Stateful Contracts for Affine Types[*]

Jesse A. Tov *Riccardo Pucella*

Northeastern University, Boston, MA 02115, USA

`{tov,riccardo}@ccs.neu.edu`

Abstract

Affine type systems manage resources by preventing some values from being used more than once. This offers expressiveness and performance benefits, but difficulty arises in interacting with components written in a conventional language whose type system provides no way to maintain the affine type system's aliasing invariants. We propose and implement a technique that uses behavioral contracts to mediate between code written in an affine language and code in a conventional typed language. We formalize our approach via a typed calculus with both affine-typed and conventionally-typed modules. We show how to preserve the guarantees of both type systems despite both languages being able to call into each other and exchange higher-order values.

This is the extended version of a paper that appeared in ESOP 2010.

[*]Our prototype implementation and related material may be found at `http://www.ccs.neu.edu/~tov/pubs/affine-contracts/`.

Contents

1 Introduction

Substructural type systems augment conventional type systems with the ability to control the number and order of uses of a data structure or operation (Walker 2005). Linear type systems (Wadler 1990; Plotkin 1993; Benton 1995; Ahmed et al. 2004), for example, ensure that values with linear type cannot be duplicated or dropped, but must be eliminated exactly once. Other substructural type systems refine these constraints. Affine type systems, which we consider here, prevent values from being duplicated but allow them to be dropped: a value of affine type may be used once or not at all.

Affine types are useful to support language features that rely on avoidance of aliasing. One example is session types (Gay and Hole 1999), which are a method to represent and statically check communication protocols. Suppose that the type declared by

$$\textbf{type}_{\mathscr{A}}\ \mathsf{prot} = (\mathsf{int\ send} \rightarrow \mathsf{string\ recv} \rightarrow \mathsf{unit})\ \mathsf{chan} \tag{1}$$

represents a channel whose protocol allows us to to send an integer, then receive a string, and finally end the session. Further, suppose that *send* and *recv* consume a channel whose type allows sending or receiving, as appropriate, and return a channel whose type is advanced to the next step in the protocol. Then we might write a function that takes two such channels and runs their protocols in parallel:

```
let_A twice (c1: prot, c2: prot, z: int): string ⊗ string =
  let once (c: prot) (_: unit) =
    let c      = send c z in
    let (s, _) = recv c   in s
  in (once c1) ||| (once c2)                                    (2)
```

The protocol is followed correctly provided that *c1* and *c2* are *different* channels. Calling *twice*(*c*, *c*, 5), for instance, would violate the protocol. An affine type system can prevent this.

In addition to session types and other forms of typestate (Strom and Yemini 1986), substructural types have been used for memory management (Jim et al. 2002), for optimization of lazy languages (Turner et al. 1995), and to handle effects in pure languages (Barendsen and Smetsers 1996). Given this range of features, a programmer may wish to take advantage of substructural types in real-world programs. Writing real systems, however, often requires access to comprehensive libraries, which mainstream programming languages usually provide but experimental implementations often do not. The prospect of rewriting a large library to work in a substructural language strikes these authors as unappealing.

It is therefore compelling to allow conventional and substructural languages to interoperate. We envision complementary scenarios:

- A programmer wishes to import legacy code for use by affine-typed client code. Unfortunately, legacy code unaware of the substructural conditions may duplicate values received from the substructural language.

- A programmer wishes to export substructural library code for access from a conventional language. A client may duplicate values received from the library and resubmit them, causing aliasing that the library could not produce on its own and bypassing the substructural type system's guarantees.

Our Contributions. We present a novel approach to regulating the interaction between an affine language and a conventionally-typed language and implement a multi-language system having several notable features:

- The non-affine language may gain access to affine values and may apply affine-language functions.
- The non-affine type system is utterly standard, making no concessions to the affine type system.
- And yet, the composite system preserves the affine language's invariants.

We model the principal features of our implementation in a multi-language calculus that enjoys type soundness. In particular, the conventional language, although it has access to the affine language's functions and values, cannot be used to subvert the affine type system.

Our solution is to wrap each exchanged value in a software contract (Findler and Felleisen 2002), which uses *one bit* of state to track when an affine value has been used. While this idea is simple, the details can be subtle.

Design Rationale and Background. Our multi-language system combines two sublanguages with different type systems. The $\mathscr{C}$ ("conventional") language is based on the call-by-value, polymorphic λ calculus (Girard 1972; Reynolds 1974) with algebraic datatypes and SML-style *abstype* (Milner et al. 1997). The $\mathscr{A}$ ("affine") language adds affine types and the ability to declare new abstract affine types, allowing us to implement affine abstractions such as session types and static read-write locks.

A program in our language consists of top-level module, value, and type definitions, each of which may be written in either of the two sublanguages. (In the example above (2), the subscripts on $\textbf{type}_{\mathscr{A}}$ and $\textbf{let}_{\mathscr{A}}$ indicate the $\mathscr{A}$ language.) Each language has access to modules written in the other language, although they view foreign types through a translation into the native type system. Affine modules are checked by an affine type system, and non-affine modules are checked by a conventional type system. Notably, non-functional affine types appear as abstract types to the conventional type system, which requires no special knowledge about affine types other than comparing them for equality.

In our introductory example, a protocol violation occurs only if the two arguments to *twice* are aliases for the same session-typed channel, which the $\mathscr{A}$ language type system prevents. Problems would arise if we could use the $\mathscr{C}$ language to subvert $\mathscr{A}$ language's type system non-aliasing invariants. To preserve the safety properties guaranteed by each individual type system and allow the two sublanguages to invoke one another and exchange values, we need to perform run-time checks in cases where the non-affine type system is too weak to express the affine type system's invariants. Because the affine type system can enforce all of the conventional type system's invariants, we may dispense with checks in the other direction.

For instance, the affine type system guarantees that an affine value created in an affine module will not be duplicated within the affine sublanguage. If, however, the value flows into a non-affine module, then static bets are off. In that case, we resort to a dynamic check that prevents the value from flowing back into an affine context more than once. Since our

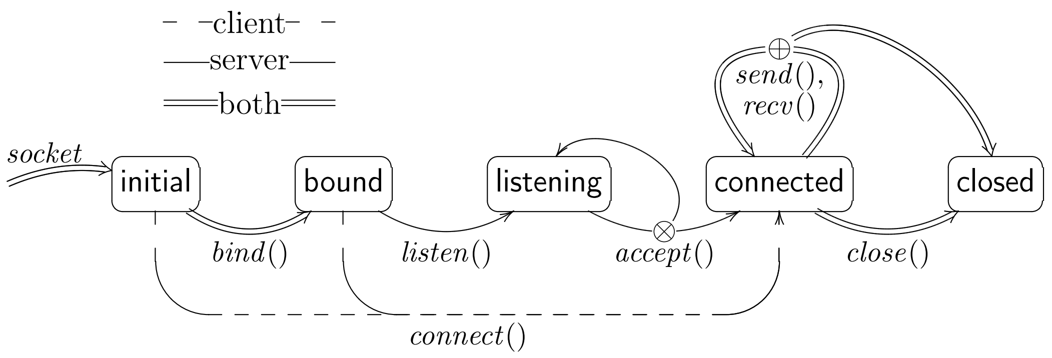

Figure 2.1: States and transitions for TCP (simplified)

language is higher-order, we use a form of higher-order contract (Findler and Felleisen 2002) to keep track of each module's obligations toward maintaining the affine invariants.

Our approach to integrating affine and conventional types borrows heavily from recent literature on multi-language interoperability (Flanagan 2006; Siek and Taha 2006). Our approach borrows from that of Typed Scheme (Tobin-Hochstadt and Felleisen 2008, 2006) and of Matthews and Findler (2007), both of which use contracts to mediate between an untyped, Scheme-like language and a typed language.

2 Example: Taming the Berkeley Sockets API

The key feature of our system is the ability to write programs that safely mix code written in an affine-typed language and a conventionally-typed language. As an example, we develop a small networking library and application, using both of our sublanguages where appropriate.

The Berkeley sockets API is the standard C language interface to network communication (Stevens 1990). Transmission Control Protocol (TCP), which provides reliable byte streams, is the standard transport layer protocol used by most internet applications (*e.g.*, SMTP, HTTP, and SSH). Setting up a TCP session using Berkeley sockets is a multi-step process (figure 2.1). A client must first create a communication end-point, called a *socket*, via the `socket` system call. It may optionally select a port to use with `bind`, and then it establishes a connection with `connect`. Once a connection is established, the client may `send` and `recv` until either the client or the other side closes the connection.

For a server, the process is more involved: it begins with `socket` and `bind` as the client does, and then it calls `listen` to allow connection requests to begin queuing. The server calls `accept` to accept a connection request. When `accept` succeeds, it returns a *new* socket that is connected to a client, and the old, listening socket is available for further `accept` calls. (For simplicity, we omit error transitions, except for failure of `send` and `recv`.)

Our $\mathscr{C}$ sublanguage provides the interface to sockets shown in figure 2.2. The socket operations are annotated with their pre- and post-conditions, but the implementation detects and signals state errors dynamically. For example, calling *listen* on a socket in state initial or calling *connect* on a socket that is already connected will raise an exception. If the other

```
module_𝒞 Socket : sig
  type socket
  val socket : unit → socket                (* ∅ ⇒ initial *)
  val bind   : socket → port → unit         (* initial ⇒ bound *)
  val listen : socket → unit                (* bound ⇒ listening *)
  val accept : socket → socket              (* listening ⇒ connected ⊗ listening *)
  val send   : socket → string → bool       (* connected ⇒ connected ⊕ closed *)
··· end
```

Figure 2.2: Selected $\mathscr{C}$ language socket operations, annotated with state transitions

side hangs up, send and recv raise exceptions, but nothing in this interface prevents further communication attempts that are bound to fail.[1]

By reimplementing the sockets API in language $\mathscr{A}$, we can use language $\mathscr{A}$'s type system to move the state transition information from comments into the type system itself. For example, we give *listen* in sublanguage $\mathscr{A}$ the type

$$\forall\alpha.\ \alpha\ \mathsf{socket} \to \alpha\ \mathsf{bound} \to \alpha\ \mathsf{listening}\,, \tag{3}$$

which means that given a socket and evidence that the socket is bound, *listen* changes the state to listening and returns evidence to that effect. These evidence tokens are *capabilities*, and the type parameter on each capability ties it to the particular socket whose state it describes. These capabilities have affine type so that when *listen* consumes the bound capability, we cannot call *listen* again on the same socket.

We reimplement the sockets API in language $\mathscr{A}$ in terms of the language $\mathscr{C}$ operations. From the vantage of language $\mathscr{A}$, $\mathscr{C}$ function types are mapped to $\mathscr{A}$ function types, but the $\mathscr{C}$ type Socket.socket is mapped to an *opaque type* {Socket.socket}. Some types are automatically converted between the two sublanguages, but for the remainder, type constructor $\{\cdot\}$ delimits foreign types referenced from the other sublanguage.

We declare a new abstract type for sockets in language $\mathscr{A}$, along with a type to represent each of the states:

```
abstype_𝒜 α socket = Sock of {Socket.socket}
      and α initial   qualifier A = Initial
      and α bound     qualifier A = Bound
      and α listening qualifier A = Listening
      and α connected qualifier A = Connected
with ··· (* operations detailed below *) ··· end                (4)
```

Several aspects of this abstype declaration bear further explanation:

- Each type has a phantom parameter α, which is used to associate a socket with the type witnessing its state.

[1] This simplifies the Berkeley sockets API by omitting address families, protocols, half-closed sockets, non-blocking IO, etc., but the stateful essence remains.

$$\begin{array}{lll}
\textbf{val } \textit{socket} & : & \text{unit} \to \exists\alpha.\, \alpha \text{ socket} \otimes \alpha \text{ initial} \\
\textbf{val } \textit{bind} & : & \forall\alpha.\, \alpha \text{ socket} \to \text{port} \to \alpha \text{ initial} \to \alpha \text{ bound} \\
\textbf{val } \textit{listen} & : & \forall\alpha.\, \alpha \text{ socket} \to \alpha \text{ bound} \to \alpha \text{ listening} \\
\textbf{val } \textit{accept} & : & \forall\alpha.\, \alpha \text{ socket} \to \alpha \text{ listening} \to (\alpha \text{ listening} \otimes \exists\beta.\, \beta \text{ socket} \otimes \beta \text{ initial}) \\
\textbf{val } \textit{connect} & : & \forall\alpha.\, \alpha \text{ socket} \to \text{host} \to \text{port} \to (\alpha \text{ initial} \oplus \alpha \text{ bound}) \to \alpha \text{ connected} \\
\textbf{val } \textit{send} & : & \forall\alpha.\, \alpha \text{ socket} \to \text{string} \to \alpha \text{ connected} \to \alpha \text{ connected} \\
\textbf{val } \textit{recv} & : & \forall\alpha.\, \alpha \text{ socket} \to \text{int} \to \alpha \text{ connected} \to \text{string} \otimes \alpha \text{ connected} \\
\textbf{val } \textit{close} & : & \forall\alpha.\, \alpha \text{ socket} \to \alpha \text{ connected} \to \text{unit}
\end{array}$$

Figure 2.3: The $\mathscr{A}$ language sockets API

- The syntax **qualifier A** on each the state type declares that outside the abstraction boundary, values of those types will appear as affine. Code inside the abstype declaration sees that they are ordinary, non-affine data types.
- Because each of the capabilities has only one constructor with no values, they need not be represented at run time.

The $\mathscr{A}$ language sockets interface appears in figure 2.3. The $\mathscr{A}$ sockets implementation relies on delegating to $\mathscr{C}$ language functions. From within $\mathscr{A}$, $\mathscr{C}$ types are viewed through a simple translation: function types, quantified types, and a few base types such as int pass through transparently, whereas other types are wrapped opaquely as Socket.socket was above. Thus, the type of $\mathit{Socket.socket}_{\mathscr{C}}$ becomes unit → {Socket.socket} when viewed from $\mathscr{A}$. Each $\mathscr{A}$ function is a minimal wrapper around its $\mathscr{C}$ counterpart:

```
let_A socket () =
  let sock = Socket.socket_C () in
    in Pack(unit, (Sock[unit] sock, Initial[unit])) as ∃β. β socket ⊗ β initial
let_A listen[α] (Sock sock as s: α socket) (_: α bound) =
  try
    Socket.listen_C sock;
    Listening[α]
  with IOError msg → raise (StillBound (freezeBound s cap, msg))             (5)
```

For $\mathit{socket}_{\mathscr{A}}$, we call $\mathit{Socket.socket}_{\mathscr{C}}$ to create the new socket, which we wrap in the Sock constructor and pack into an existential with a new initial capability. (The type abstracted by the existential is immaterial; unit will do.) Function $\mathit{listen}_{\mathscr{A}}$ calls its $\mathscr{C}$ counterpart on the socket and on success returns a listening capability tied by α to the socket. On failure, the socket is still in state bound, so it raises an exception containing the bound capability. The remaining functions are equally straightforward, but when we're done, provided we got this trusted kernel correct, we have an $\mathscr{A}$ library that enforces the correct ordering of socket operations. (The full code of the sockets library may be found in §A.)

Calling the various $\mathscr{C}$ socket operations from $\mathscr{A}$ is safe because none has a type that enables it to gain access to an $\mathscr{A}$ language value. Other situations are not as simple. Figure 2.4 shows an implementation of an echo server in language $\mathscr{A}$. (The working code is included with our prototype implementation on our web site.) The server sends back

```
let$_{\mathscr{A}}$ clientLoop[$\alpha$] (sock: $\alpha$ socket) (f: string $\to$ string) (cap: $\alpha$ connected) =
  let rec loop (cap: $\alpha$ connected): unit =
      let (str, cap) = recv sock 1024 cap in
      let cap        = send sock (f str) cap in
      loop cap
  in try
    loop cap
  with SocketError _ $\to$ ()

let interface threadFork :> (unit $\overset{a}{\multimap}$ unit) $\to$ {thread}$_{\mathscr{C}}$ = threadFork$_{\mathscr{C}}$

let rec$_{\mathscr{A}}$ acceptLoop[$\alpha$] (sock: $\alpha$ socket) (f: string $\to$ string) (cap: $\alpha$ listening): unit =
  let (cap, Pack($\beta$, (clientsock, clientcap))) = accept sock cap in
    threadFork (fun () $\to$ clientLoop clientsock f clientcap);
    acceptLoop sock f cap

let$_{\mathscr{A}}$ echoServe (port: int) (f: string $\to$ string) =
  let Pack($\alpha$, (sock, cap)) = socket () in
  let cap = bind sock port cap in
  let cap = listen sock cap in
    acceptLoop sock f cap
```

Figure 2.4: An echo server in language $\mathscr{A}$

the data it receives from each client after passing it through an unspecified string $\to$ string function *f*. The main function *echoServe* creates a socket, binds it to the requested port, and begins to listen. The type system ensures that *echoServe* performs these operations in the right order, and because the capabilities have affine types, it disallows referring to any one of them more then once. Function *echoServe* calls *acceptLoop*, which blocks in *accept* waiting for clients. For each client, it spawns a thread to handle that client and continues waiting for another client. Spawning the thread is where the multi-language interaction becomes tricky.

As in other substructural type systems, $\mathscr{A}$ requires that a function be given a type whose usage (unlimited or affine) is at least as restrictive as any variable that it closes over. Thus far, we have seen only unlimited function types ($\to$), also written $\overset{u}{\multimap}$. Language $\mathscr{A}$ also has affine function types, written $\overset{a}{\multimap}$.

The new client capability *clientcap*, returned by *accept*, has affine type β connected. Because the thunk for the new thread, (**fun** () $\to$ *clientLoop clientsock f clientcap*), closes over *clientcap*, it has affine type as well: unit $\overset{a}{\multimap}$ unit. This causes a problem: To create a new thread, we must pass the thunk to the $\mathscr{C}$ function *threadFork*$_{\mathscr{C}}$, whose type as viewed from $\mathscr{A}$ is (unit $\to$ unit) $\to$ {thread}$_{\mathscr{C}}$. Such a type makes *no guarantee* about how many times *threadFork*$_{\mathscr{C}}$ applies its argument. In order to pass the affine thunk to it, we assert that *threadFork*$_{\mathscr{C}}$ has the desired behavior:

$$\textbf{let interface}\ \textit{threadFork} :> (\text{unit} \overset{a}{\multimap} \text{unit}) \to \{\text{thread}\}_{\mathscr{C}} = \textit{threadFork}_{\mathscr{C}} \qquad (6)$$

This constitutes a checked assertion that the $\mathscr{C}$ value actually behaves according to the given

$\mathscr{A}$ type. This gets the program past $\mathscr{A}$'s type checker, and if $\mathit{threadFork}_{\mathscr{C}}$ attempts to apply its argument twice at run time, a dynamic check prevents it from doing so and signals an error.

The two sublanguages can interact in other ways:

- We may call $\mathit{echoServe}_{\mathscr{A}}$ from the $\mathscr{C}$ language, passing it a $\mathscr{C}$ function for f. This is safe because function f has type string $\to$ string, and thus can never gain access to an affine value.

- We may use the $\mathscr{A}$ language sockets library from a $\mathscr{C}$ program:

```
let𝒞 sneaky () =
  let Pack(α, (sock, cap1)) = socket𝒜 () in
  let cap2 = connect𝒜 sock "sneaky.example.org" 25 cap1 in
  let cap3 = connect𝒜 sock "sneaky2.example.org" 25 cap1 in
    ...                                                        (7)
```

  This program passes $\mathscr{C}$'s type checker but is caught when it attempts to reuse the initial capability *cap1* at run time. This misbehavior is detected because *sneaky*'s interaction with $\mathscr{A}$ is mediated by a behavioral contract.

3 Implementing Stateful Contracts

In Findler and Felleisen's formulation (2002), a contract is an agreement between two software components, or *parties*, about some property of a value. The *positive party* produces a value, which must satisfy the specified property. The *negative party* consumes the value and is held responsible for treating it appropriately. Contracts are concerned with catching violations of the property and blaming the guilty party, which may help locate the source of a bug. For first-order values the contract may be immediately checkable, but for functional values nontrivial properties are undecidable, so the check must wait until the negative party applies the function, at which point the negative party is responsible for providing a suitable argument and the positive party for producing a suitable result. Thus, for higher-order functions, checks are delayed until first-order values are reached.

In our language, the parties to contracts are modules, which must be in entirely one language or the other, and top-level functions, which we consider as singleton modules.

Contracts on first-order values check assertions about their arguments, and either return the argument or signal an error. Contracts on functions return functions that defer checking until first-order values are reached. The result of applying a contract should contextually approximate the argument. We represent a contract for a type α as a function taking two parties and a value of type α, and returning a value of the same type α:

$$\textbf{type}\ \alpha\ \mathsf{contract} = \mathsf{party} \times \mathsf{party} \to \alpha \to \alpha \tag{8}$$

A simple contract might assert something about a first-order value:

```
let evenContract (neg: party, pos: party) (x: int) =
  if isEven x then x else blame pos                            (9)
```

The contract is instantiated with the identities of the contracted parties, and then may be applied to a value. We may also construct contracts for functional values, given contracts for the domain and codomain:

```
let makeFunctionContract[$\alpha, \beta$] (dom: $\alpha$ contract, codom: $\beta$ contract)
                                    (neg: party, pos: party) (f: $\alpha \to \beta$) =
  fun (x: $\alpha$) $\to$ codom (neg, pos) (f (dom (pos, neg) x))                    (10)
```

When this contract is applied to a function, it can perform no checks immediately. Instead, it wraps the function so that, when the resulting function is applied, the domain contract is applied to the actual parameter and the codomain contract to the actual result.

We follow this approach closely, but with one small change—contracts for affine functions are stateful:

```
let makeAffineFunContract[$\alpha, \beta$] (dom: $\alpha$ contract, codom: $\beta$ contract)
                                     (neg: party, pos: party) (f: $\alpha \to \beta$) =
  let stillGood = ref true in
    fun (x: $\alpha$) $\to$
      if ! stillGood
        then stillGood $\leftarrow$ false;
             codom (neg, pos) (f (dom (pos, neg) x))
        else blame neg                                                       (11)
```

This approach works for functions because we can wrap a function to modify its behavior. But what about for other affine values such as the socket capabilities in §2? We must consider how non-functional values move between the two sublanguages.

In order to understand the solution, we need to show in greater detail how types are mapped between the two sublanguages. (The rest of the type system appears in the next section.) We define mappings $(\cdot)^{\mathscr{A}}$ and $(\cdot)^{\mathscr{C}}$ from $\mathscr{C}$ types to $\mathscr{A}$ types and $\mathscr{A}$ types to $\mathscr{C}$ types, respectively. Base types such as int and bool, which may be duplicated without restriction in both languages, map to themselves:

$$(\mathcal{B})^{\mathscr{A}} = \mathcal{B} \qquad\qquad (\mathcal{B})^{\mathscr{C}} = \mathcal{B} \tag{12}$$

Function types convert to function types. $\mathscr{C}$ function types go to unlimited functions in $\mathscr{A}$, and both unlimited and affine $\mathscr{A}$ functions collapse to ordinary ($\to$) functions in $\mathscr{C}$ (where q ranges over a and u):

$$(\tau_1 \to \tau_2)^{\mathscr{A}} = (\tau_2)^{\mathscr{A}} \overset{\mathsf{u}}{\multimap} (\tau_2)^{\mathscr{A}} \qquad\qquad (\sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2)^{\mathscr{C}} = (\sigma_1)^{\mathscr{C}} \to (\sigma_2)^{\mathscr{C}} \tag{13}$$

Quantified types map to quantified types, but they require renaming because we distinguish type variables between the two languages. In particular, $\mathscr{A}$ language type variables carry usage qualifiers, which indicate whether they may be instantiated to any type or only to unlimited types. (All type variables in §2 were of the u kind.)

$$(\forall\alpha.\, \tau)^{\mathscr{A}} = \forall\beta^{\mathsf{u}}.\, (\tau_1[\{\beta^{\mathsf{u}}\}/\alpha])^{\mathscr{A}} \qquad\qquad (\forall\alpha^{\mathsf{q}}.\, \sigma)^{\mathscr{C}} = \forall\beta.\, (\sigma_1[\{\beta\}/\alpha^{\mathsf{q}}])^{\mathscr{C}} \tag{14}$$

$$\begin{aligned}
\mathscr{CA}[\![\mathsf{int}]\!](n,p) &= \mathit{id} \\
\mathscr{CA}[\![\sigma_1 \overset{\mathsf{u}}{\multimap} \sigma_2]\!](n,p) &= \mathsf{makeFunctionContract}\ (\mathscr{AC}[\![\sigma_1]\!],\ \mathscr{CA}[\![\sigma_2]\!])\ (n,\ p) \\
\mathscr{CA}[\![\sigma_1 \overset{\mathsf{a}}{\multimap} \sigma_2]\!](n,p) &= \mathsf{makeAffineFunContract}\ (\mathscr{AC}[\![\sigma_1]\!],\ \mathscr{CA}[\![\sigma_2]\!])\ (n,\ p) \\
\mathscr{CA}[\![\sigma^o]\!](n,p) &= \mathbf{fun}\ (\mathsf{v}\colon \sigma^o) \to \mathsf{makeAffineFunContract} && \textit{(if } \sigma^o \textit{ is} \\
&\qquad (\mathit{id},\ \mathit{id})\ (n,\ p)\ (\mathbf{fun}\ () \to \mathsf{v}) && \textit{affine)}
\end{aligned}$$

$$\begin{aligned}
\mathscr{AC}[\![\mathsf{int}]\!](n,p) &= \mathit{id} \\
\mathscr{AC}[\![\sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2]\!](n,p) &= \mathsf{makeFunctionContract}\ (\mathscr{CA}[\![\sigma_1]\!],\ \mathscr{AC}[\![\sigma_2]\!])\ (n,\ p) \\
\mathscr{AC}[\![\sigma^o]\!](n,p) &= \mathbf{fun}\ (\mathsf{v}\colon \mathsf{unit} \to \sigma^o) \to \mathsf{v}\ () && \textit{(if } \sigma^o \textit{ is affine)}
\end{aligned}$$

Figure 3.1: Type-directed generation of coercions

Several algebraic data types, such as α option, map transparently when they are unlimited:

$$((\overline{\tau_i})\ c)^{\mathscr{A}} = (\overline{((\tau_i)^{\mathscr{A}})}\ c \qquad\qquad ((\overline{\sigma_i})\ c)^{\mathscr{C}} = (\overline{((\sigma_i)^{\mathscr{C}})}\ c \quad \text{if } |(\overline{\sigma_i})\ c| = \mathsf{u} \tag{15}$$

Finally, the remaining types are uninterpreted by the mapping, and merely enclosed in $\{\cdot\}$:

$$(\tau^o)^{\mathscr{A}} = \{\tau^o\}, \text{ otherwise} \qquad\qquad (\sigma^o)^{\mathscr{C}} = \{\sigma^o\}, \text{ otherwise} \tag{16}$$

Values in this class of types are inert: they have no available operations other than passing them back to their native sublanguage, which removes the $\{\cdot\}$. (We take $\{\{\tau\}\}$ to be equivalent to τ.)

This mapping implies that all non-functional, affine types in $\mathscr{A}$ map to opaque types in $\mathscr{C}$.[2] Since all that the $\mathscr{C}$ language can do with values of opaque type is pass them back to $\mathscr{A}$, we are free to wrap such values when they flow into $\mathscr{C}$ and unwrap them when they return to $\mathscr{A}$. Specifically, when an affine value v passes into $\mathscr{C}$, we wrap it in a λ abstraction, **fun** (_: unit) → v, and wrap that thunk with an affine function contract. If the wrapped value flows back into $\mathscr{A}$, we unwrap it by applying the thunk, which produces a contract error if we attempt unwrapping it more than once.

After type checking, our implementation translates $\mathscr{A}$ modules to $\mathscr{C}$ modules and wraps all interlanguage variable references with contracts that enforce the $\mathscr{A}$ language's view of the variable. In figure 3.1, we show several cases from a pair of metafunctions $\mathscr{AC}[\![\cdot]\!]$ and $\mathscr{CA}[\![\cdot]\!]$, which perform this wrapping. Metafunction $\mathscr{AC}[\![\cdot]\!]$ produces the coercion for references to $\mathscr{C}$ values from $\mathscr{A}$, and $\mathscr{CA}[\![\cdot]\!]$ is for references to $\mathscr{A}$ values from $\mathscr{C}$. Our formalization does not use this translation, but gives a semantics to the multi-language system directly.

[2] Opaque types may seem limiting, but Matthews and Findler (2007) have shown that it is possible, in what they call the "lump embedding," for each sublanguage to marshal its opaque values for the other sublanguage as desired. In practice, this amounts to exporting a fold to the other sublanguage.

$$
\begin{array}{rlcl}
\textit{variables} & \mathbf{x}, \mathbf{y} & \in & \mathit{Var}_{\mathscr{C}} \\
\textit{type variables} & \alpha, \beta & \in & \mathit{TVar}_{\mathscr{C}} \\
\textit{module names} & \mathbf{f}, \mathbf{g} & \in & \mathit{MVar}_{\mathscr{C}} \\
\textit{integers} & z & \in & \mathbb{Z} \\
\\
\textit{programs} & P & ::= & \mathbf{M}\ \mathbf{e} \\
\textit{module contexts} & M & ::= & \mathbf{m_1} \ldots \mathbf{m_k} \\
\textit{modules} & \mathbf{m} & ::= & \mathbf{module}\ \mathbf{f} : \tau = \mathbf{v} \\
\\
\textit{types} & \tau & ::= & \mathbf{int} \mid \tau \to \tau \\
 & & \mid & \forall\alpha.\,\tau \mid \alpha \\
\textit{expressions} & \mathbf{e} & ::= & \mathbf{v} \mid \mathbf{x} \mid \mathbf{f} \mid \mathbf{e}[\tau] \\
 & & \mid & \mathbf{e}\,\mathbf{e} \mid \mathbf{if0}\,\mathbf{e}\,\mathbf{e}\,\mathbf{e} \\
\textit{values} & \mathbf{v} & ::= & \Lambda\alpha.\,\mathbf{v} \mid \lambda\mathbf{x}{:}\tau.\,\mathbf{e} \mid \mathbf{c} \\
\textit{constants} & \mathbf{c} & ::= & \lceil z \rceil \mid - \mid (z-) \\
\\
\textit{type contexts} & \Delta & ::= & \cdot \mid \Delta, \alpha \\
\textit{value contexts} & \Gamma & ::= & \cdot \mid \Gamma, \mathbf{x}{:}\tau
\end{array}
$$

$$\boxed{\Delta; \Gamma \vdash^{M}_{\mathscr{C}} \mathbf{e} : \tau}$$

$$\textsc{TC-Mod}\quad \frac{\mathbf{module}\ \mathbf{f} : \tau = \mathbf{v} \in \mathbf{M} \qquad \cdot \vdash_{\mathscr{C}} \tau}{\Delta; \Gamma \vdash^{M}_{\mathscr{C}} \mathbf{f} : \tau}$$

$$\boxed{\vdash^{M} \mathbf{m}\ \text{okay}}$$

$$\textsc{TM-C}\quad \frac{\cdot; \cdot \vdash^{M}_{\mathscr{C}} \mathbf{v} : \tau}{\vdash^{M} \mathbf{module}\ \mathbf{f} : \tau = \mathbf{v}\ \text{okay}}$$

$$\boxed{\mathbf{e} \longmapsto_{M} \mathbf{e}}$$

$$\textsc{C-Mod}\quad \frac{(\mathbf{module}\ \mathbf{f} : \tau = \mathbf{v}) \in \mathbf{M}}{\mathbf{f} \underset{M}{\longmapsto} \mathbf{v}}$$

Figure 4.1: Selected syntax and semantics of $\lambda_{\mathscr{C}}$ (full semantics in §B)

4 Formalization

We model our language with a pair of calculi corresponding to the two sublanguages in the implementation. In this section, we first describe the two calculi independently, and then move on to explain how they interact.

To distinguish the two calculi, we typeset our affine calculus $\lambda^{\mathscr{A}}$ in a blue, sans-serif font and our non-affine calculus $\lambda_{\mathscr{C}}$ in a **bold, red, serif font**.

4.1 The Calculi $\lambda_{\mathscr{C}}$ and $\lambda^{\mathscr{A}}$

We model sublanguage $\mathscr{C}$ with calculus $\lambda_{\mathscr{C}}$, which is merely call-by-value System F (Girard 1972) equipped with singleton modules, each of which for simplicity declares only one name bound to one value. The syntax of $\lambda_{\mathscr{C}}$ appears in figure 4.1, including module names, which are disjoint from variable names. We include integer literals, which serve as first-order values that should pass transparently into the affine subcalculus. A program comprises a mutually recursive collection of modules M and a main expression $\mathbf{e}$. We give only the semantics relevant to modules, as the rest is standard. The expression typing judgment has the form $\Delta; \Gamma \vdash^{M}_{\mathscr{C}} \mathbf{e} : \tau$, and it carries a module context M, which rule TC-Mod uses to type module expressions. To type a program, we must type each module with rule TM-C; note that the whole module context is available to each module, allowing for recursion. Finally, C-Mod shows that module names reduce to the value of the module.

We model sublanguage $\mathscr{A}$ with calculus $\lambda^{\mathscr{A}}$, which extends $\lambda_{\mathscr{C}}$ with affine types. While $\lambda^{\mathscr{A}}$ includes all of $\lambda_{\mathscr{C}}$, we choose not to embed $\lambda_{\mathscr{C}}$ in $\lambda^{\mathscr{A}}$ to emphasize the generality of our approach, anticipating conventional language features that we do not know how to type in

$$
\begin{array}{rrcl}
\textit{variables} & \mathsf{x}, \mathsf{y} & \in & \mathit{Var}_{\mathscr{A}} \\
\textit{qualifiers} & \mathsf{q} & \in & \{\mathsf{a}, \mathsf{u}\} \\
\textit{type variables} & \alpha^{\mathsf{q}}, \beta^{\mathsf{q}} & \in & \mathit{TVar}_{\mathscr{A}} \\
\textit{module names} & \mathsf{f}, \mathsf{g} & \in & \mathit{MVar}_{\mathscr{A}} \\
\textit{integers} & z & \in & \mathbb{Z} \\
\\
\textit{modules} & \mathsf{m} & ::= & \mathsf{module}\ \mathsf{f} : \sigma = \mathsf{v} \\
\textit{types} & \sigma & ::= & \mathsf{int} \mid \sigma \overset{\mathsf{q}}{\multimap} \sigma \mid \forall \alpha^{\mathsf{q}}.\, \sigma \mid \sigma^{\circ} \\
\textit{opaque types} & \sigma^{\circ} & ::= & \alpha \mid \sigma \otimes \sigma \mid \sigma\ \mathsf{ref} \\
\textit{expressions} & \mathsf{e} & ::= & \mathsf{v} \mid \mathsf{x} \mid \mathsf{f} \mid \mathsf{e}\ \mathsf{e} \mid \mathsf{e}[\sigma] \mid \mathsf{if0}\ \mathsf{e}\ \mathsf{e}\ \mathsf{e} \\
 & & \mid & \langle \mathsf{e}, \mathsf{e} \rangle \mid \mathsf{let}\ \langle \mathsf{x}, \mathsf{x} \rangle = \mathsf{e}\ \mathsf{in}\ \mathsf{e} \\
\textit{values} & \mathsf{v} & ::= & \mathsf{c} \mid \lambda \mathsf{x}{:}\sigma.\mathsf{e} \mid \Lambda \alpha^{\mathsf{q}}.\, \mathsf{v} \mid \langle \mathsf{v}, \mathsf{v} \rangle \\
\textit{constants} & \mathsf{c} & ::= & \mathsf{new}[\sigma] \mid \mathsf{swap}[\sigma][\sigma] \mid \lceil z \rceil \mid - \mid (z-) \\
\\
\textit{value contexts} & \Gamma & ::= & \cdot \mid \Gamma, \mathsf{x}{:}\sigma \\
\textit{type contexts} & \Delta & ::= & \cdot \mid \Delta, \alpha^{\mathsf{q}}
\end{array}
$$

Figure 4.2: Syntax of $\lambda^{\mathscr{A}}$

$\boxed{\mathsf{q} \sqsubseteq \mathsf{q}}$

$$
\textsc{QRefl}\ \frac{}{\mathsf{q} \sqsubseteq \mathsf{q}} \qquad\qquad \textsc{QSubsume}\ \frac{}{\mathsf{u} \sqsubseteq \mathsf{a}}
$$

$\boxed{|\tau| = \mathsf{q}}$

$$
|\mathsf{int}| = \mathsf{u} \qquad |\sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2| = \mathsf{q} \qquad |\forall \alpha^{\mathsf{q}'}.\, \sigma| = |\sigma|
$$

$$
|\alpha^{\mathsf{q}}| = \mathsf{q} \qquad |\sigma_1 \otimes \sigma_2| = |\sigma_1| \sqcup |\sigma_2| \qquad |\sigma\ \mathsf{ref}\,| = \mathsf{a}
$$

$\boxed{|\Gamma| = \mathsf{q}}$

$$
|\Gamma| = \bigsqcup_{\mathsf{x} \in \mathrm{dom}(\Gamma)} |\Gamma(\mathsf{x})|
$$

Figure 4.3: Statics of $\lambda^{\mathscr{A}}$: qualifiers (i)

$\boxed{\Gamma \boxplus \Gamma = \Gamma}$

$$\frac{}{\cdot \boxplus \cdot = \cdot} \qquad \frac{\Gamma_1 \boxplus \Gamma_2 = \Gamma_3 \qquad |\sigma| = \mathsf{a}}{\Gamma_1 \boxplus \Gamma_2, x{:}\sigma = \Gamma_3, x{:}\sigma} \qquad \frac{\Gamma_1 \boxplus \Gamma_2 = \Gamma_3 \qquad |\sigma| = \mathsf{a}}{\Gamma_1, x{:}\sigma \boxplus \Gamma_2 = \Gamma_3, x{:}\sigma}$$

$$\frac{\Gamma_1 \boxplus \Gamma_2 = \Gamma_3 \qquad |\sigma| = \mathsf{u}}{\Gamma_1, x{:}\sigma \boxplus \Gamma_2, x{:}\sigma = \Gamma_3, x{:}\sigma}$$

Figure 4.4: Statics of $\lambda_{\mathscr{C}}$: context splitting (ii)

$\boxed{\Delta \vdash_{\mathscr{A}} \sigma}$

$$\frac{}{\Delta \vdash_{\mathscr{A}} \mathsf{int}} \qquad \frac{\Delta \vdash_{\mathscr{A}} \sigma_1 \qquad \Delta \vdash_{\mathscr{A}} \sigma_2}{\Delta \vdash_{\mathscr{A}} \sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2} \qquad \frac{\Delta, \alpha^{\mathsf{q}} \vdash_{\mathscr{A}} \sigma}{\Delta \vdash_{\mathscr{A}} \forall \alpha^{\mathsf{q}}.\, \sigma} \qquad \frac{\alpha^{\mathsf{q}} \in \Delta}{\Delta \vdash_{\mathscr{A}} \alpha^{\mathsf{q}}} \qquad \frac{\Delta \vdash_{\mathscr{A}} \sigma}{\Delta \vdash_{\mathscr{A}} \sigma\ \mathsf{ref}}$$

$$\frac{\Delta \vdash_{\mathscr{A}} \sigma_1 \qquad \Delta \vdash_{\mathscr{A}} \sigma_2}{\Delta \vdash_{\mathscr{A}} \sigma_1 \otimes \sigma_2}$$

$\boxed{\sigma <: \sigma}$

S-REFL
$$\frac{}{\sigma <: \sigma}$$

S-TRANS
$$\frac{\sigma_1 <: \sigma_2 \qquad \sigma_2 <: \sigma_3}{\sigma_1 <: \sigma_3}$$

S-PROD
$$\frac{\sigma_1 <: \sigma_1' \qquad \sigma_2 <: \sigma_2'}{\sigma_1 \otimes \sigma_2 <: \sigma_1' \otimes \sigma_2'}$$

S-ARROW
$$\frac{\sigma_1' <: \sigma_1 \qquad \sigma_2 <: \sigma_2' \qquad \mathsf{q} \sqsubseteq \mathsf{q}'}{\sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2 <: \sigma_1' \overset{\mathsf{q}'}{\multimap} \sigma_2'}$$

S-FORALL
$$\frac{\mathsf{q}_2 \sqsubseteq \mathsf{q}_1 \qquad \sigma_1[\beta^{\mathsf{q}_2}/\alpha^{\mathsf{q}_1}] <: \sigma_2}{\forall \alpha^{\mathsf{q}_1}.\, \sigma_1 <: \forall \beta^{\mathsf{q}_2}.\, \sigma_2}$$

Figure 4.5: Statics of $\lambda^{\mathscr{A}}$: types and subtyping (iii)

$\boxed{\Delta; \Gamma \vdash^{M}_{\mathscr{A}} \mathsf{e} : \sigma}$

$$\textsc{TA-Subsume}\ \frac{\Delta; \Gamma \vdash^{M}_{\mathscr{A}} \mathsf{e} : \sigma \qquad \sigma <: \sigma'}{\Delta; \Gamma \vdash^{M}_{\mathscr{A}} \mathsf{e} : \sigma'}
\qquad
\textsc{TA-TLam}\ \frac{\Delta, \alpha^{\mathsf{q}}; \Gamma \vdash^{M}_{\mathscr{A}} \mathsf{e} : \sigma}{\Delta; \Gamma \vdash^{M}_{\mathscr{A}} \Lambda\alpha^{\mathsf{q}}.\,\mathsf{v} : \forall\alpha^{\mathsf{q}}.\,\sigma}$$

$$\textsc{TA-TApp}\ \frac{\Delta; \Gamma \vdash^{M}_{\mathscr{A}} \mathsf{e} : \forall\alpha^{\mathsf{q}}.\,\sigma' \qquad \Delta \vdash_{\mathscr{A}} \sigma \qquad |\sigma| \sqsubseteq \mathsf{q}}{\Delta; \Gamma \vdash^{M}_{\mathscr{A}} \mathsf{e}[\sigma] : \sigma'[\sigma/\alpha^{\mathsf{q}}]}$$

$$\textsc{TA-Lam}\ \frac{\Delta; \Gamma, \mathsf{x} : \sigma \vdash^{M}_{\mathscr{A}} \mathsf{e} : \sigma' \qquad \Delta \vdash_{\mathscr{A}} \sigma \qquad \left|\Gamma|_{\mathrm{FV}(\lambda \mathsf{x}:\sigma.\,\mathsf{e})}\right| = \mathsf{q}}{\Delta; \Gamma \vdash^{M}_{\mathscr{A}} \lambda \mathsf{x}{:}\sigma.\,\mathsf{e} : \sigma \overset{\mathsf{q}}{\multimap} \sigma'}$$

$$\textsc{TA-App}\ \frac{\Delta; \Gamma_1 \vdash^{M}_{\mathscr{A}} \mathsf{e}_1 : \sigma' \overset{\mathsf{q}}{\multimap} \sigma \qquad \Delta; \Gamma_2 \vdash^{M}_{\mathscr{A}} \mathsf{e}_2 : \sigma'}{\Delta; \Gamma_1 \boxplus \Gamma_2 \vdash^{M}_{\mathscr{A}} \mathsf{e}_1\ \mathsf{e}_2 : \sigma}
\qquad
\textsc{TA-Pair}\ \frac{\Delta; \Gamma_1 \vdash^{M}_{\mathscr{A}} \mathsf{e}_1 : \sigma_1 \qquad \Delta; \Gamma_2 \vdash^{M}_{\mathscr{A}} \mathsf{e}_2 : \sigma_2}{\Delta; \Gamma_1 \boxplus \Gamma_2 \vdash^{M}_{\mathscr{A}} \langle \mathsf{e}_1, \mathsf{e}_2 \rangle : \sigma_1 \otimes \sigma_2}$$

$$\textsc{TA-Let}\ \frac{\Delta; \Gamma_1 \vdash^{M}_{\mathscr{A}} \mathsf{e}_1 : \sigma_{\mathsf{x}} \otimes \sigma_{\mathsf{y}} \qquad \Delta; \Gamma_2, \mathsf{x} : \sigma_{\mathsf{x}}, \mathsf{y} : \sigma_{\mathsf{y}} \vdash^{M}_{\mathscr{A}} \mathsf{e}_2 : \sigma}{\Delta; \Gamma_1 \boxplus \Gamma_2 \vdash^{M}_{\mathscr{A}} \mathsf{let}\ \langle \mathsf{x}, \mathsf{y} \rangle = \mathsf{e}_1\ \mathsf{in}\ \mathsf{e}_2 : \sigma}
\qquad
\textsc{TA-Con}\ \frac{}{\Delta; \Gamma \vdash^{M}_{\mathscr{A}} \mathsf{c} : \mathrm{ty}_{\mathscr{A}}(\mathsf{c})}$$

$$\textsc{TA-If0}\ \frac{\Delta; \Gamma_1 \vdash^{M}_{\mathscr{A}} \mathsf{e}_1 : \mathsf{int} \qquad \Delta; \Gamma_2 \vdash^{M}_{\mathscr{A}} \mathsf{e}_2 : \sigma \qquad \Delta; \Gamma_2 \vdash^{M}_{\mathscr{A}} \mathsf{e}_3 : \sigma}{\Delta; \Gamma_1 \boxplus \Gamma_2 \vdash^{M}_{\mathscr{A}} \mathsf{if0}\, \mathsf{e}_1\ \mathsf{e}_2\ \mathsf{e}_3 : \sigma}
\qquad
\textsc{TA-Var}\ \frac{}{\Delta; \Gamma, \mathsf{x} : \sigma, \Gamma' \vdash^{M}_{\mathscr{A}} \mathsf{x} : \sigma}$$

$$\textsc{TA-Mod}\ \frac{\mathsf{module}\ \mathsf{f} : \sigma = \mathsf{v} \in M \qquad \cdot \vdash_{\mathscr{A}} \sigma}{\Delta; \Gamma \vdash^{M}_{\mathscr{A}} \mathsf{f} : \sigma}$$

$\boxed{\mathrm{ty}_{\mathscr{A}}(\mathsf{c}) = \sigma}$

$$\mathrm{ty}_{\mathscr{A}}(-) = \mathsf{int} \overset{\mathsf{u}}{\multimap} \mathsf{int} \overset{\mathsf{u}}{\multimap} \mathsf{int} \qquad \mathrm{ty}_{\mathscr{A}}((z-)) = \mathsf{int} \overset{\mathsf{u}}{\multimap} \mathsf{int} \qquad \mathrm{ty}_{\mathscr{A}}(\lceil z \rceil) = \mathsf{int}$$

$$\mathrm{ty}_{\mathscr{A}}(\mathsf{new}[\sigma]) = \sigma \overset{\mathsf{u}}{\multimap} \sigma\ \mathsf{ref} \qquad \mathrm{ty}_{\mathscr{A}}(\mathsf{swap}[\sigma_1][\sigma_2]) = (\sigma_1\ \mathsf{ref} \otimes \sigma_2) \overset{\mathsf{u}}{\multimap} (\sigma_1 \otimes \sigma_2\ \mathsf{ref})$$

Figure 4.6: Statics of $\lambda^{\mathscr{A}}$: expressions and constants (iv)

$\boxed{\vdash^{M} \mathsf{m}\ \text{okay}}$

$$\text{TM-A}\quad \frac{\cdot;\cdot \vdash^{M}_{\mathscr{A}} \mathsf{v} : \sigma \qquad |\sigma| = \mathsf{u}}{\vdash^{M} \mathsf{module}\ \mathsf{f} : \sigma = \mathsf{v}\ \text{okay}}$$

Figure 4.7: Statics of $\lambda^{\mathscr{A}}$: modules (v)

$$\begin{array}{rrcl}
\textit{locations} & \ell \in \textit{Loc} & & \\
\textit{values} & \mathsf{v} & ::= & \cdots \mid \ell \\
\textit{stores} & s & ::= & \{\ell \mapsto \mathsf{v}, \ldots, \ell \mapsto \mathsf{v}\} \\
\textit{configurations} & \mathsf{C} & ::= & (s, \mathsf{e}) \\
\textit{evaluation contexts} & \mathsf{E} & ::= & [\,]_{\mathscr{A}} \mid \mathsf{E}[\sigma] \mid \mathsf{E}\,\mathsf{e} \mid \mathsf{v}\,\mathsf{E} \mid \langle \mathsf{E}, \mathsf{e}\rangle \mid \langle \mathsf{v}, \mathsf{E}\rangle \\
 & & \mid & \mathsf{if0}\ \mathsf{E}\ \mathsf{e}\ \mathsf{e} \mid \mathsf{let}\ \langle \mathsf{x}, \mathsf{y}\rangle = \mathsf{E}\ \mathsf{in}\ \mathsf{e}
\end{array}$$

$\boxed{\mathsf{C} \longmapsto_{M} \mathsf{C}}$

$$\begin{array}{lrcll}
(\text{A-}\delta) & (s, \mathsf{c}\ \mathsf{v}) & \underset{M}{\longmapsto} & \delta_{\mathscr{A}}(s, \mathsf{c}, \mathsf{v}) & \\
(\text{A-}B) & (s, (\Lambda\alpha^{\mathsf{q}}.\,\mathsf{v})[\sigma]) & \underset{M}{\longmapsto} & (s, \mathsf{v}[\sigma/\alpha^{\mathsf{q}}]) & \\
(\text{A-}\beta) & (s, (\lambda \mathsf{x}{:}\sigma.\,\mathsf{e})\ \mathsf{v}) & \underset{M}{\longmapsto} & (s, \mathsf{e}[\mathsf{v}/\mathsf{x}]) & \\
(\text{A-L}\textsc{et}) & (s, \mathsf{let}\ \langle \mathsf{x}_1, \mathsf{x}_2\rangle = \langle \mathsf{v}_1, \mathsf{v}_2\rangle\ \mathsf{in}\ \mathsf{e}) & \underset{M}{\longmapsto} & (s, \mathsf{e}[\mathsf{v}_2/\mathsf{x}_2][\mathsf{v}_1/\mathsf{x}_1]) & \\
(\text{A-I}\textsc{f}0) & (s, \mathsf{if0}\lceil 0\rceil\ \mathsf{e}_{\mathsf{t}}\ \mathsf{e}_{\mathsf{f}}) & \underset{M}{\longmapsto} & (s, \mathsf{e}_{\mathsf{t}}) & \\
(\text{A-I}\textsc{f}\text{Z}) & (s, \mathsf{if0}\lceil z\rceil\ \mathsf{e}_{\mathsf{t}}\ \mathsf{e}_{\mathsf{f}}) & \underset{M}{\longmapsto} & (s, \mathsf{e}_{\mathsf{f}}) & z \neq 0 \\
(\text{A-M}\textsc{od}) & (s, \mathsf{f}) & \underset{M}{\longmapsto} & (s, \mathsf{v}) & (\mathsf{module}\ \mathsf{f} : \sigma = \mathsf{v}) \in M \\
(\text{A-C}\textsc{xt}) & (s, \mathsf{E}[\mathsf{e}]_{\mathscr{A}}) & \underset{M}{\longmapsto} & (s', \mathsf{E}[\mathsf{e}']_{\mathscr{A}}) & \text{if } (s, \mathsf{e}) \underset{M}{\longmapsto} (s', \mathsf{e}')
\end{array}$$

$$\begin{array}{rcll}
\delta_{\mathscr{A}}(s, -, \lceil z\rceil) & = & (s, (z-)) & \\
\delta_{\mathscr{A}}(s, (z_1-), \lceil z_2\rceil) & = & (s, \lceil z_1 - z_2\rceil) & \\
\delta_{\mathscr{A}}(s, \mathsf{new}[\sigma], \mathsf{v}) & = & (s \uplus \{\ell \mapsto \mathsf{v}\}, \ell) & \ell\ \text{fresh} \\
\delta_{\mathscr{A}}(s \uplus \{\ell \mapsto \mathsf{v}_1\}, \mathsf{swap}[\sigma_1][\sigma_2], \langle \ell, \mathsf{v}_2\rangle) & = & (s \uplus \{\ell \mapsto \mathsf{v}_2\}, \langle \mathsf{v}_1, \ell\rangle) &
\end{array}$$

Figure 4.8: Dynamics of $\lambda^{\mathscr{A}}$

an affine language. The syntax of $\lambda^{\mathscr{A}}$ may be found in figure 4.2. Expressions are mostly conventional: values, which include λ and Λ abstractions, constants, and pairs; variables; application and type application; if expressions; pair construction; and pair elimination. Less conventionally, expressions also include *module names* (f), which reduce to the value of the named module. We define the free variables of an expression in the usual way, but note that this includes only regular variables (*e.g.*, y), not module names (*e.g.*, g), which we assume are distinguished syntactically.

Types include integers, function types with qualifier q, universals, and the syntactically distinguished opaque types, which include type variables, products, and reference cells. Figure 4.3 defines a lattice on qualifiers, of which there are only two: u is bottom and a is top. A qualifier is assigned to each type, with the notation $|\sigma| = \mathsf{q}$. Integers are always assigned the unlimited qualifier u, whereas references always have the affine qualifier a. Function types and type variables are annotated with their qualifiers, and products get the stronger qualifier of either of their components. We define the qualifier of a value context Γ as well, to be the maximum qualifier of any type bound in it; in other words, Γ is affine if *any* variable is affine, but if none is then it is unlimited.

Figure 4.4 defines context splitting, which is used by expression typing to distribute affine assumptions to only one use in a term, but unlimited variables to an unlimited number of mentions. When a value context must be split to type two subexpressions, in an application expression, for example (figure 4.6), variables of affine type are made available to either the operator or operand, but not both.

The subtyping relation appears in figure 4.5. It is reflexive and transitive, covariant on both pair components and function codomains, and contravariant on function domains, as usual. Subtyping arises from the qualifier lattice in two ways: an unlimited function may be used where an affine function is expected (but not vice versa), and a universal type whose bound variable has qualifier a may be instantiated by a type with qualifier u (but not vice versa).

Selected expression typing rules appear in figure 4.6. Rules TA-Lam and TA-App are the usual substructural rules for typing λ expressions and applications: for λ expressions, the qualifier q given to the resulting $\xrightarrow{\mathsf{q}}\!\circ$ type is the qualifier of the context Γ limited to the free variables of the expression; thus, the function is at least as restricted as any values it closes over. The type application rule TA-TApp requires that a type variable be at least as restrictive as any type with which it is instantiated.

