Type Safe Evolution of Live Systems

Miguel Domingues João Costa Seco
miguel@domingues.pt joao.seco@fct.unl.pt
NOVA-LINCS – Universidade Nova de Lisboa

Abstract

In this paper, we introduce a novel programming model for safe and incremental construction of live software systems. We capture a verified style of agile development that spans over the whole development life cycle of applications, from early specification and prototyping to maintenance and evolution. We enable a continuous feedback loop on programming effects, at the level of program results, and at the level of invariant verification. This approach proposes a step forward with relation to the traditional "code-compile-deploy" cycle, allowing for both code and data updates to be concurrently and safely applied during normal system execution.

We provide a language-based typeful development and runtime system, for hosting and evolving data-centric applications. Our approach is presented as a core typed imperative calculus with abstractions for the construction and composition of live systems, and a reactive and data-flow semantics. The associated type discipline ensures a correct interleaving of execution and construction of systems. The soundness of our calculus is supported by standard subject reduction, progress, and convergence results.

1. Introduction

Flexible development frameworks and agile development methodologies are increasingly popular, especially in the domain of web and cloud (data-centric) applications. However, a development process composed by many intermediate and incremental releases often causes a large amount of wasted effort on the refactoring of existing code and data. Although support for code refactoring does exist in modern development tools, it does not avoid the risk of service disruption that a software update may represent. Software updates are a natural part of the software system's life cycle, one that is particularly prone to errors, that may cause significative downtime and impact on end users [4]. Such an effect is magnified by the gap between the code, as a static artifact, and the persistent state it is designed to manipulate. The act of programming is mostly based on the developer's abstract reasoning, whose results are only visible in a separate moment, while debugging, testing and executing the program. Moreover, data-centric system updates often require explicitly programmed scripts to handle the evolution of persistent data, in an ad-hoc and (usually) synchronized way.

In this paper, we present an effective and typeful incremental way of constructing software systems without causing service disruption. We introduce a novel core programming model that provides immediate feedback on the effects of gradually introducing new programming elements in the system, or redefining existing ones. Also, our model combines the static verification of each programming step with a scheduling discipline that ensures the absence of runtime errors and interferences between execution and development. Such goals are instantiated in a core programming language that allows for a flexible style of incremental [8, 22, 28] and live programming [7, 13, 19], where typing ensures that the system is kept sound at all times. Although some language interpreters allow for the dynamic extension and redefinition of parts of their code and data [9], our approach provides a verified step-by-step style of typeful and gradual programming. Live programming environments[1] are set to reduce the feedback loop that exists between writing the code and perceiving its effects. We extend this goal to the complete life cycle of large, critical, and data-centric systems, where upgrades are defined using a uniform and universal notion of system composition and reconfiguration, that includes both code and data. Also, we foresee the extension of such an approach with richer and powerful dependent type and information flow analysis that ensure that stronger invariants, about correctness and security, are maintained, in a step-by-step development process.

Incremental and reactive frameworks and software architectures, such as Meteor[2] and AngularJS[3], providing incremental and seamless recomputations, are increasingly important in the definition of distributed and collaborative applications, as well as in many other computational models, such as incremental and adaptive programming [2]. However, the way that modifications in the application's persistent data layer are transmitted to the (active) user interfaces, is usually implemented by ad-hoc and hand-crafted code. By using suitable language constructs, we are able to specify and reason about the definition and reconfiguration of applications, at the appropriate level of abstraction. We instantiate our model in a imperative and reactive programming language, that captures the essence of reactive software architectures, and imperative characteristics of typical user and data-centric web and cloud applications. In our setting it is possible to continuously evolve code [14, 24], and reconfigure the underlying data [4, 15]. The type discipline we define, ensures that the regular execution of applications co-exists safely with the construction of new program features, as well as the redefinition of existing ones.

To the best of our knowledge, this is the first work combining a type-safe imperative and reactive language with a statically verified dynamic reconfiguration mechanism. Our key contributions can be summarized as follows:

[1] www.lighttable.com

[2] www.meteor.com

[3] www.angularjs.org

- a novel core programming model for data-centric applications, that supports type-safe dynamic evolution.
- a type system that statically ensures the safety of applications in the presence of both reactive propagation of changes and dynamic evolution.
- a data-flow operational semantics supporting the synchronized evolution of both code and data.
- type preservation and progress results that imply properties like the bound queue length and convergence of change propagation.

