs of formulas in an authorization logic are used as capabilities [4]. Our access control logic is inspired by the Flow-Limited Authorization Model [5], which uses projections to describe attenuated authority without requiring additional constructs such as roles or groups.

Contracts for Security Previous work has used contracts to enforce limited access control policies. Moore et al. [29] use contracts to constrain the use of capabilities in a secure shell scripting language. Dimoulas et al. [10] use contracts to control the flow of capabilities between components in object-capability languages. Heidegger et al. [19] use contracts to specify which fields of an object may be accessed by a component. However, each of these systems is specialized to enforce a specific type of access control policy. Disney et al. [11] introduce temporal higher-order contracts that enforce that sequences of function calls and returns match a specification. Schollier et al.'s computational contracts [35] can enforce a wide range of trace properties on programs. Unlike authorization contracts and temporal higher-order contracts, computational contracts use aspects to interpose on program events. Both of these systems support arbitrarily powerful monitors, but like inlined reference monitoring, provide limited support for writing complex access control policies like stack inspection or discretionary access control.

Scoped Aspects for Security Dutchyn et al. [12] are the first to identify a connection between modular access control and the scoping of aspects that implement authorization checks. They introduce constructs for lexical and dynamic aspect scoping and use them to enforce a simple security policy similar to the one we enforce in §2.3 with authorization contracts. With additional aspect scoping mechanisms, Toledo et al. [40] implement a complete Java stack inspection mechanism in a modular fashion. The exact relation between the expressiveness of scoped aspects and that of authorization contracts is an open question. However, as our case studies and example access control mechanisms demonstrate, authorization contracts are powerful enough to enforce a variety of access control policy classes with a single domain-specific abstraction: authority environments. Doing the same with aspects, if possible, requires brittle and complex encodings in terms of general-purpose advice and aspect scoping.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant Numbers CCF-1438271, CNS-1524052, and CCF-1526324. This research is also supported by the Air Force Research Laboratory and a Google Faculty Research Award.

References

[1] M. Abadi and C. Fournet. Access control based on execution history. In *Proceedings of the 10th Annual Network and Distributed System Security Symposium (NDSS)*, pages 107–121, February 2003.

[2] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 15(4): 706–734, September 1993.

[3] J. P. Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51, U.S. Air Force Electronic Systems Division, Deputy for Command and Management Systems, HQ Electronic Systems Division, 1972.

[4] A. W. Appel and E. W. Felten. Proof-carrying authentication. In *Proceedings of the 6th ACM Conference on Computer and Communications Security (CCS)*, pages 52–62, Nov. 1999.

[5] O. Arden, J. Liu, and A. C. Myers. Flow-limited authorization. In *Proceedings of the 28th IEEE Computer Security Foundations Symposium (CSF)*, pages 569–583, July 2015.

[6] L. Bauer, J. Ligatti, and D. Walker. Composing security policies with Polymer. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 305–314, June 2005.

[7] F. Chen and G. Roşu. Java-MOP: A monitoring oriented programming environment for Java. In *Proceedings of the Eleventh International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 546–550, April 2005.

[8] J. B. Dennis and E. C. Van Horn. Programming semantics for multiprogrammed computations. *Communications of the ACM*, 9(3):143–155, March 1966.

[9] C. Dimoulas, S. Tobin-Hochstadt, and M. Felleisen. Complete monitors for behavioral contracts. In *Proceedings of the 21st European Symposium on Programming (ESOP)*, pages 211–230, March 2012.

[10] C. Dimoulas, S. Moore, A. Askarov, and S. Chong. Declarative policies for capability control. In *Proceedings of the 27th IEEE Computer Security Foundations Symposium (CSF)*, pages 3–17, July 2014.

[11] T. Disney, C. Flanagan, and J. McCarthy. Temporal higher-order contracts. In *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 176–188, September, 2011.

[12] C. Dutchyn, D. B. Tucker, and S. Krishnamurthi. Semantics and scoping of aspects in higher-order languages. *Science of Computer Programming*, 63(3):207–239, December 2006.

[13] U. Erlingsson and F. B. Schneider. SASI enforcement of security policies: A retrospective. In *Proceedings of the 1999 Workshop on New Security Paradigms*, pages 87–95, 1999.

[14] U. Erlingsson and F. B. Schneider. IRM enforcement of Java stack inspection. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy (S&P)*, pages 246–255, May 2000.

[15] D. Evans and A. Twyman. Flexible policy-directed code safety. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy (S&P)*, pages 32–45, May 1999.

[16] R. B. Findler and M. Felleisen. Contracts for higher-order functions. In *Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 48–59, October 2002.

[17] M. Flatt and PLT. Reference: Racket. Technical Report PLT-TR-2010-1, PLT Design Inc., 2010. `http://racket-lang.org/tr1/`.

[18] M. Gasbichler and M. Sperber. Processes vs. user-level threads in Scsh. In *Proceedings of the 3rd ACM SIGPLAN Workshop on Scheme and Functional Programming*, 2002.

[19] P. Heidegger, A. Bieniusa, and P. Thiemann. Access permission contracts for scripting languages. In *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 111–122, Jan. 2012.

[20] L. Jia, J. A. Vaughan, K. Mazurak, J. Zhao, L. Zarko, J. Schorr, and S. Zdancewic. AURA: A programming language for authorization and audit. In *Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 27–38, September 2008.

[21] M. Jones and K. W. Hamlen. Enforcing IRM security policies: Two case studies. In *Proceedings of the 7th IEEE Intelligence and Security Informatics Conference (ISI)*, pages 214–216, June 2009.

[22] B. W. Lampson. Protection. *ACM SIGOPS Operating Systems Review*, 8(1):18–24, January 1974.

[23] A. Mettler, D. Wagner, and T. Close. Joe-e: A security-oriented subset of java. In *Proceedings of the 17th Annual Network and Distributed System Security Symposium (NDSS)*, March 2010.

[24] B. Meyer. *Object-Oriented Software Construction*. Prentice Hall, 1988.

[25] B. Meyer. Design by contract. In *Advances in Object-Oriented Software Engineering*, pages 1–50. Prentice Hall, 1991.

[26] B. Meyer. Applying "Design by Contract". *Computer*, 25(10): 40–51, October 1992.

[27] M. Miller. *Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control*. PhD thesis, Johns Hopkins University, May 2006.

[28] M. S. Miller, M. Samuel, B. Laurie, I. Awad, and M. Stay. Caja: Safe active content in sanitized JavaScript, 2008. Google white paper.

[29] S. Moore, C. Dimoulas, D. King, and S. Chong. Shill: A secure shell scripting language. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 183–199. USENIX, October 2014.

[30] S. Moore, C. Dimoulas, R. B. Findler, M. Flatt, and S. Chong. Extensible access control with authorization contracts. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, September, November 2016.

[31] J. H. Morris, Jr. Protection in programming languages. *Communications of the ACM*, 16(1):15–21, January 1973.

[32] P. H. Phung, M. Monshizadeh, M. Sridhar, K. W. Hamlen, and V. N. Venkatakrishnan. Between worlds: Securing mixed javascript/actionscript multi-party web content. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 12(4): 443–457, July 2015.

[33] J. H. Saltzer. Protection and the control of information sharing in multics. *Communications of the ACM*, 17(7):388–402, July 1974.

[34] F. B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1): 30–50, February 2000.

[35] C. Schollier, É. Tanter, and W. D. Meuter. Computational contracts. *Science of Computer Programming*, 98(3):360–375, October 2013.

[36] G. L. Steele, Jr. Macaroni is better than spaghetti. In *Proceedings of the 1977 Symposium on Artificial Intelligence and Programming Languages*, pages 60–66, August 1977.

[37] G. L. Steele Jr and G. J. Sussman. The revised report on SCHEME: A dialect of LISP. Technical Report AIM-452, Massachusetts Institute of Technology Artificial Intelligence Laboratory, 1978.

[38] N. Swamy, J. Chen, and R. Chugh. Enforcing stateful authorization and information flow policies in Fine. In *Proceedings of the 19th European Conference on Programming Languages and Systems (ESOP)*, pages 529–549, March 2010.

[39] A. Takikawa, T. S. Strickland, and S. Tobin-Hochstadt. Constraining delimited control with contracts. In *Proceedings of the 22nd European Conference on Programming Languages and Systems (ESOP)*, pages 229–248, March 2013.

[40] R. Toledo, A. Nunez, E. Tanter, and J. Noye. Aspectizing Java access control. *IEEE Transactions on Software Engineering*, 38(1):101–117, January 2012.

[41] D. Wallach and E. Felten. Understanding Java stack inspection. In *Proceedings of the 1998 IEEE Symposium on Security and Privacy (S&P)*, pages 52–63, May 1998.

[42] D. S. Wallach, D. Balfanz, D. Dean, and E. W. Felten. Extensible security architectures for Java. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles (SOSP)*, pages 116–128, October 1997.

A. Details of Model

A.1 Evaluation Contexts

$$
\begin{array}{rcl}
E & ::= & [\cdot] \mid E\ e \mid v\ E \mid \mathsf{let}\, x = E\, \mathsf{in}\, e \mid \mathsf{if}\ E\ \mathsf{then}\ e\ \mathsf{else}\ e \mid E \oplus e \mid v \oplus E \mid E \le e \mid v \le E \\
& \mid & \mathsf{make\text{-}parameter}\ E \mid ?E \mid E := e \mid \mathsf{p}(r) := E \\
& \mid & \mathsf{parameterize}\ E = e\, \mathsf{in}\ e \mid \mathsf{parameterize}\, \mathsf{p}(r) = E\, \mathsf{in}\ e \mid \mathsf{parameterize}\, \mathsf{p}(r) = v\, \mathsf{in}\ E \\
& \mid & \mathsf{flat/c}(E) \mid \mathsf{param/c}(E) \mid E : \tau \to (e)\ e \mid c : \tau \to (E)\ e \mid c : \tau \to (c)\ E \\
& \mid & E : \tau \to_{\mathsf{a}} (\lambda\, x : \tau.\ e)\ e \mid c : \tau \to_{\mathsf{a}} (\lambda\, x : \tau.\ e)\ E \mid \mathsf{ctx/c}(E, (e \Rightarrow e \leftarrow e), \ldots, e, (e \Rightarrow e \leftarrow e), \ldots) \\
& \mid & \mathsf{ctx/c}(v, (v \Rightarrow v \leftarrow v), \ldots, (E \Rightarrow e \leftarrow e), (e \Rightarrow e \leftarrow e), \ldots, e, (e \Rightarrow e \leftarrow e), \ldots) \\
& \mid & \mathsf{ctx/c}(v, (v \Rightarrow v \leftarrow v), \ldots, (v \Rightarrow E \leftarrow e), (e \Rightarrow e \leftarrow e), \ldots, e, (e \Rightarrow e \leftarrow e), \ldots) \\
& \mid & \mathsf{ctx/c}(v, (v \Rightarrow v \leftarrow v), \ldots, (v \Rightarrow v \leftarrow E), (e \Rightarrow e \leftarrow e), \ldots, e, (e \Rightarrow e \leftarrow e), \ldots) \\
& \mid & \mathsf{ctx/c}(v, (v \Rightarrow v \leftarrow v), \ldots, E, (e \Rightarrow e \leftarrow e), \ldots) \\
& \mid & \mathsf{ctx/c}(v, (v \Rightarrow v \leftarrow v), \ldots, v, (v \Rightarrow v \leftarrow v), \ldots, (E \Rightarrow e \leftarrow e), (e \Rightarrow e \leftarrow e), \ldots) \\
& \mid & \mathsf{ctx/c}(v, (v \Rightarrow v \leftarrow v), \ldots, v, (v \Rightarrow v \leftarrow v), \ldots, (v \Rightarrow E \leftarrow e), (e \Rightarrow e \leftarrow e), \ldots) \\
& \mid & \mathsf{ctx/c}(v, (v \Rightarrow v \leftarrow v), \ldots, v, (v \Rightarrow v \leftarrow v), \ldots, (v \Rightarrow v \leftarrow E), (e \Rightarrow e \leftarrow e), \ldots) \\
& \mid & {}^{\ell}\mathsf{mon}^{k}_{j}(E, e) \mid {}^{\ell}\mathsf{mon}^{k}_{j}(c, E) \mid \mathsf{check}^{k}_{j}(E, e) \\
& \mid & \mathsf{guard}_{j}\ (E, v, v, e) \mid \mathsf{install/p}_{j}\ (v, e, E) \mid \mathsf{install/p}_{j}\ (v, E, v) \\
& \mid & \mathsf{module}\ \ell\ \mathsf{exports}\ x\ \mathsf{with}\ x, \ldots\ \mathsf{where}\ x = v, \ldots, x = E, x = e, \ldots; p
\end{array}
$$

A.2 Typing Judgments

$\boxed{\Sigma \vdash p : \tau}$

$$
\frac{\varnothing; \Sigma \vdash m_1 \triangleright \Gamma_1 \qquad \ldots \qquad \Gamma_{n-1}; \Sigma \vdash m_n \triangleright \Gamma_n \qquad \Gamma_n; \Sigma \vdash e : \tau}{\vdash m_1\ ; \ldots m_n\ ; e : \tau}
$$

$\boxed{\Gamma; \Sigma \vdash m \triangleright \Gamma'}$

$$
\frac{\begin{array}{c}\varnothing; \Sigma \vdash e_1 : \tau_1 \qquad \ldots \qquad \{y_1 : \tau_1, \ldots, y_{n-1} : \tau_{n-1}\}; \Sigma \vdash e_n : \tau_n \qquad \Gamma' = \{y_1 : \tau_1, \ldots, y_n : \tau_n\} \restriction_{\{x_1, \ldots, x_{n'}\}} \\ \forall_{1 \le i \le n'}.\exists_{1 \le h \le n}.x_i \equiv y_h \wedge \tau_h \neq \tau\ \mathsf{ctc} \wedge \tau_h \neq \mathsf{ctx\ ctc} \qquad \forall_{1 \le i \le n'}.\exists_{1 \le h \le n}.x_{c_i} \equiv y_h \wedge \tau_h = \tau\ \mathsf{ctc}\end{array}}{\Gamma; \Sigma \vdash \mathsf{module}\ \ell\ \mathsf{exports}\ x_1\ \mathsf{with}\ x_{c_1}, \ldots, x_{n'}\ \mathsf{with}\ x_{c_{n'}}\ \mathsf{where}\ y_1 = e_1, \ldots, y_n = e_n \triangleright \Gamma \uplus \Gamma'}
$$

$\boxed{\Gamma; \Sigma \vdash e : \tau}$

$$
\frac{}{\Gamma; \Sigma \vdash () : \mathsf{Unit}} \qquad \frac{}{\Gamma; \Sigma \vdash n : \mathsf{Int}} \qquad \frac{}{\Gamma; \Sigma \vdash \#\mathsf{t} : \mathsf{Bool}} \qquad \frac{}{\Gamma; \Sigma \vdash \#\mathsf{f} : \mathsf{Bool}} \qquad \frac{\Gamma(x) = \tau}{\Gamma; \Sigma \vdash x : \tau} \qquad \frac{\Gamma; \Sigma \vdash e_i : \mathsf{Int} \qquad i \in \{1, 2\}}{\Gamma; \Sigma \vdash e_1 \oplus e_2 : \mathsf{Int}}
$$

$$
\frac{\Gamma; \Sigma \vdash e_i : \mathsf{Int} \qquad i \in \{1, 2\}}{\Gamma; \Sigma \vdash e_1 \le e_2 : \mathsf{Bool}} \qquad \frac{\Gamma; \Sigma \vdash e_1 : \tau_1 \qquad \Gamma[x \mapsto \tau_1]; \Sigma \vdash e_2 : \tau}{\Gamma; \Sigma \vdash \mathsf{let}\, x = e_1\, \mathsf{in}\, e_2 : \tau} \qquad \frac{\Gamma; \Sigma \vdash e_c : \mathsf{Bool} \qquad \Gamma; \Sigma \vdash e_i : \tau \qquad i \in \{1, 2\}}{\Gamma; \Sigma \vdash \mathsf{if}\ e_c\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau}
$$

$$
\frac{\Gamma[x \mapsto \tau_1]; \Sigma \vdash e : \tau_2}{\Gamma; \Sigma \vdash \lambda\, x : \tau_1.\ e : \tau_1 \to \tau_2} \qquad \frac{\Gamma[x \mapsto \tau]; \Sigma \vdash e : \tau}{\Gamma; \Sigma \vdash \mu\, x : \tau.\ e : \tau} \qquad \frac{\Gamma; \Sigma \vdash e_1 : \tau_1 \to \tau_2 \qquad \Gamma; \Sigma \vdash e_2 : \tau_1}{\Gamma; \Sigma \vdash e_1\ e_2 : \tau_2} \qquad \frac{\Sigma(r) = \tau}{\Gamma; \Sigma \vdash \mathsf{p}(r) : \tau\ \mathsf{param}}
$$

$$
\frac{\Gamma; \Sigma \vdash e : \tau\ \mathsf{param}}{\Gamma; \Sigma \vdash ?e : \tau} \qquad \frac{\Gamma; \Sigma \vdash e_1 : \tau\ \mathsf{param} \qquad \Gamma; \Sigma \vdash e_2 : \tau}{\Gamma; \Sigma \vdash e_1 := e_2 : \mathsf{Unit}} \qquad \frac{\Gamma; \Sigma \vdash e : \tau}{\Gamma; \Sigma \vdash \mathsf{make\text{-}parameter}\ e : \tau\ \mathsf{param}}
$$

$$
\frac{\Gamma; \Sigma \vdash e_1 : \tau_p\ \mathsf{param} \qquad \Gamma; \Sigma \vdash e_2 : \tau_p \qquad \Gamma; \Sigma \vdash e_3 : \tau}{\Gamma; \Sigma \vdash \mathsf{parameterize}\ e_1 = e_2\ \mathsf{in}\ e_3 : \tau} \qquad \frac{\Gamma; \Sigma \vdash e_1 : \tau\ \mathsf{ctc} \qquad \Gamma; \Sigma \vdash e_2 : \tau}{\Gamma; \Sigma \vdash {}^{\ell}\mathsf{mon}^{k}_{j}(e_1, e_2) : \tau}
$$

$$
\frac{\Gamma; \Sigma \vdash e_1 : \mathsf{ctx\ ctc} \qquad \Gamma; \Sigma \vdash e_2 : (\tau_d \to \tau_r)}{\Gamma; \Sigma \vdash {}^{\ell}\mathsf{mon}^{k}_{j}(e_1, e_2) : (\tau_d \to \tau_r)} \qquad \frac{\Gamma; \Sigma \vdash e_1 : \mathsf{Bool} \qquad \Gamma; \Sigma \vdash v_2 : \tau}{\Gamma; \Sigma \vdash \mathsf{check}^{k}_{j}(e_1, v_2) : \tau} \qquad \frac{\Gamma; \Sigma \vdash e : \beta \to \mathsf{Bool}}{\Gamma; \Sigma \vdash \mathsf{flat/c}(e) : \beta\ \mathsf{ctc}}
$$

$$
\frac{\Gamma; \Sigma \vdash e : \tau\ \mathsf{ctc}}{\Gamma; \Sigma \vdash \mathsf{param/c}(e) : (\tau\ \mathsf{param})\ \mathsf{ctc}} \qquad \frac{\Gamma; \Sigma \vdash e_d : \tau_d\ \mathsf{ctc} \qquad \Gamma; \Sigma \vdash e_r : \tau_r\ \mathsf{ctc} \qquad \Gamma; \Sigma \vdash e_c : \mathsf{ctx\ ctc}}{\Gamma; \Sigma \vdash e_d : \tau_d \to (e_c)\ e_r : \tau_d \to \tau_r\ \mathsf{ctc}}
$$

$$
\frac{\Gamma; \Sigma \vdash e_d : \tau_d\ \mathsf{ctc} \qquad \Gamma; \Sigma \vdash e_r : \tau_r\ \mathsf{ctc} \qquad \Gamma[x \mapsto \tau_d]; \Sigma \vdash e : \mathsf{ctx\ ctc}}{\Gamma; \Sigma \vdash e_d : \tau_d \to_{\mathsf{a}} (\lambda\, x : \tau_d.\ e)\ e_r : \tau_d \to \tau_r\ \mathsf{ctc}}
$$

$$\frac{\Gamma;\Sigma \vdash e_1 : \mathsf{Unit} \to \mathsf{Bool} \quad \Gamma;\Sigma \vdash e_2 : \mathsf{Unit} \to \mathsf{Bool} \quad \Gamma;\Sigma \vdash e_{g_{c_i}} : \mathsf{Unit} \to \mathsf{Bool} \quad \Gamma;\Sigma \vdash e_{p_{c_i}} : \tau_{c_i}\ \mathsf{param} \quad \Gamma;\Sigma \vdash e_{v_{c_i}} : \mathsf{Unit} \to \tau_{c_i} \quad \Gamma;\Sigma \vdash e_{g_{a_i}} : \mathsf{Unit} \to \mathsf{Bool} \quad \Gamma;\Sigma \vdash e_{p_{a_i}} : \tau_{a_i}\ \mathsf{param} \quad \Gamma;\Sigma \vdash e_{v_{a_i}} : \mathsf{Unit} \to \tau_{a_i}}{\Gamma;\Sigma \vdash \mathsf{ctx/c}(e_1,(e_{g_{c_1}} \Rightarrow e_{p_{c_1}} \leftarrow e_{v_{c_1}}),\ldots,e_2,(e_{g_{a_1}} \Rightarrow e_{p_{a_1}} \leftarrow e_{v_{a_1}}),\ldots) : \mathsf{ctx\ ctc}}$$

$$\frac{\Gamma;\Sigma \vdash e_c : \tau\ \mathsf{ctc} \quad \Gamma;\Sigma \vdash e : \tau\ \mathsf{param}}{\Gamma;\Sigma \vdash {}^{\ell}\mathsf{param/p}^{k}_{j}(e_c, e) : \tau\ \mathsf{param}}$$

$$\frac{\Gamma;\Sigma \vdash e_g : \mathsf{Bool} \quad \Gamma;\Sigma \vdash v_p : \tau\ \mathsf{param} \quad \Gamma;\Sigma \vdash v_v : \mathsf{Unit} \to \tau \quad \Gamma;\Sigma \vdash e_f : \tau_d \to \tau_r}{\Gamma;\Sigma \vdash \mathsf{guard}_j\ (e_g,v_p,v_v,e_f) : \tau_d \to \tau_r}$$

$$\frac{\Gamma;\Sigma \vdash v_p : \tau\ \mathsf{param} \quad \Gamma;\Sigma \vdash e_v : \tau \quad \Gamma;\Sigma \vdash e_f : \tau_d \to \tau_r}{\Gamma;\Sigma \vdash \mathsf{install/p}_j\ (v_p,e_v,e_f) : \tau_d \to \tau_r}$$

A.3 Typing Judgments for Authorization Contract Extensions

$\boxed{\Gamma;\Sigma \vdash m \triangleright \Gamma'}$

$$\frac{\begin{array}{c} a_1,\ldots,a_o = \mathsf{action}\ x_{a_1}\ (y_{a_1},\ldots)\ (ce,ae),\ldots,\mathsf{action}\ x_{a_o}\ (y_{a_o},\ldots)\ (ce,ae) \\ \varnothing;\Sigma \vdash e_1 : \tau_1 \quad \ldots \quad \{y_1:\tau_1,\ldots,y_{n-1}:\tau_{n-1}\};\Sigma \vdash e_n : \tau_n \\ \{y_1:\tau_1,\ldots,y_n:\tau_n\};\Sigma \vdash a_1 : \tau_{a_1} \quad \ldots \quad \{y_1:\tau_1,\ldots,y_n:\tau_n,\ldots,x_{a_1}:\tau_{a_1},\ldots,x_{a_{o-1}}:\tau_{a_{o-1}}\};\Sigma \vdash a_o : \tau_{a_o} \\ \{y_1:\tau_1,\ldots,y_n:\tau_n,\ldots,x_{a_1}:\tau_{a_1},\ldots,x_{a_o}:\tau_{a_o}\};\Sigma \vdash e_{n+1} : \tau_{n+1} \\ \ldots \quad \{y_1:\tau_1,\ldots,y_n:\tau_n,\ldots,x_{a_1}:\tau_{a_1},\ldots,x_{a_o}:\tau_{a_o},y_{n+1}:\tau_{n+1},\ldots,y_{m-1}:\tau_{n-1}\};\Sigma \vdash e_m : \tau_m \\ \Gamma' = \{y_1:\tau_1,\ldots,y_m:\tau_n\}\restriction_{\{x_1,\ldots,x_{n'}\}} \\ \forall_{1\le i\le n'}.(\exists_{1\le h\le m}.x_i \equiv y_h \wedge \tau_h \neq \tau\ \mathsf{ctc} \wedge \tau_h \neq \mathsf{ctx\ ctc}) \vee (\exists_{1\le h\le o}.x_i \equiv x_{a_h} \wedge \tau_{a_h} \neq \tau\ \mathsf{ctc} \wedge \tau_{a_h} \neq \mathsf{ctx\ ctc}) \\ \forall_{1\le i\le n'}.\exists_{1\le h\le m}.x_{c_i} \equiv y_h \wedge \tau_h = \tau\ \mathsf{ctc} \end{array}}{\Gamma;\Sigma \vdash \mathsf{module}\ \ell\ \mathsf{exports}\ x_1\ \mathsf{with}\ x_{c_1},\ldots,x_{n'}\ \mathsf{with}\ x_{c_{n'}}\ \mathsf{where}\ y_1 = e_1,\ldots,y_n = e_n, \mathsf{monitor}\ (a_1,\ldots,a_o), y_k = e_k, y_m = e_m \triangleright \Gamma \uplus \Gamma'}$$

$\boxed{\Gamma;\Sigma \vdash e : \tau}$

$$\frac{}{\Gamma;\Sigma \vdash \top : \mathsf{Prin}} \quad \frac{}{\Gamma;\Sigma \vdash \bot : \mathsf{Prin}} \quad \frac{}{\Gamma;\Sigma \vdash \mathcal{P} : \mathsf{Prin}} \quad \frac{}{\Gamma;\Sigma \vdash \mathcal{D} : \mathsf{Dim}} \quad \frac{}{\Gamma;\Sigma \vdash \mathsf{new\text{-}principal} : \mathsf{Prin}}$$

$$\frac{}{\Gamma;\Sigma \vdash \mathsf{new\text{-}dimension} : \mathsf{Dim}} \quad \frac{\Gamma;\Sigma \vdash e_p : \mathsf{Prin} \quad \Gamma;\Sigma \vdash e_d : \mathsf{Dim}}{\Gamma;\Sigma \vdash e_p \triangleright e_d : \mathsf{Prin}} \quad \frac{}{\Gamma;\Sigma \vdash \{\} : \mathsf{DelSet}} \quad \frac{\Gamma;\Sigma \vdash e : \mathsf{Del}}{\Gamma;\Sigma \vdash \{e\} : \mathsf{DelSet}}$$

$$\frac{\Gamma;\Sigma \vdash e_1 : \mathsf{DelSet} \quad \Gamma;\Sigma \vdash e_2 : \mathsf{DelSet}}{\Gamma;\Sigma \vdash e_1 \cup e_2 : \mathsf{DelSet}} \quad \frac{\Gamma;\Sigma \vdash e_1 : \mathsf{DelSet} \quad \Gamma;\Sigma \vdash e_2 : \mathsf{DelSet}}{\Gamma;\Sigma \vdash e_1 \setminus e_2 : \mathsf{DelSet}}$$

$$\frac{\Gamma;\Sigma \vdash e_w : \mathsf{DelSet} \quad \Gamma;\Sigma \vdash e_f : (\tau \to (\mathsf{Del} \to \tau)) \quad \Gamma;\Sigma \vdash e_i : \tau}{\Gamma;\Sigma \vdash (\mathsf{fold}\ e_w\ e_f\ e_i) : \tau}$$

$$\frac{\Gamma;\Sigma \vdash e_w : \mathsf{DelSet} \quad \Gamma;\Sigma \vdash e_s : \mathsf{Prin} \quad \Gamma;\Sigma \vdash e_l : \mathsf{Prin} \quad \Gamma;\Sigma \vdash e_r : \mathsf{Prin}}{\Gamma;\Sigma \vdash e_w\,;e_s \vdash e_l \succeq e_r : \tau}$$

$$\frac{\Gamma;\Sigma \vdash e_s : \mathsf{Prin} \quad \Gamma;\Sigma \vdash e_l : \mathsf{Prin} \quad \Gamma;\Sigma \vdash e_r : \mathsf{Prin}}{\Gamma;\Sigma \vdash e_s \succeq e_l\ @\ e_r : \mathsf{Del}} \quad \frac{\Gamma;\Sigma \vdash e_a : \mathsf{Del} \quad \Gamma[x_s \mapsto \mathsf{Prin}, x_l \mapsto \mathsf{Prin}, x_r \mapsto \mathsf{Prin}];\Sigma \vdash e : \tau}{\Gamma;\Sigma \vdash \mathsf{let}\ x_s \succeq x_l\ @\ x_r = e_a\ \mathsf{in}\ e : \tau}$$

$$\frac{\Gamma[y_1 \mapsto \tau_1,\ldots,y_n \mapsto \tau_n];\Sigma \vdash ce \quad \Gamma[y_1 \mapsto \tau_1,\ldots,y_n \mapsto \tau_n];\Sigma \vdash ae}{\Gamma;\Sigma \vdash \mathsf{action}\ x\ (y_1:\tau_1,\ldots,y_n:\tau_n)\ (ce,ae) : (\tau_1 \to (\ldots \to (\tau_n \to \mathsf{ctx\ ctc})))}$$

$\boxed{\Gamma;\Sigma \vdash ce}$

$$\frac{\begin{array}{c} \Gamma;\Sigma \vdash cee_1 : \mathsf{Del} \quad \Gamma;\Sigma \vdash cee_2 : \mathsf{DelSet} \quad \Gamma;\Sigma \vdash cee_3 : \mathsf{DelSet} \\ \Gamma;\Sigma \vdash cee_4 : \mathsf{Prin} \quad \Gamma;\Sigma \vdash cee_5 : \mathsf{Prin} \quad \Gamma;\Sigma \vdash cee_6 : \mathsf{DelSet} \end{array}}{\Gamma;\Sigma \vdash \mathsf{check:}\ cee_1\ \mathsf{add:}\ cee_2\ \mathsf{remove:}\ cee_3\ \mathsf{set!\text{-}principal:}\ ceee_4\ \mathsf{closure\text{-}principal:}\ cee_5\ \mathsf{closure\text{-}delegations:}\ cee_6}$$

$\boxed{\Gamma;\Sigma \vdash ae}$

$$\frac{\begin{array}{c} \Gamma;\Sigma \vdash aee_1 : \mathsf{Del} \quad \Gamma;\Sigma \vdash aee_2 : \mathsf{DelSet} \quad \Gamma;\Sigma \vdash aee_3 : \mathsf{DelSet} \quad \Gamma;\Sigma \vdash aee_4 : \mathsf{DelSet} \\ \Gamma;\Sigma \vdash aee_5 : \mathsf{Prin} \quad \Gamma;\Sigma \vdash aee_6 : \mathsf{Prin} \quad \Gamma;\Sigma \vdash aee_7 : \mathsf{Prin} \end{array}}{\Gamma;\Sigma \vdash \mathsf{check:}\ aee_1\ \mathsf{add:}\ aee_2\ \mathsf{remove:}\ aee_3\ \mathsf{scope:}\ aeee_4\ \mathsf{set\text{-}principal?:}\ aee_5\ \mathsf{principal:}\ aee_6\ \mathsf{set!\text{-}principal:}\ aee_7}$$

$\boxed{\Gamma;\Sigma \vdash cee : \tau}$ $\boxed{\Gamma;\Sigma \vdash aee : \tau}$

$$\frac{\Gamma;\Sigma \vdash aee_v : \tau_x \qquad \Gamma[x \mapsto \tau_x];\Sigma \vdash aee : \tau}{\Gamma;\Sigma \vdash \mathsf{let}\, x = aee_v\, \mathsf{in}\, aee : \tau} \qquad \frac{\Gamma;\Sigma \vdash cee_v : \tau_x \qquad \Gamma[x \mapsto \tau_x];\Sigma \vdash cee : \tau}{\Gamma;\Sigma \vdash \mathsf{let}\, x = cee_v\, \mathsf{in}\, cee : \tau} \qquad \frac{}{\Gamma;\Sigma \vdash \text{current-principal} : \mathsf{Prin}}$$

$$\frac{}{\Gamma;\Sigma \vdash \text{current-delegations} : \mathsf{DelSet}} \qquad \frac{}{\Gamma;\Sigma \vdash \text{closure-principal} : \mathsf{Prin}} \qquad \frac{}{\Gamma;\Sigma \vdash \text{closure-delegations} : \mathsf{DelSet}}$$

A.4 Compiling Authorization Contract Extensions

$\text{compile}[\![\text{module } \ell \text{ exports } x_1 \text{ with } x_{c_1},\ldots \text{ where } y_1 = e_2,\ldots; p]\!] =$
$\quad \text{compile-monitor}[\![\text{module } \ell \text{ exports } x_1 \text{ with } x_{c_1},\ldots \text{ where } y_1 = e_2,\ldots; \text{compile}[\![p]\!]]\!]$
$\text{compile}[\![e]\!] = e$

$\text{compile-monitor}[\![\text{module } \ell$
$\quad \text{exports } x_1 \text{ with } x_{c_1}, \quad \ldots$
$\quad \text{where } y_1 = e_1,\ldots,y_n = e_n,$
$\quad\quad \text{monitor } (\text{action } x_{a_1}\ (y_{a_1} : \tau_{y_{a_1}},\ldots)\ (ce_1,ae_1),\ldots,\text{action } x_{a_n}\ (y_{a_n} : \tau_{y_{a_n}},\ldots)\ (ce_n,ae_n)),$
$\quad\quad y_{n+1} = e_{n+1},\ldots,y_m = e_m;]\!] =$

$\quad \text{module } \ell$
$\quad \text{exports } x_1 \text{ with } x_{c_1}, \quad \ldots$
$\quad \text{where } y_1 = e_1,\ldots,y_n = e_n,$
$\quad\quad p = \text{make-parameter } \top, d = \text{make-parameter } \{\}, s = \text{make-parameter } \{\},$
$\quad\quad cp = \text{make-parameter } \top, cd = \text{make-parameter } \{\},$
$\quad\quad curp = \text{make-parameter } \top, curd = \text{make-parameter } \{\},$
$\quad\quad x_{a_1} = \text{compile-action}[\![\text{action } x_{a_1}\ (y_{a_1} : \tau_{y_{a_1}},\ldots)\ (ce_1,ae_1), p, d, s, cp, cd, curp, curd]\!],$
$\quad\quad \ldots,$
$\quad\quad x_{a_n} = \text{compile-action}[\![\text{action } x_{a_n}\ (y_{a_n} : \tau_{y_{a_n}},\ldots)\ (ce_n,ae_n), p, d, s, cp, cd, curp, curd]\!],$
$\quad\quad y_{n+1} = e_{n+1},\ldots,y_m = e_m;$
where p, d, s, cp, cd, $curp$, and $curd$ are fresh

$\text{compile-action}[\![\text{action } x\ (y : \tau_y,\ldots)\ (ce,ae), p, d, s, cp, cd, curp, curd]\!] =$
$\quad \lambda\, y : \tau_y.\ \ldots\ .$
$\quad\quad \text{ctx/c}(\text{compile-ce}_{\text{check}}[\![ce, p, d, s, curp, curd]\!],$
$\quad\quad\quad \text{compile-ce}_{\text{cp}}[\![ce, cp, curp, curd]\!],$
$\quad\quad\quad \text{compile-ce}_{\text{cd}}[\![ce, cd, curp, curd]\!],$
$\quad\quad\quad \text{compile-ae}_{\text{check}}[\![ae, p, d, s, cp, cd, curp, curd]\!],$
$\quad\quad\quad \text{compile-ae}_{\text{s}}[\![ae, s, cp, cd, curp, curd]\!],$
$\quad\quad\quad \text{compile-ae}_{\text{p}}[\![ae, p, cp, cd, curp, curd]\!])$

$\text{compile-ce}_{\text{check}}[\![\text{check: } e_1 \text{ add: } e_2 \text{ remove: } e_3 \text{ set!-principal: } e_4 \text{ closure-principal: } e_5 \text{ closure-delegations: } e_6, p, d, s, curp, curd]\!] =$
$\quad\quad \lambda\,\_ : \text{Unit}.$
$\quad\quad\quad \text{let } \_ = curp := ?p \text{ in}$
$\quad\quad\quad \text{let } \_ = curd := ?d \cup ?s \text{ in}$
$\quad\quad\quad \text{let } p_1 \succeq p_2\ @\ p_3 = \text{compile-cee}[\![e_1, curp, curd]\!] \text{ in}$
$\quad\quad\quad \text{if } (?curd)\,; p_3 \vdash p_1 \succeq p_2 \text{ then}$
$\quad\quad\quad\quad \text{let } add = \text{compile-cee}[\![e_2, curp, curd]\!] \text{ in}$
$\quad\quad\quad\quad \text{let } remove = \text{compile-cee}[\![e_3, curp, curd]\!] \text{ in}$
$\quad\quad\quad\quad \text{let } setprin = \text{compile-cee}[\![e_4, curp, curd]\!] \text{ in}$
$\quad\quad\quad\quad \text{let } \_ = d := (?d \cup add)/remove \text{ in}$
$\quad\quad\quad\quad \text{let } \_ = p := setprin \text{ in}$
$\quad\quad\quad\quad \#\text{t}$
$\quad\quad\quad \text{else } \#\text{f}$
$\text{compile-ce}_{\text{cp}}[\![\text{check: } e_1 \text{ add: } e_2 \text{ remove: } e_3 \text{ set!-principal: } e_4 \text{ closure-principal: } e_5 \text{ closure-delegations: } e_6, cp, curp, curd]\!] =$
$\quad\quad ((\lambda\,\_ : \text{Unit}.\ \#\text{t}) \Rightarrow cp \leftarrow (\lambda\,\_ : \text{Unit}.\ \text{compile-cee}[\![e_5, curp, curd]\!]))$
$\text{compile-ce}_{\text{cd}}[\![\text{check: } e_1 \text{ add: } e_2 \text{ remove: } e_3 \text{ set!-principal: } e_4 \text{ closure-principal: } e_5 \text{ closure-delegations: } e_6, cd, curp, curd]\!] =$
$\quad\quad ((\lambda\,\_ : \text{Unit}.\ \#\text{t}) \Rightarrow cd \leftarrow (\lambda\,\_ : \text{Unit}.\ \text{compile-cee}[\![e_6, curp, curd]\!]))$