Finally, the constant swap (figure 4.6) takes a pair of a σ_1 reference and a σ_2, and returns a σ_1 and a σ_2 reference. From the operational semantics, which appears in figure 4.8, it should be clear that swap swaps the σ_2 argument into the location and returns the value previously in the location. Since the type of swap does not require these two types to be the same, swap performs a *strong update*—that is, it may change the type of the value residing in a reference cell. This is why the qualifier given to references must be a: if a reference is aliased, then it becomes possible to observe the type change in a way the destroys type safety. This feature of the calculus is a stand-in for the variety of invariants that an affine type system might enforce. In the mixed calculus, $\lambda_{\mathscr{C}}$ may gain access to $\lambda^{\mathscr{A}}$ references. It has no operations available to read or write them, but it must be prevented from passing an aliased reference cell back into $\lambda^{\mathscr{A}}$ where it can cause trouble.

$$
\begin{array}{rrcl}
\textit{programs} & P & ::= & M\ \mathbf{e} \\
\textit{module contexts} & M & ::= & m_1 \ldots m_k \\
\textit{modules} & m & ::= & \mathsf{m} \mid \mathbf{m} \mid \mathbf{interface\, f} :> \sigma = \mathbf{g} \\
\lambda_{\mathscr{C}}\ \textit{types} & \tau & ::= & \cdots \mid \{\sigma\} \\
\lambda_{\mathscr{C}}\ \textit{expressions} & \mathbf{e} & ::= & \cdots \mid \mathsf{f}^{\mathbf{g}} \\
\lambda^{\mathscr{A}}\ \textit{types} & \sigma & ::= & \cdots \mid \{\tau\} \\
\lambda^{\mathscr{A}}\ \textit{expressions} & \mathsf{e} & ::= & \cdots \mid \mathbf{f}^{\mathsf{g}}
\end{array}
$$

Figure 4.9: New syntax for $\lambda^{\mathscr{A}}_{\mathscr{C}}$

$\boxed{(\tau)^{\mathscr{A}} = \sigma}$, $\boxed{(\sigma)^{\mathscr{C}} = \tau}$

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
(\mathbf{int})^{\mathscr{A}} & = & \mathsf{int} & (\mathsf{int})^{\mathscr{C}} & = & \mathbf{int} \\
(\tau_1 \to \tau_2)^{\mathscr{A}} & = & (\tau_1)^{\mathscr{A}} \overset{\mathsf{u}}{\multimap} (\tau_2)^{\mathscr{A}} & (\sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2)^{\mathscr{C}} & = & (\sigma_1)^{\mathscr{C}} \to (\sigma_2)^{\mathscr{C}} \\
(\forall \alpha.\, \tau)^{\mathscr{A}} & = & \forall \beta^{\mathsf{u}}.\, (\tau[\{\beta^{\mathsf{u}}\}/\alpha])^{\mathscr{A}} & (\forall \alpha^{\mathsf{q}}.\, \sigma)^{\mathscr{C}} & = & \forall \beta.\, (\sigma[\{\beta\}/\alpha^{\mathsf{q}}])^{\mathscr{C}} \\
(\{\sigma^{\circ}\})^{\mathscr{A}} & = & \sigma^{\circ} & (\{\tau^{\circ}\})^{\mathscr{C}} & = & \tau^{\circ} \\
(\tau^{\circ})^{\mathscr{A}} & = & \{\tau^{\circ}\} & (\sigma^{\circ})^{\mathscr{C}} & = & \{\sigma^{\circ}\}
\end{array}
$$

$\boxed{|\sigma| = \mathscr{A}}$

$$|\{\tau\}| = \mathsf{u}$$

Figure 4.10: New statics for $\lambda^{\mathscr{A}}_{\mathscr{C}}$: type translation and qualifiers (i)

4.2 Mixing It Up with $\lambda^{\mathscr{A}}_{\mathscr{C}}$

The primary aim of this work is to construct (type-safe) programs by mixing modules written in an affine language and modules written in a non-affine language, and to have them interoperate as seamlessly as possible. We can then model an affine program calling into a library written in a legacy language, or a conventional program calling into code written in an affine language. In either case, we must ensure that the non-affine portions of the program do not break the affine portions' invariants. As noted in §3, we accomplish this via run-time checks in the style of higher-order contracts (Findler and Felleisen 2002).

The additional syntax for mixed programs is in figure 4.9. The main expression in a mixed program is in subcalculus $\lambda_{\mathscr{C}}$. Modules now include $\lambda^{\mathscr{A}}$ modules, $\lambda_{\mathscr{C}}$ modules, and *interface* modules, which are used to assert a $\lambda^{\mathscr{A}}$ type about a $\lambda_{\mathscr{C}}$ module as we saw in §2.

We add to each subcalculus's expressions a production referring to modules from the other subcalculus. We decorate each such module name with the name of the module in which it appears (*e.g.*, $\mathbf{f}^{\mathsf{g}}$ for a reference to $\lambda_{\mathscr{C}}$ module $\mathbf{f}$ from $\lambda^{\mathscr{A}}$ module g) and use this name as the negative party in contracts regulating the intercalculus boundary, in order to assign blame.

$\boxed{\vdash P : \tau}$, $\boxed{\vdash^M m \text{ okay}}$

$$\textsc{Prog}\ \frac{(\forall m \in M) \vdash^M m \text{ okay} \qquad \cdot;\cdot \vdash^M_{\mathscr{C}} \mathbf{e} : \tau}{\vdash M\ \mathbf{e} : \tau}$$

$$\text{TM-I}\ \frac{(\textbf{module g} : (\sigma)^{\mathscr{C}} = \mathbf{v}) \in M \qquad |\sigma| = \mathsf{u}}{\vdash^M \textbf{interface f} :> \sigma = \mathbf{g} \text{ okay}}$$

$\boxed{\Delta; \Gamma \vdash^M_{\mathscr{C}} \mathbf{e} : \tau}$, $\boxed{\Delta; \Gamma \vdash^M_{\mathscr{A}} \mathsf{e} : \sigma}$

$$\text{TA-}\textsc{ModC}\ \frac{(\textbf{module f} : \tau = \mathbf{v}) \in M \qquad \cdot \vdash_{\mathscr{C}} \tau}{\Delta; \Gamma \vdash^M_{\mathscr{A}} \mathbf{f}^{\mathsf{g}} : (\tau)^{\mathscr{A}}}$$

$$\text{TC-}\textsc{ModA}\ \frac{(\mathsf{module\ f} : \sigma = \mathsf{v}) \in M \qquad \cdot \vdash_{\mathscr{A}} \sigma}{\Delta; \Gamma \vdash^M_{\mathscr{C}} \mathsf{f}^{\mathbf{g}} : (\sigma)^{\mathscr{C}}}$$

$$\text{TA-}\textsc{ModI}\ \frac{(\textbf{interface f} :> \sigma = \mathbf{f}') \in M \qquad \cdot \vdash_{\mathscr{A}} \sigma}{\Delta; \Gamma \vdash^M_{\mathscr{A}} \mathbf{f}^{\mathsf{g}} : \sigma}$$

Figure 4.11: New statics for $\lambda^{\mathscr{A}}_{\mathscr{C}}$: programs, modules, and expressions (ii)

Static Semantics. The type system for the mixed calculus is the union of the type systems for $\lambda^{\mathscr{A}}$ and $\lambda_{\mathscr{C}}$ (figures 4.1, B.1–B.3, and 4.3–4.7), along with additional typing rules for converting types between $\lambda^{\mathscr{A}}$ and $\lambda_{\mathscr{C}}$ (figure 4.10), for typing $\lambda^{\mathscr{A}}$ module invocations in $\lambda_{\mathscr{C}}$ expressions, and for typing $\lambda_{\mathscr{C}}$ module invocations in $\lambda^{\mathscr{A}}$ expressions (figure 4.11).

Rule TC-ModA (figure 4.11) types occurrences of $\lambda^{\mathscr{A}}$ module names in $\lambda_{\mathscr{C}}$ expressions. The rule uses the type conversion function $(\cdot)^{\mathscr{C}}$, defined in §3 (p. 9) to give a $\lambda_{\mathscr{C}}$ type to the $\lambda^{\mathscr{A}}$ module invocation. Because $\lambda^{\mathscr{A}}$ types are richer than $\lambda_{\mathscr{C}}$ types—$\lambda^{\mathscr{A}}$ function types carry extra information in the qualifier—the conversion loses information, which may need to be recovered through dynamic checks. For example, given a $\lambda^{\mathscr{A}}$ module g with type $\mathsf{int} \xrightarrow{\mathsf{u}}\!\circ\ \mathsf{int} \xrightarrow{\mathsf{a}}\!\circ\ \mathsf{int}$, the conversion rule assigns it the $\lambda_{\mathscr{C}}$ type $\mathbf{int} \to \mathbf{int} \to \mathbf{int}$. Calculus $\lambda_{\mathscr{C}}$'s type system cannot enforce that the result of applying g be applied at most once, which will need to be checked at run time.

For a $\lambda_{\mathscr{C}}$ module with type τ invoked from a $\lambda^{\mathscr{A}}$ expression, we use the module at type $(\tau)^{\mathscr{A}}$. It would be reasonable for TA-ModC to give it any $\lambda^{\mathscr{A}}$ type in the pre-image of the $\lambda^{\mathscr{A}}$-to-$\lambda_{\mathscr{C}}$ mapping, but $(\cdot)^{\mathscr{A}}$ makes the most permissive, statically safe choice, which is to map all $\lambda_{\mathscr{C}}$ arrows ($\to$) to the unlimited $\lambda^{\mathscr{A}}$ arrow ($\xrightarrow{\mathsf{u}}\!\circ$). Consider:

- If $\mathbf{f} : \mathbf{int} \to \mathbf{int}$ in $\lambda_{\mathscr{C}}$, then $\mathsf{int} \xrightarrow{\mathsf{u}}\!\circ\ \mathsf{int}$ is the right type in $\lambda^{\mathscr{A}}$. There is no reason to limit $\mathbf{f}$ to an affine function type, because $\lambda_{\mathscr{C}}$ does not impose that requirement, and subtyping allows us to use it at $\mathsf{int} \xrightarrow{\mathsf{a}}\!\circ\ \mathsf{int}$, if necessary.

- If $\mathbf{f} : (\mathbf{int} \to \mathbf{int}) \to \mathbf{int}$ in $\lambda_{\mathscr{C}}$, then $(\mathsf{int} \xrightarrow{\mathsf{u}}\!\circ\ \mathsf{int}) \xrightarrow{\mathsf{u}}\!\circ\ \mathsf{int}$ will allow the imported function to be passed unlimited functions but not affine functions. This is a safe choice, because $\lambda_{\mathscr{C}}$'s type system does not tell us whether $\mathbf{f}$ may call its argument more than once.

In the latter case, what if the programmer somehow knows that function $\mathbf{f}$ applies its argument at most once, as in the example of *threadFork*$_{\mathscr{C}}$ (p. 7)? It should not violate $\lambda^{\mathscr{A}}$'s

$$
\begin{array}{rrcl}
\lambda_{\mathscr{C}}\ \textit{terms} & \mathbf{e} & ::= & \cdots \mid {}_{\mathbf{f}}\mathbf{CA}^{\sigma}_{\mathbf{f}}(\mathsf{e}) \\
\lambda^{\mathscr{A}}\ \textit{terms} & \mathsf{e} & ::= & \cdots \mid {}^{\sigma}_{\mathsf{f}}\mathsf{AC}_{\mathbf{f}}(\mathbf{e}) \\
\lambda_{\mathscr{C}}\ \textit{values} & \mathbf{v} & ::= & \cdots \mid {}_{\mathbf{f}}\mathbf{CA}[\ell]^{\sigma}_{\mathbf{f}}(\mathsf{v}) \\
\lambda^{\mathscr{A}}\ \textit{values} & \mathsf{v} & ::= & \cdots \mid {}^{\sigma}_{\mathsf{f}}\mathsf{AC}[\,]_{\mathbf{f}}(\mathbf{v}) \\
\lambda_{\mathscr{C}}\ \textit{evaluation contexts} & \mathbf{E} & ::= & \cdots \mid {}_{\mathbf{f}}\mathbf{CA}^{\sigma}_{\mathbf{f}}(\mathsf{E}) \\
\lambda^{\mathscr{A}}\ \textit{evaluation contexts} & \mathsf{E} & ::= & \cdots \mid {}^{\sigma}_{\mathsf{f}}\mathsf{AC}_{\mathbf{f}}(\mathbf{E}) \\
\textit{configurations} & C & ::= & (s, \mathbf{e}) \mid \mathbf{blame\,f} \\
\textit{answers} & A & ::= & (s, \mathbf{v}) \mid \mathbf{blame\,f} \\
\textit{stores} & s & ::= & \{\} \mid s \uplus \{\ell \mapsto \mathbf{v}\} \mid s \uplus \{\ell \mapsto \mathsf{v}\}
\end{array}
$$

Figure 4.12: Dynamics of $\lambda^{\mathscr{A}}_{\mathscr{C}}$: run-time syntax (i)

invariants to pass an affine function to *threadFork*$_{\mathscr{C}}$, but $\lambda^{\mathscr{A}}$ cannot know this. Therefore, rule TA-ModC gives $\lambda_{\mathscr{C}}$ modules a conservative $\lambda^{\mathscr{A}}$ type that requires no run-time checks. We can use an **interface** module to coerce a $\lambda_{\mathscr{C}}$ module's type τ to a more permissive $\lambda^{\mathscr{A}}$ type in the pre-image of τ, and this, too, requires a dynamic check.

Operational Semantics. We extend the syntax of our mixed calculus with several new forms (figure 4.12). Whereas our source syntax segregates the two subcalculi into separate modules, module invocation reduces to the body of the module, which leads expressions of both subcalculi to nest at run time. Rather than allow $\lambda^{\mathscr{A}}$ terms to appear directly in $\lambda_{\mathscr{C}}$, and vice versa, we need a way to cordon off terms from one calculus embedded in the other and to ensure that the interaction is well-behaved. We call these new expression forms *boundaries*.

The new run-time syntax includes both boundary expressions ${}^{\sigma}_{\mathsf{f}}\mathsf{AC}_{\mathbf{g}}(\mathbf{e})$ for embedding $\lambda_{\mathscr{C}}$ expressions in $\lambda^{\mathscr{A}}$ and boundary expressions ${}_{\mathbf{f}}\mathbf{CA}^{\sigma}_{\mathsf{g}}(\mathsf{e})$ for embedding $\lambda^{\mathscr{A}}$ expressions in $\lambda_{\mathscr{C}}$. Each of these forms has a superscript σ, written on the $\lambda^{\mathscr{A}}$ side, which represents a contract between the two modules that gave rise to the nested expression. Some contracts, for example int, are fully enforced by both type systems. Other contracts, such as int $\xrightarrow{\mathsf{a}}\!\!\circ$ int, require dynamic checks. The type system guarantees that such a function receives and returns only integers, but this type also imposes an obligation on the negative party to apply the function at most once, which the $\lambda_{\mathscr{C}}$ type system alone does not enforce.

The right subscript of a boundary is a module name in the inner subcalculus, representing the positive party to the contract: It promises that if the enclosed subexpression reduces to a value, then the value will obey contract σ. The left subscript is the negative party, which promises to treat the resulting value properly. In particular, if the contract is affine, then the negative party promises to use the resulting value at most once.

Boundaries first arise when a module in one calculus refers to a module in the other calculus. When the name of a $\lambda_{\mathscr{C}}$ module appears in a $\lambda^{\mathscr{A}}$ term, A-ModC (figure 4.13)

$$\text{(C-CxtA)} \quad (s, \mathbf{E}[\mathsf{e}]_{\mathscr{A}}) \underset{M}{\longmapsto} (s', \mathbf{E}[\mathsf{e}']_{\mathscr{A}}) \quad \text{if } (s, \mathsf{e}) \underset{M}{\longmapsto} (s', \mathsf{e}')$$

$$\text{(C-ModA)} \quad (s, \mathsf{f}^{\mathbf{g}}) \underset{M}{\longmapsto} (s,\ \underset{\mathbf{g}\,\mathsf{f}}{\mathbf{CA}}^{\sigma}(\mathsf{f})) \quad (\mathsf{module}\ \mathsf{f} : \sigma = \mathsf{v}) \in M$$

$$\text{(A-ModC)} \quad (s, \mathbf{f}^{\mathsf{g}}) \underset{M}{\longmapsto} (s,\ {}^{(\tau)^{\mathscr{A}}}\underset{\mathsf{g}\,\mathbf{f}}{\mathsf{AC}}\,(\mathbf{f})) \quad (\mathbf{module}\ \mathbf{f} : \tau = \mathbf{v}) \in M$$

$$\text{(A-ModI)} \quad (s, \mathbf{f}^{\mathsf{g}}) \underset{M}{\longmapsto} (s,\ {}^{\sigma}\underset{\mathsf{g}\,\mathbf{f}}{\mathsf{AC}}\,(\mathbf{f}')) \quad (\mathbf{interface}\ \mathbf{f} :> \sigma = \mathbf{f}') \in M$$

$$\text{(C-Wrap)} \quad (s,\ \underset{\mathbf{f}\,\mathsf{g}}{\mathbf{CA}}^{\sigma}(\mathsf{v})) \underset{M}{\longmapsto} \mathit{coerce}_{\mathscr{C}}(s, \sigma, \mathsf{v}, \mathbf{f}, \mathsf{g})$$

$$\text{(A-Wrap)} \quad (s,\ {}^{\sigma}\underset{\mathsf{f}\,\mathbf{g}}{\mathsf{AC}}\,(\mathbf{v})) \underset{M}{\longmapsto} \mathit{coerce}_{\mathscr{A}}(s, \sigma, \mathbf{v}, \mathsf{f}, \mathbf{g})$$

$$\text{(C-}B\text{-A)} \quad (s,\ \underset{\mathbf{f}\,\mathsf{g}}{\mathbf{CA}}[\ell]^{\forall\alpha^{\mathsf{q}}.\sigma}(\mathsf{v})[\tau]) \underset{M}{\longmapsto} \mathit{check}(s, \ell, |\sigma|,\ \underset{\mathbf{f}\,\mathsf{g}}{\mathbf{CA}}^{\sigma[(\tau)^{\mathscr{A}}/\alpha^{\mathsf{q}}]}\big(\mathsf{v}[(\tau)^{\mathscr{A}}]\big), \mathbf{f})$$

$$\text{(C-}\beta\text{-A)} \quad (s,\ \underset{\mathbf{f}\,\mathsf{g}}{\mathbf{CA}}[\ell]^{\sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2}(\mathsf{v}_1)\ \mathbf{v_2}) \underset{M}{\longmapsto} \mathit{check}\Big(s, \ell, \mathsf{q},\ \underset{\mathbf{f}\,\mathsf{g}}{\mathbf{CA}}^{\sigma_2}\Big(\mathsf{v}_1\ {}^{\sigma_1}\underset{\mathsf{g}\,\mathbf{f}}{\mathsf{AC}}\,(\mathbf{v_2})\Big), \mathbf{f}\Big)$$

$$\text{(A-}B\text{-C)} \quad (s,\ {}^{\forall\alpha^{\mathsf{q}}.\sigma}\underset{\mathsf{f}\,\mathbf{g}}{\mathsf{AC}}[\,](\mathbf{v})[\sigma_{\mathsf{a}}]) \underset{M}{\longmapsto} (s,\ {}^{\sigma[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q}}]}\underset{\mathsf{f}\,\mathbf{g}}{\mathsf{AC}}\,(\mathbf{v}[(\sigma_{\mathsf{a}})^{\mathscr{C}}]))$$

$$\text{(A-}\beta\text{-C)} \quad (s,\ {}^{\sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2}\underset{\mathsf{f}\,\mathbf{g}}{\mathsf{AC}}[\,](\mathbf{v_1})\ \mathsf{v}_2) \underset{M}{\longmapsto} \Big(s,\ {}^{\sigma_2}\underset{\mathsf{f}\,\mathbf{g}}{\mathsf{AC}}\Big(\mathbf{v_1}\ \ \underset{\mathbf{g}\,\mathsf{f}}{\mathbf{CA}}^{\sigma_1}(\mathsf{v}_2)\Big)\Big)$$

$$\mathit{coerce}_{\mathscr{C}}(s, \sigma, \mathsf{v}, \mathbf{f}, \mathsf{g}) = \begin{cases} (s, \lceil z \rceil) & \text{if } \mathsf{v} = \lceil z \rceil \\ (s, \mathbf{v}') & \text{if } \mathsf{v} = {}^{\{\tau^{\circ}\}}_{\mathsf{g}'}\mathsf{AC}[\,]_{\mathbf{f}'}(\mathbf{v}') \\ (s \uplus \{\ell \mapsto \mathbf{BLSSD}\},\ \underset{\mathbf{f}\,\mathsf{g}}{\mathbf{CA}}[\ell]^{\sigma}(\mathsf{v})) & \text{otherwise} \end{cases}$$

$$\mathit{coerce}_{\mathscr{A}}(s, \sigma, \mathbf{v}, \mathsf{f}, \mathbf{g}) = \begin{cases} (s, \lceil z \rceil) & \text{if } \mathbf{v} = \lceil z \rceil \\ \mathit{check}(s, \ell, |\sigma^{\circ}|, \mathsf{v}', \mathbf{g}') & \text{if } \mathbf{v} = {}_{\mathbf{g}'}\mathbf{CA}[\ell]^{\sigma^{\circ}}_{\mathsf{f}'}(\mathsf{v}') \\ (s,\ {}^{\sigma}\underset{\mathsf{f}\,\mathbf{g}}{\mathsf{AC}}[\,](\mathbf{v})) & \text{otherwise} \end{cases}$$

$$\mathit{check}(s, \ell, \mathsf{q}, e, \mathbf{f}) = \begin{cases} (s, e) & \text{if } \mathsf{q} = \mathsf{u} \\ (s' \uplus \{\ell \mapsto \mathbf{DFNCT}\}, e) & \text{if } s = s' \uplus \{\ell \mapsto \mathbf{BLSSD}\} \\ (s, \mathbf{blame}\,\mathbf{f}) & \text{otherwise} \end{cases}$$

Figure 4.13: Dynamics of $\lambda^{\mathscr{A}}_{\mathscr{C}}$: reduction relation (ii)

wraps the module name with an AC boundary, using the $\lambda^{\mathscr{A}}$-conversion of the module's type τ as the contract. For interface modules, the contract is as declared by the interface, and the name of the interface is the positive party (A-ModI). From the other direction, a $\lambda^{\mathscr{A}}$ module invoked from a $\lambda_{\mathscr{C}}$ expression is wrapped in a **CA** boundary by rule C-ModA.

We add evaluation contexts for reduction under boundaries, which means it is now possible to construct a $\lambda_{\mathscr{C}}$ evaluation context with a $\lambda^{\mathscr{A}}$ hole, and vice versa. If the expression under a boundary reduces to a value, it is time to apply the boundary's contract to the value. There are three possibilities:

- Some values, such as integers, always satisfy the contract, so the boundary is discarded.
- Functional values and opaque affine values must have their checks deferred: functions until application time, and opaque values until they pass back into their original subcalculus. For deferred checks, we leave the value in a "sealed" boundary, ${}_{\mathbf{f}}\mathbf{CA}[\ell]^{\sigma}_{\mathsf{g}}(\mathsf{v})$ or ${}^{\sigma}_{\mathsf{f}}\mathsf{AC}[\,]_{\mathbf{g}}(\mathbf{v})$, which is itself a value form.
- When a previously sealed opaque value reaches a boundary back to its original subcalculus, both that boundary and the sealed boundary are discarded.

Rule C-Wrap implements contract application for $\lambda^{\mathscr{A}}$ values embedded in $\lambda_{\mathscr{C}}$ expressions, as indicated by metafunction $coerce_{\mathscr{C}}$. The first case of $coerce_{\mathscr{C}}$ handles immediate checks, and its second case unseals previously sealed $\lambda_{\mathscr{C}}$ values that have returned home. The second case of $coerce_{\mathscr{C}}$ seals and *blesses* a $\lambda^{\mathscr{A}}$ value, by allocating a location ℓ, to which it stores a distinguished value **blssd**; it adds this location to the boundary, which marks the sealed value as not yet used. This corresponds directly to the reference cell allocated by *makeAffineFunContract* in §3.

Rule A-Wrap implements contracts for $\lambda_{\mathscr{C}}$ values in $\lambda^{\mathscr{A}}$ expressions. Metafunction $coerce_{\mathscr{A}}$'s first case is the same as $coerce_{\mathscr{C}}$'s, and the third case seals a value for deferred checking; it need not allocate a location to track the usage of a $\lambda_{\mathscr{C}}$ value. The third case unseals a previously sealed $\lambda^{\mathscr{A}}$ value on its way back to $\lambda^{\mathscr{A}}$, and this requires checking that an affine value has not been previously unsealed. This step is specified by metafunction *check*, which also has three cases. Unlimited values are unsealed with no check. If an affine value remains blessed, *check* updates the store to mark it "defunct" and returns the unsealed value. If, on the other hand, there is an attempt to unseal a defunct affine value, *check* blames the negative party. This is the key dynamic check that enforces the affine invariant for non-functional values.

Rules C-B-A, C-β-A, A-B-C, and A-β-C all handle sealed abstractions, which are unsealed when they are applied. For sealed $\lambda^{\mathscr{A}}$ abstractions, the seal location ℓ must be checked, to ensure that an affine function or type abstraction is not unsealed and applied more than once. This is the dynamic check that enforces the affine invariant for functions.

5 Proving Type Soundness

The presence of strong updates means that aliasing a location can result in a program getting "stuck": if an aliased location is updated at a different type, reading from the alias produces a value of unexpected type. Calculus $\lambda^{\mathscr{A}}$'s type system prevents this, but adding $\lambda_{\mathscr{C}}$ means that a $\lambda^{\mathscr{A}}$ value may be aliased outside $\lambda^{\mathscr{A}}$. Our soundness criterion is that no program that gets stuck is assigned a type. In particular, all aliasing of affine values is either prevented by $\lambda^{\mathscr{A}}$'s type system or detected by a contract at run time.

In order to prove a Wright-Felleisen–style type soundness theorem (1994), we must identify precisely what property is preserved by subject reduction. We use an internal type system to track which portions of the store are reachable from $\lambda^{\mathscr{A}}$ values that have flowed into $\lambda_{\mathscr{C}}$. Under this type system, configurations enjoy standard progress and preservation, which allows us to state and prove a syntactic type soundness theorem using the internal type system's configuration typing judgment.

Figure 5.1 shows the new syntax for the internal type system. A store type (Σ) maps locations to types in either subcalculus, or to "protected" types of the form $[\sigma]^{\ell'}$. A location ℓ mapped to a protected type $[\sigma]^{\ell'}$ means that location ℓ may appear only under blessed **CA** boundaries sealed by ℓ'. In particular, the store-splitting partial function ($\boxplus$) defined in figure 5.2 allows protected locations to be duplicated arbitrarily; but as we will see, they can only be used to type locations in terms that are protected by a contract. Store splitting also duplicates locations containing $\lambda_{\mathscr{C}}$ values, but it requires locations containing $\lambda^{\mathscr{A}}$ values, both unlimited and affine, to go only one way or the other. This ensures that such locations appear only once in a well-typed term, which ensures the safety of strong updates.

5.1 The Internal Type System

Figure 5.2 also defines store typing. The type of a store contains the types of all its locations. Additionally, each location ℓ containing a $\lambda^{\mathscr{A}}$ value with type σ may appear in the store type, non-deterministically, as $\ell{:}\sigma$ or as $\ell{:}[\sigma]^{\ell'}$ for any location ℓ'.

We may apply protection to a whole store, as in figure 5.3, in which case it protects a $\lambda^{\mathscr{A}}$ locations that are not already protected. We also define the qualifier of a store type: if it maps any location to a $\lambda^{\mathscr{A}}$ type, then the qualifier is a; otherwise it is u.

The new expression type judgments $\Sigma; \Delta; \Gamma \triangleright^{M}_{\mathscr{C}} \mathbf{e} : \tau$ and $\Sigma; \Delta; \Gamma \triangleright^{M}_{\mathscr{A}} \mathsf{e} : \sigma$ (figure 5.4) add a store type to the context, which is used to type locations that appear in run-time expressions, by rule RTA-Loc. We type boundary expressions by rules RTC-Boundary and RTA-Boundary, each of which requires the $\lambda^{\mathscr{A}}$ type σ in the premiss (resp., conclusion) to convert to the $\lambda_{\mathscr{C}}$ type $(\sigma)^{\mathscr{C}}$ in the conclusion (resp., premiss). Also, notably, both drop the type and value contexts Δ and Γ (resp. Δ and Γ) in the premiss. Rule RTA-Sealed is used to type sealed AC boundaries; beyond RTA-Boundary, it requires that the sealed value have a "wrappable" type $\tau^{\mathbf{w}}$. which includes functions, type functions, and opaque types; this ensures that transparent values such as integers cannot be typed under sealed boundaries.

Three rules are used to type sealed $\lambda^{\mathscr{A}}$ values. For unlimited values, RTC-Sealed requires that the type of the $\lambda^{\mathscr{A}}$ value be a wrappable type. For sealed boundaries ${}_{\mathbf{f}}\mathbf{CA}[\ell]^{\sigma}_{\mathsf{g}}(\mathsf{v})$, which contain values of affine type, either rule RTC-Blessed or RTC-Defunct applies, depending on the type of the $\lambda_{\mathscr{C}}$ value stored at the seal location ℓ. In particular, we assume distinct types $\mathbb{B}$ and $\mathbb{D}$ for the special seal values **BLSSD** and **DFNCT**. If location ℓ maps to $\mathbb{B}$—that is, it contains **BLSSD**—then we expose any locations protected by that same location ℓ when typing the value v that appears under the seal. This means that when a sealed, affine value is duplicated by $\lambda_{\mathscr{C}}$, all locations appearing in that value may still type, provided they

$$\begin{array}{rrcl}
\textit{store contexts} & \Sigma & ::= & \cdot \mid \Sigma, \ell{:}\tau \mid \Sigma, \ell{:}\sigma \mid \Sigma, \ell{:}[\sigma]^{\ell'} \\
\textit{wrappable } \lambda_{\mathscr{C}} \textit{ types} & \tau^{\mathbf{w}} & ::= & \forall\alpha.\,\tau \mid \tau_1 \to \tau_2 \mid \tau^{\mathbf{o}} \\
\textit{wrappable } \lambda^{\mathscr{A}} \textit{ types} & \sigma^{\mathsf{w}} & ::= & \forall\alpha^{\mathsf{q}}.\,\sigma \mid \sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2 \mid \sigma^{\circ}
\end{array}$$

Figure 5.1: Internal type system: new syntax (i)

$\boxed{\Sigma \boxplus \Sigma = \Sigma}$

$$\frac{\Sigma_1 \boxplus \Sigma_2 = \Sigma_3}{\Sigma_1, \ell{:}\tau \boxplus \Sigma_2, \ell{:}\tau = \Sigma_3, \ell{:}\tau} \qquad \frac{\Sigma_1 \boxplus \Sigma_2 = \Sigma_3}{\Sigma_1, \ell{:}\sigma \boxplus \Sigma_2 = \Sigma_3, \ell{:}\sigma} \qquad \frac{\Sigma_1 \boxplus \Sigma_2 = \Sigma_3}{\Sigma_1 \boxplus \Sigma_2, \ell{:}\sigma = \Sigma_3, \ell{:}\sigma}$$

$$\frac{\Sigma_1 \boxplus \Sigma_2 = \Sigma_3}{\Sigma_1, \ell{:}[\sigma]^{\ell'} \boxplus \Sigma_2, \ell{:}[\sigma]^{\ell'} = \Sigma_3, \ell{:}[\sigma]^{\ell'}}$$

$\boxed{\Sigma \rhd^{M} s : \Sigma}$

$$\textsc{S-Empty}\quad \frac{}{\Sigma \rhd^{M} \{\} : \cdot} \qquad \textsc{S-CLoc}\quad \frac{\Sigma_1 \rhd^{M} s : \Sigma' \qquad \Sigma_2; \cdot; \cdot \rhd^{M}_{\mathscr{C}} \mathbf{v} : \tau}{\Sigma_1 \boxplus \Sigma_2 \rhd^{M} s \uplus \{\ell \mapsto \mathbf{v}\} : (\Sigma', \ell{:}\tau)} \qquad \textsc{S-ALoc}\quad \frac{\Sigma_1 \rhd^{M} s : \Sigma' \qquad \Sigma_2; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{v} : \sigma}{\Sigma_1 \boxplus \Sigma_2 \rhd^{M} s \uplus \{\ell \mapsto \mathsf{v}\} : (\Sigma', \ell{:}\sigma)}$$

$$\textsc{S-ALocProt}\quad \frac{\Sigma_1 \rhd^{M} s : \Sigma' \qquad \Sigma_2; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{v} : \sigma}{\Sigma_1 \boxplus \Sigma_2 \rhd^{M} s \uplus \{\ell \mapsto \mathsf{v}\} : (\Sigma', \ell{:}[\sigma]^{\ell'})}$$

Figure 5.2: Internal type system: store splitting and typing (ii)

$\boxed{[\Sigma]^{\ell} = \Sigma}$

$$[\cdot]^{\ell} = \cdot \qquad [\Sigma, \ell'{:}\tau]^{\ell} = [\Sigma]^{\ell}, \ell'{:}\tau \qquad [\Sigma, \ell'{:}\sigma]^{\ell} = [\Sigma]^{\ell}, \ell'{:}[\sigma]^{\ell} \qquad [\Sigma, \ell'{:}[\sigma]^{\ell''}]^{\ell} = [\Sigma]^{\ell}, \ell'{:}[\sigma]^{\ell''}$$

$\boxed{|\Sigma| = \mathsf{q}}$

$$|\cdot| = \mathsf{u} \qquad |\Sigma, \ell{:}\sigma| = \mathsf{a} \qquad |\Sigma, \ell{:}[\sigma]^{\ell'}| = |\Sigma| \qquad |\Sigma, \ell{:}\tau| = |\Sigma|$$

Figure 5.3: Internal type system: store protection and qualifiers (iii)

$\boxed{\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{C}} \mathbf{e} : \tau}$

$$\textsc{RTC-Boundary}\quad \frac{\Sigma; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{e} : \sigma}{\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{C}} \underset{\mathsf{f}\,\mathsf{g}}{\mathbf{CA}}{}^{\sigma}(\mathsf{e}) : (\sigma)^{\mathscr{C}}}
\qquad
\textsc{RTC-Sealed}\quad \frac{\Sigma; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{v} : \sigma^{\mathsf{w}} \qquad |\sigma^{\mathsf{w}}| = \mathsf{u}}{\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{C}} \underset{\mathsf{f}\,\mathsf{g}}{\mathbf{CA}}[\ell]^{\sigma^{\mathsf{w}}}(\mathsf{v}) : (\sigma^{\mathsf{w}})^{\mathscr{C}}}$$

$$\textsc{RTC-Blessed}\quad \frac{\Sigma_1, \Sigma_2; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{v} : \sigma^{\mathsf{w}} \qquad |\sigma^{\mathsf{w}}| = \mathsf{a}}{[\Sigma_1]^{\ell}, \ell{:}\mathbb{B}, [\Sigma_2]^{\ell}; \Delta; \Gamma \rhd^{M}_{\mathscr{C}} \underset{\mathsf{f}\,\mathsf{g}}{\mathbf{CA}}[\ell]^{\sigma^{\mathsf{w}}}(\mathsf{v}) : (\sigma^{\mathsf{w}})^{\mathscr{C}}}$$

$$\textsc{RTC-Defunct}\quad \frac{|\sigma^{\mathsf{w}}| = \mathsf{a}}{[\Sigma_1]^{\ell}, \ell{:}\mathbb{D}, [\Sigma_2]^{\ell}; \Delta; \Gamma \rhd^{M}_{\mathscr{C}} \underset{\mathsf{f}\,\mathsf{g}}{\mathbf{CA}}[\ell]^{\sigma^{\mathsf{w}}}(\mathsf{v}) : (\sigma^{\mathsf{w}})^{\mathscr{C}}}$$

$\boxed{\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{e} : \sigma}$

$$\textsc{RTA-Loc}\quad \frac{}{\Sigma_1, \ell{:}\sigma, \Sigma_2; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \ell : \sigma\ \mathsf{ref}}
\qquad
\textsc{RTA-Boundary}\quad \frac{\Sigma; \cdot; \cdot \rhd^{M}_{\mathscr{C}} \mathbf{e} : (\sigma)^{\mathscr{C}}}{\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} {}^{\sigma}\underset{\mathsf{f}\,\mathsf{g}}{\mathsf{AC}}(\mathbf{e}) : \sigma}
\qquad
\textsc{RTA-Sealed}\quad \frac{\Sigma; \cdot; \cdot \rhd^{M}_{\mathscr{C}} \mathbf{v} : (\sigma)^{\mathscr{C}} \qquad (\sigma)^{\mathscr{C}} = \tau^{\mathbf{w}}}{\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} {}^{\sigma}\underset{\mathsf{f}\,\mathsf{g}}{\mathsf{AC}}[\,](\mathbf{v}) : \sigma}$$

$\boxed{\mathrm{ty}_{\mathscr{C}}(\mathbf{c}) = \tau}$

$$\mathrm{ty}_{\mathscr{C}}(\mathbf{BLSSD}) = \mathbb{B} \qquad\qquad \mathrm{ty}_{\mathscr{C}}(\mathbf{DFNCT}) = \mathbb{D}$$

Figure 5.4: Internal type system: new expressions and constants (iv)

$\boxed{\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{e} : \sigma}$

RTA-Subsume
$$\frac{\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{e} : \sigma \qquad \sigma <: \sigma'}{\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{e} : \sigma'}$$

RTA-TLam
$$\frac{\Sigma; \Delta, \alpha^{q}; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{v} : \sigma}{\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \Lambda\alpha^{\mathsf{q}}.\, \mathsf{v} : \forall\alpha^{\mathsf{q}}.\, \sigma}$$

RTA-TApp
$$\frac{\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{e} : \forall\alpha^{\mathsf{q}}.\, \sigma' \qquad \Delta \vdash_{\mathscr{A}} \sigma \qquad |\sigma| \sqsubseteq \mathsf{q}}{\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{e}[\sigma] : \sigma'[\sigma/\alpha^{\mathsf{q}}]}$$

RTA-Lam
$$\frac{\Sigma; \Delta; \Gamma, \mathsf{x} : \sigma \rhd^{M}_{\mathscr{A}} \mathsf{e} : \sigma' \qquad \Delta \vdash_{\mathscr{A}} \sigma \qquad \left|\Gamma|_{\mathrm{FV}(\lambda \mathsf{x}:\sigma.\, \mathsf{e})}\right| \sqcup \left|\Sigma|_{\mathrm{FL}(\lambda \mathsf{x}:\sigma.\, \mathsf{e})}\right| = \mathsf{q}}{\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \lambda \mathsf{x}{:}\sigma.\, \mathsf{e} : \sigma \overset{\mathsf{q}}{\multimap} \sigma'}$$

RTA-App
$$\frac{\Sigma_1; \Delta; \Gamma_1 \rhd^{M}_{\mathscr{A}} \mathsf{e}_1 : \sigma' \overset{\mathsf{q}}{\multimap} \sigma \qquad \Sigma_2; \Delta; \Gamma_2 \rhd^{M}_{\mathscr{A}} \mathsf{e}_2 : \sigma'}{\Sigma_1 \boxplus \Sigma_2; \Delta; \Gamma_1 \boxplus \Gamma_2 \rhd^{M}_{\mathscr{A}} \mathsf{e}_1\ \mathsf{e}_2 : \sigma}$$

RTA-Pair
$$\frac{\Sigma_2; \Delta; \Gamma_1 \rhd^{M}_{\mathscr{A}} \mathsf{e}_1 : \sigma_1 \qquad \Sigma_2; \Delta; \Gamma_2 \rhd^{M}_{\mathscr{A}} \mathsf{e}_2 : \sigma_2}{\Sigma_1 \boxplus \Sigma_2; \Delta; \Gamma_1 \boxplus \Gamma_2 \rhd^{M}_{\mathscr{A}} \langle \mathsf{e}_1, \mathsf{e}_2 \rangle : \sigma_1 \otimes \sigma_2}$$

RTA-Let
$$\frac{\Sigma_1; \Delta; \Gamma_1 \rhd^{M}_{\mathscr{A}} \mathsf{e}_1 : \sigma_{\mathsf{x}} \otimes \sigma_{\mathsf{y}} \qquad \Sigma_2; \Delta; \Gamma_2, \mathsf{x} : \sigma_{\mathsf{x}}, \mathsf{y} : \sigma_{\mathsf{y}} \rhd^{M}_{\mathscr{A}} \mathsf{e}_2 : \sigma}{\Sigma_1 \boxplus \Sigma_2; \Delta; \Gamma_1 \boxplus \Gamma_2 \rhd^{M}_{\mathscr{A}} \mathsf{let}\ \langle \mathsf{x}, \mathsf{y} \rangle = \mathsf{e}_1\ \mathsf{in}\ \mathsf{e}_2 : \sigma}$$

RTA-Con
$$\frac{}{\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{c} : \mathrm{ty}_{\mathscr{A}}(\mathsf{c})}$$

RTA-If0
$$\frac{\Sigma_1; \Delta; \Gamma_1 \rhd^{M}_{\mathscr{A}} \mathsf{e}_1 : \mathsf{int} \qquad \Sigma_2; \Delta; \Gamma_2 \rhd^{M}_{\mathscr{A}} \mathsf{e}_2 : \tau \qquad \Sigma_1; \Gamma_2 \rhd^{M}_{\mathscr{A}} \mathsf{e}_3 : \tau}{\Sigma_1 \boxplus \Sigma_2; \Delta; \Gamma_1 \boxplus \Gamma_2 \rhd^{M}_{\mathscr{A}} \mathsf{if0}\ \mathsf{e}_1\ \mathsf{e}_2\ \mathsf{e}_3 : \tau}$$

RTA-Var
$$\frac{\Gamma(\mathsf{x}) = \sigma}{\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{x} : \sigma}$$

RTA-Mod
$$\frac{\mathsf{module}\ \mathsf{f} : \sigma = \mathsf{v} \in M \qquad \cdot \vdash_{\mathscr{A}} \sigma}{\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{f} : \sigma}$$

RTA-ModC
$$\frac{\mathbf{module\ f} : \tau = \mathbf{v} \in M \qquad \cdot \vdash_{\mathscr{C}} \tau}{\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \mathbf{f} : (\tau)^{\mathscr{A}}}$$

RTA-ModI
$$\frac{\mathbf{interface\ f} :> \sigma = \mathbf{g} \in M \qquad \cdot \vdash_{\mathscr{A}} \sigma}{\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \mathbf{f} : \sigma}$$

Figure 5.5: Internal type system: old $\lambda^{\mathscr{A}}$ expressions (v)

$\boxed{\Sigma; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{e} : \tau}$

$$\textsc{RTC-TLam}\ \frac{\Sigma; \Delta, \alpha; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{v} : \tau}{\Sigma; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{\Lambda}\alpha.\, \mathbf{v} : \forall \alpha.\, \tau}
\qquad
\textsc{RTC-TApp}\ \frac{\Sigma; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{e} : \forall \alpha.\, \tau' \qquad \Delta \vdash_{\mathscr{C}} \tau}{\Sigma; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{e}[\tau] : \tau'[\tau/\alpha]}$$

$$\textsc{RTC-Lam}\ \frac{\Sigma; \Delta; \Gamma, \mathbf{x}{:}\tau \triangleright^M_{\mathscr{C}} \mathbf{e} : \tau' \qquad \Delta \vdash_{\mathscr{C}} \tau \qquad \left|\Sigma|_{\mathrm{FL}(\lambda \mathbf{x}:\tau.\, \mathbf{e})}\right| = \mathsf{u}}{\Sigma; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \lambda \mathbf{x}{:}\tau.\, \mathbf{e} : \tau \to \tau'}$$

$$\textsc{RTC-App}\ \frac{\Sigma_1; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{e_1} : \tau' \to \tau \qquad \Sigma_2; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{e_2} : \tau'}{\Sigma_1 \boxplus \Sigma_2; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{e_1}\ \mathbf{e_2} : \tau}
\qquad
\textsc{RTC-Con}\ \frac{}{\Sigma; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{c} : \mathrm{ty}_{\mathscr{C}}(\mathbf{c})}$$

$$\textsc{RTC-If0}\ \frac{\Sigma_1; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{e_1} : \mathbf{int} \qquad \Sigma_2; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{e_2} : \tau \qquad \Sigma_2; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{e_3} : \tau}{\Sigma_1 \boxplus \Sigma_2; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{if0}\, \mathbf{e_1}\ \mathbf{e_2}\ \mathbf{e_3} : \tau}$$

$$\textsc{RTC-ModA}\ \frac{\mathsf{module\,f} : \sigma = \mathsf{v} \in M \qquad \cdot \vdash_{\mathscr{A}} \sigma}{\Sigma; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathsf{f} : (\sigma)^{\mathscr{C}}}
\qquad
\textsc{RTC-Var}\ \frac{\Gamma(\mathbf{x}) = \tau}{\Sigma; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{x} : \tau}
\qquad
\textsc{RTC-Mod}\ \frac{\mathbf{module\,f} : \tau = \mathbf{v} \in M \qquad \cdot \vdash_{\mathscr{C}} \tau}{\Sigma; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{f} : \tau}$$

Figure 5.6: Internal type system: old $\lambda_{\mathscr{C}}$ expressions (vi)

$\boxed{\triangleright^M C : \tau}$

$$\textsc{Conf}\ \frac{(\forall m \in M) \vdash^M m \text{ okay} \qquad \Sigma_1 \triangleright^M s : \Sigma_1 \boxplus \Sigma_2 \qquad \Sigma_2; \cdot; \cdot \triangleright^M_{\mathscr{C}} \mathbf{e} : \tau}{\triangleright^M (s, \mathbf{e}) : \tau}$$

$$\textsc{Blame}\ \frac{}{\triangleright^M \mathbf{blame\,f} : \tau}$$

Figure 5.7: Internal type system: configurations (vii)

all remain sealed. When one instance of the sealed value is unwrapped, location ℓ is updated to have type $\mathbb{D}$, which means that we no longer attempt to type other instances of the sealed value at all, and just give them the type indicated by the boundary. This is safe because the contract checking in the operational semantics ensures that such values can never be unwrapped.

Figures 5.5 and 5.6 update the type rules for the old expression forms for the internal type system. These rules extend each of the old rules with a store context, which is split for multiplicative forms such as application in $\lambda_{\mathscr{C}}$ as well as $\lambda^{\mathscr{A}}$. The only other change is for typing λ abstractions. For $\lambda^{\mathscr{A}}$ (RTA-LAM), we use not only the value context but the store context to determine the qualifier q in the arrow ($\xrightarrow{\mathsf{q}}\!\circ$) type. For $\lambda_{\mathscr{C}}$, rule RTC-LAM requires that the term contain no unprotected locations containing $\lambda^{\mathscr{A}}$ values.

Finally, figure 5.7 gives the type rule for configurations. It requires that the store s have some type $\Sigma_1 \boxplus \Sigma_2$, where Σ_1 is sufficient context for that store typing, and Σ_2 is used to type the configuration's expression $\mathbf{e}$.

Conventions. We define the free variables of an expression $\mathbf{e}$, written $\mathrm{FV}(\mathbf{e})$ inductively in the conventional way (and likewise for $\lambda^{\mathscr{A}}$); however, we consider the module names in a program to be syntactically distinct from the λ- and let-bound variables, and we take the free variables to exclude module names.

The free locations of an expression $\mathbf{e}$, written $\mathrm{FL}(\mathbf{e})$, is the set of locations (ℓ) that occur in $\mathbf{e}$ (and likewise for $\lambda^{\mathscr{A}}$). Note that there are no binders for locations at the expression level.

We note that exchange and weakening of store, type, and value contexts is implicit in our type syste, by inspection of the type rules: All variable, type variable, and location lookup rules look anywhere in the environment, and ignore the rest. Conversely, it should be apparent that any assumption in an environment that is not free in the subject is not needed to type the subject. We are justified in discarding such assumptions.

We follow Barendregt's convention for evasive relettering.

Road Map. In §5.2, we prove several simple properties of types, type conversion, stores, and store contexts. In §5.3, we relate the external type system from §4.2 with the internal type system, showing that programs and expressions that type in the external type system also type in the internal type system. Section 5.4 contains several lemmas about evaluation contexts and typing of terms in the hole, and about substitution. In §5.5, we prove our presevation theorem, followed by our progress theorem in §5.6. We finish with our main theorem in §5.7.

5.2 Properties of Types and Stores

Lemma 5.2.1 (Type conversion is faithful)**.**

(i) For any type τ, $((\tau)^{\mathscr{A}})^{\mathscr{C}} = \tau$.

(ii) For any opaque type σ°, $((\sigma^{\circ})^{\mathscr{C}})^{\mathscr{A}} = \sigma^{\circ}$.

(iii) For any type σ, $|((\sigma)^{\mathscr{C}})^{\mathscr{A}}| \sqsubseteq |\sigma|$.

(*iv*) *For any types σ and σ°, if $(\sigma)^{\mathscr{C}} = (\sigma^{\circ})^{\mathscr{C}}$ then $\sigma = \sigma^{\circ}$.*

Proof.

(*i*) By induction on the structure of τ.

(*ii*) $((\sigma^{\circ})^{\mathscr{C}})^{\mathscr{A}} = (\{\sigma^{\circ}\})^{\mathscr{A}} = \sigma^{\circ}$.

(*iii*) By induction on the structure of σ:

Case int.

$|((\mathsf{int})^{\mathscr{C}})^{\mathscr{A}}| = |\mathsf{int}| = \mathsf{u}$.

Case $\sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2$.

$|((\sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2)^{\mathscr{C}})^{\mathscr{A}}| = |((\sigma_1)^{\mathscr{C}} \to (\sigma_2)^{\mathscr{C}})^{\mathscr{A}}| = |((\sigma_1)^{\mathscr{C}})^{\mathscr{A}} \overset{\mathsf{u}}{\multimap} ((\sigma_2)^{\mathscr{C}})^{\mathscr{A}}| = \mathsf{u} \sqsubseteq \mathsf{q}$.

Case $\forall \alpha^{\mathsf{q}}.\, \sigma_1$.

$$\begin{aligned}
|((\forall \alpha^{\mathsf{q}}.\, \sigma_1)^{\mathscr{C}})^{\mathscr{A}}| &= |(\forall \beta.(\sigma_1[\{\beta\}/\alpha^{\mathsf{q}}])^{\mathscr{C}})^{\mathscr{A}}| \\
&= |\forall \gamma^{\mathsf{u}}.\, ((\sigma_1[\{\beta\}/\alpha^{\mathsf{q}}])^{\mathscr{C}}[\{\gamma^{\mathsf{u}}\}/\beta])^{\mathscr{A}}| \\
&= |\forall \gamma^{\mathsf{u}}.\, ((\sigma_1[\{\{\gamma^{\mathsf{u}}\}\}/\alpha^{\mathsf{q}}])^{\mathscr{C}})^{\mathscr{A}}| \\
&= |\forall \gamma^{\mathsf{u}}.\, ((\sigma_1[\gamma^{\mathsf{u}}/\alpha^{\mathsf{q}}])^{\mathscr{C}})^{\mathscr{A}}| \\
&= |\forall \alpha^{\mathsf{u}}.\, ((\sigma_1)^{\mathscr{C}})^{\mathscr{A}}| \\
&= |((\sigma_1)^{\mathscr{C}})^{\mathscr{A}}| \\
&\sqsubseteq |\sigma_1| && \text{by i.h.} \\
&= |\forall \alpha^{\mathsf{q}}.\, \sigma_1|
\end{aligned}$$

Case $\alpha^{\mathsf{q}}, \sigma_1\ \mathsf{ref}, \sigma_1 \otimes \sigma_2$.

Since σ is opaque, then $((\sigma)^{\mathscr{C}})^{\mathscr{A}} = \sigma$ by part (*ii*), so $|\sigma| \sqsubseteq |\sigma|$.

Case $\{\tau^{\circ}\}$.

$|((\{\tau^{\circ}\})^{\mathscr{C}})^{\mathscr{A}}| = |(\tau^{\circ})^{\mathscr{A}}| = |\{\tau^{\circ}\}| = \mathsf{u}$.

(*iv*) By cases on σ°:

Case α^{q}.

Then $(\alpha^{\mathsf{q}})^{\mathscr{C}} = \{\alpha^{\mathsf{q}}\}$. By inspection of the translation function, the only σ such that $(\sigma)^{\mathscr{C}} = \{\alpha^{\mathsf{q}}\}$ is α^{q}.

Case $\sigma'\ \mathsf{ref}$.

Then $(\sigma'\ \mathsf{ref})^{\mathscr{C}} = \{\sigma'\ \mathsf{ref}\}$. By inspection of the translation function, the only σ such that $(\sigma)^{\mathscr{C}} = \{\sigma'\ \mathsf{ref}\}$ is $\sigma'\ \mathsf{ref}$.

Case $\sigma_1 \otimes \sigma_2$.

Then $(\sigma_1 \otimes \sigma_2)^{\mathscr{C}} = \{\sigma_1 \otimes \sigma_2\}$. By inspection of the translation function, the only σ such that $(\sigma)^{\mathscr{C}} = \{\sigma_1 \otimes \sigma_2\}$ is $\sigma_1 \otimes \sigma_2$. □

Lemma 5.2.2 (Type translation preserves well-formedness)**.**

(*i*) *$\Delta \vdash_{\mathscr{C}} \tau$ if and only if $\Delta \vdash_{\mathscr{A}} (\tau)^{\mathscr{A}}$.*

(*ii*) *$\Delta \vdash_{\mathscr{A}} \sigma$ if and only if $\Delta \vdash_{\mathscr{C}} (\sigma)^{\mathscr{C}}$.*

Proof. By inspection of the type well-formedness rules, $\Delta \vdash_{\mathscr{C}} \tau$ if and only if $\mathrm{FTV}(\tau) \subseteq \Delta$. Likewise, $\Delta \vdash_{\mathscr{A}} \sigma$ if and only if $\mathrm{FTV}(\sigma) \subseteq \Delta$. Thus, it suffices to show that $\mathrm{FTV}(\tau) = \mathrm{FTV}((\tau)^{\mathscr{A}})$ and $\mathrm{FTV}(\sigma) = \mathrm{FTV}((\sigma)^{\mathscr{C}})$.

We use an alternative induction measure to map types into the naturals:

$$\begin{aligned} \mathcal{H}(\mathbf{int}) &= 0 \\ \mathcal{H}(\tau_1 \to \tau_2) &= \max(\mathcal{H}(\tau_1), \mathcal{H}(\tau_2)) + 1 \\ \mathcal{H}(\forall \beta.\tau') &= \mathcal{H}(\tau') + 1 \\ \mathcal{H}(\beta) &= 0 \\ \mathcal{H}(\{\sigma^{\mathbf{o}}\}) &= 2 \cdot \mathcal{H}(\sigma^{\mathbf{o}}) \end{aligned} \qquad \begin{aligned} \mathcal{H}(\mathsf{int}) &= 0 \\ \mathcal{H}(\sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2) &= \max(\mathcal{H}(\sigma_1), \mathcal{H}(\sigma_2)) + 1 \\ \mathcal{H}(\forall \beta^{\mathsf{q}}.\sigma') &= \mathcal{H}(\sigma') + 1 \\ \mathcal{H}(\beta^{\mathsf{q}}) &= 0 \\ \mathcal{H}(\{\tau^{\mathbf{o}}\}) &= 2 \cdot \mathcal{H}(\tau^{\mathbf{o}}) \\ \mathcal{H}(\sigma_1 \otimes \sigma_2) &= \max(\mathcal{H}(\sigma_1), \mathcal{H}(\sigma_2)) + 1 \\ \mathcal{H}(\sigma' \ \mathsf{ref}) &= \mathcal{H}(\sigma') + 1 \end{aligned}$$

We proceed by induction using $\mathcal{H}$.

(*i*) By cases on τ:

Case int.

Then $\mathrm{FTV}((\mathsf{int})^{\mathscr{C}}) = \mathrm{FTV}(\mathbf{int}) = \varnothing = \mathrm{FTV}(\mathsf{int})$.

Case $\tau_1 \to \tau_2$.

Then $\mathrm{FTV}((\tau_1 \overset{\mathsf{u}}{\multimap} \tau_2)^{\mathscr{C}}) = \mathrm{FTV}((\tau_1)^{\mathscr{C}} \to (\tau_2)^{\mathscr{C}}) = \mathrm{FTV}((\tau_1)^{\mathscr{C}}) \cup \mathrm{FTV}((\tau_2)^{\mathscr{C}}) = \mathrm{FTV}(\tau_1) \cup \mathrm{FTV}(\tau_2) = \mathrm{FTV}(\tau_1 \overset{\mathsf{u}}{\multimap} \tau_2)$.

Case $\forall \beta.\, \tau'$.

$$\begin{aligned} \mathrm{FTV}((\forall \beta.\, \tau')^{\mathscr{C}}) &= \mathrm{FTV}(\forall \alpha^{\mathbf{u}}.\, (\tau'[\{\alpha^{\mathbf{u}}\}/\beta])^{\mathscr{C}}) \\ &= \mathrm{FTV}((\tau'[\{\alpha^{\mathbf{u}}\}/\beta])^{\mathscr{C}}) - \{\alpha^{\mathbf{u}}\} \\ &= \mathrm{FTV}(\tau'[\{\alpha^{\mathbf{u}}\}/\beta]) - \{\alpha^{\mathbf{u}}\} && \text{induction hypothesis} \\ &= \mathrm{FTV}(\tau') - \{\beta\} \\ &= \mathrm{FTV}(\forall \beta.\, \tau'). \end{aligned}$$

Note that we can apply the induction hypothesis at $\tau'[\{\alpha^{\mathbf{u}}\}/\beta]$ because $\mathcal{H}(\{\alpha^{\mathbf{u}}\}) = 0 = \mathcal{H}(\beta)$, which means that $\mathcal{H}(\tau'[\{\alpha^{\mathbf{u}}\}/\beta]) = \mathcal{H}(\tau') < \mathcal{H}(\forall \beta.\, \tau')$.

Case β.

Then $\mathrm{FTV}((\beta)^{\mathscr{C}}) = \mathrm{FTV}(\{\beta\}) = \mathrm{FTV}(\beta)$.

Case $\{\sigma^{\mathbf{o}}\}$.

Then $\mathrm{FTV}((\{\sigma^{\mathbf{o}}\})^{\mathscr{C}}) = \mathrm{FTV}(\sigma^{\mathbf{o}}) = \mathrm{FTV}(\{\sigma^{\mathbf{o}}\})$.

(*ii*) By cases on σ:

Case int.

Then $\mathrm{FTV}((\mathsf{int})^{\mathscr{C}}) = \mathrm{FTV}(\mathbf{int}) = \varnothing = \mathrm{FTV}(\mathsf{int})$.

Case $\sigma_1 \xrightarrow{\mathsf{q}}\!\circ\, \sigma_2$.

Then $\mathrm{FTV}((\sigma_1 \xrightarrow{\mathsf{q}}\!\circ\, \sigma_2)^{\mathscr{C}}) = \mathrm{FTV}((\sigma_1)^{\mathscr{C}} \to (\sigma_2)^{\mathscr{C}}) = \mathrm{FTV}((\sigma_1)^{\mathscr{C}}) \cup \mathrm{FTV}((\sigma_2)^{\mathscr{C}}) = \mathrm{FTV}(\sigma_1) \cup \mathrm{FTV}(\sigma_2) = \mathrm{FTV}(\sigma_1 \xrightarrow{\mathsf{q}}\!\circ\, \sigma_2)$.

Case $\forall\beta^{\mathsf{q}}.\, \sigma'$.

$$\begin{aligned}
\mathrm{FTV}((\forall\beta^{\mathsf{q}}.\,\sigma')^{\mathscr{C}}) &= \mathrm{FTV}(\forall\alpha.\,(\sigma'[\{\alpha\}/\beta^{\mathsf{q}}])^{\mathscr{C}}) \\
&= \mathrm{FTV}((\sigma'[\{\alpha\}/\beta^{\mathsf{q}}])^{\mathscr{C}}) - \{\alpha\} \\
&= \mathrm{FTV}(\sigma'[\{\alpha\}/\beta^{\mathsf{q}}]) - \{\alpha\} && \text{induction hypothesis} \\
&= \mathrm{FTV}(\sigma') - \{\beta^{\mathsf{q}}\} \\
&= \mathrm{FTV}(\forall\beta^{\mathsf{q}}.\,\sigma').
\end{aligned}$$

Note that we can apply the induction hypothesis at $\sigma'[\{\alpha\}/\beta^{\mathsf{q}}]$ because $\mathcal{H}(\{\alpha\}) = 0 = \mathcal{H}(\beta^{\mathsf{q}})$, which means that $\mathcal{H}(\sigma'[\{\alpha\}/\beta^{\mathsf{q}}]) = \mathcal{H}(\sigma') < \mathcal{H}(\forall\beta^{\mathsf{q}}.\,\sigma')$.

Case β^{q}.

Then $\mathrm{FTV}((\beta^{\mathsf{q}})^{\mathscr{C}}) = \mathrm{FTV}(\{\beta^{\mathsf{q}}\}) = \mathrm{FTV}(\beta^{\mathsf{q}})$.

Case $\{\tau^{\mathbf{o}}\}$.

Then $\mathrm{FTV}((\{\tau^{\mathbf{o}}\})^{\mathscr{C}}) = \mathrm{FTV}(\tau^{\mathbf{o}}) = \mathrm{FTV}(\{\tau^{\mathbf{o}}\})$.

Case σ' ref.

Then $\mathrm{FTV}((\sigma'\ \mathsf{ref})^{\mathscr{C}}) = \mathrm{FTV}(\{\sigma'\ \mathsf{ref}\}) = \mathrm{FTV}(\sigma'\ \mathsf{ref})$.

Case $\sigma_1 \otimes \sigma_2$.

Then $\mathrm{FTV}((\sigma_1 \otimes \sigma_2)^{\mathscr{C}}) = \mathrm{FTV}(\{\sigma_1 \otimes \sigma_2\}) = \mathrm{FTV}(\sigma_1 \otimes \sigma_2)$. □

Definition 5.2.3 (Unlimited and affine restriction)**.** *We define the* ***unlimited restriction*** *of* Γ*, denoted* $\Gamma|_u$*, to be* Γ *restricted to the portion of its domain that it does not map to* affine σ *types. We define the* ***unlimited restriction*** *of* Σ*, denoted* $\Sigma|_u$*, to be* Σ *restricted to the portion of its domain that it doesn't take to* σ *types, affine or unlimited.*

That is,

$$\begin{aligned}
\cdot|_u &= \cdot \\
\Gamma, \mathsf{x}{:}\sigma|_u &= \begin{cases} \Gamma|_u, \mathsf{x}{:}\sigma & \text{if } |\sigma| = \mathsf{u} \\ \Gamma|_u & \text{if } |\sigma| = \mathsf{a} \end{cases} \\
\Sigma, \ell{:}\tau|_u &= \Sigma|_u, \ell{:}\tau \\
\Sigma, \ell{:}[\sigma]^{\ell'}|_u &= \Sigma|_u, \ell{:}[\sigma]^{\ell'} \\
\Sigma, \ell{:}\sigma|_u &= \Sigma|_u
\end{aligned}$$

Likewise, we define the ***affine restrictions*** $\Gamma|_a$ *and* $\Sigma|_a$ *to be the remaining portions of* Γ *and* Σ*, respectively. That is,* $\Gamma = \Gamma|_u, \Gamma|_a$ *and* $\Sigma = \Sigma|_u, \Sigma|_a$ *(up to exchange).*

If $\Sigma_1|_u = \Sigma_2|_u$*, we say that* $\Sigma_1 \sim_u \Sigma_2$*, and likewise for value contexts; clearly* $(\sim_u)$ *is an equivalence relation.*

Lemma 5.2.4 (Context splitting properties)**.**

Commutativity *If* $\Gamma_1 \boxplus \Gamma_2 = \Gamma$ *then* $\Gamma_2 \boxplus \Gamma_1 = \Gamma$. *If* $\Sigma_1 \boxplus \Sigma_2 = \Sigma$ *then* $\Sigma_2 \boxplus \Sigma_1 = \Sigma$.

Associativity $(\Gamma_1 \boxplus \Gamma_2) \boxplus \Gamma_3 = \Gamma$ *if and only if* $\Gamma_1 \boxplus (\Gamma_2 \boxplus \Gamma_3) = \Gamma$. $(\Sigma_1 \boxplus \Sigma_2) \boxplus \Sigma_3 = \Sigma$ *if and only if* $\Sigma_1 \boxplus (\Sigma_2 \boxplus \Sigma_3) = \Sigma$.

Absorbtion *If* $\Gamma_1 \boxplus \Gamma_2|_u = \Gamma$ *for any* Γ_1, Γ_2, *and* Γ, *then* $\Gamma_1 = \Gamma$ *and* $\Gamma_2|_u = \Gamma|_u$. *Likewise, if* $\Sigma_1 \boxplus \Sigma_2|_u = \Sigma$ *for any* Σ_1, Σ_2, *and* Σ, *then* $\Sigma_1 = \Sigma$ *and* $\Sigma_2|_u = \Sigma|_u$.

As a trivial corollary, for any Γ, $\Gamma \boxplus \Gamma|_u = \Gamma$, *and for any* Σ, $\Sigma \boxplus \Sigma|_u = \Sigma$.

Equivalence *For any* Γ_1 *and* Γ_2, *if there exists some* Γ *such that* $\Gamma_1 \boxplus \Gamma_2 = \Gamma$, *then* $\Gamma_1 \sim_u \Gamma_2$. *For any* Σ_1 *and* Σ_2, *if there exists some* Σ *such that* $\Sigma_1 \boxplus \Sigma_2 = \Sigma$, *then* $\Sigma_1 \sim_u \Sigma_2$.

Proof. Each case by a trivial structural induction. □

Lemma 5.2.5 (Protection is free)**.** *If a store has a type, then protecting or unprotecting any part of its type preserves the typing. In particular, for any* Σ_1, Σ_2, *and* ℓ,

$$\Sigma_1 \rhd^M s : \Sigma_2, \Sigma_3 \qquad \Longleftrightarrow \qquad \Sigma_1 \rhd^M s : \Sigma_2, [\Sigma_3]^\ell .$$

Proof. By induction on Σ_3 and inversion of the derivation of the antecedent:

Case $\cdot$.

That is, $\Sigma_1 \rhd^M s : \Sigma_2$. Then $[\cdot]^\ell = \cdot$.

Case $\Sigma_3', \ell'{:}\sigma$.

Then

$$\frac{\Sigma_{11} \rhd^M s' : \Sigma_2, \Sigma_3' \qquad \Sigma_{12}; \cdot \rhd^M_{\mathscr{A}} \mathsf{v} : \sigma}{\Sigma_{11} \boxplus \Sigma_{12} \rhd^M s' \uplus \{\ell' \mapsto \mathsf{v}\} : \Sigma_2, \Sigma_3', \ell'{:}\sigma}$$

$$\Leftrightarrow$$

$$\frac{\Sigma_{11} \rhd^M s' : \Sigma_2, [\Sigma_3']^\ell \qquad \Sigma_{12}; \cdot \rhd^M_{\mathscr{A}} \mathsf{v} : \sigma}{\Sigma_{11} \boxplus \Sigma_{12} \rhd^M s' \uplus \{\ell' \mapsto \mathsf{v}\} : \Sigma_2, [\Sigma_3']^\ell, \ell'{:}[\sigma]^\ell}$$

by induction at Σ_3', where $\Sigma_1 = \Sigma_{11} \boxplus \Sigma_{12}$ and $s = s' \uplus \{\ell' \mapsto \mathsf{v}\}$ and $\Sigma_2, [\Sigma_3']^\ell, \ell'{:}[\sigma]^\ell = \Sigma_2, [\Sigma_3', \ell'{:}\sigma]^\ell = \Sigma_2, [\Sigma_3]^\ell$.

Case $\Sigma_3', \ell' : [\sigma]^{\ell''}$.

Then

$$\frac{\Sigma_{11} \rhd^M s' : \Sigma_2, \Sigma_3' \qquad \Sigma_{12}; \cdot \rhd^M_{\mathscr{A}} \mathsf{v} : \sigma}{\Sigma_{11} \boxplus \Sigma_{12} \rhd^M s \uplus \{\ell' \mapsto \mathsf{v}\} : \Sigma_2, \Sigma_3', \ell' : [\sigma]^{\ell''}}$$

$$\Leftrightarrow$$

$$\frac{\Sigma_{11} \rhd^M s' : \Sigma_2, [\Sigma_3']^\ell \qquad \Sigma_{12}; \cdot \rhd^M_{\mathscr{A}} \mathsf{v} : \sigma}{\Sigma_{11} \boxplus \Sigma_{12} \rhd^M s \uplus \{\ell' \mapsto \mathsf{v}\} : \Sigma_2, [\Sigma_3']^\ell, \ell' : [\sigma]^{\ell''}}$$

by induction at Σ_3', where $\Sigma_1 = \Sigma_{11} \boxplus \Sigma_{12}$ and $s = s \uplus \{\ell' \mapsto \mathsf{v}\}$ and $\Sigma_2, [\Sigma_3']^\ell, \ell' : [\sigma]^{\ell''} = \Sigma_2, [\Sigma_3', \ell' : [\sigma]^{\ell''}]^\ell = \Sigma_2, [\Sigma_3]^\ell$.

Case $\Sigma_3', \ell' : \tau$.

Then

$$\frac{\Sigma_{11} \rhd^M s' : \Sigma_2, \Sigma_3' \qquad \Sigma_{12}; \cdot \rhd^M_{\mathscr{C}} \mathbf{v} : \tau}{\Sigma_{11} \boxplus \Sigma_{12} \rhd^M s \uplus \{\ell' \mapsto \mathbf{v}\} : \Sigma_2, \Sigma_3', \ell'{:}\tau} \Leftrightarrow \frac{\Sigma_{11} \rhd^M s' : \Sigma_2, [\Sigma_3']^\ell \qquad \Sigma_{12}; \cdot \rhd^M_{\mathscr{C}} \mathbf{v} : \tau}{\Sigma_{11} \boxplus \Sigma_{12} \rhd^M s \uplus \{\ell' \mapsto \mathbf{v}\} : \Sigma_2, [\Sigma_3']^\ell, \ell'{:}\tau}$$

by induction at Σ_3', where $\Sigma_1 = \Sigma_{11} \boxplus \Sigma_{12}$ and $s = s \uplus \{\ell' \mapsto \mathbf{v}\}$ and $\Sigma_2, [\Sigma_3']^\ell, \ell'{:}\tau = \Sigma_2, [\Sigma_3', \ell'{:}\tau]^\ell = \Sigma_2, [\Sigma_3]^\ell$. □

Lemma 5.2.6 (Contexts close typed terms). *The free variables, type variables, and locations in a well-typed term and type are contained in the contexts used to type it.*

(*i*) *If* $\Sigma; \Delta; \Gamma \rhd^M_{\mathscr{C}} \mathbf{e} : \tau$ *then* $\mathrm{FV}(\mathbf{e}) \subseteq \mathrm{dom}\,\Gamma$, $\mathrm{FTV}(\mathbf{e}) \subseteq \Delta$, $\mathrm{FL}(\mathbf{e}) \subseteq \mathrm{dom}\,\Sigma$, *and* $\mathrm{FTV}(\tau) \subseteq \mathrm{dom}\,\Gamma$.

(*ii*) *If* $\Sigma; \Delta; \Gamma \rhd^M_{\mathscr{A}} \mathsf{e} : \sigma$ *then* $\mathrm{FV}(\mathsf{e}) \subseteq \mathrm{dom}\,\Gamma$, $\mathrm{FTV}(\mathsf{e}) \subseteq \Delta$, $\mathrm{FL}(\mathsf{e}) \subseteq \mathrm{dom}\,\Sigma$, *and* $\mathrm{FTV}(\sigma) \subseteq \mathrm{dom}\,\Gamma$.

Proof. By induction on the type rules and the definition of free locations and variables. □

Lemma 5.2.7 (Store types are closed). *If* $\Sigma_1 \rhd^M s : \Sigma_2$ *then* $\mathrm{FTV}(\Sigma_2) = \varnothing$.

Proof. We proceed by induction on the derivation of $\Sigma_1 \rhd^M s : \Sigma_2$:

Case $\dfrac{}{\Sigma_1 \rhd^M \cdot : \cdot}$.

Then $\mathrm{FTV}(\cdot) = \varnothing$.

Case $\dfrac{\Sigma_{11} \rhd^M s : \Sigma_2' \qquad \Sigma_{12}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{v} : \tau}{\Sigma_{11} \boxplus \Sigma_{12} \rhd^M s \uplus \{\ell \mapsto \mathbf{v}\} : \Sigma_2', \ell{:}\tau}$.

By the induction hypothesis, $\mathrm{FTV}(\Sigma_2') = \varnothing$, and since $\mathbf{v}$ types in an empty type context, $\mathrm{FTV}(\tau) = \varnothing$; thus $\mathrm{FTV}(\Sigma_2', \ell{:}\tau) = \varnothing$ as well.

Case $\dfrac{\Sigma_{11} \rhd^M s : \Sigma_2' \qquad \Sigma_{12}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{v} : \sigma}{\Sigma_{11} \boxplus \Sigma_{12} \rhd^M s \uplus \{\ell \mapsto \mathsf{v}\} : \Sigma_2', \ell{:}\sigma}$.

By the induction hypothesis, $\mathrm{FTV}(\Sigma_2') = \varnothing$, and since v types in an empty type context, $\mathrm{FTV}(\sigma) = \varnothing$; thus $\mathrm{FTV}(\Sigma_2', \ell{:}\sigma) = \varnothing$ as well.

Case $\dfrac{\Sigma_{11} \rhd^M s : \Sigma_2' \qquad \Sigma_{12}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{v} : \sigma}{\Sigma_{11} \boxplus \Sigma_{12} \rhd^M s \uplus \{\ell \mapsto \mathsf{v}\} : \Sigma_2', \ell{:}[\sigma]^{\ell'}}$.

By the induction hypothesis, $\mathrm{FTV}(\Sigma_2') = \varnothing$, and since v types in an empty type context, $\mathrm{FTV}(\sigma) = \varnothing$; thus $\mathrm{FTV}(\Sigma_2', \ell{:}[\sigma]^{\ell'}) = \varnothing$ as well. □

5.3 External Typing Implies Internal Typing

Lemma 5.3.1 (Equivalence of expression typing). *If an expression types in the external type system* ($\vdash$), *then it types in the internal type system* ($\rhd$) *with empty store type:*

(*i*) *If* $\Delta;\Gamma \vdash^M_{\mathscr{C}} \mathbf{e} : \tau$ *then* $\cdot;\Delta;\Gamma \triangleright^M_{\mathscr{C}} \mathbf{e} : \tau$.

(*ii*) *If* $\Delta;\Gamma \vdash^M_{\mathscr{A}} \mathsf{e} : \sigma$ *then* $\cdot;\Delta;\Gamma \triangleright^M_{\mathscr{A}} \mathsf{e} : \sigma$.

Proof. By induction on the type derivation.

$$\text{Case}\ \frac{\dfrac{\mathcal{A}}{\Delta,\alpha;\Gamma \vdash^M_{\mathscr{C}} \mathbf{v} : \tau}}{\Delta;\Gamma \vdash^M_{\mathscr{C}} \mathbf{\Lambda}\alpha.\,\mathbf{v} : \forall\alpha.\,\tau}.$$

$$\frac{\dfrac{\text{i.h. at } \mathcal{A}}{\cdot;\Delta,\alpha;\Gamma \triangleright^M_{\mathscr{C}} \mathbf{v} : \tau}}{\cdot;\Delta;\Gamma \triangleright^M_{\mathscr{C}} \mathbf{\Lambda}\alpha.\,\mathbf{v} : \forall\alpha.\,\tau}$$

$$\text{Case}\ \frac{\dfrac{\mathcal{A}}{\Delta;\Gamma,\mathbf{x}{:}\tau \vdash^M_{\mathscr{C}} \mathbf{e} : \tau'} \qquad \dfrac{\mathcal{B}}{\Delta \vdash_{\mathscr{C}} \tau}}{\Delta;\Gamma \vdash^M_{\mathscr{C}} \lambda\mathbf{x}{:}\tau.\,\mathbf{e} : \tau \to \tau'}.$$

$$\frac{\dfrac{\text{i.h. at } \mathcal{A}}{\cdot;\Delta;\Gamma,\mathbf{x}{:}\tau \triangleright^M_{\mathscr{C}} \mathbf{e} : \tau'} \qquad \dfrac{\mathcal{B}}{\Delta \vdash_{\mathscr{C}} \tau} \qquad \dfrac{}{\left|\cdot\right|_{\mathrm{FL}(\lambda\mathbf{x}:\tau.\,\mathbf{e})} = \mathsf{u}}}{\cdot;\Delta;\Gamma \triangleright^M_{\mathscr{C}} \lambda\mathbf{x}{:}\tau.\,\mathbf{e} : \tau \to \tau'}$$

$$\text{Case}\ \frac{}{\Delta;\Gamma \vdash^M_{\mathscr{C}} \mathbf{c} : \mathrm{ty}_{\mathscr{C}}(\mathbf{c})}.$$

$$\frac{}{\cdot;\Delta;\Gamma \triangleright^M_{\mathscr{C}} \mathbf{c} : \mathrm{ty}_{\mathscr{C}}(\mathbf{c})}$$

$$\text{Case}\ \frac{\Gamma(\mathbf{x}) = \tau}{\Delta;\Gamma \vdash^M_{\mathscr{C}} \mathbf{x} : \tau}.$$

$$\frac{\Gamma(\mathbf{x}) = \tau}{\cdot;\Delta;\Gamma \triangleright^M_{\mathscr{C}} \mathbf{x} : \tau}$$

$$\text{Case}\ \frac{\dfrac{\mathcal{A}}{\mathbf{module}\ \mathbf{f} : \tau = \mathbf{v} \in M} \qquad \dfrac{\mathcal{B}}{\cdot \vdash_{\mathscr{C}} \tau}}{\cdot;\Delta;\Gamma \vdash^M_{\mathscr{C}} \mathbf{f} : \tau}.$$

$$\frac{\dfrac{\mathcal{A}}{\mathbf{module}\ \mathbf{f} : \tau = \mathbf{v} \in M} \qquad \dfrac{\mathcal{B}}{\cdot \vdash_{\mathscr{C}} \tau}}{\cdot;\Delta;\Gamma \triangleright^M_{\mathscr{C}} \mathbf{f} : \tau}$$

$$\text{Case}\ \frac{\dfrac{\mathcal{A}}{\Delta;\Gamma \vdash^M_{\mathscr{C}} \mathbf{e} : \forall\alpha.\,\tau} \qquad \dfrac{\mathcal{B}}{\Delta \vdash_{\mathscr{C}} \tau}}{\Delta;\Gamma \vdash^M_{\mathscr{C}} \mathbf{e}[\tau] : \tau'[\tau/\alpha]}.$$

$$\dfrac{\dfrac{\text{i.h. at }\mathcal{A}}{\cdot;\Delta;\Gamma \rhd^{M}_{\mathscr{C}} \mathbf{e} : \forall\alpha.\,\tau} \qquad \dfrac{\mathcal{B}}{\Delta \vdash_{\mathscr{C}} \tau}}{\cdot;\Delta;\Gamma \rhd^{M}_{\mathscr{C}} \mathbf{e}[\tau] : \tau'[\tau/\alpha]}$$

Case $$\dfrac{\dfrac{\mathcal{A}}{\Delta;\Gamma \vdash^{M}_{\mathscr{C}} \mathbf{e_1} : \tau' \to \tau} \qquad \dfrac{\mathcal{B}}{\Delta;\Gamma \vdash^{M}_{\mathscr{C}} \mathbf{e_2} : \tau'}}{\Delta;\Gamma \vdash^{M}_{\mathscr{C}} \mathbf{e_1}\ \mathbf{e_2} : \tau}.$$

$$\dfrac{\dfrac{\text{i.h. at }\mathcal{A}}{\cdot;\Delta;\Gamma \rhd^{M}_{\mathscr{C}} \mathbf{e_1} : \tau' \to \tau} \qquad \dfrac{\text{i.h. at }\mathcal{B}}{\cdot;\Delta;\Gamma \rhd^{M}_{\mathscr{C}} \mathbf{e_2} : \tau'}}{\cdot \boxplus \cdot;\Delta;\Gamma \rhd^{M}_{\mathscr{C}} \mathbf{e_1}\ \mathbf{e_2} : \tau}$$

Case $$\dfrac{\dfrac{\mathcal{A}}{\Delta;\Gamma \vdash^{M}_{\mathscr{C}} \mathbf{e_1} : \mathbf{int}} \qquad \dfrac{\mathcal{B}}{\Delta;\Gamma \vdash^{M}_{\mathscr{C}} \mathbf{e_2} : \tau} \qquad \dfrac{\mathcal{C}}{\Delta;\Gamma \vdash^{M}_{\mathscr{C}} \mathbf{e_3} : \tau}}{\Delta;\Gamma \vdash^{M}_{\mathscr{C}} \mathbf{if0}\,\mathbf{e_1}\ \mathbf{e_2}\ \mathbf{e_3} : \tau}.$$

$$\dfrac{\dfrac{\text{i.h. at }\mathcal{A}}{\cdot;\Delta;\Gamma \rhd^{M}_{\mathscr{C}} \mathbf{e_1} : \mathbf{int}} \qquad \dfrac{\text{i.h. at }\mathcal{B}}{\cdot;\Delta;\Gamma \rhd^{M}_{\mathscr{C}} \mathbf{e_2} : \tau} \qquad \dfrac{\text{i.h. at }\mathcal{C}}{\cdot;\Delta;\Gamma \rhd^{M}_{\mathscr{C}} \mathbf{e_3} : \tau}}{\cdot \boxplus \cdot;\Delta;\Gamma \rhd^{M}_{\mathscr{C}} \mathbf{if0}\,\mathbf{e_1}\ \mathbf{e_2}\ \mathbf{e_3} : \tau}$$

Case $$\dfrac{\dfrac{\mathcal{A}}{\Delta;\Gamma \vdash^{M}_{\mathscr{A}} \mathsf{e} : \sigma} \qquad \dfrac{\mathcal{B}}{\sigma <: \sigma'}}{\Delta;\Gamma \vdash^{M}_{\mathscr{A}} \mathsf{e} : \sigma'}.$$

$$\dfrac{\dfrac{\text{i.h. at }\mathcal{A}}{\cdot;\Delta;\Gamma \rhd^{M}_{\mathscr{A}} \mathsf{e} : \sigma} \qquad \dfrac{\mathcal{B}}{\sigma <: \sigma'}}{\Delta;\Gamma \rhd^{M}_{\mathscr{A}} \mathsf{e} : \sigma'}$$

Case $$\dfrac{\dfrac{\mathcal{A}}{\Delta,\alpha^{\mathsf{q}};\Gamma \vdash^{M}_{\mathscr{A}} \mathsf{v} : \sigma}}{\Delta;\Gamma \vdash^{M}_{\mathscr{A}} \Lambda\alpha^{\mathsf{q}}.\,\mathsf{v} : \forall\alpha^{\mathsf{q}}.\,\sigma}.$$

$$\dfrac{\dfrac{\text{i.h. at }\mathcal{A}}{\cdot;\Delta,\alpha^{\mathsf{q}};\Gamma \rhd^{M}_{\mathscr{A}} \mathsf{v} : \sigma}}{\Delta;\Gamma \rhd^{M}_{\mathscr{A}} \Lambda\alpha^{\mathsf{q}}.\,\mathsf{v} : \forall\alpha^{\mathsf{q}}.\,\sigma}$$

Case $$\dfrac{\dfrac{\mathcal{A}}{\Delta;\Gamma,\mathsf{x}{:}\sigma \vdash^{M}_{\mathscr{A}} \mathsf{e} : \sigma'} \qquad \dfrac{\mathcal{B}}{\Delta \vdash_{\mathscr{A}} \sigma} \qquad \dfrac{\mathcal{C}}{\left|\Gamma|_{\mathrm{FV}(\lambda \mathsf{x}:\sigma.\,\mathsf{e})}\right| = \mathsf{q}}}{\Delta;\Gamma \vdash^{M}_{\mathscr{A}} \lambda \mathsf{x}{:}\sigma.\,\mathsf{e} : \sigma \overset{\mathsf{q}}{\multimap} \sigma'}.$$

$$\dfrac{\dfrac{\text{i.h. at }\mathcal{A}}{\cdot;\Delta;\Gamma,\mathsf{x}{:}\sigma \rhd^{M}_{\mathscr{A}} \mathsf{e}:\sigma'} \qquad \dfrac{\mathcal{B}}{\Delta\vdash_{\mathscr{A}}\sigma} \qquad \dfrac{\overset{\mathcal{C}}{\dfrac{}{\left|\Gamma|_{\mathrm{FV}(\lambda \mathsf{x}:\sigma.\,\mathsf{e})}\right| = \mathsf{q}} \qquad \dfrac{}{\left|\cdot|_{\mathrm{FL}(\lambda \mathsf{x}:\sigma.\,\mathsf{e})}\right| = \mathsf{u}}}}{\left|\Gamma|_{\mathrm{FV}(\lambda \mathsf{x}:\sigma.\,\mathsf{e})}\right| \sqcup \left|\cdot|_{\mathrm{FL}(\lambda \mathsf{x}:\sigma.\,\mathsf{e})}\right| = \mathsf{q}}}{\Delta;\Gamma \rhd^{M}_{\mathscr{A}} \lambda \mathsf{x}{:}\sigma.\,\mathsf{e} : \sigma \overset{\mathsf{q}}{\multimap} \sigma'}$$

Case $\dfrac{}{\Delta;\Gamma \vdash^{M}_{\mathscr{A}} \mathsf{c} : \mathrm{ty}_{\mathscr{A}}(\mathsf{c})}$.

$$\dfrac{}{\cdot;\Delta;\Gamma \rhd^{M}_{\mathscr{A}} \mathsf{c} : \mathrm{ty}_{\mathscr{A}}(\mathsf{c})}$$

Case $\dfrac{\Gamma(\mathsf{x}) = \sigma}{\Delta;\Gamma \vdash^{M}_{\mathscr{A}} \mathsf{x} : \sigma}$.

$$\dfrac{\Gamma(\mathsf{x}) = \sigma}{\cdot;\Delta;\Gamma \rhd^{M}_{\mathscr{A}} \mathsf{x} : \sigma}$$

Case $\dfrac{\dfrac{\mathcal{A}}{\mathsf{module\,f} : \sigma = \mathsf{v} \in M} \qquad \dfrac{\mathcal{B}}{\cdot \vdash_{\mathscr{A}} \tau}}{\Delta;\Gamma \vdash^{M}_{\mathscr{A}} \mathsf{f} : \sigma}$.

$$\dfrac{\dfrac{\mathcal{A}}{\mathsf{module\,f} : \sigma = \mathsf{v} \in M} \qquad \dfrac{\mathcal{B}}{\cdot \vdash_{\mathscr{A}} \sigma}}{\cdot;\Delta;\Gamma \rhd^{M}_{\mathscr{A}} \mathsf{f} : \sigma}$$

Case $\dfrac{\dfrac{\mathcal{A}}{\Delta;\Gamma \vdash^{M}_{\mathscr{A}} \mathsf{e} : \forall\alpha^{\mathsf{q}}.\,\sigma} \qquad \dfrac{\mathcal{B}}{\Delta \vdash_{\mathscr{A}} \sigma} \qquad \dfrac{\mathcal{C}}{|\sigma| \sqsubseteq \mathsf{q}}}{\Delta;\Gamma \vdash^{M}_{\mathscr{A}} \mathsf{e}[\sigma] : \sigma'[\sigma/\alpha^{\mathsf{q}}]}$.

$$\dfrac{\dfrac{\text{i.h. at }\mathcal{A}}{\cdot;\Delta;\Gamma \rhd^{M}_{\mathscr{A}} \mathsf{e} : \forall\alpha^{\mathsf{q}}.\,\sigma} \qquad \dfrac{\mathcal{B}}{\Delta \vdash_{\mathscr{A}} \sigma} \qquad \dfrac{\mathcal{C}}{|\sigma| \sqsubseteq \mathsf{q}}}{\cdot;\Delta;\Gamma \rhd^{M}_{\mathscr{A}} \mathsf{e}[\sigma] : \sigma'[\sigma/\alpha^{\mathsf{q}}]}$$

Case $\dfrac{\dfrac{\mathcal{A}}{\Delta;\Gamma_1 \vdash^{M}_{\mathscr{A}} \mathsf{e}_1 : \sigma' \overset{\mathsf{q}}{\multimap} \sigma} \qquad \dfrac{\mathcal{B}}{\Delta;\Gamma_2 \vdash^{M}_{\mathscr{A}} \mathsf{e}_2 : \sigma'}}{\Delta;\Gamma_1 \boxplus \Gamma_2 \vdash^{M}_{\mathscr{A}} \mathsf{e}_1\ \mathsf{e}_2 : \sigma}$.

$$\dfrac{\dfrac{\text{i.h. at }\mathcal{A}}{\cdot;\Delta;\Gamma_1 \rhd^{M}_{\mathscr{A}} \mathsf{e}_1 : \sigma' \overset{\mathsf{q}}{\multimap} \sigma} \qquad \dfrac{\text{i.h. at }\mathcal{B}}{\cdot;\Delta;\Gamma_2 \rhd^{M}_{\mathscr{A}} \mathsf{e}_2 : \sigma'}}{\cdot \boxplus \cdot;\Delta;\Gamma_1 \boxplus \Gamma_2 \rhd^{M}_{\mathscr{A}} \mathsf{e}_1\ \mathsf{e}_2 : \sigma}$$

Case $\dfrac{\dfrac{\mathcal{A}}{\Delta;\Gamma_1 \vdash^{M}_{\mathscr{A}} \mathsf{e}_1 : \mathsf{int}} \qquad \dfrac{\mathcal{B}}{\Delta;\Gamma_2 \vdash^{M}_{\mathscr{A}} \mathsf{e}_2 : \tau} \qquad \dfrac{\mathcal{C}}{\Delta;\Gamma_2 \vdash^{M}_{\mathscr{A}} \mathsf{e}_3 : \tau}}{\Delta;\Gamma_1 \boxplus \Gamma_2 \vdash^{M}_{\mathscr{A}} \mathsf{if0}\,\mathsf{e}_1\ \mathsf{e}_2\ \mathsf{e}_3 : \tau}$.

$$\dfrac{\dfrac{\text{i.h. at } \mathcal{A}}{\cdot;\Delta;\Gamma_1 \triangleright^M_{\mathscr{A}} \mathsf{e}_1 : \mathsf{int}} \qquad \dfrac{\text{i.h. at } \mathcal{B}}{\cdot;\Delta;\Gamma_2 \triangleright^M_{\mathscr{A}} \mathsf{e}_2 : \tau} \qquad \dfrac{\text{i.h. at } \mathcal{C}}{\cdot;\Delta;\Gamma_2 \triangleright^M_{\mathscr{A}} \mathsf{e}_3 : \tau}}{\cdot \boxplus \cdot;\Delta;\Gamma_1 \boxplus \Gamma_2 \triangleright^M_{\mathscr{A}} \mathsf{if0}\ \mathsf{e}_1\ \mathsf{e}_2\ \mathsf{e}_3 : \tau}$$

Case $\dfrac{\dfrac{\mathcal{A}}{\Delta;\Gamma_1 \vdash^M_{\mathscr{A}} \mathsf{e}_1 : \sigma_1} \qquad \dfrac{\mathcal{B}}{\Delta;\Gamma_2 \vdash^M_{\mathscr{A}} \mathsf{e}_2 : \sigma_2}}{\Delta;\Gamma_1 \boxplus \Gamma_2 \vdash^M_{\mathscr{A}} \langle \mathsf{e}_1, \mathsf{e}_2 \rangle : \sigma_1 \otimes \sigma_2}$.

$$\dfrac{\dfrac{\text{i.h. at } \mathcal{A}}{\cdot;\Delta;\Gamma_1 \triangleright^M_{\mathscr{A}} \mathsf{e}_1 : \sigma_1} \qquad \dfrac{\text{i.h. at } \mathcal{B}}{\cdot;\Delta;\Gamma_2 \triangleright^M_{\mathscr{A}} \mathsf{e}_2 : \sigma_2}}{\cdot \boxplus \cdot;\Delta;\Gamma_1 \boxplus \Gamma_2 \triangleright^M_{\mathscr{A}} \langle \mathsf{e}_1, \mathsf{e}_2 \rangle : \sigma_1 \otimes \sigma_2}$$

Case $\dfrac{\dfrac{\mathcal{A}}{\Delta;\Gamma_1 \vdash^M_{\mathscr{A}} \mathsf{e}_1 : \sigma_\mathsf{x} \otimes \sigma_\mathsf{y}} \qquad \dfrac{\mathcal{B}}{\Delta;\Gamma_2, \mathsf{x}{:}\sigma_\mathsf{x}, y{:}\sigma_y \vdash^M_{\mathscr{A}} \mathsf{e}_2 : \sigma}}{\Delta;\Gamma_1 \boxplus \Gamma_2 \vdash^M_{\mathscr{A}} \mathsf{let}\ \langle \mathsf{x}, \mathsf{y} \rangle = \mathsf{e}_1\ \mathsf{in}\ \mathsf{e}_2 : \sigma}$.

$$\dfrac{\dfrac{\text{i.h. at } \mathcal{A}}{\cdot;\Delta;\Gamma_1 \triangleright^M_{\mathscr{A}} \mathsf{e}_1 : \sigma_\mathsf{x} \otimes \sigma_\mathsf{y}} \qquad \dfrac{\text{i.h. at } \mathcal{B}}{\cdot;\Delta;\Gamma_2, x{:}\sigma_x, y{:}\sigma_y \triangleright^M_{\mathscr{A}} \mathsf{e}_2 : \sigma}}{\cdot \boxplus \cdot;\Delta;\Gamma_1 \boxplus \Gamma_2 \triangleright^M_{\mathscr{A}} \mathsf{let}\ \langle \mathsf{x}, \mathsf{y} \rangle = \mathsf{e}_1\ \mathsf{in}\ \mathsf{e}_2 : \sigma}$$

Case $\dfrac{\dfrac{\mathcal{A}}{\mathsf{module}\ \mathsf{f} : \sigma = \mathsf{v} \in M} \qquad \dfrac{\mathcal{B}}{\cdot \vdash_{\mathscr{A}} \sigma}}{\Delta;\Gamma \vdash^M_{\mathscr{C}} \mathsf{f} : (\sigma)^{\mathscr{C}}}$.

$$\dfrac{\dfrac{\mathcal{A}}{\mathsf{module}\ \mathsf{f} : \sigma = \mathsf{v} \in M} \qquad \dfrac{\mathcal{B}}{\cdot \vdash_{\mathscr{A}} \sigma}}{\cdot;\Delta;\Gamma \triangleright^M_{\mathscr{C}} \mathsf{f} : (\sigma)^{\mathscr{C}}}$$

Case $\dfrac{\dfrac{\mathcal{A}}{\mathbf{module}\ \mathbf{f} : \tau = \mathbf{v} \in M} \qquad \dfrac{\mathcal{B}}{\cdot \vdash_{\mathscr{C}} \tau}}{\Delta;\Gamma \vdash^M_{\mathscr{A}} \mathbf{f} : (\tau)^{\mathscr{A}}}$.

$$\dfrac{\dfrac{\mathcal{A}}{\mathbf{module}\ \mathbf{f} : \tau = \mathbf{v} \in M} \qquad \dfrac{\mathcal{B}}{\cdot \vdash_{\mathscr{C}} \tau}}{\cdot;\Delta;\Gamma \triangleright^M_{\mathscr{A}} \mathbf{f} : (\tau)^{\mathscr{A}}}$$

Case $\dfrac{\dfrac{\mathcal{A}}{\mathbf{interface}\ \mathbf{f} :> \sigma = \mathbf{g} \in M} \qquad \dfrac{\mathcal{B}}{\cdot \vdash_{\mathscr{A}} \sigma}}{\Delta;\Gamma \vdash^M_{\mathscr{A}} \mathbf{f} : \sigma}$.

$$\dfrac{\dfrac{\mathcal{A}}{\mathbf{interface}\ \mathbf{f} :> \sigma = \mathbf{g} \in M} \qquad \dfrac{\mathcal{B}}{\cdot \vdash_{\mathscr{A}} \sigma}}{\cdot;\Delta;\Gamma \triangleright^M_{\mathscr{A}} \mathbf{f} : \sigma}$$

□

Theorem 5.3.2 (Programs to configurations)**.** *If* $\vdash M\ \mathbf{e} : \tau$ *then* $\triangleright^M (\cdot, \mathbf{e}) : \tau$.

Proof. By inversion of PROG, all modules in M are okay. Furthermore, $\cdot \vdash^M_{\mathscr{C}} \mathbf{e} : \tau$, and by Lemma 5.3.1, $\cdot; \cdot \triangleright^M_{\mathscr{C}} \mathbf{e} : \tau$. Since $s = \cdot$, $\Sigma = \cdot$, and thus, $\cdot \triangleright^M s : \cdot \boxplus \cdot$. Thus, by CONF, $\triangleright^M (\cdot, \mathbf{e}) : \tau$. □

5.4 Evaluation Contexts and Substitution

In this section, we prove several lemmas about terms in holes and about substitution. In Lemma 5.4.2, we show that if a well-typed term is decomposed into an evaluation context and a subterm in the hole, then the subterm types, and the evaluation context types with a suitable replacement term in the hole as well; unlike the usual replacement theorem, we require the replacement term to type in empty contexts. In the lemma after that (Lemma 5.4.3), we show that the hole may be re-filled with any a term that types in a non-empty store context. Breaking this into two lemmas this allows us to manipulate the store context in which the evaluation context is typed separately before replacing the term in the hole.

We begin, however, with an observation about how we may often ignore subsumption rule (RTA-SUBSUME), which is not syntax directed, when dealing with type derivations.

Observation 5.4.1 (Subsumption and proof by inversion)**.** We first observe that multiple adjacent applications of the type rule RTA-SUBSUME may always be condensed into one, by the transitivity of ($<:$). By induction, any instance of multiple adjacent subsumptions may be rewritten to have only one subsumption. Furthermore, any derivation in $\lambda^{\mathscr{A}}$ that does *not* end with a subsumption may have a subsumption added at the root, by reflexivity of the subtype relation. Thus, without loss of generality, we may consider any type derivation in $\lambda^{\mathscr{A}}$ to end with rule A-SUBSUME, with a *different* rule preceding it in the derivation.

Now we consider inverting type judgments of the form $\Sigma; \Gamma \triangleright^M_{\mathscr{A}} \mathsf{e} : \sigma$. The subsumption rule may always appear at the root, and in general only one or two other rules will match the syntax of e. Denote the applicable syntax-specific rule for e as rule R. Because we do not consider proofs with multiple adjacent subsumptions, the premiss to RTA-SUBSUME must be the conclusion of a different rule. But because e is the same, only rule R applies!:

$$\dfrac{\dfrac{A_1 \ \cdots \ A_k}{\Sigma; \Delta; \Gamma \triangleright^M_{\mathscr{A}} \mathsf{e} : \sigma_<}\ \text{R} \qquad \sigma_< <: \sigma}{\Sigma; \Delta; \Gamma \triangleright^M_{\mathscr{A}} \mathsf{e} : \sigma}\ \text{RTA-SUBSUME}$$

Thus, when inverting a type judgment for the $\lambda^{\mathscr{A}}$ subcalculus, we may safely consider inverting the syntax-specific judgment for e at an arbitrary type $\sigma_< <: \sigma$. If our goal is reconstruct a new type judgment giving σ, by subsumption it is sufficient to reconstruct a type judgment giving $\sigma_<$.

Lemma 5.4.2 (Terms in holes are typeable)**.**

(*i*) *If* $\Sigma; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{E}[\mathbf{e}']_{\mathscr{C}} : \tau$, *then there exist some* $\Sigma_1 \boxplus \Sigma_2 = \Sigma$ *and* τ' *such that* $\Sigma_1; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{e}' : \tau'$, *and for any other* $\mathbf{e}''$ *such that* $\cdot; \cdot; \cdot \triangleright^M_{\mathscr{C}} \mathbf{e}'' : \tau'$, *it types with* $\Sigma_2; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{E}[\mathbf{e}'']_{\mathscr{C}} : \tau$

(*ii*) *If* $\Sigma; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{E}[\mathsf{e}']_{\mathscr{A}} : \tau$, *then there exist some* $\Sigma_1 \boxplus \Sigma_2 = \Sigma$ *and* σ' *such that* $\Sigma_1; \Delta; \Gamma \triangleright^M_{\mathscr{A}} \mathsf{e}' : \sigma'$, *and for any other* e'' *such that* $\cdot; \cdot; \cdot \triangleright^M_{\mathscr{A}} \mathsf{e}'' : \sigma'$, *it types with* $\Sigma_2; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{E}[\mathsf{e}'']_{\mathscr{A}} : \tau$

(*iii*) *If* $\Sigma; \Delta; \Gamma \triangleright^M_{\mathscr{A}} \mathsf{E}[\mathsf{e}']_{\mathscr{A}} : \sigma$, *then there exist some* $\Sigma_1 \boxplus \Sigma_2 = \Sigma$, $\Gamma_1 \boxplus \Gamma_2 = \Gamma$, *and* τ' *such that* $\Sigma_1; \Delta; \Gamma_1 \triangleright^M_{\mathscr{A}} \mathsf{e}' : \sigma'$, *and for any other* e'' *such that* $\cdot; \cdot; \cdot \triangleright^M_{\mathscr{A}} \mathsf{e}'' : \sigma'$, *it types with* $\Sigma_2; \Delta; \Gamma \triangleright^M_{\mathscr{A}} \mathsf{E}[\mathsf{e}'']_{\mathscr{A}} : \sigma$

(*iv*) *If* $\Sigma'; \Delta; \Gamma \triangleright^M_{\mathscr{A}} \mathsf{E}[\mathbf{e}']_{\mathscr{C}} : \sigma$, *then there exist some* $\Sigma_1 \boxplus \Sigma_2 = \Sigma$, $\Gamma_1 \boxplus \Gamma_2 = \Gamma$, *and* τ' *such that* $\Sigma_1; \Delta; \Gamma_1 \triangleright^M_{\mathscr{C}} \mathbf{e}' : \tau'$, *and for any other* $\mathbf{e}''$ *such that* $\cdot; \cdot; \cdot \triangleright^M_{\mathscr{C}} \mathbf{e}'' : \tau'$, *it types with* $\Sigma_2; \Delta; \Gamma \triangleright^M_{\mathscr{A}} \mathsf{E}[\mathbf{e}'']_{\mathscr{C}} : \sigma$

In particular, if $\mathsf{E}[\mathsf{e}']_{\mathscr{A}}$ *is closed, then so is* e' *(and likewise for the other three cases).*

Proof. We take the statement of the theorem as an induction hypothesis in four parts and proceed by mutual induction on the structures of $\mathbf{E}$ and E.

(*i*) Consider first $\mathbf{E}$:

Case $[\,]_{\mathscr{C}}$.

Then $\mathbf{E}[\mathbf{e}']_{\mathscr{C}} = \mathbf{e}'$.

Let $\tau' = \tau$, $\Sigma_1 = \Sigma$ and $\Sigma_2 = \Sigma|_u$.

Note that $\mathbf{E}[\mathbf{e}'']_{\mathscr{C}} = \mathbf{e}''$.

If $\cdot; \cdot; \cdot \triangleright^M_{\mathscr{C}} \mathbf{e}'' : \tau'$, then by weakening, $\Sigma|_u; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{e}'' : \tau'$.

Case $\mathbf{E}'[\tau_{\mathbf{a}}]$.

This only types if $\Sigma; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{E}'[\mathbf{e}']_{\mathscr{C}} : \forall\alpha.\, \tau_{\mathbf{b}}$ where $\tau = \tau_{\mathbf{b}}[\tau_{\mathbf{a}}/\alpha]$.

By induction, there exist some τ' and $\Sigma_1 \boxplus \Sigma_2 = \Sigma$ such that $\Sigma_1; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{e}' : \tau'$ and $\Sigma_2; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{E}'[\mathbf{e}'']_{\mathscr{C}} : \forall\alpha.\, \tau_{\mathbf{b}}$ for suitable $\mathbf{e}''$.

By RTC-TApp, $\Sigma_2; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{E}'[\mathbf{e}'']_{\mathscr{C}}[\tau_{\mathbf{a}}] : \tau$.

Case $\mathbf{E}'\ \mathbf{e_2}$.

This only types if $\Sigma_1; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{E}'[\mathbf{e}']_{\mathscr{C}} : \tau_{\mathbf{1}} \to \tau$ and $\Sigma_2; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{e_2} : \tau_{\mathbf{1}}$ where $\Sigma_1 \boxplus \Sigma_2 = \Sigma$.

By induction, there exist some τ' and $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$ such that $\Sigma_{11}; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{e}' : \tau'$ and $\Sigma_{12}; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{E}'[\mathbf{e}'']_{\mathscr{C}} : \tau_{\mathbf{1}} \to \tau$ for suitable $\mathbf{e}''$.

By RTC-App, $\Sigma_{12} \boxplus \Sigma_2; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{E}'[\mathbf{e}'']_{\mathscr{C}}\ \mathbf{e_2} : \tau$.

Case $\mathbf{v}\ \mathbf{E}'$.

This only types if $\Sigma_1; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{v} : \tau_{\mathbf{1}} \to \tau$ and $\Sigma_2; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{E}'[\mathbf{e}']_{\mathscr{C}} : \tau_{\mathbf{1}}$ where $\Sigma_1 \boxplus \Sigma_2 = \Sigma$.

By induction, there exist some τ' and $\Sigma_{21} \boxplus \Sigma_{22} = \Sigma_2$ such that $\Sigma_{21}; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{e}' : \tau'$ and $\Sigma_{22}; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{E}'[\mathbf{e}'']_{\mathscr{C}} : \tau_{\mathbf{1}}$ for suitable $\mathbf{e}''$.

By RTC-App, $\Sigma_1 \boxplus \Sigma_{22}; \Delta; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{v}\ \mathbf{E}'[\mathbf{e}'']_{\mathscr{C}} : \tau$.

Case $\mathbf{if0\,E'\ e_2\ e_3}$.

This only types if $\Sigma_1; \Delta; \Gamma \rhd^M_{\mathscr{C}} \mathbf{E'[e']_{\mathscr{C}}} : \mathbf{int}$, $\Sigma_2; \Delta; \Gamma \rhd^M_{\mathscr{C}} \mathbf{e_2} : \tau$, and $\Sigma_2; \Delta; \Gamma \rhd^M_{\mathscr{C}} \mathbf{e_2} : \tau$ where $\Sigma_1 \boxplus \Sigma_2 = \Sigma$.

By induction, there exists some τ' and $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$ such that $\Sigma_{11}; \Delta; \Gamma \rhd^M_{\mathscr{C}} \mathbf{e'} : \tau'$ and $\Sigma_{22}; \Delta; \Gamma \rhd^M_{\mathscr{C}} \mathbf{E'[e'']_{\mathscr{C}}} : \mathbf{int}$ for suitable $\mathbf{e''}$.

By RTC-IF0, $\Sigma_{12} \boxplus \Sigma_2; \Delta; \Gamma \rhd^M_{\mathscr{C}} \mathbf{if0\,E'[e'']_{\mathscr{C}}\ e_2\ e_3} : \tau$.

Case ${}_{\mathsf{f}}\mathbf{CA}^{\sigma}_{\mathsf{g}}(\mathsf{E'})$.

This only types if $(\sigma)^{\mathscr{C}} = \tau$ and if $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E'}[\mathbf{e'}]_{\mathscr{C}} : \sigma$.

By part (iv) of the induction hypothesis, there exist some τ' and $\Sigma_1 \boxplus \Sigma_2 = \Sigma$ such that $\Sigma_1; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{e'} : \tau'$ and $\Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E'}[\mathbf{e''}]_{\mathscr{C}} : \sigma$ for suitable $\mathbf{e''}$.

By RTC-BOUNDARY, $\Sigma_2; \Delta; \Gamma \rhd^M_{\mathscr{C}} {}_{\mathsf{f}}\mathbf{CA}^{\sigma}_{\mathsf{g}}(\mathsf{E'}[\mathbf{e''}]_{\mathscr{C}}) : \tau$.

This concludes the proof of first part.

(ii) The second part proceeds *mutatis mutandis*, with two notable changes:

- The $\mathsf{E'} = [\,]_{\mathscr{C}}$ case is vacuous.
- The $\mathbf{CA}$ boundary cases appeal to part (iii) of the induction hypothesis.

(iii) For the third part, we consider cases on E.

Case $[\,]_{\mathscr{A}}$.

Then $\mathsf{E}[\mathsf{e'}]_{\mathscr{A}} = \mathsf{e'}$.

Let $\sigma' = \sigma$, $\Sigma_1 = \Sigma$, $\Sigma_2 = \Sigma|_u$, $\Gamma_1 = \Gamma$, and $\Gamma_2 = \Gamma|_u$.

Note that $\mathsf{E}[\mathsf{e''}]_{\mathscr{A}} = \mathsf{e''}$.

If $\cdot; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{e''} : \sigma$, then by weakening $\Sigma|_u; \Delta; \Gamma|_u \rhd^M_{\mathscr{A}} \mathsf{e''} : \sigma'$.

Case $\mathsf{E'}[\sigma_{\mathsf{a}}]$.

Consider the type derivation of $\mathsf{E'}[\mathsf{e'}]_{\mathscr{A}}[\sigma_{\mathsf{a}}]$. According to Observation 5.4.1, without loss of generality, there exists some $\sigma_< <: \sigma$, with rule RTA-TAPP concluding that $\mathsf{E'}[\mathsf{e'}]_{\mathscr{A}}[\sigma_{\mathsf{a}}]$ has that type, followed by a subsumption. This can be the case only if $\Sigma; \Delta; \Gamma \rhd^M_{\mathscr{A}} \mathsf{E'}[\mathsf{e'}]_{\mathscr{A}} : \forall \alpha^{\mathsf{q}}.\, \sigma_{\mathsf{b}}$ where $\sigma_< = \sigma_{\mathsf{b}}[\sigma_{\mathsf{a}}/\alpha]$ and $|\sigma_{\mathsf{a}}| \sqsubseteq \mathsf{q}$.

By induction, there exist some σ', $\Sigma_1 \boxplus \Sigma_2 = \Sigma$, and $\Gamma_1 \boxplus \Gamma_2 = \Gamma$ such that $\Sigma_1; \Delta; \Gamma_1 \rhd^M_{\mathscr{A}} \mathsf{e'} : \sigma'$ and $\Sigma_2; \Delta; \Gamma_2 \rhd^M_{\mathscr{A}} \mathsf{E'}[\mathsf{e''}]_{\mathscr{A}} : \forall \alpha^{\mathsf{q}}.\, \sigma_{\mathsf{b}}$ for suitable $\mathsf{e''}$.

By RTA-TAPP and RTA-SUBSUME, $\Sigma_2; \Delta; \Gamma_2 \rhd^M_{\mathscr{A}} \mathsf{E'}[\mathsf{e''}]_{\mathscr{A}}[\sigma_{\mathsf{a}}] : \sigma$.

Case $\mathsf{E}\ \mathsf{e}_2$.

Consider the type derivation of $\mathsf{E'}[\mathsf{e'}]_{\mathscr{A}}\ \mathsf{e}_2$. According to Observation 5.4.1, without loss of generality, there exists some $\sigma_< <: \sigma$, with rule RTA-APP concluding that $\mathsf{E'}[\mathsf{e'}]_{\mathscr{A}}\ \mathsf{e}_2$ has that type, followed by a subsumption. This can be the case only if $\Sigma_1; \Delta; \Gamma_1 \rhd^M_{\mathscr{A}} \mathsf{E'}[\mathsf{e'}]_{\mathscr{A}} : \sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_<$ and $\Sigma_2; \Delta; \Gamma_2 \rhd^M_{\mathscr{A}} \mathsf{e}_2 : \sigma_1$ for some q, σ_1, $\Sigma_1 \boxplus \Sigma_2 = \Sigma$, and $\Gamma_1 \boxplus \Gamma_2 = \Gamma$.