$\text{compile-ae}_{\text{check}}[\![\text{check: } e_1 \text{ add: } e_2 \text{ remove: } e_3 \text{ scope: } e_4 \text{ set-principal?: } e_5 \text{ principal: } e_6 \text{ set!-principal: } e_7, p, d, s, cp, cd, curp, curd]\!] =$
$\lambda \_ : \text{Unit}.$
$\quad \text{let } \_ = curp := ?p \text{ in}$
$\quad \text{let } \_ = curd := ?d \cup ?s \text{ in}$
$\quad \text{let } p_1 \succeq p_2 \mathbin{@} p_3 = \text{compile-aee}[\![e_1, curp, curd, cp, cd]\!] \text{ in}$
$\quad \text{if } (?curd)\,; p_3 \vdash p_1 \succeq p_2 \text{ then}$
$\quad\quad \text{let } add = \text{compile-aee}[\![e_2, curp, curd, cp, cd]\!] \text{ in}$
$\quad\quad \text{let } remove = \text{compile-aee}[\![e_3, curp, curd, cp, cd]\!] \text{ in}$
$\quad\quad \text{let } setprin = \text{compile-aee}[\![e_7, curp, curd, cp, cd]\!] \text{ in}$
$\quad\quad \text{let } \_ = d := (?d \cup add)/remove \text{ in}$
$\quad\quad \text{let } \_ = p := setprin \text{ in}$
$\quad\quad \#t$
$\quad \text{else } \#f$

$\text{compile-ae}_{\text{p}}[\![\text{check: } e_1 \text{ add: } e_2 \text{ remove: } e_3 \text{ scope: } e_4 \text{ set-principal?: } e_5 \text{ principal: } e_6 \text{ set!-principal: } e_7, p, cp, cd, curp, curd]\!] =$
$(\text{compile-aee}[\![e_5, curp, curd, cp, cd]\!] \Rightarrow p \leftarrow \text{compile-aee}[\![e_6, curp, curd, cp, cd]\!])$

$\text{compile-ae}_{\text{s}}[\![\text{check: } e_1 \text{ add: } e_2 \text{ remove: } e_3 \text{ scope: } e_4 \text{ set-principal?: } e_5 \text{ principal: } e_6 \text{ set!-principal: } e_7, s, cp, cd, curp, curd]\!] =$
$((\lambda \_ : \text{Unit}.\ \#t) \Rightarrow s \leftarrow \text{compile-aee}[\![e_4, curp, curd, cp, cd]\!])$

$$
\begin{array}{lcl}
\text{compile-cee}[\![\text{let } x = cee_1 \text{ in } cee_2, curp, curd]\!] & = & \text{let } x = \text{compile-cee}[\![cee_1, curp, curd]\!] \\
 & & \text{in compile-cee}[\![cee_2, curp, curd]\!] \\
\text{compile-aee}[\![\text{let } x = aee_1 \text{ in } aee_2, curp, curd, cp, cd]\!] & = & \text{let } x = \text{compile-aee}[\![aee_1, curp, curd, cp, clod]\!] \\
 & & \text{in compile-aee}[\![aee_2, curp, curd, cp, cd]\!] \\
\text{compile-cee}[\![\text{current-principal}, curp, curd]\!] & = & ?curp \\
\text{compile-aee}[\![\text{current-principal}, curp, curd, cp, cd]\!] & = & ?curp \\
\text{compile-cee}[\![\text{current-delegations}, curp, curd]\!] & = & ?curd \\
\text{compile-aee}[\![\text{current-delegations}, curp, curd, cp, cd]\!] & = & ?curd \\
\text{compile-aee}[\![\text{closure-principal}, curp, curd, cp, cd]\!] & = & ?cp \\
\text{compile-aee}[\![\text{closure-delegations}, curp, curd, cp, cd]\!] & = & ?cd \\
\text{compile-cee}[\![e, curp, curd]\!] & = & e \\
\text{compile-aee}[\![e, curp, curd, cp, cd]\!] & = & e
\end{array}
$$

A.5 Authorization Contract Extension Evaluation Contexts

$$
\begin{array}{lcl}
E & ::= & \ldots \mid E \triangleright e \mid v \triangleright c \mid E\,; e \vdash e \succeq e \mid v\,; E \vdash e \succeq e \mid v\,; v \vdash E \succeq e \mid v\,; v \vdash v \succeq E \\
 & \mid & E \succeq e \mathbin{@} e \mid v \succeq E \mathbin{@} e \mid v \succeq v \mathbin{@} E \mid \{E\} \mid E \cup e \mid v \cup E \mid E \setminus e \mid v \setminus E \\
 & \mid & (\text{fold } E\ e\ e) \mid (\text{fold } v\ E\ e) \mid (\text{fold } v\ v\ E)
\end{array}
$$

A.6 Reduction Semantics for Authorization Contract Extensions

$$
\begin{array}{lcl}
\langle E[\text{new-principal}], \sigma\rangle & \rightarrow & \langle E[p], \sigma\rangle \text{ where } p \text{ is fresh} \\
\langle E[\text{new-dimension}], \sigma\rangle & \rightarrow & \langle E[d], \sigma\rangle \text{ where } d \text{ is fresh} \\
\langle E[\{p_{1_s} \succeq p_{1_l} \mathbin{@} p_{1_r}, \ldots\}\,; p_s \vdash p_l \succeq p_r], \sigma\rangle & \rightarrow & \langle E[\#t], \sigma\rangle \text{ if } \{p_{1_l} \succeq p_{1_r} \mathbin{@} p_{1_s}, \ldots\}\,; p_s \vdash p_l \succeq p_r \\
\langle E[\{p_{1_s} \succeq p_{1_l} \mathbin{@} p_{1_r}, \ldots\}\,; p_s \vdash p_l \succeq p_r], \sigma\rangle & \rightarrow & \langle E[\#f], \sigma\rangle \text{ if } \{p_{1_l} \succeq p_{1_r} \mathbin{@} p_{1_s}, \ldots\}; p_s \nvdash p_l \succeq p_r \\
\langle E[\{v_{1_1}, \ldots, v_{1_n}\} \cup \{v_{2_1}, \ldots, v_{2_m}\}], \sigma\rangle & \rightarrow & \langle E[\{v_{3_1}, \ldots, v_{3_k}\}], \sigma\rangle \\
 & & \text{where } \{v_{3_1}, \ldots, v_{3_k}\} = \{v_{1_1}, \ldots, v_{1_n}\} \cup \{v_{2_1}, \ldots, v_{2_m}\} \\
\langle E[\{v_{1_1}, \ldots, v_{1_n}\} \setminus \{v_{2_1}, \ldots, v_{2_m}\}], \sigma\rangle & \rightarrow & \langle E[\{v_{3_1}, \ldots, v_{3_k}\}], \sigma\rangle \\
 & & \text{where } \{v_{3_1}, \ldots, v_{3_k}\} = \{v_{1_1}, \ldots, v_{1_n}\} \setminus \{v_{2_1}, \ldots, v_{2_m}\} \\
\langle E[(\text{fold } \{\}\ v_f\ v)], \sigma\rangle & \rightarrow & \langle E[v], \sigma\rangle \\
\langle E[(\text{fold } \{v_1, v_2, \ldots\}\ v_f\ v)], \sigma\rangle & \rightarrow & \langle E[(\text{fold } \{v_2, \ldots\}\ v_f\ ((v_f\ v)\ v_1))], \sigma\rangle \\
\langle E[\text{let } x_s \succeq x_l \mathbin{@} x_r = v_s \succeq v_l \mathbin{@} v_r \text{ in } e], \sigma\rangle & \rightarrow & \langle E[\{^{v_l, v_r, v_s}/_{x_l, x_r, x_s}\}e], \sigma\rangle
\end{array}
$$

B. Proof of complete monitoring

This appendix demonstrates that CtxPCF satisfies complete monitoring. Complete monitoring states that a contract system correctly assigns blame to components that violate their contracts and, crucially, that the contract system can interpose on all uses of a value in a component that did not create that value. We follow Dimoulas et. al's standard techniques for stating and proving complete monitoring. The formalization has two key aspects:

1. We develop an independent mechanism to provide a ground truth for ownership of values. To accomplish this, we devise an annotated version of the CtxPCF semantics that has the same behavior, but also keeps track of ownership by annotating values as they flow from one component to another. To ensure that the annotated semantics is truly an independent system for tracking ownership, the semantic rules that implement contract boundaries do not modify ownership annotations on values that flow through contracts.
2. The semantics of the annotated language are modified so that programs get stuck when a component accesses a "foreign" value, i.e., a value from another component that is not protected by the contract system.

Using this annotated semantics, complete monitoring can be stated as a soundness property that states that well-formed programs (those whose initial ownership annotations agree with the placement of contracts in the surface program) do not get stuck because components access "foreign" values. Using a straightforward subject-reduction technique, we show that CtxPCF (and Ctrl+CtxPCF, its extension with first-class control operators) satisfies complete monitoring.

B.1 Surface CtxPCF

B.1.1 Surface Programs

$$
\begin{array}{rcl}
p & ::= & m\,;p \mid e \\
m & ::= & \mathsf{module}\ \ell\ \mathsf{exports}\ x\ \mathsf{with}\ x,\ldots\ \mathsf{where}\ x = e,\ldots \\
e & ::= & x \mid v \mid e\ e \mid \mu\, x : \tau.\ e \mid \mathsf{let}\, x = e\, \mathsf{in}\, e \mid \mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e \mid e \oplus e \mid e \leq e \\
 & & \mid \mathsf{make\text{-}parameter}\ e \mid \mathsf{parameterize}\, e = e\, \mathsf{in}\, e \mid ?e \mid e := e \\
 & & \mid \mathsf{flat/c}(e) \mid \mathsf{param/c}(e) \mid e : \tau \to (e)\ e \mid e : \tau \to_{\mathsf{a}} (\lambda\, x : \tau.\ e)\ e \\
 & & \mid \mathsf{ctx/c}(e,(e \Rightarrow e \leftarrow e)\ldots,e,(e \Rightarrow e \leftarrow e)\ldots) \\
v & ::= & () \mid n \mid \#\mathsf{t} \mid \#\mathsf{f} \mid \lambda\, x : \tau.\ e \mid c \\
c & ::= & \mathsf{flat/c}(v) \mid \mathsf{param/c}(c) \mid c : \tau \to (c)\ c \mid c : \tau \to_{\mathsf{a}} (\lambda\, x : \tau.\ e)\ c \\
 & & \mid \mathsf{ctx/c}(v,(v \Rightarrow v \leftarrow v)\ldots,v,(v \Rightarrow v \leftarrow v)\ldots) \\
\tau & ::= & \beta \mid \tau \to \tau \mid \tau\ \mathsf{param} \mid \tau\ \mathsf{ctc} \mid \mathsf{ctx\ ctc} \\
\beta & ::= & \mathsf{Unit} \mid \mathsf{Int} \mid \mathsf{Bool}
\end{array}
$$

B.1.2 Well-typed Surface Programs

$\boxed{\vdash p : \tau}$

$$\frac{\varnothing \vdash m_1 \triangleright \Gamma_1 \qquad \ldots \qquad \Gamma_{n-1} \vdash m_n \triangleright \Gamma_n \qquad \Gamma_n \vdash e : \tau}{\vdash m_1\,; \ldots m_n\,; e : \tau}$$

$\boxed{\Gamma \vdash m \triangleright \Gamma'}$

$$\frac{\begin{array}{c}\varnothing \vdash e_1 : \tau_1 \qquad \ldots \qquad \{y_1 : \tau_1, \ldots, y_{n-1} : \tau_{n-1}\} \vdash e_n : \tau_n \qquad \Gamma' = \{y_1 : \tau_1, \ldots, y_n : \tau_n\} \restriction_{\{x_1,\ldots,x_{n'}\}} \\ \forall_{1\leq i\leq n'}.\exists_{1\leq h\leq n}.x_i \equiv y_h \wedge \tau_h \neq \tau\ \mathsf{ctc} \wedge \tau_h \neq \mathsf{ctx\ ctc} \qquad \forall_{1\leq i\leq n'}.\exists_{1\leq h\leq n}.x_{c_i} \equiv y_h \wedge \tau_h = \tau\ \mathsf{ctc}\end{array}}{\Gamma \vdash \mathsf{module}\ \ell\ \mathsf{exports}\ x_1\ \mathsf{with}\ x_{c_1},\ldots,x_{n'}\ \mathsf{with}\ x_{c_{n'}}\ \mathsf{where}\ y_1 = e_1,\ldots,y_n = e_n \triangleright \Gamma \uplus \Gamma'}$$

$\boxed{\Gamma \vdash e : \tau}$

$$\frac{}{\Gamma \vdash () : \mathsf{Unit}} \qquad \frac{}{\Gamma \vdash n : \mathsf{Int}} \qquad \frac{}{\Gamma \vdash \#\mathsf{t} : \mathsf{Bool}} \qquad \frac{}{\Gamma \vdash \#\mathsf{f} : \mathsf{Bool}} \qquad \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma \vdash e_i : \mathsf{Int} \qquad i \in \{1,2\}}{\Gamma \vdash e_1 \oplus e_2 : \mathsf{Int}}$$

$$\frac{\Gamma \vdash e_i : \mathsf{Int} \qquad i \in \{1,2\}}{\Gamma \vdash e_1 \le e_2 : \mathsf{Bool}} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma[x \mapsto \tau_1] \vdash e_2 : \tau}{\Gamma \vdash \mathsf{let}\, x = e_1 \,\mathsf{in}\, e_2 : \tau} \qquad \frac{\Gamma \vdash e_c : \mathsf{Bool} \qquad \Gamma \vdash e_i : \tau \qquad i \in \{1,2\}}{\Gamma \vdash \mathsf{if}\ e_c\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau}$$

$$\frac{\Gamma[x \mapsto \tau_1] \vdash e : \tau_2}{\Gamma \vdash \lambda\, x : \tau_1.\ e : \tau_1 \to \tau_2} \qquad \frac{\Gamma[x \mapsto \tau] \vdash e : \tau}{\Gamma \vdash \mu\, x : \tau.\ e : \tau} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2} \qquad \frac{\Gamma \vdash e : \tau\ \mathsf{param}}{\Gamma \vdash ?e : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau\ \mathsf{param} \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 := e_2 : \mathsf{Unit}} \qquad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathsf{make\text{-}parameter}\ e : \tau\ \mathsf{param}}$$

$$\frac{\Gamma \vdash e_1 : \tau_p\ \mathsf{param} \qquad \Gamma \vdash e_2 : \tau_p \qquad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \mathsf{parameterize}\, e_1 = e_2 \,\mathsf{in}\, e_3 : \tau} \qquad \frac{\Gamma \vdash e : \beta \to \mathsf{Bool}}{\Gamma \vdash \mathsf{flat/c}(e) : \beta\ \mathsf{ctc}} \qquad \frac{\Gamma \vdash e : \tau\ \mathsf{ctc}}{\Gamma \vdash \mathsf{param/c}(e) : (\tau\ \mathsf{param})\ \mathsf{ctc}}$$

$$\frac{\Gamma \vdash e_d : \tau_d\ \mathsf{ctc} \qquad \Gamma \vdash e_r : \tau_r\ \mathsf{ctc} \qquad \Gamma \vdash e_c : \mathsf{ctx}\ \mathsf{ctc}}{\Gamma \vdash e_d : \tau_d \to (e_c)\ e_r : \tau_d \to \tau_r\ \mathsf{ctc}} \qquad \frac{\Gamma \vdash e_d : \tau_d\ \mathsf{ctc} \qquad \Gamma \vdash e_r : \tau_r\ \mathsf{ctc} \qquad \Gamma[x \mapsto \tau_d] \vdash e : \mathsf{ctx}\ \mathsf{ctc}}{\Gamma \vdash e_d : \tau_d \to_{\mathsf{a}} (\lambda\, x : \tau_d.\ e)\ e_r : \tau_d \to \tau_r\ \mathsf{ctc}}$$

$$\frac{\begin{array}{c}\Gamma \vdash e_1 : \mathsf{Unit} \to \mathsf{Bool} \qquad \Gamma \vdash e_2 : \mathsf{Unit} \to \mathsf{Bool} \\ \Gamma \vdash e_{g_{c_i}} : \mathsf{Unit} \to \mathsf{Bool} \qquad \Gamma \vdash e_{p_{c_i}} : \tau_{c_i}\ \mathsf{param} \qquad \Gamma \vdash e_{v_{c_i}} : \mathsf{Unit} \to \tau_{c_i} \\ \Gamma \vdash e_{g_{a_i}} : \mathsf{Unit} \to \mathsf{Bool} \qquad \Gamma \vdash e_{p_{a_i}} : \tau_{a_i}\ \mathsf{param} \qquad \Gamma \vdash e_{v_{a_i}} : \mathsf{Unit} \to \tau_{a_i}\end{array}}{\Gamma \vdash \mathsf{ctx/c}(e_1, (e_{g_{c_1}} \Rightarrow e_{p_{c_1}} \leftarrow e_{v_{c_1}}), \ldots, e_2, (e_{g_{a_1}} \Rightarrow e_{p_{a_1}} \leftarrow e_{v_{a_1}}), \ldots) : \mathsf{ctx}\ \mathsf{ctc}}$$

B.2 Annotated Surface CtxPCF

B.2.1 Annotated Surface Programs

$$e \quad ::= \quad \ldots \mid {}^{\ell}\langle\langle {}^{\ell} x \rangle\rangle$$

B.2.2 Additional Typing Rules for Annotated Surface Programs

$\boxed{\Gamma \vdash e : \tau}$

$$\frac{\Gamma \vdash x : \tau}{\Gamma \vdash {}^{\ell}\langle\langle {}^{k} x \rangle\rangle : \tau}$$

B.2.3 Well-formed Annotated Surface Programs

$\boxed{\ell_0 \Vdash p}$

$$\frac{\varnothing \Vdash m_1 \rhd \Delta_1 \qquad \ldots \qquad \Delta_{n-1} \Vdash m_n \rhd \Delta_n \qquad \Delta_n; \ell_0 \Vdash e}{\ell_0 \Vdash m_1 \,; \ldots m_n \,; e}$$

$\boxed{\Delta \Vdash m \rhd \Delta'}$

$$\frac{\varnothing; \ell \Vdash e_1 \qquad \ldots \qquad \{y_1 : \ell, \ldots, y_{n-1} : \ell\}; \ell \Vdash e_n \qquad \Gamma' = \{y_1 : \ell, \ldots, y_n : \ell\} \restriction_{\{x_1, \ldots, x_{n'}\}}}{\Delta \Vdash \mathsf{module}\ \ell\ \mathsf{exports}\ x_1\ \mathsf{with}\ x_{c_1}, \ldots, x_{n'}\ \mathsf{with}\ x_{c_{n'}}\ \mathsf{where}\ y_1 = e_1, \ldots, y_n = e_n \rhd \Delta \uplus \Delta'}$$

$\boxed{\Delta; \ell \Vdash e}$

$$\frac{}{\Delta; \ell \Vdash ()} \qquad \frac{}{\Delta; \ell \Vdash n} \qquad \frac{}{\Delta; \ell \Vdash \#\mathsf{t}} \qquad \frac{}{\Delta; \ell \Vdash \#\mathsf{f}} \qquad \frac{\Delta(x) = \ell}{\Delta; \ell \Vdash x} \qquad \frac{\Delta; \ell \Vdash e_i \qquad i \in \{1, 2\}}{\Delta; \ell \Vdash e_1 \oplus e_2}$$

$$\frac{\Delta; \ell \Vdash e_i \qquad i \in \{1, 2\}}{\Delta; \ell \Vdash e_1 \leq e_2} \qquad \frac{\Delta; \ell \Vdash e_1 \qquad \Delta[x \mapsto \ell]; \Pi \Vdash \ell e_2}{\Delta; \ell \Vdash \mathsf{let}\, x = e_1\, \mathsf{in}\, e_2} \qquad \frac{\Delta; \ell \Vdash e_c \qquad \Delta; \ell \Vdash e_i \qquad i \in \{1, 2\}}{\Delta; \ell \Vdash \mathsf{if}\, e_c\, \mathsf{then}\, e_1\, \mathsf{else}\, e_2}$$

$$\frac{\Delta[x \mapsto \ell]; \ell \Vdash e}{\Delta; \ell \Vdash \lambda\, x : \tau.\ e} \qquad \frac{\Delta[x \mapsto \ell]; \ell \Vdash e}{\Delta; \ell \Vdash \mu\, x : \tau.\ e} \qquad \frac{\Delta; \ell \Vdash e_1 \qquad \Delta; \ell \Vdash e_2}{\Delta; \ell \Vdash e_1\ e_2} \qquad \frac{\Delta; \ell \Vdash e}{\Delta; \ell \Vdash ?e} \qquad \frac{\Delta; \ell \Vdash e_1 \qquad \Delta; \ell \Vdash e_2}{\Delta; \ell \Vdash e_1 := e_2}$$

$$\frac{\Delta; \ell \Vdash e}{\Delta; \ell \Vdash \mathsf{make\text{-}parameter}\ e} \qquad \frac{\Delta; \ell \Vdash e_1 \qquad \Delta; \ell \Vdash e_2 \qquad \Delta; \ell \Vdash e_3}{\Delta; \ell \Vdash \mathsf{parameterize}\, e_1 = e_2\, \mathsf{in}\, e_3} \qquad \frac{\Delta; \ell \Vdash e}{\Delta; \ell \Vdash \mathsf{flat/c}(e)} \qquad \frac{\Delta; \ell \Vdash e}{\Delta; \ell \Vdash \mathsf{param/c}(e)}$$

$$\frac{\Delta; \ell \Vdash e_1 \qquad \Delta; \ell \Vdash e_2 \qquad \Delta; \ell \Vdash e_3}{\Delta; \ell \Vdash e_1 : \tau \to_{\mathsf{a}} (\lambda\, x : \tau.\ e_2)\ e_3} \qquad \frac{\Delta; \ell \Vdash e_1 \qquad \Delta; \ell \Vdash e_2 \qquad \Delta; \ell \Vdash e_3}{\Delta; \ell \Vdash e_1 : \tau \to (e_2)\ e_3}$$

$$\frac{\Delta; \ell \Vdash e_1 \qquad \Delta; \ell \Vdash e_2 \qquad \Delta; \ell \Vdash e_{g_{c_i}} \qquad \Delta; \ell \Vdash e_{p_{c_i}} \qquad \Delta; \ell \Vdash e_{v_{c_i}} \qquad \Delta; \ell \Vdash e_{g_{a_i}} \qquad \Delta; \ell \Vdash e_{p_{a_i}} \qquad \Delta; \ell \Vdash e_{v_{a_i}}}{\Delta; \ell \Vdash \mathsf{ctx/c}(e_1, (e_{g_{c_1}} \Rightarrow e_{p_{c_1}} \leftarrow e_{v_{c_1}}), \ldots, e_2, (e_{g_{a_1}} \Rightarrow e_{p_{a_1}} \leftarrow e_{v_{a_1}}), \ldots)}$$

$$\frac{\Delta; k \Vdash x}{\Delta; \ell \Vdash {}^{\ell}\langle\langle {}^{k} x \rangle\rangle}$$

B.3 CtxPCF

B.3.1 Programs

$$
\begin{array}{rcl}
p & ::= & m\,;p \mid e \\
m & ::= & \text{module } \ell \text{ exports } x \text{ with } x,\ldots \text{ where } x = e,\ldots \\
e & ::= & v \mid e\ e \mid \mu\, x : \tau.\ e \mid \text{let } x = e \text{ in } e \mid \text{if } e \text{ then } e \text{ else } e \mid e \oplus e \mid e \leq e \\
 & \mid & \text{make-parameter } e \mid \text{parameterize } e = e \text{ in } e \mid ?e \mid e := e \\
 & \mid & \text{flat/c}(e) \mid \text{param/c}(e) \mid e : \tau \to (e)\ e \mid e : \tau \to_{\text{a}} (\lambda\, x : \tau.\ e)\ e \\
 & \mid & \text{ctx/c}(e,(e \Rightarrow e \leftarrow e)\ldots,e,(e \Rightarrow e \leftarrow e)\ldots) \\
 & \mid & {}^{\ell}\text{mon}_j^k(e,e) \mid \text{guard}_j\ (e,v,v,e) \mid \text{install/p}_j\ (v,e,e) \mid \text{check}_j^k(e,e) \mid \text{error}_j^k \\
v & ::= & () \mid n \mid \#\text{t} \mid \#\text{f} \mid \lambda\, x : \tau.\ e \mid c \mid \text{p}(r) \\
 & \mid & {}^{\ell}\text{param/p}_j^k(c,v) \mid {}^{\ell}\text{ctx/p}_j^k\ (v,(v \Rightarrow v \leftarrow v),\ldots,v) \mid \text{install/p}_j\ (v,v,v) \\
c & ::= & \text{flat/c}(v) \mid \text{param/c}(c) \mid c : \tau \to (c)\ c \mid c : \tau \to_{\text{a}} (\lambda\, x : \tau.\ e)\ c \\
 & \mid & \text{ctx/c}(v,(v \Rightarrow v \leftarrow v)\ldots,v,(v \Rightarrow v \leftarrow v)\ldots) \\
\tau & ::= & \beta \mid \tau \to \tau \mid \tau\ \text{param} \mid \tau\ \text{ctc} \mid \text{ctx ctc} \\
\beta & ::= & \text{Unit} \mid \text{Int} \mid \text{Bool}
\end{array}
$$

B.3.2 Well-typed Programs

$\boxed{\Sigma \vdash p : \tau}$

$$\frac{\varnothing;\Sigma \vdash m_1 \triangleright \Gamma_1 \qquad \ldots \qquad \Gamma_{n-1};\Sigma \vdash m_n \triangleright \Gamma_n \qquad \Gamma_n;\Sigma \vdash e : \tau}{\vdash m_1\,;\ldots m_n\,;e : \tau}$$

$\boxed{\Gamma;\Sigma \vdash m \triangleright \Gamma'}$

$$\frac{\begin{array}{c}\varnothing;\Sigma \vdash e_1 : \tau_1 \qquad \ldots \qquad \{y_1 : \tau_1,\ldots,y_{n-1} : \tau_{n-1}\};\Sigma \vdash e_n : \tau_n \qquad \Gamma' = \{y_1 : \tau_1,\ldots,y_n : \tau_n\} \restriction_{\{x_1,\ldots,x_{n'}\}} \\ \forall_{1\leq i\leq n'}.\exists_{1\leq h\leq n}.x_i \equiv y_h \wedge \tau_h \neq \tau\ \text{ctc} \wedge \tau_h \neq \text{ctx ctc} \qquad \forall_{1\leq i\leq n'}.\exists_{1\leq h\leq n}.x_{c_i} \equiv y_h \wedge \tau_h = \tau\ \text{ctc}\end{array}}{\Gamma;\Sigma \vdash \text{module } \ell \text{ exports } x_1 \text{ with } x_{c_1},\ldots,x_{n'} \text{ with } x_{c_{n'}} \text{ where } y_1 = e_1,\ldots,y_n = e_n \triangleright \Gamma \uplus \Gamma'}$$

$\boxed{\Gamma;\Sigma \vdash e : \tau}$

$$\frac{}{\Gamma;\Sigma \vdash () : \text{Unit}} \qquad \frac{}{\Gamma;\Sigma \vdash n : \text{Int}} \qquad \frac{}{\Gamma;\Sigma \vdash \#\text{t} : \text{Bool}} \qquad \frac{}{\Gamma;\Sigma \vdash \#\text{f} : \text{Bool}} \qquad \frac{\Gamma(x) = \tau}{\Gamma;\Sigma \vdash x : \tau}$$

$$\frac{\Gamma;\Sigma \vdash e_i : \text{Int} \qquad i \in \{1,2\}}{\Gamma;\Sigma \vdash e_1 \oplus e_2 : \text{Int}} \qquad \frac{\Gamma;\Sigma \vdash e_i : \text{Int} \qquad i \in \{1,2\}}{\Gamma;\Sigma \vdash e_1 \leq e_2 : \text{Bool}} \qquad \frac{\Gamma;\Sigma \vdash e_1 : \tau_1 \qquad \Gamma[x \mapsto \tau_1];\Sigma \vdash e_2 : \tau}{\Gamma;\Sigma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau}$$

$$\frac{\Gamma;\Sigma \vdash e_c : \text{Bool} \qquad \Gamma;\Sigma \vdash e_i : \tau \qquad i \in \{1,2\}}{\Gamma;\Sigma \vdash \text{if } e_c \text{ then } e_1 \text{ else } e_2 : \tau} \qquad \frac{\Gamma[x \mapsto \tau_1];\Sigma \vdash e : \tau_2}{\Gamma;\Sigma \vdash \lambda\, x : \tau_1.\ e : \tau_1 \to \tau_2} \qquad \frac{\Gamma[x \mapsto \tau];\Sigma \vdash e : \tau}{\Gamma;\Sigma \vdash \mu\, x : \tau.\ e : \tau}$$

$$\frac{\Gamma;\Sigma \vdash e_1 : \tau_1 \to \tau_2 \qquad \Gamma;\Sigma \vdash e_2 : \tau_1}{\Gamma;\Sigma \vdash e_1\ e_2 : \tau_2}$$

$$\frac{\Sigma(r) = \tau}{\Gamma;\Sigma \vdash \mathsf{p}(r) : \tau\ \mathsf{param}} \qquad \frac{\Gamma;\Sigma \vdash e : \tau\ \mathsf{param}}{\Gamma;\Sigma \vdash ?e : \tau} \qquad \frac{\Gamma;\Sigma \vdash e_1 : \tau\ \mathsf{param} \qquad \Gamma;\Sigma \vdash e_2 : \tau}{\Gamma;\Sigma \vdash e_1 := e_2 : \mathsf{Unit}}$$

$$\frac{\Gamma;\Sigma \vdash e : \tau}{\Gamma;\Sigma \vdash \mathsf{make\text{-}parameter}\ e : \tau\ \mathsf{param}} \qquad \frac{\Gamma;\Sigma \vdash e_1 : \tau_p\ \mathsf{param} \qquad \Gamma;\Sigma \vdash e_2 : \tau_p \qquad \Gamma;\Sigma \vdash e_3 : \tau}{\Gamma;\Sigma \vdash \mathsf{parameterize}\ e_1 = e_2\ \mathsf{in}\ e_3 : \tau}$$

$$\frac{\Gamma;\Sigma \vdash e : \beta \to \mathsf{Bool}}{\Gamma;\Sigma \vdash \mathsf{flat/c}(e) : \beta\ \mathsf{ctc}} \qquad \frac{\Gamma;\Sigma \vdash e : \tau\ \mathsf{ctc}}{\Gamma;\Sigma \vdash \mathsf{param/c}(e) : (\tau\ \mathsf{param})\ \mathsf{ctc}} \qquad \frac{\Gamma;\Sigma \vdash e : (typ_d \to \tau_r)\ \mathsf{ctc}}{\Gamma;\Sigma \vdash \mathsf{tag/c}(e) : ((\tau_d \to \tau_r)\ \mathsf{tag})\ \mathsf{ctc}}$$

$$\frac{\Gamma;\Sigma \vdash e_d : \tau_d\ \mathsf{ctc} \qquad \Gamma;\Sigma \vdash e_r : \tau_r\ \mathsf{ctc} \qquad \Gamma;\Sigma \vdash e_c : \mathsf{ctx\ ctc}}{\Gamma;\Sigma \vdash e_d : \tau_d \to (e_c)\ e_r : \tau_d \to \tau_r\ \mathsf{ctc}}$$

$$\frac{\Gamma;\Sigma \vdash e_d : \tau_d\ \mathsf{ctc} \qquad \Gamma;\Sigma \vdash e_r : \tau_r\ \mathsf{ctc} \qquad \Gamma[x \mapsto \tau_d];\Sigma \vdash e : \mathsf{ctx\ ctc}}{\Gamma;\Sigma \vdash e_d : \tau_d \to_{\mathsf{a}} (\lambda\, x : \tau_d.\ e)\ e_r : \tau_d \to \tau_r\ \mathsf{ctc}}$$

$$\frac{\begin{array}{c}\Gamma;\Sigma \vdash e_1 : \mathsf{Unit} \to \mathsf{Bool} \qquad \Gamma;\Sigma \vdash e_2 : \mathsf{Unit} \to \mathsf{Bool} \\ \Gamma;\Sigma \vdash e_{g_{c_i}} : \mathsf{Unit} \to \mathsf{Bool} \qquad \Gamma;\Sigma \vdash e_{p_{c_i}} : \tau_{c_i}\ \mathsf{param} \qquad \Gamma;\Sigma \vdash e_{v_{c_i}} : \mathsf{Unit} \to \tau_{c_i} \\ \Gamma;\Sigma \vdash e_{g_{a_i}} : \mathsf{Unit} \to \mathsf{Bool} \qquad \Gamma;\Sigma \vdash e_{p_{a_i}} : \tau_{a_i}\ \mathsf{param} \qquad \Gamma;\Sigma \vdash e_{v_{a_i}} : \mathsf{Unit} \to \tau_{a_i}\end{array}}{\Gamma;\Sigma \vdash \mathsf{ctx/c}(e_1,(e_{g_{c_1}} \Rightarrow e_{p_{c_1}} \leftarrow e_{v_{c_1}}),\ldots,e_2,(e_{g_{a_1}} \Rightarrow e_{p_{a_1}} \leftarrow e_{v_{a_1}}),\ldots) : \mathsf{ctx\ ctc}}$$

$$\frac{\Gamma;\Sigma \vdash e_1 : \tau\ \mathsf{ctc} \qquad \Gamma;\Sigma \vdash e_2 : \tau}{\Gamma;\Sigma \vdash {}^{\ell}\mathsf{mon}^k_j(e_1,e_2) : \tau} \qquad \frac{\Gamma;\Sigma \vdash e_1 : \mathsf{ctx\ ctc} \qquad \Gamma;\Sigma \vdash e_2 : (\tau_d \to \tau_r)}{\Gamma;\Sigma \vdash {}^{\ell}\mathsf{mon}^k_j(e_1,e_2) : (\tau_d \to \tau_r)}$$

$$\frac{\Gamma;\Sigma \vdash e_1 : \mathsf{Bool} \qquad \Gamma;\Sigma \vdash v_2 : \tau}{\Gamma;\Sigma \vdash \mathsf{check}^k_j(e_1,v_2) : \tau} \qquad \frac{\Gamma;\Sigma \vdash e_c : \tau\ \mathsf{ctc} \qquad \Gamma;\Sigma \vdash e : \tau\ \mathsf{param}}{\Gamma;\Sigma \vdash {}^{\ell}\mathsf{param/p}^k_j(e_c,e) : \tau\ \mathsf{param}}$$

$$\frac{\Gamma;\Sigma \vdash e_g : \mathsf{Bool} \qquad \Gamma;\Sigma \vdash v_p : \tau\ \mathsf{param} \qquad \Gamma;\Sigma \vdash v_v : \mathsf{Unit} \to \tau \qquad \Gamma;\Sigma \vdash e_f : \tau_d \to \tau_r}{\Gamma;\Sigma \vdash \mathsf{guard}_j\ (e_g,v_p,v_v,e_f) : \tau_d \to \tau_r}$$

$$\frac{\Gamma;\Sigma \vdash v_p : \tau\ \mathsf{param} \qquad \Gamma;\Sigma \vdash e_v : \tau \qquad \Gamma;\Sigma \vdash e_f : \tau_d \to \tau_r}{\Gamma;\Sigma \vdash \mathsf{install/p}_j\ (v_p,e_v,e_f) : \tau_d \to \tau_r}$$

$\boxed{\Gamma;\Sigma \vdash \sigma}$

$$\frac{\mathrm{dom}(\Sigma) = \mathrm{dom}(\sigma) \qquad \forall p \in \mathrm{dom}(\sigma), \Gamma;\Sigma \vdash \Sigma(p) : \sigma(p)}{\Gamma;\Sigma \vdash \sigma}$$

B.3.3 Evaluation Contexts

Note. Values v appearing in evaluation contexts are closed terms.

$$
\begin{array}{rcl}
E & ::= & [\cdot] \mid E\ e \mid v\ E \mid \mathsf{let}\, x = E\, \mathsf{in}\, e \mid \mathsf{if}\ E\ \mathsf{then}\ e\ \mathsf{else}\ e \mid E \oplus e \mid v \oplus E \\
& \mid & E \leq e \mid v \leq E \mid \mathsf{make\text{-}parameter}\ E \mid ?E \mid E := e \mid \mathsf{p}(r) := E \\
& \mid & \mathsf{parameterize}\ E = e\ \mathsf{in}\ e \mid \mathsf{parameterize}\ \mathsf{p}(r) = E\ \mathsf{in}\ e \mid \mathsf{parameterize}\ \mathsf{p}(r) = v\ \mathsf{in}\ E \\
& \mid & \mathsf{flat/c}(E) \mid \mathsf{param/c}(E) \mid \mathsf{tag/c}(E) \\
& \mid & E : \tau \to (e)\ e \mid c : \tau \to (E)\ e \mid c : \tau \to (c)\ E \\
& \mid & E : \tau \to_{\mathsf{a}} (\lambda\, x : \tau.\ e)\ e \mid c : \tau \to_{\mathsf{a}} (\lambda\, x : \tau.\ e)\ E \\
& \mid & \mathsf{ctx/c}(E,(e \Rightarrow e \leftarrow e), \ldots, e, (e \Rightarrow e \leftarrow e), \ldots) \\
& \mid & \mathsf{ctx/c}(v,(v \Rightarrow v \leftarrow v), \ldots, (E \Rightarrow e \leftarrow e), (e \Rightarrow e \leftarrow e), \ldots, e, (e \Rightarrow e \leftarrow e), \ldots) \\
& \mid & \mathsf{ctx/c}(v,(v \Rightarrow v \leftarrow v), \ldots, (v \Rightarrow E \leftarrow e), (e \Rightarrow e \leftarrow e), \ldots, e, (e \Rightarrow e \leftarrow e), \ldots) \\
& \mid & \mathsf{ctx/c}(v,(v \Rightarrow v \leftarrow v), \ldots, (v \Rightarrow v \leftarrow E), (e \Rightarrow e \leftarrow e), \ldots, e, (e \Rightarrow e \leftarrow e), \ldots) \\
& \mid & \mathsf{ctx/c}(v,(v \Rightarrow v \leftarrow v), \ldots, E, (e \Rightarrow e \leftarrow e), \ldots)\tau \\
& \mid & \mathsf{ctx/c}(v,(v \Rightarrow v \leftarrow v), \ldots, v, (v \Rightarrow v \leftarrow v), \ldots, (E \Rightarrow e \leftarrow e), (e \Rightarrow e \leftarrow e), \ldots) \\
& \mid & \mathsf{ctx/c}(v,(v \Rightarrow v \leftarrow v), \ldots, v, (v \Rightarrow v \leftarrow v), \ldots, (v \Rightarrow E \leftarrow e), (e \Rightarrow e \leftarrow e), \ldots) \\
& \mid & \mathsf{ctx/c}(v,(v \Rightarrow v \leftarrow v), \ldots, v, (v \Rightarrow v \leftarrow v), \ldots, (v \Rightarrow v \leftarrow E), (e \Rightarrow e \leftarrow e), \ldots) \\
& \mid & {}^{\ell}\mathsf{mon}^{k}_{j}(E,e) \mid {}^{\ell}\mathsf{mon}^{k}_{j}(c,E) \mid \mathsf{check}^{k}_{j}(E,e) \\
& \mid & \mathsf{guard}_j\ (E,v,v,e) \mid \mathsf{install/p}_j\ (v,e,E) \mid \mathsf{install/p}_j\ (v,E,v) \\
& \mid & \mathsf{module}\ \ell\ \mathsf{exports}\ x\ \mathsf{with}\ x, \ldots\ \mathsf{where}\ x = v, \ldots, x = E, x = e, \ldots; p
\end{array}
$$

B.3.4 Reduction Semantics

$$
\begin{array}{l}
\langle \mathsf{module}\ \ell\ \mathsf{exports}\ x_1\ \mathsf{with}\ x_{c_1}, \ldots\ \mathsf{where}\ y_1 = v_1, \ldots, y_n = v_n, y_{e_1} = e_1, \ldots y_{e_m} = e_m; p, \sigma \rangle \\
\quad \to \quad \langle \mathsf{module}\ \ell\ \mathsf{exports}\ x_1\ \mathsf{with}\ x_{c_1}, \ldots\ \mathsf{where}\ y_1 = v_1, \ldots, y_n = v_n, y_{e_1} = \{^{v_i}/_{y_i}\}e_1, y_{e_2} = e_2, \ldots y_{e_m} = e_m; p, \sigma \rangle \\
\langle \mathsf{module}\ \ell\ \mathsf{exports}\ x_1\ \mathsf{with}\ x_{c_1}, \ldots\ \mathsf{where}\ \ldots, x_1 = v_1, \ldots, x_{c_1} = c_1, \ldots; p, \sigma \rangle \\
\quad \to \quad \langle \mathrm{import}[\![\ell, (x_1, \ldots, x_n), (v_1, \ldots, v_n), (c_1, \ldots, c_n), p]\!], \sigma \rangle
\end{array}
$$

$$
\begin{array}{lcl}
\langle E[(\lambda\, x : \tau.\ e)\ v], \sigma \rangle & \to & \langle E[\{^{v}/_{x}\}e], \sigma \rangle \\
\langle E[\mu\, x : \tau.\ e], \sigma \rangle & \to & \langle E[\{^{\mu\, x:\tau.\ e}/_{x}\}e], \sigma \rangle \\
\langle E[\mathsf{let}\, x = v\, \mathsf{in}\, e], \sigma \rangle & \to & \langle E[\{^{v}/_{x}\}e], \sigma \rangle \\
\langle E[\mathsf{if\ \#t\ then}\ e_1\ \mathsf{else}\ e_2], \sigma \rangle & \to & \langle E[e_1], \sigma \rangle \\
\langle E[\mathsf{if\ \#f\ then}\ e_1\ \mathsf{else}\ e_2], \sigma \rangle & \to & \langle E[e_2], \sigma \rangle \\
\langle E[v_1 \oplus v_2], \sigma \rangle & \to & \langle E[v], \sigma \rangle \text{ where } v = v_1 \oplus v_2 \\
\langle E[v_1 \leq v_2], \sigma \rangle & \to & \langle E[v], \sigma \rangle \text{ where } v = v_1 \leq v_2 \\
\langle E[\mathsf{make\text{-}parameter}\ v], \sigma \rangle & \to & \langle E[\mathsf{p}(r)], \sigma[r \mapsto v] \rangle \text{ where } r \text{ is fresh in } \sigma \\
\langle E[?\mathsf{p}(r)], \sigma \rangle & \to & \langle v, \sigma \rangle \text{ where } \sigma(r) = v \text{ and} \\
 & & E \text{ does not contain } \mathsf{parameterize}\ \mathsf{p}(r) = v' \text{ in } E' \\
\langle E[\mathsf{parameterize}\ \mathsf{p}(r) = v\ \mathsf{in}\ E'[?r]], \sigma \rangle & \to & \langle E[\mathsf{parameterize}\ \mathsf{p}(r) = v\ \mathsf{in}\ E'[v]], \sigma \rangle \\
 & & \text{where } E' \text{ does not contain } \mathsf{parameterize}\ \mathsf{p}(r) = v' \text{ in } E'' \\
\langle E[\mathsf{p}(r) := v], \sigma \rangle & \to & \langle E[()], \sigma[r \mapsto v] \rangle \\
 & & \text{where } E \text{ does not contain } \mathsf{parameterize}\ \mathsf{p}(r) = v' \text{ in } E' \\
\langle E[\mathsf{parameterize}\ \mathsf{p}(r) = v\ \mathsf{in}\ E'[\mathsf{p}(r) := v']], \sigma \rangle & \to & \langle E[\mathsf{parameterize}\ \mathsf{p}(r) = v'\ \mathsf{in}\ E'[v']], \sigma \rangle \\
 & & \text{where } E' \text{ does not contain } \mathsf{parameterize}\ \mathsf{p}(r) = v'' \text{ in } E'' \\
\langle E[\mathsf{parameterize}\ \mathsf{p}(r) = v\ \mathsf{in}\ v'], \sigma \rangle & \to & \langle E[v'], \sigma \rangle
\end{array}
$$

$\langle E[{}^{\ell}\mathsf{mon}^{k}_{j}(\mathsf{flat/c}(v_c),v)],\sigma\rangle \quad\to\quad \langle E[\mathsf{check}^{k}_{j}((v_c\ v),v)],\sigma\rangle$

$\langle E[{}^{\ell}\mathsf{mon}^{k}_{j}(\mathsf{param/c}(c),v)],\sigma\rangle \quad\to\quad \langle E[{}^{\ell}\mathsf{param/p}^{k}_{j}(c,v)],\sigma\rangle$

$\langle E[{}^{\ell}\mathsf{mon}^{k}_{j}(c_d : \tau \to (c_c)\ c_r,v)],\sigma\rangle \quad\to\quad \langle E[({}^{\ell}\mathsf{mon}^{k}_{j}(c_c,\lambda\, x : \tau_d.\ {}^{\ell}\mathsf{mon}^{k}_{j}(c_r,v_f\ {}^{k}\mathsf{mon}^{\ell}_{j}(c_d,x)))\ v)],\sigma\rangle$
where x is fresh

$\langle E[{}^{\ell}\mathsf{mon}^{k}_{j}(c_d : \tau \to_{\mathsf{a}} (\lambda\, x : \tau.\ e_c)\ c_r,v)],\sigma\rangle$
$\quad\to\quad \langle E[\lambda\, y : \tau_d.\ {}^{\ell}\mathsf{mon}^{k}_{j}(\{{}^{k\mathsf{mon}^{\ell}_{j}(c_d,y)}/_{x}\}e_c,{}^{\ell}\mathsf{mon}^{k}_{j}(c_r,v\ {}^{k}\mathsf{mon}^{\ell}_{j}(c_d,y)))],\sigma\rangle$ where y is fresh

$\langle E[{}^{\ell}\mathsf{mon}^{k}_{j}(\mathsf{ctx/c}(v_c,(v_{c_{g_1}} \Rightarrow v_{c_{p_1}} \leftarrow v_{c_{v_1}}),\ldots(v_{c_{g_n}} \Rightarrow v_{c_{p_n}} \leftarrow v_{c_{v_n}}),v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\ldots),v)],\sigma\rangle$
$\quad\to\quad \langle E[\mathsf{check}^{\ell}_{j}((v_c\ ()),e)],\sigma\rangle$
where $e = \mathsf{guard}_j\ ((v_{c_{g_1}}\ ()),v_{c_{p_1}},v_{c_{v_1}},\ldots\mathsf{guard}_j\ ((v_{c_{g_n}}\ ()),v_{c_{p_n}},v_{c_{v_n}},{}^{\ell}\mathsf{ctx/p}^{k}_{j}\ (v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\ldots,v)))$

$\langle E[({}^{\ell}\mathsf{ctx/p}^{k}_{j}\ (v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\ldots(v_{a_{g_n}} \Rightarrow v_{a_{p_n}} \leftarrow v_{a_{v_n}}),v_f)\ v)],\sigma\rangle$
$\quad\to\quad \langle E[(\mathsf{check}^{\ell}_{j}((v_a\ ()),e)\ v)],\sigma\rangle$
where $e = \mathsf{guard}_j\ ((v_{a_{g_1}}\ ()),v_{a_{p_1}},v_{a_{v_1}},\ldots\mathsf{guard}_j\ ((v_{a_{g_n}}\ ()),v_{a_{p_n}},v_{a_{v_n}},v_f))$

$\langle E[\mathsf{guard}_j\ (\#\mathsf{f},v_p,v_v,e_f)],\sigma\rangle \quad\to\quad \langle E[e_f],\sigma\rangle$

$\langle E[\mathsf{guard}_j\ (\#\mathsf{t},v_p,v_v,e_f)],\sigma\rangle \quad\to\quad \langle E[\mathsf{install/p}_j\ (v_p,(v_v\ ()),e_f)],\sigma\rangle$

$\langle E[(\mathsf{install/p}_j\ (v_p,v_v,v_f)\ v)],\sigma\rangle \quad\to\quad \langle E[\mathsf{parameterize}\ v_p = v_v\ \mathsf{in}\ (v_f\ v)],\sigma\rangle$

$\langle E[?{}^{\ell}\mathsf{param/p}^{k}_{j}(c,v_p)],\sigma\rangle \quad\to\quad \langle E[{}^{\ell}\mathsf{mon}^{k}_{j}(c,?v_p)],\sigma\rangle$

$\langle E[\mathsf{parameterize}\ {}^{\ell}\mathsf{param/p}^{k}_{j}(c,v_p) = v\ \mathsf{in}\ e],\sigma\rangle \quad\to\quad \langle E[\mathsf{parameterize}\ v_p = {}^{k}\mathsf{mon}^{\ell}_{j}(c,v)\ \mathsf{in}\ e],\sigma\rangle$

$\langle E[{}^{\ell}\mathsf{param/p}^{k}_{j}(c,v_p) := v],\sigma\rangle \quad\to\quad \langle E[v_p := {}^{k}\mathsf{mon}^{\ell}_{j}(c,v)],\sigma\rangle$

$\langle E[\mathsf{check}^{k}_{j}(\#\mathsf{t},e)],\sigma\rangle \quad\to\quad \langle E[e],\sigma\rangle$

$\langle E[\mathsf{check}^{k}_{j}(\#\mathsf{f},v)],\sigma\rangle \quad\to\quad \langle \mathsf{error}^{k}_{j},\sigma\rangle$

$\mathsf{import}[\![k,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),m_1;\ldots;m_n;e]\!] =$
$\quad \mathsf{import}[\![k,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),m_1]\!];$
$\quad \ldots;$
$\quad \mathsf{import}[\![k,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),m_n]\!];$
$\quad \mathsf{import}[\![k,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),e]\!]$
$\mathsf{import}[\![k,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),\mathsf{module}\ \ell\ \mathsf{exports}\ x_{v_1}\ \mathsf{with}\ x_{c_1},\ldots\ \mathsf{where}\ y_1 = e_1,\ldots,y_n = e_n]\!] =$
$\quad \mathsf{module}\ \ell\ \mathsf{exports}\ x_{v_1}\ \mathsf{with}\ x_{c_1},\ldots\ \mathsf{where}\ y_1 = \{{}^{\ell\mathsf{mon}^{k}_{k}(c_i,v_i)}/_{x_i}\}e_1,\ldots,y_n = \{{}^{\ell\mathsf{mon}^{k}_{k}(c_i,v_i)}/_{x_i}\}e_n$
$\mathsf{import}[\![k,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),e]\!] \quad = \quad \{{}^{\ell_0\mathsf{mon}^{k}_{k}(c_i,v_i)}/_{x_i}\}e$

B.4 Annotated CtxPCF

B.4.1 Annotated CtxPCF Programs

$$
\begin{array}{rcl}
e & ::= & \ldots \mid {}^{\ell}\langle\langle^{\ell} x\rangle\rangle \mid |^{\ell} e| \mid {}^{\ell}\langle^{\ell} e\rangle \mid \lfloor e \rfloor^{\vec{\ell}} \\
v & ::= & \ldots \mid |^{\ell} v| \mid {}^{\ell}\langle^{\ell} v\rangle \\
c & ::= & \ldots \mid |^{\ell} \ldots |^{\ell'} \mathsf{flat/c}(v)| \ldots| \mid |^{\ell} \ldots |^{\ell'} \mathsf{param/c}(c)| \ldots| \mid |^{\ell} \ldots |^{\ell'} c : \tau \to (c)\ c| \ldots| \\
 & \mid & |^{\ell} \ldots |^{\ell'} c : \tau \to_{\mathsf{a}} (\lambda\, x : \tau.\ e)\ c| \ldots| \mid |^{\ell} \ldots |^{\ell'} \mathsf{ctx/c}(v,(v \Rightarrow v \leftarrow v) \ldots,v,(v \Rightarrow v \leftarrow v) \ldots)| \ldots| \\
 & \mid & \lfloor |^{\ell} \ldots |^{\ell'} \mathsf{flat/c}(v)| \ldots| \rfloor^{\vec{\ell}} \mid \lfloor |^{\ell} \ldots |^{\ell'} \mathsf{ctx/c}(v,(v \Rightarrow v \leftarrow v) \ldots,v,(v \Rightarrow v \leftarrow v) \ldots)| \ldots| \rfloor^{\vec{\ell}}
\end{array}
$$

B.4.2 Additional Typing Rules for Annotated CtxPCF Programs

$\boxed{\Gamma; \Sigma \vdash e : \tau}$

$$\frac{\Gamma; \Sigma \vdash x : \tau}{\Gamma; \Sigma \vdash {}^{\ell}\langle\langle^{k} x\rangle\rangle : \tau} \qquad \frac{\Gamma; \Sigma \vdash e : \tau}{\Gamma; \Sigma \vdash |^{\ell} e| : \tau} \qquad \frac{\Gamma; \Sigma \vdash e : \tau}{\Gamma; \Sigma \vdash {}^{\ell}\langle^{k} e\rangle : \tau} \qquad \frac{\Gamma; \Sigma \vdash e : \tau}{\Gamma; \Sigma \vdash \lfloor e \rfloor^{\vec{\ell}} : \tau}$$

B.4.3 Annotated Evaluation Contexts

Note We write $\|^{\ell} e\|$ to denote that the ownership annotations (if present) in e are consistent with ownership by ℓ. That is, either e has no ownership annotations or all ownership annotations in e have owner ℓ: $\|^{\ell} e\| = |^{\ell} \ldots \|^{\ell} e\| \ldots|$ where for all labels k and terms e', $e \neq \|^{k} e'\|$.

$$
\begin{array}{rcl}
E^{\ell} & ::= & F^{\ell} \\
E^{\ell_0} & ::= & F \\
F & ::= & [\cdot] \mid F\ e \mid v\ F \mid \mathsf{let}\, x = F\, \mathsf{in}\, e \mid \mathsf{if}\, F\, \mathsf{then}\, e\, \mathsf{else}\, e \mid F \oplus e \mid v \oplus F \\
 & \mid & F \leq e \mid v \leq F \mid \mathsf{make\text{-}parameter}\ F \mid ?F \mid F := e \mid \mathsf{p}(r) := F \\
 & \mid & \mathsf{parameterize}\, F = e\, \mathsf{in}\, e \mid \mathsf{parameterize}\, \|^{\ell''} \mathsf{p}(r)\| = F\, \mathsf{in}\, e \mid \mathsf{parameterize}\, \|^{\ell} \mathsf{p}(r)\| = v\, \mathsf{in}\, F \\
 & \mid & \mathsf{flat/c}(F) \mid \mathsf{param/c}(F) \mid F : \tau \to (e)\ e \mid c : \tau \to (F)\ e \mid c : \tau \to (v)\ F \\
 & \mid & F : \tau \to_{\mathsf{a}} (\lambda x : \tau.\ e)\ e \mid c : \tau \to_{\mathsf{a}} (\lambda x : \tau.\ e)\ F \\
 & \mid & \to_{\mathsf{ctx}} : \tau\, (F, (e \Rightarrow e \leftarrow e), \ldots, e, (e \Rightarrow e \leftarrow e), \ldots) \\
 & \mid & \to_{\mathsf{ctx}} : \tau\, (v, (v \Rightarrow v \leftarrow v), \ldots, (F \Rightarrow e \leftarrow e), (e \Rightarrow e \leftarrow e), \ldots, e, (e \Rightarrow e \leftarrow e), \ldots) \\
 & \mid & \to_{\mathsf{ctx}} : \tau\, (v, (v \Rightarrow v \leftarrow v), \ldots, (v \Rightarrow F \leftarrow e), (e \Rightarrow e \leftarrow e), \ldots, e, (e \Rightarrow e \leftarrow e), \ldots) \\
 & \mid & \to_{\mathsf{ctx}} : \tau\, (v, (v \Rightarrow v \leftarrow v), \ldots, (v \Rightarrow v \leftarrow F), (e \Rightarrow e \leftarrow e), \ldots, e, (e \Rightarrow e \leftarrow e), \ldots) \\
 & \mid & \to_{\mathsf{ctx}} : \tau\, (v, (v \Rightarrow v \leftarrow v), \ldots, F, (e \Rightarrow e \leftarrow e), \ldots) \\
 & \mid & \to_{\mathsf{ctx}} : \tau\, (v, (v \Rightarrow v \leftarrow v), \ldots, v, (v \Rightarrow v \leftarrow v), \ldots, (F \Rightarrow e \leftarrow e), (e \Rightarrow e \leftarrow e), \ldots) \\
 & \mid & \to_{\mathsf{ctx}} : \tau\, (v, (v \Rightarrow v \leftarrow v), \ldots, v, (v \Rightarrow v \leftarrow v), \ldots, (v \Rightarrow F \leftarrow e), (e \Rightarrow e \leftarrow e), \ldots) \\
 & \mid & \to_{\mathsf{ctx}} : \tau\, (v, (v \Rightarrow v \leftarrow v), \ldots, v, (v \Rightarrow v \leftarrow v), \ldots, (v \Rightarrow v \leftarrow F), (e \Rightarrow e \leftarrow e), \ldots) \\
 & \mid & \mathsf{install/p}_j\ (v, v, F) \mid \lfloor F \rfloor^{\vec{k}} \\
 & & \\
F^{\ell} & ::= & F^{\ell}\ e \mid v\ F^{\ell} \mid \mathsf{let}\, x = F^{\ell}\, \mathsf{in}\, e \mid \mathsf{if}\, E\, \mathsf{then}\, e\, \mathsf{else}\, e \mid F^{\ell} \oplus e \mid v \oplus F^{\ell} \\
 & \mid & F^{\ell} \leq e \mid v \leq F^{\ell} \mid \mathsf{make\text{-}parameter}\ F^{\ell} \mid ?F^{\ell} \mid {}^{j}B^{l}[F := e] \mid {}^{j}B^{l}[\mathsf{p}(r) := F] \\
 & \mid & {}^{j}B^{k}[F^{\ell} := e] \mid {}^{j}B^{k}[\mathsf{p}(r) := F^{\ell}] \mid \mathsf{parameterize}\, F^{\ell} = e\, \mathsf{in}\, e \mid {}^{j}B^{k}[\mathsf{parameterize}\, \|^{\ell''} \mathsf{p}(r)\| = F^{\ell}\, \mathsf{in}\, {}^{k'}B^{j'}[e]] \\
 & \mid & {}^{j}B^{k}[\mathsf{parameterize}\, \|^{\ell''} \mathsf{p}(r)\| = v\, \mathsf{in}\, {}^{k'}B^{j'}[F^{\ell}]] \mid {}^{j}B^{l}[\mathsf{parameterize}\, \|^{\ell''} \mathsf{p}(r)\| = F\, \mathsf{in}\, {}^{k'}B^{j'}[e]] \\
 & \mid & {}^{j}B^{k}[\mathsf{parameterize}\, \|^{\ell''} \mathsf{p}(r)\| = v\, \mathsf{in}\, {}^{k'}B^{l}[F]] \\
 & \mid & \mathsf{flat/c}(F^{\ell}) \mid \mathsf{param/c}(F^{\ell}) \mid F^{\ell} : \tau \to (e)\ e \mid c : \tau \to (F^{\ell})\ e \mid c : \tau \to (v)\ F^{\ell} \\
 & \mid & F^{\ell} : \tau \to_{\mathsf{a}} (\lambda x : \tau.\ e)\ e \mid c : \tau \to_{\mathsf{a}} (\lambda x : \tau.\ e)\ F^{\ell} \\
 & \mid & \to_{\mathsf{ctx}} : \tau\, (F^{\ell}, (e \Rightarrow e \leftarrow e), \ldots, e, (e \Rightarrow e \leftarrow e), \ldots) \\
 & \mid & \to_{\mathsf{ctx}} : \tau\, (v, (v \Rightarrow v \leftarrow v), \ldots, (F^{\ell} \Rightarrow e \leftarrow e), (e \Rightarrow e \leftarrow e), \ldots, e, (e \Rightarrow e \leftarrow e), \ldots) \\
 & \mid & \to_{\mathsf{ctx}} : \tau\, (v, (v \Rightarrow v \leftarrow v), \ldots, (v \Rightarrow F^{\ell} \leftarrow e), (e \Rightarrow e \leftarrow e), \ldots, e, (e \Rightarrow e \leftarrow e), \ldots) \\
 & \mid & \to_{\mathsf{ctx}} : \tau\, (v, (v \Rightarrow v \leftarrow v), \ldots, (v \Rightarrow v \leftarrow F^{\ell}), (e \Rightarrow e \leftarrow e), \ldots, e, (e \Rightarrow e \leftarrow e), \ldots) \\
 & \mid & \to_{\mathsf{ctx}} : \tau\, (v, (v \Rightarrow v \leftarrow v), \ldots, F^{\ell}, (e \Rightarrow e \leftarrow e), \ldots) \\
 & \mid & \to_{\mathsf{ctx}} : \tau\, (v, (v \Rightarrow v \leftarrow v), \ldots, v, (v \Rightarrow v \leftarrow v), \ldots, (F^{\ell} \Rightarrow e \leftarrow e), (e \Rightarrow e \leftarrow e), \ldots) \\
 & \mid & \to_{\mathsf{ctx}} : \tau\, (v, (v \Rightarrow v \leftarrow v), \ldots, v, (v \Rightarrow v \leftarrow v), \ldots, (v \Rightarrow F^{\ell} \leftarrow e), (e \Rightarrow e \leftarrow e), \ldots) \\
 & \mid & \to_{\mathsf{ctx}} : \tau\, (v, (v \Rightarrow v \leftarrow v), \ldots, v, (v \Rightarrow v \leftarrow v), \ldots, (v \Rightarrow v \leftarrow F^{\ell}), (e \Rightarrow e \leftarrow e), \ldots) \\
 & \mid & {}^{\ell'}\mathsf{mon}^{\ell}_{j}(c, F) \mid {}^{\ell'}\mathsf{mon}^{k}_{j}(c, F^{\ell}) \mid {}^{\ell'}\mathsf{mon}^{k}_{\ell}(F, e) \mid {}^{\ell'}\mathsf{mon}^{k}_{j}(F^{\ell}, e) \\
 & \mid & \mathsf{check}^{k}_{\ell}(F, e) \mid \mathsf{check}^{k}_{j}(F^{\ell}, e) \mid \mathsf{guard}_{\ell}\ (F, v, v, e) \mid \mathsf{guard}_{j}\ (F^{\ell}, v, v, e) \\
 & \mid & \mathsf{install/p}_{\ell}\ (v, F, v) \mid \mathsf{install/p}_{j}\ (v, F^{\ell}, v) \mid \mathsf{install/p}_{j}\ (v, v, F^{\ell}) \\
 & \mid & \|^{\ell} F\| \mid \|^{\ell'} F^{\ell}\| \mid \lfloor F^{\ell} \rfloor^{\vec{k}} \\
 & \mid & \mathsf{module}\, \ell\, \mathsf{exports}\, x\, \mathsf{with}\, x, \ldots\, \mathsf{where}\, x = v, \ldots, x = F, x = e, \ldots; p \\
 & \mid & \mathsf{module}\, \ell'\, \mathsf{exports}\, x\, \mathsf{with}\, x, \ldots\, \mathsf{where}\, x = v, \ldots, x = F^{\ell}, x = e, \ldots; p \\
{}^{\ell}B^{k} & ::= & {}^{\ell}\langle {}^{k}[\cdot] \rangle \\
{}^{\ell}B^{\ell} & ::= & {}^{\ell}\langle {}^{\ell}[\cdot] \rangle
\end{array}
$$

B.4.4 Annotated Reduction Semantics

$$\langle \mathsf{module}\ \ell\ \mathsf{exports}\ x_1\ \mathsf{with}\ x_{c_1},\ldots\ \mathsf{where}\ y_1 = v_1,\ldots,y_n = v_n,y_{e_1} = e_1,\ldots y_{e_m} = e_m;\ p,\sigma\rangle$$
$$\quad \to_{ann} \quad \langle \mathsf{module}\ \ell\ \mathsf{exports}\ x_1\ \mathsf{with}\ x_{c_1},\ldots\ \mathsf{where}\ y_1 = v_1,\ldots,y_i = v_i,y_{e_1} = \{^{v_n}/_{y_n}\}e_1,y_{e_2} = e_2,\ldots y_{e_m} = e_m;\ p,\sigma\rangle$$

$$\langle \mathsf{module}\ \ell\ \mathsf{exports}\ x_{v_1}\ \mathsf{with}\ x_{c_1},\ldots\ \mathsf{where}\ \ldots,x_{v_1} = v_1,\ldots,x_{c_1} = c_1,\ldots;\ p,\sigma\rangle$$
$$\to \quad \langle \mathsf{import}[\![\ell,x_{v_1},v_1,c_1,\ldots \mathsf{import}[\![\ell,x_{v_n},v_n,c_n,p]\!]]\!],\sigma\rangle$$

$$\langle E^\ell[\|^\ell \lambda\, x:\tau.\ e\|\ \|^\ell v\|],\sigma\rangle \to_{ann} \langle E^\ell[\{^{|^\ell v|}/_x\}e],\sigma\rangle$$
$$\langle E^\ell[\mu\, x:\tau.\ e],\sigma\rangle \to_{ann} \langle E^\ell[\{^{|^\ell \mu\, x:\tau.\ e|}/_x\}e],\sigma\rangle$$
$$\langle E^\ell[\mathsf{let}\ x = \|^\ell v\|\ \mathsf{in}\ e],\sigma\rangle \to_{ann} \langle E^\ell[\{^{|^\ell v|}/_x\}e],\sigma\rangle$$
$$\langle E^\ell[\mathsf{if}\ \|^\ell \#\mathsf{t}\|\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2],\sigma\rangle \to_{ann} \langle E^\ell[e_1],\sigma\rangle$$
$$\langle E^\ell[\mathsf{if}\ \|^\ell \#\mathsf{f}\|\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2],\sigma\rangle \to_{ann} \langle E^\ell[e_2],\sigma\rangle$$
$$\langle E^\ell[\|^\ell v_1\| \oplus \|^\ell v_2\|],\sigma\rangle \to_{ann} \langle E^\ell[|^\ell v|],\sigma\rangle \text{ where } v = v_1 \oplus v_2$$
$$\langle E^\ell[\|^\ell v_1\| \le \|^\ell v_2\|],\sigma\rangle \to_{ann} \langle E^\ell[|^\ell v|],\sigma\rangle \text{ where } v = v_1 \le v_2$$
$$\langle E^\ell[\mathsf{make\text{-}parameter}\ \|^\ell v\|],\sigma\rangle \to_{ann} \langle E^\ell[|^\ell \mathsf{p}(r)|],\sigma[r \mapsto v]\rangle \text{ where } r \text{ is fresh in } \sigma$$
$$\langle E^\ell[?\|^\ell \mathsf{p}(r)\|],\sigma\rangle \to_{ann} \langle E^\ell[|^\ell v|],\sigma\rangle$$
$$\text{where } \sigma(r) = v \text{ and } E \text{ does not contain } \mathsf{parameterize}^{\ell'} B^k[\mathsf{p}(r)] = v' \text{ in } {}^{k_2}B^{\ell'_2}[E']$$

$$\langle E^\ell[{}^\ell B^k[\mathsf{parameterize}\ \|^k \mathsf{p}(r)\| = \|^k v\|\ \mathsf{in}\ {}^k B^\ell[E'^k[?\|^k \mathsf{p}(r)\|]]]],\sigma\rangle$$
$$\to_{ann} \quad \langle E^\ell[{}^\ell B^k[\mathsf{parameterize}\ |^k \mathsf{p}(r)| = |^k v|\ \mathsf{in}\ {}^k B^\ell[E'^k[|^k v|]]]],\sigma\rangle$$
$$\text{where } E'^{\ell'} \text{ does not contain } {}^{\ell^*}B''^{k^*}[\mathsf{parameterize}\ \|^{k^*}\mathsf{p}(r)\| = v' \text{ in } {}^{k_2^*}B^{\ell_2^*}[E''^{\ell''}]]$$

$$\langle E^\ell[{}^\ell B^k[\|^k \mathsf{p}(r)\| := \|^k v\|]],\sigma\rangle \to_{ann} \langle E^\ell[|^\ell ()|],\sigma[r \mapsto v]\rangle$$
$$\text{where } E^\ell \text{ does not contain } {}^{\ell^*}B'^{k^*}[\mathsf{parameterize}\ \|^{k^*}\mathsf{p}(r)\| = v' \text{ in } {}^{k_2^*}B^{\ell_2^*}[E'^{\ell'}]]$$

$$\langle E^\ell[{}^\ell B^k[\mathsf{parameterize}\ \|^k \mathsf{p}(r)\| = \|^k v\|\ \mathsf{in}\ {}^k B^\ell[E'^{\ell'}[{}^{\ell'} B'^k[\|^k \mathsf{p}(r)\| := \|^k v'\|]]]]],\sigma\rangle$$
$$\to_{ann} \quad \langle E^\ell[{}^\ell B^k[\mathsf{parameterize}\ |^k \mathsf{p}(r)| = |^k v'|\ \mathsf{in}\ {}^k B^\ell[E'^{\ell'}[|^{\ell'}()|]]]],\sigma\rangle$$
$$\text{where } E'^{\ell'} \text{ does not contain } {}^{\ell^*}B''^{k^*}[\mathsf{parameterize}\ \|^{k^*}\mathsf{p}(r)\| = v'' \text{ in } {}^{k_2^*}B^{\ell_2^*}[E''^{\ell''}]]$$

$$\langle E^\ell[{}^\ell B^k[\mathsf{parameterize}\ \|^k \mathsf{p}(r)\| = \|^k v\|\ \mathsf{in}\ {}^k B'^\ell[v']]],\sigma\rangle$$
$$\to_{ann} \quad \langle E^\ell[|^\ell v'|],\sigma\rangle$$

$$\langle E^\ell[{}^\ell \mathsf{mon}^k_j(\lfloor\|^j \mathsf{flat/c}(v_c)\|\rfloor^{\vec{k}},\|^k v\|)],\sigma\rangle \to_{ann} \langle E^\ell[\mathsf{check}^k_j((v_c\ v),v)],\sigma\rangle$$

$$\langle E^\ell[{}^\ell \mathsf{mon}^k_j(\|^j \mathsf{param/c}(c)\|,v)],\sigma\rangle \to_{ann} \langle E^\ell[{}^\ell \mathsf{param/p}^k_j(c,v)],\sigma\rangle$$

$$\langle E^\ell[{}^\ell \mathsf{mon}^k_j(\|^j c_d : \tau \to (c_c)\ c_r\|,v)],\sigma\rangle \to_{ann} \langle E^\ell[{}^\ell \mathsf{mon}^k_j(c_c,\lambda\, x:\tau_d.\ {}^\ell \mathsf{mon}^k_j(c_r,v\ {}^k \mathsf{mon}^\ell_j(c_d,x)))],\sigma\rangle$$
$$\text{where } x \text{ is fresh}$$

$$\langle E^\ell[{}^\ell \mathsf{mon}^k_j(\|^j c_d : \tau \to_{\mathsf{a}} (\lambda\, x:\tau.\ e_c)\ c_r\|,v)],\sigma\rangle$$
$$\to_{ann} \quad \langle E^\ell[\lambda\, y:\tau_d.\ {}^\ell \mathsf{mon}^k_j(\{^{{}^k \mathsf{mon}^\ell_j(c_d,y)}/_x\}e_c,{}^\ell \mathsf{mon}^k_j(c_r,v\ {}^k \mathsf{mon}^\ell_j(c_d,y)))],\sigma\rangle \quad \text{where } y \text{ is fresh}$$

$$\langle E^\ell[{}^\ell \mathsf{mon}^k_j(\lfloor\|^j \mathsf{ctx/c}(v_c,(v_{c_{g_1}} \Rightarrow v_{c_{p_1}} \leftarrow v_{c_{v_1}}),\ldots(v_{c_{g_n}} \Rightarrow v_{c_{p_n}} \leftarrow v_{c_{v_n}}),v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\ldots)\|\rfloor^{\vec{\ell}},v)],\sigma\rangle$$
$$\to_{ann} \quad \langle E^\ell[\mathsf{check}^\ell_j((v_c\ ()),e)],\sigma\rangle$$
$$\text{where } e = \mathsf{guard}_j\ ((v_{c_{g_1}}\ ()),v_{c_{p_1}},v_{c_{v_1}},\ldots \mathsf{guard}_j\ ((v_{c_{g_n}}\ ()),v_{c_{p_n}},v_{c_{v_n}},{}^\ell \mathsf{ctx/p}^k_j\ (v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\ldots,v)))$$

$$\langle E^\ell[(\|^{\ell\,\ell}\mathsf{ctx/p}^k_j\ (v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\ldots(v_{a_{g_n}} \Rightarrow v_{a_{p_n}} \leftarrow v_{a_{v_n}}),v_f)\|\ v)],\sigma\rangle$$
$$\quad \to_{ann} \quad \langle E^\ell[(\mathsf{check}^\ell_j((v_a\ ()),e)\ v)],\sigma\rangle$$
$$\text{where } e = \mathsf{guard}_j\ ((v_{a_{g_1}}\ ()),v_{a_{p_1}},v_{a_{v_1}},\ldots\mathsf{guard}_j\ ((v_{a_{g_n}}\ ()),v_{a_{p_n}},v_{a_{v_n}},v_f))$$

$$\langle E^\ell[\mathsf{guard}_j\ (\|^k\#\mathsf{f}\|,v_p,v_v,e_f)],\sigma\rangle \quad \to_{ann} \quad \langle E^\ell[e_f],\sigma\rangle$$

$$\langle E^\ell[\mathsf{guard}_j\ (\|^k\#\mathsf{t}\|,v_p,v_v,e_f)],\sigma\rangle \quad \to_{ann} \quad \langle E^\ell[\mathsf{install/p}_j\ (v_p,(v_v\ ()),e_f)],\sigma\rangle$$

$$\langle E^\ell[(\|^\ell\mathsf{install/p}_j\ (v_p,v_v,v_f)\|\ v)],\sigma\rangle \quad \to_{ann} \quad \langle E^\ell[\langle^j\mathsf{parameterize}\ v_p = v_v\ \mathsf{in}\ ^j\langle^\ell(v_f\ v)\rangle\rangle],\sigma\rangle$$

$$\langle E^\ell[?\|^{\ell\,\ell}\mathsf{param/p}^k_j(c,v_p)\|],\sigma\rangle \quad \to_{ann} \quad \langle E^\ell[^\ell\mathsf{mon}^k_j(c,?v_p)],\sigma\rangle$$

$$\langle E^\ell[^\ell B^{\ell'}[\mathsf{parameterize}\ \|^{\ell'\,\ell'}\mathsf{param/p}^k_j(c,v_p)\| = v\ \mathsf{in}\ ^{\ell'}B'^\ell[e]]],\sigma\rangle$$
$$\quad \to_{ann} \quad \langle E^\ell[^\ell\langle^k\mathsf{parameterize}\ v_p = {}^k\mathsf{mon}^{\ell'}_j(c,v)\ \mathsf{in}\ ^k\langle^\ell e\rangle\rangle],\sigma\rangle$$

$$\langle E^\ell[^\ell B^{\ell'}[^{\ell'}\mathsf{param/p}^k_j(c,v_p) := v]],\sigma\rangle \quad \to_{ann} \quad \langle E^\ell[^\ell\langle^k v_p := {}^k\mathsf{mon}^{\ell'}_j(c,v)\rangle],\sigma\rangle$$

$$\langle E^\ell[\mathsf{check}^k_j(\|^j\#\mathsf{t}\|,e)],\sigma\rangle \quad \to_{ann} \quad \langle E^\ell[e],\sigma\rangle$$

$$\langle E^\ell[\mathsf{check}^k_j(\|^j\#\mathsf{f}\|,e)],\sigma\rangle \quad \to_{ann} \quad \langle \mathsf{error}^k_j,\sigma\rangle$$

$$\mathsf{import}[\![k,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),m_1;\ldots;m_n;e]\!] =$$
$$\quad \mathsf{import}[\![k,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),m_1]\!];$$
$$\quad \ldots;$$
$$\quad \mathsf{import}[\![k,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),m_n]\!];$$
$$\quad \mathsf{import}[\![k,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),e]\!]$$
$$\mathsf{import}[\![k,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),\mathsf{module}\ \ell\ \mathsf{exports}\ x_{v_1}\ \mathsf{with}\ x_{c_1},\ldots\ \mathsf{where}\ y_1 = e_1,\ldots,y_n = e_n]\!] =$$
$$\quad \mathsf{module}\ \ell\ \mathsf{exports}\ x_{v_1}\ \mathsf{with}\ x_{c_1},\ldots\ \mathsf{where}\ y_1 = \{^{\ell}\mathsf{mon}^k_k(\mathrm{obl}[\![c_i,\{k\},\{\ell\},k]\!],|^k v_i|)/_{\ell\langle\langle^k x_i\rangle\rangle}\}e_1,$$
$$\qquad\qquad\qquad\qquad \ldots,$$
$$\qquad\qquad\qquad\qquad y_n = \{^{\ell}\mathsf{mon}^k_k(\mathrm{obl}[\![c_i,\{k\},\{\ell\},k]\!],|^k v_i|)/_{\ell\langle\langle^k x_i\rangle\rangle}\}e_n$$
$$\mathsf{import}[\![k,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),e]\!] \quad = \quad \{^{\ell_0}\mathsf{mon}^k_k(\mathrm{obl}[\![c_i,\{k\},\{\ell_0\},k]\!],|^k v_i|)/_{\ell_0\langle\langle^k x_i\rangle\rangle}\}e$$

$$\mathrm{obl}[\![\mathsf{flat/c}(v),\vec{k},\vec{\ell},j]\!] = \lfloor\mathsf{flat/c}(v)\rfloor^{\vec{k}}$$
$$\mathrm{obl}[\![\mathsf{param/c}(c),\vec{k},\vec{\ell},j]\!] = \mathsf{param/c}(\mathrm{obl}[\![c,\vec{k}\vec{\ell},\vec{k}\vec{\ell},j]\!])$$
$$\mathrm{obl}[\![c_d : \tau \to (c_c)\ c_r,\vec{k},\vec{\ell},j]\!] = \mathrm{obl}[\![c_d,\vec{\ell},\vec{k},j]\!] : \tau \to (\mathrm{obl}[\![c_c,\vec{k},\vec{\ell},j]\!])\ \mathrm{obl}[\![c_r,\vec{k},\vec{\ell},j]\!]$$
$$\mathrm{obl}[\![c_d : \tau \to_{\mathsf{a}} (\lambda x : \tau.\ e)\ c_r,\vec{k},\vec{\ell},j]\!] = \mathrm{obl}[\![c_d,\vec{\ell}j,\vec{k},j]\!] : \tau \to_{\mathsf{a}} (\lambda x : \tau.\ \lfloor e\rfloor^{\vec{\ell}})\ \mathrm{obl}[\![c_r,\vec{k},\vec{\ell},j]\!]$$
$$\mathrm{obl}[\![\mathsf{ctx/c}(v,(v \Rightarrow v \leftarrow v)\ldots,v,(v \Rightarrow v \leftarrow v)\ldots),\vec{k},\vec{\ell},j]\!] = \lfloor\mathsf{ctx/c}(v,(v \Rightarrow v \leftarrow v)\ldots,v,(v \Rightarrow v \leftarrow v)\ldots)\rfloor^{\vec{\ell}}$$