By induction, there exist some σ', $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$, and $\Gamma_{11} \boxplus \Gamma_{12} = \Gamma_1$ such that $\Sigma_{11}; \Delta; \Gamma_{11} \triangleright^M_{\mathscr{A}} e' : \sigma'$ and $\Sigma_{12}; \Delta; \Gamma_{12} \triangleright^M_{\mathscr{A}} E'[e'']_{\mathscr{A}} : \sigma_1 \overset{q}{\multimap} \sigma_<$ for suitable e''.

By RTA-App and RTA-Subsume, $\Sigma_{12} \boxplus \Sigma_2; \Delta; \Gamma_{12} \boxplus \Gamma_2 \triangleright^M_{\mathscr{A}} E'[e'']_{\mathscr{A}}\ e_2 : \sigma$.

For subsequent cases, we consider subsumption implicitly.

Case $v\ E'$.

This only types if $\Sigma_1; \Delta; \Gamma_1 \triangleright^M_{\mathscr{A}} v : \sigma_1 \overset{q}{\multimap} \sigma_<$ and $\Sigma_2; \Delta; \Gamma_2 \triangleright^M_{\mathscr{A}} E'[e']_{\mathscr{A}} : \sigma_1$ where $\Sigma_1 \boxplus \Sigma_2 = \Sigma$ and $\Gamma_1 \boxplus \Gamma_2 = \Gamma$.

By induction, there exist some σ', $\Sigma_{21} \boxplus \Sigma_{22} = \Sigma_2$, and $\Gamma_{21} \boxplus \Gamma_{22} = \Gamma_2$ such that $\Sigma_{21}; \Delta; \Gamma_{21} \triangleright^M_{\mathscr{A}} e' : \sigma'$ and $\Sigma_{22}; \Delta; \Gamma_{22} \triangleright^M_{\mathscr{A}} E'[e'']_{\mathscr{A}} : \sigma_1$ for suitable e''.

By RTA-App, $\Sigma_1 \boxplus \Sigma_{22}; \Delta; \Gamma_1 \boxplus \Gamma_{22} \triangleright^M_{\mathscr{A}} v\ E'[e'']_{\mathscr{A}} : \sigma_<$.

Case $\mathsf{if0}\ E'\ e_2\ e_3$.

This only types if $\Sigma_1; \Delta; \Gamma_1 \triangleright^M_{\mathscr{A}} E'[e']_{\mathscr{A}} : \mathsf{int}$, $\Sigma_2; \Delta; \Gamma_2 \triangleright^M_{\mathscr{A}} e_2 : \sigma_<$, and $\Sigma_2; \Delta; \Gamma_2 \triangleright^M_{\mathscr{A}} e_3 : \sigma_<$ where $\Sigma_1 \boxplus \Sigma_2 = \Sigma$ and $\Gamma_1 \boxplus \Gamma_2 = \Gamma$.

By induction, there exist some σ', $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$, and $\Gamma_{11} \boxplus \Gamma_{12} = \Gamma_1$ such that $\Sigma_{11}; \Delta; \Gamma_{11} \triangleright^M_{\mathscr{A}} e' : \sigma'$ and $\Sigma_{12}; \Delta; \Gamma_{12} \triangleright^M_{\mathscr{A}} E'[e'']_{\mathscr{A}} : \mathsf{int}$ for suitable e''.

By RTA-If0, $\Sigma_{12} \boxplus \Sigma_2; \Delta; \Gamma_{12} \boxplus \Gamma_2 \triangleright^M_{\mathscr{A}} \mathsf{if0}\ E'[e'']_{\mathscr{A}}\ e_2\ e_3 : \sigma_<$.

Case $\langle E, e_2 \rangle$.

This only types if $\Sigma_1; \Delta; \Gamma_1 \triangleright^M_{\mathscr{A}} E'[e']_{\mathscr{A}} : \sigma_1$ and $\Sigma_2; \Delta; \Gamma_2 \triangleright^M_{\mathscr{A}} e_2 : \sigma_2$ for some σ_1, σ_2, Σ_1, Σ_2, Γ_1, and Γ_2 such that $\sigma_< = \sigma_1 \otimes \sigma_2$, $\Sigma_1 \boxplus \Sigma_2 = \Sigma$ and $\Gamma_1 \boxplus \Gamma_2 = \Gamma$.

By induction, there exist some σ', $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$, and $\Gamma_{11} \boxplus \Gamma_{12} = \Gamma_1$ such that $\Sigma_{11}; \Delta; \Gamma_{11} \triangleright^M_{\mathscr{A}} e' : \sigma'$ and $\Sigma_{12}; \Delta; \Gamma_{12} \triangleright^M_{\mathscr{A}} E'[e'']_{\mathscr{A}} : \sigma_1$ for suitable e''.

By RTA-Pair, $\Sigma_{12} \boxplus \Sigma_2; \Delta; \Gamma_{12} \boxplus \Gamma_2 \triangleright^M_{\mathscr{A}} \langle E'[e'']_{\mathscr{A}}, e_2 \rangle : \sigma_<$.

Case $\langle v, E' \rangle$.

This only types if $\Sigma_1; \Delta; \Gamma_1 \triangleright^M_{\mathscr{A}} v : \sigma_1$ and $\Sigma_2; \Delta; \Gamma_2 \triangleright^M_{\mathscr{A}} E'[e']_{\mathscr{A}} : \sigma_2$ for some σ_1, σ_2, Σ_1, Σ_2, Γ_1, and Γ_2 such that $\sigma_< = \sigma_1 \otimes \sigma_2$, $\Sigma_1 \boxplus \Sigma_2 = \Sigma$ and $\Gamma_1 \boxplus \Gamma_2 = \Gamma$.

By induction, there exist some σ', $\Sigma_{21} \boxplus \Sigma_{22} = \Sigma_2$, and $\Gamma_{21} \boxplus \Gamma_{22} = \Gamma_2$ such that $\Sigma_{21}; \Delta; \Gamma_{21} \triangleright^M_{\mathscr{A}} e' : \sigma'$ and $\Sigma_{22}; \Delta; \Gamma_{22} \triangleright^M_{\mathscr{A}} E'[e'']_{\mathscr{A}} : \sigma_2$ for suitable e''.

By RTA-Pair, $\Sigma_1 \boxplus \Sigma_{22}; \Delta; \Gamma_1 \boxplus \Gamma_{22} \triangleright^M_{\mathscr{A}} \langle v, E'[e'']_{\mathscr{A}} \rangle : \sigma_<$.

Case $\mathsf{let}\ \langle y_1, y_2 \rangle = E'\ \mathsf{in}\ e_2$.

This only types if $\Sigma_1; \Delta; \Gamma_1 \triangleright^M_{\mathscr{A}} E'[e']_{\mathscr{A}} : \sigma_1 \otimes \sigma_2$ and $\Sigma_2; \Delta; \Gamma_2, y_1 : \sigma_1, y_2 : \sigma_2 \triangleright^M_{\mathscr{A}} e_2 : \sigma_<$ for some σ_1, σ_2, Σ_1, Σ_2, Γ_1, and Γ_2 such that $\Sigma_1 \boxplus \Sigma_2 = \Sigma$ and $\Gamma_1 \boxplus \Gamma_2 = \Gamma$.

By induction, there exist some σ', $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$, and $\Gamma_{11} \boxplus \Gamma_{12} = \Gamma_1$ such that $\Sigma_{11}; \Delta; \Gamma_{11} \triangleright^M_{\mathscr{A}} e' : \sigma'$ and $\Sigma_{12}; \Delta; \Gamma_{12} \triangleright^M_{\mathscr{A}} E'[e'']_{\mathscr{A}} : \sigma_1 \otimes \sigma_2$ for suitable e''.

By RTA-Let, $\Sigma_{11} \boxplus \Sigma_2; \Delta; \Gamma_{11} \boxplus \Gamma_2 \triangleright^M_{\mathscr{A}} \mathsf{let}\ \langle y_1, y_2 \rangle = E'[e'']_{\mathscr{A}}\ \mathsf{in}\ e_2 : \sigma_<$.

Case ${}^{\sigma_<}_{\mathsf{f}}\mathsf{AC}_{\mathbf{g}}(\mathbf{E}')$.

This only types if $\Sigma; \cdot; \cdot \triangleright^M_{\mathscr{C}} \mathbf{E}'[e']_{\mathscr{A}} : (\sigma_<)^{\mathscr{C}}$.

By part (*iv*) of the induction hypothesis, there exist some σ', Σ_1, and Σ_2 such that $\Sigma_1; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{e}' : \sigma'$ and $\Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}'[\mathsf{e}'']_{\mathscr{A}} : (\sigma_<)^{\mathscr{C}}$ for suitable e''.

By RTA-Boundary, $\Sigma_2; \Delta; \Gamma_2 \rhd^M_{\mathscr{A}} {}^{\sigma_<}_{\mathsf{f}}\mathsf{AC}_{\mathbf{g}}(\mathbf{E}'[\mathsf{e}'']_{\mathscr{A}}) : \sigma_<$.

(*iv*) The proof of the fourth part follows the proof of the third, again *mutatis mutandis*, where again the hole case is vacuous and the AC boundary case appeals to part (*i*). □

Lemma 5.4.3 (Terms in holes are replaceable)**.**

(*i*) *If* $\Sigma_1; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}[\mathbf{e}'']_{\mathscr{C}} : \tau$ *and* $\cdot; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{e}'' : \tau'$ *and* $\Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{e}' : \tau'$ *where* $\Sigma_1 \boxplus \Sigma_2 = \Sigma$, *then* $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}[\mathbf{e}']_{\mathscr{C}} : \tau$.

(*ii*) *If* $\Sigma_1; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}[\mathsf{e}'']_{\mathscr{A}} : \tau$ *and* $\cdot; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{e}'' : \sigma'$ *and* $\Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{e}' : \sigma'$ *where* $\Sigma_1 \boxplus \Sigma_2 = \Sigma$, *then* $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}[\mathsf{e}']_{\mathscr{A}} : \tau$.

(*iii*) *If* $\Sigma_1; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}[\mathsf{e}'']_{\mathscr{A}} : \sigma$ *and* $\cdot; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{e}'' : \sigma'$ *and* $\Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{e}' : \sigma'$ *where* $\Sigma_1 \boxplus \Sigma_2 = \Sigma$, *then* $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}[\mathsf{e}']_{\mathscr{A}} : \sigma$.

(*iv*) *If* $\Sigma_1; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}[\mathbf{e}'']_{\mathscr{C}} : \sigma$ *and* $\cdot; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{e}'' : \tau'$ *and* $\Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{e}' : \tau'$ *where* $\Sigma_1 \boxplus \Sigma_2 = \Sigma$, *then* $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}[\mathbf{e}']_{\mathscr{C}} : \sigma$.

Proof. We take the statement of the theorem as an induction hypothesis in four parts and proceed by mutual induction on the structures of $\mathbf{E}$ and E.

(*i*) Consider first $\mathbf{E}$:

Case $[\,]_{\mathscr{C}}$.

Then $\mathbf{E}[\mathbf{e}'']_{\mathscr{C}} = \mathbf{e}''$, so we know that $\tau' = \tau$.

Then $\Sigma; \Delta; \Gamma \rhd^M_{\mathscr{C}} \mathbf{e}' : \tau$ by weakening.

Case $\mathbf{E}'[\tau_{\mathbf{a}}]$.

This can be the case only if $\Sigma_1; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}'[\mathbf{e}'']_{\mathscr{C}} : \forall\alpha.\, \tau_{\mathbf{b}}$ where $\tau = \tau_{\mathbf{b}}[\tau_{\mathbf{a}}/\alpha]$.

Then by induction, $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}'[\mathbf{e}']_{\mathscr{C}} : \forall\alpha.\, \tau_{\mathbf{b}}$.

By RTC-TApp, $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}'[\mathbf{e}']_{\mathscr{C}}[\tau_{\mathbf{a}}] : \tau$.

Case $\mathbf{E}'\ \mathbf{e_2}$.

This can be the case only if $\Sigma_{11}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}'[\mathbf{e}'']_{\mathscr{C}} : \tau_{\mathbf{1}} \to \tau$ and $\Sigma_{12}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{e_2} : \tau_{\mathbf{1}}$ for some $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$.

Then by induction, $\Sigma_{11} \boxplus \Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}'[\mathbf{e}']_{\mathscr{C}} : \tau_{\mathbf{1}} \to \tau$.

By RTC-App, $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}'[\mathbf{e}']_{\mathscr{C}}\ \mathbf{e_2} : \tau$.

Case $\mathbf{v}\ \mathbf{E}'$.

This can be the case only if $\Sigma_{11}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{v} : \tau_{\mathbf{1}} \to \tau$ and $\Sigma_{12}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}'[\mathbf{e}'']_{\mathscr{C}} : \tau_{\mathbf{1}}$ for some $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$.

Then by induction, $\Sigma_{11} \boxplus \Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}'[\mathbf{e}']_{\mathscr{C}} : \tau_{\mathbf{1}}$.

By RTC-App, $\Sigma_1 \boxplus \Sigma_{22}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{v}\ \mathbf{E}'[\mathbf{x}]_{\mathscr{C}} : \tau$.

Case $\mathbf{if0}\,\mathbf{E}'\ \mathbf{e_2}\ \mathbf{e_3}$.

This can be the case only if $\Sigma_{11}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}'[\mathbf{e}'']_{\mathscr{C}} : \mathbf{int}$, $\Sigma_{12}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{e_2} : \tau$, and $\Sigma_{12}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{e_2} : \tau$ for some $\Sigma_1 \boxplus \Sigma_2 = \Sigma$.

Then by induction, $\Sigma_{11} \boxplus \Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}'[\mathbf{e}']_{\mathscr{C}} : \mathbf{int}$.

By RTC-If0, $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{if0}\,\mathbf{E}'[\mathbf{e}']_{\mathscr{C}}\ \mathbf{e_2}\ \mathbf{e_3} : \tau$.

Case ${}_{\mathbf{f}}\mathbf{CA}^{\sigma}_{\mathsf{g}}(\mathsf{E}')$.

This can be the case only if $(\sigma)^{\mathscr{C}} = \tau$ and if $\Sigma_1; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}'[\mathbf{e}'']_{\mathscr{C}} : \sigma$.

Then by part (iv) of the induction hypothesis, $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}'[\mathbf{e}']_{\mathscr{C}} : \sigma$.

By RTC-Boundary, $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{C}} {}_{\mathbf{f}}\mathbf{CA}^{\sigma}_{\mathsf{g}}(\mathsf{E}'[\mathbf{e}']_{\mathscr{C}}) : \tau$.

This concludes the proof of first part.

(ii) The second part proceeds *mutatis mutandis*, with two notable changes:

- The $\mathsf{E}' = [\,]_{\mathscr{C}}$ case is vacuous.
- The **CA** boundary cases appeal to part (iii) of the induction hypothesis.

(iii) For the third part, we consider cases on E.

Case $[\,]_{\mathscr{A}}$.

Then $\mathsf{E}[\mathsf{x}]_{\mathscr{A}} = \mathsf{x}$, so we know that $\sigma' = \sigma$.

Then $\Sigma; \cdot \rhd^M_{\mathscr{A}} \mathsf{e}' : \sigma$ by weakening.

Case $\mathsf{E}'[\sigma_{\mathsf{a}}]$.

Consider the type derivation of $\mathsf{E}'[\mathsf{e}'']_{\mathscr{A}}[\sigma_{\mathsf{a}}]$. According to Observation 5.4.1, without loss of generality, there exists some $\sigma_< <: \sigma$, with rule RTA-TApp concluding that $\mathsf{E}'[\mathsf{e}'']_{\mathscr{A}}[\sigma_{\mathsf{a}}]$ has that type, followed by a subsumption. This can be the case only if $\Sigma_1; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}'[\mathsf{e}'']_{\mathscr{A}} : \forall \alpha^{\mathsf{q}}.\, \sigma_{\mathsf{b}}$ where $\sigma_< = \sigma_{\mathsf{b}}[\sigma_{\mathsf{a}}/\alpha]$ and $|\sigma_{\mathsf{a}}| \sqsubseteq \mathsf{q}$.

Then by induction, $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}'[\mathsf{e}']_{\mathscr{A}} : \forall \alpha^{\mathsf{q}}.\, \sigma_{\mathsf{b}}$.

By RTA-TApp and RTA-Subsume, $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}'[\mathsf{e}']_{\mathscr{A}}[\sigma_{\mathsf{a}}] : \sigma$.

For subsequent cases, we consider subsumption implicitly.

Case $\mathsf{E}\ \mathsf{e}_2$.

This can be the case only if $\Sigma_{11}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}'[\mathsf{e}'']_{\mathscr{A}} : \sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_<$ and $\Sigma_{12}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{e}_2 : \sigma_1$ for some q, σ_1 and $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$.

Then by induction, $\Sigma_{11} \boxplus \Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}'[\mathsf{e}']_{\mathscr{A}} : \sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_<$.

By RTA-App, $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}'[\mathsf{e}']_{\mathscr{A}}\ \mathsf{e}_2 : \sigma_<$.

Case $\mathsf{v}\ \mathsf{E}'$.

This can be the case only if $\Sigma_{11}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{v} : \sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_<$ and $\Sigma_{12}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}'[\mathsf{e}'']_{\mathscr{A}} : \sigma_1$ for some $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$.

Then by induction, $\Sigma_{12} \boxplus \Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}'[\mathsf{e}']_{\mathscr{A}} : \sigma_1$.

By RTA-App, $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{v}\ \mathsf{E}'[\mathsf{e}']_{\mathscr{A}} : \sigma_<$.

Case $\mathsf{if0}\ \mathsf{E}'\ \mathsf{e}_2\ \mathsf{e}_3$.

This can be the case only if $\Sigma_{11}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}'[\mathsf{e}'']_{\mathscr{A}} : \mathsf{int}$, $\Sigma_{12}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{e}_2 : \sigma_<$, and $\Sigma_{12}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{e}_3 : \sigma_<$ for some $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$.

Then by induction, $\Sigma_{11} \boxplus \Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}'[\mathsf{e}']_{\mathscr{A}} : \mathsf{int}$.

By RTA-IF0, $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{if0}\ \mathsf{E}'[\mathsf{e}']_{\mathscr{A}}\ \mathsf{e}_2\ \mathsf{e}_3 : \sigma_<$.

Case $\langle \mathsf{E}, \mathsf{e}_2 \rangle$.

This can be the case only if $\Sigma_{11}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}'[\mathsf{e}'']_{\mathscr{A}} : \sigma_1$ and $\Sigma_{12}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{e}_2 : \sigma_2$ for some σ_1, σ_2 and $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$ such that $\sigma_< = \sigma_1 \otimes \sigma_2$.

Then by induction, $\Sigma_{11} \boxplus \Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}'[\mathsf{e}']_{\mathscr{A}} : \sigma_1$.

By RTA-PAIR, $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{A}} \langle \mathsf{E}'[\mathsf{e}']_{\mathscr{A}}, \mathsf{e}_2 \rangle : \sigma_<$.

Case $\langle \mathsf{v}, \mathsf{E}' \rangle$.

This can be the case only if $\Sigma_{11}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{v} : \sigma_1$ and $\Sigma_{12}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}'[\mathsf{e}'']_{\mathscr{A}} : \sigma_2$ for some σ_1, σ_2 and $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$ such that $\sigma_< = \sigma_1 \otimes \sigma_2$.

Then by induction, $\Sigma_{12} \boxplus \Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}'[\mathsf{e}']_{\mathscr{A}} : \sigma_2$.

By RTA-PAIR, $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{A}} \langle \mathsf{v}, \mathsf{E}'[\mathsf{e}']_{\mathscr{A}} \rangle : \sigma_<$.

Case $\mathsf{let}\ \langle \mathsf{y}_1, \mathsf{y}_2 \rangle = \mathsf{E}'\ \mathsf{in}\ \mathsf{e}_2$.

This can be the case only if $\Sigma_{11}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}'[\mathsf{e}'']_{\mathscr{A}} : \sigma_1 \otimes \sigma_2$ and $\Sigma_{12}; \cdot; \cdot, \mathsf{y}_1{:}\sigma_1, \mathsf{y}_2{:}\sigma_2 \rhd^M_{\mathscr{A}} \mathsf{e}_2 : \sigma_<$ for some σ_1, σ_2 and $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$.

Then by induction, $\Sigma_{11} \boxplus \Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{E}'[\mathsf{e}']_{\mathscr{A}} : \sigma_1 \otimes \sigma_2$.

By RTA-LET, $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{let}\ \langle \mathsf{y}_1, \mathsf{y}_2 \rangle = \mathsf{E}'[\mathsf{e}']_{\mathscr{A}}\ \mathsf{in}\ \mathsf{e}_2 : \sigma_<$.

Case ${}^{\sigma_<}_{\mathsf{f}}\mathsf{AC}_{\mathbf{g}}(\mathbf{E}')$.

This can be the case only if $\Sigma_1; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}'[\mathsf{e}'']_{\mathscr{A}} : (\sigma_<)^{\mathscr{C}}$.

Then by part (*iv*) of the induction hypothesis, $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}'[\mathsf{e}']_{\mathscr{A}} : (\sigma_<)^{\mathscr{C}}$.

By RTA-BOUNDARY, $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{A}} {}^{\sigma_<}_{\mathsf{f}}\mathsf{AC}_{\mathbf{g}}(\mathbf{E}'[\mathsf{e}']_{\mathscr{A}}) : \sigma_<$.

(*iv*) The proof of the fourth part follows the proof of the third, again *mutatis mutandis*, where again the hole case is vacuous and the AC boundary case appeals to part (*i*). □

The next several lemmas concern substitution of types on types, types on value contexts, types on expressions, and values on expressions.

Lemma 5.4.4 (Type substitution on types preserves well-formedness and qualifiers)**.**

(*i*) *If* $\Delta, \alpha \vdash_{\mathscr{C}} \tau$ *and* $\Delta \vdash_{\mathscr{C}} \tau'$, *then* $\Delta \vdash_{\mathscr{C}} \tau[\tau'/\alpha]$.

(*ii*) *If* $\Delta, \alpha^{\mathsf{q}} \vdash_{\mathscr{C}} \tau$ *and* $\Delta \vdash_{\mathscr{A}} \sigma'$ *then* $\Delta \vdash_{\mathscr{C}} \tau[\sigma'/\alpha^{\mathsf{q}}]$*; if* $|\sigma'| \sqsubseteq \mathsf{q}$ *then* $|(\tau[\sigma'/\alpha^{\mathsf{q}}])^{\mathscr{A}}| \sqsubseteq |(\tau)^{\mathscr{A}}|$.

(*iii*) *If* $\Delta, \alpha^{\mathsf{q}} \vdash_{\mathscr{A}} \sigma$ *and* $\Delta \vdash_{\mathscr{A}} \sigma'$ *then* $\Delta \vdash_{\mathscr{A}} \sigma[\sigma'/\alpha^{\mathsf{q}}]$*; if* $|\sigma'| \sqsubseteq \mathsf{q}$ *then* $|\sigma[\sigma'/\alpha^{\mathsf{q}}]| \sqsubseteq |\sigma|$.

(*iv*) *If* $\Delta, \alpha \vdash_{\mathscr{A}} \sigma$ *and* $\Delta \vdash_{\mathscr{C}} \tau'$ *then* $\Delta \vdash_{\mathscr{A}} \sigma[\tau'/\alpha]$.

Proof. By mutual induction on the structure of τ and σ:

(*i*) By cases on τ:

Case $\mathbf{int}$.

Then $\tau[\tau'/\alpha] = \tau = \mathbf{int}$, so $\Delta \vdash_{\mathscr{C}} \mathbf{int}$.

Case $\tau_1 \rightarrow \tau_2$.

Then $\tau[\tau'/\alpha] = \tau_1[\tau'/\alpha] \rightarrow \tau_2[\tau'/\alpha]$.

By inversion, $\Delta \vdash_{\mathscr{C}} \tau_1$ and $\Delta \vdash_{\mathscr{C}} \tau_2$.

By the induction hypothesis (twice), $\Delta \vdash_{\mathscr{C}} \tau_1[\tau'/\alpha]$ and $\Delta \vdash_{\mathscr{C}} \tau_2[\tau'/\alpha]$.

Thus, $\Delta \vdash_{\mathscr{C}} (\tau_1 \rightarrow \tau_2)[\tau'/\alpha]$.

Case $\forall\beta.\,\tau_1$.

By inversion, $\Delta, \alpha, \beta \vdash_{\mathscr{C}} \tau_1$, and by well-formedness, $\alpha \neq \beta$.

Then $\tau[\tau'/\alpha] = \forall\beta.(\tau_1[\tau'/\alpha])$.

By the induction hypothesis and exchange, $\Delta, \beta \vdash_{\mathscr{C}} \tau_1[\tau'/\alpha]$.

Thus, $\Delta \vdash_{\mathscr{C}} (\forall\beta.\,\tau_1)[\tau'/\alpha]$.

Case β.

If $\alpha = \beta$, then $\tau[\tau'/\alpha] = \tau'$. Thus, $\Delta \vdash_{\mathscr{C}} \beta[\tau'/\alpha]$.

If $\alpha \neq \beta$, then $\tau[\tau'/\alpha] = \tau$. By inversion, $\beta \in \Delta$. Thus, $\Delta \vdash_{\mathscr{C}} \beta[\tau'/\alpha]$.

Case $\{\sigma^{\circ}\}$.

Then $\tau[\tau'/\alpha] = (\sigma^{\circ}[\tau'/\alpha])^{\mathscr{C}}$.

By inversion, $\Delta, \alpha \vdash_{\mathscr{A}} \sigma^{\circ}$.

By part (*iv*) of the induction hypothesis, $\Delta \vdash_{\mathscr{A}} \sigma^{\circ}[\tau'/\alpha]$, and by Lemma 5.2.2, $\Delta \vdash_{\mathscr{C}} (\sigma^{\circ}[\tau'/\alpha])^{\mathscr{C}}$.

(*ii*) By cases on τ:

Case $\mathbf{int}$.

Then $\tau[\sigma'/\alpha^{\mathsf{q}}] = \tau = \mathbf{int}$, so $\Delta \vdash_{\mathscr{C}} \mathbf{int}$, with $|(\tau[\sigma'/\alpha^{\mathsf{q}}])^{\mathscr{A}}| = \mathsf{u} = |(\tau)^{\mathscr{A}}|$.

Case $\tau_1 \rightarrow \tau_2$.

Then $\tau[\sigma'/\alpha^{\mathsf{q}}] = \tau_1[\sigma'/\alpha^{\mathsf{q}}] \overset{\mathsf{u}}{\multimap} \tau_2[\sigma'/\alpha^{\mathsf{q}}]$.

By inversion, $\Delta \vdash_{\mathscr{C}} \tau_1$ and $\Delta \vdash_{\mathscr{C}} \tau_2$.

By the induction hypothesis (twice), $\Delta \vdash_{\mathscr{C}} \tau_1[\sigma'/\alpha^{\mathsf{q}}]$ and $\Delta \vdash_{\mathscr{C}} \tau_2[\sigma'/\alpha^{\mathsf{q}}]$.

Thus, $\Delta \vdash_{\mathscr{C}} (\tau_1 \rightarrow \tau_2)[\sigma'/\alpha^{\mathsf{q}}]$, with $|((\tau_1 \rightarrow \tau_2)[\sigma'/\alpha^{\mathsf{q}}])^{\mathscr{A}}| = \mathsf{u} = |(\tau_1 \rightarrow \tau_2)^{\mathscr{A}}|$.

Case $\forall\beta.\,\tau_1$.

By inversion, $\Delta, \alpha, \beta \vdash_{\mathscr{C}} \tau_1$, and by well-formedness, $\alpha \neq \beta$.

Then $\tau[\sigma'/\alpha^{\mathsf{q}}] = \forall\beta.(\tau_1[\sigma'/\alpha^{\mathsf{q}}])$.

By the induction hypothesis and exchange, $\Delta, \beta \vdash_{\mathscr{C}} \tau_1[\sigma'/\alpha^{\mathsf{q}}]$.

Thus, $\Delta \vdash_{\mathscr{C}} (\forall\beta.\,\tau_1)[\sigma'/\alpha^{\mathsf{q}}]$, with $|(\forall\beta.\,\tau_1[\sigma'/\alpha^{\mathsf{q}}])^{\mathscr{A}}| = \mathsf{u} = |(\forall\beta.\,\tau_1)^{\mathscr{A}}|$.

Case β.

Since $\alpha^{\mathsf{q}} \neq \beta$, we know that $\tau[\sigma'/\alpha^{\mathsf{q}}] = \tau$.

By inversion, $\beta \in \Delta$.

Thus, $\Delta \vdash_{\mathscr{C}} \beta[\sigma'/\alpha^{\mathsf{q}}]$, with $|(\beta[\sigma'/\alpha^{\mathsf{q}}])^{\mathscr{A}}| = \mathsf{u} = |(\beta)^{\mathscr{A}}|$.

Case $\{\sigma^{\mathsf{o}}\}$.

Then $\tau[\sigma'/\alpha^{\mathsf{q}}] = (\sigma^{\mathsf{o}}[\sigma'/\alpha^{\mathsf{q}}])^{\mathscr{C}}$.

By inversion, $\Delta, \alpha \vdash_{\mathscr{A}} \sigma^{\mathsf{o}}$.

By part (iii) of the induction hypothesis, and because $|\sigma'| \sqsubseteq \mathsf{q}$, $\Delta \vdash_{\mathscr{A}} \sigma^{\mathsf{o}}[\sigma'/\alpha^{\mathsf{q}}]$, so by Lemma 5.2.2, $\Delta \vdash_{\mathscr{C}} (\sigma^{\mathsf{o}}[\sigma'/\alpha^{\mathsf{q}}])^{\mathscr{C}}$.

If $|\sigma'| \sqsubseteq \mathsf{q}$, then by the induction hypothesis, $|\sigma^{\mathsf{o}}[\sigma'/\alpha^{\mathsf{q}}]| \sqsubseteq |\sigma^{\mathsf{o}}|$.

By Lemma 5.2.1, we conclude that $|((\sigma^{\mathsf{o}}[\sigma'/\alpha^{\mathsf{q}}])^{\mathscr{C}})^{\mathscr{A}}| \sqsubseteq |\sigma^{\mathsf{o}}| = |(\{\sigma^{\mathsf{o}}\})^{\mathscr{A}}|$.

(iii) By cases on σ:

Case int.

Then $\sigma[\sigma'/\alpha^{\mathsf{q}}] = \sigma = \mathsf{int}$, so $\Delta \vdash_{\mathscr{A}} \mathsf{int}$ and $|\sigma[\sigma'/\alpha^{\mathsf{q}}]| = |\mathsf{int}| = \mathsf{u}$.

Case $\sigma_1 \overset{\mathsf{q}'}{\multimap} \sigma_2$.

Then $\sigma[\sigma'/\alpha^{\mathsf{q}}] = \sigma_1[\sigma'/\alpha^{\mathsf{q}}] \overset{\mathsf{q}'}{\multimap} \sigma_2[\sigma'/\alpha^{\mathsf{q}}]$.

By inversion, $\Delta \vdash_{\mathscr{A}} \sigma_1$ and $\Delta \vdash_{\mathscr{A}} \sigma_2$.

By the induction hypothesis (twice), $\Delta \vdash_{\mathscr{A}} \sigma_1[\sigma'/\alpha^{\mathsf{q}}]$ and $\Delta \vdash_{\mathscr{A}} \sigma_2[\sigma'/\alpha^{\mathsf{q}}]$.

Thus, $\Delta \vdash_{\mathscr{A}} (\sigma_1 \overset{\mathsf{q}'}{\multimap} \sigma_2)[\sigma'/\alpha^{\mathsf{q}}]$, which has qualifier $\mathsf{q}' = |\sigma|$.

Case $\forall\beta^{\mathsf{q}'}.\,\sigma_1$.

By inversion, $\Delta, \alpha^{\mathsf{q}}, \beta^{\mathsf{q}'} \vdash_{\mathscr{A}} \sigma_1$, and by well-formedness, $\alpha^{\mathsf{q}} \neq \beta^{\mathsf{q}'}$.

Then $\sigma[\sigma'/\alpha^{\mathsf{q}}] = \forall\beta^{\mathsf{q}'}.\,(\sigma_1[\sigma'/\alpha^{\mathsf{q}}])$.

By the induction hypothesis and exchange, $\Delta, \beta^{\mathsf{q}'} \vdash_{\mathscr{A}} \sigma_1[\sigma'/\alpha^{\mathsf{q}}]$.

Thus, $\Delta \vdash_{\mathscr{A}} (\forall\beta^{\mathsf{q}'}.\,\sigma_1)[\sigma'/\alpha^{\mathsf{q}}]$.

Note that $|\sigma[\sigma'/\alpha^{\mathsf{q}}]| = |\sigma_1[\sigma'/\alpha^{\mathsf{q}}]| \sqsubseteq |\sigma_1| = |\sigma|$.

Case $\beta^{\mathsf{q}'}$.

If $\alpha^{\mathsf{q}} = \beta^{\mathsf{q}'}$, then $\sigma[\sigma'/\alpha^{\mathsf{q}}] = \sigma'$ and $\mathsf{q} = \mathsf{q}'$. Thus, $\Delta \vdash_{\mathscr{A}} \beta^{\mathsf{q}'}[\sigma'/\alpha^{\mathsf{q}}]$, which has qualifier $|\sigma'| \sqsubseteq \mathsf{q} = \mathsf{q}' = |\sigma|$.

If $\alpha^{\mathsf{q}} \neq \beta^{\mathsf{q}'}$, then $\sigma[\sigma'/\alpha^{\mathsf{q}}] = \sigma$. By inversion, $\beta^{\mathsf{q}'} \in \Delta$. Thus, $\Delta \vdash_{\mathscr{A}} \beta^{\mathsf{q}'}[\sigma'/\alpha^{\mathsf{q}}]$, which has qualifier $\mathsf{q}' \sqsubseteq |\sigma|$.

Case $\{\tau^{\mathsf{o}}\}$.

Then $\sigma[\sigma'/\alpha^{\mathsf{q}}] = (\tau^{\mathsf{o}}[\sigma'/\alpha^{\mathsf{q}}])^{\mathscr{A}}$.

By inversion, $\Delta, \alpha^{\mathsf{q}} \vdash_{\mathscr{C}} \tau^{\mathsf{o}}$.

By part (ii) of the induction hypothesis, $\Delta \vdash_{\mathscr{C}} \tau^{\mathsf{o}}[\sigma'/\alpha^{\mathsf{q}}]$, and by Lemma 5.2.2, $\Delta \vdash_{\mathscr{A}} (\tau^{\mathsf{o}}[\sigma'/\alpha^{\mathsf{q}}])^{\mathscr{A}}$.

If $|\sigma'| \sqsubseteq \mathsf{q}$, then by the induction hypothesis, $|(\tau^{\mathbf{o}}[\sigma'/\alpha^{\mathsf{q}}])^{\mathscr{A}}| \sqsubseteq |(\tau^{\mathbf{o}})^{\mathscr{A}}| = |\{\tau^{\mathbf{o}}\}|$.

Case σ_1 ref.

Then $\sigma[\sigma'/\alpha^{\mathsf{q}}] = \sigma_1[\sigma'/\alpha^{\mathsf{q}}]$ ref.

By inversion, $\Delta, \alpha^{\mathsf{q}} \vdash_{\mathscr{A}} \sigma_1$.

By the induction hypothesis, $\Delta \vdash_{\mathscr{A}} \sigma_1[\sigma'/\alpha^{\mathsf{q}}]$, and thus $\Delta \vdash_{\mathscr{A}} \sigma_1[\sigma'/\alpha^{\mathsf{q}}]$ ref, with $|\sigma_1[\sigma'/\alpha^{\mathsf{q}}] \text{ ref}| = \mathsf{a} = |\sigma_1 \text{ ref}|$.

Case $\sigma_1 \otimes \sigma_2$.

Then $\sigma[\sigma'/\alpha^{\mathsf{q}}] = \sigma_1[\sigma'/\alpha^{\mathsf{q}}] \otimes \sigma_2[\sigma'/\alpha^{\mathsf{q}}]$.

By inversion, $\Delta, \alpha^{\mathsf{q}} \vdash_{\mathscr{A}} \sigma_1$ and $\Delta, \alpha^{\mathsf{q}} \vdash_{\mathscr{A}} \sigma_2$.

By the induction hypothesis, $\Delta \vdash_{\mathscr{A}} \sigma_1[\sigma'/\alpha^{\mathsf{q}}]$ and $\Delta \vdash_{\mathscr{A}} \sigma_2[\sigma'/\alpha^{\mathsf{q}}]$, and thus $\Delta \vdash_{\mathscr{A}} \sigma_1[\sigma'/\alpha^{\mathsf{q}}] \otimes \sigma_2[\sigma'/\alpha^{\mathsf{q}}]$.

If $|\sigma'| \sqsubseteq \mathsf{q}$, then by the induction hypothesis, $|\sigma_1[\sigma'/\alpha^{\mathsf{q}}]| \sqsubseteq |\sigma_1|$ and $|\sigma_2[\sigma'/\alpha^{\mathsf{q}}]| \sqsubseteq |\sigma_2|$; thus $|\sigma_1[\sigma'/\alpha^{\mathsf{q}}] \otimes \sigma_2[\sigma'/\alpha^{\mathsf{q}}]| = |\sigma_1[\sigma'/\alpha^{\mathsf{q}}]| \sqcup |\sigma_2[\sigma'/\alpha^{\mathsf{q}}]| \sqsubseteq |\sigma_1| \sqcup |\sigma_2| = |\sigma_1 \otimes \sigma_2|$.

(*iv*) *Mutatis mutandem*, with two notable changes:

Case $\beta^{\mathsf{q}'}$.

The $\alpha^{\mathsf{q}} = \beta^{\mathsf{q}'}$ case is vacuous.

Case $\{\tau^{\mathbf{o}}\}$.

By part (*i*) of the induction hypothesis. □

Corollary 5.4.5 (Type substitution preserves value context qualifiers)**.**
If $|\sigma| \sqsubseteq \mathsf{q}$ *then* $\left|\Gamma[\sigma/\alpha^{\mathsf{q}}]\right| \sqsubseteq \left|\Gamma\right|$.

Proof. By structural induction on Γ with Lemma 5.4.4. □

Lemma 5.4.6 (Type substitution on expressions preserves types)**.**

For all Σ *such that* $\mathrm{FTV}(\Sigma) = \varnothing$,

(*i*) *if* $\Sigma; \Delta, \alpha; \Gamma \rhd^{M}_{\mathscr{C}} \mathbf{e} : \tau$ *and* $\cdot \vdash_{\mathscr{C}} \tau_{\mathbf{a}}$, *where* $\alpha \notin \mathrm{FTV}(\Gamma)$,
then $\Sigma; \Delta; \Gamma[\tau_{\mathbf{a}}/\alpha] \rhd^{M}_{\mathscr{C}} \mathbf{e}[\tau_{\mathbf{a}}/\alpha] : \tau[\tau_{\mathbf{a}}/\alpha]$.

(*ii*) *if* $\Sigma; \Delta, \alpha^{\mathsf{q_a}}; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{e} : \sigma$ *and* $\cdot \vdash_{\mathscr{A}} \sigma_{\mathsf{a}}$, *where* $|\sigma_{\mathsf{a}}| \sqsubseteq \mathsf{q_a}$ *and* $\alpha^{\mathsf{q}_\alpha} \notin \mathrm{FTV}(\Gamma)$,
then $\Sigma; \Delta; \Gamma[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] \rhd^{M}_{\mathscr{A}} \mathsf{e}[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] : \sigma[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}]$.

Proof. Note first that $\mathrm{FV}(\mathbf{e}) = \mathrm{FV}(\mathbf{e}[\tau_{\mathbf{a}}/\alpha])$ and $\mathrm{FL}(\mathbf{e}) = \mathrm{FL}(\mathbf{e}[\tau_{\mathbf{a}}/\alpha])$, and likewise that $\mathrm{FV}(\mathsf{e}) = \mathrm{FV}(\mathsf{e}[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}])$ and $\mathrm{FL}(\mathsf{e}) = \mathrm{FL}(\mathsf{e}[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}])$.

(*i*) By induction on the structure of $\mathbf{e}$:

Case $\mathbf{\Lambda}\beta.\,\mathbf{v}'$.

By RTC-TLam, $\tau = \forall\beta.\,\tau'$, so it must be the case that

$$\dfrac{\dfrac{\mathcal{A}}{\Sigma; \Delta, \alpha, \beta; \Gamma \rhd^{M}_{\mathscr{C}} \mathbf{v}' : \tau'}}{\Sigma; \Delta, \alpha; \Gamma \rhd^{M}_{\mathscr{C}} \boldsymbol{\Lambda}\beta.\, \mathbf{v}' : \forall\beta.\, \tau'},$$

where well-formedness ensures that $\alpha \neq \beta$.

By exchange and the induction hypothesis, $\Sigma; \Delta, \beta; \Gamma[\tau_{\mathbf{a}}/\alpha] \rhd^{M}_{\mathscr{C}} \mathbf{v}'[\tau_{\mathbf{a}}/\alpha] : \tau'[\tau_{\mathbf{a}}/\alpha]$. Then,

$$\dfrac{\dfrac{\mathcal{A}, \text{exchange, IH}}{\Sigma; \Delta, \beta; \Gamma[\tau_{\mathbf{a}}/\alpha] \rhd^{M}_{\mathscr{C}} \mathbf{v}'[\tau_{\mathbf{a}}/\alpha] : \tau'[\tau_{\mathbf{a}}/\alpha]}}{\Sigma; \Delta; \Gamma[\tau_{\mathbf{a}}/\alpha] \rhd^{M}_{\mathscr{C}} \boldsymbol{\Lambda}\beta.\, \mathbf{e}'[\tau_{\mathbf{a}}/\alpha] : \forall\beta.\, \tau'[\tau_{\mathbf{a}}/\alpha]}.$$

Case $\lambda\mathbf{x}{:}\tau_{\mathbf{x}}.\, \mathbf{e}'$.

By RTC-Lam, $\tau = \tau_{\mathbf{x}} \rightarrow \tau'$, so it must be the case that

$$\dfrac{\dfrac{\mathcal{A}}{\Sigma; \Delta, \alpha; \Gamma, \mathbf{x}{:}\tau_{\mathbf{x}} \rhd^{M}_{\mathscr{C}} \mathbf{e}' : \tau'} \qquad \dfrac{\mathcal{B}}{\left|\Sigma|_{\mathrm{FL}(\lambda\mathbf{x}:\tau_{\mathbf{x}}.\mathbf{e}')}\right| = \mathsf{u}}}{\Sigma; \Delta, \alpha; \Gamma \rhd^{M}_{\mathscr{C}} \lambda\mathbf{x}{:}\tau_{\mathbf{x}}.\, \mathbf{e}' : \tau_{\mathbf{x}} \rightarrow \tau'}.$$

By the induction hypothesis, $\Sigma; \Delta; \Gamma[\tau_{\mathbf{a}}/\alpha], \mathbf{x}{:}\tau_{\mathbf{x}}[\tau_{\mathbf{a}}/\alpha] \rhd^{M}_{\mathscr{C}} \mathbf{e}'[\tau_{\mathbf{a}}/\alpha] : \tau'[\tau_{\mathbf{a}}/\alpha]$. Since $\alpha \notin \mathrm{FTV}(\Gamma)$, we know that $\Gamma[\tau_{\mathbf{a}}/\alpha] = \Gamma$. Thus,

- $\left|\Sigma|_{\mathrm{FL}(\mathbf{e}'[\tau_{\mathbf{a}}/\alpha])}\right| \sqcup \left|\Gamma[\tau_{\mathbf{a}}/\alpha]|_{\mathrm{FV}(\mathbf{e}'[\tau_{\mathbf{a}}/\alpha])}\right| = \mathsf{u}$.

Note that because $\mathrm{FTV}(\tau_{\mathbf{a}}) = \varnothing$, we know that $\alpha \notin \mathrm{FTV}((\Gamma, \mathbf{x}{:}\tau_{\mathbf{x}})[\tau_{\mathbf{a}}/\alpha])$. Then,

$$\dfrac{\dfrac{\mathcal{A}, \text{exchange, IH}}{\Sigma; \Delta; (\Gamma, \mathbf{x}{:}\tau_{\mathbf{x}})[\tau_{\mathbf{a}}/\alpha] \rhd^{M}_{\mathscr{C}} \mathbf{e}'[\tau_{\mathbf{a}}/\alpha] : \tau'[\tau_{\mathbf{a}}/\alpha]} \qquad \mathcal{B}, \Gamma[\tau_{\mathbf{a}}/\alpha] = \Gamma}{\Sigma; \Delta; \Gamma[\tau_{\mathbf{a}}/\alpha] \rhd^{M}_{\mathscr{C}} (\lambda\mathbf{x}{:}\tau_{\mathbf{x}}.\, \mathbf{e}')[\tau_{\mathbf{a}}/\alpha] : (\tau_{\mathbf{x}} \rightarrow \tau')[\tau_{\mathbf{a}}/\alpha]}.$$

Case $\mathbf{c}$.

By RTC-Con, it must be the case that

$$\dfrac{\mathrm{ty}_{\mathscr{C}}(\mathbf{c}) = \tau}{\Sigma; \Delta, \alpha; \Gamma \rhd^{M}_{\mathscr{C}} \mathbf{c} : \tau}.$$

Since $\mathbf{c}[\tau_{\mathbf{a}}/\alpha] = \mathbf{c}$ and $\mathrm{ty}_{\mathscr{C}}(\mathbf{c})[\tau_{\mathbf{a}}/\alpha] = \mathrm{ty}_{\mathscr{C}}(\mathbf{c})$,

$$\dfrac{\mathrm{ty}_{\mathscr{C}}(\mathbf{c}) = \tau}{\Sigma; \Delta; \Gamma[\tau_{\mathbf{a}}/\alpha] \rhd^{M}_{\mathscr{C}} \mathbf{c}[\tau_{\mathbf{a}}/\alpha] : \tau[\tau_{\mathbf{a}}/\alpha]}.$$

Case $\mathbf{x}$.

By RTC-Var, it must be the case that

$$\dfrac{}{\Sigma; \Delta, \alpha; \Gamma_1, \mathbf{x}{:}\tau, \Gamma_2 \rhd^{M}_{\mathscr{C}} \mathbf{x} : \tau}.$$

Since $\mathbf{x}[\tau_{\mathbf{a}}/\alpha] = \mathbf{x}$,

$$\frac{}{\Sigma; \Delta; (\Gamma_1, \mathbf{x}{:}\tau, \Gamma_2)[\tau_{\mathbf{a}}/\alpha] \rhd^M_{\mathscr{C}} \mathbf{x}[\tau_{\mathbf{a}}/\alpha] : \tau[\tau_{\mathbf{a}}/\alpha]}.$$

Case $\mathbf{f}$.

By RTC-Mod, it must be the case that

$$\frac{\mathbf{module\, f} : \tau = \mathbf{v} \in M \qquad \cdot \vdash_{\mathscr{C}} \tau}{\Sigma; \Delta, \alpha; \Gamma \rhd^M_{\mathscr{C}} \mathbf{f} : \tau}.$$

Thus $\tau[\tau_{\mathbf{a}}/\alpha] = \tau$, and since $\mathbf{f}[\tau_{\mathbf{a}}/\alpha] = \mathbf{f}$,

$$\frac{\mathbf{module\, f} : \tau = \mathbf{v} \in M \qquad \cdot \vdash_{\mathscr{C}} \tau}{\Sigma; \Delta; \Gamma[\tau_{\mathbf{a}}/\alpha] \rhd^M_{\mathscr{C}} \mathbf{f}[\tau_{\mathbf{a}}/\alpha] : \tau[\tau_{\mathbf{a}}/\alpha]}.$$

Case $\mathbf{e}'[\tau_{\mathbf{1}}]$.

By RTC-TApp, it must be the case that

$$\frac{\dfrac{\mathcal{A}}{\Sigma; \Delta, \alpha; \Gamma \rhd^M_{\mathscr{C}} \mathbf{e}' : \forall\beta.\, \tau_{\mathbf{2}}} \qquad \dfrac{\mathcal{B}}{\Delta, \alpha \vdash_{\mathscr{C}} \tau_{\mathbf{1}}}}{\Sigma; \Delta, \alpha; \Gamma \rhd^M_{\mathscr{C}} \mathbf{e}'[\tau_{\mathbf{1}}] : \tau_{\mathbf{2}}[\tau_{\mathbf{1}}/\beta]},$$

where $\tau = \tau_{\mathbf{2}}[\tau_{\mathbf{1}}/\beta]$.

By Barendregt's convention, we may assume that $\alpha \neq \beta$, and thus $(\forall\beta.\, \tau_{\mathbf{2}})[\tau_{\mathbf{a}}/\alpha] = \forall\beta.\, (\tau_{\mathbf{2}}[\tau_{\mathbf{a}}/\alpha])$. Then,

$$\frac{\dfrac{\mathcal{A}, \text{induction hypothesis}}{\Sigma; \Delta; \Gamma[\tau_{\mathbf{a}}/\alpha] \rhd^M_{\mathscr{C}} \mathbf{e}'[\tau_{\mathbf{a}}/\alpha] : (\forall\beta.\, \tau_{\mathbf{2}})[\tau_{\mathbf{a}}/\alpha]} \qquad \dfrac{\mathcal{B}, \text{Lemma 5.4.4}}{\Delta \vdash_{\mathscr{C}} \tau_{\mathbf{1}}[\tau_{\mathbf{a}}/\alpha]}}{\Sigma; \Delta; \Gamma[\tau_{\mathbf{a}}/\alpha] \rhd^M_{\mathscr{C}} (\mathbf{e}'[\tau_{\mathbf{a}}/\alpha])[\tau_{\mathbf{1}}[\tau_{\mathbf{a}}/\alpha]] : (\tau_{\mathbf{2}}[\tau_{\mathbf{a}}/\alpha])[\tau_{\mathbf{1}}[\tau_{\mathbf{a}}/\alpha]/\beta])}.$$

Case $\mathbf{e_1}\ \mathbf{e_2}$.

By RTC-App, it must be the case that

$$\frac{\dfrac{\mathcal{A}}{\Sigma_1; \Delta, \alpha; \Gamma_1 \rhd^M_{\mathscr{C}} \mathbf{e_1} : \tau_{\mathbf{1}}} \qquad \dfrac{\mathcal{B}}{\Sigma_2; \Delta, \alpha; \Gamma_2 \rhd^M_{\mathscr{C}} \mathbf{e_2} : \tau_{\mathbf{2}}}}{\Sigma_1 \boxplus \Sigma_2; \Delta, \alpha; \Gamma_1 \boxplus \Gamma_2 \rhd^M_{\mathscr{C}} \mathbf{e_1}\ \mathbf{e_2} : \tau}.$$

Then,

$$\frac{\dfrac{\mathcal{A}, \text{induction hypothesis}}{\Sigma_1; \Delta; \Gamma_1[\tau_{\mathbf{a}}/\alpha] \rhd^M_{\mathscr{C}} \mathbf{e_1}[\tau_{\mathbf{a}}/\alpha] : \tau_{\mathbf{1}}[\tau_{\mathbf{a}}/\alpha]} \qquad \dfrac{\mathcal{B}, \text{induction hypothesis}}{\Sigma_2; \Delta; \Gamma_2[\tau_{\mathbf{a}}/\alpha] \rhd^M_{\mathscr{C}} \mathbf{e_2}[\tau_{\mathbf{a}}/\alpha] : \tau_{\mathbf{2}}[\tau_{\mathbf{a}}/\alpha]}}{\Sigma_1 \boxplus \Sigma_2; \Delta; (\Gamma_1 \boxplus \Gamma_2)[\tau_{\mathbf{a}}/\alpha] \rhd^M_{\mathscr{C}} (\mathbf{e_1}\ \mathbf{e_2})[\tau_{\mathbf{a}}/\alpha] : \tau[\tau_{\mathbf{a}}/\alpha]}.$$

Case $\mathbf{if0\, e_1\ e_2\ e_3}$.

Likewise.

Case $\mathsf{g}^{\mathbf{f}}$.

By RTC-ModA, it must be the case that

$$\frac{\mathsf{module}\ \mathsf{g} : \sigma = \mathsf{v} \in M \qquad \cdot \vdash_{\mathscr{A}} \sigma}{\Sigma; \Delta, \alpha; \Gamma \rhd^{M}_{\mathscr{C}} \mathbf{g} : (\sigma)^{\mathscr{C}}},$$

where $\tau = (\sigma)^{\mathscr{A}}$.

Thus $\tau[\tau_{\mathbf{a}}/\alpha] = \tau$, and since $\mathbf{g}[\tau_{\mathbf{a}}/\alpha] = \mathbf{g}$,

$$\frac{\mathsf{module}\ \mathsf{g} : \sigma = \mathsf{v} \in M \qquad \cdot \vdash_{\mathscr{A}} \sigma}{\Sigma; \Delta; \Gamma[\tau_{\mathbf{a}}/\alpha] \rhd^{M}_{\mathscr{C}} \mathbf{g}[\tau_{\mathbf{a}}/\alpha] : (\sigma)^{\mathscr{C}}[\tau_{\mathbf{a}}/\alpha]}.$$

Case ${}_{\mathbf{f}}\mathbf{CA}^{\sigma}_{\mathsf{g}}(\mathsf{e}')$.

By RTC-Boundary, it must be the case that

$$\frac{\dfrac{\mathcal{A}}{\Sigma; \cdot; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{e}' : \sigma}}{\Sigma; \Delta, \alpha; \Gamma \rhd^{M}_{\mathscr{C}} \underset{\mathbf{f}\ \mathsf{g}}{\mathbf{CA}}{}^{\sigma}(\mathsf{e}') : (\sigma)^{\mathscr{C}}},$$

where $(\sigma)^{\mathscr{C}} = \tau$.

Since $\alpha \notin \mathrm{FTV}(\Gamma)$, we know from $\mathcal{A}$ and inspection of the type rules that $\mathsf{e}'[\tau_{\mathbf{a}}/\alpha] = \mathsf{e}'$ and $\sigma[\tau_{\mathbf{a}}/\alpha] = \sigma$. Then,

$$\frac{\dfrac{\mathcal{A}}{\Sigma; \cdot; \Gamma[\tau_{\mathbf{a}}/\alpha] \rhd^{M}_{\mathscr{A}} \mathsf{e}'[\tau_{\mathbf{a}}/\alpha] : \sigma[\tau_{\mathbf{a}}/\alpha]}}{\Sigma; \Delta; \Gamma[\tau_{\mathbf{a}}/\alpha] \rhd^{M}_{\mathscr{C}} \underset{\mathbf{f}\ \mathsf{g}}{\mathbf{CA}}{}^{\sigma[\tau_{\mathbf{a}}/\alpha]}(\mathsf{e}'[\tau_{\mathbf{a}}/\alpha]) : (\sigma)^{\mathscr{C}}[\tau_{\mathbf{a}}/\alpha]}.$$

Case ${}_{\mathbf{f}}\mathbf{CA}[\ell]^{\sigma}_{\mathsf{g}}(\mathsf{v}')$.