B.4.5 Well-formed Annotated Programs

$\boxed{\ell_0; \Pi \Vdash p}$

$$\frac{\varnothing; \Pi \Vdash m_1 \rhd \Delta_1 \qquad \ldots \qquad \Delta_{n-1}; \Pi \Vdash m_n \rhd \Delta_n \qquad \Delta_n; \Pi; \ell_0 \Vdash e}{\ell_0; \Pi \Vdash m_1 \,;\, \ldots m_n \,; e}$$

$\boxed{\Delta; \Pi \Vdash m \rhd \Delta'}$

$$\frac{\varnothing; \Pi; \ell \Vdash e_1 \qquad \ldots \qquad \{y_1 : \ell, \ldots, y_{n-1} : \ell\}; \Pi; \ell \Vdash e_n \qquad \Gamma' = \{y_1 : \ell, \ldots, y_n : \ell\} \restriction_{\{x_1, \ldots, x_{n'}\}}}{\Delta; \Pi \Vdash \mathsf{module}\ \ell\ \mathsf{exports}\ x_1\ \mathsf{with}\ x_{c_1}, \ldots, x_{n'}\ \mathsf{with}\ x_{c_{n'}}\ \mathsf{where}\ y_1 = e_1, \ldots, y_n = e_n \rhd \Delta \uplus \Delta'}$$

$\boxed{\Delta; \Pi; \ell \Vdash e}$

$$\frac{}{\Delta; \Pi; \ell \Vdash ()} \qquad \frac{}{\Delta; \Pi; \ell \Vdash n} \qquad \frac{}{\Delta; \Pi; \ell \Vdash \#\mathsf{t}} \qquad \frac{}{\Delta; \Pi; \ell \Vdash \#\mathsf{f}} \qquad \frac{\Delta(x) = \ell}{\Delta; \Pi; \ell \Vdash x} \qquad \frac{\Delta; \Pi; \ell \Vdash e_i \qquad i \in \{1, 2\}}{\Delta; \Pi; \ell \Vdash e_1 \oplus e_2}$$

$$\frac{\Delta; \Pi; \ell \Vdash e_i \qquad i \in \{1, 2\}}{\Delta; \Pi; \ell \Vdash e_1 \leq e_2} \qquad \frac{\Delta; \Pi; \ell \Vdash e_1 \qquad \Delta[x \mapsto \ell]; \Pi; \ell \Vdash e_2}{\Delta; \Pi; \ell \Vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2} \qquad \frac{\Delta; \Pi; \ell \Vdash e_c \qquad \Delta; \Pi; \ell \Vdash e_i \qquad i \in \{1, 2\}}{\Delta; \Pi; \ell \Vdash \mathsf{if}\ e_c\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2}$$

$$\frac{\Delta[x \mapsto \ell]; \Pi; \ell \Vdash e}{\Delta; \Pi; \ell \Vdash \lambda\, x : \tau.\ e} \qquad \frac{\Delta[x \mapsto \ell]; \Pi; \ell \Vdash e}{\Delta; \Pi; \ell \Vdash \mu\, x : \tau.\ e} \qquad \frac{\Delta; \Pi; \ell \Vdash e_1 \qquad \Delta; \Pi; \ell \Vdash e_2}{\Delta; \Pi; \ell \Vdash e_1\ e_2} \qquad \frac{\Delta; \Pi; \ell \Vdash e}{\Delta; \Pi; \ell \Vdash ?e}$$

$$\frac{\Delta; \Pi; k \Vdash e_1 \qquad \Delta; \Pi; k \Vdash e_2}{\Delta; \Pi; \ell \Vdash {}^{\ell}B^{k}[e_1 := e_2]} \qquad \frac{\Delta; \Pi; \ell \Vdash e}{\Delta; \Pi; \ell \Vdash \mathsf{make\text{-}parameter}\ e} \qquad \frac{\Delta; \Pi; k \Vdash e_1 \qquad \Delta; \Pi; k \Vdash e_2 \qquad \Delta; \Pi; \ell \Vdash e_3}{\Delta; \Pi; \ell \Vdash {}^{\ell}B^{k}[\mathsf{parameterize}\ e_1 = e_2\ \mathsf{in}\ {}^{k}B^{\ell}[e_3]]}$$

$$\frac{\Pi(r) = \ell}{\Delta; \Pi; \ell \Vdash \mathsf{p}(r)} \qquad \frac{\Delta; \Pi; \ell \Vdash e}{\Delta; \Pi; \ell \Vdash \mathsf{flat/c}(e)} \qquad \frac{\Delta; \Pi; \ell \Vdash e}{\Delta; \Pi; \ell \Vdash \mathsf{param/c}(e)} \qquad \frac{\Delta; \Pi; \ell \Vdash e_1 \qquad \Delta; \Pi; \ell \Vdash e_2 \qquad \Delta; \Pi; \ell \Vdash e_3}{\Delta; \Pi; \ell \Vdash e_1 : \tau \to_{\mathsf{a}} (\lambda\, x : \tau.\ e_2)\ e_3}$$

$$\frac{\Delta; \Pi; \ell \Vdash e_1 \qquad \Delta; \Pi; \ell \Vdash e_2 \qquad \Delta; \Pi; \ell \Vdash e_3}{\Delta; \Pi; \ell \Vdash e_1 : \tau \to_{\mathsf{a}} (\lambda\, x : \tau.\ \lfloor e_2 \rfloor^{\vec{\ell}})\ e_3} \qquad \frac{\Delta; \Pi; \ell \Vdash e_1 \qquad \Delta; \Pi; \ell \Vdash e_2 \qquad \Delta; \Pi; \ell \Vdash e_3}{\Delta; \Pi; \ell \Vdash e_1 : \tau \to (e_2)\ e_3}$$

$$\frac{\begin{array}{c}\Delta; \Pi; \ell \Vdash e_1 \qquad \Delta; \Pi; \ell \Vdash e_2 \\ \Delta; \Pi; \ell \Vdash e_{g_{c_i}} \qquad \Delta; \Pi; \ell \Vdash e_{p_{c_i}} \qquad \Delta; \Pi; \ell \Vdash e_{v_{c_i}} \qquad \Delta; \Pi; \ell \Vdash e_{g_{a_i}} \qquad \Delta; \Pi; \ell \Vdash e_{p_{a_i}} \qquad \Delta; \Pi; \ell \Vdash e_{v_{a_i}}\end{array}}{\Delta; \Pi; \ell \Vdash \mathsf{ctx/c}(e_1, (e_{g_{c_1}} \Rightarrow e_{p_{c_1}} \leftarrow e_{v_{c_1}}), \ldots, e_2, (e_{g_{a_1}} \Rightarrow e_{p_{a_1}} \leftarrow e_{v_{a_1}}), \ldots)}$$

$$\frac{\Delta; \Pi; j \Vdash e_1 \qquad \Delta; \Pi; \ell \Vdash e_2}{\Delta; \Pi; \ell \Vdash \mathsf{check}^{k}_{j}(e_1, e_2)} \qquad \frac{\Delta; \Pi; j \Vdash e_g \qquad \Delta; \Pi; j \Vdash v_p \qquad \Delta; \Pi; j \Vdash v_v \qquad \Delta; \Pi; \ell \Vdash e_f}{\Delta; \Pi; \ell \Vdash \mathsf{guard}_j\ (e_g, v_p, v_v, e_f)}$$

$$\frac{\Delta;\Pi;\{k\};\{\ell\};j \rhd c \qquad \Delta;\Pi;k \Vdash v}{\Delta;\Pi;\ell \Vdash {}^{\ell}\mathsf{param/p}^{k}_{j}(c,v)} \qquad \frac{\Delta;\Pi;j \Vdash v_a \qquad \Delta;\Pi;j \Vdash v_{a_{g_1}} \qquad \Delta;\Pi;j \Vdash v_{a_{v_1}} \qquad \Delta;\Pi;l \Vdash v_f}{\Delta;\Pi;\ell \Vdash {}^{\ell}\mathsf{ctx/p}^{k}_{j}\,(v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\ldots(v_{a_{g_n}} \Rightarrow v_{a_{p_n}} \leftarrow v_{a_{v_n}}),v_f)}$$

$$\frac{\Delta;\Pi;j \Vdash v_p \qquad \Delta;\Pi;j \Vdash e_v \qquad \Delta;\Pi;\ell \Vdash e_f}{\Delta;\Pi;\ell \Vdash \mathsf{install/p}_j\,(v_p,e_v,e_f)} \qquad \frac{\Delta;\Pi;k \Vdash x}{\Delta;\Pi;\ell \Vdash {}^{\ell}\langle\langle^{k}x\rangle\rangle} \qquad \frac{\Delta;\Pi;\ell \Vdash e}{\Delta;\Pi;\ell \Vdash |^{\ell}e|}$$

$$\frac{c \neq \lfloor\|^{j}\mathsf{ctx/c}(e_1,(e_{g_{c_1}} \Rightarrow e_{p_{c_1}} \leftarrow e_{v_{c_1}}),\ldots,e_2,(e_{g_{a_1}} \Rightarrow e_{p_{a_1}} \leftarrow e_{v_{a_1}}),\ldots)\|\rfloor^{\vec{\ell'}} \qquad \Delta;\Pi;\{k\};\{\ell\};j \rhd c \qquad \Delta;\Pi;k \Vdash e}{\Delta;\Pi;\ell \Vdash {}^{\ell}\mathsf{mon}^{k}_{j}(c,e)}$$

$$\frac{c \equiv \lfloor\|^{j}\mathsf{ctx/c}(e_1,(e_{g_{c_1}} \Rightarrow e_{p_{c_1}} \leftarrow e_{v_{c_1}}),\ldots,e_2,(e_{g_{a_1}} \Rightarrow e_{p_{a_1}} \leftarrow e_{v_{a_1}}),\ldots)\|\rfloor^{\vec{\ell'}} \qquad \Delta;\Pi;\{k\};\{\ell\};j \rhd c \qquad \Delta;\Pi;\ell \Vdash e}{\Delta;\Pi;\ell \Vdash {}^{\ell}\mathsf{mon}^{k}_{j}(c,e)}$$

$$\frac{e_1 \neq \|^{j}c\| \qquad \Delta;\Pi;j \Vdash e_1 \qquad \Delta;\Pi;k \Vdash e_2}{\Delta;\Pi;\ell \Vdash {}^{\ell}\mathsf{mon}^{k}_{j}(\lfloor e_1\rfloor^{\vec{\ell}},e_2)} \qquad \frac{}{\Delta;\Pi;\ell \Vdash \mathsf{error}^{k}_{j}}$$

$\boxed{\Delta;\Pi \vdash \sigma}$

$$\frac{\mathrm{dom}(\Pi) = \mathrm{dom}(\sigma) \qquad \forall p \in \mathrm{dom}(\sigma), \Delta;\Pi;\Pi(p) \Vdash \sigma(p)}{\Delta;\Pi \Vdash \sigma}$$

$\boxed{\Delta;\Pi;\vec{k};\vec{\ell};j \rhd c}$

$$\frac{\Delta;\Pi;j \Vdash e \qquad \vec{k'} \subseteq \vec{k}}{\Delta;\Pi;\vec{k};\vec{\ell};j \rhd \lfloor\|^{j}\mathsf{flat/c}(e)\|\rfloor^{\vec{k'}}} \qquad \frac{\vec{i} = \vec{k} \cup \vec{\ell} \qquad \Delta;\Pi;\vec{i};\vec{i};j \rhd c}{\Delta;\Pi;\vec{k};\vec{\ell};j \rhd \|^{j}\mathsf{param/c}(\|^{j}c\|)\|}$$

$$\frac{\Delta;\Pi;\vec{\ell};\vec{k};j \rhd c_1 \qquad \Delta;\Pi;j \Vdash e_2 \qquad \Delta;\Pi;\vec{k};\vec{\ell};j \rhd c_3}{\Delta;\Pi;\vec{k};\vec{\ell};j \rhd \|^{j}\|^{j}c_1\| : \tau \to_{\mathsf{a}} (\lambda x : \tau.\ \lfloor e_2\rfloor^{\vec{\ell}})\ \|^{j}c_3\|\|}$$

$$\frac{\Delta;\Pi;\vec{\ell};\vec{k};j \rhd c_1 \qquad \Delta;\Pi;\vec{k};\vec{\ell};j \rhd c_2 \qquad \Delta;\Pi;\vec{k};\vec{\ell};j \rhd ctc_3}{\Delta;\Pi;\vec{k};\vec{\ell};j \rhd \|^{j}\|^{j}c_1\| : \tau \to (\|^{j}c_2\|)\ \|^{j}c_3\|\|}$$

$$\frac{\begin{array}{c}\Delta;\Pi;j \Vdash e_1 \qquad \Delta;\Pi;j \Vdash e_2 \\ \Delta;\Pi;\ell \Vdash e_{g_{c_i}} \qquad \Delta;\Pi;\ell \Vdash e_{p_{c_i}} \qquad \Delta;\Pi;\ell \Vdash e_{v_{c_i}} \qquad \Delta;\Pi;\ell \Vdash e_{g_{a_i}} \qquad \Delta;\Pi;\ell \Vdash e_{p_{a_i}} \qquad \Delta;\Pi;\ell \Vdash e_{v_{a_i}} \qquad \vec{\ell'} \subseteq \vec{\ell}\end{array}}{\Delta;\Pi;\vec{k};\vec{\ell};j \rhd \lfloor\|^{j}\mathsf{ctx/c}(e_1,(e_{g_{c_1}} \Rightarrow e_{p_{c_1}} \leftarrow e_{v_{c_1}}),\ldots,e_2,(e_{g_{a_1}} \Rightarrow e_{p_{a_1}} \leftarrow e_{v_{a_1}}),\ldots)\|\rfloor^{\vec{\ell'}}}$$

B.5 Metatheory of CtxPCF

Theorem 1 (Type Soundness for Surface CtxPCF)**.** *For all programs p of Surface CtxPCF such that $\vdash p : \tau$*
- $\langle p, \varnothing\rangle \rightarrow^* \langle v, \sigma\rangle$ *or;*
- $\langle p, \varnothing\rangle \rightarrow^* \langle \mathsf{error}^k_j, \sigma\rangle$ *or,*
- $\langle p, \varnothing\rangle \rightarrow^* \langle p', \sigma\rangle$ *and there exist p'' such that $p' \rightarrow p''$.*

Proof. Direct consequence of Lemmas 1 and 2. □

Lemma 1 (Type Progress for CtxPCF Terms)**.** *If $\Sigma \vdash p : \tau$, then either p is a value v, p is error^k_j, or for all σ such that $\varnothing; \Sigma \vdash \sigma$, there exist p' and σ' such that $\langle p, \sigma\rangle \rightarrow \langle p', \sigma'\rangle$.*

Proof. By straight-forward case analysis on p using Lemmas 3, 4, and 5. □

Lemma 2 (Type Preservation for CtxPCF Terms)**.** *If $\Sigma \vdash p : \tau$, $\varnothing; \Sigma \vdash \sigma$, and $\langle p, \sigma\rangle \rightarrow \langle p', \sigma'\rangle$, then either there exists $\Sigma' \supseteq \Sigma$ such that $\Sigma' \vdash p' : \tau$ and $\Gamma; \Sigma' \vdash \sigma'$, or $p' = \mathsf{error}^k_j$.*

Proof. By straight-forward case analysis on the rules of the reduction semantics using Lemma 3 and Lemma 4. □

Lemma 3 (Unique decomposition for CtxPCF)**.** *For all well-typed programs p such that $p \neq v$, $p \neq \mathsf{error}^k_j$, and $p \neq \mathsf{module}\,\ell\,\mathsf{exports}\,x_{v_1}\,\mathsf{with}\,x_{c_1},\ldots\,\mathsf{where}\,y_1 = e_1,\ldots,y_n = e_n,\ldots;\,p'$, there are unique e', and E such that $p = E[e']$ and e' is one of the following:*

1. $v_0\ v_1$
2. $\mu\, x : \tau.\ e_0$
3. $\mathsf{let}\, x = v_0\ \mathsf{in}\ e_1$
4. $\mathsf{if}\ v_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$
5. $v_0 \oplus v_1$
6. $v_0 \leq v_1$
7. $\mathsf{make\text{-}parameter}\ v_0$
8. $\mathsf{parameterize}\, v_0 = v_1\ \mathsf{in}\ e_2$ *where* $v_0 \neq \mathsf{p}(r)$
9. $\mathsf{parameterize}\, \mathsf{p}(r) = v_1\ \mathsf{in}\ v_2$
10. $?v_0$
11. $v_0 := v_1$
12. $^{\ell}\mathsf{mon}^k_j(c_0, v_1)$
13. $\mathsf{check}^k_j(v_0, e_1)$
14. $\mathsf{guard}_j\ (v_g, v_p, v_v, e_f)$

Proof. By induction on the size of p. □

Lemma 4 (Well-typed evaluation context plugs for CtxPCF)**.** *If $\Sigma \vdash E[e] : \tau$, $\varnothing; \Pi \vdash e : \tau'$, and $\varnothing; \Pi \vdash e' : \tau'$ then $\Sigma \vdash E[e'] : \tau$.*

Proof. Induction on E. □

Lemma 5 (Type focusing for CtxPCF)**.** *If $\Sigma \vdash E[e] : \tau$ then $\varnothing; \Sigma \vdash e : \tau'$ for some τ'.*

Proof. Induction on E. □

Theorem 2 (Type Soundness for Annotated Surface CtxPCF)**.** *For all programs p of Annotated Surface CtxPCF such that $\vdash p : \tau$*
- $\langle p, \varnothing\rangle \rightarrow^*_{ann} \langle v, \sigma\rangle$ *or;*
- $\langle p, \varnothing\rangle \rightarrow^*_{ann} \langle \mathsf{error}^k_j, \sigma\rangle$ *or,*
- $\langle p, \varnothing\rangle \rightarrow^*_{ann} \langle p', \sigma\rangle$ *and there exist p'' such that $p' \rightarrow p''$.*

Proof. Direct consequence of Lemmas 6 and 7. □

Lemma 6 (Type Progress for Annotated CtxPCF Terms)**.** *If $\Sigma \vdash p : \tau$, then either p is a value v, p is error^k_j, or for all σ such that $\varnothing; \Sigma \vdash \sigma$, there exist p' and σ' such that $\langle p, \sigma\rangle \rightarrow \langle p', \sigma'\rangle$.*

Proof. By straight-forward case analysis on p using Lemmas 8, 9, and 10. □

Lemma 7 (Type Preservation for Annotated CtxPCF Terms)**.** *If* $\Sigma \vdash p : \tau$, $\varnothing; \Sigma \vdash \sigma$, *and* $\langle p, \sigma\rangle \rightarrow_{ann} \langle p', \sigma'\rangle$, *then either there exists* $\Sigma' \supseteq \Sigma$ *such that* $\Sigma' \vdash p' : \tau$ *and* $\varnothing; \Sigma' \vdash \sigma'$, *or* $p' = \mathsf{error}^k_j$.

Proof. By straight-forward case analysis on the rules of the reduction semantics using Lemma 8 and Lemma 9. □

Lemma 8 (Unique decomposition for Annotated CtxPCF)**.** *For all well-typed programs* p *such that* $p \neq v$, $p \neq \mathsf{error}^k_j$, *and* $p \neq \mathsf{module}\,\ell\,\mathsf{exports}\,x_{v_1}\,\mathsf{with}\,x_{c_1},\ldots\,\mathsf{where}\,y_1 = e_1,\ldots,y_n = e_n,\ldots; p'$, *there are unique* e', ℓ', *and* $E^{\ell'}$ *such that* $p = E^{\ell'}[e']$ *and* e' *is one of the following:*

1. $v_0\ v_1$
2. $\mu\, x : \tau.\ e_0$
3. $\mathsf{let}\, x = v_0\ \mathsf{in}\ e_1$
4. $\mathsf{if}\ v_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$
5. $v_0 \oplus v_1$
6. $v_0 \leq v_1$
7. $\mathsf{make\text{-}parameter}\ v_0$
8. ${}^{\ell'}B^{\ell''}[\mathsf{parameterize}\, v_0 = v_1\ \mathsf{in}^{\,j'} B'^{j}[e_2]]$ *where* $v_0 \neq \|^{\ell^*}\mathsf{p}(r)\|$
9. ${}^{\ell'}B^{\ell''}[\mathsf{parameterize}\, \|^{\ell^*}\mathsf{p}(r)\| = v_1\ \mathsf{in}^{\,j'} B'^{j}[v_2]]$
10. $?v_0$
11. ${}^{\ell'}B^{\ell''}[v_0 := v_1]$
12. ${}^{\ell}\mathsf{mon}^k_j(c_0, v_1)$
13. $\mathsf{check}^k_j(v_0, e_1)$
14. $\mathsf{guard}_j\,(v_g, v_p, v_v, e_f)$

Proof. By induction on the size of p. □

Lemma 9 (Well-typed evaluation context plugs for Annotated CtxPCF)**.** *If* $\Sigma \vdash E^k[e] : \tau$, $\varnothing; \Pi \vdash e : \tau'$, *and* $\varnothing; \Pi \vdash e' : \tau'$ *then* $\Sigma \vdash E^k[e'] : \tau$.

Proof. Induction on E^k. □

Lemma 10 (Type focusing for Annotated CtxPCF)**.** *If* $\Sigma \vdash E^k[e] : \tau$ *then* $\varnothing; \Sigma \vdash e : \tau'$ *for some* τ'.

Proof. Induction on E^k. □

Theorem 3 (Annotation transparency)**.** *The following statements hold for Surface CtxPCF:*

1. *Let* e *be a well-typed and well-formed Annotated Surface CtxPCF program:* $\vdash p : \tau$ *and* $\ell_0 \Vdash p$. *Let* $\bar{p}$ *be the Surface CtxPCF expression that is like* p *but without annotations. If* $\langle p, \varnothing\rangle \rightarrow^*_{ann} \langle p', \sigma\rangle$, *then* $\langle \bar{p}, \varnothing\rangle \rightarrow^*_{ann} \langle \bar{p'}, \bar{\sigma}\rangle$, *where* $\bar{p'}$ *is the CtxPCF program that is like* p' *but without annotations.*
2. *Let* $\bar{p}$ *be a well-typed Surface CtxPCF program:* $\vdash \bar{p} : \tau$. *Then there exists some Annotated Surface CtxPCF program* p *such that* $\vdash p : \tau$ *and* $\ell_0 \Vdash p$. *Furthermore, if* $\langle \bar{p}, \varnothing\rangle \rightarrow^*_{ann} \langle \bar{p'}, \bar{\sigma}\rangle$, *then* $\langle p, \varnothing\rangle \rightarrow^*_{ann} \langle p', \sigma\rangle$ *for some* p', *where* $\bar{p'}$ *is the CtxPCF program that is like* p' *but without annotations.*

Proof. By a straightforward lock-step bi-simulation between the two reduction steps using the obvious annotation erasure function to relate the corresponding configurations in each step. For the second point of the Theorem, we construct p from $\bar{p}$ by adding to each imported variable in a term of $\bar{p}$ the appropriate annotation. □

Definition 1 (Complete monitoring)**.** *CtxPCF is a complete monitor if any only if for all* p *such that* $\vdash p_0 : \tau$ *and* $\ell_0 \Vdash p$,

- $\langle p_0, \varnothing\rangle \rightarrow^*_{ann} \langle v, \sigma\rangle$, *or*
- *for all* p_1 *and* σ_1 *such that* $\langle p_0, \varnothing\rangle \rightarrow^*_{ann} \langle p_1, \sigma_1\rangle$ *there exists* p_2 *and* σ_2 *such that* $\langle p_1, \sigma_1\rangle \rightarrow_{ann} \langle p_2, \sigma_2\rangle$, *or*
- $\langle p_0, \varnothing\rangle \rightarrow_{ann} \langle p_1, \sigma_1\rangle \rightarrow^*_{ann} \langle \mathsf{error}^k_j, \sigma_2\rangle$ *and* p_1 *is of the form* $E^{\ell}[{}^{\ell}\mathsf{mon}^k_j(\lfloor \mathsf{flat/c}(p)\rfloor^{\vec{\ell}}, v)]$ *and for all such terms* p_1, $v = \|^k v'\|$ *and* $k \in \vec{\ell}$.
- $\langle p_0, \varnothing\rangle \rightarrow_{ann} \langle p_1, \sigma_1\rangle \rightarrow^*_{ann} \langle \mathsf{error}^{\ell}_j, \sigma_2\rangle$ *and* p_1 *is of the form*

$$E^{\ell}[{}^{\ell}\mathsf{mon}^k_j(\lfloor \mathsf{ctx/c}(v_c, (v_{g_{c_1}} \Rightarrow v_{p_{c_1}} \leftarrow v_{v_{c_1}}), \ldots, v_a, (v_{g_{a_1}} \Rightarrow v_{p_{a_1}} \leftarrow v_{v_{a_1}}), \ldots)\rfloor^{\vec{\ell}}, v)]$$

and for all such terms p_1, $\ell \in \vec{\ell}$.

Theorem 4. *CtxPCF is a complete monitor.*

Proof. As a direct consequence of Lemmas 6, 7, 11, and 12, we have that for all programs p_0 such that $\vdash p_0 : \tau$ and $\ell_0 \Vdash p_0$, either

- $\langle p_0, \varnothing\rangle \to^*_{ann} \langle v, \sigma\rangle$, or
- for all p_1 and σ_1 such that $\langle p_0, \varnothing\rangle \to^*_{ann} \langle p_1, \sigma_1\rangle$ there exists p_2 and σ_2 such that $\langle p_1, \sigma_1\rangle \to_{ann} \langle p_2, \sigma_2\rangle$, or
- $\langle p_0, \varnothing\rangle \to^*_{ann} \langle \mathsf{error}^k_j, \sigma_2\rangle$.

In the last case, since $\ell_0 \Vdash p_0$, we know that error^k_j does not appear in p_0. Therefore it must have been introduced by a reduction. In particular, it must have been the result of the reduction $\langle E^\ell[\mathsf{check}^k_j(\|^j\#\mathsf{f}\|,e)], \sigma_2\rangle \to_{ann} \langle \mathsf{error}^k_j, \sigma_2\rangle$. Again, since $\ell_0 \Vdash p_0$, $\mathsf{check}^k_j(\|^j\#\mathsf{f}\|,e)$ must not occur in p_0. Hence it must be the result of a reduction. There are three rules that introduce checks:

1. $\langle E^\ell[{}^\ell\mathsf{mon}^k_j(\lfloor\|^j\mathsf{flat/c}(v_c)\|\rfloor^{\vec{k}},\|^k v\|)], \sigma\rangle \to_{ann} \langle E^\ell[\mathsf{check}^k_j((v_c\ v),v)], \sigma\rangle$: In this case, we can deduce

$$\begin{aligned}\langle p_0, \varnothing\rangle &\to^*_{ann} \langle E^\ell[{}^\ell\mathsf{mon}^k_j(\lfloor\|^j\mathsf{flat/c}(v_c)\|\rfloor^{\vec{k}},\|^k v\|)], \sigma_1\rangle\\ &\to_{ann} \langle E^\ell[\mathsf{check}^k_j((v_c\ v),v)], \sigma_1\rangle\\ &\to^*_{ann} \langle E'^\ell[\mathsf{check}^k_j(\|^j\#\mathsf{f}\|,e)], \sigma_2\rangle\\ &\to_{ann} \langle \mathsf{error}^k_j, \sigma_2\rangle\end{aligned}$$

   By Lemma 12, we can deduce that for some Π such that $\varnothing; \Pi \Vdash \sigma_1$, $\ell_0; \Pi \Vdash E^\ell[{}^\ell\mathsf{mon}^k_j(\lfloor\|^j\mathsf{flat/c}(v_c)\|\rfloor^{\vec{k}},\|^k v\|)]$. By Lemma 13, we derive $\varnothing; \Pi; \ell \Vdash {}^\ell\mathsf{mon}^k_j(\lfloor\|^j\mathsf{flat/c}(v_c)\|\rfloor^{\vec{k}},\|^k v\|)$, and thus by well-formedness that $k \in \vec{k}$.

2. $\langle E^\ell[{}^\ell\mathsf{mon}^k_j(\lfloor\|^j c\|\rfloor^{\vec{\ell}},v)], \sigma\rangle \to_{ann} \langle E^\ell[\mathsf{check}^\ell_j((v_c\ ()),e)], \sigma\rangle$ where
   $c = \mathsf{ctx/c}(v_c,(v_{c_{g_1}} \Rightarrow v_{c_{p_1}} \leftarrow v_{c_{v_1}}),\dots(v_{c_{g_n}} \Rightarrow v_{c_{p_n}} \leftarrow v_{c_{v_n}}),v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\dots)$ and
   $e = \mathsf{guard}_j\ ((v_{c_{g_1}}\ ()),v_{c_{p_1}},v_{c_{v_1}},\dots\mathsf{guard}_j\ ((v_{c_{g_n}}\ ()),v_{c_{p_n}},v_{c_{v_n}},{}^\ell\mathsf{ctx/p}^k_j\ (v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\dots,v)))$: In this case, we can deduce

$$\begin{aligned}\langle p_0, \varnothing\rangle &\to^*_{ann} \langle E^\ell[{}^\ell\mathsf{mon}^k_j(\lfloor\|^j c\|\rfloor^{\vec{\ell}},v)], \sigma_1\rangle\\ &\to_{ann} \langle E^\ell[\mathsf{check}^\ell_j((v_c\ ()),e)], \sigma_1\rangle\\ &\to^*_{ann} \langle E'^\ell[\mathsf{check}^\ell_j(\|^j\#\mathsf{f}\|,e)], \sigma_2\rangle\\ &\to_{ann} \langle \mathsf{error}^\ell_j, \sigma_2\rangle\end{aligned}$$

   By Lemma 12, we can deduce that for some Π such that $\varnothing; \Pi \Vdash \sigma_1$, $\ell_0; \Pi \Vdash E^\ell[{}^\ell\mathsf{mon}^k_j(\lfloor\|^j c\|\rfloor^{\vec{\ell}},v)]$. By Lemma 13, we derive $\varnothing; \Pi; \ell \Vdash {}^\ell\mathsf{mon}^k_j(\lfloor\|^j c\|\rfloor^{\vec{\ell}},v)$, and thus by well-formedness that $\ell \in \vec{\ell}$.

3. $\langle E^\ell[(\|^{\ell\ell}\mathsf{ctx/p}^k_j\ (v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\dots(v_{a_{g_n}} \Rightarrow v_{a_{p_n}} \leftarrow v_{a_{v_n}}),v_f)\|\ v)], \sigma\rangle \to_{ann}$
   $\langle E^\ell[(\mathsf{check}^k_j((v_a\ ()),e)\ v)], \sigma\rangle$ where $e = \mathsf{guard}_j\ ((v_{a_{g_1}}\ ()),v_{a_{p_1}},v_{a_{v_1}},\dots\mathsf{guard}_j\ ((v_{a_{g_n}}\ ()),v_{a_{p_n}},v_{a_{v_n}},v_f))$: Again, since $\ell_0 \Vdash p_0$, p_0 does not contain ${}^\ell\mathsf{ctx/p}^k_j\ (v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\dots(v_{a_{g_n}} \Rightarrow v_{a_{p_n}} \leftarrow v_{a_{v_n}}),v_f)$. This it must have been introduced by a reduction step $\langle E^\ell[{}^\ell\mathsf{mon}^k_j(\lfloor\|^j c\|\rfloor^{\vec{\ell}},v_f)], \sigma\rangle \to_{ann} \langle E^\ell[\mathsf{check}^\ell_j((v_c\ ()),e)], \sigma\rangle$ where
   $c = \mathsf{ctx/c}(v_c,(v_{c_{g_1}} \Rightarrow v_{c_{p_1}} \leftarrow v_{c_{v_1}}),\dots(v_{c_{g_n}} \Rightarrow v_{c_{p_n}} \leftarrow v_{c_{v_n}}),v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\dots)$ and
   $e = \mathsf{guard}_j\ ((v_{c_{g_1}}\ ()),v_{c_{p_1}},v_{c_{v_1}},\dots\mathsf{guard}_j\ ((v_{c_{g_n}}\ ()),v_{c_{p_n}},v_{c_{v_n}},{}^\ell\mathsf{ctx/p}^k_j\ (v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\dots,v_f)))$. In this case, we can deduce a longer series of reductions

$$\begin{aligned}\langle p_0, \varnothing\rangle &\to^*_{ann} \langle E^\ell[{}^\ell\mathsf{mon}^k_j(\lfloor\|^j c\|\rfloor^{\vec{\ell}},v_f)], \sigma_1\rangle\\ &\to_{ann} \langle E^\ell[\mathsf{check}^\ell_j((v_c\ ()),e)], \sigma_1\rangle\\ &\to^*_{ann} \langle E'^\ell[(\|^{\ell\ell}\mathsf{ctx/p}^k_j\ (v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\dots(v_{a_{g_n}} \Rightarrow v_{a_{p_n}} \leftarrow v_{a_{v_n}}),v_f)\|\ v)], \sigma_1\rangle\\ &\to_{ann} \langle E'^\ell[(\mathsf{check}^k_j((v_a\ ()),e)\ v)], \sigma_1\rangle\\ &\to^*_{ann} \langle E''^\ell[\mathsf{check}^\ell_j(\|^j\#\mathsf{f}\|,e)], \sigma_2\rangle\\ &\to_{ann} \langle \mathsf{error}^\ell_j, \sigma_2\rangle\end{aligned}$$

   By Lemma 12, we can deduce that for some Π such that $\varnothing; \Pi \Vdash \sigma_1$, $\ell_0; \Pi \Vdash E^\ell[{}^\ell\mathsf{mon}^k_j(\lfloor\|^j c\|\rfloor^{\vec{\ell}},v_f)]$. By Lemma 13, we derive $\varnothing; \Pi; \ell \Vdash {}^\ell\mathsf{mon}^k_j(\lfloor\|^j c\|\rfloor^{\vec{\ell}},v)$, and thus by well-formedness that $\ell \in \vec{\ell}$.

Combining these three cases with the results above suffice to show that CtxPCF is a complete monitor. □

Lemma 11 (Progress)**.** *For all programs p, stores σ, store typings Σ, and store labelings Π such that*
- $\Sigma \vdash p : \tau$,
- $\varnothing; \Sigma \vdash \sigma$, *and*
- $\ell_0; \Pi \Vdash p$,

then either
- $p = v$,
- $p = \mathsf{error}^k_j$, *or*
- $\langle p, \sigma\rangle \to_{ann} \langle p', \sigma'\rangle$.

Proof. From Lemma 8 we know that there either $p = v$, $p = \mathsf{error}^k_j$, $p = \mathsf{module}\,\ell\,\mathsf{exports}\,x_{v_1}\,\mathsf{with}\,x_{c_1},\ldots\,\mathsf{where}\,y_1 = e_1,\ldots,y_n = e_n,\ldots;\,p'$, or there exist e^*, ℓ^*, and E^{ℓ^*} such that $p = E^{\ell^*}[e^*]$. The first two cases are immediate. If $p = \mathsf{module}\,\ell\,\mathsf{exports}\,x_{v_1}\,\mathsf{with}\,x_{c_1},\ldots\,\mathsf{where}\,y_1 = e_1,\ldots,y_n = e_n,\ldots;\,p'$, then either p is one of the following:

1. $\mathsf{module}\,\ell\,\mathsf{exports}\,x_{v_1}\,\mathsf{with}\,x_{c_1},\ldots\,\mathsf{where}\,\ldots,x_{v_1} = v_1,\ldots,x_{c_1} = c_1,\ldots;\,p'$: Then by the reduction relation we have $\langle p, \sigma\rangle \to_{ann} \langle \mathsf{import}[\![\ell,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),p']\!], \sigma\rangle$, since meta function import is total.
2. $\mathsf{module}\,\ell\,\mathsf{exports}\,x_1\,\mathsf{with}\,x_{c_1},\ldots\,\mathsf{where}\,y_1 = v_1,\ldots,y_n = v_n, y_{e_1} = e_1,\ldots y_{e_m} = e_m;\,p'$: Then by the reduction relation we have $\langle \mathsf{module}\,\ell\,\mathsf{exports}\,x_1\,\mathsf{with}\,x_{c_1},\ldots\,\mathsf{where}\,y_1 = v_1,\ldots,y_n = v_n, y_{e_1} = e_1,\ldots y_{e_m} = e_m;\,p', \sigma\rangle \to_{ann}$ $\langle \mathsf{module}\,\ell\,\mathsf{exports}\,x_1\,\mathsf{with}\,x_{c_1},\ldots\,\mathsf{where}\,y_1 = v_1,\ldots,y_n = v_n, y_{e_1} = \{^{v_i}/_{y_i}\}e_1, y_{e_2} = e_2,\ldots y_{e_m} = e_m;\,p', \sigma\rangle$.

In the final case, using Lemma 13 and $\ell_0; \Pi \Vdash p$ we derive $\varnothing; \Pi; \ell^* \Vdash e^*$. We proceed by case analysis on e^*:

1. $v_0\ v_1$: By assumption, we have $\Sigma \vdash E^{\ell^*}[v_0\ v_1] : \tau$ and thus by Lemma 10 that $\varnothing; \Sigma \vdash v_0\ v_1 : \tau_r$ for some τ_r. By the typing relation it must be the case that $\varnothing; \Sigma \vdash v_0 : \tau_d \to \tau_r$ and $\varnothing; \Sigma \vdash v_1 : \tau_d$ for some τ_d. Since v_0 is a value with type $\tau_d \to \tau_r$ and is well-formed (by further applications of Lemma 13) one of the following must be the case:
   - $v_0 = \|^k \lambda\, x : \tau_d.\ e_0\|$ for some x, e_0, and k. Similarly, v_1 can be decomposed such that $v_1 = \|^j v_1'\|$ for some v_1' and j. Since $\varnothing; \Pi; \ell^* \Vdash \|^k \lambda\, x : \tau_d.\ e_0\|\ \|^j v_1'\|$, we can deduce that $j = k = \ell^*$. Thus according to the reduction relation and Lemma 14 we have that $\langle p, \sigma\rangle \to_{ann} \langle E^{\ell^*}[\{^{|^{\ell^*} v_1'|}/_x\}e_0], \sigma\rangle$.
   - $v_0 = \|^{k\ell}\mathsf{ctx/p}^{k'}_j\ (v_a, (v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}), \ldots (v_{a_{g_n}} \Rightarrow v_{a_{p_n}} \leftarrow v_{a_{v_n}}), v_f)\|$. Since $\varnothing; \Pi; \ell^* \Vdash \|^{k\ell}\mathsf{ctx/p}^{k'}_j\ (v_a, (v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}), \ldots (v_{a_{g_n}} \Rightarrow v_{a_{p_n}} \leftarrow v_{a_{v_n}}), v_f)\|\ v_1$, we can deduce that $k = \ell = \ell^*$. Thus according to the reduction relation and Lemma 14 we have that

     $$\langle p, \sigma\rangle \to_{ann} \langle E^{\ell^*}[(\mathsf{check}^{k'}_j\ ((v_a\ ()), e')\ v)], \sigma\rangle$$

     where $e' = \mathsf{guard}_j\ ((v_{a_{g_1}}\ ()), v_{a_{p_1}}, v_{a_{v_1}}, \ldots \mathsf{guard}_j\ ((v_{a_{g_n}}\ ()), v_{a_{p_n}}, v_{a_{v_n}}, v_f))$.
   - $v_0 = \|^k \mathsf{install/p}_j\ (v_p, v_v, v_f)\|$. Since $\varnothing; \Pi; \ell^* \Vdash \|^k \mathsf{install/p}_j\ (v_p, v_v, v_f)\|\ v_1$, we can deduce that $k = \ell^*$. Therefore by the reduction relation and Lemma 14 we have that

     $$\langle p, \sigma\rangle \to \langle E^{\ell^*}[\langle^{\ell^*}\langle^j \mathsf{parameterize}\ v_p = v_v\ \mathsf{in}\ ^j\langle^{\ell^*}(v_f\ v_1)\rangle\rangle], \sigma\rangle.$$

2. $\mu\, x : \tau.\ e_0$: According to the reduction relation and Lemma 14, we have $\langle p, \sigma\rangle \to_{ann} \langle E^{\ell^*}[\{^{|^{\ell^*} \mu\, x:\tau.\ e_0|}/_x\}e_0], \sigma\rangle$.

3. $\mathsf{let}\, x = v_0\, \mathsf{in}\, e_1$: Since $\varnothing; \Pi; \ell^* \Vdash \mathsf{let}\, x = v_0\, \mathsf{in}\, e_1$, it must be the case that $\varnothing; \Pi; \ell^* \Vdash v_0$, so $v_0 = \|^{\ell^*} v_0'\|$. Thus by the reduction relation and Lemma 14, we have $\langle p, \sigma\rangle \to_{ann} \langle E^{\ell^*}[\{^{|^{\ell^*} v_0'|}/_x\}e_1], \sigma\rangle$.

4. $\mathsf{if}\ v_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$: Since $\varnothing; \Pi; \ell^* \Vdash \mathsf{if}\ v_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$, it must be the case that $\varnothing; \Pi; \ell^* \Vdash v_0$, so $v_0 = \|^{\ell^*} v_0'\|$. By assumption, we have $\Sigma \vdash E^{\ell^*}[\mathsf{if}\ v_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2] : \tau$ and thus by Lemma 10 that $\varnothing; \Sigma \vdash \mathsf{if}\ v_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau'$ for some τ'. By the typing relation it must be the case that $\varnothing; \Sigma \vdash v_0 : \mathsf{Bool}$. Since $v_0 = \|^{\ell^*} v_0'\|$ is a value with type Bool, it must be the case that either $v_0 = \|^{\ell^*} \#\mathsf{t}\|$ or $v_0 = \|^{\ell^*} \#\mathsf{f}\|$. Thus by the reduction relation and Lemma 14, we have either $\langle p, \sigma\rangle \to_{ann} \langle E^{\ell^*}[e_0], \sigma\rangle$ or $\langle p, \sigma\rangle \to_{ann} \langle E^{\ell^*}[e_1], \sigma\rangle$.

5. $v_0 \oplus v_1$: By assumption, we have $\Sigma \vdash E^{\ell^*}[v_0 \oplus v_1] : \tau$. By Lemma 10 we have $\varnothing; \Sigma \vdash v_0 \oplus v_1 : \mathsf{Int}$ and thus that $v_0 = \|^k n_0\|$ and $v_1 = \|^j n_1\|$. Since $\varnothing; \Pi; \ell^* \Vdash \|^k n_0\| \oplus \|^j n_1\|$, we can deduce by well-formedness that $k = j = \ell^*$. Thus according to the reduction relation and Lemma 14 we have that $\langle p, \sigma\rangle \to_{ann} \langle E^{\ell^*}[\|^{\ell^*} n\|], \sigma\rangle$ where $n = n_1 \oplus n_2$.

6. $v_0 \leq v_1$: Similar to the case $v_0 \oplus v_1$.

7. make-parameter v_0: Since $\varnothing; \Pi; \ell^* \Vdash$ make-parameter v_0, it must be the case that $\varnothing; \Pi; \ell^* \Vdash v_0$ and thus $v_0 = \|^{\ell^*} v_0' \|$ for some ℓ^*. Then by the reduction relation and Lemma 14 we have $\langle p, \sigma \rangle \rightarrow_{ann} \langle E^{\ell^*}[|^{\ell^*} \mathsf{p}(r)|], \sigma[r \mapsto v_0'] \rangle$, where p is fresh.

8. ${}^{\ell}B^{k}[\mathsf{parameterize}\, v_0 = v_1 \,\mathsf{in}\, {}^{j'}B'^{j}[e_2]]$ where $v_0 \neq \|^{\ell'} \mathsf{p}(r)\|$: By assumption, we have $\Sigma \vdash E^{\ell^*}[{}^{\ell}B^{k}[\mathsf{parameterize}\, v_0 = v_1 \,\mathsf{in}\, {}^{j'}B'^{j}[e_2]]] : \tau$ and thus by Lemma 10 that $\varnothing; \Sigma \vdash {}^{\ell}B^{k}[\mathsf{parameterize}\, v_0 = v_1 \,\mathsf{in}\, {}^{j'}B'^{j}[e_2]] : \tau'$ for some τ'. By inversion, we can derive $\varnothing; \Sigma \vdash v_0 : \tau^*$ param and $\varnothing; \Sigma \vdash v_1 : \tau^*$ for some τ^*. Decomposing $\varnothing; \Pi; \ell^* \Vdash {}^{\ell}B^{k}[\mathsf{parameterize}\, v_0 = v_1 \,\mathsf{in}\, {}^{j'}B'^{j}[e_2]]$ we have that $\ell = j = \ell^*$, $j' = k$, $v_0 = \|^{k} v_0' \|$ and $v_1 = \|^{k} v_1' \|$ for some k. Further discriminating by $\varnothing; \Sigma \vdash v_0 : \tau^*$ param we have that either $v_0' = \mathsf{p}(r)$ for some r or $v_0' = {}^{k}\mathsf{param/p}^{k'}_{j^*}(c, v_0'')$ for some k' and j^*. Since $v_0 \neq \|^{\ell'} \mathsf{p}(r)\|$, it must be the case that $v_0' = {}^{k}\mathsf{param/p}^{k'}_{j^*}(c, v_0'')$. Thus by the reduction relation and Lemma 14 we have that $\langle p, \sigma \rangle \rightarrow_{ann} \langle E^{\ell^*}[{}^{\ell^*}B^{k}[{}^{k}\langle {}^{k'}\mathsf{parameterize}\, v_0'' = {}^{k'}\mathsf{mon}^{k}_{j^*}(c, v_1') \,\mathsf{in}\, {}^{k'}\langle {}^{kk}B'^{\ell^*}[e_2] \rangle \rangle]], \sigma \rangle$.

9. ${}^{\ell}B^{k}[\mathsf{parameterize}\, \|^{\ell'} \mathsf{p}(r)\| = v_1 \,\mathsf{in}\, {}^{j^*}B'^{j}[v_2]]$: Inverting $\varnothing; \Pi; \ell^* \Vdash {}^{\ell}B^{k}[\mathsf{parameterize}\, \|^{\ell'} \mathsf{p}(r)\| = v_1 \,\mathsf{in}\, {}^{j^*}B'^{j}[v_2]]$ we have that $\ell = j = \ell^*$, $j^* = k$, $v_1 = \|^{k} v_1' \|$, and $v_2 = \|^{\ell^*} v_2' \|$. Thus by the reduction relation and Lemma 14 we have that $\langle p, \sigma \rangle \rightarrow_{ann} \langle E^{\ell^*}[|^{\ell^*} v_2'|], \sigma \rangle$.

10. $?v_0$: By assumption, we have $\Sigma \vdash E^{\ell^*}[?v_0] : \tau$ and thus by Lemma 10 that $\varnothing; \Sigma \vdash ?v_0 : \tau'$ for some τ'. By inversion, it must be the case that $\varnothing; \Sigma \vdash v_0 : \tau'$ param. This, combined with the fact that $\varnothing; \Pi; \ell^* \Vdash ?v_0$ allows us to derive that either $v_0 = \|^{\ell^*} \mathsf{p}(r)\|$ for some p or $v_0 = \|^{\ell^* \ell^*}\mathsf{param/p}^{k}_{j}(c, v_0')\|$. In the first case, we must consider two possibilities: when E^{ℓ^*} contains ${}^{\ell'}B^{k}[\mathsf{parameterize}\, \|^{k'} \mathsf{p}(r)\| = v' \,\mathsf{in}\, {}^{k_2}B^{\ell_2'}[E'^{\ell^*}[?v_0]]]$ for some k, k_2, k', ℓ', ℓ_2', v', and E'^{ℓ^*} that doesn't contain another parameterize expression for $\mathsf{p}(r)$, and when E^{ℓ^*} does not contain such a context. We consider each of these cases in turn:
   (a) $v_0 = \|^{\ell^*} \mathsf{p}(r)\|$ and $E^{\ell^*} = E'^{\ell^*}[{}^{\ell'}B^{k}[\mathsf{parameterize}\, \|^{k'} \mathsf{p}(r)\| = v' \,\mathsf{in}\, {}^{k_2}B^{\ell_2'}[E''^{\ell^*}[?v_0]]]]$ where E''^{ℓ^*} contains no parameterize for $\mathsf{p}(r)$: Inverting $\varnothing; \Pi; \ell^* \Vdash \|^{\ell^*} \mathsf{p}(r)\|$ gives $\Pi(p) = \ell^*$. Since by assumption $\varnothing; \Pi; \ell_0 \Vdash E'^{\ell''}[{}^{\ell'}B^{k}[\mathsf{parameterize}\, \|^{k'} \mathsf{p}(r)\| = v' \,\mathsf{in}\, {}^{k_2}B^{\ell_2'}[E''^{\ell^*}[?v_0]]]]$, we can again apply Lemma 13 to derive $\varnothing; \Pi; \ell'' \Vdash {}^{\ell'}B^{k}[\mathsf{parameterize}\, \|^{k'} \mathsf{p}(r)\| = v' \,\mathsf{in}\, {}^{k_2}B^{\ell_2'}[E''^{\ell^*}[?v_0]]]$. By the well-formedness rule for parameterize, $\ell' = \ell''$ and since $\Pi(p) = \ell^*$ it must be the case that $k = k_2 = k' = \ell^*$ $\ell' = \ell_2'$, and $v' = \|^{\ell^*} v'' \|$. Thus by the reduction relation and Lemma 14 we have that

$$\langle p, \sigma \rangle \rightarrow_{ann} \langle E'^{\ell'}[{}^{\ell'}B^{\ell^*}[\mathsf{parameterize}\, \|^{\ell^*} \mathsf{p}(r)\| = |^{\ell^*} v'| \,\mathsf{in}\, {}^{\ell^*}B^{\ell'}[E''^{\ell^*}[|^{\ell^*} v'|]]]], \sigma \rangle.$$

   (b) $v_0 = \|^{\ell^*} \mathsf{p}(r)\|$ and $E^{\ell^*} \neq E'^{\ell''}[{}^{\ell'}B^{k}[\mathsf{parameterize}\, \|^{k'} \mathsf{p}(r)\| = v' \,\mathsf{in}\, {}^{k_2}B^{\ell_2'}[E''^{\ell^*}[?v_0]]]]$: By the reduction semantics and Lemma 14, we have $\langle p, \sigma \rangle \rightarrow_{ann} \langle E^{\ell^*}[|^{\ell^*} v|], \sigma \rangle$ where $v = \sigma(r)$. It must be the case that $r \in \sigma$ since $\varnothing; \Sigma \vdash \|^{\ell^*} \mathsf{p}(r)\| : \tau'$ param implies $\Sigma(r) = \tau'$ and $\varnothing; \Sigma \vdash \sigma$.
   (c) $v_0 = \|^{\ell^* \ell^*}\mathsf{param/p}^{k}_{j}(c, v_0')\|$: By the reduction relation and Lemma 14, we have $\langle p, \sigma \rangle \rightarrow_{ann} \langle E^{\ell^*}[{}^{\ell^*}\mathsf{mon}^{k}_{j}(c, ?v_0')], \sigma \rangle$.

11. ${}^{\ell'}B^{\ell''}[v_0 := v_1]$: By assumption, we have $\Sigma \vdash E^{\ell^*}[{}^{\ell'}B^{\ell''}[v_0 := v_1]] : \tau$ and thus by Lemma 10 that $\varnothing; \Sigma \vdash {}^{\ell'}B^{\ell''}[v_0 := v_1] :$ Unit. By inversion, it must be the case that $\varnothing; \Sigma \vdash v_0 : \tau'$ param for some τ'. This, combined with the fact that $\varnothing; \Pi; \ell^* \Vdash E^{\ell^*}[{}^{\ell'}B^{\ell''}[v_0 := v_1]]$ allows us to derive that either $v_0 = \|^{\ell''} \mathsf{p}(r)\|$ for some r or $v_0 = \|^{\ell'' \ell''}\mathsf{param/p}^{k}_{j}(c, v_0')\|$. In the first case, we must consider two possibilities: when E^{ℓ^*} contains ${}^{j}B^{k}[\mathsf{parameterize}\, \|^{k'} \mathsf{p}(r)\| = v' \,\mathsf{in}\, {}^{k_2}B^{j_2}[E'^{\ell^*}[{}^{\ell'}B^{\ell''}[v_0 := v_1]]]]$ for some k, k', j, v', and E'^{ℓ^*} that doesn't contain another parameterize expression for $\mathsf{p}(r)$, and when E^{ℓ^*} does not contain such a context. We consider each of these cases in turn:
   (a) $v_0 = \|^{\ell''} \mathsf{p}(r)\|$ and $E^{\ell^*} = E'^{\ell^+}[{}^{j}B^{k}[\mathsf{parameterize}\, \|^{k'} \mathsf{p}(r)\| = v' \,\mathsf{in}\, {}^{k_2}B^{j_2}[E''^{\ell^*}[{}^{\ell'}B^{\ell''}[v_0 := v_1]]]]]$ where E''^{ℓ^*} contains no parameterize for $\mathsf{p}(r)$: Inverting $\varnothing; \Pi; \ell'' \Vdash \|^{\ell''} \mathsf{p}(r)\|$ gives $\Pi(r) = \ell''$. Since by assumption $\varnothing; \Pi; \ell_0 \Vdash E'^{\ell^+}[{}^{j}B^{k}[\mathsf{parameterize}\, \|^{k'} \mathsf{p}(r)\| = v' \,\mathsf{in}\, B[E''^{\ell^*}[{}^{\ell'}B^{\ell''}[v_0 := v_1]]]]]$, we can again apply Lemma 13 to derive $\varnothing; \Pi; \ell^+ \Vdash {}^{j}B^{k}[\mathsf{parameterize}\, \|^{k'} \mathsf{p}(r)\| = v' \,\mathsf{in}\, {}^{k_2}B^{j_2}[E''^{\ell^*}[{}^{\ell'}B^{\ell''}[v_0 := v_1]]]]$. By the well-formedness rule for parameterize, $\ell' = \ell^*$, $j = j_2 = \ell^+$ and since $\Pi(r) = \ell''$ it must be the case that $k = k_2 = k' = \ell''$ and $v' = \|^{\ell''} v'' \|$. Thus by the reduction relation and Lemma 14 we have that

$$\langle p, \sigma \rangle \rightarrow_{ann} \langle E'^{\ell^+}[{}^{\ell^+}B^{\ell''}[\mathsf{parameterize}\, \|^{\ell''} \mathsf{p}(r)\| = \|^{\ell''} v'' \| \,\mathsf{in}\, {}^{\ell''}B^{\ell^*}[E''^{\ell^*}[|^{\ell^*}()|]]]], \sigma \rangle.$$

(b) $v_0 = \|^{\ell''}\mathsf{p}(r)\|$ and $E^{\ell^*} \neq E'^{\ell''}[^{\ell'} B^k[\mathsf{parameterize}\ \|^{k'}\mathsf{p}(r)\| = v' \text{ in } {}^{k_2}B^{j_2}[E''^{\ell^*}[^{\ell'} B^{\ell''}[v_0 := v_1]]]]]$: By further applications of well-formedness, we can derive that $v_1 = \|^{\ell''} v_1'\|$ for some v_1' and $\ell' = \ell^*$. Thus by the reduction semantics and Lemma 14, we have $\langle p, \sigma\rangle \rightarrow_{ann} \langle E^{\ell^*}[|^{\ell^*}()|], \sigma[r \mapsto v_1']\rangle$.

(c) $v_0 = \|^{\ell''\,\ell''}\mathsf{param/p}_j^k(c, v_0')\|$: By inverting $\varnothing; \Pi; \ell^* \Vdash {}^{\ell'} B^{\ell''}[v_0 := v_1]$ we derive $\ell' = \ell^*$. Thus by the reduction relation and Lemma 14 we have $\langle p, \sigma\rangle \rightarrow_{ann} \langle E^{\ell^*}[^{\ell^*}\langle^{\ell''} v_0' := {}^k\mathsf{mon}_j^{\ell''}(c, v_1)\rangle], \sigma\rangle$.

12. ${}^{\ell}\mathsf{mon}_j^k(c_0, v_1)$: Deconstructing c_0 we consider seven cases (for simplicity, we reduce $|^{\ell'} \ldots |^{\ell''} c| \ldots|$ to $\|^{\ell'} c\|$ since well-formedness will ensure it):

(a) $\|^{\ell'}\mathsf{flat/c}(v_0)\|$: This case is a contradiction, since $\varnothing; \Pi; \ell^* \Vdash {}^{\ell}\mathsf{mon}_j^k(\|^{\ell'}\mathsf{flat/c}(v_0)\|, v_1)$ requires $\varnothing; \Pi; j; \{k\}; \{\ell\} \rhd \|^{\ell'}\mathsf{flat/c}(v_0)\|$, which does not hold.

(b) $\|^{\ell'}\mathsf{param/c}(c_0')\|$: Since $\varnothing; \Pi; \ell^* \Vdash {}^{\ell}\mathsf{mon}_j^k(\|^{\ell'}\mathsf{param/c}(c_0')\|, v_1)$ we can deduce $\ell = \ell^*$ and $\ell' = j$. Thus by the reduction relation and Lemma 14, we have $\langle p, \sigma\rangle \rightarrow_{ann} \langle E^{\ell^*}[^{\ell^*}\mathsf{param/p}_j^k(c_0', v_1)], \sigma\rangle$.

(c) $\|^{\ell'} c_d : \tau_d \rightarrow (c_c)\ c_r\|$: Since $\varnothing; \Pi; \ell^* \Vdash {}^{\ell}\mathsf{mon}_j^k(\|^{\ell'} c_d : \tau_d \rightarrow (c_c)\ c_r\|, v_1)$ we can deduce $\ell = \ell^*$ and $\ell' = j$. Thus by the reduction relation and Lemma 14, we have $\langle p, \sigma\rangle \rightarrow_{ann} \langle E^{\ell^*}[^{\ell^*}\mathsf{mon}_j^k(c_c, \lambda\, x : \tau_d.\ {}^{\ell^*}\mathsf{mon}_j^k(c_r, v\ {}^k\mathsf{mon}_j^{\ell^*}(c_d, x)))], \sigma\rangle$.

(d) $\|^{\ell'} c_d : \tau_d \rightarrow_{\mathsf{a}} (\lambda\, x : \tau_d.\ e_c)\ c_r\|$: Since $\varnothing; \Pi; \ell^* \Vdash {}^{\ell}\mathsf{mon}_j^k(\|^{\ell'} c_d : \tau_d \rightarrow_{\mathsf{a}} (\lambda\, x : \tau_d.\ e_c)\ c_r\|, v_1)$ we can deduce $\ell = \ell^*$ and $\ell' = j$. Thus by the reduction relation and Lemma 14, we have $\langle p, \sigma\rangle \rightarrow_{ann} \langle E^{\ell^*}[\lambda\, x : \tau_d.\ {}^{\ell^*}\mathsf{mon}_j^k(\{^{{}^k\mathsf{mon}_j^{\ell^*}(c_d, v)}/_x\}e_c, {}^{\ell^*}\mathsf{mon}_j^k(c_r, v\ {}^k\mathsf{mon}_j^{\ell^*}(c_d, x)))], \sigma\rangle$.

(e) $\|^{\ell'}\mathsf{ctx/c}(v_c, (v_{c_{g_1}} \Rightarrow v_{c_{p_1}} \leftarrow v_{c_{v_1}}), \ldots (v_{c_{g_n}} \Rightarrow v_{c_{p_n}} \leftarrow v_{c_{v_n}}), v_a, (v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}), \ldots)\|$: This case is a contradiction, since $\varnothing; \Pi; \ell^* \Vdash {}^{\ell}\mathsf{mon}_j^k(\|^{\ell'}\mathsf{ctx/c}(v_c, (v_{c_{g_1}} \Rightarrow v_{c_{p_1}} \leftarrow v_{c_{v_1}}), \ldots (v_{c_{g_n}} \Rightarrow v_{c_{p_n}} \leftarrow v_{c_{v_n}}), v_a, (v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}), \ldots)\|, v_1)$ requires $\varnothing; \Pi; j; \{k\}; \{\ell\} \rhd \|^{\ell'}\mathsf{ctx/c}(v_c, (v_{c_{g_1}} \Rightarrow v_{c_{p_1}} \leftarrow v_{c_{v_1}}), \ldots (v_{c_{g_n}} \Rightarrow v_{c_{p_n}} \leftarrow v_{c_{v_n}}), v_a, (v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}), \ldots)\|$, which does not hold.

(f) $\lfloor\|^{\ell'}\mathsf{flat/c}(v_0)\|\rfloor^{\vec{\ell}}$: Since $\varnothing; \Pi; \ell^* \Vdash {}^{\ell}\mathsf{mon}_j^k(\lfloor\|^{\ell'}\mathsf{flat/c}(v_0)\|\rfloor^{\vec{\ell}}, v_1)$ we can deduce $\ell = \ell^*$ and $\ell' = j$. Thus by the reduction relation and Lemma 14, we have $\langle p, \sigma\rangle \rightarrow_{ann} \langle E^{\ell^*}[\mathsf{check}_j^k((v_0\ v_1), v_1)], \sigma\rangle$.

(g) $\lfloor\|^{\ell'}\mathsf{ctx/c}(v_c, (v_{c_{g_1}} \Rightarrow v_{c_{p_1}} \leftarrow v_{c_{v_1}}), \ldots (v_{c_{g_n}} \Rightarrow v_{c_{p_n}} \leftarrow v_{c_{v_n}}), v_a, (v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}), \ldots)\|\rfloor^{\vec{\ell}}$: Since $\varnothing; \Pi; \ell^* \Vdash {}^{\ell}\mathsf{mon}_j^k(\lfloor\|^{\ell'}\mathsf{ctx/c}(v_c, (v_{c_{g_1}} \Rightarrow v_{c_{p_1}} \leftarrow v_{c_{v_1}}), \ldots (v_{c_{g_n}} \Rightarrow v_{c_{p_n}} \leftarrow v_{c_{v_n}}), v_a, (v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}), \ldots)\|\rfloor^{\vec{\ell}}, v_1)$ we can deduce $\ell = \ell^*$ and $\ell' = j$. Thus by the reduction relation and Lemma 14, we have $\langle p, \sigma\rangle \rightarrow_{ann} \langle E^{\ell^*}[\mathsf{check}_j^k((v_c\ ()), e_0)], \sigma\rangle$ where $e_0 = \mathsf{guard}_j\ ((v_{c_{g_1}}\ ()), v_{c_{p_1}}, v_{c_{v_1}}, \ldots \mathsf{guard}_j\ ((v_{c_{g_n}}\ ()), v_{c_{p_n}}, v_{c_{v_n}}, {}^{\ell^*}\mathsf{ctx/p}_j^k\ (v_a, (v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}), \ldots, v_1)))$.

13. $\mathsf{check}_j^k(v_0, e_1)$: By assumption, we have $\Sigma \vdash E^{\ell^*}[\mathsf{check}_j^k(v_0, e_1)] : \tau$. By Lemma 10 we have $\varnothing; \Sigma \vdash \mathsf{check}_j^k(v_0, e_1) : \tau'$. By inversion, $\varnothing; \Sigma \vdash v_0 : \mathsf{Bool}$. This, combined with further decomposing $\varnothing; \Pi; \ell^* \Vdash \mathsf{check}_j^k(v_0, e_1)$, is sufficient to derive that either $v_0 = \|^j\#\mathsf{t}\|$ or $v_0 = \|^j\#\mathsf{f}\|$. In the first case and appealing to Lemma 14 we have $\langle p, \sigma\rangle \rightarrow_{ann} \langle E^{\ell^*}[e_1], \sigma\rangle$. In the latter, we have $\langle p, \sigma\rangle \rightarrow_{ann} \langle \mathsf{error}_j^k, \sigma\rangle$.

14. $\mathsf{guard}_j\ (v_g, v_p, v_v, e_f)$: By assumption, we have $\Sigma \vdash E^{\ell^*}[\mathsf{guard}_j\ (v_g, v_p, v_v, e_f)] : \tau$. By Lemma 10 we have $\varnothing; \Sigma \vdash \mathsf{guard}_j\ (v_g, v_p, v_v, e_f) : \tau'$. By inversion, $\varnothing; \Sigma \vdash v_g : \mathsf{Bool}$. This, combined with further decomposing $\varnothing; \Pi; \ell^* \Vdash \mathsf{guard}_j\ (v_g, v_p, v_v, e_f)$, is sufficient to derive that either $v_g = \|^j\#\mathsf{t}\|$ or $v_g = \|^j\#\mathsf{f}\|$. Using Lemma 14, we can now construct reductions for each case. In the first, we have $\langle p, \sigma\rangle \rightarrow_{ann} \langle E^{\ell^*}[e_f], \sigma\rangle$. In the latter, we have $\langle p, \sigma\rangle \rightarrow_{ann} \langle E^{\ell^*}[\mathsf{install/p}_j\ (v_p, (v_v\ ()), e_f)], \sigma\rangle$.

□

Lemma 12 (Preservation). *For all programs p, stores σ, store typings Σ, and store labelings Π such that*

- $\Sigma \vdash p : \tau$,
- $\varnothing; \Sigma \vdash \sigma$,
- $\ell_0; \Pi \Vdash p$, *and*
- $\langle p, \sigma\rangle \rightarrow_{ann} \langle p', \sigma'\rangle$,

there exists a store labeling Π' such that

- $\Pi' \supseteq \Pi$,
- $\ell_0; \Pi' \Vdash p'$.

Proof. By case analysis on the reduction $\langle p, \sigma\rangle \rightarrow_{ann} \langle p', \sigma'\rangle$:

1. $\langle$module ℓ exports x_1 with x_{c_1},... where $y_1 = v_1$,...,$y_n = v_n$,$y_{e_1} = e_1$,... $y_{e_m} = e_m$; $p, \sigma\rangle$ $\rightarrow_{ann}$ $\langle$module ℓ exports x_1 with x_{c_1},... where $y_1 = v_1$,...,$y_n = v_n$,$y_{e_1} = \{^{v_i}/_{y_i}\}e_1$,$y_{e_2} = e_2$,... $y_{e_m} = e_m$; $p, \sigma\rangle$: Let $\Pi' = \Pi$. It suffices to show that given $\cdot; \Pi \Vdash$ module ℓ exports x_{v_1} with x_{c_1},... where $x_1 = v_1$,...,$x_n = v_n$,$x_{e_1} = e_1$,... $\triangleright \Delta$, $\cdot; \Pi \Vdash$ module ℓ exports x_1 with x_{c_1},... where $y_1 = v_1$,...,$y_n = v_n$,$y_{e_1} = \{^{v_i}/_{y_i}\}e_1$,$y_{e_2} = e_2$,... $y_{e_m} = e_m \triangleright \Delta$. From the premise, we have that $\varnothing; \Pi; \ell \Vdash v_1, \ldots, \{y_1 : \ell, \ldots, y_{n-1} : \ell\}; \Pi; \ell \Vdash v_n, \{y_1 : \ell, \ldots, y_n : \ell\}; \Pi; \ell \Vdash e_1, \ldots, \{y_1 : \ell, \ldots, y_n : \ell, y_{e_1} : \ell, \ldots, y_{e_{m-1}} : \ell\}; \Pi; \ell \Vdash e_m$, and $\Gamma' = \{y_1 : \ell, \ldots, y_n : \ell, y_{e_1} : \ell, \ldots, y_{e_m} : \ell\} \upharpoonright_{\{x_1,\ldots,x_{n'}\}}$. These suffice to show the desired result, except we must also show $\{y_1 : \ell, \ldots, y_n : \ell\}; \Pi; \ell \Vdash \{^{v_i}/_{y_i}\}e_1$ by applying Lemma 15.

2. $\langle$module ℓ exports x_{v_1} with x_{c_1},... where ...,$x_{v_1} = v_1$,...,$x_{c_1} = c_1$,...; $p', \sigma\rangle$ $\rightarrow_{ann}$ $\langle$import$[\![\ell,(x_1,\ldots,x_n),(v_1,\ldots,v_n),(c_1,\ldots,c_n),p']\!], \sigma\rangle$: By Lemma 20 since from the premise we have $\Delta; \Pi \Vdash m_1 \triangleright \Delta_1$, $\varnothing; \Pi \Vdash m_n \triangleright \Delta_n$,..., $\varnothing; \Pi \Vdash m_n \triangleright \Delta_n$, $\Delta_n; \Pi; \ell_0 \Vdash e$, where $p' = m_1; \ldots; m_n; e$ and $\Delta = \{x_1 : \ell, \ldots, x_m : \ell\}$ and also $\Delta; \Pi; \ell \Vdash v_1$, ..., and $\Delta; \Pi; \ell \Vdash c_1$, ...

3. $\langle E^\ell[[\|^\ell \lambda\, x : \tau.\ e\| \ \|^\ell v\|]], \sigma\rangle$ $\rightarrow_{ann}$ $\langle E^\ell[\{^{|^\ell v|}/_x\}e], \sigma\rangle$: By Lemma 14, it suffices to show that $\varnothing; \Pi; \ell \Vdash \{^{|^\ell v|}/_x\}e$. By Lemma 13 and well-formedness, we have that $\varnothing[x \mapsto \ell]; \Pi; \ell \Vdash e$ and $\varnothing; \Pi; \ell \Vdash v$, so by Lemma 15 $\varnothing; \Pi; \ell \Vdash \{^{|^\ell v|}/_x\}e$.

4. $\langle E^\ell[\mu\, x : \tau.\ e], \sigma\rangle$ $\rightarrow_{ann}$ $\langle E^\ell[\{^{|^\ell \mu\, x:\tau.\ e|}/_x\}e], \sigma\rangle$: By Lemma 14, it suffices to show that $\varnothing; \Pi; \ell \Vdash \{^{|^\ell \mu\, x:\tau.\ e|}/_x\}e$. By Lemma 13 and well-formedness, we have that $\varnothing[x \mapsto \ell]; \Pi; \ell \Vdash e$ and $\varnothing; \Pi; \ell \Vdash \mu\, x : \tau.\ e$, so by Lemma 15 $\varnothing; \Pi; \ell \Vdash \{^{|^\ell \mu\, x:\tau.\ e|}/_x\}e$.

5. $\langle E^\ell[\mathsf{let}\, x = \|^\ell v\| \,\mathsf{in}\, e], \sigma\rangle$ $\rightarrow_{ann}$ $\langle E^\ell[\{^{|^\ell v|}/_x\}e], \sigma\rangle$: By Lemma 14, it suffices to show that $\varnothing; \Pi; \ell \Vdash \{^{|^\ell v|}/_x\}e$. By Lemma 13 and well-formedness, we have that $\varnothing[x \mapsto \ell]; \Pi; \ell \Vdash e$ and $\varnothing; \Pi; \ell \Vdash v$, so by Lemma 15 $\varnothing; \Pi; \ell \Vdash \{^{|^\ell v|}/_x\}e$.

6. $\langle E^\ell[\mathsf{if}\ \|^\ell \#\mathsf{t}\|\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2], \sigma\rangle$ $\rightarrow_{ann}$ $\langle E^\ell[e_1], \sigma\rangle$: By Lemma 14, it suffices to show that $\varnothing; \Pi; \ell \Vdash e_1$, which holds by Lemma 14 and well-formedness.

7. $\langle E^\ell[\mathsf{if}\ \|^\ell \#\mathsf{f}\|\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2], \sigma\rangle$ $\rightarrow_{ann}$ $\langle E^\ell[e_2], \sigma\rangle$: By Lemma 14, it suffices to show that $\varnothing; \Pi; \ell \Vdash e_2$, which holds by Lemma 14 and well-formedness.

8. $\langle E^\ell[[\|^\ell v_1\| \oplus \|^\ell v_2\|], \sigma\rangle$ $\rightarrow_{ann}$ $\langle E^\ell[|^\ell v|], \sigma\rangle$ where $v = v_1 \oplus v_2$: By Lemma 14, it suffices to show that $\varnothing; \Pi; \ell \Vdash |^\ell v|$. By the type system, v_1, v_2, and v must be integers, so $\varnothing; \Pi; \ell \Vdash |^\ell v|$ holds directly.

9. $\langle E^\ell[[\|^\ell v_1\| \leq \|^\ell v_2\|], \sigma\rangle$ $\rightarrow_{ann}$ $\langle E^\ell[|^\ell v|], \sigma\rangle$ where $v = v_1 \leq v_2$: By Lemma 14, it suffices to show that $\varnothing; \Pi; \ell \Vdash \|^\ell v\|$. v is either #t or #f, so $\varnothing; \Pi; \ell \Vdash \|^\ell v\|$ holds directly.

10. $\langle E^\ell[\mathsf{make\text{-}parameter}\ \|^\ell v\|], \sigma\rangle$ $\rightarrow_{ann}$ $\langle E^\ell[|^\ell \mathsf{p}(r)|], \sigma[r \mapsto v]\rangle$ where r is fresh: Let $\Pi' = \Pi[r \mapsto \ell]$. We must show that $\ell_0; \Pi' \Vdash E^\ell[|^\ell \mathsf{p}(r)|]$. Since by Lemma 16, $\ell_0; \Pi' \Vdash E^\ell[\mathsf{make\text{-}parameter}\ \|^\ell v\|]$, by Lemma 14 it suffices to show that $\varnothing; \Pi'; \ell \Vdash \|^\ell |^\ell \mathsf{p}(r)|\|$, which holds by inspection.

11. $\langle E^\ell[?\|^\ell \mathsf{p}(r)\|], \sigma\rangle$ $\rightarrow_{ann}$ $\langle E^\ell[|^\ell v|], \sigma\rangle$ where $\sigma(r) = v$ and E does not contain $\mathsf{parameterize}^{\ell'} B^k[\mathsf{p}(r)] = v'\ \mathsf{in}\ E'$: By Lemma 14, it suffices to show that $\varnothing; \Pi; \ell \Vdash |^\ell v|$. By Lemma 13, we have that $\varnothing; \Pi; \ell \Vdash ?\|^\ell \mathsf{p}(r)\|$, so by inversion we can derive $\Pi(r) = \ell$ and $\varnothing; \Pi \Vdash \sigma$. Combining these facts, we can derive $\varnothing; \Pi; \ell \Vdash v$, which is sufficient to show $\varnothing; \Pi; \ell \Vdash |^\ell v|$.