There are three ways to type such an expression:

- If RTC-Blessed, it must be the case that

$$\frac{\dfrac{\mathcal{A}}{\Sigma_1, \Sigma_2; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{v}' : \sigma} \qquad \dfrac{\mathcal{B}}{|\sigma| = \mathsf{a}}}{[\Sigma_1]^{\ell}, \ell{:}\,\mathbb{B}, [\Sigma_2]^{\ell}; \Delta, \alpha; \Gamma \rhd^{M}_{\mathscr{C}} \underset{\mathbf{f}\ \mathsf{g}}{\mathbf{CA}}[\ell]^{\sigma}(\mathsf{v}') : (\sigma)^{\mathscr{C}}}$$

  where $\tau = (\sigma)^{\mathscr{C}}$ and $\Sigma = [\Sigma_1]^{\ell}, \ell{:}\,\mathbb{B}, [\Sigma_2]^{\ell}$.

  Since $\alpha \notin \mathrm{FTV}(\cdot)$, we know from $\mathcal{A}$ and inspection of the type rules that $\mathsf{v}'[\tau_{\mathbf{a}}/\alpha] = \mathsf{v}'$ $\sigma[\tau_{\mathbf{a}}/\alpha] = \sigma$. Then,

$$\frac{\dfrac{\mathcal{A}}{\Sigma_1, \Sigma_2; \cdot; \cdot[\tau_{\mathbf{a}}/\alpha] \rhd^{M}_{\mathscr{A}} \mathsf{v}'[\tau_{\mathbf{a}}/\alpha] : \sigma[\tau_{\mathbf{a}}/\alpha]} \qquad \dfrac{\mathcal{B}}{|\sigma[\tau_{\mathbf{a}}/\alpha]| = \mathsf{a}}}{[\Sigma_1]^{\ell}, \ell{:}\,\mathbb{B}, [\Sigma_2]^{\ell}; \Delta; \Gamma[\tau_{\mathbf{a}}/\alpha] \rhd^{M}_{\mathscr{C}} \underset{\mathbf{f}\ \mathsf{g}}{\mathbf{CA}}[\ell]^{\sigma[\tau_{\mathbf{a}}/\alpha]}(\mathsf{v}'[\tau_{\mathbf{a}}/\alpha]) : (\sigma)^{\mathscr{C}}[\tau_{\mathbf{a}}/\alpha]}.$$

- If RTC-Defunct, it must be the case that

$$\dfrac{\dfrac{\mathcal{A}}{|\sigma| = \mathsf{a}}}{[\Sigma_1]^{\ell}, \ell{:}\, \mathbb{D}, [\Sigma_2]^{\ell}; \Delta, \alpha; \Gamma \rhd^{M}_{\mathscr{C}} \underset{\mathbf{f\,g}}{\mathbf{CA}}[\ell]^{\sigma}(\mathsf{v}') : (\sigma)^{\mathscr{C}}}$$

where $\tau = (\sigma)^{\mathscr{C}}$ and $\Sigma = [\Sigma_1]^{\ell}, \ell{:}\, \mathbb{D}, [\Sigma_2]^{\ell}$.

Since $\alpha \notin \mathrm{FTV}(\cdot)$, we know by $\mathcal{A}$ and inspection of the type rules $\sigma[\tau_{\mathbf{a}}/\alpha] = \sigma$.

Then,

$$\dfrac{\dfrac{\mathcal{A}}{|\sigma[\tau_{\mathbf{a}}/\alpha]| = \mathsf{a}}}{[\Sigma_1]^{\ell}, \ell{:}\, \mathbb{D}, [\Sigma_2]^{\ell}; \Delta; \Gamma[\tau_{\mathbf{a}}/\alpha] \rhd^{M}_{\mathscr{C}} \underset{\mathbf{f\,g}}{\mathbf{CA}}[\ell]^{\sigma[\tau_{\mathbf{a}}/\alpha]}(\mathsf{v}'[\tau_{\mathbf{a}}/\alpha]) : (\sigma)^{\mathscr{C}}[\tau_{\mathbf{a}}/\alpha]}.$$

- If RTC-Sealed, it must be the case that

$$\dfrac{\dfrac{\mathcal{A}}{\Sigma; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{v}' : \sigma} \qquad \dfrac{\mathcal{B}}{|\sigma| = \mathsf{u}}}{\Sigma; \Delta, \alpha; \Gamma \rhd^{M}_{\mathscr{C}} \underset{\mathbf{f\,g}}{\mathbf{CA}}[\ell]^{\sigma}(\mathsf{v}') : (\sigma)^{\mathscr{C}}}$$

where $\tau = (\sigma)^{\mathscr{C}}$.

Since $\alpha \notin \mathrm{FTV}(\cdot)$, we know from $\mathcal{A}$ and inspection of the type rules that $\mathsf{v}'[\tau_{\mathbf{a}}/\alpha] = \mathsf{v}'$ $\sigma[\tau_{\mathbf{a}}/\alpha] = \sigma$. Then,

$$\dfrac{\dfrac{\mathcal{A}}{\Sigma; \cdot; \cdot[\tau_{\mathbf{a}}/\alpha] \rhd^{M}_{\mathscr{A}} \mathsf{v}'[\tau_{\mathbf{a}}/\alpha] : \sigma[\tau_{\mathbf{a}}/\alpha]} \qquad \dfrac{\mathcal{B}}{|\sigma[\tau_{\mathbf{a}}/\alpha]| = \mathsf{u}}}{\Sigma; \Delta; \Gamma[\tau_{\mathbf{a}}/\alpha] \rhd^{M}_{\mathscr{C}} \underset{\mathbf{f\,g}}{\mathbf{CA}}[\ell]^{\sigma[\tau_{\mathbf{a}}/\alpha]}(\mathsf{v}'[\tau_{\mathbf{a}}/\alpha]) : (\sigma)^{\mathscr{C}}[\tau_{\mathbf{a}}/\alpha]}.$$

(*ii*) By induction on the structure of e:

Case $\Lambda\beta^{\mathsf{q}'}.\,\mathsf{v}'$.

By RTA-TLam, $\sigma = \forall\beta^{\mathsf{q}}.\,\sigma'$, so it must be the case that

$$\dfrac{\dfrac{\mathcal{A}}{\Sigma; \Delta, \alpha^{\mathsf{q_a}}, \beta^{\mathsf{q}}; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{v}' : \sigma'}}{\Sigma; \Delta, \alpha^{\mathsf{q_a}}; \Gamma \rhd^{M}_{\mathscr{A}} \Lambda\beta^{\mathsf{q}}.\,\mathsf{v}' : \forall\beta^{\mathsf{q}}.\,\sigma'},$$

where well-formedness ensures that $\alpha^{\mathsf{q_a}} \neq \beta^{\mathsf{q}}$.

By exchange and the induction hypothesis, $\Sigma; \Delta, \beta^{\mathsf{q}}; \Gamma[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] \rhd^{M}_{\mathscr{A}} \mathsf{v}'[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] : \sigma'[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}]$.

Then,

$$\dfrac{\dfrac{\mathcal{A}, \text{exchange, IH}}{\Sigma; \Delta, \beta^{\mathsf{q}}; \Gamma[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] \rhd^{M}_{\mathscr{A}} \mathsf{v}'[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] : \sigma'[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}]}}{\Sigma; \Delta; \Gamma[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] \rhd^{M}_{\mathscr{A}} \Lambda\beta^{\mathsf{q}}.\,\mathsf{v}'[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] : \forall\beta^{\mathsf{q}}.\,\sigma'[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}]}.$$

Case $\lambda\mathsf{x}{:}\sigma_{\mathsf{x}}.\,\mathsf{e}'$.

By RTA-Lam, $\sigma = \sigma_x \xrightarrow{q}\!\circ\, \sigma'$, so it must be the case that

$$\frac{\dfrac{\mathcal{A}}{\Sigma; \Delta, \alpha^{q_a}; \Gamma, x{:}\sigma_x \rhd^M_{\mathscr{A}} e' : \sigma'} \qquad \dfrac{\mathcal{B}}{\left|\Sigma|_{\mathrm{FL}(\lambda x:\sigma_x.e')}\right| \sqcup \left|\Gamma|_{\mathrm{FV}(\lambda x:\sigma_x.e')}\right| = q}}{\Sigma; \Delta, \alpha^{q_a}; \Gamma \rhd^M_{\mathscr{A}} \lambda x{:}\sigma_x.\, e' : \sigma_x \xrightarrow{q}\!\circ\, \sigma'}.$$

By the i.h., $\Sigma; \Delta; \Gamma[\sigma_a/\alpha^{q_a}], x{:}\sigma_x[\sigma_a/\alpha^{q_a}] \rhd^M_{\mathscr{A}} e'[\sigma_a/\alpha^{q_a}] : \sigma'[\sigma_a/\alpha^{q_a}]$.
Since $\alpha^{q_a} \notin \mathrm{FTV}(\Gamma)$, we know that $\Gamma[\sigma_a/\alpha^{q_a}] = \Gamma$.
Thus, $\left|\Sigma|_{\mathrm{FL}(e'[\sigma_a/\alpha^{q_a}])}\right| \sqcup \left|\Gamma[\sigma_a/\alpha^{q_a}]|_{\mathrm{FV}(e'[\sigma_a/\alpha^{q_a}])}\right| = q$.
Note that because $\mathrm{FTV}(\sigma_a) = \varnothing$, we know that $\alpha^{q_a} \notin \mathrm{FTV}((\Gamma, x{:}\sigma_x)[\sigma_a/\alpha^{q_a}])$.
Then,

$$\frac{\dfrac{\mathcal{A}, \text{exchange, IH}}{\Sigma; \Delta; (\Gamma, x{:}\sigma_x)[\sigma_a/\alpha^{q_a}] \rhd^M_{\mathscr{A}} e'[\sigma_a/\alpha^{q_a}] : \sigma'[\sigma_a/\alpha^{q_a}]} \qquad \mathcal{B}, \Gamma[\sigma_a/\alpha^{q_a}] = \Gamma}{\Sigma; \Delta; \Gamma[\sigma_a/\alpha^{q_a}] \rhd^M_{\mathscr{A}} (\lambda x{:}\sigma_x.\, e')[\sigma_a/\alpha^{q_a}] : (\sigma_x \xrightarrow{q}\!\circ\, \sigma')[\sigma_a/\alpha^{q_a}]},$$

Case c.

By RTA-Con, it must be the case that

$$\frac{\mathrm{ty}_{\mathscr{A}}(c) = \sigma}{\Sigma; \Delta, \alpha^{q_a}; \Gamma \rhd^M_{\mathscr{A}} c : \sigma}.$$

Since $c[\sigma_a/\alpha^{q_a}] = c$ and $\mathrm{ty}_{\mathscr{A}}(c)[\sigma_a/\alpha^{q_a}] = \mathrm{ty}_{\mathscr{A}}(c)$,

$$\frac{\mathrm{ty}_{\mathscr{A}}(c) = \sigma}{\Sigma; \Delta; \Gamma[\sigma_a/\alpha^{q_a}] \rhd^M_{\mathscr{A}} c[\sigma_a/\alpha^{q_a}] : \sigma[\sigma_a/\alpha^{q_a}]}.$$

Case x.

By RTA-Var, it must be the case that

$$\frac{}{\Sigma; \Delta, \alpha^{q_a}; \Gamma_1, x{:}\sigma, \Gamma_2 \rhd^M_{\mathscr{A}} x : \sigma}.$$

Since $x[\sigma_a/\alpha^{q_a}] = x$,

$$\frac{}{\Sigma; \Delta; (\Gamma_1, x{:}\sigma, \Gamma_2)[\sigma_a/\alpha^{q_a}] \rhd^M_{\mathscr{A}} x[\sigma_a/\alpha^{q_a}] : \sigma[\sigma_a/\alpha^{q_a}]}.$$

Case f.

By RTA-Mod, it must be the case that

$$\frac{\mathsf{module}\ f : \sigma = v \in M \qquad \cdot \vdash_{\mathscr{A}} \sigma}{\Sigma; \Delta, \alpha^{q_a}; \Gamma \rhd^M_{\mathscr{A}} f : \sigma}.$$

Thus $\sigma[\sigma_a/\alpha^{q_a}] = \sigma$, and since $f[\sigma_a/\alpha^{q_a}] = f$,

$$\frac{\mathsf{module}\ f : \sigma = v \in M \qquad \cdot \vdash_{\mathscr{A}} \sigma}{\Sigma; \Delta; \Gamma[\sigma_a/\alpha^{q_a}] \rhd^M_{\mathscr{A}} f[\sigma_a/\alpha^{q_a}] : \sigma[\sigma_a/\alpha^{q_a}]}.$$

Case $\mathsf{e}'[\sigma_1]$.

By RTA-TApp, it must be the case that

$$\dfrac{\dfrac{\mathcal{A}}{\Sigma; \Delta, \alpha^{\mathsf{q_a}}; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{e}' : \forall\beta^{\mathsf{q}}.\, \sigma_2} \qquad \dfrac{\mathcal{B}}{\Delta, \alpha^{\mathsf{q_a}} \vdash_{\mathscr{A}} \sigma_1} \qquad \dfrac{\mathcal{C}}{|\sigma_1| \sqsubseteq \mathsf{q}'}}{\Sigma; \Delta, \alpha^{\mathsf{q_a}}; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{e}'[\sigma_1] : \sigma_2[\sigma_1/\beta^{\mathsf{q}}]},$$

where $\sigma = \sigma_2[\sigma_1/\beta^{\mathsf{q}}]$.

By Barendregt's convention, we assume that $\alpha^{\mathsf{q_a}} \neq \beta^{\mathsf{q}}$; thus $(\forall\beta^{\mathsf{q}}.\, \sigma_2)[\sigma_a/\alpha^{\mathsf{q_a}}] = \forall\beta^{\mathsf{q}}.\, (\sigma_2[\sigma_a/\alpha^{\mathsf{q_a}}])$. Then,

$$\dfrac{\dfrac{\mathcal{A}, \text{induction hypothesis}}{\Sigma; \Delta; \Gamma[\sigma_a/\alpha^{\mathsf{q_a}}] \rhd^{M}_{\mathscr{A}} \mathsf{e}'[\sigma_a/\alpha^{\mathsf{q_a}}] : (\forall\beta^{\mathsf{q}}.\, \sigma_2)[\sigma_a/\alpha^{\mathsf{q_a}}]} \qquad \dfrac{\mathcal{B}, \text{Lemma 5.4.4}}{\Delta \vdash_{\mathscr{A}} \sigma_1[\sigma_a/\alpha^{\mathsf{q_a}}]} \qquad \mathcal{D}}{\Sigma; \Delta; \Gamma[\sigma_a/\alpha^{\mathsf{q_a}}] \rhd^{M}_{\mathscr{A}} (\mathsf{e}'[\sigma_a/\alpha^{\mathsf{q_a}}])[\sigma_1[\sigma_a/\alpha^{\mathsf{q_a}}]] : (\sigma_2[\sigma_a/\alpha^{\mathsf{q_a}}])[\sigma_1[\sigma_a/\alpha^{\mathsf{q_a}}]/\beta^{\mathsf{q}}])}$$

where

$$\mathcal{D} = \dfrac{\dfrac{|\sigma_a| \sqsubseteq \alpha^{\mathsf{q_a}}, \text{Lemma 5.4.4}}{|\sigma_1[\sigma_a/\alpha^{\mathsf{q_a}}]| \sqsubseteq |\sigma_1|} \qquad \dfrac{\mathcal{C}}{|\sigma_1| \sqsubseteq \mathsf{q}'}}{|\sigma_1[\sigma_a/\alpha^{\mathsf{q_a}}]| \sqsubseteq \mathsf{q}'}.$$

Case $\mathsf{e}_1\ \mathsf{e}_2$.

By RTA-App, it must be the case that

$$\dfrac{\dfrac{\mathcal{A}}{\Sigma_1; \Delta, \alpha^{\mathsf{q_a}}; \Gamma_1 \rhd^{M}_{\mathscr{A}} \mathsf{e}_1 : \sigma_1} \qquad \dfrac{\mathcal{B}}{\Sigma_2; \Delta, \alpha^{\mathsf{q_a}}; \Gamma_2 \rhd^{M}_{\mathscr{A}} \mathsf{e}_2 : \sigma_2}}{\Sigma_1 \boxplus \Sigma_2; \Delta, \alpha^{\mathsf{q_a}}; \Gamma_1 \boxplus \Gamma_2 \rhd^{M}_{\mathscr{A}} \mathsf{e}_1\ \mathsf{e}_2 : \sigma}.$$

Then,

$$\dfrac{\dfrac{\mathcal{A}, \text{induction hypothesis}}{\Sigma_1; \Delta; \Gamma_1[\sigma_a/\alpha^{\mathsf{q_a}}] \rhd^{M}_{\mathscr{A}} \mathsf{e}_1[\sigma_a/\alpha^{\mathsf{q_a}}] : \sigma_1[\sigma_a/\alpha^{\mathsf{q_a}}]} \qquad \dfrac{\mathcal{B}, \text{induction hypothesis}}{\Sigma_2; \Delta; \Gamma_2[\sigma_a/\alpha^{\mathsf{q_a}}] \rhd^{M}_{\mathscr{A}} \mathsf{e}_2[\sigma_a/\alpha^{\mathsf{q_a}}] : \sigma_2[\sigma_a/\alpha^{\mathsf{q_a}}]}}{\Sigma_1 \boxplus \Sigma_2; \Delta; (\Gamma_1 \boxplus \Gamma_2)[\sigma_a/\alpha^{\mathsf{q_a}}] \rhd^{M}_{\mathscr{A}} (\mathsf{e}_1\ \mathsf{e}_2)[\sigma_a/\alpha^{\mathsf{q_a}}] : \sigma[\sigma_a/\alpha^{\mathsf{q_a}}]}.$$

Case $\mathsf{if0}\ \mathsf{e}_1\ \mathsf{e}_2\ \mathsf{e}_3$.

Likewise.

Case $\langle \mathsf{e}_1, \mathsf{e}_2 \rangle$.

Likewise.

Case $\mathsf{let}\ \langle \mathsf{x}, \mathsf{y} \rangle = \mathsf{e}_1\ \mathsf{in}\ \mathsf{e}_2$.

By RTA-Let, it must be the case that

$$\dfrac{\dfrac{\mathcal{A}}{\Sigma_1; \Delta, \alpha^{\mathsf{q_a}}; \Gamma_1 \rhd^{M}_{\mathscr{A}} \mathsf{e}_1 : \sigma_{\mathsf{x}} \otimes \sigma_{\mathsf{y}}} \qquad \dfrac{\mathcal{B}}{\Sigma_2; \Delta, \alpha^{\mathsf{q_a}}; \Gamma_2, \mathsf{x}{:}\sigma_{\mathsf{x}}, \mathsf{y}{:}\sigma_{\mathsf{y}} \rhd^{M}_{\mathscr{A}} \mathsf{e}_2 : \sigma}}{\Sigma_1 \boxplus \Sigma_2; \Delta, \alpha^{\mathsf{q_a}}; \Gamma_1 \boxplus \Gamma_2 \rhd^{M}_{\mathscr{A}} \mathsf{let}\ \langle \mathsf{x}, \mathsf{y} \rangle = \mathsf{e}_1\ \mathsf{in}\ \mathsf{e}_2 : \sigma}.$$

Then,

$$\dfrac{\dfrac{\mathcal{A}\text{, induction hypothesis}}{(\Sigma_1; \Delta; \Gamma_1 \triangleright^M_{\mathscr{A}} \mathsf{e}_1 : \sigma_{\mathsf{x}} \otimes \sigma_{\mathsf{y}})[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}]} \qquad \dfrac{\mathcal{B}\text{, induction hypothesis}}{(\Sigma_2; \Delta; \Gamma_2, \mathsf{x}{:}\sigma_{\mathsf{x}}, \mathsf{y}{:}\sigma_{\mathsf{y}} \triangleright^M_{\mathscr{A}} \mathsf{e}_2 : \sigma)[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}]}}{(\Sigma_1 \boxplus \Sigma_2; \Delta; \Gamma_1 \boxplus \Gamma_2 \triangleright^M_{\mathscr{A}} \mathsf{let}\ \langle \mathsf{x}, \mathsf{y} \rangle = \mathsf{e}_1\ \mathsf{in}\ \mathsf{e}_2 : \sigma)[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}]}.$$

Case $\mathbf{g}^{\mathsf{f}}$.

There are two rules for typing a $\lambda_{\mathscr{C}}$ module reference in an $\lambda^{\mathscr{A}}$ expression:

- If RTA-ModC, it must be the case that

$$\dfrac{\mathbf{module\,g} : \tau = \mathbf{v} \in M \qquad \cdot \vdash_{\mathscr{C}} \tau}{\Sigma; \Delta, \alpha^{\mathsf{q_a}}; \Gamma \triangleright^M_{\mathscr{A}} \mathsf{g} : (\tau)^{\mathscr{A}}},$$

  where $\sigma = (\tau)^{\mathscr{C}}$.

  Thus $\sigma[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] = \sigma$, and since $\mathsf{g}[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] = \mathsf{g}$,

$$\dfrac{\mathbf{module\,g} : \tau = \mathbf{v} \in M \qquad \cdot \vdash_{\mathscr{C}} \tau}{\Sigma; \Delta; \Gamma[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] \triangleright^M_{\mathscr{A}} \mathsf{g}[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] : (\tau)^{\mathscr{A}}[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}]}.$$

- If RTA-ModI, it must be the case that

$$\dfrac{\mathbf{interface\,g} :> \sigma = \mathbf{f} \in M \qquad \cdot \vdash_{\mathscr{A}} \sigma}{\Sigma; \Delta, \alpha^{\mathsf{q_a}}; \Gamma \triangleright^M_{\mathscr{A}} \mathsf{g} : \sigma}.$$

  Thus $\sigma[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] = \sigma$, and since $\mathsf{g}[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] = \mathsf{g}$,

$$\dfrac{\mathbf{interface\,g} :> \sigma = \mathbf{f} \in M \qquad \cdot \vdash_{\mathscr{A}} \sigma}{\Sigma; \Delta; \Gamma[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] \triangleright^M_{\mathscr{A}} \mathsf{g}[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] : \sigma[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}]}.$$

Case ℓ.

By RTA-Loc, it must be the case that

$$\dfrac{}{\Sigma_1, \ell{:}\sigma_\ell, \Sigma_1; \Delta, \alpha^{\mathsf{q_a}}, \Gamma \triangleright^M_{\mathscr{A}} \ell : \sigma_\ell\ \mathsf{ref}},$$

where $\sigma = \sigma_\ell\ \mathsf{ref}$.

Thus $\ell[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] = \ell$, and since $\mathrm{FTV}(\Sigma) = \varnothing$, we know that $\sigma_\ell[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] = \sigma_\ell$. Then,

$$\dfrac{}{\Sigma_1, \ell{:}\sigma_\ell, \Sigma_1; \Delta, \Gamma[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] \triangleright^M_{\mathscr{A}} \ell[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] : \sigma_\ell[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}]\ \mathsf{ref}}.$$

Case ${}^{\sigma}_{\mathsf{f}}\mathsf{AC}_{\mathbf{g}}(\mathbf{e}')$.

By RTA-Boundary, it must be the case that

$$\dfrac{\dfrac{\mathcal{A}}{\Sigma; \cdot; \Gamma \triangleright^M_{\mathscr{C}} \mathbf{e}' : (\sigma)^{\mathscr{C}}}}{\Sigma; \Delta, \alpha^{\mathsf{q_a}}; \Gamma \triangleright^M_{\mathscr{A}} {}^{\sigma}\underset{\mathsf{f}\ \mathbf{g}}{\mathsf{AC}}(\mathbf{e}') : \sigma}.$$

Since $\alpha^{\mathsf{q_a}} \notin \mathrm{FTV}(\Gamma)$, we know from $\mathcal{A}$ and inspection of the type rules that $\mathbf{e}'[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] = \mathbf{e}'$ and $\sigma[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q_a}}] = \sigma$. Then,

$$\dfrac{\dfrac{\mathcal{A}}{\Sigma; \cdot; \Gamma[\sigma_a/\alpha^{q_a}] \triangleright^{M}_{\mathscr{C}} \mathbf{e}'[\sigma_a/\alpha^{q_a}] : (\sigma[\sigma_a/\alpha^{q_a}])^{\mathscr{C}}}}{\Sigma; \Delta; \Gamma[\sigma_a/\alpha^{q_a}] \triangleright^{M}_{\mathscr{A}} {}^{\sigma[\sigma_a/\alpha^{q_a}]}\mathsf{AC}_{\mathsf{f}\,\mathbf{g}} (\mathbf{e}'[\sigma_a/\alpha^{q_a}]) : \sigma[\sigma_a/\alpha^{q_a}]}.$$

Case ${}^{\sigma}_{\mathsf{f}}\mathsf{AC}[\,]_{\mathbf{g}}(\mathbf{v}')$.

Likewise, by rule RTA-Sealed, but with a second premiss that $(\sigma)^{\mathscr{A}} = \tau^{\mathsf{w}}$. □

Definition 5.4.7 (Promotion-Worthy)**.** *We say that a term* $\mathbf{e}$ *is* ***worthy with respect to*** Σ *if* $\left|\Sigma|_{\mathrm{FL}(\mathbf{e})}\right| = \mathsf{u}$*. Likewise, a term* e *is* ***worthy with respect to*** Γ *if* $\left|\Gamma|_{\mathrm{FV}(\mathsf{e})}\right| = \mathsf{u}$ *and is* ***worthy with respect to*** Σ *if* $\left|\Sigma|_{\mathrm{FL}(\mathsf{e})}\right| = \mathsf{u}$*.*

If e *is worthy with respect to* Σ *and* Γ*, then we write* $\Sigma; \Gamma \triangleright_{\mathscr{A}} \mathsf{e}$ worthy*; otherwise, we write* $\Sigma; \Gamma \not\triangleright_{\mathscr{A}} \mathsf{e}$ worthy*. Likewise for* $\mathbf{e}$*.*

Worthiness captures our notion of terms that can be "promoted" to allow for unlimited use. In particular, λ closures in subcalculus $\lambda^{\mathscr{A}}$ are given an unlimited (u) type if they are worthy, and an affine (a) type if they are not. Closures in subcalculus $\lambda_{\mathscr{C}}$ are required to be worthy, since they should not close over affine things.

Note that we impose no such requirement on Λ closures, as they have the same qualifier as their body, which regulates their usage accordingly.

Lemma 5.4.8 (No hidden locations)**.** *The type of a value tells us information about whether and where locations might appear in that value:*

(*i*) *If* $\Sigma; \Delta; \cdot \triangleright^{M}_{\mathscr{C}} \mathbf{v} : \tau$ *then* $\left|\Sigma|_{\mathrm{FL}(\mathbf{v})}\right| = \mathsf{u}$*; that is,* $\mathbf{v}$ *is worthy.*

(*ii*) *If* $\Sigma; \Delta; \cdot \triangleright^{M}_{\mathscr{A}} \mathsf{v} : \sigma$ *then* $\left|\Sigma|_{\mathrm{FL}(\mathsf{v})}\right| \sqsubseteq |\sigma|$*; that is, if* σ *is unlimited then* v *must be worthy.*

Proof. By mutual induction on v and $\mathbf{v}$.

(*i*) By cases on $\mathbf{v}$:

Case $\mathbf{\Lambda}\alpha.\,\mathbf{v}'$.

By inversion of RTC-TLam and the induction hypothesis at $\mathbf{v}'$, since $\mathrm{FL}(\mathbf{v}') = \mathrm{FL}(\mathbf{\Lambda}\alpha.\,\mathbf{v}')$.

Case $\lambda\mathbf{x}{:}\tau'.\,\mathbf{e}$.

By inversion of RTC-Lam.

Case $\mathbf{c}$.

Since $\mathrm{FL}(\mathbf{c}) = \varnothing$, $\Sigma|_{\varnothing} = \cdot$, and $\left|\cdot\right| = \mathsf{u}$.

Case $(z-)$.

As for $\mathbf{c}$.

Case ${}_{\mathbf{f}}\mathbf{CA}[\ell]^{\sigma'}_{\mathsf{g}}(\mathsf{v}')$.

There three possible rules for typing this term: RTC-Blessed, RTC-Defunct, and RTC-Sealed.

The first two require that $\Sigma = [\Sigma_1]^\ell, \ell{:}\tau, [\Sigma_2]^\ell$ for particular Σ_1, Σ_2, and τ. By inspection of the definition of $[\Sigma_i]^\ell$, it should be clear that there are no σ types in the range of Σ. Thus, $\left|\Sigma|_{\mathrm{FL}(\mathbf{v})}\right| = \mathsf{u}$.

For RTC-Sealed, it must be the case that $|\sigma'| = \mathsf{u}$ and that $\Sigma; \Delta; \cdot \rhd^M_{\mathscr{A}} \mathsf{v}' : \sigma'$. By the induction hypothesis (ii), $\left|\Sigma|_{\mathrm{FL}(\mathsf{v}')}\right| \sqsubseteq |\sigma'| = \mathsf{u}$. Since $\mathrm{FL}({}_{\mathbf{f}}\mathbf{CA}[\ell]^{\sigma'}_{\mathsf{g}}(\mathsf{v}')) = \mathrm{FL}(\mathsf{v}') \cup \{\ell\}$, and since $\Sigma(\ell) = \tau'$, we see that $\left|\Sigma|_{\mathrm{FL}(\mathbf{v})}\right| = \mathsf{u}$.

(ii) By cases on v:

Case $\Lambda\alpha^{\mathsf{q}}.\,\mathsf{v}'$.

By RTA-TLam, it must be the case that $\Sigma; \Delta; \cdot \rhd^M_{\mathscr{A}} \Lambda\alpha^{\mathsf{q}}.\,\mathsf{v}' : \forall\alpha^{\mathsf{q}}\,\sigma'$. where $\forall\alpha^{\mathsf{q}}.\,\sigma' = \sigma$, and thus $|\sigma| = |\forall\alpha^{\mathsf{q}}.\,\sigma'| = |\sigma'|$. By inversion, it must be the case that $\Sigma; \Delta, \alpha^{\mathsf{q}}; \cdot \rhd^M_{\mathscr{A}} \mathsf{v}' : \sigma'$ By the induction hypothesis, $\left|\Sigma|_{\mathrm{FL}(\mathsf{v}')}\right| \sqsubseteq |\sigma'|$, and since $\mathrm{FL}(\mathsf{v}') = \mathrm{FL}(\mathsf{v})$, we have that $\left|\Sigma|_{\mathrm{FL}(\mathsf{v})}\right| \sqsubseteq |\sigma|$.

Case $\lambda \mathsf{x}{:}\sigma'.\,\mathsf{e}$.

Let $\mathsf{q} = |\sigma|$. Then by inversion of RTA-Lam, $\mathsf{q} = \mathsf{q}_1 \sqcup \mathsf{q}_2$ where $\left|\cdot|_{\mathrm{FV}(\mathsf{v})}\right| = \mathsf{q}_1$ and $\left|\Sigma|_{\mathrm{FL}(\mathsf{v})}\right| = \mathsf{q}_2$. Since $\mathsf{q}_1 = \mathsf{u} = \bot$, we know that $\mathsf{q}_2 = \mathsf{q} = |\sigma|$.

Case c.

Since $\mathrm{FL}(\mathsf{c}) = \varnothing$, we know that $\left|\Sigma_{\mathrm{FL}(\mathsf{c})}\right| = \mathsf{u} \sqsubseteq \mathsf{q}$ for all q.

Case $\langle \mathsf{v}_1, \mathsf{v}_2 \rangle$.

This has a pair type of the form $\sigma_1 \otimes \sigma_2$, and therefore

$$\begin{aligned}
\left|\Sigma|_{\mathrm{FL}(\mathsf{v})}\right| &= \left|\Sigma|_{\mathrm{FL}(\mathsf{v}_1)\cup\mathrm{FL}(\mathsf{v}_2)}\right| && \text{def. of } \mathrm{FL}(\langle v_1, v_2\rangle) \\
&= \left|\Sigma|_{\mathrm{FL}(\mathsf{v}_1)} \cup \Sigma|_{\mathrm{FL}(\mathsf{v}_2)}\right| && \text{set theory} \\
&= \left|\Sigma|_{\mathrm{FL}(\mathsf{v}_1)}\right| \sqcup \left|\Sigma|_{\mathrm{FL}(\mathsf{v}_2)}\right| && \text{monotonicity of } |\cdot| \\
&\sqsubseteq |\sigma_1| \sqcup |\sigma_2| && \text{i.h. twice; monotonicity of } \sqcup \\
&= |\sigma| && \text{def. of } |\sigma_1 \otimes \sigma_2|.
\end{aligned}$$

Case $(z-), \mathsf{new}[\sigma_1], \mathsf{swap}[\sigma_1], \mathsf{swap}[\sigma_1][\sigma_2]$.

As for c.

Case ℓ.

By inversion of RTA-Loc, this has type $\sigma'\ \mathsf{ref}$ if and only if $\ell \in \operatorname{dom}\Sigma$ and $\Sigma(\ell) = \sigma'$. Then

$$\begin{aligned}
|\sigma| &= |\sigma'\ \mathsf{ref}| \\
&= \mathsf{a} \\
&= \left|\cdot, \ell : \sigma'\right| \\
&= \left|\Sigma|_{\{\ell\}}\right| \\
&= \left|\Sigma|_{\mathrm{FL}(\ell)}\right|.
\end{aligned}$$

Case ${}^{\sigma'}_{\mathsf{f}}\mathsf{AC}[\,]_{\mathbf{g}}(\mathbf{v}')$.

By inversion of RTA-SEALED, we know that $\Sigma; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{v}' : \tau$ for some type τ. Then by part (i) of the induction hypothesis, $\left|\Sigma|_{\mathrm{FL}(\mathbf{v}')}\right| = \mathsf{u}$. Since $\mathrm{FL}(\mathsf{v}) = \mathrm{FL}(\mathbf{v}')$, we have that $\left|\Sigma|_{\mathrm{FL}(\mathsf{v})}\right| = \mathsf{u} \sqsubseteq \mathsf{q}$ for all q. □

Lemma 5.4.9 (Substitution)**.**

(i) *If* $\Sigma_1; \Delta; \Gamma, \mathbf{x} : \tau_{\mathbf{x}} \rhd^M_{\mathscr{C}} \mathbf{e} : \tau$ *and* $\Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{v} : \tau_{\mathbf{x}}$ *where* $\Sigma_1 \boxplus \Sigma_2 = \Sigma$*, then* $\Sigma; \Delta; \Gamma \rhd^M_{\mathscr{C}} \mathbf{e}[\mathbf{v}/\mathbf{x}] : \tau$*. If* $\mathbf{e}$ *and* $\mathbf{v}$ *are worthy in their respective contexts, then* $\mathbf{e}[\mathbf{v}/\mathbf{x}]$ *is worthy as well.*

(ii) *If* $\Sigma_1; \Delta; \Gamma, \mathsf{x} : \sigma_{\mathsf{x}} \rhd^M_{\mathscr{A}} \mathsf{e} : \sigma$ *and* $\Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{v} : \sigma_{\mathsf{x}}$ *where* $\Sigma_1 \boxplus \Sigma_2 = \Sigma$*, then* $\Sigma; \Delta; \Gamma \rhd^M_{\mathscr{A}} \mathsf{e}[\mathsf{v}/\mathsf{x}] : \sigma$*. If* e *and* v *are worthy in their respective contexts, then* $\mathsf{e}[\mathsf{v}/\mathsf{x}]$ *is worthy as well.*

Proof. By induction on the structure of the type derivation for $\mathbf{e}$ or e. We consider each proof tree by the expression in its conclusion (where possible).

(i) By cases in $\mathbf{e}$, considering multiple type rules where necessary.

Case $\mathbf{\Lambda}\alpha.\,\mathbf{v}'$.

By rule RTC-TLAM, it must be the case that

- $\Sigma_1; \Delta, \alpha; \Gamma, \mathbf{x} : \tau_{\mathbf{x}} \rhd^M_{\mathscr{C}} \mathbf{v}' : \tau'$, where
- $\tau = \forall\alpha.\,\tau'$.

By the induction hypothesis,

- $\Sigma; \Delta, \alpha; \Gamma \rhd^M_{\mathscr{C}} \mathbf{v}'[\mathbf{v}/\mathbf{x}] : \tau'$ and
- $\Sigma; \Gamma \rhd_{\mathscr{C}} \mathbf{v}'[\mathbf{v}/\mathbf{x}]$ worthy.

By the Barendregt condition, we assume that $(\mathbf{\Lambda}\alpha.\,\mathbf{v}')[\mathbf{v}/\mathbf{x}] = \mathbf{\Lambda}\alpha.(\mathbf{v}'[\mathbf{v}/\mathbf{x}])$.
By rule RTC-TLAM,

- $\Sigma; \Delta; \Gamma \rhd^M_{\mathscr{C}} (\mathbf{\Lambda}\alpha.\,\mathbf{v}')[\mathbf{v}/\mathbf{x}] : \forall\alpha.\,\tau'$.

Since $\mathbf{\Lambda}\alpha.\,(\mathbf{v}'[\mathbf{v}/\mathbf{x}])$ is a value, by Lemma 5.4.8, it is worthy.

Case $\lambda\mathbf{y}{:}\tau_{\mathbf{y}}.\,\mathbf{e}'$.

Either $\mathbf{x} = \mathbf{y}$ or $\mathbf{x} \neq \mathbf{y}$:

Case $\mathbf{x} = \mathbf{y}$.

Then $\mathbf{e}[\mathbf{v}/\mathbf{x}] = \mathbf{e}$.
Since $\mathbf{x} \notin \mathrm{FV}((\lambda\mathbf{y}{:}\tau_{\mathbf{y}}\,\mathbf{e}')[\mathbf{v}/\mathbf{x}])$, and by weakening,

- $\Sigma_1 \boxplus \Sigma_2; \Delta; \Gamma \rhd^M_{\mathscr{C}} \mathbf{e}[\mathbf{v}/\mathbf{x}] : \tau$.

Furthermore, if $\Sigma_1; \Gamma \rhd \mathbf{e}$ worthy and since it types only if $\mathrm{FL}(\mathbf{e}) \subseteq \mathrm{dom}\,\Sigma_1$,

- $\Sigma_1 \boxplus \Sigma_2; \Gamma \rhd \mathbf{e}$ worthy.

Several other base cases in which $\mathbf{x}$ is not free in $\mathbf{e}$ proceed accordingly.

Case $\mathbf{x} \neq \mathbf{y}$.

By rule RTC-Lam, it must be the case that

- $\Sigma_1; \Delta; \Gamma, \mathbf{x}{:}\tau_{\mathbf{x}}, \mathbf{y}{:}\tau_{\mathbf{y}} \rhd^{M}_{\mathscr{C}} \mathbf{e}' : \tau'$ where
- $\tau = \tau_{\mathbf{y}} \to \tau'$ and
- $\Sigma_1; \Gamma, \mathbf{x}{:}\tau_{\mathbf{x}} \rhd \lambda\mathbf{y}{:}\tau_{\mathbf{y}}.\, \mathbf{e}'$ worthy.

By our exchange observation, we have that

- $\Sigma_1; \Delta; \Gamma, \mathbf{y}{:}\tau_{\mathbf{y}}, \mathbf{x}{:}\tau_{\mathbf{x}} \rhd^{M}_{\mathscr{C}} \mathbf{e}' : \tau$,

and applying induction,

- $\Sigma_1 \boxplus \Sigma_2; \Delta; \Gamma, \mathbf{y}{:}\tau_{\mathbf{y}} \rhd^{M}_{\mathscr{C}} \mathbf{e}'[\mathbf{v}/\mathbf{x}] : \tau$.

Since $\mathrm{FL}(\mathbf{e}') = \mathrm{FL}(\lambda\mathbf{y}{:}\tau_{\mathbf{y}}.\, \mathbf{e}')$, we know that $\mathbf{e}'$ is worthy.

Furthermore, since $\mathbf{v}$ is worthy by Lemma 5.4.8, by induction, $\mathbf{e}'[\mathbf{v}/\mathbf{x}]$ is worthy, and thus $\lambda\mathbf{y}{:}\tau_{\mathbf{y}}.\, \mathbf{e}'[\mathbf{v}/\mathbf{x}]$, which has no more free locations, must be worthy as well.

Thus, by RTC-Lam,

- $\Sigma_1 \boxplus \Sigma_2; \Delta; \Gamma \rhd^{M}_{\mathscr{C}} (\lambda\mathbf{y}{:}\tau_{\mathbf{y}}.,\mathbf{e}')[\mathbf{v}/\mathbf{x}] : \tau$.

Case $\mathbf{c}$.

As before when $\mathbf{x} \notin \mathrm{FV}(\mathbf{e})$

Case $\mathbf{y}$.

Either $\mathbf{x} = \mathbf{y}$ or $\mathbf{x} \neq \mathbf{y}$:

Case $\mathbf{x} = \mathbf{y}$.

Then $\tau = \tau_{\mathbf{x}}$, by RTC-Var.

Since $\mathbf{x}[\mathbf{v}/\mathbf{x}] = \mathbf{v}$, we thus have that

- $\Sigma_2; \cdot; \cdot \rhd^{M}_{\mathscr{C}} \mathbf{x}[\mathbf{v}/\mathbf{x}] : \tau$.

By our weakening observation,

- $\Sigma_1 \boxplus \Sigma_2; \Delta; \Gamma \rhd^{M}_{\mathscr{C}} \mathbf{x}[\mathbf{v}/\mathbf{x}] : \tau$.

Furthermore, if $\mathbf{v}$ is worthy, so is $\mathbf{x}[\mathbf{v}/\mathbf{x}]$.

Case $\mathbf{x} \neq \mathbf{y}$.

If $\mathbf{x} \neq \mathbf{y}$, then as before when $\mathbf{x} \notin \mathrm{FV}(\mathbf{e})$.

Case $\mathbf{e}'[\tau_{\mathbf{a}}]$.

By inverting RTC-TApp, we have that

- $\Sigma_1; \Delta; \Gamma, \mathbf{x}{:}\tau_{\mathbf{x}} \rhd^{M}_{\mathscr{C}} \mathbf{e}' : \forall\alpha.\, \tau_{\mathbf{b}}$ where
- $\tau = \tau_{\mathbf{b}}[\tau_{\mathbf{a}}/\alpha]$.

By the induction hypothesis,

- $\Sigma; \Delta; \Gamma \rhd^M_{\mathscr{C}} \mathbf{e}'[\mathbf{v}/\mathbf{x}] : \forall\alpha.\, \tau_\mathbf{b}$ as well.

Then, by RTC-TApp,

- $\Sigma; \Delta; \Gamma \rhd^M_{\mathscr{C}} \mathbf{e}'[\mathbf{v}/\mathbf{x}][\tau_\mathbf{a}] : \tau_\mathbf{b}[\tau_\mathbf{a}/\alpha]$.

By the Barendregt condition, $\alpha \notin \mathrm{FTV}(\mathbf{v})$, and therefore $\mathbf{e}'[\mathbf{v}/\mathbf{x}][\tau_\mathbf{a}] = \mathbf{e}'[\tau_\mathbf{a}][\mathbf{v}/\mathbf{x}]$. If $\mathbf{e}'[\tau_\mathbf{a}]$ is worthy, then $\mathbf{e}'$ is worthy as well, since it has the same free locations. By induction, then $\mathbf{e}'[\mathbf{v}/\mathbf{x}]$ is worthy, and thus $\mathbf{e}'[\tau_\mathbf{a}][\mathbf{v}/\mathbf{x}]$ is worthy as well, since it has the same free locations as $\mathbf{e}'[\mathbf{v}/\mathbf{x}]$.

Case $\mathbf{e_1}\ \mathbf{e_2}$.

By inverting RTC-App, we have that

- $\Sigma_{11}; \Delta; \Gamma, \mathbf{x}{:}\tau_\mathbf{x} \rhd^M_{\mathscr{C}} \mathbf{e_1} : \tau' \to \tau$ and
- $\Sigma_{12}; \Delta; \Gamma, \mathbf{x}{:}\tau_\mathbf{x} \rhd^M_{\mathscr{C}} \mathbf{e_2} : \tau'$ for some τ', where
- $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$.

Let $\Sigma_{21} = \Sigma_2|_u$, that is, Σ_2 restricted so that its image contains no σ types; by Lemma 5.4.8, $\mathbf{v}$ is worthy, thus Σ_{21} is sufficient for typing $\mathbf{v}$.

Furthermore, by Lemma 5.2.4,

- $\Sigma_{21} \boxplus \Sigma_{21} = \Sigma_{21}$ and
- $\Sigma_2 \boxplus \Sigma_{21} = \Sigma_2$.

By induction on both subterms $\mathbf{e_1}$ and $\mathbf{e_2}$,

- $\Sigma_{11} \boxplus \Sigma_{21}; \Delta; \Gamma \rhd^M_{\mathscr{C}} \mathbf{e_1}[\mathbf{v}/\mathbf{x}] : \tau' \to \tau$ and
- $\Sigma_{12} \boxplus \Sigma_{21}; \Delta; \Gamma \rhd^M_{\mathscr{C}} \mathbf{e_2}[\mathbf{v}/\mathbf{x}] : \tau'$.

Then by RTC-App,

- $(\Sigma_{11} \boxplus \Sigma_{21}) \boxplus (\Sigma_{12} \boxplus \Sigma_{21}); \Gamma \rhd^M_{\mathscr{C}} \mathbf{e_1}[\mathbf{v}/\mathbf{x}]\ \mathbf{e_2}[\mathbf{v}/\mathbf{x}] : \tau$.

By associativity and commutativity of $(\boxplus)$,

- $\Sigma_1 \boxplus \Sigma_{21}; \Gamma \rhd^M_{\mathscr{C}} \mathbf{e_1}[\mathbf{v}/\mathbf{x}]\ \mathbf{e_2}[\mathbf{v}/\mathbf{x}] : \tau$,

and by weakening and the definition of substitution,

- $\Sigma_1 \boxplus \Sigma_2; \Delta; \Gamma \rhd^M_{\mathscr{C}} (\mathbf{e_1}\ \mathbf{e_2})[\mathbf{v}/\mathbf{x}] : \tau$.

If $\mathbf{e}$ is worthy, then clearly $\mathbf{e_1}$ and $\mathbf{e_2}$ are. If $\mathbf{e_1}$ is worthy, then by induction, $\mathbf{e_1}[\mathbf{v}/\mathbf{x}]$ is worthy as well; likewise $\mathbf{e_2}[\mathbf{v}/\mathbf{x}]$. Thus, $\mathbf{e}[\mathbf{v}/\mathbf{x}]$ is worthy if $\mathbf{e}$ is.

Case $\mathbf{if0}\,\mathbf{e_1}\ \mathbf{e_2}\ \mathbf{e_3}$.

By inverting RTC-If0, we have that

- $\Sigma_{11}; \Delta; \Gamma, \mathbf{x} : \tau_\mathbf{x} \rhd^M_{\mathscr{C}} \mathbf{e_1} : \mathbf{int}$,
- $\Sigma_{12}; \Delta; \Gamma, \mathbf{x} : \tau_\mathbf{x} \rhd^M_{\mathscr{C}} \mathbf{e_2} : \tau$, and
- $\Sigma_{12}; \Delta; \Gamma, \mathbf{x} : \tau_\mathbf{x} \rhd^M_{\mathscr{C}} \mathbf{e_3} : \tau$, where

- $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$.

Let $\Sigma_{21} = \Sigma_2|_u$, and by Lemma 5.4.8, since $\mathbf{v}$ is worthy, Σ_{21} can type $\mathbf{v}$.

Note that $\Sigma_{21} + \Sigma_{21} = \Sigma_{21}$ and $\Sigma_2 + \Sigma_{21} = \Sigma_2$.

By induction on all three subterms $\mathbf{e_i}$,

- $\Sigma_{11} \boxplus \Sigma_{21}; \Gamma \rhd^{M}_{\mathscr{C}} \mathbf{e_1}[\mathbf{v}/\mathbf{x}] : \mathbf{int}$,
- $\Sigma_{12} \boxplus \Sigma_{21}; \Gamma \rhd^{M}_{\mathscr{C}} \mathbf{e_2}[\mathbf{v}/\mathbf{x}] : \tau$, and
- $\Sigma_{12} \boxplus \Sigma_{21}; \Gamma \rhd^{M}_{\mathscr{C}} \mathbf{e_2}[\mathbf{v}/\mathbf{x}] : \tau$.

Then by RTC-If0,

- $(\Sigma_{11} \boxplus \Sigma_{21}) \boxplus (\Sigma_{12} \boxplus \Sigma_{21}); \Gamma \rhd^{M}_{\mathscr{C}} \mathbf{if0}\, \mathbf{e_1}[\mathbf{v}/\mathbf{x}]\ \mathbf{e_2}[\mathbf{v}/\mathbf{x}]\ \mathbf{e_3}[\mathbf{v}/\mathbf{x}] : \tau$,

By associativity and commutativity of $(\boxplus)$,

- $\Sigma_1 \boxplus \Sigma_{21}; \Gamma_1 \boxplus \Gamma_2 \rhd^{M}_{\mathscr{C}} \mathbf{if0}\, \mathbf{e_1}[\mathbf{v}/\mathbf{x}]\ \mathbf{e_2}[\mathbf{v}/\mathbf{x}]\ \mathbf{e_3}[\mathbf{v}/\mathbf{x}] : \tau$,

and by weakening and the definition of substitution,

- $\Sigma_1 \boxplus \Sigma_2; \Delta; \Gamma_1 \boxplus \Gamma_2 \rhd^{M}_{\mathscr{C}} (\mathbf{if0}\, \mathbf{e_1}\ \mathbf{e_2}\ \mathbf{e_3})[\mathbf{v}/\mathbf{x}] : \tau$.

If $\mathbf{e}$ is worthy, then clearly $\mathbf{e_1}$, $\mathbf{e_2}$ and $\mathbf{e_3}$ are. Then by induction, all of $\mathbf{e_1}[\mathbf{v}/\mathbf{x}]$, $\mathbf{e_2}[\mathbf{v}/\mathbf{x}]$, and $\mathbf{e_3}[\mathbf{v}/\mathbf{x}]$ must be worthy as well, and thus $\mathbf{e}[\mathbf{v}/\mathbf{x}]$ is worthy.

Case $\mathbf{f}$.

As before when $\mathbf{x} \notin \mathrm{FV}(\mathbf{e})$.

Case f.

As before when $\mathbf{x} \notin \mathrm{FV}(\mathbf{e})$

Case ${}_{\mathbf{f}}\mathbf{CA}^{\sigma}_{\mathsf{g}}(\mathsf{e}')$.

By inversion of RTC-Boundary, it must be the case that

- $\Sigma_1; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{e}' : \sigma$.

Thus $\mathbf{e}$ is closed, so as before when $\mathbf{x} \notin \mathrm{FV}(\mathbf{e})$.

If $\mathbf{e}$ is worthy then e' is, as they have the same free locations. Then by induction, $\mathsf{e}'[\mathbf{v}/\mathbf{x}]$ is worthy, and thus so is $\mathbf{e}[\mathbf{v}/\mathbf{x}]$.

Case ${}_{\mathbf{f}}\mathbf{CA}[\ell]^{\sigma}_{\mathsf{g}}(\mathsf{v}')$.

There are three rules that may be at the root of our type derivation:

Case RTC-Blessed.

By inversion, we know that there exists some Σ'_1 such that

- $\Sigma'_1; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{v}' : \sigma$

Thus v' is closed, and $\mathsf{v}'[\mathbf{v}/\mathbf{x}] = \mathsf{v}'$.

Therefore, this case is as before when $\mathbf{x} \notin \mathrm{FV}(\mathbf{e})$

Case RTC-Defunct.

Then $(\sigma)^{\mathscr{C}} = \tau$, and by inversion, we know that

- $\Sigma_1 = [\Sigma_{11}]^\ell, \ell{:}\mathbb{D}, [\Sigma_{12}]^\ell$,
- $\sigma = \sigma^{\mathbf{w}}$, and
- $|\sigma| = \mathsf{a}$.

This is sufficient to prove that

- $\Sigma_1; \Delta; \Gamma_* \triangleright^M_{\mathscr{C}}\ {}_{\mathbf{f}}\mathbf{CA}[\ell]^{\sigma}_{\mathsf{g}}(\mathsf{v}_*) : \tau$

for *any* Γ_* and v_*, including Γ and $\mathsf{v}'[\mathbf{v}/\mathbf{x}]$.

Case RTC-Sealed.

By inversion, we know that there exists some Σ_1' such that

- $\Sigma_1'; \cdot; \cdot \triangleright^M_{\mathscr{A}} \mathsf{v}' : \sigma$.

Thus v' is closed, and $\mathsf{v}'[\mathbf{v}/\mathbf{x}] = \mathsf{v}'$.

Therefore, this case is as before when $\mathbf{x} \notin \mathrm{FV}(\mathbf{e})$

By Lemma 5.4.8, since $\mathbf{e}[\mathbf{v}/\mathbf{x}]$ is a value and has type τ, it is worthy.

(*ii*) The structural cases for e are insufficient due to rules with overlapping conclusions, so in the case of subsumption, we identify the rule at the root of the derivation; when unambiguous among the remaining cases, we identify the subject term at the root.

Case RTA-Subsume.

Then by inversion, we know that there exists some $\sigma_<$ such that

- $\Sigma_1; \Delta; \Gamma, \mathsf{x} : \sigma_{\mathsf{x}} \triangleright^M_{\mathscr{A}} \mathsf{e} : \sigma_<$ and
- $\sigma_< <: \sigma$.

Then by induction,

- $\Sigma; \Delta; \Gamma \triangleright^M_{\mathscr{A}} \mathsf{e}[\mathsf{v}/\mathsf{x}] : \sigma_<$.

Reapplying RTA-Subsume yields our result.

Case $\Lambda\alpha^{\mathsf{q}}.\, \mathsf{v}'$.

By inversion of rule RTA-TLam, we know that

- $\Sigma_1; \Delta, \alpha^{\mathsf{q}}; \Gamma, \mathsf{x} : \sigma_{\mathsf{x}} \triangleright^M_{\mathscr{A}} \mathsf{v}' : \sigma'$ where
- $\sigma = \forall \alpha^{\mathsf{q}}.\, \sigma'$,

and by induction,

- $\Sigma; \Delta, \alpha^{\mathsf{q}}; \Gamma \triangleright^M_{\mathscr{A}} \mathsf{v}'[\mathsf{v}/\mathsf{x}] : \sigma'$.

Then reapplying RTA-TLam, we have that

- $\Sigma; \Delta; \Gamma \triangleright^M_{\mathscr{A}} \Lambda\alpha^{\mathsf{q}}.\mathsf{v}'[\mathsf{v}/\mathsf{x}] : \sigma$.

Furthermore, if $\Lambda\alpha^{\mathsf{q}}.\, \mathsf{v}'$ is worthy with respect to Σ_1 then so is v', since they have the same free locations. By the induction hypothesis, $\mathsf{v}'[\mathsf{v}/\mathsf{x}]$ is worthy with respect to Σ, and thus so is $\Lambda\alpha^{\mathsf{q}}.\mathsf{v}'[\mathsf{v}/\mathsf{x}]$.

Case $\lambda \mathsf{y}{:}\sigma_\mathsf{y}.\, \mathsf{e}'$.

If $\mathsf{x} = \mathsf{y}$ then as before when $\mathsf{x} \notin \mathrm{FV}(\mathsf{e})$.

Otherwise, $\mathsf{x} \neq \mathsf{y}$. By inversion of rule RTA-Lam, we know that

- $\Sigma_1; \Delta; \Gamma, \mathsf{x}{:}\sigma_\mathsf{x}, \mathsf{y}{:}\sigma_\mathsf{y} \rhd^M_{\mathscr{A}} \mathsf{e}' : \sigma_\mathsf{r}$ where
- $\sigma = \sigma_\mathsf{y} \overset{\mathsf{q}}{\multimap} \sigma_\mathsf{r}$,

and by exchange and induction,

- $\Sigma; \Delta; \Gamma, \mathsf{y}{:}\sigma_\mathsf{y} \rhd^M_{\mathscr{A}} \mathsf{e}'[\mathsf{v}/\mathsf{x}] : \sigma_\mathsf{r}$.

By cases on q:

Case u.

Then e' is worthy with respect to Σ_1 and $\Gamma, \mathsf{x}{:}\sigma_\mathsf{x}$, by the same inversion.

This means that either:

Case $\mathsf{x} \notin \mathrm{FV}(\mathsf{e}')$.

Then $\mathsf{e}'[\mathsf{v}/\mathsf{x}] = \mathsf{e}'$, and thus $\mathsf{e}'[\mathsf{v}/\mathsf{x}]$ is worthy;

Case $|\sigma_\mathsf{x}| = \mathsf{u}$.

Then by Lemma 5.4.8, v is worthy, and by induction $\mathsf{e}'[\mathsf{v}/\mathsf{x}]$ is worthy.

Thus, $\mathsf{e}'[\mathsf{v}/\mathsf{x}]$ is worthy.

Reapplying RTA-Lam, we get that

- $\Sigma; \Delta; \Gamma \rhd^M_{\mathscr{A}} \lambda \mathsf{y}{:}\sigma_\mathsf{y}.\, (\mathsf{e}'[\mathsf{v}/\mathsf{x}]) : \sigma_\mathsf{y} \overset{\mathsf{u}}{\multimap} \sigma_\mathsf{r}$.

By the Barendregt condition, $\lambda \mathsf{y}{:}\sigma_\mathsf{y}.(\mathsf{e}'[\mathsf{v}/\mathsf{x}]) = (\lambda \mathsf{y}{:}\sigma_\mathsf{y}.\, \mathsf{e}')[\mathsf{v}/\mathsf{x}]$.

Finally, in this case, by Lemma 5.4.8, $\mathsf{e}[\mathsf{v}/\mathsf{x}]$ is worthy.

Case a.

Then by RTA-Lam, there exists some qualifier q' such that

- $\Sigma; \Delta; \Gamma \rhd^M_{\mathscr{A}} (\lambda \mathsf{y}{:}\sigma_\mathsf{y}.\, \mathsf{e}')[\mathsf{v}/\mathsf{x}] : \sigma_\mathsf{y} \overset{\mathsf{q}'}{\multimap} \sigma_\mathsf{r}$.

Since $\mathsf{q}' \sqsubseteq \mathsf{a}$ for any q', by Derelict,

- $\sigma_\mathsf{y} \overset{\mathsf{q}'}{\multimap} \sigma_\mathsf{r} <: \sigma_\mathsf{y} \overset{\mathsf{a}}{\multimap} \sigma_\mathsf{r}$,

Hence, by RTA-Subsume,

- $\Sigma; \Delta; \Gamma \rhd^M_{\mathscr{A}} \mathsf{e}[\mathsf{v}/\mathsf{x}] : \sigma$.

Case c.

As before when $\mathsf{x} \notin \mathrm{FV}(\mathsf{e})$

Case y.

If $\mathsf{x} \neq \mathsf{y}$, then as before when $\mathsf{x} \notin \mathrm{FV}(\mathsf{e})$

If $\mathsf{x} = \mathsf{y}$, then $\sigma = \sigma_\mathsf{x}$, by RTA-Var.

Since $\mathsf{x}[\mathsf{v}/\mathsf{x}] = \mathsf{v}$, we thus have that

- $\Sigma_2; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{x}[\mathsf{v}/\mathsf{x}] : \sigma$.

By our weakening observation,

- $\Sigma_1 \boxplus \Sigma_2; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{x}[\mathsf{v}/\mathsf{x}] : \sigma$.

If v is worthy then of course $\mathsf{e}[\mathsf{v}/\mathsf{x}]$ is as well.

Case $\mathsf{e}'[\sigma_{\mathsf{a}}]$.

By inverting RTA-TApp, we have that

- $\Sigma_1; \Delta; \Gamma, \mathsf{x} : \sigma_{\mathsf{x}} \rhd^{M}_{\mathscr{A}} \mathsf{e}' : \forall \alpha^{\mathsf{q}}. \, \sigma_{\mathsf{b}}$ where
- $\sigma = \sigma_{\mathsf{b}}[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q}}]$ and
- $|\sigma_{\mathsf{a}}| \sqsubseteq \alpha^{\mathsf{q}}$.

By the induction hypothesis,

- $\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{e}'[\mathsf{v}/\mathsf{x}] : \forall \alpha^{\mathsf{q}}. \, \sigma_{\mathsf{b}}$.

Then, by RTA-TApp,

- $\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{e}'[\mathsf{v}/\mathsf{x}][\sigma_{\mathsf{a}}] : \sigma_{\mathsf{b}}[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q}}]$.

By the Barendregt condition, $\alpha^{\mathsf{q}} \notin \mathrm{FTV}(\mathsf{v})$, and therefore $\mathsf{e}'[\mathsf{v}/\mathsf{x}][\sigma_{\mathsf{a}}] = \mathsf{e}'[\sigma_{\mathsf{a}}][\mathsf{v}/\mathsf{x}]$.

If $\mathsf{e}'[\sigma_{\mathsf{a}}]$ is worthy, then e' is worthy as well, since it has the same free variables. By induction, then $\mathsf{e}'[\mathsf{v}/\mathsf{x}]$ is worthy, and thus $\mathsf{e}'[\sigma_{\mathsf{a}}][\mathsf{v}/\mathsf{x}]$ is worthy as well, since it has the same free variables as $\mathsf{e}'[\mathsf{v}/\mathsf{x}]$.

Case $\mathsf{e}_1 \; \mathsf{e}_2$.

By inverting RTA-App, there exist some Σ_{11} and Σ_{12} such that

- $\Sigma_{11}; \Delta; \Gamma_1 \rhd^{M}_{\mathscr{A}} \mathsf{e}_1 : \sigma' \xrightarrow{\mathsf{q}}\!\circ\; \sigma$ and
- $\Sigma_{12}; \Delta; \Gamma_2 \rhd^{M}_{\mathscr{A}} \mathsf{e}_2 : \sigma'$ for some σ', where
- $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$.

By the definition of $(\boxplus)$, there are three ways to reach that conclusion:

Case $\dfrac{\Gamma_1' \boxplus \Gamma_2 = \Gamma}{\Gamma_1', \mathsf{x} : \sigma_{\mathsf{x}} \boxplus \Gamma_2 = \Gamma, \mathsf{x} : \sigma_{\mathsf{x}}}$.

In particular, $\mathsf{x} \notin \operatorname{dom} \Gamma_2$, so it must not be free in e_2; thus $\mathsf{e}_2[\mathsf{v}/\mathsf{x}] = \mathsf{e}_2$.

We apply the induction hypothesis only to e_1, yielding

- $\Sigma_{11} \boxplus \Sigma_2; \Delta; \Gamma_1' \rhd^{M}_{\mathscr{A}} \mathsf{e}_1[\mathsf{v}/\mathsf{x}] : \sigma' \xrightarrow{\mathsf{q}}\!\circ\; \sigma$.

Applying RTA-App, we have

- $\Sigma_1 \boxplus \Sigma_2; \Delta; \Gamma_1' \boxplus \Gamma_2 \rhd^{M}_{\mathscr{A}} \mathsf{e}_1[\mathsf{v}/\mathsf{x}] \; \mathsf{e}_2[\mathsf{v}/\mathsf{x}] : \sigma$.

Given that $\Gamma_1' \boxplus \Gamma_2 = \Gamma$ and by the definition of substitution, we have our conclusion. (In this case, $|\sigma_{\mathsf{x}}| = \mathsf{a}$.)

Case $\dfrac{\Gamma_1 \boxplus \Gamma_2' = \Gamma}{\Gamma_1 \boxplus \Gamma_2', \mathsf{x} : \sigma_{\mathsf{x}} = \Gamma, \mathsf{x} : \sigma_{\mathsf{x}}}$.

By symmetry with the previous case, $\mathsf{x} \notin \mathrm{FV}(\mathsf{e}_1)$ and we apply induction only to e_2.

Case $\dfrac{\Gamma'_1 \boxplus \Gamma'_2 = \Gamma}{\Gamma'_1, \mathsf{x} : \sigma_{\mathsf{x}} \boxplus \Gamma'_2, \mathsf{x} : \sigma_{\mathsf{x}} = \Gamma, \mathsf{x} : \sigma_{\mathsf{x}}}$.

In this case, $|\sigma_{\mathsf{x}}| = \mathsf{u}$.

Thus, by Lemma 5.4.8, v is worthy, so if we let $\Sigma_{21} = \Sigma_2|_u$, then $\Sigma_{21} \boxplus \Sigma_{21} = \Sigma_{21}$.

By induction on both e_1 and e_2, with v typing in Σ_{21}, we have that

- $\Sigma_{11} \boxplus \Sigma_{21}; \Delta; \Gamma'_1 \rhd^M_{\mathscr{A}} \mathsf{e}_1[\mathsf{v}/\mathsf{x}] : \sigma' \xrightarrow{\mathsf{q}}\!\!\circ\, \sigma$ and
- $\Sigma_{12} \boxplus \Sigma_{21}; \Delta; \Gamma'_2 \rhd^M_{\mathscr{A}} \mathsf{e}_2[\mathsf{v}/\mathsf{x}] : \sigma'$.

Then apply RTA-App and weakening, yielding

- $\Sigma_1 \boxplus \Sigma_2; \Delta; \Gamma'_1 \boxplus \Gamma'_2 \rhd^M_{\mathscr{A}} \mathsf{e}_1[\mathsf{v}/\mathsf{x}]\ \mathsf{e}_2[\mathsf{v}/\mathsf{x}] : \sigma$.

If e and v are both worthy, then the e_i are worthy too; by induction, both $\mathsf{e}_\mathsf{i}[\mathsf{v}/\mathsf{x}]$ are worthy, and thus $\mathsf{e}[\mathsf{v}/\mathsf{x}]$ is.

Case $\mathsf{if0}\ \mathsf{e}_1\ \mathsf{e}_2\ \mathsf{e}_3$.

As in the RTA-App case above, we invert the RTA-If0 type rule and then consider how the environments might be split. In particular, x may belong only to the environment for e_1, only to the environment for e_2 and e_3, or it may be distributed into both. In any case, we apply induction to the cases where x is free and recognize that substitution for x is identity on the other components, as above.

Likewise, if e is worthy, then by induction on all three subexpressions, it follows that $\mathsf{e}[\mathsf{v}/\mathsf{x}]$ is worthy.

Case f.

As before when $\mathsf{x} \notin \mathrm{FV}(\mathsf{e})$.

Case $\mathbf{f}$.

As before when $\mathsf{x} \notin \mathrm{FV}(\mathsf{e})$

Case ℓ.

As before when $\mathsf{x} \notin \mathrm{FV}(\mathsf{e})$

Case $\langle \mathsf{e}_1, \mathsf{e}_2 \rangle$.

As in the RTA-App case above.

Case $\mathsf{let}\ \langle \mathsf{y}, \mathsf{z} \rangle = \mathsf{e}_1\ \mathsf{in}\ \mathsf{e}_2$.

As in the RTA-App case above.

Case ${}^{\sigma}_{\mathsf{f}}\mathsf{AC}_{\mathbf{g}}(\mathbf{e}')$.

By inverting RTA-Boundary, it must be the case that

- $\Sigma_1; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{e}' : (\sigma)^{\mathscr{C}}$.

Thus, e is closed, so as before when $\mathsf{x} \notin \mathrm{FV}(\mathsf{e})$.

If e is worthy then $\mathbf{e}'$ is, as they have the same free locations. Then by induction, $\mathbf{e}'[\mathsf{v}/\mathsf{x}]$ is worthy, and thus so is $\mathsf{e}[\mathsf{v}/\mathsf{x}]$.

Case ${}^{\sigma}_{\mathsf{f}}\mathsf{AC}[\,]_{\mathbf{g}}(\mathbf{v}')$.

As for ${}^{\sigma}_{\mathsf{f}}\mathsf{AC}_{\mathbf{g}}(\mathbf{e}')$. □

5.5 Preservation

Observation 5.5.1 (Classification of types)**.** Consider the various syntactic categories of types:

τ	$\tau^{\mathbf{w}}$	$\tau^{\mathbf{o}}$
$\mathbf{int}$		
$\tau_1 \rightarrow \tau_2$	●	
$\forall\alpha.\,\tau$	●	
α	●	●
$\{\sigma\}$		

σ	σ^{w}	σ^{o}
int		
$\sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2$	●	
$\forall\alpha^{\mathsf{q}}.\,\sigma$	●	
α^{q}	●	●
σ ref	●	●
$\sigma_1 \otimes \sigma_2$	●	●
$\{\tau\}$		

Thus,

(*i*) For any type τ, if

- $\tau \neq \mathbf{int}$ and
- there is no σ such that $\tau = \{\sigma\}$,

then τ is a wrappable type of the form $\tau^{\mathbf{w}}$.

(*ii*) For any type σ, if

- $\sigma \neq \mathsf{int}$ and
- there is no τ such that $\sigma = \{\tau\}$,

then σ is a wrappable type of the form σ^{w}.

Changing the type of a location from $\mathbb{B}$ to $\mathbb{D}$ in a store context Σ does not break the typing of an expression using Σ. Furthermore, changing the value in a location in the store from **BLSSD** to **DFNCT** does not change the typing of the store, *except* that it updates the type associated with that location in the store context. To be precise:

Lemma 5.5.2 (Going defunct)**.**

(*i*) *If* $\Sigma_1, \ell{:}\mathbb{B}; \Delta; \cdot \rhd^{M}_{\mathscr{C}} \mathbf{e} : \tau$ *then* $\Sigma_1, \ell{:}\mathbb{D}; \Delta; \cdot \rhd^{M}_{\mathscr{C}} \mathbf{e} : \tau$.

(*ii*) *If* $\Sigma_1, \ell{:}\mathbb{B}; \Delta; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{e} : \sigma$ *then* $\Sigma_1, \ell{:}\mathbb{D}; \Delta; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{e} : \sigma$.

(*iii*) *If* $\Sigma_1, [\Sigma_2]^{\ell}, \ell{:}\,\mathbb{D}; \Delta; \cdot \rhd^{M}_{\mathscr{C}} \mathbf{e} : \tau$ *then* $\Sigma_1, \Sigma_2|_u, \ell{:}\,\mathbb{D}; \Delta; \cdot \rhd^{M}_{\mathscr{C}} \mathbf{e} : \tau$

(*iv*) *If* $\Sigma_1, [\Sigma_2]^\ell, \ell{:}\mathbb{D}; \Delta; \cdot \rhd^M_{\mathscr{A}} \mathsf{e} : \sigma$ *then* $\Sigma_1, \Sigma_2|_u, \ell{:}\mathbb{D}; \Delta; \cdot \rhd^M_{\mathscr{A}} \mathsf{e} : \sigma$

(*v*) *If* $\Sigma_1, [\Sigma']^\ell, \ell{:}\mathbb{B} \rhd^M s \uplus \{\ell \mapsto \mathbf{BLSSD}\} : \Sigma_2, [\Sigma']^\ell, \ell{:}\mathbb{B}$,
then $\Sigma_1, \Sigma'|_u, \ell{:}\mathbb{D} \rhd^M s \uplus \{\ell \mapsto \mathbf{DFNCT}\} : \Sigma_2, \Sigma', \ell{:}\mathbb{D}$.

Proof.

(*i*) Observe that there are only two rules that mention store context bindings of the form $\ell{:}\tau'$:

RTC-Blessed Then the subterm types in the new store context by RTC-Defunct.

RTC-Defunct Vacuous, as it requires that $\ell{:}\mathbb{D}$, which contradicts the assumption.

Thus, we can construct a new derivation.

(*ii*) Likewise.

(*iii*) By induction on the length of Σ_2. The only rule that makes use of a protected binding like $\ell'{:}[\sigma]^\ell$ is RTC-Blessed. But since $\ell{:}\mathbb{D}$, that rule never applies. Thus, such a binding for ℓ' is irrelevant to the typing. The remaining bindings are present in $\Sigma_2|_u$.

(*iv*) Likewise.

(*v*) Inverting S-CLoc,

$$\frac{\dfrac{\mathcal{A}}{\Sigma_1, [\Sigma']^\ell, \ell{:}\mathbb{B} \rhd^M s : \Sigma_2, [\Sigma']^\ell} \qquad \overline{\Sigma_1|_u, [\Sigma']^\ell, \ell{:}\mathbb{B}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{BLSSD} : \mathbb{B}}}{\Sigma_1, [\Sigma']^\ell, \ell{:}\mathbb{B} \rhd^M s \uplus \{\ell \mapsto \mathbf{BLSSD}\} : \Sigma_2, [\Sigma']^\ell, \ell{:}\mathbb{B}}$$

It suffices to prove $\mathcal{B}$, which allows us to construct a derivation for the desired result:

$$\frac{\dfrac{\mathcal{B}}{\Sigma_1, \Sigma'|_u, \ell{:}\mathbb{D} \rhd^M s : \Sigma_2, \Sigma'} \qquad \overline{\Sigma_1|_u, \Sigma'|_u, \ell{:}\mathbb{D}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{DFNCT} : \mathbb{D}}}{\Sigma_1, \Sigma'|_u, \ell{:}\mathbb{D} \rhd^M s \uplus \{\ell \mapsto \mathbf{DFNCT}\}) : \Sigma_2, \Sigma', \ell{:}\mathbb{D}}$$

We proceed to prove $\mathcal{B}$ by induction on the structure of Σ';

Case $\cdot$.

Then $\mathcal{A}$ gives us that $\Sigma_1, \ell{:}\mathbb{B} \rhd^M s : \Sigma_1 \boxplus \Sigma_2$.

Case $\Sigma'', \ell'{:}\tau$.

From $\mathcal{A}$ and inversion of rule S-CLoc,

$$\frac{\dfrac{\mathcal{D}}{\Sigma_{11}, [\Sigma'']^\ell, \ell'{:}\tau, \ell{:}\mathbb{B} \rhd^M s' : \Sigma_2, [\Sigma'']^\ell} \qquad \dfrac{\mathcal{C}}{\Sigma_{12}, [\Sigma'']^\ell, \ell'{:}\tau, \ell{:}\mathbb{B}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{v} : \tau}}{(\Sigma_{11} \boxplus \Sigma_{12}), [\Sigma'']^\ell, \ell'{:}\tau, \ell{:}\mathbb{B} \rhd^M s' \uplus \{\ell' \mapsto \mathbf{v}\} : \Sigma_2, [\Sigma'']^\ell, \ell'{:}\tau}$$

for some $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$. Then by S-CLoc,

$$\dfrac{\dfrac{\mathcal{D}, \text{IH at } \Sigma''}{\Sigma_{11}, \Sigma''|_u, \ell'{:}\tau, \ell{:}\mathbb{D} \rhd^M s' : \Sigma_2, \Sigma''} \qquad \dfrac{\mathcal{C}, \text{parts } (i) \text{ and } (iii)}{\Sigma_{12}, \Sigma''|_u, \ell'{:}\tau, \ell{:}\mathbb{D}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{v} : \tau}}{(\Sigma_{11} \boxplus \Sigma_{12}), \Sigma''|_u, \ell'{:}\tau, \ell{:}\mathbb{D} \rhd^M s' \uplus \{\ell' \mapsto \mathbf{v}\} : \Sigma_2, \Sigma'', \ell'{:}\tau}.$$

Case $\Sigma'', \ell'{:}[\sigma]^{\ell''}$.

From $\mathcal{A}$ and inversion of rule S-ALocProt,

$$\dfrac{\dfrac{\mathcal{D}}{\Sigma_{11}, [\Sigma'']^{\ell}, \ell'{:}[\sigma]^{\ell''}, \ell{:}\mathbb{B} \rhd^M s' : \Sigma_2, [\Sigma'']^{\ell}} \qquad \dfrac{\mathcal{C}}{\Sigma_{12}, [\Sigma'']^{\ell}, \ell'{:}[\sigma]^{\ell''}, \ell{:}\mathbb{B}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{v} : \sigma}}{(\Sigma_{11} \boxplus \Sigma_{12}), [\Sigma'']^{\ell}, \ell'{:}[\sigma]^{\ell''}, \ell{:}\mathbb{B} \rhd^M s' \uplus \{\ell' \mapsto \mathsf{v}\} : \Sigma_2, [\Sigma'']^{\ell}, \ell'{:}[\sigma]^{\ell''}}$$

for some $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$. Then by S-ALocProt,

$$\dfrac{\dfrac{\mathcal{D}, \text{IH at } \Sigma''}{\Sigma_{11}, \Sigma''|_u, \ell'{:}[\sigma]^{\ell''}, \ell{:}\mathbb{D} \rhd^M s' : \Sigma_2, \Sigma''} \qquad \dfrac{\mathcal{C}, \text{parts } (ii) \text{ and } (iv)}{\Sigma_{12}, \Sigma''|_u, \ell'{:}[\sigma]^{\ell''}, \ell{:}\mathbb{D}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{v} : \sigma}}{(\Sigma_{11} \boxplus \Sigma_{12}), \Sigma''|_u, \ell'{:}[\sigma]^{\ell''}, \ell{:}\mathbb{D} \rhd^M s' \uplus \{\ell' \mapsto \mathsf{v}\} : \Sigma_2, \Sigma'', \ell'{:}[\sigma]^{\ell''}}.$$

Case $\Sigma'', \ell'{:}\sigma$.

From $\mathcal{A}$ and inversion of rule S-ALoc,

$$\dfrac{\dfrac{\mathcal{D}}{\Sigma_{11}, [\Sigma'']^{\ell}, \ell'{:}[\sigma]^{\ell}, \ell{:}\mathbb{B} \rhd^M s' : \Sigma_2, [\Sigma'']^{\ell}} \qquad \dfrac{\mathcal{C}}{\Sigma_{12}, [\Sigma'']^{\ell}, \ell'{:}[\sigma]^{\ell}, \ell{:}\mathbb{B}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{v} : \sigma}}{(\Sigma_{11} \boxplus \Sigma_{12}), [\Sigma'']^{\ell}, \ell'{:}[\sigma]^{\ell}, \ell{:}\mathbb{B} \rhd^M s' \uplus \{\ell' \mapsto \mathsf{v}\} : \Sigma_2, [\Sigma'']^{\ell}, \ell'{:}[\sigma]^{\ell}}$$

for some $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$. Then by S-ALoc,

$$\dfrac{\dfrac{\mathcal{D}, \text{IH at } \Sigma''}{\Sigma_{11}, \Sigma''|_u, \ell{:}\mathbb{D} \rhd^M s' : \Sigma_2, \Sigma''} \qquad \dfrac{\mathcal{C}, \text{parts } (ii) \text{ and } (iv)}{\Sigma_{12}, \Sigma''|_u, \ell{:}\mathbb{D}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{v} : \sigma}}{(\Sigma_{11} \boxplus \Sigma_{12}), \Sigma''|_u, \ell{:}\mathbb{D} \rhd^M s' \uplus \{\ell' \mapsto \mathsf{v}\} : (\Sigma_1 \boxplus \Sigma_2), \Sigma'', \ell'{:}\sigma}. \qquad \square$$

Theorem 5.5.3 (Preservation). *If* $\rhd^M C : \tau$ *and* $C \longmapsto_M C'$ *then* $\rhd^M C' : \tau$.

Proof. We proceed by cases on the reduction relation ($\longmapsto_M$):

Case $(s, \mathbf{E}[\mathbf{e}]_{\mathscr{C}}) \longmapsto_M (s', \mathbf{E}[\mathbf{e}']_{\mathscr{C}})$ if $(s, \mathbf{e}) \longmapsto_M (s', \mathbf{e}')$.

By inversion on Conf, we know that

(i) $\vdash^M m$ okay for every module m in M,

(ii) $\Sigma_1 \rhd^M s : \Sigma_1 \boxplus \Sigma_2$, and

(iii) $\Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}[\mathbf{e}]_{\mathscr{C}} : \tau$.

By Lemma 5.4.2, there exist some τ' and $\Sigma_{21} \boxplus \Sigma_{22} = \Sigma_2$ such that

- $\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{e} : \tau'$, and
- $\Sigma_{22}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}[\mathbf{e}'']_{\mathscr{C}} : \tau$ for all $\mathbf{e}''$ such that $\cdot; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{e}'' : \tau'$.

Without loss of generality, we assume that $(s, \mathbf{e}) \longmapsto_M (s', \mathbf{e}')$ by some rule other than C-CXT or C-CXTA: If the former, then $\mathbf{e} = \mathbf{E}'[\mathbf{e_1}]_{\mathscr{C}}$ and $\mathbf{e}' = \mathbf{E}'[\mathbf{e}'_\mathbf{1}]_{\mathscr{C}}$, so we consider the context $\mathbf{E}[\mathbf{E}']_{\mathscr{C}}$ with $\mathbf{e_1}$ and $\mathbf{e}'_\mathbf{1}$ in the hole instead. If the latter, then $\mathbf{e} = \mathbf{E}'[\mathsf{e}_1]_{\mathscr{A}}$ and $\mathbf{e}' = \mathbf{E}'[\mathsf{e}'_1]_{\mathscr{A}}$, so we consider the context $\mathbf{E}[\mathbf{E}']_{\mathscr{C}}$ with e_1 and e'_1 in the hole as an instance of C-CXT instead.

In cases where $s = s'$, it is sufficient to show that $\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{e}' : \tau'$. By Lemma 5.4.3, we have that $\Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}[\mathbf{e}']_{\mathscr{C}} : \tau'$, and by rule CONF, we have the desired result.

In cases where $s \neq s'$, we will need to rebuild the configuration typing using the new store.

Now, by cases on $(s, \mathbf{e}) \longmapsto_M (s', \mathbf{e}')$:

Case $(s, \mathbf{c}\ \mathbf{v}) \longmapsto_M \delta_{\mathscr{C}}(s, \mathbf{c}, \mathbf{v})$.

Metafunction $\delta_{\mathscr{C}}$ is defined in only two cases:

Case $\delta_{\mathscr{C}}(s, -, \lceil z \rceil) = (s, (z-))$.

Since $\mathrm{ty}_{\mathscr{C}}(-) = \mathbf{int} \to \mathbf{int} \to \mathbf{int}$ and $\lceil z \rceil$ has type $\mathbf{int}$, we know that $\tau' = \mathbf{int} \to \mathbf{int}$, which is also the type of $(z-)$.

Case $\delta_{\mathscr{C}}(s, (z_1-), \lceil z_2 \rceil) = (s, \lceil z_1 - z_2 \rceil)$.

Since $\mathrm{ty}_{\mathscr{C}}((z_1-)) = \mathbf{int} \to \mathbf{int}$ and $\lceil z_2 \rceil$ has type $\mathbf{int}$, we know that $\tau' = \mathbf{int}$, which is also the type of $\lceil z_1 - z_2 \rceil$.

Since $s' = s$ in both cases, it is sufficient to show that τ' is preserved.

Case $(s, (\mathbf{\Lambda}\alpha.\, \mathbf{v})[\tau_\mathbf{a}]) \longmapsto_M (s, \mathbf{v}[\tau_\mathbf{a}/\alpha])$.

By inversion of RTC-TAPP, we know that

- $\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{\Lambda}\alpha.\, \mathbf{v} : \forall\alpha.\, \tau_\mathbf{b}$ and
- $\cdot \vdash_{\mathscr{C}} \tau_\mathbf{a}$, where
- $\tau' = \tau_\mathbf{b}[\tau_\mathbf{a}/\alpha]$.

Then, by inversion of RTC-TLAM, we know that

- $\Sigma_{21}; \cdot, \alpha; \cdot \rhd^M_{\mathscr{C}} \mathbf{v} : \tau_\mathbf{b}$.

By (ii) and Lemma 5.2.7, $\mathrm{FTV}(\Sigma_{21}) = \varnothing$, and $\alpha \notin \mathrm{FTV}(\cdot)$, so by Lemma 5.4.6, we then conclude that $\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{v}[\tau_\mathbf{a}/\alpha] : \tau_\mathbf{b}[\tau_\mathbf{a}/\alpha]$.

Case $(s, (\lambda\mathbf{x}{:}\tau_\mathbf{x}.\, \mathbf{e})\ \mathbf{v}) \longmapsto_M (s, \mathbf{e}[\mathbf{v}/\mathbf{x}])$.

By inversion of RTC-APP, we know that there exist some Σ_{211} and Σ_{212} such that

- $\Sigma_{211}; \cdot; \cdot \rhd^M_{\mathscr{C}} \lambda\mathbf{x}{:}\tau_\mathbf{x}.\, \mathbf{e} : \tau_\mathbf{x} \to \tau'$ and
- $\Sigma_{212}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{v} : \tau_\mathbf{x}$, where
- $\Sigma_{211} \boxplus \Sigma_{212} = \Sigma_{21}$.

Then, by inversion of RTC-LAM on the former, we know that

- $\Sigma_{211}; \cdot; \cdot, \mathbf{x} : \tau_\mathbf{x} \rhd^M_{\mathscr{C}} \mathbf{e} : \tau'$.

By Lemma 5.4.9, we have that $\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{e}[\mathbf{v}/\mathbf{x}] : \tau'$ as well.

Case $(s, \mathbf{if0}\ \lceil 0 \rceil\ \mathbf{e_t}\ \mathbf{e_f}) \longmapsto_M (s, \mathbf{e_t})$.

By inversion on RTC-If0, we know that

- $\Sigma_{212}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{e_t} : \tau'$, where
- $\Sigma_{211} \boxplus \Sigma_{212} = \Sigma_{21}$.

By weakening, $\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{e_t} : \tau'$.

Case $(s, \mathbf{if0}\ \lceil z \rceil\ \mathbf{e_t}\ \mathbf{e_f}) \longmapsto_M (s, \mathbf{e_f})$ $(z \neq 0)$.

By symmetry.

Case $(s, \mathbf{f}) \longmapsto_M (s, \mathbf{v})$ $(\mathbf{module}\ \mathbf{f} : \tau'' = \mathbf{v} \in M)$.

By inversion of RTC-Mod, τ'' must equal τ'.

Furthermore, premiss (i) from the inversion of Conf above tells us that $\vdash^M m$ okay for every module m in M, and for $\mathbf{module}\ \mathbf{f} : \tau' = \mathbf{v}$ in particular. This judgment can only be the conclusion of rule TM-C, from which inversion tells us that

- $\cdot; \cdot \vdash^M_{\mathscr{C}} \mathbf{v} : \tau'$.

By Lemma 5.3.1,

- $\cdot; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{v} : \tau'$

By weakening, $\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{v} : \tau'$.

Case $(s, \mathsf{f}^{\mathbf{g}}) \longmapsto_M (s, {}_{\mathbf{g}}\mathbf{CA}^{\sigma}_{\mathsf{f}}(\mathsf{f}))$ $(\mathsf{module}\ \mathsf{f} : \sigma = \mathsf{v} \in M)$.

By inversion of RTC-ModA, $\tau' = (\sigma)^{\mathscr{C}}$ and $\cdot \vdash_{\mathscr{A}} \sigma$.

Then by RTA-Mod and RTC-Boundary,

$$\dfrac{\dfrac{\mathsf{module}\ \mathsf{f} : \sigma = \mathsf{v} \in M \qquad \cdot \vdash_{\mathscr{A}} \sigma}{\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{f} : \sigma}}{\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{C}} \underset{\mathbf{g}\ \mathsf{f}}{\mathbf{CA}}{}^{\sigma}(\mathsf{f}) : (\sigma)^{\mathscr{C}}}.$$

Case $(s, {}_{\mathbf{f}}\mathbf{CA}^{\sigma}_{\mathsf{g}}(\mathsf{v})) \longmapsto_M \mathit{coerce}_{\mathscr{C}}(s, \sigma, \mathsf{v}, \mathbf{f}, \mathsf{g})$.

There are three possibilities:

- If $\mathsf{v} = \lceil z \rceil$ then $(s, \mathbf{e}) \longmapsto_M (s, \lceil z \rceil)$.

  The only rule to type $\mathbf{e}$ is RTC-Boundary, which gives it the type $(\mathsf{int})^{\mathscr{C}}$, which equals $\mathbf{int}$.

  The only rule to type $\mathbf{e'}$ is RTC-Con, which gives $\mathrm{ty}_{\mathscr{C}}(\lceil z \rceil) = \mathbf{int}$ as well.

- If $\mathsf{v} = {}^{(\tau^{\mathbf{o}})^{\mathscr{A}}}_{\mathbf{g'}}\mathsf{AC}[\,]_{\mathbf{f'}}(\mathbf{v'})$ then $(s, \mathbf{e}) \longmapsto_M (s, \mathbf{v'})$.

  We know there must be a derivation

$$\dfrac{\dfrac{\dfrac{\mathcal{A}}{\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{v} : ((\tau^{\mathbf{o}})^{\mathscr{A}})^{\mathscr{C}}}}{\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{A}} {}^{(\tau^{\mathbf{o}})^{\mathscr{A}}}\underset{\mathbf{g}'\ \mathbf{f}'}{\mathsf{AC}}[\,](\mathbf{v}) : (\tau^{\mathbf{o}})^{\mathscr{A}}}}{\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{C}} \underset{\mathbf{f}\ \mathbf{g}}{\mathbf{CA}}{}^{\sigma}\left({}^{(\tau^{\mathbf{o}})^{\mathscr{A}}}\underset{\mathbf{g}'\ \mathbf{f}'}{\mathsf{AC}}[\,](\mathbf{v})\right) : \tau^{\mathbf{o}}},$$

where $\tau' = \tau^{\mathbf{o}} = ((\tau^{\mathbf{o}})^{\mathscr{A}})^{\mathscr{C}}$. Then $\mathcal{A}$ suffices.

- Otherwise, $(s, \mathbf{e}) \longmapsto_M (s \uplus \{\ell \mapsto \mathbf{BLSSD}\}, {}_{\mathbf{f}}\mathbf{CA}[\ell]^{\sigma}_{\mathsf{g}}(\mathsf{v}))$.

  Furthermore, since the previous two cases covered int and $\{\tau^{\mathbf{o}}\}$, by Observation 5.5.1, we may let $\sigma^{\mathsf{w}} = \sigma$.

  On the left, RTC-Boundary gives us that

  – $\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{C}} {}_{\mathbf{f}}\mathbf{CA}^{\sigma^{\mathsf{w}}}_{\mathsf{g}}(\mathsf{v}) : (\sigma^{\mathsf{w}})^{\mathscr{C}}$, where

  – $\tau' = (\sigma^{\mathsf{w}})^{\mathscr{C}}$.

  By inversion, it must be the case that

  – $\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{v} : \sigma^{\mathsf{w}}$.

  Now by cases on $|\sigma^{\mathsf{w}}|$:

  Case u.

  By weakening and RTC-Sealed,

$$\dfrac{\Sigma_{21}, \ell{:}\,\mathbb{B}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{v} : \sigma^{\mathsf{w}} \qquad |\sigma^{\mathsf{w}}| = \mathsf{u}}{\Sigma_{21}, \ell{:}\,\mathbb{B}; \cdot; \cdot \rhd^M_{\mathscr{C}} \underset{\mathbf{f}\ \mathsf{g}}{\mathbf{CA}}[\ell]^{\sigma^{\mathsf{w}}}(\mathsf{v}) : (\sigma^{\mathsf{w}})^{\mathscr{C}}}.$$

  Furthermore, the new store types by rule S-CLoc.

  Case a.

  By RTC-Blessed,

$$\dfrac{\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{v} : \sigma^{\mathsf{w}} \qquad |\sigma^{\mathsf{w}}| = \mathsf{a}}{[\Sigma_{21}]^{\ell}, \ell : \mathbb{B}; \cdot; \cdot \rhd^M_{\mathscr{C}} \underset{\mathbf{f}\ \mathsf{g}}{\mathbf{CA}}[\ell]^{\sigma^{\mathsf{w}}}(\mathsf{v}) : (\sigma^{\mathsf{w}})^{\mathscr{C}}}.$$

  Consider decomposing $[\Sigma_{21}]^{\ell}$ as

  – $[\Sigma_{21}]^{\ell} = \Sigma_{21}|_u, [\Sigma_{21}|_a]^{\ell}$.

  Since $\Sigma_{21} \sim_u \Sigma_{22}$, we see that $[\Sigma_{21}|_a]^{\ell}, \Sigma_{22}$ is well-formed.

  Furthermore, since $\{\ell \mapsto \mathbf{BLSSD}\}$ is disjoint from s, we know that

  – $\ell \notin \operatorname{dom} \Sigma_2$, so

  – $[\Sigma_{21}|_a]^{\ell}, \Sigma_{22}, \ell{:}\,\mathbb{B}$ is well-formed. (Call this store context Σ_2'.)

  Recall that $\Sigma_{22}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}[\mathbf{e}'']_{\mathscr{C}} : \tau$, which we can weaken to

  – $\Sigma_2'; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}[\mathbf{e}'']_{\mathscr{C}} : \tau$.

Note that $[\Sigma_{21}]^{\ell}, \ell{:}\mathbb{B} = \Sigma_2'|_u$.

Thus,

– $\Sigma_2' \boxplus ([\Sigma_{21}]^{\ell}, \ell{:}\mathbb{B}) = \Sigma_2'$.

and by Lemma 5.4.3,

– $\Sigma_2'; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}[\mathbf{e}']_{\mathscr{C}} : \tau$.

It is now sufficient to show that $\Sigma_1' \rhd^M s' : \Sigma_1' \boxplus \Sigma_2'$ for some Σ_1'.

Let $\Sigma_1' = \Sigma_1, \ell{:}\mathbb{B}$. Since ℓ is fresh, Σ_1' is well-formed.

$$\begin{array}{lll}
& \Sigma_1 \rhd^M s : \Sigma_1 \boxplus \Sigma_2 & (ii) \\
\Rightarrow & \Sigma_1 \rhd^M s \uplus \{\ell \mapsto \mathbf{BLSSD}\} : (\Sigma_1 \boxplus \Sigma_2), \ell{:}\mathbb{B} & \text{rule S-CLoc} \\
\Leftrightarrow & \Sigma_1 \rhd^M s' : \Sigma_1' \boxplus (\Sigma_2, \ell{:}\mathbb{B}) & \text{defs. of } s' \text{ and } \Sigma_1' \\
\Leftrightarrow & \Sigma_1 \rhd^M s' : \Sigma_1' \boxplus (\Sigma_{21}|_a, \Sigma_{22}, \ell{:}\mathbb{B}) & \text{algebra} \\
\Leftrightarrow & \Sigma_1 \rhd^M s' : \Sigma_1' \boxplus ([\Sigma_{21}|_a]^{\ell}, \Sigma_{22}, \ell{:}\mathbb{B}) & \text{lem. 5.2.5} \\
\Leftrightarrow & \Sigma_1 \rhd^M s' : \Sigma_1' \boxplus \Sigma_2' & \text{def. } \Sigma_2' \\
\Rightarrow & \Sigma_1' \rhd^M s' : \Sigma_1' \boxplus \Sigma_2' & \text{weakening.}
\end{array}$$

Case $(s, {}_{\mathbf{f}}\mathbf{CA}[\ell]^{\forall\alpha^{\mathsf{q}}.\,\sigma}_{\mathbf{g}}(\mathsf{v})[\tau_{\mathbf{a}}]) \longmapsto_M \mathit{check}(s, \ell, |\forall\alpha^{\mathsf{q}}.\,\sigma|, {}_{\mathbf{f}}\mathbf{CA}^{\sigma[(\tau_{\mathbf{a}})^{\mathscr{A}}/\alpha^{\mathsf{q}}]}_{\mathbf{g}}(\mathsf{v}[(\tau_{\mathbf{a}})^{\mathscr{A}}]), \mathbf{blame\,f})$.

There are three possibilities:

Case $|\forall\alpha^{\mathsf{q}}.\,\sigma| = \mathsf{u}$.

Then $(s, \mathbf{e}) \longmapsto_M (s, {}_{\mathbf{f}}\mathbf{CA}^{\sigma[(\tau_{\mathbf{a}})^{\mathscr{A}}/\alpha^{\mathsf{q}}]}_{\mathbf{g}}(\mathsf{v}[(\tau_{\mathbf{a}})^{\mathscr{A}}]))$.

We know there must be a derivation of the form

$$\dfrac{\dfrac{\dfrac{\mathcal{A}}{\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{v} : \forall\alpha^{\mathsf{q}}.\,\sigma} \qquad \dfrac{}{|\forall\alpha^{\mathsf{q}}.\,\sigma| = \mathsf{u}}}{\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{C}} \underset{\mathbf{f}\ \mathbf{g}}{\mathbf{CA}}[\ell]^{\forall\alpha^{\mathsf{q}}.\,\sigma}(\mathsf{v}) : \forall\beta.\,(\sigma[\{\beta\}/\alpha^{\mathsf{q}}])^{\mathscr{C}}} \qquad \dfrac{\mathcal{B}}{\cdot \vdash_{\mathscr{C}} \tau_{\mathbf{a}}}}{\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{C}} \underset{\mathbf{f}\ \mathbf{g}}{\mathbf{CA}}[\ell]^{\forall\alpha^{\mathsf{q}}.\,\sigma}(\mathsf{v})[\tau_{\mathbf{a}}] : (\sigma[(\tau_{\mathbf{a}})^{\mathscr{A}}/\alpha^{\mathsf{q}}])^{\mathscr{C}}},$$

where $\Sigma_{211}, \ell{:}\tau_{\mathbf{a}}, \Sigma_{212} = \Sigma_{21}$.

Then we can thus construct a derivation:

$$\dfrac{\dfrac{\dfrac{\mathcal{A}}{\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{v} : \forall\alpha^{\mathsf{u}}.\,\sigma} \qquad \dfrac{\mathcal{B}, \text{def. } (-)^{\mathscr{A}}}{\cdot \vdash_{\mathscr{A}} (\tau_{\mathbf{a}})^{\mathscr{A}}}}{\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{v}[(\tau_{\mathbf{a}})^{\mathscr{A}}] : \sigma[(\tau_{\mathbf{a}})^{\mathscr{A}}/\alpha^{\mathsf{q}}]}}{\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{C}} \underset{\mathbf{f}\ \mathbf{g}}{\mathbf{CA}}^{\sigma[(\tau_{\mathbf{a}})^{\mathscr{A}}/\alpha^{\mathsf{q}}]}(\mathsf{v}[(\tau_{\mathbf{a}})^{\mathscr{A}}]) : (\sigma[(\tau_{\mathbf{a}})^{\mathscr{A}}/\alpha^{\mathsf{q}}])^{\mathscr{C}}}.$$

Case $s = s'' \uplus \{\ell \mapsto \mathbf{BLSSD}\}$ and $|\forall\alpha^{\mathsf{q}}.\,\sigma| = \mathsf{a}$.

Then $(s, \mathbf{e}) \longmapsto_M (s'' \uplus \{\ell \mapsto \mathbf{BLSSD}\}, {}_{\mathbf{f}}\mathbf{CA}^{\sigma[(\tau_{\mathbf{a}})^{\mathscr{A}}/\alpha^{\mathsf{q}}]}_{\mathbf{g}}(\mathsf{v}[(\tau_{\mathbf{a}})^{\mathscr{A}}]))$.

We know there must be a derivation of the form

$$\dfrac{\dfrac{\dfrac{\mathcal{A}}{\Sigma'_{21};\cdot;\cdot \rhd^{M}_{\mathscr{A}} \mathsf{v} : \forall\alpha^{\mathsf{q}}.\,\sigma} \qquad \dfrac{}{|\forall\alpha^{\mathsf{q}}.\,\sigma| = \mathsf{a}}}{[\Sigma'_{21}]^{\ell}, \ell\colon \mathbb{B};\cdot;\cdot \rhd^{M}_{\mathscr{C}} \ \underset{\mathbf{f}\ \mathbf{g}}{\mathbf{CA}}[\ell]^{\forall\alpha^{\mathsf{q}}.\,\sigma}(\mathsf{v}) : \forall\beta.\,(\sigma[\{\beta\}/\alpha^{\mathsf{q}}])^{\mathscr{C}}} \qquad \dfrac{\mathcal{B}}{\cdot \vdash_{\mathscr{C}} \tau_{\mathbf{a}}}}{[\Sigma'_{21}]^{\ell}, \ell\colon \mathbb{B};\cdot;\cdot \rhd^{M}_{\mathscr{C}} \ \underset{\mathbf{f}\ \mathbf{g}}{\mathbf{CA}}[\ell]^{\forall\alpha^{\mathsf{q}}.\,\sigma}(\mathsf{v})[\tau_{\mathbf{a}}] : (\sigma[(\tau_{\mathbf{a}})^{\mathscr{A}}/\alpha^{\mathsf{q}}])^{\mathscr{C}}},$$

where $[\Sigma'_{21}]^{\ell}, \ell\colon \mathbb{B} = \Sigma_{21}$.

We can thus construct a derivation:

$$\dfrac{\dfrac{\dfrac{\mathcal{A}, \text{weakening}}{\Sigma'_{21}, \ell\colon \mathbb{D};\cdot;\cdot \rhd^{M}_{\mathscr{A}} \mathsf{v} : \forall\alpha^{\mathsf{q}}.\,\sigma} \qquad \dfrac{\mathcal{B}, \text{def. } (-)^{\mathscr{A}}}{\cdot \vdash_{\mathscr{A}} (\tau_{\mathbf{a}})^{\mathscr{A}}}}{\Sigma'_{21}, \ell\colon \mathbb{D};\cdot;\cdot \rhd^{M}_{\mathscr{A}} \mathsf{v}[(\tau_{\mathbf{a}})^{\mathscr{A}}] : \sigma[(\tau_{\mathbf{a}})^{\mathscr{A}}/\alpha^{\mathsf{q}}]}}{\Sigma'_{21}, \ell\colon \mathbb{D};\cdot;\cdot \rhd^{M}_{\mathscr{C}} \ \underset{\mathbf{f}\ \mathbf{g}}{\mathbf{CA}}^{\sigma[(\tau_{\mathbf{a}})^{\mathscr{A}}/\alpha^{\mathsf{q}}]}(\mathsf{v}[(\tau_{\mathbf{a}})^{\mathscr{A}}]) : (\sigma[(\tau_{\mathbf{a}})^{\mathscr{A}}/\alpha^{\mathsf{q}}])^{\mathscr{C}}}.$$

Note that we can decompose Σ_2 as

- $\Sigma_2 = \Sigma_{21}, \Sigma_{22}|_a$.

Since $\Sigma_{21} = [\Sigma'_{21}]^{\ell}, \ell\colon \mathbb{B}$, we can decompose Σ_2 further as

- $\Sigma_2 = [\Sigma'_{21}]^{\ell}, \ell\colon \mathbb{B}, \Sigma_{22}|_a$.

Since $\Sigma_2|_a = \Sigma_{22}|_a$ and $\Sigma_2 \sim_u \Sigma_{22}$, we know that

- $\Sigma_2 = \Sigma_{22}$.

Recall that $\Sigma_{22};\cdot;\cdot \rhd^{M}_{\mathscr{C}} \mathbf{E}[\mathbf{e}'']_{\mathscr{C}} : \tau$. By Lemma 5.5.2, we can type $\mathbf{E}[\mathbf{e}'']_{\mathscr{C}}$ with $\Sigma'_{21}|_u, \ell\colon \mathbb{D}, \Sigma_{22}|_a$.

Let $\Sigma'_2 = (\Sigma'_{21}, \ell\colon \mathbb{D}) \boxplus (\Sigma'_{21}|_u, \ell\colon \mathbb{D}, \Sigma_{22}|_a)$, which is clearly well-formed.

Then, by Lemma 5.4.3,

- $\Sigma'_2;\cdot;\cdot \rhd^{M}_{\mathscr{C}} \mathbf{E}[\mathbf{e}']_{\mathscr{C}} : \tau$.

It now suffices to show that $\Sigma'_1 \rhd^{M} s'' : \Sigma'_1 \boxplus \Sigma'_2$ for some Σ'_1. Let $\Sigma'_1 = \Sigma_1|_a, \Sigma'_{21}|_u, \ell\colon \mathbb{D}$. Since ℓ is fresh and $\operatorname{dom}\Sigma_1|_a$ is disjoint from $\operatorname{dom}\Sigma'_{21}$, we know that Σ'_1 is well-formed.

$$\begin{array}{lll}
& \Sigma_1 \rhd^{M} s'' \uplus \{\ell \mapsto \mathbf{BLSSD}\} : \Sigma_1 \boxplus \Sigma_2 & (ii) \\
\Leftrightarrow & \Sigma_1|_a, [\Sigma'_{21}]^{\ell}, \ell\colon \mathbb{B} \rhd^{M} s'' \uplus \{\ell \mapsto \mathbf{BLSSD}\} : (\Sigma_1|_a \boxplus \Sigma_2|_a), [\Sigma'_{21}]^{\ell}, \ell\colon \mathbb{B} & \text{algebra} \\
\Rightarrow & \Sigma_1|_a, \Sigma'_{21}|_u, \ell\colon \mathbb{D} \rhd^{M} s'' \uplus \{\ell \mapsto \mathbf{DFNCT}\} : (\Sigma_1|_a \boxplus \Sigma_2|_a), \Sigma'_{21}, \ell\colon \mathbb{D} & \text{lem. } 5.5.2 \\
\Leftrightarrow & \Sigma'_1 \rhd^{M} s'' \uplus \{\ell \mapsto \mathbf{DFNCT}\} : \Sigma'_1 \boxplus \Sigma'_2 & \text{defs. } \Sigma'_i.
\end{array}$$

Otherwise.

We have that $(s, \mathbf{e}) \longmapsto_M \mathbf{blame}\,\mathbf{f}$. Then by BLAME, $\mathbf{blame}\,\mathbf{f}$ has whatever type is needed.

Case $(s, {}_{\mathbf{f}}\mathbf{CA}[\ell]_{\mathsf{g}}^{\sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2}(\mathsf{v}_1)\ \mathbf{v_2}) \longmapsto_M \mathit{check}(s, \ell, \mathsf{q}, {}_{\mathbf{f}}\mathbf{CA}_{\mathsf{g}}^{\sigma_2}(\mathsf{v}_1\ {}_{\mathsf{g}}^{\sigma_1}\mathsf{AC}_{\mathbf{f}}(\mathbf{v_2})), \mathbf{blame}\,\mathbf{f})$.

There are three possibilities:

Case $\mathsf{q} = \mathsf{u}$.

Then $(s, \mathbf{e}) \longmapsto_M (s, {}_{\mathbf{f}}\mathbf{CA}^{\sigma_2}_{\mathsf{g}}(\mathsf{v}_1\ {}^{\sigma_1}_{\mathsf{g}}\mathsf{AC}_{\mathbf{f}}(\mathbf{v_2})))$.

We know there must be a derivation of the form

$$\dfrac{\dfrac{\dfrac{\mathcal{A}}{\Sigma_{211}; \cdot; \cdot \triangleright^{M}_{\mathscr{A}} \mathsf{v}_1 : \sigma_1 \overset{\mathsf{u}}{\multimap} \sigma_2} \qquad \dfrac{}{|\sigma_1 \overset{\mathsf{u}}{\multimap} \sigma_2| = \mathsf{u}}}{\Sigma_{211}; \cdot; \cdot \triangleright^{M}_{\mathscr{C}} \underset{\mathbf{f}\,\mathbf{g}}{\mathbf{CA}}[\ell]^{\sigma_1 \overset{\mathsf{u}}{\multimap} \sigma_2}(\mathsf{v}_1) : (\sigma_1)^{\mathscr{C}} \to (\sigma_2)^{\mathscr{C}}} \qquad \dfrac{\mathcal{B}}{\Sigma_{212}; \cdot; \cdot \triangleright^{M}_{\mathscr{C}} \mathbf{v_2} : (\sigma_1)^{\mathscr{C}}}}{\Sigma_{211} \boxplus \Sigma_{212}; \cdot; \cdot \triangleright^{M}_{\mathscr{C}} \underset{\mathbf{f}\,\mathbf{g}}{\mathbf{CA}}[\ell]^{\sigma_1 \overset{\mathsf{u}}{\multimap} \sigma_2}(\mathsf{v}_1)\ \mathbf{v_2} : (\sigma_2)^{\mathscr{C}}},$$

where $\tau' = (\sigma_2)^{\mathscr{C}}$ and $\Sigma_{21} \boxplus \Sigma_{22} = \Sigma_2$.

Then,

$$\dfrac{\dfrac{\dfrac{\mathcal{A}}{\Sigma_{211}; \cdot; \cdot \triangleright^{M}_{\mathscr{A}} \mathsf{v}_1 : \sigma_1 \overset{\mathsf{u}}{\multimap} \sigma_2} \qquad \dfrac{\dfrac{\mathcal{B}}{\Sigma_{212}; \cdot; \cdot \triangleright^{M}_{\mathscr{C}} \mathbf{v_2} : (\sigma_1)^{\mathscr{C}}}}{\Sigma_{212}; \cdot; \cdot \triangleright^{M}_{\mathscr{A}} {}^{\sigma_1}\underset{\mathsf{g}\,\mathbf{f}}{\mathsf{AC}}\,(\mathbf{v_2}) : \sigma_1}}{\Sigma_{211} \boxplus \Sigma_{212}; \cdot; \cdot \triangleright^{M}_{\mathscr{A}} \mathsf{v}_1\ {}^{\sigma_1}\underset{\mathsf{g}\,\mathbf{f}}{\mathsf{AC}}\,(\mathbf{v_2}) : \sigma_2}}{\Sigma_{211} \boxplus \Sigma_{212}; \cdot; \cdot \triangleright^{M}_{\mathscr{C}} \underset{\mathbf{f}\,\mathbf{g}}{\mathbf{CA}}{}^{\sigma_2}\left(\mathsf{v}_1\ {}^{\sigma_1}\underset{\mathsf{g}\,\mathbf{f}}{\mathsf{AC}}\,(\mathbf{v_2})\right) : (\sigma_2)^{\mathscr{C}}}.$$

Case $s = s'' \uplus \{\ell \mapsto \mathbf{BLSSD}\}$ and $\mathsf{q} = \mathsf{a}$.

Then $(s, \mathbf{e}) \longmapsto_M (s'' \uplus \{\ell \mapsto \mathbf{BLSSD}\}, {}_{\mathbf{f}}\mathbf{CA}^{\sigma_2}_{\mathsf{g}}(\mathsf{v}_1\ {}^{\sigma_1}_{\mathsf{g}}\mathsf{AC}_{\mathbf{f}}(\mathbf{v_2})))$.

We know there must be a derivation of the form

$$\dfrac{\dfrac{\dfrac{\mathcal{A}}{\Sigma'_{211}; \cdot; \cdot \triangleright^{M}_{\mathscr{A}} \mathsf{v}_1 : \sigma_1 \overset{\mathsf{a}}{\multimap} \sigma_2} \qquad \dfrac{}{|\sigma_1 \overset{\mathsf{a}}{\multimap} \sigma_2| = \mathsf{a}}}{[\Sigma'_{211}]^{\ell}, \ell{:}\,\mathbb{B}; \cdot; \cdot \triangleright^{M}_{\mathscr{C}} \underset{\mathbf{f}\,\mathbf{g}}{\mathbf{CA}}[\ell]^{\sigma_1 \overset{\mathsf{a}}{\multimap} \sigma_2}(\mathsf{v}_1) : (\sigma_1)^{\mathscr{C}} \to (\sigma_2)^{\mathscr{C}}} \qquad \dfrac{\mathcal{B}}{\Sigma_{21}; \cdot; \cdot \triangleright^{M}_{\mathscr{C}} \mathbf{v_2} : (\sigma_1)^{\mathscr{C}}}}{\Sigma_{21}; \cdot; \cdot \triangleright^{M}_{\mathscr{C}} \underset{\mathbf{f}\,\mathbf{g}}{\mathbf{CA}}[\ell]^{\sigma_1 \overset{\mathsf{a}}{\multimap} \sigma_2}(\mathsf{v}_1)\ \mathbf{v_2} : (\sigma_2)^{\mathscr{C}}},$$

where $[\Sigma'_{211}]^{\ell}, \ell{:}\,\mathbb{B} = \Sigma_{21}|_u$.