12. $\langle E^\ell[{}^\ell B^k[\mathsf{parameterize}\ \|^k \mathsf{p}(r)\| = \|^k v\|\ \mathsf{in}\ {}^k B^\ell[E'^k[?\|^k \mathsf{p}(r)\|]]]], \sigma\rangle$ $\rightarrow_{ann}$ $\langle E^\ell[{}^\ell B^k[\mathsf{parameterize}\ |^k \mathsf{p}(r)| = |^k v|\ \mathsf{in}\ {}^k B^\ell[E'^k[|^k v|]]]], \sigma\rangle$ where $E'^{\ell'}$ does not contain ${}^{\ell^*} B''^{k^*}[\mathsf{parameterize}\ \|^{k^*} \mathsf{p}(r)\| = v'\ \mathsf{in}\ {}^{k^*} B^{\ell_2^*}[E''^{\ell''}]]$: By Lemma 14, it suffices to show that $\varnothing; \Pi; \ell \Vdash {}^\ell B^k[\mathsf{parameterize}\ |^k \mathsf{p}(r)| = |^k v|\ \mathsf{in}\ {}^k B^\ell[E'^k[|^k v|]]]$. This can be shown by applying the well-formedness of parameterize using lemma 14 to demonstrate that $\varnothing; \Pi; \ell \Vdash E'^k[|^k v|]$ by showing that $\varnothing; \Pi; \ell \Vdash |^k v|$. By Lemma 13, we have that $\varnothing; \Pi; k \Vdash ?\|^k \mathsf{p}(r)\|$, so by inversion we can derive $\Pi(r) = k$ and $\varnothing; \Pi \Vdash \sigma$. Combining these facts, we can derive $\varnothing; \Pi; k \Vdash v$, which is sufficient to show $\varnothing; \Pi; k \Vdash |^k v|$.

13. $\langle E^\ell[{}^\ell B^k[[\|^k \mathsf{p}(r)\| := \|^k v\|]], \sigma\rangle$ $\rightarrow_{ann}$ $\langle E^\ell[|^\ell ()|], \sigma[r \mapsto v]\rangle$ where E^ℓ does not contain ${}^{\ell^*} B'^{k^*}[\mathsf{parameterize}\ \|^{k^*} \mathsf{p}(r)\| = v'\ \mathsf{in}\ {}^{k^*} B^{\ell^*}[E'^{\ell'}]]$: Let $\Pi' = \Pi$. Then it suffices to show that $\Pi(r) = k$, $\varnothing; \Pi; k \Vdash v$, and

(by Lemma 14, $\varnothing;\Pi;\ell \Vdash |^{\ell}()|$. The last holds by inspection. $\Pi(r) = k$ and $\varnothing;\Pi;k \Vdash v$ hold by Lemma 13 and well-formedness or parameter assignment.

14. $\langle E^{\ell}[^{\ell}B^{k}[\mathsf{parameterize}\ \|^{k}\mathsf{p}(r)\| = \|^{k}v\|\ \mathsf{in}\ ^{k}B^{\ell}[E'^{\ell'}[^{\ell'}B'^{k}[\|^{k}\mathsf{p}(r)\| := \|^{k}v'\|]]]]], \sigma\rangle \quad \rightarrow_{ann}$
$\langle E^{\ell}[^{\ell}B^{k}[\mathsf{parameterize}\ |^{k}\mathsf{p}(r)| = |^{k}v'|\ \mathsf{in}\ ^{k}B^{\ell}[E'^{\ell'}[|^{\ell'}()|]]]], \sigma\rangle$ where $E'^{\ell'}$ does not contain $^{\ell^*}B''^{k^*}[\mathsf{parameterize}\ \|^{k^*}\mathsf{p}(r)\| = v''\ \mathsf{in}\ ^{k_2^*}B^{\ell_2^*}[E''^{\ell''}]]$: By Lemma 14, it suffices to show that $\varnothing;\Pi;\ell \Vdash {}^{\ell}B^{k}[\mathsf{parameterize}\ |^{k}\mathsf{p}(r)| = |^{k}v'|\ \mathsf{in}\ ^{k}B^{\ell}[E'^{\ell'}[|^{\ell'}()|]]]$. This holds by well-formedness of parameterize, coupled with an application of Lemma 14 to show that $\varnothing;\Pi;\ell \Vdash E'^{\ell}[|^{\ell'}()|]$ since $\varnothing;\Pi;\ell' \Vdash |^{\ell'}()|$.

15. $\langle E^{\ell}[^{\ell}B^{k}[\mathsf{parameterize}\ \|^{k}\mathsf{p}(r)\| = \|^{k}v\|\ \mathsf{in}\ ^{k}B'^{\ell}[v']]], \sigma\rangle \quad \rightarrow_{ann} \quad \langle E^{\ell}[|^{\ell}v'|], \sigma\rangle$: By Lemma 14, it suffices to show that $\varnothing;\Pi;\ell \Vdash |^{\ell}v'|$, which follows from Lemma 13 and well-formedness.

16. $\langle E^{\ell}[^{\ell}\mathsf{mon}_j^k(\lfloor\|^{j}\mathsf{flat/c}(v_c)\|\rfloor^{\vec{k}},\|^{k}v\|)], \sigma\rangle \quad \rightarrow_{ann} \quad \langle E^{\ell}[\mathsf{check}_j^k((v_c\ v),v)], \sigma\rangle$: By Lemma 14, it suffices to show that $\varnothing;\Pi;\ell \Vdash \mathsf{check}_j^k((v_c\ v),v)$. In turn, we must show that $\varnothing;\Pi;j \Vdash (v_c\ v)$ and $\varnothing;\Pi;\ell \Vdash v$. By Lemma 10 we can deduce that $\varnothing;\Sigma \vdash {}^{\ell}\mathsf{mon}_j^k(\lfloor\|^{j}\mathsf{flat/c}(v_c)\|\rfloor^{\vec{k}},\|^{k}v\|) : \beta$ for some β, and thus that $\varnothing;\Sigma \vdash v : \beta$. Thus v is either (), #t, #f, or an integer, and so $\varnothing;\Pi;\ell^* \Vdash v$ for all ℓ^*. Finally, $\varnothing;\Pi;j \Vdash (v_c\ v)$ since $\varnothing;\Pi;j \Vdash v$ and $\varnothing;\Pi;j \Vdash v_c$ by well-formedness of contracts for $\mathsf{flat/c}$.

17. $\langle E^{\ell}[^{\ell}\mathsf{mon}_j^k(\|^{j}\mathsf{param/c}(c)\|,v)], \sigma\rangle \quad \rightarrow_{ann} \quad \langle E^{\ell}[^{\ell}\mathsf{param/p}_j^k(c,v)], \sigma\rangle$: By Lemma 14, it suffices to show that $\varnothing;\Pi;\ell \Vdash {}^{\ell}\mathsf{param/p}_j^k(c,v)$, which requires that $\varnothing;\Pi;\{k\};\{\ell\};j \rhd c$ and $\varnothing;\Delta;k \Vdash v$. By Lemma 13 we have that $\varnothing;\Pi;\ell \Vdash {}^{\ell}\mathsf{mon}_j^k(\|^{j}\mathsf{param/c}(c)\|,v)$. Thus by inversion we have $\varnothing;\Pi;k \Vdash v$ and $\varnothing;\Pi;\{k\};\{\ell\};j \rhd \mathsf{param/c}(c)$. Inverting the latter a second time, we have that $\varnothing;\Pi;\{k,\ell\};\{k,\ell\};j \rhd c$, which by Lemma 17 implies $\varnothing;\Pi;\{k\};\{\ell\};j \rhd c$.

18. $\langle E^{\ell}[^{\ell}\mathsf{mon}_j^k(\|^{j}c_d : \tau \rightarrow (c_c)\ c_r\|,v)], \sigma\rangle \quad \rightarrow_{ann} \quad \langle E^{\ell}[^{\ell}\mathsf{mon}_j^k(c_c,\lambda\, x : \tau_d.\ {}^{\ell}\mathsf{mon}_j^k(c_r,v\ {}^{k}\mathsf{mon}_j^{\ell}(c_d,x)))], \sigma\rangle$: By Lemma 14, it suffices to show that $\varnothing;\Pi;\ell \Vdash {}^{\ell}\mathsf{mon}_j^k(c_c,\lambda\, x : \tau_d.\ {}^{\ell}\mathsf{mon}_j^k(c_r,v\ {}^{k}\mathsf{mon}_j^{\ell}(c_d,x)))$. We first must establish that $c_c = \|^{j}\lfloor\|^{j}\mathsf{ctx/c}(v_c,(v_{c_{g_1}} \Rightarrow v_{c_{p_1}} \leftarrow v_{c_{v_1}}),\ldots(v_{c_{g_n}} \Rightarrow v_{c_{p_n}} \leftarrow v_{c_{v_n}}),v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\ldots)\|\rfloor^{\vec{\ell}}\|$ since by Lemma 10 and repeated inversions we can show that $\varnothing;\Sigma \vdash c_c : \mathsf{ctx}\ \mathsf{ctc}$. Thus by well-formedness for monitors for context contracts, we reduce what needs to be shown to $\varnothing;\Pi;\{k\};\{\ell\};j \rhd c'_d$ and $\varnothing;\Pi;\ell \Vdash \lambda\, x : \tau_d.\ {}^{\ell}\mathsf{mon}_j^k(c_r,v\ {}^{k}\mathsf{mon}_j^{\ell}(c_d,x))$. The first follows from the fact that $\varnothing;\Pi;\{k\};\{\ell\};j \rhd c_d : \tau \rightarrow (c_c)\ c_r$. For the second, we must show that $\varnothing[x \mapsto \ell];\Pi;\ell \Vdash {}^{\ell}\mathsf{mon}_j^k(c_r,v\ {}^{k}\mathsf{mon}_j^{\ell}(c_d,x))$. For this, we must show that $\varnothing[x \mapsto \ell];\Pi;\{k\};\{\ell\};j \rhd c_r$ and (since by Lemma 10 and repeated inversions we can show that $\varnothing;\Sigma \vdash c_r : \tau_r\ \mathsf{ctc}$ for some τ_r) $\varnothing[x \mapsto \ell];\Pi;k \Vdash (v\ {}^{k}\mathsf{mon}_j^{\ell}(c_d,x))$. This in turn requires showing that $\varnothing[x \mapsto \ell];\Pi;\ell \Vdash v$, $\varnothing[x \mapsto \ell];\Pi;\ell \Vdash x$, and $\varnothing[x \mapsto \ell];\Pi;\{\ell\};\{k\};j \rhd c_d$ (since again by Lemma 10 and repeated inversions we can show that $\varnothing;\Sigma \vdash c_d : \tau_d\ \mathsf{ctc}$ for some τ_d). $\varnothing[x \mapsto \ell];\Pi;\ell \Vdash x$ holds trivially. For the remainder, we apply Lemmas 18 and 19 to the facts $\varnothing;\Pi;\{k\};\{\ell\};j \rhd c_r$, $\varnothing;\Pi;\ell \Vdash v$, and $\varnothing[x \mapsto \ell];\Pi;\{\ell\};\{k\};j \rhd c_d$, which can be obtained by Lemma 13 applied to the premise followed by repeated inversions.

19. $\langle E^{\ell}[^{\ell}\mathsf{mon}_j^k(\|^{j}c_d : \tau \rightarrow_{\mathsf{a}} (\lambda\, x : \tau.\ e_c)\ c_r\|,v)], \sigma\rangle \quad \rightarrow_{ann}$
$\langle E^{\ell}[\lambda\, y : \tau_d.\ {}^{\ell}\mathsf{mon}_j^k(\{{}^{k}\mathsf{mon}_j^{\ell}(c_d,y)/_x\}e_c,{}^{\ell}\mathsf{mon}_j^k(c_r,v\ {}^{k}\mathsf{mon}_j^{\ell}(c_d,y)))], \sigma\rangle$: By Lemma 14, it suffices to show that $\varnothing;\Pi;\ell \Vdash \lambda\, y : \tau_d.\ {}^{\ell}\mathsf{mon}_j^k(\{{}^{k}\mathsf{mon}_j^{\ell}(c_d,y)/_x\}e_c,{}^{\ell}\mathsf{mon}_j^k(c_r,v\ {}^{k}\mathsf{mon}_j^{\ell}(c_d,y)))$. This follows from $\varnothing[y \mapsto \ell];\Pi;\ell \Vdash {}^{\ell}\mathsf{mon}_j^k(\{{}^{k}\mathsf{mon}_j^{\ell}(c_d,y)/_x\}e_c,{}^{\ell}\mathsf{mon}_j^k(c_r,v\ {}^{k}\mathsf{mon}_j^{\ell}(c_d,y)))$. The proof of this fact mirrors the proof in case (18), except that we must also appeal to Lemma 15 to verify the well-formedness of $\{{}^{\ell}\mathsf{mon}_j^k(c_d,y)/_x\}e_c$.

20. $\langle E^{\ell}[^{\ell}\mathsf{mon}_j^k(\lfloor\|^{j}\mathsf{ctx/c}(v_c,(v_{c_{g_1}} \Rightarrow v_{c_{p_1}} \leftarrow v_{c_{v_1}}),\ldots(v_{c_{g_n}} \Rightarrow v_{c_{p_n}} \leftarrow v_{c_{v_n}}),v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\ldots)\|\rfloor^{\vec{\ell}},v)], \sigma\rangle$
$\rightarrow_{ann} \quad \langle E^{\ell}[\mathsf{check}_j^k((v_c\ ()),e)], \sigma\rangle$ where
$e = \mathsf{guard}_j\ ((v_{c_{g_1}}\ ()),v_{c_{p_1}},v_{c_{v_1}},\ldots\mathsf{guard}_j\ ((v_{c_{g_n}}\ ()),v_{c_{p_n}},v_{c_{v_n}},{}^{\ell}\mathsf{ctx/p}_j^k\ (v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\ldots,v)))$: By Lemma 14, it suffices to show that $\varnothing;\Pi;\ell \Vdash \mathsf{check}_j^k((v_c\ ()),e)$. To do so, we apply Lemma 13 and repeated inversions to deduce $\varnothing;\Pi;j \Vdash v_c$, $\varnothing;\Pi;j \Vdash v_a$, $\varnothing;\Pi;j \Vdash v_{c_{g_i}}$, $\varnothing;\Pi;j \Vdash v_{c_{p_i}}$, $\varnothing;\Pi;j \Vdash v_{c_{v_i}}$, $\varnothing;\Pi;j \Vdash v_{a_{g_i}}$, $\varnothing;\Pi;j \Vdash v_{a_{p_i}}$, $\varnothing;\Pi;j \Vdash v_{a_{v_i}}$, and $\varnothing;\Pi;\ell \Vdash v$ and use these facts to demonstrate the necessary well-formedness property.

21. $\langle E^{\ell}[(\|^{\ell\ell}\mathsf{ctx/p}_j^k\ (v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\ldots(v_{a_{g_n}} \Rightarrow v_{a_{p_n}} \leftarrow v_{a_{v_n}}),v_f)\|\ v)], \sigma\rangle \quad \rightarrow_{ann}$
$\langle E^{\ell}[(\mathsf{check}_j^k((v_a\ ()),e)\ v)], \sigma\rangle$ where $e = \mathsf{guard}_j\ ((v_{a_{g_1}}\ ()),v_{a_{p_1}},v_{a_{v_1}},\ldots\mathsf{guard}_j\ ((v_{a_{g_n}}\ ()),v_{a_{p_n}},v_{a_{v_n}},v_f))$: By Lemma 14, it suffices to show that $\varnothing;\Pi;\ell \Vdash (\mathsf{check}_j^k((v_a\ ()),e)\ v)$. In turn, we must show $\varnothing;\Pi;\ell \Vdash \mathsf{check}_j^k((v_a\ ()),e)$

and $\varnothing;\Pi;\ell \Vdash v$. The latter holds by Lemma 13 and well-formedness. The former requires further showing that $\varnothing;\Pi;j \Vdash v_a$, $\varnothing;\Pi;j \Vdash ()$, and $\varnothing;\Pi;\ell \Vdash e$. This is straightforward, since we can derive $\varnothing;\Pi;j \Vdash v_a$, $\varnothing;\Pi;j \Vdash v_{a_{g_i}}$, $\varnothing;\Pi;j \Vdash v_{a_{p_i}}$, $\varnothing;\Pi;j \Vdash v_{a_{v_i}}$, and $\varnothing;\Pi;\ell \Vdash v_f$ by Lemma 13 and repeated applications of well-formedness.

22. $\langle E^\ell[\mathsf{guard}_j\ (\|^k\#\mathsf{f}\|,v_p,v_v,e_f)],\sigma\rangle \quad \to_{ann} \quad \langle E^\ell[e_f],\sigma\rangle$: By Lemma 14, it suffices to show that $\varnothing;\Pi;\ell \Vdash e_f$. This holds since by Lemma 13 and inversion $\varnothing;\Pi;\ell \Vdash \mathsf{guard}_j\ (\|^k\#\mathsf{f}\|,v_p,v_v,e_f)$, which requires that $\varnothing;\Pi;\ell \Vdash e_f$.

23. $\langle E^\ell[\mathsf{guard}_j\ (\|^k\#\mathsf{t}\|,v_p,v_v,e_f)],\sigma\rangle \quad \to_{ann} \quad \langle E^\ell[\mathsf{install/p}_j\ (v_p,(v_v\ ()),e_f)],\sigma\rangle$: By Lemma 14, it suffices to show that $\varnothing;\Pi;\ell \Vdash \mathsf{install/p}_j\ (v_p,(v_v\ ()),e_f)$. By well-formedness, we must show that $\varnothing;\Pi;j \Vdash v_p$, $\varnothing;\Pi;j \Vdash (v_v\ ())$, $\varnothing;\Pi;\ell \Vdash e_f$. By Lemma 13 and inversion we have $\varnothing;\Pi;\ell \Vdash \mathsf{guard}_j\ (\|^k\#\mathsf{t}\|,v_p,v_v,e_f)$, which inverting again is sufficient to show the required well-formedness properties.

24. $\langle E^\ell[(\|^\ell\mathsf{install/p}_j\ (v_p,v_v,v_f)\|\ v)],\sigma\rangle \quad \to_{ann} \quad \langle E^\ell[^\ell\langle^j\mathsf{parameterize}\ v_p = v_v\ \mathsf{in}\ ^j\langle^\ell(v_f\ v)\rangle\rangle],\sigma\rangle$: By Lemma 14, it suffices to show that $\varnothing;\Pi;\ell \Vdash {}^\ell\langle{}^j\mathsf{parameterize}\ v_p = v_v\ \mathsf{in}\ {}^j\langle{}^\ell(v_f\ v)\rangle\rangle$. By the well-formedness rule for parameterize, we must show that $\varnothing;\Pi;j \Vdash v_p$, $\varnothing;\Pi;j \Vdash v_v$, and $\varnothing;\Pi;\ell \Vdash (v_f\ v)$. By Lemma 13 and inversion we can derive $\varnothing;\Pi;\ell \Vdash \mathsf{install/p}_j\ (v_p,v_v,v_f)$ and $\varnothing;\Pi;\ell \Vdash v$. Inverting the former, we can further infer $\varnothing;\Pi;j \Vdash v_p$, $\varnothing;\Pi;j \Vdash v_v$, and $\varnothing;\Pi;\ell \Vdash v_f$. All that remains to be show is $\varnothing;\Pi;\ell \Vdash (v_f\ v)$, which follows through a single application of the well-formedness rule for application.

25. $\langle E^\ell[?\|^{\ell\ell}\mathsf{param/p}^k_j(c,v_p)\|],\sigma\rangle \quad \to_{ann} \quad \langle E^\ell[^\ell\mathsf{mon}^k_j(c,?v_p)],\sigma\rangle$: By Lemma 14, it suffices to show that $\varnothing;\Pi;\ell \Vdash {}^\ell\mathsf{mon}^k_j(c,?v_p)$. By Lemma 10, we can derive $\varnothing;\Sigma \vdash ?\|^{\ell\ell}\mathsf{param/p}^k_j(c,v_p)\| : \tau'$ for some τ', and by further inversion that $\varnothing;\Sigma \vdash c : \tau''$ ctc for some τ''. Thus $c \neq \lfloor\|^j\mathsf{ctx/c}(v_c,(v_{c_{g_1}} \Rightarrow v_{c_{p_1}} \leftarrow v_{c_{v_1}}),\dots(v_{c_{g_n}} \Rightarrow v_{c_{p_n}} \leftarrow v_{c_{v_n}}),v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\dots)\|\rfloor^{\vec{\ell}}$, and so by the well-formedness rule for monitors we must show that $\varnothing;\Pi;\{k\};\{\ell\};j \rhd c$ and $\varnothing;\Pi;k \Vdash ?v_p$. By Lemma 13 and inversion, we can derive $\varnothing;\Pi;\ell \Vdash ?\|^{\ell\ell}\mathsf{param/p}^k_j(c,v_p)\|$ and thus by inversion the required facts.

26. $\langle E^\ell[^\ell B^{\ell'}[\mathsf{parameterize}\ \|^{\ell'\ell'}\mathsf{param/p}^k_j(c,v_p)\| = v\ \mathsf{in}\ ^{\ell'}B'^\ell[e]]],\sigma\rangle \quad \to_{ann}$
$\langle E^\ell[^\ell\langle{}^k\mathsf{parameterize}\ v_p = {}^k\mathsf{mon}^{\ell'}_j(c,v)\ \mathsf{in}\ {}^k\langle{}^\ell e\rangle\rangle],\sigma\rangle$: By Lemma 14, it suffices to show that $\varnothing;\Pi;\ell \Vdash {}^\ell\langle{}^k\mathsf{parameterize}\ v_p = {}^k\mathsf{mon}^{\ell'}_j(c,v)\ \mathsf{in}\ {}^k\langle{}^\ell e\rangle\rangle$, which by the well-formedness of parameterize simply requires that $\varnothing;\Pi;k \Vdash v_p$, $\varnothing;\Pi;\ell \Vdash e$, and $\varnothing;\Pi;k \Vdash {}^k\mathsf{mon}^{\ell'}_j(c,v)$. The first two facts can be established by inverting $\varnothing;\Pi;\ell \Vdash {}^\ell B^{\ell'}[\mathsf{parameterize}\ \|^{\ell'\ell'}\mathsf{param/p}^k_j(c,v_p)\| = v\ \mathsf{in}\ {}^{\ell'}B'^\ell[e]]$, which can be shown using Lemma 13. For the third, we first must establish that $c \neq \lfloor\|^j\mathsf{ctx/c}(v_c,(v_{c_{g_1}} \Rightarrow v_{c_{p_1}} \leftarrow v_{c_{v_1}}),\dots(v_{c_{g_n}} \Rightarrow v_{c_{p_n}} \leftarrow v_{c_{v_n}}),v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\dots)\|\rfloor^{\vec{\ell}}$, since by Lemma 10 and repeated inversions we can show that $\varnothing;\Sigma \vdash c : \tau'$ ctc for some τ'. Then by the remaining rule for monitor well-formedness, we simply must show that $\varnothing;\Pi;\{k\};\{\ell'\};j \rhd c$ and $\varnothing;\Pi;\ell' \Vdash v$. Both facts follow from inverting $\varnothing;\Pi;\ell \Vdash {}^\ell B^{\ell'}[\mathsf{parameterize}\ \|^{\ell'\ell'}\mathsf{param/p}^k_j(c,v_p)\| = v\ \mathsf{in}\ {}^{\ell'}B'^\ell[e]]$ as before.

27. $\langle E^\ell[^\ell B^{\ell'}[^{\ell'}\mathsf{param/p}^k_j(c,v_p) := v]],\sigma\rangle \quad \to_{ann} \quad \langle E^\ell[^\ell\langle{}^k v_p := {}^k\mathsf{mon}^{\ell'}_j(c,v)\rangle],\sigma\rangle$: By Lemma 14, it suffices to show that $\varnothing;\Pi;\ell \Vdash {}^\ell\langle{}^k v_p := {}^k\mathsf{mon}^{\ell'}_j(c,v)\rangle$. By the well-formedness rule for parameter assignment, we must show that $\varnothing;\Pi;k \Vdash v_p$ and $\varnothing;\Pi;k \Vdash {}^k\mathsf{mon}^{\ell'}_j(c,v)$. As in the above cases, we use type focusing and inversion to demonstrate that c is not a context contract, and then Lemma 13 and repeated inversions to demonstrate the necessary well-formedness judgments.

28. $\langle E^\ell[\mathsf{check}^k_j(\|^j\#\mathsf{t}\|,e)],\sigma\rangle \quad \to_{ann} \quad \langle E^\ell[e],\sigma\rangle$: By Lemma 14, it suffices to show that $\varnothing;\Pi;\ell \Vdash e$. This follows from the fact that, by Lemma 13, $\varnothing;\Pi;\ell \Vdash \mathsf{check}^k_j(\|^j\#\mathsf{t}\|,e)$.

29. $\langle E^\ell[\mathsf{check}^k_j(\|^j\#\mathsf{f}\|,e)],\sigma\rangle \quad \to_{ann} \quad \langle \mathsf{error}^k_j,\sigma\rangle$: Trivial.

□

Lemma 13 (Label focusing for Annotated CtxPCF)**.** *If* $\ell;\Pi \Vdash E^k[e]$ *then* $\varnothing;\Pi;k \Vdash e$.

Proof. Induction on E^k. □

Lemma 14 (Well-formed evaluation context plugs for Annotated CtxPCF)**.** *If* $\ell;\Pi \Vdash E^k[e]$ *and* $\varnothing;\Pi;k \Vdash e'$ *then* $\ell;\Pi \Vdash E^k[e']$.

Proof. Induction on E^k. □

Lemma 15 (Well-formed substitution for Annotated CtxPCF)**.** *If* $\Delta[x \mapsto \ell]; \Pi; k \Vdash e_1$ *and* $\Delta; \Pi; \ell \Vdash e_2$ *then* $\Delta; \Pi; k \Vdash \{^{|^{\ell} e_2|}/_{x}\} e_1$.

Proof. Mutual induction on well-formedness and well-formed contracts. □

Lemma 16 (Weakening for Well-formedness for Annotated CtxPCF)**.** *If* $\Delta; \Pi; \ell \Vdash e_1$ *and* $r \notin e_1$*, then* $\Delta; \Pi[r \mapsto k]; \ell \Vdash e_1$.

Proof. Mutual induction on well-formedness and well-formed contracts. □

Lemma 17 (Weakening well-formed contracts for Annotated CtxPCF)**.** *If* $\Delta; \Pi; \vec{k}; \vec{\ell}; j \rhd c$*, then* $\Delta; \Pi; \vec{k'}; \vec{\ell'}; j \rhd c$ *for all* $\vec{k'}$ *and* $\vec{\ell'}$ *such that* $\vec{k} \subseteq \vec{k'}$ *and* $\vec{\ell} \subseteq \vec{\ell'}$.

Proof. Induction on well-formed contracts. □

Lemma 18 (Weakening label environments for well-formedness for Annotated CtxPCF)**.** *If* $\Delta; \Pi; \ell \Vdash e$*, then* $\Delta[x \mapsto \ell']; \Pi; \ell \Vdash e$ *for any* x *and* ℓ' *such that* x *is not in the free variables of* e.

Proof. Mutual induction on well-formed contracts and well-formed expressions. □

Lemma 19 (Weakening label environments for well-formed contracts for Annotated CtxPCF)**.** *If* $\Delta; \Pi; \vec{k}; \vec{\ell}; j \rhd c$*, then* $\Delta[x \mapsto \ell']; \Pi; \vec{k}; \vec{\ell}; j \rhd c$ *for any* x *and* ℓ' *such that* x *is not in the free variables of* c.

Proof. Mutual induction on well-formed contracts and well-formed expressions. □

Lemma 20 (Well-formed imports for Annotated CtxPCF)**.** *If for* $\Delta = \{x_1 : \ell, \ldots, x_m : \ell\}$*:*
$\Delta; \Pi \Vdash m_1 \rhd \Delta_1$,
$\ldots$,
$\Delta_{n-1}; \Pi \Vdash m_n \rhd \Delta_n$*, and*
$\Delta_n; \Pi; \ell_0 \Vdash e$*, and also*
$\Delta; \Pi; \ell \Vdash v_1$, $\ldots$, $\Delta; \Pi; \ell \Vdash v_m$ *and* $\Delta; \Pi; \ell \Vdash c_1$, $\ldots$, $\Delta; \Pi; \ell \Vdash v_m$*, then*
$\ell_0; \Pi \Vdash \mathsf{import}[\![\ell, (x_1, \ldots, x_m), (v_1, \ldots, v_m), (c_1, \ldots, c_m), m_1; \ldots; m_n; e]\!]$.

Proof. Induction on the size of $m_1; \ldots; m_n; e$. □

B.6 Surface Ctrl+CtxPCF

B.6.1 Surface Programs

Note. The following extend the syntax of Surface CtxPCF.

$$
\begin{array}{rcl}
e & ::= & \ldots \mid \mathsf{make\text{-}tag} : (\tau \to \tau) \mid \mathsf{reset}\ e\ \mathsf{in}\ e \mid \mathsf{shift}\ e\ \mathsf{as}\ x\ \mathsf{in}\ e \mid \mathsf{tag/c}(e, e, e) \\
c & ::= & \ldots \mid \mathsf{tag/c}(c, c, c) \\
\tau & ::= & \ldots \mid (\tau \to \tau)\ \mathsf{tag}
\end{array}
$$

B.7 Well-typed Surface Programs

Note. The following extend the typing rules of Surface CtxPCF.

$\boxed{\Gamma \vdash e : \tau}$

$$\frac{}{\Gamma \vdash \textsf{make-tag} : (\tau_d \to \tau_r) : (\tau_d \to \tau_r)\ \textsf{tag}}$$

$$\frac{\Gamma \vdash e_t : (\tau_d \to \tau_r)\ \textsf{tag} \qquad \Gamma \vdash e_v : \tau_r}{\Gamma \vdash \textsf{reset}\ e_t\ \textsf{in}\ e_v : \tau_r}$$

$$\frac{\Gamma \vdash e_t : (\tau_d \to \tau_r)\ \textsf{tag} \qquad \Gamma[x \mapsto (\tau_d \to \tau_r)] \vdash e_v : \tau_r}{\Gamma \vdash \textsf{shift}\ e_t\ \textsf{as}\ x\ \textsf{in}\ e_v : \tau_d}$$

$$\frac{\Gamma \vdash e_1 : (\tau_d \to \tau_r)\ \textsf{ctc} \qquad \Gamma \vdash e_2 : \tau_d\ \textsf{ctc} \qquad \Gamma \vdash e_3 : \tau_r\ \textsf{ctc}}{\Gamma \vdash \textsf{tag/c}(e_1, e_2, e_3) : ((\tau_d \to \tau_r)\ \textsf{tag})\ \textsf{ctc}}$$

B.8 Annotated Surface Ctrl+CtxPCF

Note. The following extend the syntax and typing rules of Surface Ctrl+CtxPCF unless stated otherwise.

B.8.1 Annotated Surface Programs

$$e \quad ::= \quad \ldots \mid {}^{\ell}\langle\langle^{\ell} x\rangle\rangle$$

B.8.2 Additional Typing Rules for Annotated Surface Programs

$\boxed{\Gamma \vdash e : \tau}$

$$\frac{\Gamma \vdash x : \tau}{\Gamma \vdash {}^{\ell}\langle\langle^{k} x\rangle\rangle : \tau}$$

B.8.3 Well-formed Annotated Surface Programs

Note. The following extend the well-formedness rules of Annotated Surface CtxPCF.

$\boxed{\Delta; \ell \Vdash e}$

$$\frac{}{\Delta; \ell \Vdash \mathsf{make\text{-}tag} : (\tau_d \to \tau_r)} \qquad \frac{\Delta; \ell \Vdash e_t \qquad \Delta; \ell \Vdash e_v}{\Delta; \ell \Vdash \mathsf{reset}\ e_t\ \mathsf{in}\ e_v}$$

$$\frac{\Delta; \ell \Vdash e_t \qquad \Delta[x \mapsto \ell]; \ell \Vdash e_v}{\Delta; \ell \Vdash \mathsf{shift}\ e_t\ \mathsf{as}\ x\ \mathsf{in}\ e_v} \qquad \frac{\Delta; \ell \Vdash e_i}{\Delta; \ell \Vdash \mathsf{tag/c}(e_1, e_2, e_3)}$$

B.9 Ctrl+CtxPCF

B.9.1 Programs

Note. The following extend the syntax of CtxPCF.

$$
\begin{array}{rcl}
e & ::= & \ldots \mid \mathsf{make\text{-}tag} : (\tau \to \tau) \mid \mathsf{reset}\ e\ \mathsf{in}\ e \mid \mathsf{shift}\ e\ \mathsf{as}\ x\ \mathsf{in}\ e \mid \mathsf{tag/c}(e,e,e) \\
v & ::= & \ldots \mid \mathsf{t}(t) : (\tau \to \tau) \mid {}^{\ell}\mathsf{tag/p}^{k}_{j}(c,c,c,v) \\
c & ::= & \ldots \mid \mathsf{tag/c}(c,c,c) \\
\tau & ::= & \ldots \mid (\tau \to \tau)\ \mathsf{tag}
\end{array}
$$

B.9.2 Well-typed Programs

Note. The following extend the typing rules of CtxPCF.

$\boxed{\Gamma;\Sigma \vdash e : \tau}$

$$\frac{}{\Gamma;\Sigma \vdash \mathsf{make\text{-}tag} : (\tau_d \to \tau_r) : (\tau_d \to \tau_r)\ \mathsf{tag}} \qquad \frac{\Gamma;\Sigma \vdash e_t : (\tau_d \to \tau_r)\ \mathsf{tag} \qquad \Gamma;\Sigma \vdash e_v : \tau_r}{\Gamma;\Sigma \vdash \mathsf{reset}\ e_t\ \mathsf{in}\ e_v : \tau_r}$$

$$\frac{\Gamma;\Sigma \vdash e_t : (\tau_d \to \tau_r)\ \mathsf{tag} \qquad \Gamma[x \mapsto (\tau_d \to \tau_r)];\Sigma \vdash e_v : \tau_r}{\Gamma;\Sigma \vdash \mathsf{shift}\ e_t\ \mathsf{as}\ x\ \mathsf{in}\ e_v : \tau_d}$$

$$\frac{}{\Delta;\ell \Vdash (\tau_d \to \tau_r)\ \mathsf{tag}}$$

$$\frac{\Gamma;\Sigma \vdash c_1 : (\tau_d \to \tau_r)\ \mathsf{ctc} \qquad \Gamma;\Sigma \vdash c_2 : \tau_d\ \mathsf{ctc} \qquad \Gamma;\Sigma \vdash c_3 : \tau_r\ \mathsf{ctc} \qquad \Gamma;\Sigma \vdash v : (\tau_d \to \tau_r)\ \mathsf{tag}}{\Gamma;\Sigma \vdash {}^{\ell}\mathsf{tag/p}^{k}_{j}(c_1,c_2,c_3,v) : (\tau_d \to \tau_r)\ \mathsf{tag}}$$

$$\frac{\Gamma;\Sigma \vdash e_1 : (\tau_d \to \tau_r)\ \mathsf{ctc} \qquad \Gamma;\Sigma \vdash e_2 : \tau_d\ \mathsf{ctc} \qquad \Gamma;\Sigma \vdash e_3 : \tau_r\ \mathsf{ctc}}{\Gamma;\Sigma \vdash \mathsf{tag/c}(e_1,e_2,e_3) : ((\tau_d \to \tau_r)\ \mathsf{tag})\ \mathsf{ctc}}$$

$\boxed{\Gamma;\Sigma \vdash \sigma}$

$$\frac{\mathrm{dom}(\Sigma) = \mathrm{dom}(\sigma) \qquad \forall r \in \mathrm{dom}(\sigma).\Gamma;\Sigma \vdash \Sigma(r) : \sigma(r) \qquad \forall t \in \mathrm{dom}(\sigma).\Gamma;\Sigma \vdash \Sigma(t) : \mathsf{Unit}}{\Gamma;\Sigma \vdash \sigma}$$

B.9.3 Evaluation Contexts

Note. The following extend the evaluation contexts of CtxPCF.

$$
\begin{array}{rcl}
E & ::= & \ldots \mid \mathsf{reset}\ E\ \mathsf{in}\ e \mid \mathsf{reset}\ v\ \mathsf{in}\ E \mid \mathsf{shift}\ E\ \mathsf{as}\ x\ \mathsf{in}\ e \\
 & \mid & \mathsf{tag/c}(E,e,e) \mid \mathsf{tag/c}(c,E,e) \mid \mathsf{tag/c}(c,c,E) \\
{}^{\ell}T^{k} & ::= & [\cdot] \mid {}^{\ell}\mathsf{tag/p}^{\ell'}_{j}(c_1,c_2,c_3,{}^{\ell'}T^{k})
\end{array}
$$

B.9.4 Reduction Semantics

Note. The following extend the reduction rules of CtxPCF.

$$\langle E[\mathsf{make\text{-}tag} : (\tau_1 \to \tau_2)], \sigma\rangle \quad \to \quad \langle E[\mathsf{t}(t) : (\tau_1 \to \tau_2)], \sigma[t \mapsto ()]\rangle \text{ where } t \text{ is fresh in } \sigma$$

$$\langle E[\mathsf{reset}\ {}^{\ell}T^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\ \mathsf{in}\ v], \sigma\rangle \quad \to \quad \langle E[v], \sigma\rangle$$

$$\begin{aligned}&\langle E[\mathsf{reset}\ {}^{\ell}T^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\ \mathsf{in}\ E'[\mathsf{shift}\ {}^{\ell'}T'^{k'}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\ \mathsf{as}\ x\ \mathsf{in}\ e]], \sigma\rangle \\ &\quad \to \quad \langle E[\mathsf{reset}\ {}^{\ell}T^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\ \mathsf{in}\ \{{}^{v'}/_{x}\}e'], \sigma\rangle \\ &\qquad\qquad \text{where } v_k = \lambda\, y : \tau_d.\ \mathsf{reset}\ {}^{\ell}T^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\ \mathsf{in}\ E'[e_y] \\ &\qquad\qquad \text{and } v' = \mathrm{wrap}_1^{+}[\![{}^{\ell'}T'^{k'}[\mathsf{t}(t) : (\tau_d \to \tau_r)], \mathrm{wrap}_1^{-}[\![{}^{\ell}T^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)], v_k]\!]\!] \\ &\qquad\qquad \text{and } e_y = \mathrm{wrap}_2^{+}[\![{}^{\ell'}T'^{k'}[\mathsf{t}(t) : (\tau_d \to \tau_r)], \mathrm{wrap}_2^{-}[\![{}^{\ell}T^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)], y]\!]\!] \\ &\qquad\qquad \text{and } e' = \mathrm{wrap}_3^{+}[\![{}^{\ell}T^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)], \mathrm{wrap}_3^{-}[\![{}^{\ell'}T'^{k'}[\mathsf{t}(t) : (\tau_d \to \tau_r)], e]\!]\!] \\ &\qquad\qquad \text{and } E' \text{ does not contain } \mathsf{reset}\ {}^{\ell''}T''^{k''}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\ \mathsf{in}\ E''\end{aligned}$$

$$\langle E[{}^{\ell}\mathsf{mon}_j^{k}(\mathsf{tag/c}(c_1, c_2, c_3), v)], \sigma\rangle \quad \to \quad \langle E[{}^{\ell}\mathsf{tag/p}_j^{k}(c_1, c_2, c_3, v)], \sigma\rangle$$

$$\begin{aligned}
\mathrm{wrap}_1^{+}[\![\mathsf{t}(t) : (\tau_d \to \tau_r), e]\!] &= e \\
\mathrm{wrap}_1^{+}[\![{}^{\ell}\mathsf{tag/p}_j^{k}(c_1, c_2, c_3, {}^{\ell'}T^{k'}[\mathsf{t}(t) : (\tau_d \to \tau_r)]), e]\!] &= {}^{\ell}\mathsf{mon}_j^{k}(c_1, \mathrm{wrap}_1^{+}[\![{}^{\ell'}T^{k'}[\mathsf{t}(t) : (\tau_d \to \tau_r)], e]\!]) \\
\mathrm{wrap}_1^{-}[\![\mathsf{t}(t) : (\tau_d \to \tau_r), e]\!] &= e \\
\mathrm{wrap}_1^{-}[\![{}^{\ell}\mathsf{tag/p}_j^{k}(c_1, c_2, c_3, {}^{\ell'}T^{k'}[\mathsf{t}(t) : (\tau_d \to \tau_r)]), e]\!] &= {}^{k}\mathsf{mon}_j^{\ell}(c_1, \mathrm{wrap}_1^{-}[\![{}^{\ell'}T^{k'}[\mathsf{t}(t) : (\tau_d \to \tau_r)], e]\!]) \\
\mathrm{wrap}_2^{+}[\![\mathsf{t}(t) : (\tau_d \to \tau_r), e]\!] &= e \\
\mathrm{wrap}_2^{+}[\![{}^{\ell}\mathsf{tag/p}_j^{k}(c_1, c_2, c_3, {}^{\ell'}T^{k'}[\mathsf{t}(t) : (\tau_d \to \tau_r)]), e]\!] &= {}^{\ell}\mathsf{mon}_j^{k}(c_2, \mathrm{wrap}_2^{+}[\![{}^{\ell'}T^{k'}[\mathsf{t}(t) : (\tau_d \to \tau_r)], e]\!]) \\
\mathrm{wrap}_2^{-}[\![\mathsf{t}(t) : (\tau_d \to \tau_r), e]\!] &= e \\
\mathrm{wrap}_2^{-}[\![{}^{\ell}\mathsf{tag/p}_j^{k}(c_1, c_2, c_3, {}^{\ell'}T^{k'}[\mathsf{t}(t) : (\tau_d \to \tau_r)]), e]\!] &= {}^{k}\mathsf{mon}_j^{\ell}(c_2, \mathrm{wrap}_2^{-}[\![{}^{\ell'}T^{k'}[\mathsf{t}(t) : (\tau_d \to \tau_r)], e]\!]) \\
\mathrm{wrap}_3^{+}[\![\mathsf{t}(t) : (\tau_d \to \tau_r), e]\!] &= e \\
\mathrm{wrap}_3^{+}[\![{}^{\ell}\mathsf{tag/p}_j^{k}(c_1, c_2, c_3, {}^{\ell'}T^{k'}[\mathsf{t}(t) : (\tau_d \to \tau_r)]), e]\!] &= {}^{\ell}\mathsf{mon}_j^{k}(c_3, \mathrm{wrap}_3^{+}[\![{}^{\ell'}T^{k'}[\mathsf{t}(t) : (\tau_d \to \tau_r)], e]\!]) \\
\mathrm{wrap}_3^{-}[\![\mathsf{t}(t) : (\tau_d \to \tau_r), e]\!] &= e \\
\mathrm{wrap}_3^{-}[\![{}^{\ell}\mathsf{tag/p}_j^{k}(c_1, c_2, c_3, {}^{\ell'}T^{k'}[\mathsf{t}(t) : (\tau_d \to \tau_r)]), e]\!] &= {}^{k}\mathsf{mon}_j^{\ell}(c_3, \mathrm{wrap}_3^{-}[\![{}^{\ell'}T^{k'}[\mathsf{t}(t) : (\tau_d \to \tau_r)], e]\!])
\end{aligned}$$

B.10 Annotated Ctrl+CtxPCF

Note. The following extend the syntax and typing rules of Ctrl+CtxPCF unless stated otherwise.

B.10.1 Annotated Ctrl+CtxPCF Programs

$$
\begin{array}{rcl}
e & ::= & \ldots \mid {}^{\ell}\langle\langle {}^{\ell}x\rangle\rangle \mid |^{\ell}e| \mid {}^{\ell}\langle {}^{\ell}e\rangle \mid \lfloor e \rfloor^{\vec{\ell}} \\
v & ::= & \ldots \mid |^{\ell}v| \mid {}^{\ell}\langle {}^{\ell}v\rangle \\
c & ::= & \ldots \mid |^{\ell}c| \mid \lfloor |^{\ell}\ldots|^{\ell}\mathsf{flat/c}(v)|\ldots|\rfloor^{\vec{\ell}} \mid \lfloor |^{\ell}\ldots|^{\ell}\mathsf{ctx/c}(v,(v \Rightarrow v \leftarrow v)\ldots,v,(v \Rightarrow v \leftarrow v)\ldots)|\ldots|\rfloor^{\vec{\ell}}
\end{array}
$$

B.10.2 Additional Typing Rules for Annotated Ctrl+CtxPCF Programs

$\boxed{\Gamma;\Sigma \vdash e : \tau}$

$$\frac{\Gamma;\Sigma \vdash x : \tau}{\Gamma;\Sigma \vdash {}^{\ell}\langle\langle {}^{k}x\rangle\rangle : \tau} \qquad \frac{\Gamma;\Sigma \vdash e : \tau}{\Gamma;\Sigma \vdash |^{\ell}e| : \tau} \qquad \frac{\Gamma;\Sigma \vdash e : \tau}{\Gamma;\Sigma \vdash {}^{\ell}\langle {}^{k}e\rangle : \tau} \qquad \frac{\Gamma;\Sigma \vdash e : \tau}{\Gamma;\Sigma \vdash \lfloor e \rfloor^{\vec{\ell}} : \tau}$$

B.10.3 Annotated Evaluation Contexts

Note. The following extend the evaluation contexts of Annotated Ctrl+CtxPCF.

$$
\begin{array}{rcl}
F & ::= & \ldots \mid \mathsf{reset}\ F\ \mathsf{in}\ e \mid \mathsf{reset}\ v\ \mathsf{in}\ F \mid \mathsf{shift}\ F\ \mathsf{as}\ x\ \mathsf{in}\ e \\
 & \mid & \mathsf{tag/c}(F,e,e) \mid \mathsf{tag/c}(c,F,e) \mid \mathsf{tag/c}(c,c,F) \\
F^{\ell} & ::= & \ldots \mid \mathsf{reset}\ F^{\ell}\ \mathsf{in}\ e \mid \mathsf{reset}\ v\ \mathsf{in} \mid \mathsf{shift}\ F^{\ell}\ \mathsf{as}\ x\ \mathsf{in}\ e \\
 & \mid & \mathsf{tag/c}(F^{\ell},e,e) \mid \mathsf{tag/c}(c,F^{\ell},e) \mid \mathsf{tag/c}(c,c,F^{\ell}) \\
{}^{\ell}T^{k} & ::= & [\cdot] \mid {}^{\ell}\mathsf{tag/p}^{\ell'}_{j}(c_1,c_2,c_3,\|^{\ell'\,\ell'}T^{k'}\|)
\end{array}
$$

B.10.4 Annotated Reduction Semantics

Note. The following extend the reduction rules of Annotated CtxPCF.

$$\langle E^{\ell}[\mathsf{make\text{-}tag} : (\tau_1 \to \tau_2)], \sigma\rangle \quad \to_{ann} \quad \langle E^{\ell}[|^{\ell}\mathsf{t}(t) : (\tau_1 \to \tau_2)|], \sigma[t \mapsto ()]\rangle \text{ where } t \text{ is fresh } \sigma$$

$$\langle E^{\ell}[\mathsf{reset}\ \|^{\ell\ell}T^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{in}\ v], \sigma\rangle \quad \to_{ann} \quad \langle E^{\ell}[v], \sigma\rangle$$

$$
\begin{array}{l}
\langle E^{\ell}[\mathsf{reset}\ \|^{\ell\ell}T^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{in}\ E'^{\ell'}[\mathsf{shift}\ \|^{\ell'\,\ell'}T'^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{as}\ x\ \mathsf{in}\ e]], \sigma\rangle \\
\quad \to_{ann} \quad \langle E^{\ell}[\mathsf{reset}\ {}^{\ell}T^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\ \mathsf{in}\ \{{}^{v'}/_{x}\}e'], \sigma\rangle \\
\qquad\qquad \text{where } v_k = |^{\ell}\lambda\, y : \tau_d.\ \mathsf{reset}\ {}^{\ell}T^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\ \mathsf{in}\ E'^{\ell'}[e_y]| \\
\qquad\qquad \text{and } v' = |^{\ell'}\mathrm{wrap}_1^{+}[\![^{\ell'}T'^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)], \mathrm{wrap}_1^{-}[\![^{\ell}T^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)], v_k]\!]]| \\
\qquad\qquad \text{and } e_y = |^{\ell'}\mathrm{wrap}_2^{+}[\![^{\ell'}T'^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)], \mathrm{wrap}_2^{-}[\![^{\ell}T^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)], y]\!]]| \\
\qquad\qquad \text{and } e' = |^{\ell}\mathrm{wrap}_3^{+}[\![^{\ell}T^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)], \mathrm{wrap}_3^{-}[\![^{\ell'}T'^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)], e]\!]]| \\
\qquad\qquad \text{and } E'^{\ell'} \text{ does not contain } \mathsf{reset}\ \|^{\ell''\,\ell''}T''^{k''}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{in}\ E''^{\ell''''}
\end{array}
$$

$$\langle E^{\ell}[{}^{\ell}\mathsf{mon}^{k}_{j}(\|^{j}\mathsf{tag/c}(c_1,c_2)c_3\|, v)], \sigma\rangle \quad \to_{ann} \quad \langle E^{\ell}[{}^{\ell}\mathsf{tag/p}^{k}_{j}(c_1,c_2,c_3)v], \sigma\rangle$$

Note. The following metafunction replaces that of Annotated CtxPCF.

$$
\begin{array}{lcl}
\mathrm{obl}[\![\|^{\ell}\mathsf{flat/c}(v)\|,\vec{k},\vec{\ell},j]\!] & = & \lfloor\|^{\ell}\mathsf{flat/c}(v)\|\rfloor^{\vec{k}} \\
\mathrm{obl}[\![\|^{\ell}\mathsf{param/c}(c)\|,\vec{k},\vec{\ell},j]\!] & = & \|^{\ell}\mathsf{param/c}(\mathrm{obl}[\![c,\vec{k}\vec{\ell},\vec{k}\vec{\ell},j]\!])\| \\
\mathrm{obl}[\![\|^{\ell}c_d : \tau \to (c_c)\ c_r\|,\vec{k},\vec{\ell},j]\!] & = & \|^{\ell}\mathrm{obl}[\![c_d,\vec{\ell},\vec{k},j]\!] : \tau \to (\mathrm{obl}[\![c_c,\vec{k},\vec{\ell},j]\!])\ \mathrm{obl}[\![c_r,\vec{k},\vec{\ell},j]\!]\| \\
\mathrm{obl}[\![\|^{\ell}c_d : \tau \to_{\mathsf{a}} (\lambda\, x : \tau.\ e)\ c_r\|,\vec{k},\vec{\ell},j]\!] & = & \|^{\ell}\mathrm{obl}[\![c_d,\vec{\ell}j,\vec{k},j]\!] : \tau \to_{\mathsf{a}} (\lambda\, x : \tau.\ \lfloor e\rfloor^{\vec{\ell}})\ \mathrm{obl}[\![c_r,\vec{k},\vec{\ell},j]\!]\| \\
\mathrm{obl}[\![\|^{\ell}\mathsf{ctx/c}(v,(v \Rightarrow v \leftarrow v)\ldots,v,(v \Rightarrow v \leftarrow v)\ldots)\|,\vec{k},\vec{\ell},j]\!] & = & \lfloor\|^{\ell}\mathsf{ctx/c}(v,(v \Rightarrow v \leftarrow v)\ldots,v,(v \Rightarrow v \leftarrow v)\ldots)\|\rfloor^{\vec{\ell}} \\
\mathrm{obl}[\![\|^{\ell}\mathsf{tag/c}(c_1,c_2,c_3)\|,\vec{k},\vec{\ell},j]\!] & = & \|^{\ell}\mathsf{tag/c}(\mathrm{obl}[\![c_1,\vec{k}\vec{\ell},\vec{k}\vec{\ell},j]\!],\mathrm{obl}[\![c_2,\vec{k}\vec{\ell},\vec{k}\vec{\ell},j]\!],\mathrm{obl}[\![c_3,\vec{k}\vec{\ell},\vec{k}\vec{\ell},j]\!])\|
\end{array}
$$

B.10.5 Well-formed Annotated Programs

Note. The following extend the well-formedness rules of Annotated CtxPCF.

$\boxed{\Delta; \Pi; \ell \Vdash e}$

$$\frac{\Delta; \Pi; \ell \Vdash e_1 \qquad \Delta; \Pi; \ell \Vdash e_2}{\Delta; \Pi; \ell \Vdash \mathsf{reset}\ e_1\ \mathsf{in}\ e_2} \qquad \frac{\Delta; \Pi; \ell \Vdash e_1 \qquad \Delta[x \mapsto \ell]; \Pi; \ell \Vdash e_2}{\Delta; \Pi; \ell \Vdash \mathsf{shift}\ e_1\ \mathsf{as}\ x\ \mathsf{in}\ e_2} \qquad \frac{\Delta; \Pi; \ell \Vdash e_i}{\Delta; \Pi; \ell \Vdash \mathsf{tag/c}(e_1, e_2, e_3)}$$

$$\frac{\Delta; \Pi; \vec{k}\vec{\ell}; \vec{k}\vec{\ell}; j \rhd c_i \qquad \Delta; \Pi; k \Vdash v}{\Delta; \Pi; \ell \Vdash {}^{\ell}\mathsf{tag/p}^{k}_{j}(c_1, c_2, c_3, v)}$$

$\boxed{\Delta; \Pi \vdash \sigma}$

$$\frac{\mathrm{dom}(\Pi) = \mathrm{dom}(\sigma) \qquad \forall r \in \mathrm{dom}(\sigma), \Delta; \Pi; \Pi(r) \Vdash \sigma(r) \qquad \forall t \in \mathrm{dom}(\sigma), \Delta; \Pi; \Pi(t) \Vdash \sigma(t)}{\Delta; \Pi \Vdash \sigma}$$

$\boxed{\Delta; \Pi; \vec{k}; \vec{\ell}; j \rhd c}$

$$\frac{\Delta; \Pi; \vec{k}\vec{\ell}; \vec{k}\vec{\ell}; j \rhd c_i}{\Delta; \Pi; \vec{k}; \vec{\ell}; j \rhd \|^{j}\mathsf{tag/c}(\|^{j} c_1\|, \|^{j} c_2\|, \|^{j} c_3\|)\|}$$

B.11 Metatheory of Ctrl+CtxPCF

Theorem 5 (Type Soundness for Surface Ctrl+CtxPCF)**.** *For all programs p of Surface Ctrl+CtxPCF such that $\vdash p : \tau$*

- $\langle p, \varnothing \rangle \rightarrow^{*} \langle v, \sigma \rangle$ *or;*
- $\langle p, \varnothing \rangle \rightarrow^{*} \langle \mathsf{error}^{k}_{j}, \sigma \rangle$ *or;*
- $\langle p, \varnothing \rangle \rightarrow^{*} \langle p', \sigma \rangle$ *where* $p' = E[\mathsf{shift}^{\ell'} T'^{k'}\ [t(t) : (\tau_d \rightarrow \tau_r)]\ \mathsf{as}\ x\ \mathsf{in}\ e]$ *where E does not contain* $\mathsf{reset}^{\ell''} T''^{k''}\ [t(t) : (\tau_d \rightarrow \tau_r)]\ \mathsf{in}\ E'$ *or;*
- $\langle p, \varnothing \rangle \rightarrow^{*} \langle p', \sigma \rangle$ *and there exist p'' such that $p' \rightarrow p''$.*

Proof. Direct consequence of Theorems 21 and 22. □

Lemma 21 (Type Progress for Ctrl+CtxPCF Programs)**.** *If $\Sigma \vdash p : \tau$, then either p is a value v, p is* error^{k}_{j}*, p is* $E[\mathsf{shift}^{\ell'} T'^{k'}\ [t(t) : (\tau_d \rightarrow \tau_r)]\ \mathsf{as}\ x\ \mathsf{in}\ e]$ *where E does not contain* $\mathsf{reset}^{\ell''} T''^{k''}\ [t(t) : (\tau_d \rightarrow \tau_r)]\ \mathsf{in}\ E'$*, or for all σ such that $\varnothing; \Sigma \vdash \sigma$, there exist p' and σ' such that $\langle p, \sigma \rangle \rightarrow \langle p', \sigma' \rangle$.*

Proof. By straight-forward case analysis on p using Lemmas 23, 24, and 25. □

Lemma 22 (Type Preservation for Ctrl+CtxPCF Programs)**.** *If $\Sigma \vdash p : \tau$, $\varnothing; \Sigma \vdash \sigma$, and $\langle p, \sigma \rangle \rightarrow \langle p', \sigma' \rangle$, then either there exists $\Sigma' \supseteq \Sigma$ such that $\Sigma' \vdash p' : \tau$ and $\Gamma; \Sigma' \vdash \sigma'$, or $p' = \mathsf{error}^{k}_{j}$.*

Proof. By straight-forward case analysis on the rules of the reduction semantics using Lemma 23 and Lemma 24. □

Lemma 23 (Unique decomposition for Ctrl+CtxPCF Programs)**.** *For all well-typed programs p such that $p \neq v$, $p \neq \mathsf{error}^{k}_{j}$, and $p \neq \mathsf{module}\ \ell\ \mathsf{exports}\ x_{v_1}\ \mathsf{with}\ x_{c_1}, \ldots\ \mathsf{where}\ y_1 = e_1, \ldots, y_n = e_n, \ldots; p'$, there are unique e', and E such that $p = E[e']$ and e' is one of the following:*

1. *one of the corresponding cases of the statement of Lemma 3*
2. *make-tag* $: (\tau_1 \rightarrow \tau_2)$
3. $\mathsf{reset}^{\ell} T^{k}[t(t) : (\tau_d \rightarrow \tau_r)]\ \mathsf{in}\ v$
4. $\mathsf{reset}^{\ell} T^{k}[t(t) : (\tau_d \rightarrow \tau_r)]\ \mathsf{in}\ E'[\mathsf{shift}^{\ell'} T'^{k'}\ [t(t) : (\tau_d \rightarrow \tau_r)]\ \mathsf{as}\ x\ \mathsf{in}\ e]$ *where E' does not contain* $\mathsf{reset}^{\ell''} T''^{k''}\ [t(t) : (\tau_d \rightarrow \tau_r)]\ \mathsf{in}\ E''$

5. $\mathsf{shift}^{\ell'} T'^{k'}[\mathsf{t}(t) : (\tau_d \to \tau_r)]$ as x in e *where* E *does not contain* $\mathsf{reset}^{\ell''} T''^{k''}[\mathsf{t}(t) : (\tau_d \to \tau_r)]$ in E'
6. ${}^{\ell}\mathsf{mon}_j^k(\mathsf{tag/c}(c_1, c_2, c_3), v)$

Proof. By induction on the size of p. □

Lemma 24 (Well-typed evaluation context plugs for Ctrl+CtxPCF)**.** *If* $\Sigma \vdash E[e] : \tau$, $\varnothing; \Pi \vdash e : \tau'$, *and* $\varnothing; \Pi \vdash e' : \tau'$ *then* $\Sigma \vdash E[e'] : \tau$.

Proof. Induction on E. □

Lemma 25 (Type focusing for Ctrl+CtxPCF)**.** *If* $\Sigma \vdash E[e] : \tau$ *then* $\varnothing; \Sigma \vdash e : \tau'$ *for some* τ'.

Proof. Induction on E. □

Theorem 6 (Type Soundness for Annotated Surface Ctrl+CtxPCF)**.** *For all programs* p *of Annotated Surface Ctrl+CtxPCF such that* $\vdash p : \tau$
- $\langle p, \varnothing \rangle \to^*_{ann} \langle v, \sigma \rangle$ *or;*
- $\langle p, \varnothing \rangle \to^*_{ann} \langle \mathsf{error}_j^k, \sigma \rangle$ *or,*
- $\langle p, \varnothing \rangle \to^*_{ann} \langle p', \sigma \rangle$ *and* $p' = E^{\ell}[\mathsf{shift}\ \|^{\ell^\dagger\ \ell'} T'^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{as}\ x\ \mathsf{in}\ e]$ *where* E^{ℓ} *does not contain* $\mathsf{reset}\ \|^{\ell'''\ \ell''} T'''^{k''}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{in}\ E'^{\ell''''}$ *or;*
- $\langle p, \varnothing \rangle \to^*_{ann} \langle p', \sigma \rangle$ *and there exist* p'' *such that* $p' \to p''$.

Proof. Direct consequence of Theorems 26 and 27. □

Lemma 26 (Type Progress for Annotated Ctrl+CtxPCF Terms)**.** *If* $\Sigma \vdash p : \tau$, *then either* p *is a value* v, p *is* error_j^k, p *is* $E^l[\mathsf{shift}\ \|^{\ell^\dagger\ \ell'} T'^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{as}\ x\ \mathsf{in}\ e]$ *where* E^l *does not contain* $\mathsf{reset}\ \|^{\ell'''\ \ell''} T'''^{k''}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{in}\ E'^{\ell''''}$, *or for all* σ *such that* $\varnothing; \Sigma \vdash \sigma$, *there exist* p' *and* σ' *such that* $\langle p, \sigma \rangle \to \langle p', \sigma' \rangle$.

Proof. By straight-forward case analysis on p using Lemmas 28, 29, and 30. □

Lemma 27 (Type Preservation for Annotated Ctrl+CtxPCF Terms)**.** *If* $\Sigma \vdash p : \tau$, $\varnothing; \Sigma \vdash \sigma$, *and* $\langle p, \sigma \rangle \to_{ann} \langle p', \sigma' \rangle$, *then either there exists* $\Sigma' \supseteq \Sigma$ *such that* $\Sigma' \vdash p' : \tau$ *and* $\varnothing; \Sigma' \vdash \sigma'$, *or* $p' = \mathsf{error}_j^k$.

Proof. By straight-forward case analysis on the rules of the reduction semantics using Lemma 28 and Lemma 29. □

Lemma 28 (Unique decomposition for Annotated Ctrl+CtxPCF)**.** *For all well-typed programs* p *such that* $p \neq v$, $p \neq \mathsf{error}_j^k$, *and* $p \neq \mathsf{module}\ \ell\ \mathsf{exports}\ x_{v_1}\ \mathsf{with}\ x_{c_1}, \ldots\ \mathsf{where}\ y_1 = e_1, \ldots, y_n = e_n, \ldots; p'$, *there are unique* e', ℓ', *and* $E^{\ell'}$ *such that* $p = E^{\ell'}[e']$ *and* e' *is one of the following:*
1. *one of the corresponding cases of the statement of Lemma 8*
2. $\mathsf{make\text{-}tag} : (\tau_1 \to \tau_2)$
3. $\mathsf{reset}\ \|^{\ell^*\ \ell} T^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{in}\ v$
4. $\mathsf{reset}\ \|^{\ell^*\ \ell} T^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{in}\ E'^{\ell'}[\mathsf{shift}\ \|^{\ell^\dagger\ \ell'} T'^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{as}\ x\ \mathsf{in}\ e]$ *where* E' *does not contain* $\mathsf{reset}\ \|^{\ell'''\ \ell''} T'''^{k''}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{in}\ E''^{\ell''''}$
5. $\mathsf{shift}\ \|^{\ell^\dagger\ \ell'} T'^{k}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{as}\ x\ \mathsf{in}\ e$ *where* E^l *does not contain* $\mathsf{reset}\ \|^{\ell'''\ \ell''} T'''^{k''}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{in}\ E''^{\ell''''}$
6. ${}^{\ell}\mathsf{mon}_j^k(\|^{l^*}\ \mathsf{tag/c}(c_1, c_2)c_3\|, v)$

Proof. By induction on the size of p. □

Lemma 29 (Well-typed evaluation context plugs for Annotated Ctrl+CtxPCF)**.** *If* $\Sigma \vdash E^k[e] : \tau$, $\varnothing; \Pi \vdash e : \tau'$, *and* $\varnothing; \Pi \vdash e' : \tau'$ *then* $\Sigma \vdash E^k[e'] : \tau$.

Proof. Induction on E^k. □

Lemma 30 (Type focusing for Annotated Ctrl+CtxPCF)**.** *If* $\Sigma \vdash E^k[e] : \tau$ *then* $\varnothing; \Sigma \vdash e : \tau'$ *for some* τ'.

Proof. Induction on E^k. □

Theorem 7 (Annotation transparency)**.** *The following statements hold for Surface Ctrl+CtxPCF:*

1. *Let e be a well-typed and well-formed Annotated Surface Ctrl+CtxPCF program: $\vdash p : \tau$ and $\ell_0 \Vdash p$. Let $\bar{p}$ be the Surface Ctrl+CtxPCF expression that is like p but without annotations. If $\langle p, \varnothing\rangle \rightarrow^*_{ann} \langle p', \sigma\rangle$, then $\langle \bar{p}, \varnothing\rangle \rightarrow^*_{ann} \langle \bar{p'}, \bar{\sigma}\rangle$, where $\bar{p'}$ is the Ctrl+CtxPCF program that is like p' but without annotations.*
2. *Let $\bar{p}$ be a well-typed Surface Ctrl+CtxPCF program: $\vdash \bar{p} : \tau$. Then there exists some Annotated Surface Ctrl+CtxPCF program p such that $\vdash p : \tau$ and $\ell_0 \Vdash p$. Furthermore, if $\langle \bar{p}, \varnothing\rangle \rightarrow^*_{ann} \langle \bar{p'}, \bar{\sigma}\rangle$, then $\langle p, \varnothing\rangle \rightarrow^*_{ann} \langle p', \sigma\rangle$ for some p', where $\bar{p'}$ is the Ctrl+CtxPCF program that is like p' but without annotations.*

Proof. By a straightforward lock-step bi-simulation between the two reduction steps using the obvious annotation erasure function to relate the corresponding configurations in each step. For the second point of the theorem's statement, we construct p from $\bar{p}$ by adding to each imported variable in a term of $\bar{p}$ the appropriate annotation. □

Definition 2 (Complete monitoring)**.** *Ctrl+CtxPCF is a complete monitor if any only if for all p such that $\vdash p_0 : \tau$ and $\ell_0; \Pi \Vdash p$,*

- $\langle p_0, \varnothing\rangle \rightarrow^*_{ann} \langle v, \sigma\rangle$, *or*
- $\langle p, \varnothing\rangle \rightarrow^*_{ann} \langle p', \sigma\rangle$ *and* $p' = E^{\ell}[\mathsf{shift}\ \|^{\ell^\dagger\,\ell'} T'^{k}[\mathsf{t}(t) : (\tau_d \rightarrow \tau_r)]\|\ \mathsf{as}\ x\ \mathsf{in}\ e]$ *where E^{ℓ} does not contain* $\mathsf{reset}\ \|^{\ell'''\,\ell''} T''^{k''}[\mathsf{t}(t) : (\tau_d \rightarrow \tau_r)]\|\ \mathsf{in}\ E'^{\ell''''}$, *or*
- *for all p_1 and σ_1 such that $\langle p_0, \varnothing\rangle \rightarrow^*_{ann} \langle p_1, \sigma_1\rangle$ there exists p_2 and σ_2 such that $\langle p_1, \sigma_1\rangle \rightarrow_{ann} \langle p_2, \sigma_2\rangle$, or*
- $\langle p_0, \varnothing\rangle \rightarrow_{ann} \langle p_1, \sigma_1\rangle \rightarrow^*_{ann} \langle \mathsf{error}^k_j, \sigma_2\rangle$ *and p_1 is of the form $E^{\ell}[{}^{\ell}\mathsf{mon}^k_j(\lfloor \mathsf{flat/c}(p)\rfloor^{\vec{\ell}}, v)]$ and for all such terms p_1, $v = \|^k v'\|$ and $k \in \vec{\ell}$, or*
- $\langle p_0, \varnothing\rangle \rightarrow_{ann} \langle p_1, \sigma_1\rangle \rightarrow^*_{ann} \langle \mathsf{error}^{\ell}_j, \sigma_2\rangle$ *and p_1 is of the form*

$$E^{\ell}[{}^{\ell}\mathsf{mon}^k_j(\lfloor \mathsf{ctx/c}(v_c, (v_{g_{c_1}} \Rightarrow v_{p_{c_1}} \leftarrow v_{v_{c_1}}), \ldots, v_a, (v_{g_{a_1}} \Rightarrow v_{p_{a_1}} \leftarrow v_{v_{a_1}}), \ldots)\rfloor^{\vec{\ell}}, v)]$$

and for all such terms p_1, $\ell \in \vec{\ell}$.

Theorem 8. *Ctrl+CtxPCF is a complete monitor.*

Proof. As a direct consequence of Lemmas 26, 27, 31, and 32, we have that for all programs p_0 such that $\vdash p_0 : \tau$ and $\ell_0 \Vdash p_0$, either

- $\langle p_0, \varnothing\rangle \rightarrow^*_{ann} \langle v, \sigma\rangle$, or
- $\langle p, \varnothing\rangle \rightarrow^*_{ann} \langle p', \sigma\rangle$ and $p' = E^{\ell}[\mathsf{shift}\ \|^{\ell^\dagger\,\ell'} T'^{k}[\mathsf{t}(t) : (\tau_d \rightarrow \tau_r)]\|\ \mathsf{as}\ x\ \mathsf{in}\ e]$ where E^{ℓ} does not contain $\mathsf{reset}\ \|^{\ell'''\,\ell''} T''^{k''}[\mathsf{t}(t) : (\tau_d \rightarrow \tau_r)]\|\ \mathsf{in}\ E'^{\ell''''}$, or
- for all p_1 and σ_1 such that $\langle p_0, \varnothing\rangle \rightarrow^*_{ann} \langle p_1, \sigma_1\rangle$ there exists p_2 and σ_2 such that $\langle p_1, \sigma_1\rangle \rightarrow_{ann} \langle p_2, \sigma_2\rangle$, or
- $\langle p_0, \varnothing\rangle \rightarrow^*_{ann} \langle \mathsf{error}^k_j, \sigma_2\rangle$.

In the last case, since $\ell_0 \Vdash p_0$, we know that error^k_j does not appear in p_0. Therefore it must have been introduced by a reduction. In particular, it must have been the result of the reduction $\langle E^{\ell}[\mathsf{check}^k_j(\|^j \#\mathsf{f}\|, e)], \sigma_2\rangle \rightarrow_{ann} \langle \mathsf{error}^k_j, \sigma_2\rangle$. Again, since $\ell_0 \Vdash p_0$, $\mathsf{check}^k_j(\|^j \#\mathsf{f}\|, e)$ must not occur in p_0. Hence it must be the result of a reduction. There are three rules that introduce checks:

1. $\langle E^{\ell}[{}^{\ell}\mathsf{mon}^k_j(\lfloor \|^j \mathsf{flat/c}(v_c)\|\rfloor^{\vec{k}}, \|^k v\|)], \sigma\rangle \rightarrow_{ann} \langle E^{\ell}[\mathsf{check}^k_j((v_c\ v), v)], \sigma\rangle$: In this case, we can deduce

$$\begin{aligned}
&\langle p_0, \varnothing\rangle \rightarrow^*_{ann} \langle E^{\ell}[{}^{\ell}\mathsf{mon}^k_j(\lfloor \|^j \mathsf{flat/c}(v_c)\|\rfloor^{\vec{k}}, \|^k v\|)], \sigma_1\rangle \\
&\rightarrow_{ann} \langle E^{\ell}[\mathsf{check}^k_j((v_c\ v), v)], \sigma_1\rangle \\
&\rightarrow^*_{ann} \langle E'^{\ell}[\mathsf{check}^k_j(\|^j \#\mathsf{f}\|, e)], \sigma_2\rangle \\
&\rightarrow_{ann} \langle \mathsf{error}^k_j, \sigma_2\rangle
\end{aligned}$$

   By Lemma 32, we can deduce that for some Π such that $\varnothing; \Pi \Vdash \sigma_1$, $\ell_0; \Pi \Vdash E^{\ell}[{}^{\ell}\mathsf{mon}^k_j(\lfloor \|^j \mathsf{flat/c}(v_c)\|\rfloor^{\vec{k}}, \|^k v\|)]$. By Lemma 33, we derive $\varnothing; \Pi; \ell \Vdash {}^{\ell}\mathsf{mon}^k_j(\lfloor \|^j \mathsf{flat/c}(v_c)\|\rfloor^{\vec{k}}, \|^k v\|)$, and thus by well-formedness that $k \in \vec{k}$.

2. $\langle E^{\ell}[{}^{\ell}\mathsf{mon}^k_j(\lfloor \|^j c\|\rfloor^{\vec{\ell}}, v)], \sigma\rangle \rightarrow_{ann} \langle E^{\ell}[\mathsf{check}^{\ell}_j((v_c\ ()), e)], \sigma\rangle$ where
   $c = \mathsf{ctx/c}(v_c, (v_{c_{g_1}} \Rightarrow v_{c_{p_1}} \leftarrow v_{c_{v_1}}), \ldots (v_{c_{g_n}} \Rightarrow v_{c_{p_n}} \leftarrow v_{c_{v_n}}), v_a, (v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}), \ldots)$ and

$e = \mathsf{guard}_j\ ((v_{c_{g_1}}\ ()),v_{c_{p_1}},v_{c_{v_1}},\ldots \mathsf{guard}_j\ ((v_{c_{g_n}}\ ()),v_{c_{p_n}},v_{c_{v_n}},{}^{\ell}\mathsf{ctx/p}_j^k\ (v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\ldots,v)))$: In this case, we can deduce

$$\begin{aligned}
&\langle p_0, \varnothing\rangle \rightarrow^*_{ann} \langle E^\ell[{}^{\ell}\mathsf{mon}_j^k(\lfloor\!\|^j c\|\!\rfloor^{\vec{\ell}},v)], \sigma_1\rangle \\
&\rightarrow_{ann} \langle E^\ell[\mathsf{check}_j^\ell((v_c\ ()),e)], \sigma_1\rangle \\
&\rightarrow^*_{ann} \langle E'^{\ell}[\mathsf{check}_j^\ell(\|^j \#\mathsf{f}\|,e)], \sigma_2\rangle \\
&\rightarrow_{ann} \langle \mathsf{error}_j^\ell, \sigma_2\rangle
\end{aligned}$$

By Lemma 32, we can deduce that for some Π such that $\varnothing; \Pi \Vdash \sigma_1$, $\ell_0; \Pi \Vdash E^\ell[{}^{\ell}\mathsf{mon}_j^k(\lfloor\!\|^j c\|\!\rfloor^{\vec{\ell}},v)]$. By Lemma 33, we derive $\varnothing; \Pi; \ell \Vdash {}^{\ell}\mathsf{mon}_j^k(\lfloor\!\|^j c\|\!\rfloor^{\vec{\ell}},v)$, and thus by well-formedness that $\ell \in \vec{\ell}$.

3. $\langle E^\ell[(\|^{\ell\ell}\mathsf{ctx/p}_j^k\ (v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\ldots(v_{a_{g_n}} \Rightarrow v_{a_{p_n}} \leftarrow v_{a_{v_n}}),v_f)\|\ v)], \sigma\rangle \quad \rightarrow_{ann}$ $\langle E^\ell[(\mathsf{check}_j^k((v_a\ ()),e)\ v)], \sigma\rangle$ where $e = \mathsf{guard}_j\ ((v_{a_{g_1}}\ ()),v_{a_{p_1}},v_{a_{v_1}},\ldots \mathsf{guard}_j\ ((v_{a_{g_n}}\ ()),v_{a_{p_n}},v_{a_{v_n}},v_f))$: Again, since $\ell_0 \Vdash p_0$, p_0 does not contain ${}^{\ell}\mathsf{ctx/p}_j^k\ (v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\ldots(v_{a_{g_n}} \Rightarrow v_{a_{p_n}} \leftarrow v_{a_{v_n}}),v_f)$. This it must have been introduced by a reduction step $\langle E^\ell[{}^{\ell}\mathsf{mon}_j^k(\lfloor\!\|^j c\|\!\rfloor^{\vec{\ell}},v_f)], \sigma\rangle \quad \rightarrow_{ann} \quad \langle E^\ell[\mathsf{check}_j^\ell((v_c\ ()),e)], \sigma\rangle$ where $c = \mathsf{ctx/c}(v_c,(v_{c_{g_1}} \Rightarrow v_{c_{p_1}} \leftarrow v_{c_{v_1}}),\ldots(v_{c_{g_n}} \Rightarrow v_{c_{p_n}} \leftarrow v_{c_{v_n}}),v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\ldots)$ and $e = \mathsf{guard}_j\ ((v_{c_{g_1}}\ ()),v_{c_{p_1}},v_{c_{v_1}},\ldots \mathsf{guard}_j\ ((v_{c_{g_n}}\ ()),v_{c_{p_n}},v_{c_{v_n}},{}^{\ell}\mathsf{ctx/p}_j^k\ (v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\ldots,v_f)))$. In this case, we can deduce a longer series of reductions

$$\begin{aligned}
&\langle p_0, \varnothing\rangle \rightarrow^*_{ann} \langle E^\ell[{}^{\ell}\mathsf{mon}_j^k(\lfloor\!\|^j c\|\!\rfloor^{\vec{\ell}},v_f)], \sigma_1\rangle \\
&\rightarrow_{ann} \langle E^\ell[\mathsf{check}_j^\ell((v_c\ ()),e)], \sigma_1\rangle \\
&\rightarrow^*_{ann} \langle E'^{\ell}[(\|^{\ell\ell}\mathsf{ctx/p}_j^k\ (v_a,(v_{a_{g_1}} \Rightarrow v_{a_{p_1}} \leftarrow v_{a_{v_1}}),\ldots(v_{a_{g_n}} \Rightarrow v_{a_{p_n}} \leftarrow v_{a_{v_n}}),v_f)\|\ v)], \sigma_1\rangle \\
&\rightarrow_{ann} \langle E'^{\ell}[(\mathsf{check}_j^k((v_a\ ()),e)\ v)], \sigma_1\rangle \\
&\rightarrow^*_{ann} \langle E''^{\ell}[\mathsf{check}_j^\ell(\|^j \#\mathsf{f}\|,e)], \sigma_2\rangle \\
&\rightarrow_{ann} \langle \mathsf{error}_j^\ell, \sigma_2\rangle
\end{aligned}$$

By Lemma 32, we can deduce that for some Π such that $\varnothing; \Pi \Vdash \sigma_1$, $\ell_0; \Pi \Vdash E^\ell[{}^{\ell}\mathsf{mon}_j^k(\lfloor\!\|^j c\|\!\rfloor^{\vec{\ell}},v_f)]$. By Lemma 33, we derive $\varnothing; \Pi; \ell \Vdash {}^{\ell}\mathsf{mon}_j^k(\lfloor\!\|^j c\|\!\rfloor^{\vec{\ell}},v)$, and thus by well-formedness that $\ell \in \vec{\ell}$.