Note that we can decompose Σ_{21} as

- $\Sigma_{21} = \Sigma_{21}|_a, \Sigma'_{211}|_u, [\Sigma'_{211}|_a]^{\ell}, \ell{:}\,\mathbb{B}$.

By Lemma 5.5.2,

- $\Sigma_{21}|_a, \Sigma'_{211}|_u, [\Sigma'_{211}|_a]^{\ell}, \ell{:}\,\mathbb{D}; \cdot; \cdot \triangleright^{M}_{\mathscr{C}} \mathbf{v_2} : (\sigma_1)^{\mathscr{C}}$

Note that $\operatorname{dom} \Sigma_{21}|_a$ and $\operatorname{dom} \Sigma'_{211}|_a$ are disjoint.

Let $\Sigma'_{212} = \Sigma_{21}|_a, \Sigma'_{211}|_u, \ell{:}\,\mathbb{D}$.

Note that $\Sigma'_{212}|_a = \Sigma_{21}|_a$, which means that $\operatorname{dom} \Sigma'_{211}|_a$ and $\operatorname{dom} \Sigma'_{212}|_a$ are disjoint.

Then, by Lemma 5.5.2 again,

- $\Sigma'_{212}; \cdot; \cdot \triangleright^{M}_{\mathscr{C}} \mathbf{v_2} : (\sigma_2)^{\mathscr{C}}$

Note also that because $\ell \notin \operatorname{dom} \Sigma'_{211}$, we know that $\Sigma'_{211}, \ell{:}\, \mathbb{D}$ is well-formed, and by weakening,

- $\Sigma'_{211}, \ell{:}\, \mathbb{D}; \cdot; \cdot \triangleright^{M}_{\mathscr{A}} \mathsf{v}_1 : \sigma_1 \overset{\mathsf{a}}{\multimap} \sigma_2$

Finally, let $\Sigma'_{21} = (\Sigma'_{211}, \ell{:}\, \mathbb{D}) \boxplus \Sigma'_{212}$, which is defined because $\operatorname{dom} \Sigma'_{211}|_a$ and $\operatorname{dom} \Sigma'_{212}|_a$ are disjoint.

We can thus construct a derivation:

$$\dfrac{\dfrac{\dfrac{\mathcal{A}\text{, weakening}}{\Sigma'_{211}, \ell{:}\, \mathbb{D}; \cdot; \cdot \triangleright^{M}_{\mathscr{A}} \mathsf{v}_1 : \sigma_1 \overset{\mathsf{a}}{\multimap} \sigma_2} \qquad \dfrac{\dfrac{\mathcal{B}\text{, Lemma 5.5.2}}{\Sigma'_{212}; \cdot; \cdot \triangleright^{M}_{\mathscr{C}} \mathbf{v_2} : (\sigma_1)^{\mathscr{C}}}}{\Sigma'_{212}; \cdot; \cdot \triangleright^{M}_{\mathscr{A}} {}^{\sigma_1}\mathsf{AC}_{\mathsf{g\,f}}\,(\mathbf{v_2}) : \sigma_1}}{\Sigma'_{21}; \cdot; \cdot \triangleright^{M}_{\mathscr{A}} \mathsf{v}_1\ {}^{\sigma_1}\mathsf{AC}_{\mathsf{g\,f}}\,(\mathbf{v_2}) : \sigma_2}}{\Sigma'_{21}; \cdot; \cdot \triangleright^{M}_{\mathscr{C}} \mathbf{CA}^{\sigma_2}_{\mathbf{f\,g}} \left(\mathsf{v}_1\ {}^{\sigma_1}\mathsf{AC}_{\mathsf{g\,f}}\,(\mathbf{v_2})\right) : (\sigma_2)^{\mathscr{C}}}$$

Since $\Sigma_{21} \sim_u \Sigma_{22}$, we can decompose Σ_{22} as

- $\Sigma_{22} = \Sigma_{22}|_a, \Sigma_{21}|_u$, and thus
- $\Sigma_{22} = \Sigma_{22}|_a, [\Sigma'_{211}]^{\ell}, \ell{:}\, \mathbb{B}$.

Note that the domains of $\Sigma_{22}|_a$ and Σ'_{211} are disjoint.

Let $\Sigma'_{22} = \Sigma_{22}|_a, \Sigma'_{211}|_u, \ell{:}\, \mathbb{D}$.

Recall that $\Sigma_{22}; \cdot; \cdot \triangleright^{M}_{\mathscr{C}} \mathbf{E}[\mathbf{e}'']_{\mathscr{C}} : \tau$. By Lemma 5.5.2,

- $\Sigma'_{22}; \cdot; \cdot \triangleright^{M}_{\mathscr{C}} \mathbf{E}[\mathbf{e}'']_{\mathscr{C}} : \tau$

We previously defined $\Sigma'_{21} = (\Sigma'_{211}, \ell{:}\, \mathbb{D}) \boxplus \Sigma'_{212}$, which we can also decompose as

- $\Sigma'_{21} = (\Sigma'_{211}|_a \boxplus \Sigma_{21}|_a), \Sigma'_{211}|_u, \ell{:}\, \mathbb{D})$.

Let $\Sigma'_2 = \Sigma'_{21} \boxplus \Sigma'_{22}$, which is defined because the domains of $\Sigma'_{211}|_a$, $\Sigma_{21}|_a$, and $\Sigma_{22}|_a$ are all disjoint,

Then, by Lemma 5.4.3,

- $\Sigma'_2; \cdot; \cdot \triangleright^{M}_{\mathscr{C}} \mathbf{E}[\mathbf{e}']_{\mathscr{C}} : \tau$.

It now suffices to show that $\Sigma'_1 \triangleright^{M} s'' : \Sigma'_1 \boxplus \Sigma'_2$ for some Σ'_1. Let $\Sigma'_1 = \Sigma_1|_a, \Sigma'_{211}|_u, \ell{:}\, \mathbb{D}$. Since ℓ is fresh and $\operatorname{dom} \Sigma_1|_a$ is disjoint from $\operatorname{dom} \Sigma'_{211}$, it is well-formed.

$$\begin{aligned}
&\Sigma_1 \triangleright^{M} s'' \uplus \{\ell \mapsto \mathbf{BLSSD}\} : \Sigma_1 \boxplus \Sigma_2 && \textit{(ii)}\\
\Leftrightarrow\ &\Sigma_1|_a, [\Sigma'_{211}]^{\ell}, \ell{:}\, \mathbb{B} \triangleright^{M} s'' \uplus \{\ell \mapsto \mathbf{BLSSD}\} : (\Sigma_1|_a \boxplus \Sigma_2|_a), [\Sigma'_{211}]^{\ell}, \ell{:}\, \mathbb{B} && \text{algebra}\\
\Rightarrow\ &\Sigma_1|_a, \Sigma'_{211}|_u, \ell{:}\, \mathbb{D} \triangleright^{M} s'' \uplus \{\ell \mapsto \mathbf{DFNCT}\} : (\Sigma_1|_a \boxplus \Sigma_2|_a), \Sigma'_{211}, \ell{:}\, \mathbb{D} && \text{lem. 5.5.2}\\
\Leftrightarrow\ &\Sigma'_1 \triangleright^{M} s'' \uplus \{\ell \mapsto \mathbf{DFNCT}\} : \Sigma'_1 \boxplus \Sigma'_2 && \text{defs. } \Sigma'_i.
\end{aligned}$$

Otherwise.

We have that $(s, \mathbf{e}) \longmapsto_M (s, \mathbf{blame\,f})$. Then by RTC-Blame, $\mathbf{blame\,f}$ has whatever type is needed.

Case $(s, \mathbf{E}[\mathsf{e}]_{\mathscr{A}}) \longmapsto_M (s', \mathbf{E}[\mathsf{e}']_{\mathscr{A}})$ if $(s, \mathsf{e}) \longmapsto_M (s', \mathsf{e}')$.

By inversion on Conf-A, we know that

(*i*) $\vdash^M m$ okay for every module m in M,

(*ii*) $\Sigma_1 \rhd^M s : \Sigma_1 \boxplus \Sigma_2$, and

(*iii*) $\Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathbf{E}[\mathsf{e}]_{\mathscr{A}} : \tau$.

By Lemma 5.4.2, there exist some σ' and $\Sigma_{21} \boxplus \Sigma_{22} = \Sigma_2$ such that

- $\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{e} : \sigma'$, and
- $\Sigma_{22}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathbf{E}[\mathsf{e}'']_{\mathscr{A}} : \tau$ for all e'' such that $\cdot; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{e}'' : \sigma'$.

In cases where $s = s'$, it is sufficient to show that $\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{e}' : \sigma'$. By Lemma 5.4.3, we have that $\Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathbf{E}[\mathsf{e}']_{\mathscr{A}} : \tau$, and by rule Conf-A, we have the desired result.

In cases where $s \neq s'$, we will need to rebuild the configuration typing using the new store.

Now, by cases on $(s, \mathsf{e}) \longmapsto_M (s', \mathsf{e}')$:

Case $(s, \mathsf{c\ v}) \longmapsto_M \delta_{\mathscr{A}}(s, \mathsf{c}, \mathsf{v})$.

Metafunction $\delta_{\mathscr{A}}$ is defined in only four cases:

Case $\delta_{\mathscr{A}}(s, -, \lceil z \rceil) = (s, (z-))$.

Since $\mathrm{ty}_{\mathscr{A}}(-) = \mathsf{int} \overset{\mathsf{u}}{\multimap} \mathsf{int} \overset{\mathsf{u}}{\multimap} \mathsf{int}$ and $\lceil z \rceil$ has type int, we know that $\sigma' = \mathsf{int} \overset{\mathsf{u}}{\multimap} \mathsf{int}$, which is also the type of $(z-)$. Since s does not change, this is sufficient.

Case $\delta_{\mathscr{A}}(s, (z_1-), \lceil z_2 \rceil) = (s, \lceil z_1 - z_2 \rceil)$.

Since $\mathrm{ty}_{\mathscr{A}}((z_1-)) = \mathsf{int} \overset{\mathsf{u}}{\multimap} \mathsf{int}$ and $\lceil z_2 \rceil$ has type int, we know that $\sigma' = \mathsf{int}$, which is also the type of $\lceil z_1 - z_2 \rceil$. Since s does not change, this is sufficient.

Case $\delta_{\mathscr{A}}(s, \mathsf{new}[\sigma''], \mathsf{v}) = (s \uplus \{\ell \mapsto \mathsf{v}\}, \ell)$.

There must be a derivation

$$\dfrac{\dfrac{}{\Sigma_{21}|_u; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{new}[\sigma''] : \sigma'' \overset{\mathsf{u}}{\multimap} \sigma''\ \mathsf{ref}} \qquad \dfrac{\mathcal{B}}{\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{v} : \sigma''}}{\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{new}[\sigma'']\ \mathsf{v} : \sigma''\ \mathsf{ref}},$$

where $\sigma' = \sigma''\ \mathsf{ref}$.

Since $\mathsf{e}' = \ell$, by RTA-Loc,

- $\cdot, \ell{:}\sigma''; \cdot \rhd^M_{\mathscr{A}} \mathsf{e}' : \sigma''\ \mathsf{ref}$

By weakening, we can type e' with $\Sigma|_u, \ell{:}\sigma''$.

Recall that $\Sigma_{22}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathbf{E}[\mathsf{e}'']_{\mathscr{A}} : \tau$. Then by Conf-A,

$$\dfrac{(i) \qquad \mathcal{D} \qquad \dfrac{\text{Lemma 5.4.3}}{\Sigma_{22}, \ell{:}\sigma''; \cdot \triangleright^M_{\mathscr{A}} \mathbf{E}[\mathsf{e}']_{\mathscr{A}} : \tau}}{\triangleright^M (s \uplus \{\ell \mapsto \mathsf{v}\}, \mathbf{E}[\mathsf{e}']_{\mathscr{A}}) : \tau}$$

where

$$\mathcal{D} = \dfrac{\dfrac{(ii),\ \Sigma_{21} \boxplus \Sigma_{22} = \Sigma_2}{\Sigma_1 \triangleright^M s : \Sigma_1 \boxplus (\Sigma_{21} \boxplus \Sigma_{22})} \qquad \dfrac{\mathcal{B}}{\Sigma_{21}; \cdot; \cdot \triangleright^M_{\mathscr{A}} \mathsf{v} : \sigma''}}{\Sigma_1 \boxplus \Sigma_{21} \triangleright^M s \uplus \{\ell \mapsto \mathsf{v}\} : (\Sigma_1 \boxplus \Sigma_{21}) \boxplus \Sigma_{22}, \ell{:}\sigma''}.$$

Case $\delta_{\mathscr{A}}(s'' \uplus \{\ell \mapsto \mathsf{v}_1\}, \mathsf{swap}[\sigma_1][\sigma_2], \langle \ell, \mathsf{v}_2 \rangle) = (s'' \uplus \{\ell \mapsto \mathsf{v}_2\}, \langle \mathsf{v}_1, \ell \rangle)$.

There must be a derivation

$$\dfrac{\dfrac{}{\Sigma'_{21}|_u; \cdot; \cdot \triangleright^M_{\mathscr{A}} \mathsf{swap}[\sigma_1][\sigma_2] : \cdots} \qquad \dfrac{\dfrac{}{\Sigma'_{21}|_u, \ell{:}\sigma_1; \cdot; \cdot \triangleright^M_{\mathscr{A}} \ell : \sigma_1 \ \mathsf{ref}} \qquad \dfrac{\mathcal{A}}{\Sigma'_{21}; \cdot; \cdot \triangleright^M_{\mathscr{A}} \mathsf{v}_2 : \sigma_2}}{\Sigma'_{21}, \ell{:}\sigma_1; \cdot; \cdot \triangleright^M_{\mathscr{A}} \langle \ell, \mathsf{v}_2 \rangle : \sigma_1 \ \mathsf{ref} \otimes \sigma_2}}{\Sigma'_{21}, \ell{:}\sigma_1; \cdot; \cdot \triangleright^M_{\mathscr{A}} \mathsf{swap}[\sigma_1][\sigma_2]\ \langle \ell, \mathsf{v}_2 \rangle : \sigma_1 \otimes \sigma_2 \ \mathsf{ref}},$$

where $\sigma' = \sigma_1 \otimes \sigma_2 \ \mathsf{ref}$ and $\Sigma_{21} = \Sigma'_{21}, \ell{:}\sigma_1$.

From (ii) we can say that $\Sigma_1 \triangleright^M s : \Sigma_1 \boxplus \Sigma'_{21} \boxplus \Sigma_{22}, \ell{:}\sigma_1$.

Considering the type rules for stores, there must therefore be a derivation

$$\dfrac{\dfrac{\mathcal{B}}{\Sigma_{11} \triangleright^M s'' : \Sigma_1 \boxplus \Sigma'_{21} \boxplus \Sigma_{22}} \qquad \dfrac{\mathcal{C}}{\Sigma_{12}; \cdot; \cdot \triangleright^M_{\mathscr{A}} \mathsf{v}_1 : \sigma_1}}{\Sigma_1 \triangleright^M s'' \uplus \{\ell \mapsto \mathsf{v}_1\} : \Sigma_1 \boxplus \Sigma'_{21} \boxplus \Sigma_{22}, \ell{:}\sigma_1},$$

where $\Sigma_{11} \boxplus \Sigma_{12} = \Sigma_1$.

Now we can construct a type derivation:

$$\dfrac{\dfrac{\mathcal{C}}{\Sigma_{12}; \cdot; \cdot \triangleright^M_{\mathscr{A}} \mathsf{v}_1 : \sigma_1} \qquad \dfrac{}{\Sigma_{12}|_u, \ell{:}\sigma_2 \triangleright^M_{\mathscr{A}} \ell : \sigma_2 \ \mathsf{ref}}}{\Sigma_{12}, \ell{:}\sigma_2; \cdot; \cdot \triangleright^M_{\mathscr{A}} \langle \mathsf{v}_1, \ell \rangle : \sigma_1 \otimes \sigma_2 \ \mathsf{ref}}.$$

Recall that $\Sigma_{22}; \cdot; \cdot \triangleright^M_{\mathscr{A}} \mathbf{E}[\mathsf{e}'']_{\mathscr{A}} : \tau$. Then by CONF-A,

$$\dfrac{(i) \qquad \mathcal{D} \qquad \dfrac{\text{Lemma 5.4.3}}{\Sigma_{12} \boxplus \Sigma_{22}, \ell{:}\sigma_2 \triangleright^M_{\mathscr{A}} \mathbf{E}[\mathsf{e}']_{\mathscr{A}} : \tau}}{\triangleright^M s'' \uplus \{\ell \mapsto \mathsf{v}_2\}, \mathbf{E}[\mathsf{e}']_{\mathscr{A}}) : \tau}$$

where

$$\mathcal{D} = \dfrac{\dfrac{\mathcal{B}}{\Sigma_{11} \triangleright^M s'' : \Sigma_{11} \boxplus \Sigma'_{21} \boxplus \Sigma_{12} \boxplus \Sigma_{22}} \qquad \dfrac{\mathcal{A}}{\Sigma'_{21}; \cdot; \cdot \triangleright^M_{\mathscr{A}} \mathsf{v}_2 : \sigma_2}}{\Sigma_{11} \boxplus \Sigma'_{21} \triangleright^M s'' \uplus \{\ell \mapsto \mathsf{v}_2\} : \Sigma_{11} \boxplus \Sigma'_{21} \boxplus \Sigma_{12} \boxplus \Sigma_{22}, \ell{:}\sigma_2}.$$

Case $(s, (\Lambda\alpha^{\mathsf{q}}.\, \mathsf{v})[\sigma_{\mathsf{a}}]) \longmapsto_M (s, \mathsf{v}[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q}}])$.

By inversion of RTA-TAPP, we know that

- $\Sigma_{21}; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \Lambda\alpha^{\mathsf{q}}.\, \mathsf{v} : \forall\alpha^{\mathsf{q}}.\, \sigma_{\mathsf{b}}$,
- $\cdot \vdash_{\mathscr{A}} \sigma_{\mathsf{a}}$, and
- $|\sigma_{\mathsf{a}}| \sqsubseteq \mathsf{q}$, where
- $\sigma' = \sigma_{\mathsf{b}}[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q}}]$.

Then, by inversion of RTA-TLam, we know that

- $\Sigma_{21}; \cdot, \alpha^{\mathsf{q}}; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{v} : \sigma_{\mathsf{b}}$.

By (ii) and Lemma 5.2.7, $\mathrm{FTV}(\Sigma_{21}) = \varnothing$, and $\alpha^{\mathsf{q}} \notin \mathrm{FTV}(\cdot)$, so by Lemma 5.4.6, we conclude that $\Sigma_{21}; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{v}[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q}}] : \sigma_{\mathsf{b}}[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q}}]$.

Case $(s, (\lambda \mathsf{x}{:}\sigma_{\mathsf{x}}.\, \mathsf{e}_1)\ \mathsf{v}) \longmapsto_M (s, \mathsf{e}_1[\mathsf{v}/\mathsf{x}])$.

By inversion of RTA-App, we know that there exist some Σ_{211} and Σ_{212} such that

- $\Sigma_{211}; \cdot \rhd^{M}_{\mathscr{A}} \lambda \mathsf{x}{:}\sigma_{\mathsf{x}}.\, \mathsf{e}_1 : \sigma_{\mathsf{x}} \xrightarrow{\mathsf{q}}\!\circ\, \sigma'$ and
- $\Sigma_{212}; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{v} : \sigma_{\mathsf{x}}$, where
- $\Sigma_{211} \boxplus \Sigma_{212} = \Sigma_{21}$.

Then, by inversion of TA-Lam on the former, we know that

- $\Sigma_{211}; \mathsf{x} : \sigma_{\mathsf{x}} \rhd^{M}_{\mathscr{A}} \mathsf{e}_1 : \sigma'$.

By Lemma 5.4.9, we conclude that $\Sigma_{21}; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{e}_1[\mathsf{v}/\mathsf{x}] : \sigma'$ as well.

Case $(s, \mathsf{let}\ \langle \mathsf{x}_1, \mathsf{x}_2 \rangle = \langle \mathsf{v}_1, \mathsf{v}_2 \rangle\ \mathsf{in}\ \mathsf{e}_1) \longmapsto_M (s, \mathsf{e}_1[\mathsf{v}_2/\mathsf{x}_2][\mathsf{v}_1/\mathsf{x}_1])$.

By inversion of RTA-Let and RTA-Pair, there must be a derivation:

$$\dfrac{\dfrac{\dfrac{\mathcal{A}}{\Sigma_{2111}; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{v}_1 : \sigma_1} \qquad \dfrac{\mathcal{B}}{\Sigma_{2112}; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{v}_2 : \sigma_2}}{\Sigma_{211}; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \langle \mathsf{v}_1, \mathsf{v}_2 \rangle : \sigma_1 \otimes \sigma_2} \qquad \dfrac{\mathcal{C}}{\Sigma_{212}; \cdot; \cdot, \mathsf{x}_1{:}\sigma_1, \mathsf{x}_2{:}\sigma_2 \rhd^{M}_{\mathscr{A}} \mathsf{e}_1 : \sigma}}{\Sigma_{21}; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{let}\ \langle \mathsf{x}_1, \mathsf{x}_2 \rangle = \langle \mathsf{v}_1, \mathsf{v}_2 \rangle\ \mathsf{in}\ \mathsf{e}_1 : \sigma'},$$

for some $\Sigma_{211} \boxplus \Sigma_{212} = \Sigma_{21}$ and $\Sigma_{2111} \boxplus \Sigma_{2112} = \Sigma_{211}$.

Then by Lemma 5.4.9,

- $\Sigma_{212} \boxplus \Sigma_{2112}; \cdot; \cdot, \mathsf{x}_1{:}\sigma_1 \rhd^{M}_{\mathscr{A}} \mathsf{e}_1[\mathsf{v}_2/\mathsf{x}_2] : \sigma'$,

and by Lemma 5.4.9 again, $\Sigma_{212} \boxplus \Sigma_{2112} \boxplus \Sigma_{2111}; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{e}_1[\mathsf{v}_2/\mathsf{x}_2][\mathsf{v}_1/\mathsf{x}_1] : \sigma'$.

Case $(s, \mathsf{if0}\lceil 0 \rceil\ \mathsf{e}_{\mathsf{t}}\ \mathsf{e}_{\mathsf{f}}) \longmapsto_M (s, \mathsf{e}_{\mathsf{t}})$.

By inversion on RTA-If0, we know that $\Sigma_{212}; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{e}_{\mathsf{t}} : \sigma'$ where $\Sigma_{211} \boxplus \Sigma_{212} = \Sigma_{21}$, and by weakening, $\Sigma_{21}; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{e}_{\mathsf{t}} : \sigma'$.

Case $(s, \mathsf{if0}\lceil z \rceil\ \mathsf{e}_{\mathsf{t}}\ \mathsf{e}_{\mathsf{f}}) \longmapsto_M (s, \mathsf{e}_{\mathsf{f}})\ (z \neq 0)$.

By symmetry.

Case $(s, \mathsf{f}) \longmapsto_M (s, \mathsf{v})\ (\mathsf{module}\ \mathsf{f} : \sigma = \mathsf{v} \in M)$.

By inversion of RTA-Mod, σ must equal σ'.

Furthermore, premiss (i) from the inversion of CONF-A above tells us that $\vdash^{M} m$ okay for every module m in M, and for $\mathsf{module}\ \mathsf{f} : \sigma' = \mathsf{v}$ in particular. This judgment can only be the conclusion of rule TM-A, from which inversion tells us that

- $\cdot;\cdot \vdash^{M}_{\mathscr{A}} \mathsf{v} : \sigma'$.

By Lemma 5.3.1,

- $\cdot;\cdot;\cdot \rhd^{M}_{\mathscr{A}} \mathsf{v} : \sigma'$,

and by weakening, $\Sigma_{21};\cdot;\cdot \rhd^{M}_{\mathscr{A}} \mathsf{v} : \sigma'$.

Case $(s, \mathbf{f}^{\mathsf{g}}) \longmapsto_M (s, {}^{(\tau_{\mathbf{f}})^{\mathscr{A}}}_{\mathsf{g}}\mathsf{AC}_{\mathbf{f}}(\mathbf{f}))$ $(\mathbf{module}\ \mathbf{f} : \tau_{\mathbf{f}} = \mathbf{v} \in M)$.

By inversion of RTA-MODC, $\sigma' = (\tau_{\mathbf{f}})^{\mathscr{A}}$ and $\cdot \vdash_{\mathscr{C}} \tau_{\mathbf{f}}$.

Then, by RTA-MOD and RTA-BOUNDARY,

$$\dfrac{\dfrac{\mathbf{module}\ \mathbf{f} : \tau_{\mathbf{f}} = \mathbf{v} \in M \qquad \cdot \vdash_{\mathscr{C}} \tau_{\mathbf{f}}}{\Sigma_{21};\cdot;\cdot \rhd^{M}_{\mathscr{C}} \mathbf{f} : \tau_{\mathbf{f}}}}{\Sigma_{21};\cdot;\cdot \rhd^{M}_{\mathscr{A}} {}^{\sigma'}\underset{\mathsf{g}\ \mathbf{f}}{\mathsf{AC}}(\mathbf{f}) : \sigma'}.$$

Case $(s, \mathbf{f}^{\mathsf{g}}) \longmapsto_M (s, {}^{\sigma}_{\mathsf{g}}\mathsf{AC}_{\mathbf{f}}(\mathbf{f}'))$ $(\mathbf{interface}\ \mathbf{f} :> \sigma = \mathbf{f}' \in M)$.

By inversion of RTA-MODI, $\sigma' = \sigma$ and $\cdot \vdash_{\mathscr{A}} \sigma$.

Inverting CONF-A, the configuration C types only if

- $\vdash^{M} \mathbf{interface}\ \mathbf{f} :> \sigma' : \mathbf{f}'$ okay.

The only rule with this conclusion is TM-I, so there must exist some $\mathbf{v}$ such that

- $\mathbf{module}\ \mathbf{f}' : (\sigma')^{\mathscr{C}} = \mathbf{v} \in M$.

Then,

$$\dfrac{\dfrac{\mathbf{module}\ \mathbf{f}' : (\sigma')^{\mathscr{C}} = \mathbf{v} \in M \qquad \dfrac{\text{Lemma 5.2.2}}{\cdot \vdash_{\mathscr{C}} (\sigma')^{\mathscr{C}}}}{\Sigma_{21};\cdot;\cdot \rhd^{M}_{\mathscr{C}} \mathbf{f}' : (\sigma')^{\mathscr{C}}}}{\Sigma_{21};\cdot;\cdot \rhd^{M}_{\mathscr{A}} {}^{\sigma'}\underset{\mathsf{g}\ \mathbf{f}}{\mathsf{AC}}(\mathbf{f}') : \sigma'}.$$

Case $(s, {}^{\sigma}_{\mathsf{f}}\mathsf{AC}_{\mathbf{g}}(\mathbf{v})) \longmapsto_M \mathit{coerce}_{\mathscr{A}}(s, \sigma, \mathbf{v}, \mathsf{f}, \mathbf{g})$.

There are three possibilities:

Case $\mathbf{v} = \ulcorner z \urcorner$.

Then $(s, \mathsf{e}) \longmapsto_M (s, \ulcorner z \urcorner)$.

The only rule to type e is RTA-BOUNDARY, which gives it the type $(\mathbf{int})^{\mathscr{A}} = \mathsf{int}$.

The only rule to type e' is RTA-CON, which gives $\mathrm{ty}_{\mathscr{A}}(\ulcorner z \urcorner) = \mathsf{int}$ as well.

Case $\mathbf{v} = {}_{\mathsf{g}'}\mathbf{CA}[\ell]^{\sigma^{\circ}}_{\mathsf{f}'}(\mathsf{v}')$.

Then $(s, \mathsf{e}) \longmapsto_M \mathit{check}(s, \ell, |\sigma^{\circ}|, \mathsf{v}', {}^{\sigma}_{\mathsf{f}}\mathsf{AC}_{\mathsf{g}}(\mathbf{blame\,f'}))$.

Note that if $(\sigma^{\circ})^{\mathscr{C}} = (\sigma)^{\mathscr{C}}$, then then $\sigma^{\circ} = \sigma$, by Lemma 5.2.1.

Then there are three subsidiary possibilities:

Case $|\sigma| = \mathsf{u}$.

Then $(s, \mathsf{e}) \longmapsto_M (s, \mathsf{v}')$.

Because $|\sigma| = \mathsf{u}$, only rule RTC-Sealed applies for typing the $\mathbf{CA}$ subterm.

Thus, we know there must be a derivation of the form

$$\dfrac{\dfrac{\dfrac{\mathcal{A}}{\Sigma_{21}; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{v} : \sigma^{\mathsf{w}}} \qquad \dfrac{\mathcal{B}}{|\sigma^{\mathsf{w}}| = \mathsf{u}}}{\Sigma_{21}; \Delta; \Gamma \rhd^{M}_{\mathscr{C}} \underset{\mathsf{g}'\,\mathsf{f}'}{\mathbf{CA}}[\ell]^{\sigma^{\mathsf{w}}}(\mathsf{v}) : (\sigma^{\mathsf{w}})^{\mathscr{C}}}}{\Sigma_{21}; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} {}^{\sigma^{\mathsf{w}}}\underset{\mathsf{f}\,\mathsf{g}}{\mathsf{AC}}\left(\underset{\mathsf{g}'\,\mathsf{f}'}{\mathbf{CA}}[\ell]^{\sigma^{\mathsf{w}}}(\mathsf{v})\right) : \sigma^{\mathsf{w}}},$$

where $\sigma^{\mathsf{w}} = \sigma^{\circ} = \sigma' = \sigma$.

Then $\mathcal{A}$ suffices.

Case $s = s'' \uplus \{\ell \mapsto \mathbf{BLSSD}\}$ and $|\sigma| = \mathsf{a}$.

Then $(s'', \mathsf{e}) \longmapsto_M (s \uplus \{\ell \mapsto \mathbf{DFNCT}\}, \mathsf{v}')$

By inspection of s, it must be the case that $\Sigma(\ell) = \mathbb{B}$. Thus, rule RTC-Defunct will not apply to the AC subterm.

Furthermore, since $|\sigma| = \mathsf{a}$, RTC-Sealed does not apply.

Thus, by inversion of RTA-Boundary and RTC-Blessed, there must be a derivation

$$\dfrac{\dfrac{\dfrac{\mathcal{A}}{\Sigma'_{21}; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{v}' : \sigma^{\mathsf{w}}} \qquad |\sigma^{\mathsf{w}}| = \mathsf{a}}{[\Sigma'_{21}]^{\ell}, \ell{:}\,\mathbb{B}; \cdot; \cdot \rhd^{M}_{\mathscr{C}} \underset{\mathsf{g}'\,\mathsf{f}'}{\mathbf{CA}}[\ell]^{\sigma^{\mathsf{w}}}(\mathsf{v}') : (\sigma^{\mathsf{w}})^{\mathscr{C}}}}{[\Sigma'_{21}]^{\ell}, \ell{:}\,\mathbb{B}; \cdot; \cdot \rhd^{M}_{\mathscr{A}} {}^{\sigma^{\mathsf{w}}}\underset{\mathsf{f}\,\mathsf{g}}{\mathsf{AC}}\left(\underset{\mathsf{g}'\,\mathsf{f}'}{\mathbf{CA}}[\ell]^{\sigma^{\mathsf{w}}}(\mathsf{v}')\right) : \sigma^{\mathsf{w}}},$$

where $\sigma^{\mathsf{w}} = \sigma^{\circ} = \sigma' = \sigma$ and $\Sigma_{21} = [\Sigma'_{21}]^{\ell}, \ell{:}\,\mathbb{B}$.

From $\mathcal{A}$ and by weakening,

- $\Sigma'_{21}, \ell{:}\,\mathbb{D}; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{v} : \sigma'$.

Note that we can decompose Σ_2 as

- $\Sigma_2 = \Sigma_{21}, \Sigma_{22}|_a$.

Since $\Sigma_{21} = [\Sigma'_{21}]^{\ell}, \ell{:}\,\mathbb{B}$, we can decompose Σ_2 further as

- $\Sigma_2 = [\Sigma'_{21}]^{\ell}, \ell{:}\,\mathbb{B}, \Sigma_{22}|_a$.

Since $\Sigma_2|_a = \Sigma_{22}|_a$ and $\Sigma_2 \sim_u \Sigma_{22}$, we know that $\Sigma_2 = \Sigma_{22}$.

Recall that $\Sigma_{22}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathbf{E}[\mathsf{e}'']_{\mathscr{A}} : \tau$. By Lemma 5.5.2, we can type $\mathbf{E}[\mathsf{e}'']_{\mathscr{A}}$ with $\Sigma'_{21}|_u, \ell{:}\,\mathbb{D}, \Sigma_{22}|_a$.

Let $\Sigma'_2 = (\Sigma'_{21}, \ell{:}\,\mathbb{D}) \boxplus (\Sigma'_{21}|_u, \ell{:}\,\mathbb{D}, \Sigma_{22}|_a)$, which is clearly well-formed. Then, by Lemma 5.4.3,

- $\Sigma'_2; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathbf{E}[\mathsf{e}']_{\mathscr{A}} : \tau$.

It now suffices to show that $\Sigma'_1 \rhd^M s' : \Sigma'_1 \boxplus \Sigma'_2$ for some Σ'_1. Let $\Sigma'_1 = \Sigma_1|_a, \Sigma'_{21}|_u, \ell{:}\,\mathbb{D}$. Since ℓ is fresh and $\operatorname{dom}\Sigma_1|_a$ is disjoint from $\operatorname{dom}\Sigma'_{21}$, we know that Σ'_1 is well-formed.

$$\begin{array}{ll}
\Sigma_1 \rhd^M s'' \uplus \{\ell \mapsto \mathbf{BLSSD}\} : \Sigma_1 \boxplus \Sigma_2 & (ii) \\
\Leftrightarrow \Sigma_1|_a, [\Sigma'_{21}]^{\ell}, \ell{:}\,\mathbb{B} \rhd^M s'' \uplus \{\ell \mapsto \mathbf{BLSSD}\} : (\Sigma_1|_a \boxplus \Sigma_2|_a), [\Sigma'_{21}]^{\ell}, \ell{:}\,\mathbb{B} & \text{algebra} \\
\Rightarrow \Sigma_1|_a, \Sigma'_{21}|_u, \ell{:}\,\mathbb{D} \rhd^M s'' \uplus \{\ell \mapsto \mathbf{DFNCT}\} : (\Sigma_1|_a \boxplus \Sigma_2|_a), \Sigma'_{21}, \ell{:}\,\mathbb{D} & \text{lem. } 5.5.2 \\
\Leftrightarrow \Sigma'_1 \rhd^M s'' \uplus \{\ell \mapsto \mathbf{DFNCT}\} : \Sigma'_1 \boxplus \Sigma'_2 & \text{defs. } \Sigma'_i.
\end{array}$$

Otherwise.

We know that $(s, \mathsf{e}) \longmapsto_M (s, {}^{\sigma}_{\mathsf{f}}\mathsf{AC}_{\mathbf{g}}(\mathbf{blame}\,\mathbf{f}))$.

Then,

$$\frac{\overline{\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{blame}\,\mathbf{f} : (\sigma')^{\mathscr{C}}}}{\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{A}} {}^{\sigma'}\underset{\mathbf{g}'\ \mathbf{f}'}{\mathsf{AC}}\,(\mathbf{blame}\,\mathbf{f}) : \sigma'}$$

Otherwise.

We know that $(s, \mathsf{e}) \longmapsto_M (s, {}^{\sigma}_{\mathsf{f}}\mathsf{AC}[\,]_{\mathbf{g}}(\mathbf{v}))$.

Furthermore, since the previous two cases covered $\mathbf{int}$ and $\{\sigma^{\circ}\}$, by Observation 5.5.1, we may let $\tau^{\mathbf{w}} = (\sigma)^{\mathscr{C}}$.

By inversion of RTA-Boundary, there must be a derivation

$$\frac{\dfrac{\mathcal{A}}{\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{v} : (\sigma)^{\mathscr{C}}}}{\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{A}} {}^{\sigma}\underset{\mathsf{f}\ \mathbf{g}}{\mathsf{AC}}\,(\mathbf{v}) : \sigma}.$$

Then by RTA-Sealed,

$$\frac{\dfrac{\mathcal{A}}{\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{v} : (\sigma)^{\mathscr{C}}} \qquad (\sigma)^{\mathscr{C}} = \tau^{\mathbf{w}}}{\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{A}} {}^{\sigma}\underset{\mathsf{f}\ \mathbf{g}}{\mathsf{AC}}[\,](\mathbf{v}) : \sigma}.$$

Case $(s, {}^{\forall\alpha^{\mathsf{q}}.\,\sigma_{\mathsf{b}}}_{\mathsf{f}}\mathsf{AC}[\,]_{\mathbf{g}}(\mathbf{v})[\sigma_{\mathsf{a}}]) \longmapsto_M (s, {}^{\sigma_{\mathsf{b}}[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q}}]}_{\mathsf{f}}\mathsf{AC}_{\mathbf{g}}(\mathbf{v}[(\sigma_{\mathsf{a}})^{\mathscr{C}}]))$.

Rule RTA-TApp gives us that

- $\cdot \vdash_{\mathscr{A}} \sigma_{\mathsf{a}}$.

Furthermore RTA-Sealed gives us that

- $\sigma' = \forall \alpha^{\mathsf{q}}.\, \sigma_{\mathsf{b}}$.

By inversion, it must be the case that

- $\Sigma_{21}; \cdot; \cdot \rhd^{M}_{\mathscr{C}} \mathbf{v} : \forall \gamma.\, (\sigma_{\mathsf{b}}[\{\gamma\}/\alpha^{\mathsf{q}}])^{\mathscr{C}}$.

By Lemma 5.4.8,

- $\Sigma_{21}; \cdot \rhd \mathbf{v}$ worthy.

Then,

$$\dfrac{\dfrac{\dfrac{\text{Inv. RTA-Sealed}}{\Sigma_{21}; \cdot; \cdot \rhd^{M}_{\mathscr{C}} \mathbf{v} : \forall \gamma.\, (\sigma_{\mathsf{b}}[\{\gamma\}/\alpha^{\mathsf{q}}])^{\mathscr{C}}} \qquad \dfrac{\text{Inv. RTA-TApp, Lemma 5.2.2}}{\cdot \vdash_{\mathscr{C}} (\sigma_{\mathsf{a}})^{\mathscr{C}}}}{\Sigma_{21}; \cdot; \cdot \rhd^{M}_{\mathscr{C}} \mathbf{v}[(\sigma_{\mathsf{a}})^{\mathscr{C}}] : (\sigma_{\mathsf{b}}[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q}}])^{\mathscr{C}}}}{\Sigma_{21}; \cdot; \cdot \rhd^{M}_{\mathscr{A}} {}^{\sigma_{\mathsf{b}}[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q}}]}\underset{\mathsf{f}\,\mathbf{g}}{\mathsf{AC}} \left(\mathbf{v}[(\sigma_{\mathsf{a}})^{\mathscr{C}}]\right) : \sigma_{\mathsf{b}}[\sigma_{\mathsf{a}}/\alpha^{\mathsf{q}}]}.$$

Case $(s, {}^{\sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2}_{\mathsf{f}}\mathsf{AC}[\,]_{\mathbf{g}}(\mathbf{v_1})\ \mathsf{v}_2) \longmapsto_M (s, {}^{\sigma_2}_{\mathsf{f}}\mathsf{AC}_{\mathbf{g}}(\mathbf{v_1}\ {}_{\mathbf{g}}\mathbf{CA}^{\sigma_1}_{\mathsf{f}}(\mathsf{v}_2)))$.

Rule RTA-Boundary gives us that

- $\sigma' = \sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2$

Furthermore, RTA-App tells us that there exist some Σ_{211} and Σ_{212} such that

- $\Sigma_{212}; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{v}_2 : \sigma_1$, where
- $\Sigma_{211} \boxplus \Sigma_{212} = \Sigma_{21}$.

By inversion, it must be the case that

- $\Sigma_{21}; \cdot; \cdot \rhd^{M}_{\mathscr{C}} \mathbf{v_1} : (\sigma_1)^{\mathscr{C}} \to (\sigma_2)^{\mathscr{C}}$.

Then,

$$\dfrac{\dfrac{\Sigma_{211}; \cdot; \cdot \rhd^{M}_{\mathscr{C}} \mathbf{v_1} : (\sigma_1)^{\mathscr{C}} \to (\sigma_2)^{\mathscr{C}} \qquad \Sigma_{212}; \cdot; \cdot \rhd^{M}_{\mathscr{C}} \mathbf{v_2} : \sigma_1}{\Sigma_{211} \boxplus \Sigma_{212}; \cdot; \cdot \rhd^{M}_{\mathscr{C}} \mathbf{v_1}\ \underset{\mathbf{g}\,\mathsf{f}}{\mathbf{CA}}{}^{\sigma_1}(\mathsf{v}_2) : (\sigma_2)^{\mathscr{C}}}}{\Sigma_{211} \boxplus \Sigma_{212}; \cdot; \cdot \rhd^{M}_{\mathscr{A}} {}^{\sigma_2}\underset{\mathsf{f}\,\mathbf{g}}{\mathsf{AC}} \left(\mathbf{v_1}\ \underset{\mathbf{g}\,\mathsf{f}}{\mathbf{CA}}{}^{\sigma_1}(\mathsf{v}_2)\right) : \sigma_2}.$$

All other cases are subsumed by first case above, letting $\mathbf{E} = [\,]_{\mathscr{C}}$. $\square$

5.6 Progress

Definition 5.6.1 (Faulty expressions and configurations)**.** *We define the* ***faulty expressions with respect to store*** s *inductively as follows:*

$$\begin{aligned}
\mathbf{Q}_s \ ::= \ & \mathbf{Q}^{\mathbf{\Lambda}}_s[\tau] \\
& \textit{where } \mathbf{Q}^{\mathbf{\Lambda}}_s \ ::= \ \mathbf{c} \ \mid \ \lambda \mathbf{x}{:}\tau.\, \mathbf{e} \\
& \qquad\qquad\quad \mid \ \underset{\mathsf{f}\,\mathbf{g}}{\mathbf{CA}}[\ell]^{\sigma}(\mathsf{v}) \qquad\qquad (\sigma \neq \forall \alpha^{\mathsf{q}}.\, \sigma') \\
\mid \ & \mathbf{Q}^{\lambda}_{s,\mathbf{v}}\ \mathbf{v} \\
& \textit{where } \mathbf{Q}^{\lambda}_{s,\mathbf{v}} \ ::= \ \lceil z \rceil \ \mid \ \mathbf{\Lambda} \alpha.\, \mathbf{e}
\end{aligned}$$

$$
\begin{array}{llr}
 & \qquad \mid \; - \;\mid\; (z-) & (\mathbf{v} \neq \lceil z_2 \rceil) \\
 & \qquad \mid \; \underset{\mathbf{f}\,\mathbf{g}}{\mathbf{CA}}[\ell]^{\sigma}(\mathsf{v}) & (\sigma \neq \sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2) \\
 & \mid \; \mathbf{if0}\;\mathbf{v}\;\mathbf{e_t}\;\mathbf{e_f} & (\mathbf{v} \neq \lceil z \rceil) \\
 & \mid \; \mathbf{E}[\mathsf{Q}_s]_{\mathscr{C}} \;\mid\; \mathbf{E}[\mathsf{Q}_s]_{\mathscr{A}} & \\
\mathsf{Q}_s ::= & \mathsf{Q}_s^{\Lambda}[\tau] & \\
 & \textit{where } \mathsf{Q}_s^{\Lambda} ::= \mathsf{c} \mid \ell \mid \langle \mathsf{v}_1, \mathsf{v}_2 \rangle \mid \lambda \mathsf{x}{:}\sigma.\,\mathsf{e} & \\
 & \qquad\qquad \mid \; {}^{\sigma}\underset{\mathsf{f}\,\mathbf{g}}{\mathsf{AC}}[\,](\mathbf{v}) & (\sigma \neq \forall \alpha^{\mathsf{q}}.\,\sigma') \\
 & \mid \; \mathsf{Q}_{s,\mathsf{v}}^{\lambda}\;\mathsf{v} & \\
 & \textit{where } \mathsf{Q}_{s,\mathsf{v}}^{\lambda} ::= \lceil z \rceil \mid \ell \mid \langle \mathsf{v}_1, \mathsf{v}_2 \rangle \mid \Lambda \alpha^{\mathsf{q}}.\,\mathsf{e} & \\
 & \qquad\qquad \mid \; - \;\mid\; (z-) & (\mathsf{v} \neq \lceil z_2 \rceil) \\
 & \qquad\qquad \mid \; \mathsf{swap}[\sigma_1][\sigma_2] & (\neg \exists \ell \in \operatorname{dom} s, \mathsf{v} = \langle \ell, \mathsf{v}_2' \rangle) \\
 & \qquad\qquad \mid \; {}^{\sigma}\underset{\mathsf{f}\,\mathbf{g}}{\mathsf{AC}}[\,](\mathbf{v}) & (\sigma \neq \sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2) \\
 & \mid \; \mathsf{if0}\;\mathsf{v}\;\mathsf{e_t}\;\mathsf{e_f} & (\mathsf{v} \neq \lceil z \rceil) \\
 & \mid \; \mathsf{let}\;\langle \mathsf{x}_1, \mathsf{x}_2 \rangle = \mathsf{v}\;\mathsf{in}\;\mathsf{e} & (\mathsf{v} \neq \langle \mathsf{v}_1, \mathsf{v}_2 \rangle) \\
 & \mid \; \mathsf{E}[\mathsf{Q}_s]_{\mathscr{A}} \;\mid\; \mathsf{E}[\mathbf{Q}_s]_{\mathscr{C}} &
\end{array}
$$

*A **faulty configuration** is a configuration whose expression is faulty with respect to its store.*

Definition 5.6.2 (Redexes)**.** *In the definition of the relation ($\longmapsto_M$), every rule other than* C-CXT *and* C-CXTA *has either the form $(s, \mathbf{e_r}) \longmapsto_M C'$ or the form $(s, \mathsf{e_r}) \longmapsto_M C'$. We call the expressions $\mathbf{e_r}$ and $\mathsf{e_r}$ ($\lambda_{\mathscr{C}}$ and $\lambda^{\mathscr{A}}$) **redexes**, and denote them with the metasyntatic variables $\mathbf{R}$ and R, respectively.*

Lemma 5.6.3 (Redexes and evaluation contexts)**.**

If $(s, \mathbf{e}) \longmapsto_M (s', \mathbf{e}')$, then either:

- *We can decompose $\mathbf{e} = \mathbf{E}[\mathbf{R}]_{\mathscr{C}}$ and $\mathbf{e}' = \mathbf{E}[\mathbf{e_s}]_{\mathscr{C}}$. Then for any other evaluation context $\mathbf{E}'[\,]_{\mathscr{C}}$, we have that $(s, \mathbf{E}'[\mathbf{R}]_{\mathscr{C}}) \longmapsto_M (s', \mathbf{E}'[\mathbf{e_s}]_{\mathscr{C}})$ as well.*
- *We can decompose $\mathbf{e} = \mathbf{E}[\mathsf{R}]_{\mathscr{A}}$ and $\mathbf{e}' = \mathbf{E}[\mathsf{e_s}]_{\mathscr{A}}$. Then for any other evaluation context $\mathbf{E}'[\,]_{\mathscr{A}}$, we have that $(s, \mathbf{E}'[\mathsf{R}]_{\mathscr{A}}) \longmapsto_M (s', \mathbf{E}'[\mathsf{e_s}]_{\mathscr{A}})$ as well.*

Proof. By induction on the derivation of $(s, \mathbf{e}) \longmapsto_M (s', \mathbf{e}')$. □

Definition 5.6.4 (Closed configurations and module contexts)**.** *We consider a configuration C to be closed when all locations in the expression and the store are mapped by the store. We consider a module context M to be closed when all module names occuring in M are also defined in M. We consider C to be closed with respect to M when C is closed and all module names occuring in C are defined in M.*

Lemma 5.6.5 (Uniform evaluation)**.** *For any C closed with respect to M, either C is faulty or an answer, or there exists some C' closed with respect to M such that $C \longmapsto_M C'$.*

Proof. If $C = \mathbf{blame}\,\mathbf{f}$ for some module $\mathbf{f}$, then C is an answer. Otherwise, C must be of the form $(s, \mathbf{e})$.

We therefore generalize our induction hypothesis as follows.

(i) For any s and $\mathbf{e}$, if the configuration $(s, \mathbf{e})$ is closed with respect to closed M, then one of:

(Q) $\mathbf{e}$ is faulty with respect to s (and hence the configuration is faulty),

(A) $\mathbf{e}$ is a value (and hence the configuration is an answer),

(R) there exist some s' and $\mathbf{e}'$ such that $(s, \mathbf{e}) \longmapsto_M (s', \mathbf{e}')$, which is also closed with respect to M (let $C' = (s', \mathbf{e}')$).

(ii) For any s and e, if the configuration (s, e) is closed with respect to closed M, then one of:

(Q) e is faulty with respect to s,

(A) e is a value,

(R) there exist some s' and e' such that $(s, \mathsf{e}) \longmapsto_M (s', \mathsf{e}')$, which is also closed with respect to M.

We proceed by mutual induction on the structures of $\mathbf{e}$ and e.

(i) Cases on $\mathbf{e}$:

Case $\mathbf{v}$.

Then (A).

Case $\mathbf{x}$.

Vacuous, because $\mathbf{e}$ is closed.

Case $\mathbf{f}$.

Because C is closed in M, we know that there exists some $\mathbf{module\ f} : \tau = \mathbf{v} \in M$, thus $(s, \mathbf{x}) \longmapsto_M (s, \mathbf{v})$; because M is closed, we know that $\mathbf{v}$ is closed in M. Hence (R).

Case $\mathbf{e_1}[\tau]$.

Consider first the induction hypothesis at $\mathbf{e_1}$, noting that $\mathbf{E_1} = [\,]_{\mathscr{C}}[\tau]$ is an evaluation context.

(Q) Then (Q).

(A) Let $\mathbf{v_1} = \mathbf{e_1}$. Now by cases on $\mathbf{v_1}$:

Case $\mathbf{c}$.

Then (Q).

Case $\forall\alpha.\mathbf{e_{11}}$.

Then $(s, \mathbf{e}) \longmapsto_M (s, \mathbf{e_{11}}[\tau/\alpha])$, hence (R).

Case $\lambda\mathbf{x}{:}\tau.\mathbf{e_{11}}$.

Then (Q).

Case ${}_{\mathsf{f}}\mathbf{CA}[\ell]^{\sigma}_{\mathsf{g}}(\mathsf{v}_{11})$.

If $\sigma = \forall\alpha^{\mathsf{q}}.\,\sigma'$, then $(s, \mathbf{e}) \longmapsto_M \mathit{check}(\cdots)$, hence (R); otherwise (Q).

(R) That is, $(s, \mathbf{e_1}) \longmapsto_M (s', \mathbf{e}_1')$. Then $(s, \mathbf{E_1}[\mathbf{e_1}]_{\mathscr{C}}) \longmapsto_M (s', \mathbf{E_1}[\mathbf{e}_1']_{\mathscr{C}})$ by Lemma 5.6.3, hence, (R).

Case $\mathbf{e_1}\ \mathbf{e_2}$.

Consider first the induction hypothesis at $\mathbf{e_1}$, noting that $\mathbf{E_1} = [\,]_{\mathscr{C}}\ \mathbf{e_2}$ is an evaluation context.

(Q) Then (Q).

(A) Let $\mathbf{v_1} = \mathbf{e_1}$, and note that $\mathbf{E_2} = \mathbf{v_1}\ [\,]_{\mathscr{C}}$ is an evaluation context. We now apply the induction hypothesis to $\mathbf{e_2}$:

(Q) Then (Q).

(A) Let $\mathbf{v_2} = \mathbf{e_2}$. Now by cases on $\mathbf{v_1}$:

Case $\mathbf{c}$.

By cases on $\mathbf{c}$:

Case $\lceil z \rceil$.

Then (Q).

Case $(z-)$.

If $\mathbf{v_2} = \lceil z_2 \rceil$ then $(s, \mathbf{e}) \longmapsto_M (s, \lceil z - z_2 \rceil)$, hence (R); otherwise (Q).

Case $-$.

If $\mathbf{v_2} = \lceil z \rceil$ for some z, then $(s, \mathbf{e}) \longmapsto_M (s, (z-))$, hence (R); otherwise (Q).

Case $\forall\alpha.\mathbf{e_{11}}$.

Then (Q).

Case $\lambda\mathbf{x}{:}\tau.\mathbf{e_{11}}$.

Then $(s, \mathbf{e}) \longmapsto_M (s, \mathbf{e_{11}}[\mathbf{v_2}/\mathbf{x}])$, hence (R).

Case ${}_{\mathsf{f}}\mathbf{CA}[\ell]^{\sigma}_{\mathsf{g}}(\mathsf{v}_{11})$.

If $\sigma = \sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2$, then $(s, \mathbf{e}) \longmapsto_M \mathit{check}(\cdots)$, hence (R); otherwise (Q).

(R) That is, $(s, \mathbf{e_2}) \longmapsto_M (s', \mathbf{e}_2')$. Then $(s, \mathbf{E_2}[\mathbf{e_2}]_{\mathscr{C}}) \longmapsto_M (s', \mathbf{E_2}[\mathbf{e}_2']_{\mathscr{C}})$ by Lemma 5.6.3, hence, (R).

(R) That is, $(s, \mathbf{e_1}) \longmapsto_M (s', \mathbf{e}_1')$. Then $(s, \mathbf{E_1}[\mathbf{e_1}]_{\mathscr{C}}) \longmapsto_M (s', \mathbf{E_1}[\mathbf{e}_1']_{\mathscr{C}})$ by Lemma 5.6.3, hence, (R).

Case $\mathbf{if0}\,\mathbf{e_1}\,\mathbf{e_2}\,\mathbf{e_3}$.

Apply induction at $\mathbf{e_1}$, noting that $\mathbf{E_1} = \mathbf{if0}\,[\,]_{\mathscr{C}}\,\mathbf{e_2}\,\mathbf{e_3}$ is an evaluation context.

(Q) Then (Q).

(A) If $\mathbf{e_1} = \lceil z \rceil$ for some z, then (R) by one of the two **if0** rules; otherwise, (Q) by the definition of faulty expressions.

(R) Then (R) by Lemma 5.6.3.

Case $\mathsf{g}^{\mathbf{f}}$.

Because C is closed in M, we know that there exists some $\mathsf{module}\, \mathsf{g} : \sigma = \mathsf{v} \in M$, thus $(s,\ \mathsf{g}) \longmapsto_M (s,\ {}_{\mathbf{f}}\mathbf{CA}^{\sigma}_{\mathsf{g}}(\mathsf{g}))$; because M is closed, we know that v is closed in M. Hence (R).

Case ${}_{\mathbf{f}}\mathbf{CA}^{\sigma}_{\mathsf{g}}(\mathsf{e}_1)$.

Apply part (ii) of the induction hypothesis to e_1, noting that $\mathbf{E}' = {}_{\mathbf{f}}\mathbf{CA}^{\sigma}_{\mathsf{g}}([\,]_{\mathscr{A}})$ is an evaluation context:

(Q) Then (Q).

(A) Let $\mathbf{v_1} = \mathbf{e_1}$. Then $(s, \mathbf{e}) \longmapsto_M \mathit{coerce}_{\mathscr{C}}(\cdots)$, hence (R).

(R) That is, $(s, \mathsf{e}) \longmapsto_M (s', \mathsf{e}')$. Then (R).

(ii) Cases on e:

Case v.

Then (A).

Case x.

Vacuous, because e is closed.

Case f.

Because C is closed in M, we know that there exists some $\mathsf{module}\, \mathsf{x} : \sigma = \mathsf{v} \in M$, thus $(s, \mathsf{x}) \longmapsto_M (s, \mathsf{v})$; and since M is closed, v is closed in M. Hence (R).

Case $\mathsf{e}_1[\sigma]$.

Consider first the induction hypothesis at e_1, noting that $\mathsf{E}_1 = [\,]_{\mathscr{A}}[\sigma]$ is an evaluation context.

(Q) Then (Q).

(A) Let $\mathsf{v}_1 = \mathsf{e}_1$. Now by cases on v_1:

Case c.

Then (Q).

Case $\forall\alpha^{\mathsf{q}'}.\mathsf{e}_{11}$.

Then $(s, \mathsf{e}) \longmapsto_M (s, \mathsf{e}_{11}[\sigma/\alpha^{\mathsf{q}'}])$, hence (R).

Case $\lambda\mathsf{x}{:}\tau.\mathsf{e}_{11}$.

Then (Q).

Case $\langle \mathsf{v}_1, \mathsf{v}_2 \rangle$.

Then (Q).

Case ℓ.

Then (Q).

Case ${}^{\sigma'}_{\mathsf{f}}\mathsf{AC}[\,]_{\mathsf{g}}(\mathbf{v_{11}})$.

If $\sigma' = \forall\alpha^{\mathsf{q}}.\,\sigma''$, then $(s, \mathsf{e}) \longmapsto_M (s, {}^{\sigma'[\sigma/\alpha^{\mathsf{q}}]}_{\mathsf{f}}\mathsf{AC}_{\mathsf{g}}(\mathbf{v_{11}}[(\sigma)^{\mathscr{C}}]))$ hence (R); otherwise (Q).

(R) That is, $(s, \mathsf{e}_1) \longmapsto_M (s', \mathsf{e}_1')$.

Then $(s, \mathsf{E}_1[\mathsf{e}_1]_{\mathscr{A}}) \longmapsto_M (s', \mathsf{E}_1[\mathsf{e}_1']_{\mathscr{A}})$ by Lemma 5.6.3, hence, (R).

Case $\mathsf{e}_1\ \mathsf{e}_2$.

Consider first the induction hypothesis on at e_1, noting that $\mathsf{E}_1 = [\,]_{\mathscr{A}}\ \mathsf{e}_2$ is an evaluation context.

(Q) Then (Q).

(A) Let $\mathsf{v}_1 = \mathsf{e}_1$, and note that $\mathsf{E}_2 = \mathsf{v}_1\ [\,]_{\mathscr{A}}$ is an evaluation context. We now apply the induction hypothesis to e_2:

(Q) Then (Q).

(A) Let $\mathsf{v}_2 = \mathsf{e}_2$. Now by cases on v_1:

Case c.

By cases on c:

Case $\lceil z \rceil$.

Then (Q).

Case $(z-)$.

If $\mathsf{v}_2 = \lceil z \rceil_2$ then $(s, \mathsf{e}) \longmapsto_M (s, \lceil z - z_2 \rceil)$, hence (R); otherwise (Q).

Case $-$.

If v_2 is an integer constant $\lceil z \rceil$, then $(s, \mathsf{e}) \longmapsto_M (s, (z-))$, hence (R); otherwise (Q).

Case $\mathsf{new}[\sigma_1]$.

Then $(s, \mathsf{e}) \longmapsto_M ((s, \ell \mapsto \mathsf{v}_2), \ell)$, hence (R).

Case $\mathsf{swap}[\sigma_1][\sigma_2]$.

If $\mathsf{v}_2 = \langle \ell, \mathsf{v}_{22} \rangle$ where $s = (s_1, \ell \mapsto \mathsf{v}_{21}, s_2)$ for some s_1, s_2, and v_{21}, then $(s, \mathsf{e}) \longmapsto_M ((s_1, \ell \mapsto \mathsf{v}_{22}, s_2), \langle \mathsf{v}_{21}, \ell \rangle)$, hence (R); otherwise (Q).

Case $\Lambda\alpha^{\mathsf{q}}.\mathsf{e}_{11}$.

Then (Q).

Case $\lambda\mathsf{x}{:}\tau.\mathsf{e}_{11}$.

Then $(s, \mathsf{e}) \longmapsto_M (s, \mathsf{e}_{11}[\mathsf{v}_2/\mathsf{x}])$, hence (R).

Case ℓ.

Then (Q).

Case $\langle \mathsf{v}_1, \mathsf{v}_2 \rangle$.

Then (Q).

Case ${}^{\sigma'}_{\mathsf{f}}\mathsf{AC}[\,]_{\mathbf{g}}(\mathbf{v_{11}})$.

If $\sigma' = \sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2$, then $(s, \mathsf{e}) \longmapsto_M (s, {}^{\sigma_2}_{\mathsf{f}}\mathsf{AC}_{\mathbf{g}}(\mathbf{v_1}\ {}_{\mathbf{g}}\mathbf{CA}^{\sigma_1}_{\mathsf{f}}(\mathsf{v}_2)))$, hence (R); otherwise (Q).

(R) That is, $(s, \mathsf{e}_2) \longmapsto_M (s', \mathsf{e}'_2)$. Then by Lemma 5.6.3 with E_1, (R).

(R) That is, $(s, \mathsf{e}_1) \longmapsto_M (s', \mathsf{e}'_1)$.

Then by Lemma 5.6.3, $(s, \mathsf{E}_1[\mathsf{e}_1]_{\mathscr{A}}) \longmapsto_M (s', \mathsf{E}_1[\mathsf{e}'_1]_{\mathscr{A}})$, hence, (R).

Case $\mathsf{if0}\ \mathsf{e}_1\ \mathsf{e}_2\ \mathsf{e}_3$.

Apply induction at e_1, noting that $\mathsf{E}_1 = \mathsf{if0}[\,]_{\mathscr{A}}\ \mathsf{e}_2\ \mathsf{e}_3$ is an evaluation context:

(Q) Then (Q).

(A) If $\mathsf{e}_1 = \lceil z \rceil$ for some z, then (R) by one of the two $\mathsf{if0}$ rules; otherwise, (Q) by the definition of faulty.

(R) Then (R) in E_1, by Lemma 5.6.3.

Case $\langle \mathsf{e}_1, \mathsf{e}_2 \rangle$.

Consider first the induction hypothesis on at e_1, noting that $\mathsf{E}_1 = \langle [\,]_{\mathscr{A}}, \mathsf{e}_2 \rangle$ is an evaluation context.

(Q) Then (Q).

(A) Let $\mathsf{v}_1 = \mathsf{e}_1$, and note that $\mathsf{E}_2 = \langle \mathsf{v}_1, [\,]_{\mathscr{A}} \rangle$ is an evaluation context. We now apply the induction hypothesis to e_2:

(Q) Then (Q).

(A) Then (A), since $\langle \mathsf{v}_1, \mathsf{v}_2 \rangle$ is a value.

(R) That is, $(s, \mathsf{e}_2) \longmapsto_M (s', \mathsf{e}'_2)$. Then $(s, \mathsf{E}_2[\mathsf{e}_2]_{\mathscr{A}}) \longmapsto_M (s', \mathsf{E}_2[\mathsf{e}'_2]_{\mathscr{A}})$ by Lemma 5.6.3, hence, (R).

(R) That is, $(s, \mathsf{e}_1) \longmapsto_M (s', \mathsf{e}'_1)$.

Then $(s, \mathsf{E}_1[\mathsf{e}_1]_{\mathscr{A}}) \longmapsto_M (s', \mathsf{E}_1[\mathsf{e}'_1]_{\mathscr{A}})$ by Lemma 5.6.3, hence, (R).

Case $\mathsf{let}\ \langle \mathsf{x}, \mathsf{y} \rangle = \mathsf{e}_1\ \mathsf{in}\ \mathsf{e}_2$.

Apply induction at e_1, noting that $\mathsf{E}_1 = \mathsf{let}\ \langle \mathsf{x}, \mathsf{y} \rangle = [\,]_{\mathscr{A}}\ \mathsf{in}\ \mathsf{e}_2$ is an evaluation context:

(Q) Then (Q).

(A) If $\mathsf{e}_1 = \langle \mathsf{v}_1, \mathsf{v}_2 \rangle$ then $(s, \mathsf{e}) \longmapsto_M (s, \mathsf{e}_2[\mathsf{v}_2/\mathsf{y}][\mathsf{v}_1/\mathsf{x}])$, hence (R); otherwise (Q).

(R) That is, $(s, \mathsf{e}_1) \longmapsto_M (s', \mathsf{e}'_1)$.

Then $(s, \mathsf{E}_1[\mathsf{e}_1]_{\mathscr{A}}) \longmapsto_M (s', \mathsf{E}_1[\mathsf{e}'_1]_{\mathscr{A}})$ by Lemma 5.6.3, hence, (R).

Case $\mathbf{g}^{\mathsf{f}}$.

Because C is closed in M, we know that either

- there exists some **module** $\mathbf{g} : \tau = \mathbf{v} \in M$,
  and thus $(s, \mathbf{g}^{\mathsf{f}}) \longmapsto_M (s, {}^{(\tau)^{\mathscr{A}}}_{\mathsf{f}}\mathsf{AC}_{\mathbf{g}}(\mathbf{g}))$, or
- there exists some **interface** $\mathbf{g} :> \sigma^{\mathsf{u}} = \mathbf{g}' \in M$,
  and thus $(s, \mathbf{g}^{\mathsf{f}}) \longmapsto_M (s, {}^{\sigma^{\mathsf{u}}}_{\mathsf{f}}\mathsf{AC}_{\mathbf{g}}(\mathbf{g}'))$,

hence (R).

Case ${}^{\sigma'}_{\mathsf{f}'}\mathsf{AC}_{\mathbf{g}'}(\mathbf{e_1})$.

Apply part (i) of the induction hypothesis to $\mathbf{e_1}$, noting that $\mathsf{E}_1 = {}^{\sigma'}_{\mathsf{f}'}\mathsf{AC}_{\mathbf{g}'}([\,]_{\mathscr{C}})$ is an evaluation context:

(Q) Then (Q).

(A) Let $\mathsf{v}_1 = \mathsf{e}_1$. Then $(s, \mathsf{e}) \longmapsto_M \mathit{coerce}_{\mathscr{A}}(\cdots)$.

(R) That is, $(s, \mathbf{e_1}) \longmapsto_M (s', \mathbf{e}'_1)$.
Then $(s, \mathsf{E}_1[\mathbf{e_1}]_{\mathscr{C}}) \longmapsto_M (s', \mathsf{E}_1[\mathbf{e}'_1]_{\mathscr{C}})$ by Lemma 5.6.3, hence (R). □

Lemma 5.6.6 (Canonical Forms)**.**

(i) *For the* $\lambda_{\mathscr{C}}$ *subcalculus:*

(a) *If* $\Sigma; \Delta; \Gamma \rhd^M_{\mathscr{C}} \mathbf{v} : \mathbf{int}$ *then* $\mathbf{v} = \lceil z \rceil$ *for some* z.

(b) *If* $\Sigma; \Delta; \Gamma \rhd^M_{\mathscr{C}} \mathbf{v} : \forall\alpha.\,\tau$ *then* $\mathbf{v}$ *is either:*

- $\Lambda\alpha.\,\mathbf{e}$ *for some* $\mathbf{e}$, *or*
- ${}_{\mathbf{f}}\mathbf{CA}[\ell]^{\forall\beta^{\mathsf{q}}.\,\sigma}_{\mathsf{g}}(\mathsf{v}')$ *for some* ℓ, β^{q}, σ, $\mathbf{f}$, g, *and* v' *s.t.* $\forall\alpha.\,\tau = \forall\alpha.\,(\sigma[\{\alpha\}/\beta^{\mathsf{q}}])^{\mathscr{C}}$.

(c) *If* $\Sigma; \Delta; \Gamma \rhd^M_{\mathscr{C}} \mathbf{v} : \tau_1 \to \tau_2$ *then* $\mathbf{v}$ *is one of:*

- *The constant* $-$, *with* $\tau_1 = \mathbf{int}$ *and* $\tau_2 = \mathbf{int} \to \mathbf{int}$;
- *The constant* $(z-)$ *for some* z, *with* $\tau_1 = \tau_2 = \mathbf{int}$;
- $\lambda\mathbf{x}{:}\tau_1.\,\mathbf{e}$ *for some* $\mathbf{e}$, *or*
- ${}_{\mathbf{f}}\mathbf{CA}[\ell]^{\sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2}_{\mathsf{g}}(\mathsf{v}')$ *for some* ℓ, σ_1, σ_2, $\mathbf{f}$, g, *and* v' *such that* $\tau_1 = (\sigma_1)^{\mathscr{C}}$ *and* $\tau_2 = (\sigma_2)^{\mathscr{C}}$..

(d) *If* $\Sigma; \Delta; \Gamma \rhd^M_{\mathscr{C}} \mathbf{v} : \{\sigma\}$ *then* $\mathbf{v} = {}_{\mathbf{f}}\mathbf{CA}[\ell]^{\sigma^{\circ}}_{\mathsf{g}}(\mathsf{v}')$ *for some* ℓ, σ°, $\mathbf{f}$, g, *and* v' *such that* $\sigma = \sigma^{\circ}$.

(ii) *For the* $\lambda^{\mathscr{A}}$ *subcalculus:*

(a) *If* $\Sigma; \Delta; \Gamma \rhd^M_{\mathscr{A}} \mathsf{v} : \mathsf{int}$ *then* $\mathsf{v} = \lceil z \rceil$ *for some* z.

(b) *If* $\Sigma; \Delta; \Gamma \rhd^M_{\mathscr{A}} \mathsf{v} : \forall\alpha^{\mathsf{q}}.\,\sigma$ *then* v *is either:*

- $\Lambda\alpha^{\mathsf{q}}.\,\mathsf{e}$ *for some* e, *or*

  - ${}^{\forall\alpha^{\mathsf{q}}.\sigma}_{\mathsf{f}}\mathsf{AC}[\,]_{\mathbf{g}}(\mathbf{v}')$ *for some* ℓ, f, $\mathbf{g}$, *and* $\mathbf{v}'$.

(*c*) *If* $\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{v} : \sigma_1 \xrightarrow{\mathsf{q}}\!\circ\, \sigma_2$ *then* v *is either:*

  - *The constant* $-$, *with* $\sigma_1 = \mathsf{int}$ *and* $\sigma_2 = \mathsf{int} \xrightarrow{\mathsf{u}}\!\circ\, \mathsf{int}$;
  - *The constant* $(z-)$ *for some* z, *with* $\sigma_1 = \sigma_2 = \mathsf{int}$;
  - *The constant* $\mathsf{new}[\sigma_1]$, *with* $\sigma_2 = \sigma_1\ \mathsf{ref}$;
  - *The constant* $\mathsf{swap}[\sigma_1'][\sigma_2']$ *for some* σ_1' *and* σ_2' *such that* $\sigma_1 = \sigma_1'\ \mathsf{ref} \otimes \sigma_2'$ *and* $\sigma_2 = \sigma_1' \otimes \sigma_2'\ \mathsf{ref}$;
  - $\lambda \mathsf{x}{:}\sigma_1.\,\mathsf{e}$ *for some* e; *or*
  - ${}^{\sigma_1 \xrightarrow{\mathsf{q}}\circ \sigma_2}_{\mathsf{f}}\mathsf{AC}[\,]_{\mathbf{g}}(\mathbf{v}')$ *for some* ℓ, f, $\mathbf{g}$, *and* $\mathbf{v}'$.

(*d*) *If* $\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{v} : \sigma_1 \otimes \sigma_2$ *then* $\mathsf{v} = \langle \mathsf{v}_1, \mathsf{v}_2 \rangle$ *for some* v_1 *and* v_2.

(*e*) *If* $\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{v} : \sigma\ \mathsf{ref}$ *then* $\mathsf{v} = \ell$ *for some* ℓ.

(*f*) *If* $\Sigma; \Delta; \Gamma \rhd^{M}_{\mathscr{A}} \mathsf{v} : \{\tau\}$ *then* $\mathsf{v} = {}^{\{\tau\}}_{\mathsf{f}}\mathsf{AC}[\ell]_{\mathbf{g}}(\mathsf{v}')$ *for some* ℓ, f, $\mathbf{g}$, *and* $\mathbf{v}'$.

Proof. We exhaustively consider the values and their possible types:

(*i*) By cases on $\mathbf{v}$:

Case $\mathbf{\Lambda}\alpha.\,\mathbf{e}$.

This types only by rule RTC-TLam, which gives it a type of the form $\forall\alpha.\,\tau$. Therefore, $\mathbf{\Lambda}\alpha.\,\mathbf{e}$ is a possibility for part (*b*).

Case $\lambda\mathbf{x}{:}\tau.\,\mathbf{e}$.

This types only by rule RTC-Lam, which gives it a type of the form $\tau \to \tau'$. Therefore, $\lambda\mathbf{x}{:}\tau.\,\mathbf{e}$ is a possibility for part (*c*).

Case $\mathbf{c}$.

This types only by rule RTC-Con, which gives it type $\mathrm{ty}_{\mathscr{C}}(\mathbf{c})$. By cases on $\mathbf{c}$:

Case $\lceil z \rceil$.

Then $\mathrm{ty}_{\mathscr{C}}(\mathbf{c}) = \mathbf{int}$. Therefore, $\lceil z \rceil$ is a possibility for part (*a*).

Case $(z-)$.

Then $\mathrm{ty}_{\mathscr{C}}(\mathbf{c}) = \mathbf{int} \to \mathbf{int}$. Therefore, $(z-)$ is a possibility for part (*c*).

Case $-$.

Then $\mathrm{ty}_{\mathscr{C}}(\mathbf{c}) = \mathbf{int} \to \mathbf{int} \to \mathbf{int}$. Therefore, $-$ is a possibility for part (*c*).

Case ${}_{\mathbf{f}}\mathbf{CA}[\ell]^{\sigma}_{\mathsf{g}}(\mathsf{v}')$.

This types only by rules RTC-Sealed, RTC-Blessed, and RTC-Defunct, each of which requires that σ be a σ^{w} type. Because σ^{w} is one of $\forall\alpha^{\mathsf{q}}.\,\sigma$, $\sigma_1 \xrightarrow{\mathsf{q}}\!\circ\, \sigma_2$, or σ°, the type of the value must be one of $\forall\beta.(\sigma[\{\beta\}/\alpha^{\mathsf{q}}])^{\mathscr{C}}$, $(\sigma_1)^{\mathscr{C}} \to (\sigma_2)^{\mathscr{C}}$, or $\{\sigma^{\circ}\}$. Therefore, ${}_{\mathbf{f}}\mathbf{CA}[\ell]^{\sigma}_{\mathsf{g}}(\mathsf{v}')$ is a possibility for parts (*b*), (*c*), and (*d*).

(*ii*) In the $\lambda^{\mathscr{A}}$ subcalculus, besides the rules mentioned for each syntactic form, each may type by RTA-Subsume with the syntax-specific rule proving the antecedant to the subsumption. We merely note that subsumption relates only types that are the same but for potentially different qualifiers q on each function type constructor ($\overset{\mathsf{q}}{\multimap}$), which we do not distinguish in this lemma.

By cases on v:

Case $\Lambda\alpha^{\mathsf{q}}.\,\mathsf{e}$.

This types only by rule RTA-TLam, which gives it a type of the form $\forall\alpha^{\mathsf{q}}.\,\sigma$. Therefore, $\Lambda\alpha^{\mathsf{q}}.\,\mathsf{e}$ is a possibility for part (*b*).

Case $\lambda\mathsf{x}{:}\sigma.\,\mathsf{e}$.

This types only by rule RTA-Lam, which gives it a type of the form $\sigma \overset{\mathsf{u}}{\multimap} \sigma'$. Therefore, $\lambda\mathsf{x}{:}\sigma.\,\mathsf{e}$ is a possibility for part (*c*).

Case c.

This types only by rule RTA-Con, which gives it type $\mathrm{ty}_{\mathscr{A}}(\mathsf{c})$. By cases on c:

Case $\lceil z \rceil$.

Then $\mathrm{ty}_{\mathscr{A}}(\mathsf{c}) = \mathsf{int}$. Therefore, $\lceil z \rceil$ is a possibility for part (*a*).

Case $(z-)$.

Then $\mathrm{ty}_{\mathscr{A}}(\mathsf{c}) = \mathsf{int} \overset{\mathsf{u}}{\multimap} \mathsf{int}$. Therefore, $(z-)$ is a possibility for part (*c*).

Case $-$.

Then $\mathrm{ty}_{\mathscr{A}}(\mathsf{c}) = \mathsf{int} \overset{\mathsf{u}}{\multimap} \mathsf{int} \overset{\mathsf{u}}{\multimap} \mathsf{int}$. Therefore, $-$ is a possibility for part (*c*).

Case $\mathsf{new}[\sigma_1]$.

Then $\mathrm{ty}_{\mathscr{A}}(\mathsf{new}[\sigma_1]) = \sigma_1 \overset{\mathsf{u}}{\multimap} \sigma_1\ \mathsf{ref}$. Therefore, $\mathsf{new}[\sigma_1]$ is a possibility for part (*c*).

Case $\mathsf{swap}[\sigma_1][\sigma_2]$.

Then $\mathrm{ty}_{\mathscr{A}}(\mathsf{swap}[\sigma_1][\sigma_2]) = (\sigma_1\ \mathsf{ref} \otimes \sigma_2) \overset{\mathsf{u}}{\multimap} (\sigma_1 \otimes \sigma_2\ \mathsf{ref})$. Therefore, $\mathsf{swap}[\sigma_1][\sigma_2]$ is a possibility for part (*c*).

Case $\langle \mathsf{v}_1, \mathsf{v}_2 \rangle$.

This types only by rule RTA-Pair, which gives it a type of the form $\sigma_1 \otimes \sigma_2$. Therefore, $\langle \mathsf{v}_1, \mathsf{v}_2 \rangle$ is a possibility for part (*d*).

Case ℓ.

This types only by rule RTA-Loc, which gives it a type of the form $\sigma\ \mathsf{ref}$. Therefore, ℓ is a possibility for part (*e*).

Case ${}^{\sigma}_{\mathsf{f}}\mathsf{AC}[\,]_{\mathbf{g}}(\mathbf{v}')$.

This types only by rule RTA-Sealed, which requires that $(\sigma)^{\mathscr{C}}$ be a $\tau^{\mathbf{w}}$ type. Because $\tau^{\mathbf{w}}$ is one of $\forall\alpha.\,\tau$, $\tau_1 \to \tau_2$, or $\tau^{\mathbf{o}}$, the type of the value must be one of $\forall\beta^{\mathsf{q}}.(\sigma[\{\beta^{\mathsf{q}}\}/\alpha])^{\mathscr{A}}$, $(\tau_1)^{\mathscr{A}} \overset{\mathsf{u}}{\multimap} (\tau_2)^{\mathscr{A}}$, or $\{\tau^{\mathbf{o}}\}$. Therefore, ${}^{\sigma}_{\mathsf{f}}\mathsf{AC}[\,]_{\mathbf{g}}(\mathbf{v}')$ is a possibility for parts (*b*), (*c*), and (*f*). □

Lemma 5.6.7 (Faulty expressions are ill-typed)**.**

(*i*) *If* $\mathbf{e} \in \mathbf{Q}_s$ *is faulty with respect to* s*, then there exist no* M*,* Σ_1*,* Σ_2*, and* τ *such that*

- $\Sigma_1 \rhd^M s : \Sigma_1 \boxplus \Sigma_2$ *and*
- $\Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{e} : \tau$.

(*ii*) *If* e *is faulty with respect to* s*, then there exist no* M*,* Σ_1*,* Σ_2*, and* σ *such that*

- $\Sigma_1 \rhd^M s : \Sigma_1 \boxplus \Sigma_2$ *and*
- $\Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{e} : \sigma$.

Proof by contradiction. We proceed by mutual induction on the structure of $\mathbf{Q}_s$ and Q_s.

(*i*) Suppose that $\Sigma_1; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{Q}_s : \tau$. Then by cases on $\mathbf{Q}_s$:

Case $\mathbf{Q}^{\mathbf{\Lambda}}_s[\tau_{\mathbf{a}}]$.

This types only by rule RTC-TAPP, which requires that $\mathbf{Q}^{\mathbf{\Lambda}}_s$ have a type $\forall\alpha.\,\tau_{\mathbf{b}}$.

By Lemma 5.6.6, we see that the only values with such a type are $\mathbf{\Lambda}\alpha.\,\mathbf{e}$ and ${}_{\mathbf{f}}\mathbf{CA}[\ell]^{\forall\beta^{\mathsf{q}}.\,\sigma}_{\mathsf{g}}(\mathsf{v}')$, neither of which is an instance of $\mathbf{Q}^{\mathbf{\Lambda}}_s$, contradicting our assumption.

Case $\mathbf{Q}^{\lambda}_{s,\mathbf{v}}\ \mathbf{v}$.

This types only by rule RTC-APP, which requires that $\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{Q}^{\lambda}_{s,\mathbf{v}} : \tau' \to \tau$ and $\Sigma_{22}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{v} : \tau'$, where $\Sigma_{21} \boxplus \Sigma_{22} = \Sigma_2$.

By cases on $\mathbf{Q}^{\lambda}_{s,\mathbf{v}}$:

Case $\lceil z \rceil$.

This does not have type $\tau' \to \tau$.

Case $\mathbf{\Lambda}\alpha.\,\mathbf{e}$.

This does not have type $\tau' \to \tau$.

Case $-$ where $\mathbf{v} \neq \lceil z_2 \rceil$.

This has type $\mathbf{int} \to (\mathbf{int} \to \mathbf{int})$, which means that $\mathbf{v}$ must have type $\mathbf{int}$.

By Lemma 5.6.6, $\mathbf{v}$ must be an integer constant, which contradicts the side condition.

Case $(z-)$ where $\mathbf{v} \neq \lceil z_2 \rceil$.

This has type $\mathbf{int} \to \mathbf{int}$, which means that $\mathbf{v}$ must have type $\mathbf{int}$.

By Lemma 5.6.6, $\mathbf{v}$ must be an integer constant, which contradicts the side condition.

Case ${}_{\mathbf{f}}\mathbf{CA}[\ell]^{\sigma}_{\mathsf{g}}(\mathsf{v}')$ where $\sigma \neq \sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2$.

This has type $(\sigma)^{\mathscr{C}}$, which must equal $\tau' \to \tau$.

This can be the case only if $\sigma = \sigma_1 \overset{\mathsf{q}}{\multimap} \sigma_2$, which contradicts the side condition.

Case $\mathbf{if0}\,\mathbf{v}\,\mathbf{e_t}\,\mathbf{e_f}$ where $\mathbf{v} \neq \lceil z \rceil$.

The types only by rule RTC-IF0, which requires that $\mathbf{v}$ have type $\mathbf{int}$.

By Lemma 5.6.6, $\mathbf{v} = \lceil z \rceil$ for some integer z, which contradicts the side condition.

Case $\mathbf{E}[\mathbf{Q}'_s]_{\mathscr{C}}$.

By our assumption, $\Sigma_1 \rhd^M s : \Sigma_1 \boxplus \Sigma_2$ and $\Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}[\mathbf{Q}'_s]_{\mathscr{C}} : \tau$.

Then by Lemma 5.4.2 (i), $\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{Q}'_s : \tau'$ for some $\Sigma_{21} \boxplus \Sigma_{22} = \Sigma_2$.

By weakening, then, $\Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{Q}'_s : \tau'$, but by the induction hypothesis (i), this cannot be so.

Case $\mathbf{E}[\mathsf{Q}'_s]_{\mathscr{A}}$.

By our assumption, $\Sigma_1 \rhd^M s : \Sigma_1 \boxplus \Sigma_2$ and $\Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathbf{E}[\mathsf{Q}'_s]_{\mathscr{A}} : \tau$.

Then by Lemma 5.4.2 (ii), $\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathsf{Q}'_s : \sigma'$ for some $\Sigma_{21} \boxplus \Sigma_{22} = \Sigma_2$.

By weakening, then, $\Sigma_2; \cdot; \cdot \rhd^M_{\mathscr{C}} \mathsf{Q}'_s : \sigma'$, but by the induction hypothesis (ii), this cannot be so.

(ii) Suppose that $\Sigma_1; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{Q}_s : \sigma$.

Then by cases on Q_s:

Case $\mathsf{Q}^{\Lambda}_s[\sigma_{\mathsf{a}}]$.

This types only by rule RTA-TAPP, which requires that Q^{Λ}_s have a type $\forall \alpha^{\mathsf{q}}.\, \tau_{\mathsf{b}}$. By Lemma 5.6.6, we see that the only values with such a type are $\Lambda\alpha^{\mathsf{q}}.\, \mathsf{e}$ and ${}_{\mathbf{f}}\mathbf{CA}[\ell]^{\forall\alpha^{\mathsf{q}}.\,\sigma}_{\mathbf{g}}(\mathsf{v}')$, neither of which is an instance of Q^{Λ}_s, contradicting our assumption.

Case $\mathsf{Q}^{\lambda}_{s,\mathsf{v}}\ \mathsf{v}$.

This types only by rule RTA-APP, which requires that $\Sigma_{21}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{Q}^{\lambda}_{s,\mathsf{v}} : \sigma' \overset{\mathsf{q}}{\multimap} \sigma$ and $\Sigma_{22}; \cdot; \cdot \rhd^M_{\mathscr{A}} \mathsf{v} : \sigma'$ where $\Sigma_{21} \boxplus \Sigma_{22} = \Sigma_2$.

By cases on $\mathsf{Q}^{\lambda}_{s,\mathsf{v}}$:

Case $\lceil z \rceil$.

This does not have type $\tau' \overset{\mathsf{u}}{\multimap} \tau$.

Case ℓ.

This does not have type $\tau' \overset{\mathsf{u}}{\multimap} \tau$.

Case $\langle \mathsf{v}_1, \mathsf{v}_2 \rangle$.

This does not have type $\tau' \overset{\mathsf{u}}{\multimap} \tau$.

Case $\Lambda\alpha.\, \mathsf{e}$.

This does not have type $\tau' \overset{\mathsf{u}}{\multimap} \tau$.

Case $-$ where $\mathsf{v} \neq \lceil z_2 \rceil$.

This has type $\mathsf{int} \overset{\mathsf{u}}{\multimap} (\mathsf{int} \overset{\mathsf{u}}{\multimap} \mathsf{int})$, which means that v must have type int.

By Lemma 5.6.6, v must be an integer constant, which contradicts the side condition.

Case $(z-)$ where $\mathsf{v} \neq \lceil z_2 \rceil$.

This has type $\mathsf{int} \overset{\mathsf{u}}{\multimap} \mathsf{int}$, which means that v must have type int.

By Lemma 5.6.6, v must be an integer constant, which contradicts the side condition.

Case $\mathsf{swap}[\sigma_1][\sigma_2]$ where $\neg\exists \ell' \in \operatorname{dom} s$ s.t. $\mathsf{v} = \langle \ell', \mathsf{v}'' \rangle$.

Then $\sigma = \sigma_1 \otimes \sigma_2\ \mathsf{ref}$ and $\sigma' = \sigma_1\ \mathsf{ref} \otimes \sigma_2$.

By Lemma 5.6.6, twice, v has the latter type only if it is a pair $\mathsf{v} = \langle \ell, \mathsf{v}' \rangle$.

Then $Q^{\lambda}_{s,v}\ v$ only types with a derivation of the form

$$\dfrac{\dfrac{\vdots}{\cdots \rhd^{M}_{\mathscr{A}} \mathsf{swap}[\sigma_1][\sigma_2] : \cdots} \qquad \dfrac{\Sigma_2'|_u, \ell{:}\sigma_1 \rhd^{M}_{\mathscr{A}} \ell : \sigma_1\ \mathsf{ref} \qquad \Sigma_2' \rhd^{M}_{\mathscr{A}} \mathsf{v}' : \sigma_2}{\Sigma_2', \ell{:}\sigma_1; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \langle \ell, \mathsf{v}' \rangle : \sigma_1\ \mathsf{ref} \otimes \sigma_2}}{\Sigma_2', \ell{:}\sigma_1; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{swap}[\sigma_1][\sigma_2]\ \langle \ell, \mathsf{v}' \rangle : \sigma_1 \otimes \sigma_2\ \mathsf{ref}},$$

where $\Sigma_2 = \Sigma_2', \ell{:}\sigma_1$.

Furthermore, the induction hypothesis states that $\Sigma_1 \rhd^{M} s : \Sigma_1 \boxplus \Sigma_2$, that is, $\Sigma_1 \rhd^{M} s : \Sigma_1 \boxplus \Sigma_2', \ell{:}\sigma_1$. By inversion of S-ALoc, this can be the case only if $s = s' \uplus \{\ell \mapsto \mathsf{v}_1\}$ for some v_1. This contradicts the side condition on s.

Case ${}^{\sigma''}_{\mathsf{f}}\mathsf{AC}[\,]_{\mathbf{g}}(\mathbf{v}')$ where $\sigma'' \neq \sigma_1' \overset{\mathsf{q}}{\multimap} \sigma_2'$.

This has type σ'', which must equal $\sigma' \overset{\mathsf{q}}{\multimap} \sigma$, which contradicts the side condition.

Case $\mathsf{if0}\ \mathsf{v}\ \mathsf{e_t}\ \mathsf{e_f}$ where $\mathsf{v} \neq \lceil z \rceil$.

This types only by rules RTA-If0, which requires that v have type int.

By Lemma 5.6.6, $\mathsf{v} = \lceil z \rceil$ for some integer z, which contradicts the side condition.

Case $\mathsf{let}\ \langle \mathsf{x}_1, \mathsf{x}_2 \rangle = \mathsf{v}\ \mathsf{in}\ \mathsf{e}$ where $\mathsf{v} \neq \langle \mathsf{v}_1, \mathsf{v}_2 \rangle$.

This types only by rules RTA-Let, which requires that v have a type $\sigma_1 \otimes \sigma_2$.

By Lemma 5.6.6, $\mathsf{v} = \langle \mathsf{v}_1, \mathsf{v}_2 \rangle$, which contradicts the side condition.

Case $\mathsf{E}[\mathsf{Q}_s']_{\mathscr{A}}$.

By our assumption, $\Sigma_1 \rhd^{M} s : \Sigma_1 \boxplus \Sigma_2$ and $\Sigma_2; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{E}[\mathsf{Q}_s']_{\mathscr{A}} : \sigma$. Then by Lemma 5.4.2 (*iii*), $\Sigma_{21}; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{Q}_s' : \sigma'$ for some $\Sigma_{21} \boxplus \Sigma_{22} = \Sigma_2$.

By weakening, then, $\Sigma_2; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{Q}_s' : \sigma'$, but by the induction hypothesis (*ii*), this cannot be so.

Case $\mathsf{E}[\mathbf{Q}_s']_{\mathscr{C}}$.

By our assumption, $\Sigma_1 \rhd^{M} s : \Sigma_1 \boxplus \Sigma_2$ and $\Sigma_2; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathsf{E}[\mathbf{Q}_s']_{\mathscr{C}} : \sigma$.

Then by Lemma 5.4.2 (*iv*), $\Sigma_{21}; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathbf{Q}_s' : \tau'$ for some $\Sigma_{21} \boxplus \Sigma_{22} = \Sigma_2$.

By weakening, then, $\Sigma_2; \cdot; \cdot \rhd^{M}_{\mathscr{A}} \mathbf{Q}_s' : \tau'$, but by the induction hypothesis (*i*), this cannot be so. □

Theorem 5.6.8 (Progress). *If $\rhd^M C : \tau$, then either C is an answer or there exists some C' such that $C \longmapsto_M C'$.*

Proof. Since C types, it is closed; thus, by Lemma 5.6.5, it is an answer, it takes a step, or it is faulty. Because it types, we can eliminate the faulty case by Lemma 5.6.7. □

5.7 Type Soundness

Main Theorem (Type Soundness). *If $\vdash M\ \mathbf{e} : \tau$ and $(\{\}, \mathbf{e}) \longmapsto_M^* C$ such that configuration C cannot take another step, then C is an answer with $\rhd^M C : \tau$.*

Proof. By Theorems 5.3.2 (Programs to configurations), 5.6.8 (Progress), and 5.5.3 (Preservation), and induction on the length of the reduction sequence. □

6 Conclusion

Our work is part of an ongoing program to investigate practical aspects of substructural type systems, and this paper describes one step in that program. Here, we have focused on the problem of interaction between substructural and non-substructural code, each governed by its own type system, and explored the use of higher-order contracts to prevent the conventional language from breaking the substructural language's invariants. Our answer to the problem at hand naturally raises more questions.

Exceptions. In a production language with a contract system, contract violations should not always terminate the program. Real programs may catch an exception and either try to mitigate the condition that caused it, try something easier instead, or report an error and go on with some other task. To ensure soundness, it suffices to prevent the questionable actions from occurring.

On one hand, we believe that ML-style exceptions should not provide too much difficulty in an affine setting. In our prototype, *try-with* expressions are multiplicative, in the sense that the type environment needs to be split between an expression and its exception handler, not given in whole to both.

On the other hand, we do not know how exceptions or any sort of blame might work in a linear setting—this is one reason why we chose an affine calculus. Terminating the program is problematic because of the implicit discarding of linear values, but catching an exception once part of a continuation containing linear values has been discarded seems even worse. Exceptions in linear languages remain an open question.

Linearity. Our work emphasizes contract-based interaction with affine type systems rather than linear type systems because it remains unclear to us what linear contracts ought to mean. We may want a conventional language to interoperate with a language that (at least sometimes) prohibits discarding values. However, unlike affine guarantees, which are safety properties, relevance guarantees—that a value is used at some point in the future—are a form of liveness property.

One approximation is to consider a contract representing a relevance guarantee to be violated if at any point we can determine that the contract necessarily will be violated. Detecting the violation of such a liveness property is undecidable in general, but tracing garbage collection approximates a liveness property very close to the one we desire. In an idealized semantics, we might garbage collect the store after each reduction step and signal a violation if the seal location of a not-yet-used linear value has become unreachable. In a real implementation, finalizers on linear values could detect discarding. If we detect a violation, we probably could do nothing to prevent it, but at worst we could file a bug report.

Our work suggests that adding substructural libraries to a conventional programming language such as ML does not require a particularly complicated implementation, and our results yield a realistic contract-based design.

Acknowledgments. We wish to thank Daniel Brown, Ryan Culpepper, Jed Davis, Matthias Felleisen, Alec Heller, Sam Tobin-Hochstadt, Aaron Turon, and the anonymous referees for their helpful comments, discussion, and corrections. This research was supported in part by AFOSR grant FA9550-09-1-0110.

References

A. Ahmed, M. Fluet, and G. Morrisett. $\mathbf{L}^3$: A linear language with locations. Technical Report TR-24-04, Harvard University, 2004.

E. Barendsen and S. Smetsers. Uniqueness typing for functional languages with graph rewriting semantics. *Mathematical Structures in Computer Science*, 6(6), 1996.

P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. In *CSL'94*, number 933 in LNCS, pages 121–135. Springer-Verlag, 1995.

R. B. Findler and M. Felleisen. Contracts for higher-order functions. In *ICFP'02*, pages 48–59. ACM Press, 2002.

C. Flanagan. Hybrid type checking. In *POPL'06*, volume 41, pages 245–256. ACM Press, 2006.

S. J. Gay and M. J. Hole. Types and subtypes for client-server interactions. In *ESOP'09*, volume 1576 of *LNCS*, pages 74–90. Springer-Verlag, 1999.

J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris VI, 1972.

T. Jim, G. Morrisett, D. Grossman, M. Hicks, J. Cheney, and Y. Wang. Cyclone: A safe dialect of C. In *Proc. USENIX Annual Technical Conference*, 2002.

J. Matthews and R. B. Findler. Operational semantics for multi-language programs. In *POPL'07*, volume 42, pages 3–10. ACM Press, 2007.

R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML*. MIT Press, revised edition, 1997.

G. Plotkin. Type theory and recursion. *LICS'93*, 1993.

J. C. Reynolds. Towards a theory of type structure. In *Proc. Colloque sur la Programmation*, volume 19 of *LNCS*, pages 408–425. Springer-Verlag, 1974.

J. G. Siek and W. Taha. Gradual typing for functional languages. In *Workshop on Scheme and Functional Programming*, pages 81–92. ACM Press, 2006.

W. R. Stevens. *UNIX Network programming*. Prentice-Hall, 1990.

R. Strom and S. Yemini. Typestate: A programming language concept for enhancing software reliability. *IEEE Transactions on Software Engineering*, 12(1), 1986.

S. Tobin-Hochstadt and M. Felleisen. Interlanguage migration: From scripts to programs. In *OOPSLA'06*, pages 964–974. ACM Press, 2006.

S. Tobin-Hochstadt and M. Felleisen. The design and implementation of Typed Scheme. In *POPL'07*, pages 395–406. ACM Press, 2008.

D. N. Turner, P. Wadler, and C. Mossin. Once upon a type. In *FPCA'95*, pages 1–11. ACM Press, 1995.

P. Wadler. Linear types can change the world. In *Programming Concepts and Methods*, pages 347–359. North Holland, 1990.

D. Walker. Substructural type systems. In B. C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 1, pages 3–44. MIT Press, 2005.

A. K. Wright and M. Felleisen. A syntactic approach to type soundness. *Information and Computation*, 115(1):38–94, 1994.

List of Figures

A The Affine Sockets Library

This is the full code listing for the sockets library from §2. It includes the details of error handing that we omit from the shorter presentation.

When we raise an exception, we "freeze" the capability. We can thaw the frozen capability if we have the socket that it goes with. (This requires a dynamic check.) This lets us recover the capability with a type paramater that matches any extant sockets that go with it:

```
module ASocket = struct[A]
  module S = Socket
  let getAddrByName = S.getAddrByName

  abstype α socket  = Sock of {S.socket}
      and α initial    qualifier A = Initial
      and α bound      qualifier A = Bound
      and α listening  qualifier A = Listening
      and α connected  qualifier A = Connected
  with
    abstype frozenInitial   qualifier A = FInitial   of {S.socket}
        and frozenBound     qualifier A = FBound     of {S.socket}
        and frozenListening qualifier A = FListening of {S.socket}
        and frozenConnected qualifier A = FConnected of {S.socket}
    with
      let freezeInitial[α] (Sock sock: α socket) (_: α initial) =
            FInitial sock

      let thawInitial[α] (Sock sock: α socket)
                         (FInitial sock': frozenInitial) =
        if sock == sock'
          then Right[frozenInitial, α initial] Initial[α]
          else Left [frozenInitial, α initial] (FInitial sock')

      let freezeBound[α] (Sock sock: α socket) (_: α bound) =
        FBound sock

      let thawBound[α] (Sock sock: α socket)
                       (FBound sock': frozenBound) =
        if sock == sock'
          then Right[frozenBound, α bound] Bound[α]
          else Left [frozenBound, α bound] (FBound sock')

      let freezeListening[α] (Sock sock: α socket) (_: α listening) =
        FListening sock

      let thawListening[α] (Sock sock: α socket)
                           (FListening sock': frozenListening) =
        if sock == sock'
          then Right[frozenListening, α listening] Listening[α]
          else Left [frozenListening, α listening] (FListening sock')

      let freezeConnected[α] (Sock sock: α socket) (_: α connected) =
        FConnected sock

      let thawConnected[α] (Sock sock: α socket)
                           (FConnected sock': frozenConnected) =
        if sock == sock'
          then Right[frozenConnected, α connected] Connected[α]
          else Left [frozenConnected, α connected] (FConnected sock')
    end

    exception SocketError    of string
    exception StillInitial   of frozenInitial × string
    exception StillBound     of frozenBound × string
    exception StillListening of frozenListening × string
    exception StillConnected of frozenConnected × string

    let socket (): ∃α. α socket × α initial =
      try
        let sock = S.socket ()
         in Pack(unit, Sock[unit] sock, Initial[unit])
      with
        IOError s → raise (SocketError s)

    let bind[α] (Sock sock as s: α socket) (port: int) (cap: α initial)
                : α bound =
      try
        S.bind sock port;
        Bound[α]
      with
        IOError msg → raise (StillInitial (freezeInitial s cap, msg))

    let connect[α] (Sock sock as s: α socket) (host: string)
                   (port: string) (cap: α initial + α bound)
                   : α connected =
      try
        S.connect sock host port;
        Connected[α]
      with
        IOError msg → match cap with
          | Left cap  → raise
                           (StillInitial (freezeInitial s cap, msg))
          | Right cap → raise (StillBound (freezeBound s cap, msg))

    let listen[α] (Sock sock as s: α socket) (cap: α bound)
                  : α listening =
      try
        S.listen sock;
        Listening[α]
      with
        IOError msg → raise (StillBound (freezeBound s cap, msg))

    let accept[α] (Sock sock as s: α socket) (cap: α listening)
                  : (∃'s. 's socket × 's connected) × α listening =
      try
        let newsock = S.accept sock in
          (Pack(unit, Sock[unit] newsock, Connected[unit]),
           Listening[α])
      with
        IOError msg → raise
                         (StillListening (freezeListening s cap, msg))

    let send[α] (Sock sock: α socket) (data: string) (_: α connected)
                : α connected =
      try
        S.send sock data;
        Connected[α]
      with
        IOError msg → raise (SocketError msg)

    let recv[α] (Sock sock: α socket) (len: int) (_: α connected)
                : string × α connected =
      try
```

```
        let str = S.recv sock len
         in (str, Connected[α])
      with
        IOError msg → raise (SocketError msg)

    let close[α] (Sock sock: α socket) (_: α connected): unit =
      try
        S.close sock
      with
        IOError s → raise (SocketError s)
  end

  let catchInitial[α,β^a] (sock: α socket) (body: unit ⊸^a β^a)
                          (handler: α initial ⊸^a β^a) =
    try body () with
    | StillInitial (frz, msg) →
        match thawInitial sock frz with
        | Left frz  → raise (StillInitial (frz, msg))
        | Right cap → handler cap

  let catchBound[α,β^a] (sock: α socket) (body: unit ⊸^a β^a)
                        (handler: α bound ⊸^a β^a) =
    try body () with
    | StillBound (frz, msg) →
        match thawBound sock frz with
        | Left frz  → raise (StillBound (frz, msg))
        | Right cap → handler cap

  let catchListening[α,β^a] (sock: α socket) (body: unit ⊸^a β^a)
                            (handler: α listening ⊸^a β^a) =
    try body () with
    | StillListening (frz, msg) →
        match thawListening sock frz with
        | Left frz  → raise (StillListening (frz, msg))
        | Right cap → handler cap

  let catchConnected[α,β^a] (sock: α socket) (body: unit ⊸^a β^a)
                            (handler: α connected ⊸^a β^a) =
    try body () with
    | StillConnected (frz, msg) →
        match thawConnected sock frz with
        | Left frz  → raise (StillConnected (frz, msg))
        | Right cap → handler cap
end
```

B Semantics of $\lambda_{\mathscr{C}}$

The syntax of $\lambda_{\mathscr{C}}$ may be found in figure 4.1.

$$\boxed{\Delta \vdash_{\mathscr{C}} \tau}$$

$$\textsc{CC-Int}\ \frac{}{\Delta \vdash_{\mathscr{C}} \mathbf{int}} \qquad \textsc{CC-Arr}\ \frac{\Delta \vdash_{\mathscr{C}} \tau_1 \qquad \Delta \vdash_{\mathscr{C}} \tau_2}{\Delta \vdash_{\mathscr{C}} \tau_1 \to \tau_2} \qquad \textsc{CC-All}\ \frac{\Delta, \alpha \vdash_{\mathscr{C}} \tau}{\Delta \vdash_{\mathscr{C}} \forall \alpha.\, \tau} \qquad \textsc{CC-Var}\ \frac{\alpha \in \Delta}{\Delta \vdash_{\mathscr{C}} \alpha}$$

Figure B.1: Statics of $\lambda_{\mathscr{C}}$: types (i)

$$\boxed{\Delta; \Gamma \vdash^{M}_{\mathscr{C}} \mathbf{e} : \tau}$$

$$\textsc{TC-TLam}\ \frac{\Delta, \alpha; \Gamma \vdash^{M}_{\mathscr{C}} \mathbf{v} : \tau}{\Delta; \Gamma \vdash^{M}_{\mathscr{C}} \Lambda\alpha.\, \mathbf{v} : \forall \alpha.\, \tau} \qquad \textsc{TC-TApp}\ \frac{\Delta; \Gamma \vdash^{M}_{\mathscr{C}} \mathbf{e} : \forall \alpha.\, \tau' \qquad \Delta \vdash_{\mathscr{C}} \tau}{\Delta; \Gamma \vdash^{M}_{\mathscr{C}} \mathbf{e}[\tau] : \tau'[\tau/\alpha]}$$

$$\textsc{TC-Lam}\ \frac{\Delta; \Gamma, \mathbf{x} : \tau \vdash^{M}_{\mathscr{C}} \mathbf{e} : \tau' \qquad \Delta \vdash_{\mathscr{C}} \tau}{\Delta; \Gamma \vdash^{M}_{\mathscr{C}} \lambda \mathbf{x}{:}\tau.\, \mathbf{e} : \tau \to \tau'} \qquad \textsc{TC-App}\ \frac{\Delta; \Gamma \vdash^{M}_{\mathscr{C}} \mathbf{e_1} : \tau' \to \tau \qquad \Delta; \Gamma \vdash^{M}_{\mathscr{C}} \mathbf{e_2} : \tau'}{\Delta; \Gamma \vdash^{M}_{\mathscr{C}} \mathbf{e_1}\ \mathbf{e_2} : \tau}$$

$$\textsc{TC-Con}\ \frac{}{\Delta; \Gamma \vdash^{M}_{\mathscr{C}} \mathbf{c} : \mathrm{ty}_{\mathscr{C}}(\mathbf{c})} \qquad \textsc{TC-If0}\ \frac{\Delta; \Gamma \vdash^{M}_{\mathscr{C}} \mathbf{e_1} : \mathbf{int} \qquad \Delta; \Gamma \vdash^{M}_{\mathscr{C}} \mathbf{e_2} : \tau \qquad \Delta; \Gamma \vdash^{M}_{\mathscr{C}} \mathbf{e_3} : \tau}{\Delta; \Gamma \vdash^{M}_{\mathscr{C}} \mathbf{if0}\, \mathbf{e_1}\ \mathbf{e_2}\ \mathbf{e_3} : \tau}$$

$$\textsc{TC-Var}\ \frac{\Gamma(\mathbf{x}) = \tau}{\Delta; \Gamma \vdash^{M}_{\mathscr{C}} \mathbf{x} : \tau} \qquad \textsc{TC-Mod}\ \frac{\mathbf{module\, f} : \tau = \mathbf{v} \in M \qquad \cdot \vdash_{\mathscr{C}} \tau}{\Delta; \Gamma \vdash^{M}_{\mathscr{C}} \mathbf{f} : \tau}$$

$$\boxed{\mathrm{ty}_{\mathscr{C}}(\mathbf{c}) = \tau}$$

$$\mathrm{ty}_{\mathscr{C}}(-) = \mathbf{int} \to \mathbf{int} \to \mathbf{int} \qquad \mathrm{ty}_{\mathscr{C}}((z-)) = \mathbf{int} \to \mathbf{int} \qquad \mathrm{ty}_{\mathscr{C}}(\lceil z \rceil) = \mathbf{int}$$

Figure B.2: Statics of $\lambda_{\mathscr{C}}$: expressions and constants (ii)

$\boxed{\vdash P}$, $\boxed{\vdash^M \mathbf{m}\ \text{okay}}$

$$\textsc{Prog-C}\quad \frac{(\forall m \in M) \vdash^M m\ \text{okay} \qquad \cdot;\cdot \vdash^M_{\mathscr{C}} \mathbf{e} : \tau}{\vdash M\ \mathbf{e} : \tau} \qquad\qquad \text{TM-C}\quad \frac{\cdot;\cdot \vdash^M_{\mathscr{C}} \mathbf{v} : \tau}{\vdash^M \mathbf{module\ f} : \tau = \mathbf{v}\ \text{okay}}$$

Figure B.3: Statics of $\lambda_{\mathscr{C}}$: programs and modules (iii)

$$\begin{array}{rrcl}
\textit{evaluation contexts} & \mathbf{E} & ::= & [\,]_{\mathscr{C}} \mid \mathbf{E}[\tau] \mid \mathbf{E}\ \mathbf{e} \mid \mathbf{v}\ \mathbf{E} \mid \mathbf{if0}\ \mathbf{E}\ \mathbf{e}\ \mathbf{e} \\
\textit{configurations} & C & ::= & (s, \mathbf{e}) \\
\textit{stores} & s & ::= & \cdots
\end{array}$$

$\boxed{C \longmapsto_M C}$, $\boxed{\delta_{\mathscr{C}}(s, \mathbf{c}, \mathbf{v}) = (s, \mathbf{v})}$

$$\begin{array}{llll}
(\text{C-}\delta) & (s, \mathbf{c}\ \mathbf{v}) & \underset{M}{\longmapsto} & \delta_{\mathscr{C}}(s, \mathbf{c}, \mathbf{v}) & \\
(\text{C-}B) & (s, (\mathbf{\Lambda}\alpha.\,\mathbf{v})[\tau]) & \underset{M}{\longmapsto} & (s, \mathbf{v}[\tau/\alpha]) & \\
(\text{C-}\beta) & (s, (\lambda\mathbf{x}{:}\tau.\,\mathbf{e})\ \mathbf{v}) & \underset{M}{\longmapsto} & (s, \mathbf{e}[\mathbf{v}/\mathbf{x}]) & \\
(\text{C-}\textsc{If0}) & (s, \mathbf{if0}\ \lceil 0 \rceil\ \mathbf{e_t}\ \mathbf{e_f}) & \underset{M}{\longmapsto} & (s, \mathbf{e_t}) & \\
(\text{C-}\textsc{IfZ}) & (s, \mathbf{if0}\ \lceil z \rceil\ \mathbf{e_t}\ \mathbf{e_f}) & \underset{M}{\longmapsto} & (s, \mathbf{e_f}) & z \neq 0 \\
(\text{C-}\textsc{Mod}) & (s, \mathbf{f}) & \underset{M}{\longmapsto} & (s, \mathbf{v}) & (\mathbf{module\ f} : \tau = \mathbf{v}) \in M \\
(\text{C-}\textsc{Cxt}) & (s, \mathbf{E}[\mathbf{e}]_{\mathscr{C}}) & \underset{M}{\longmapsto} & (s', \mathbf{E}[\mathbf{e}']_{\mathscr{C}}) & \text{if } (s, \mathbf{e}) \underset{M}{\longmapsto} (s', \mathbf{e}')
\end{array}$$

$$\delta_{\mathscr{C}}(s, -, \lceil z \rceil) = (s, (z-))$$
$$\delta_{\mathscr{C}}(s, (z_1-), \lceil z_2 \rceil) = (s, \lceil z_1 - z_2 \rceil)$$

Figure B.4: Dynamics of $\lambda_{\mathscr{C}}$