Combining these three cases with the results above suffice to show that CtxPCF is a complete monitor. □

Lemma 31 (Progress)**.** *For all programs p, stores σ, store typings Σ, and store labelings Π such that*

- $\Sigma \vdash p : \tau$,
- $\varnothing; \Sigma \vdash \sigma$, *and*
- $\ell_0; \Pi \Vdash p$,

then either

- $p = v$,
- $p = E^\ell[\mathsf{shift}\ \|^{\ell^\dagger\ell'}T'^k[t(t) : (\tau_d \rightarrow \tau_r)]\|\ \mathsf{as}\ x\ \mathsf{in}\ e]$ *where* E^ℓ *does not contain* $\mathsf{reset}\ \|^{\ell'''\ell''}T''^{k''}[t(t) : (\tau_d \rightarrow \tau_r)]\|\ \mathsf{in}\ E'^{\ell''''}$, *or*
- $p = \mathsf{error}_j^k$, *or*
- $\langle p, \sigma\rangle \rightarrow_{ann} \langle p', \sigma'\rangle$.

Proof. From Lemma 28 we know that there either $p = v$, $p = \mathsf{error}_j^k$, or $p = E^\ell[\mathsf{shift}\ \|^{\ell^\dagger\ell'}T'^k[t(t) : (\tau_d \rightarrow \tau_r)]\|\ \mathsf{as}\ x\ \mathsf{in}\ e]$ where E^ℓ does not contain $\mathsf{reset}\ \|^{\ell'''\ell''}T''^{k''}[t(t) : (\tau_d \rightarrow \tau_r)]\|\ \mathsf{in}\ E'^{\ell''''}$, or $p = \mathsf{module}\ \ell\ \mathsf{exports}\ x_{v_1}\ \mathsf{with}\ x_{c_1},\ldots\ \mathsf{where}\ y_1 = e_1,\ldots,y_n = e_n,\ldots; p'$, or there exist e, ℓ, and E^ℓ such that $p = E^\ell[e]$. The first three cases are immediate. If $p = \mathsf{module}\ \ell\ \mathsf{exports}\ x_{v_1}\ \mathsf{with}\ x_{c_1},\ldots\ \mathsf{where}\ \ldots,x_{v_1} = v_1,\ldots,x_{c_1} = c_1,\ldots; p'$, then by the reduction relation we have $\langle p, \sigma\rangle \rightarrow_{ann} \langle \mathrm{import}[\![\ell,(x_{v_1},\ldots,x_{v_n}),(v_1,\ldots,v_n),(c_1,\ldots,c_n),p']\!], \sigma\rangle$, since meta function import is total. In the final case, using Lemma 33 and $\ell_0; \Pi \Vdash p$ we derive $\varnothing; \Pi; \ell \Vdash e$. We proceed by case analysis on e (We only show here the redexes from the reduction rules of Annotated Ctrl+CtxPCF that do not overlap with those of Annotated CtxPCF. The cases for the redexes from the rules that overlap with those of Annoted CtxPCF transfer to this proof after adjusting the proof text to refer to lemmas for Annotated Ctrl+CtxPCF instead of the corresponding ones for Annotated CtxPCF):

1. $\mathsf{make\text{-}tag} : (\tau_1 \to \tau_2)$: By the reduction relation and Lemma 34 we get $\langle E^\ell[\mathsf{make\text{-}tag} : (\tau_1 \to \tau_2)], \sigma\rangle \to_{ann} \langle E^\ell[|^\ell \mathsf{t}(t) : (\tau_1 \to \tau_2)|], \sigma[t \mapsto ()]\rangle$ where t is fresh σ.
2. $\mathsf{reset}\ \|^{\ell^\dagger\,\ell} T^k[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{in}\ v$: Since $\varnothing; \Pi; \ell \Vdash \mathsf{reset}\ \|^{\ell^\dagger\,\ell} T^k[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{in}\ v$, it must be the case that $\ell^\dagger = \ell$ $\varnothing; \Pi; \ell \Vdash v$. Thus by the reduction relation and Lemma 34, we have $\langle E^\ell[\mathsf{reset}\ \|^{\ell\ell} T^k[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{in}\ v], \sigma\rangle \to_{ann} \langle E^\ell[v], \sigma\rangle$.
3. $\mathsf{reset}\ \|^{\ell^*\,\ell} T^k[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{in}\ E'^{\ell'}[\mathsf{shift}\ \|^{\ell^\dagger\,\ell'} T'^{k'}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{as}\ x\ \mathsf{in}\ e]$ where E' does not contain $\mathsf{reset}\ \|^{\ell'''\,\ell''} T''^{k''}[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{in}\ E''^{\ell''''}$: With the same reasoning as in the previous case we derive that $\ell^* = \ell$, $\ell^\dagger = \ell'$, and $k' = k$. Thus by by the reduction relation and Lemma 34 and given that metafunctions wrap_i^+ and wrap_i^- are total, we have $\langle E^\ell[\mathsf{reset}\ \|^{\ell\ell} T^k[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{in}\ E'^{\ell'}[\mathsf{shift}\ \|^{\ell'\,\ell'} T'^k[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{as}\ x\ \mathsf{in}\ e]], \sigma\rangle \to_{ann}$
   $\langle E^\ell[\mathsf{reset}\ ^\ell T^k[\mathsf{t}(t) : (\tau_d \to \tau_r)]\ \mathsf{in}\ \{^{v'}/_x\}e'], \sigma\rangle$ where $v_k = |^\ell \lambda\, y : \tau_d.\ \mathsf{reset}\ ^\ell T^k[\mathsf{t}(t) : (\tau_d \to \tau_r)]\ \mathsf{in}\ E'^{\ell'}[e_y]|$ and $v' = |^{\ell'} \mathrm{wrap}_1^+[\![^{\ell'} T'^k[\mathsf{t}(t) : (\tau_d \to \tau_r)], \mathrm{wrap}_1^-[\![^\ell T^k[\mathsf{t}(t) : (\tau_d \to \tau_r)], v_k]\!]]\!|$ and $e_y = |^{\ell'} \mathrm{wrap}_2^+[\![^{\ell'} T'^k[\mathsf{t}(t) : (\tau_d \to \tau_r)], \mathrm{wrap}_2^-[\![^\ell T^k[\mathsf{t}(t) : (\tau_d \to \tau_r)], y]\!]]\!|$ and $e' = |^\ell \mathrm{wrap}_3^+[\![^\ell T^k[\mathsf{t}(t) : (\tau_d \to \tau_r)], \mathrm{wrap}_3^-[\![^{\ell'} T'^k[\mathsf{t}(t) : (\tau_d \to \tau_r)], e]\!]]\!|$.
4. $^\ell\mathsf{mon}_j^k(\|^{l^*}\mathsf{tag/c}(c_1, c_2)c_3\|, v)$: With the same reasoning as in the previous case we derive that $\ell^* = j$. Thus by the reduction relation and Lemma 34, we have $\langle E^\ell[^\ell\mathsf{mon}_j^k(\|^j\mathsf{tag/c}(c_1, c_2)c_3\|, v)], \sigma\rangle \to_{ann} \langle E^\ell[^\ell\mathsf{tag/p}_j^k(c_1,c_2,c_3)v], \sigma\rangle$

□

Lemma 32 (Preservation)**.** *For all programs p, stores σ, store typings Σ, and store labelings Π such that*
- $\Sigma \vdash p : \tau$,
- $\varnothing; \Sigma \vdash \sigma$,
- $\ell_0; \Pi \Vdash p$, *and*
- $\langle p, \sigma\rangle \to_{ann} \langle p, \sigma'\rangle$,

there exists a store labeling Π' such that
- $\Pi' \supseteq \Pi$,
- $\ell_0; \Pi' \Vdash p$.

Proof. By case analysis on the reduction $\langle p, \sigma\rangle \to_{ann} \langle p', \sigma'\rangle$ (We only show here the reduction rules of Annotated Ctrl+CtxPCF that do not overlap with those of Annotated CtxPCF. The cases for the rules that overlap with those of Annoted CtxPCF transfer to this proof after adjusting the proof text to refer to lemmas for Annotated Ctrl+CtxPCF instead of the corresponding ones for Annotated CtxPCF):
- $\langle E^\ell[\mathsf{make\text{-}tag} : (\tau_1 \to \tau_2)], \sigma\rangle \to_{ann} \langle E^\ell[|^\ell \mathsf{t}(t) : (\tau_1 \to \tau_2)|], \sigma[t \mapsto ()]\rangle$ where t is fresh σ: Let $\Pi' = \Pi[t \mapsto \ell]$. We must show that $\varnothing; \Pi'; \ell_0 \Vdash E^\ell[|^\ell \mathsf{t}(t) : (\tau_1 \to \tau_2)|]$. Since by Lemma 36, $\varnothing; \Pi'; \ell_0 \Vdash E^\ell[\mathsf{make\text{-}tag} : (\tau_1 \to \tau_2)]$, by Lemma 34 it suffices to show that $\varnothing; \Pi'; \ell \Vdash E^l[\mathsf{t}(t) : (\tau_1 \to \tau_2)]$, which holds by inspection.
- $\langle E^\ell[\mathsf{reset}\ \|^{\ell\ell} T^k[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{in}\ v], \sigma\rangle \to_{ann} \langle E^\ell[v], \sigma\rangle$: By Lemma 34, it suffices to show that $\varnothing; \Pi; \ell \Vdash E^l[v]$. As usual, by Lemma 33 and inversion of well-formedness, we obtain that $\varnothing; \Pi; l \Vdash v$.
- $\langle E^\ell[\mathsf{reset}\ \|^{\ell\ell} T^k[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{in}\ E'^{\ell'}[\mathsf{shift}\ \|^{\ell'\,\ell'} T'^k[\mathsf{t}(t) : (\tau_d \to \tau_r)]\|\ \mathsf{as}\ x\ \mathsf{in}\ e]], \sigma\rangle \to_{ann}$
  $\langle E^\ell[\mathsf{reset}\ ^\ell T^k[\mathsf{t}(t) : (\tau_d \to \tau_r)]\ \mathsf{in}\ \{^{v'}/_x\}e'], \sigma\rangle$ where $v_k = |^\ell \lambda\, y : \tau_d.\ \mathsf{reset}\ ^\ell T^k[\mathsf{t}(t) : (\tau_d \to \tau_r)]\ \mathsf{in}\ E'^{\ell'}[e_y]|$ and $v' = |^{\ell'} \mathrm{wrap}_1^+[\![^{\ell'} T'^k[\mathsf{t}(t) : (\tau_d \to \tau_r)], \mathrm{wrap}_1^-[\![^\ell T^k[\mathsf{t}(t) : (\tau_d \to \tau_r)], v_k]\!]]\!|$ and $e_y = |^{\ell'} \mathrm{wrap}_2^+[\![^{\ell'} T'^k[\mathsf{t}(t) : (\tau_d \to \tau_r)], \mathrm{wrap}_2^-[\![^\ell T^k[\mathsf{t}(t) : (\tau_d \to \tau_r)], y]\!]]\!|$ and $e' = |^\ell \mathrm{wrap}_3^+[\![^\ell T^k[\mathsf{t}(t) : (\tau_d \to \tau_r)], \mathrm{wrap}_3^-[\![^{\ell'} T'^k[\mathsf{t}(t) : (\tau_d \to \tau_r)], e]\!]]\!|$: By Lemma 34, it suffices to show that $\varnothing; \Pi; \ell \Vdash \mathsf{reset}\ ^\ell T^k[\mathsf{t}(t) : (\tau_d \to \tau_r)]\ \mathsf{in}\ \{^{v'}/_x\}e'$, which requires that $\varnothing; \Pi; \ell \Vdash \{^{v'}/_x\}e'$. As usual, by Lemma 33 and inversion of well-formedness, we obtain that $\{x : \ell'\}; \Pi; \ell' \Vdash e$. This by Lemma 38 we have obtain $\{x : \ell'\}; \Pi; \ell \Vdash e'$. Furthermore by Lemma 38, we obtain $\{y : \ell'\}; \Pi; l \Vdash e_y$. Thus by the rules for well-formedness and Lemma 34, we obtain $\varnothing; \Pi; l \Vdash v_k$. From this fact and Lemma 38, we obtain that $\varnothing; \Pi; l' \Vdash v'$. Putting all the above facts together with Lemma 35, we derive that $\varnothing; \Pi; \ell \Vdash \mathsf{reset}\ ^\ell T^k[\mathsf{t}(t) : (\tau_d \to \tau_r)]\ \mathsf{in}\ \{^{v'}/_x\}e'$.
- $\langle E^\ell[^\ell\mathsf{mon}_j^k(\|^j\mathsf{tag/c}(c_1, c_2)c_3\|, v)], \sigma\rangle \to_{ann} \langle E^\ell[^\ell\mathsf{tag/p}_j^k(c_1,c_2,c_3)v], \sigma\rangle$: By Lemma 34, it suffices to show that $\varnothing; \Pi; \ell \Vdash {}^\ell\mathsf{tag/p}_j^k(c_1,c_2,c_3)v$, which requires that $\varnothing; \Pi; k \Vdash v$ and $\varnothing; \Pi; \vec{k}; \vec{\ell}; j \rhd c_i$. As usual, by Lemma 33 and inversion of well-formedness, we obtain that $\varnothing; \Pi; k \Vdash v$. Also using Lemma 33, inversion of well-formedness, and Lemma 37, we obtain $\varnothing; \Pi; \vec{k}; \vec{\ell}; j \rhd c_i$.

□

Lemma 33 (Label focusing for Annotated Ctrl+CtxPCF)**.** *If $\ell; \Pi \Vdash E^k[e]$ then $\varnothing; \Pi; k \Vdash e$.*

Proof. Induction on E^k. □

Lemma 34 (Well-formed evaluation context plugs for Annotated Ctrl+CtxPCF)**.** *If* $\ell; \Pi \Vdash E^k[e]$ *and* $\varnothing; \Pi; k \Vdash e'$ *then* $\ell; \Pi \Vdash E^k[e']$.

Proof. Induction on E^k. □

Lemma 35 (Well-formed substitution for Annotated Ctrl+CtxPCF)**.** *If* $\Delta[x \mapsto \ell]; \Pi; k \Vdash e_1$ *and* $\Delta; \Pi; \ell \Vdash e_2$ *then* $\Delta; \Pi; k \Vdash \{^{|^{\ell} e_2|}/_x\} e_1$.

Proof. Mutual induction on well-formedness and well-formed contracts. □

Lemma 36 (Weakening for Well-formedness for Annotated Ctrl+CtxPCF)**.** *If* $\Delta; \Pi; \ell \Vdash e$ *and* $r \notin \Pi$*, then* $\Delta; \Pi[t \mapsto k]; \ell \Vdash e$. *If* $\Delta; \Pi; \ell \Vdash e_1$ *and* $t \notin \Pi$*, then* $\Delta; \Pi[t \mapsto k]; \ell \Vdash \exp$.

Proof. Mutual induction on well-formedness and well-formed contracts. □

Lemma 37 (Weakening well-formed contracts for Annotated Ctrl+CtxPCF)**.** *If* $\Delta; \Pi; \vec{k}; \vec{\ell}; j \rhd c$*, then* $\Delta; \Pi; \vec{k'}; \vec{\ell'}; j \rhd c$ *for all* $\vec{k'}$ *and* $\vec{\ell'}$ *such that* $\vec{k} \subseteq \vec{k'}$ *and* $\vec{\ell} \subseteq \vec{\ell'}$.

Proof. Induction on well-formed contracts. □

Lemma 38 (Well-formed wrapping)**.** *Let* $i \in \{1, 2, 3\}$ *and* e *such that* $\Delta; \Pi; \ell \Vdash e$. *If* $e' = \mathrm{wrap}_i^+ [\![{}^{\ell'} T'^k[t(t) : (\tau_d \to \tau_r)], \mathrm{wrap}_i^- [\![{}^{\ell} T^k[t(t) : (\tau_d \to \tau_r)], e]\!]]\!]$ *then* $\Delta; \Pi; \ell' \Vdash e'$

Proof. By lexicographic induction on the sizes of ${}^{\ell'} T'^k$ and ${}^{\ell} T^k$. □

C. Example monitors

Note, in addition to the results that are described in Section 3.3, hooks may return an additional value `add-lifetime` specified using `#:add-lifetime`. This value is a set of delegations to be added to the global delegation set along with those of the `add` value. However, to control the size of the global delegation set, the library removes these delegations from the global delegation set when the garbage collector of Racket collects the wrapped function.

```
(define-monitor dac
 (monitor-interface make-object/c make-user/c grant/c revoke/c)
 (action
  [make-object/c (name setuser)
   #:on-create (do-create #:closure-principal (▷ current-principal (dim name)))
   #:on-apply  (let ([owner  (proj-principal closure-principal)]
                     [objdim (set-first (proj-dims closure-principal))])
                 (do-apply #:check (≽ @ current-principal (▷ closure-principal objdim use)
                                      (▷ closure-principal objdim use))
                           #:set-principal (if setuser owner #f)))]
  [make-user/c (name set-auth)
   #:on-create (let ([new-pcpl (pcpl name)])
                 (do-create #:add-lifetime (list (≽ current-principal (▷ new-pcpl use invoke)
                                                    @ new-pcpl))
                            #:closure-principal new-pcpl))
   #:on-apply  (do-apply #:check (≽ @ current-principal (▷ closure-principal use invoke)
                                    closure-principal)
                         #:set!-principal (if set-auth closure-principal #f)
                         #:set-principal  (if set-auth #f closure-principal))]
  [grant/c (object recipient)
   #:on-create (do-create)
   #:on-apply  (let ([recipient-principal (recipient-principal recipient)]
                     [object-principal    (object-principal object)])
                 (do-apply #:add (list (make-lifetime (≽ recipient-principal object-principal)
                                                      current-principal object))))]
  [revoke/c (object recipient)
   #:on-create (do-create)
   #:on-apply  (let ([recipient-principal (▷ (recipient-principal recipient) use)]
                     [object-principal    (object-principal object)])
                 (define (equal-recipient? p) (equal? p recipient-principal))
                 (define (matching-target? p) (equal? p object-principal))
                 (define (to-remove delegations current)
                   (filter (λ (d) (match d
                     [(labeled (actsfor (? equal-recipient? l) (? matching-target? r)) ll)
                     #:when (acts-for? current ll ll) #t]
                     [_ #f]))))
                 (do-apply #:remove (to-remove current-delegations current-principal)))])
 (extra
  (define use (dim 'use))
  (define invoke (dim 'invoke))

  (define (recipient-principal recipient)
    (cdr (make-user/c-authority recipient)))

  (define (object-principal object)
    (cond
      [(make-user/c? object)
       (▷ (cdr (make-user/c-authority object)) invoke)]
      [(make-object/c? object)
       (let* ([pcpl (cdr (make-object/c-authority object))])
         (▷ pcpl use))])))
```

Figure 11. Discretionary Access Control Monitor

```
(define-monitor stack-inspection
 (monitor-interface make-permission permission? check-permission/c enable-permission/c
                    do-privileged/c context/c unprivileged/c privileged/c coerce-to-unprivileged)
 (monitor-syntax-interface define/rights)
 (action #:search (list use-static search-delegates-left)
  [check-permission/c (perm)
   #:on-create (do-create)
   #:on-apply
   (do-apply #:check (≽ @ (▷ current-principal active) (▷ ⊤ perm) (▷ ⊤ perm)))]
  [enable-permission/c (perm)
   #:on-create (do-create)
   #:on-apply
   (do-apply #:add (list (≽ @ (▷ current-principal enable perm) (▷ current-principal static perm)
                             current-principal)))]
  [privileged/c (perms)
   #:on-create (do-create #:check (≽ current-principal ⊤ @ ⊤))
   #:on-apply
   (let* ([callee (pcpl (gensym 'frame))]
          [static-principal (normalize (disj (list->set (map (λ (p) (▷ ⊤ p)) perms))))]
          [scoped-delegations
           (list (≽ @ (▷ callee static) static-principal ⊤)
                 (≽ @ (▷ callee enable) (▷ current-principal active) current-principal)
                 (≽ @ (▷ callee active) (∨ (▷ callee enable) (▷ callee static)) callee))])
     (do-apply #:add-scoped scoped-delegations #:set-principal callee))]
  [do-privileged/c
   #:on-create (do-create)
   #:on-apply
   (do-apply #:add-scoped (list (≽ @ (▷ current-principal enable) (▷ current-principal static)
                                    current-principal)))]
  [context/c
   #:on-create (do-create #:add-lifetime (list (≽ @ (← ⊤ current-delegations) ⊤ ⊤)))
   #:on-apply
   (let* ([callee (pcpl (gensym 'frame))]
          [closure-pcpl (→ closure-principal closure-delegations)]
          [scoped-delegations
            (list (≽ @ (▷ callee static) (▷ closure-pcpl active) ⊤)
                  (≽ @ (▷ callee enable) (▷ current-principal active) current-principal)
                  (≽ @ (▷ callee active) (∨ (▷ callee enable) (▷ callee static)) callee))])
    (do-apply #:add-scoped scoped-delegations #:set-principal callee))]
  [unprivileged/c
   #:on-create (do-create)
   #:on-apply  (do-apply #:set-principal ⊥)])
 (extra
  (define active (dim 'active)) (define enable (dim 'enable)) (define static (dim 'static))
  (define (use-static l r ll ds)
    (if (and (not (proj? l)) (match r [(proj top _) #t] [_ #f])) (set (▷ l static)) (set)))
  (define (make-permission name) (dim name))
  (define (permission? val) (dim? val))
  (define coerce-to-unprivileged
    (make-contract #:name "coerce-to-unprivileged"
      #:projection (λ (blame) (λ (val)
        (cond [(privileged/c? val) val]
              [(unprivileged/c? val) val]
              [(context/c? val) val]
              [(procedure? val) (((contract-projection unprivileged/c) blame) val)]
              [else val]))))))
 (syntax
  (make-define/contract/free-vars/contract define/auth/contract
    (membrane/c coerce-to-unprivileged coerce-to-unprivileged)
    (membrane/c coerce-to-unprivileged coerce-to-unprivileged))
  (define-syntax define/rights
    (make-set!-transformer
     (lambda (stx) (syntax-case stx (set!)
       [(set! binder e) (raise-syntax-error 'set! (format "cannot mutate ~a" 'define/right))]
       [(define/right (head . args) (right (... ...)) ctc body (... ...))
        #'(define/auth/contract (head . args) (and/c ctc (privileged/c (list right (... ...))))
            body (... ...))]
       [(define/right head (right (... ...)) ctc body)
        #'(define/auth head (and/c ctc (privileged/c (list right (... ...)))) body)])))))
```

Figure 12. Stack Inspection Monitor

```
(define-monitor history-based
 (monitor-interface make-permission permission? check-permission/c grant/c accept/c
                    unprivileged/c privileged/c coerce-to-unprivileged)
  (monitor-syntax-interface define/rights)
  (action #:search (list use-static search-delegates-left)
   [check-permission/c (perm)
    #:on-create (do-create)
    #:on-apply  (do-apply #:check (≽ @ (▷ current-principal active) (▷ ⊤ perm) (▷ ⊤ perm)))]
   [privileged/c (perms)
    #:on-create (do-create #:check (≽ @ current-principal ⊤ ⊤))
    #:on-apply
    (let* ([callee (pcpl (gensym 'frame))]
           [static-principal (normalize (disj (list->set (map (λ (p) (▷ ⊤ p)) perms))))]
           [to-remove (filter (match-lambda
                                [(labeled (actsfor l r) ll)
                                 #:when (and (equal? l (▷ current-principal enable))
                                             (not (equal? r (▷ current-principal static))))
                                 #t]
                                [_ #f])
                              (delegation-set->list current-delegations))]
           [to-add (cons (≽ @ (▷ callee enable) (▷ current-principal active) current-principal)
                         (map (match-lambda
                                [(labeled (actsfor l r) ll)
                                 (labeled (actsfor l (normalize (∨ r (▷ callee static)))) ll)])
                              to-remove))]
           [to-scope (list (≽ @ (▷ callee static) static-principal ⊤)
                           (≽ @ (▷ callee active) (∨ (▷ callee enable) (▷ callee static)) callee))])
     (do-apply #:add to-add #:remove to-remove #:add-scoped to-scope #:set-principal callee))]
    [grant/c
     #:on-create (do-create)
     #:on-apply
     (do-apply #:add-scoped (list (≽ @ (▷ current-principal enable) (▷ current-principal static)
                                      current-principal)))]
    [accept-context/c
     #:on-create (do-create #:add-lifetime (list (≽ @ (← ⊤ current-delegations) ⊤ ⊤)))
     #:on-apply  (do-apply #:add    (delegation-set->list closure-delegations)
                           #:remove (delegation-set->list current-delegations)
                           #:set-principal closure-principal)]
    [unprivileged/c
     #:on-create (do-create) #:on-apply  (do-apply #:set-principal ⊥)])
 (extra
  (define active  (dim 'active)) (define enable  (dim 'enable)) (define static  (dim 'static))
  (define (use-static l r ll ds)
    (if (and (not (proj? l)) (match r [(proj top _) #t] [_ #f])) (set (▷ l static)) (set)))
  (define (make-permission name) (dim name)) (define (permission? val) (dim? val))
  (define accept/c
    (make-contract #:name "accept/c"
      #:projection (λ (blame) (λ (val)
         (make-keyword-procedure
           (λ (kwds kwd-args . other-args)
             (call/cc (λ (k)
               (let ([cont (with-contract #:region expression accept/c #:result accept-context/c k)])
                  (call-with-values (λ () (keyword-apply val kwds kwd-args other-args)) cont)))))))))
  (define coerce-to-unprivileged
    (make-contract #:name "coerce-to-unprivileged"
      #:projection (λ (blame) (λ (val)
        (cond [(unprivileged/c? val) val]
              [(privileged/c? val) val]
              [(procedure? val) (((contract-projection unprivileged/c) blame) val)]
              [else val]))))))
(syntax
 (make-define/contract/free-vars/contract define/auth/contract
   (membrane/c coerce-to-unprivileged coerce-to-unprivileged)
   (membrane/c coerce-to-unprivileged coerce-to-unprivileged))
 (define-syntax define/rights
   (make-set!-transformer (lambda (stx) (syntax-case stx (set!)
    [(set! define/rights e) (raise-syntax-error 'set! (format "cannot mutate ~a" 'define/rights))]
    [(define/rights (head . args) (right (... ...)) ctc body (... ...))
     #'(define/auth/contract (head . args) (and/c ctc (privileged/c (list right (... ...))))
         body (... ...))]
    [(define/rights head (right (... ...)) ctc body)
     #'(define/auth head (and/c ctc (privileged/c (list right (... ...)))) body)])))))
```

Figure 13. History-based Access Control Monitor

```
(define-monitor ocap
 (monitor-interface capability/c unprivileged-capability/c
                    capability/c? unprivileged-capability/c?
                    coerce-to-unprivileged-capability)
 (monitor-syntax-interface define/cap)
 (action #:search (list search-caps search-delegates-left)
  [capability/c
   #:on-create
   (let* ([child (pcpl (gensym 'capability))]
          [parent current-principal]
          [parenthood (≽ @ (▷ parent caps) (▷ child invoke) @ child)]
          [endowment  (≽ @ (▷ child caps) (→ (▷ parent caps) current-delegations) parent)]
          [validity   (≽ @ (← parent current-delegations) parent parent)])
     (do-create #:add-lifetime (list parenthood endowment validity) #:closure-principal child))
   #:on-apply
   (let ([introductions
          (filter-map (λ (arg) (let ([arg-cap (cdr arg)])
                                 (≽ @ (▷ closure-principal caps) (▷ arg-cap invoke) current-principal)))
          closure-args)]
         [return-hook
          (λ (results)
          (let ([result-introductions
             (map (λ (res) (make-lifetime (≽ (▷ current-principal caps) (▷ (cdr res) invoke))
                                          closure-principal (car res)))
             (filter id results))])
              (do-return #:add result-introductions)))])
     (do-apply
      #:check (≽ @ current-principal (▷ closure-principal invoke) (▷ closure-principal invoke))
      #:add-lifetime introductions #:set-principal closure-principal #:on-return return-hook))]
  [unprivileged-capability/c
     #:on-create
     (let* ([child (pcpl (gensym 'unprivileged))]
            [parent current-principal]
            [parenthood (≽ @ (▷ parent caps) (▷ child invoke) child)])
       (do-create #:add-lifetime (list parenthood) #:closure-principal child))
   #:on-apply
   (let ([introductions
          (filter-map (λ (arg) (let ([arg-cap (cdr arg)])
                                 (≽ @ (▷ closure-principal caps) (▷ arg-cap invoke) current-principal)))
          closure-args)]
         [return-hook
          (λ (results)
          (let ([result-introductions
             (map (λ (res) (make-lifetime (≽ (▷ current-principal caps) (▷ (cdr res) invoke))
                                          closure-principal (car res)))
             (filter id results))])
              (do-return #:add result-introductions)))])
     (do-apply
      #:check (≽ @ current-principal (▷ closure-principal invoke) (▷ closure-principal invoke))
      #:add-lifetime introductions #:set-principal closure-principal #:on-return return-hook))])
 (extra
  (define invoke (dim 'invoke)) (define caps (dim 'caps))
  (define (search-caps l r ll ds) (if (pcpl? l) (set (▷ l caps)) (set)))
  (define coerce-to-unprivileged-capability
   (make-contract #:name "coerce-to-unprivileged-capability"
     #:projection (λ (blame) (λ (val)
        (cond [(capability/c? val) val]
              [(unprivileged-capability/c? val) val]
              [(procedure? val) (((contract-projection unprivileged-capability/c) blame) val)]
              [else val]))))))
 (syntax
  (define-syntax define/cap
    (make-set!-transformer (λ (stx) (syntax-case stx (set!)
     [(set! binder e) (raise-syntax-error 'set! (format "cannot mutate ~a" 'binder))]
     [(binder head body0 body (... ...))
      #`(define/contract head capability/c body0 body (... ...))])))))
```

Figure 14. Object-capabilities Monitor

```
(define-monitor driver-monitor
  (monitor-interface safe/c deprivilege/c switch-player/c check-player/c)

  (action #:search (list search-delegates-left)
   [safe/c
    #:on-create (do-create)
    #:on-apply (do-apply #:set-principal ⊤)]
   [deprivilege/c
    #:on-create (do-create)
    #:on-apply (do-apply #:set-principal ⊥)]
   [switch-player/c (name)
    #:on-create (do-create)
    #:on-apply (let ([pcpl (player->pcpl name)])
                 (do-apply #:set-principal pcpl))]
   [check-player/c (name)
    #:on-create (do-create)
    #:on-apply (let ([pcpl (player->pcpl name)])
                 (do-apply #:check (≽@ current-principal pcpl pcpl)))])
  (extra
   (define player-pcpls (make-weak-hash))
   (define (player->pcpl name)
     (hash-ref! player-pcpls name (thunk (pcpl name))))))
```

Figure 15. Dominion Monitor

```
(define-monitor keybindings-monitor
  (monitor-interface make-permission permission?
                     check-and-switch-permissions/c
                     enable-permissions/c
                     permissions-closure/c)

  (action #:search (list search-delegates-left)
   [check/c (perms)
    #:on-create (do-create)
    #:on-apply  (do-apply #:check (≽ @ current-principal pcpl pcpl)
                          #:set-principal pcpl)]
   [enable/c (perms)
    #:on-create (do-create)
    #:on-apply  (do-apply #:set-principal (∧ current-principal perms))]
   [closure/c
    #:on-create (do-create)
    #:on-apply  (do-apply #:set-principal closure-principal)])
  (extra
   (define (make-permission name) (pcpl name))
   (define permission? pcpl?)))
```

Figure 16. DrRacket Keybindings Monitor

```
(object/c
 [add-canvas                          (check/c UnSafe)]
 [add-undo                            (-> any/c
                                          closure/c
                                          any)]
 [adjust-cursor                       (check/c ChangeEditorView)]
 [after-edit-sequence                 (check/c InEditSequence)]
 [after-load-file                     (check/c InLoadFile)]
 [after-save-file                     (check/c InSaveFile)]
 [auto-wrap                           (check/c SetSoftlineBreaks)]
 [begin-edit-sequence                 (enable/c InEditSequence)]
 [begin-write-header-footer-to-file   (check/c WriteFile)]
 [blink-caret                         (check/c ChangeEditorView)]
 [can-do-edit-operation?              (check/c InEditOperation)]
 [can-load-file?                      (check/c InLoadFile)]
 [can-save-file?                      (check/c InSaveFile)]
 [clear-undos                         (check/c ClearHistory)]
 [copy                                (and/c
                                        (check/c WriteClipboard)
                                        (enable/c (∧ WriteClipboard InCopy)))]
 [copy-self                           (check/c Safe)]
 [copy-self-to                        (check/c GetEditorInfo)]
 [cut                                 (check/c (∧ Delete WriteClipboard))]
 [dc-location-to-editor-location      (check/c Safe)]
 [default-style-name                  (check/c GetEditorInfo)]
 [do-edit-operation              (->a ([this any/c] [op symbol?])
                                      ([r any/c] [t any/c])
                                      #:auth (op)
                                      (cond [(symbol=? op 'undo) (check/c NavigateHistory)]
                                            [(symbol=? op 'redo) (check/c NavigateHistory)]
                                            [(symbol=? op 'clear) (check/c Delete)]
                                            [(symbol=? op 'cut) (check/c (∧ Delete WriteClipboard))]
                                            [(symbol=? op 'paste)
                                             (and/c (check/c (∧ ReadClipboard Insert))
                                                    (enable/c (∧ InPaste  ReadClipboard Insert)))]
                                            [(symbol=? op 'select-all) (check/c Select)]
                                            [(symbol=? op 'insert-text-box)
                                             (and/c (check/c Insert)
                                                    (enable/c (∧ InInsertBox Insert)))]
                                            [(symbol=? op 'insert-pasteboard-box)
                                             (and/c (check/c Insert)
                                                    (enable/c InInsertBox))]
                                            [(symbol=? op 'insert-image)
                                             (and/c (check/c Insert)
                                                    (enable/c (∧ InInsertBox Insert)))])
                                      any)]
 [paste                               (and/c (check/c (∧ ReadClipboard Insert))
                                             (enable/c (∧ ReadClipboard Insert InPaste)))]
 [move-position                       (->a ([this any/c] [code symbol?])
                                           ([extend? any/c] [kind symbol?])
                                           #:auth (extend?)
                                      (cond
                                        [(and (not (unsupplied-arg? extend?)) extend?)
                                         (check/c (∧ Select GetEditorInfo GetText SetInsertionPoint))]
                                        [else (check/c (∧ GetEditorInfo GetText SetInsertionPoint))])
                                           any)]
 ; additional method policies
 ...)
```

Figure 17. The contract for DrRacket's object

```
(define-monitor pkg-monitor
  (monitor-interface is-author/c is-curator/c as-user/c grant-curator/c revoke-curator/c
      authority-closure/c deprivilege/c)
  (action
   [is-author/c (pkg)
    #:on-create (do-create)
    #:on-apply
    (let* ([authors (string-split (hash-ref pkg 'author))]
           [authorpcpl (disj (list->set (map author->pcpl authors)))])
      (do-apply #:check (≽ @ current-principal authorpcpl authorpcpl)))]
   [is-curator/c
    #:on-create (do-create)
    #:on-apply (do-apply #:check (≽ @ current-principal curator curator))]
   [as-user/c (user)
    #:on-create (do-create)
    #:on-apply  (do-apply #:check (≽ @ current-principal (author->pcpl user) (author->pcpl user))
                          #:set-principal (author->pcpl user))]
   [grant-curator/c (user)
    #:on-create (do-create)
    #:on-apply  (do-apply #:add (list (≽ @ (author->pcpl user) curator curator)))]
   [revoke-curator/c (user)
    #:on-create (do-create)
    #:on-apply  (do-apply #:remove (list (≽ @ (author->pcpl user) curator curator)))]
   [authority-closure/c
    #:on-create (do-create #:closure-principal current-principal)
    #:on-apply  (do-apply #:set-principal closure-principal)]
   [deprivilege/c
    #:on-create (do-create)
    #:on-apply  (do-apply #:set!-principal unpriv)])
  (extra
   (define curator (pcpl 'curator))
   (define unpriv (pcpl 'unprivileged))
   (define pcpls (make-weak-hash))
   (define (author->pcpl author)
     (hash-ref! pcpls author (λ () (pcpl (string->symbol author)))))))
```

Figure 18. Racket package index monitor

```
(provide
 (contract-out
  [authenticate (and/c
                 authority-closure/c
                 (->i ([operation symbol?]
                       #:email [email string?]
                       #:password [password string?]
                       #:on-success [success (email) (->a () #:auth () (as-user/c email) any)]
                       #:on-failure [failure (-> symbol? any)])
                      [result any/c]))]
```

Figure 19. The contract for the package index authenticate function