The remainder of this paper is structured as follows. Section 2 introduces our programming language by means of a small example. In Section 3, we formally present our core reactive and imperative programming language, its type system, operational semantics, and the incremental construction mechanism. Section 4 states the soundness properties of our language. Finally, Sections 5 and 6 provide a comparison with related work, and a final discussion on our work.

2. Live and Reactive Programming

We introduce three main ingredients that model our target scenario of data-centric applications: state variables, data transformation expressions, and actions. We use bound *state variables* to model the persistent (data) layer of a system. The application logic layer is modeled by bound *data transformation expressions*, representing either a query over persistent state, processing of intermediate results, or code that formats data to serve a view of the system. Finally, we have *actions*, that enclose imperative updates to the data layer. The imperative behavior of systems, which is related to insert or update queries, is isolated in a monadic style. *Actions* are delayed computations that, in our setting, model event handlers in web pages, and trigger data updates.

We follow a layered model where we provide a remote interface, a console or web-service, to manage the application code on top of a persistent, reactive, and operational layer (c.f. [23, 24]). Construction operations are interpreted at the systems' interface level to (re)define parts of an application, and also to trigger data updates. All core operations submitted to the system's interface are statically checked against the current system's (type) specification, and all ill-typed operations are rejected. This model captures the common architectural conventions of data-centric applications, where resources (named values) can be obtained on well specified URIs, and where the (re)definition of names and the execution of actions (data updates) correspond to the invocation of available web-services. Moreover, we follow a data-driven operational semantics that relates to known frameworks where changes in the systems resources (e.g. web page elements) are pro-actively pushed to clients.

We next illustrate the syntax and semantics of our language by means of a simple running example, developed in two different stages, simulating two development sprints.

2.1 Building the System

Consider the problem of developing a small bulletin board application for mobile devices, with a user interface (UI) similar to the one in Figure 1. This application includes a list of messages stored in a persistent way; a "thumbs-up" icon for each message in the list, linked to requests (actions) that increment a counter of "likes" associated to the corresponding message; a text box where a new message may be written; and a button with label "Post", linked to an action that adds the message to the persistent state of the application.

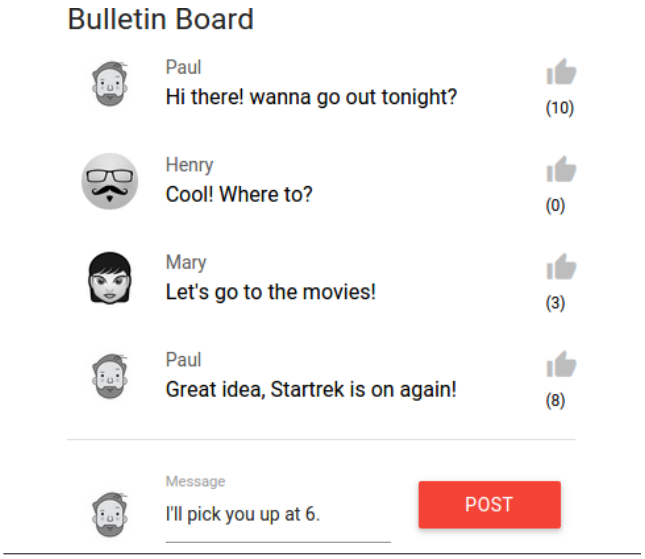


Figure 1. Bulletin Board UI.

We now illustrate the sequence of construction operations that may be used to support such user interface (see Figure 2 for the full program and corresponding dependency graph). Our model is parametric in the language used for expressions, which in our case is an extension of the simply-typed λ-calculus with collections and names. In the example that follows, we use an extended language, with records, and data manipulation constructs – which we show, in Section 3, to be encodable in our core language.

The application starts with an empty system. We first define a state variable containing a collection ($[\ldots]$) of sample messages, records ($\{\ldots\}$).

$$\textbf{var}\ \mathsf{messages}{=}[\left\{\begin{array}{ll} \mathit{id} & = 0, \\ \mathit{author} & = \text{“Paul”}, \\ \mathit{message} & = \text{“Hi there! ...”}, \\ \mathit{likes} & = 10 \end{array}\right\}, \ldots]$$

This operation defines the initial (persistent) state of the application, and is applied to the system as an independent construction operation. Each record in the collection messages contains a message identifier (*id*), the name of the *author*, the *message* text, and the number of *likes* for that particular message. The scope of name messages includes all future (re)definitions in the application. We now define a data transformation expression

$$\textbf{def}\ \mathsf{size}{=}\textbf{foreach}(x\ \textbf{in}\ \mathsf{messages}\ \textbf{with}\ y = 0)\ y + 1$$

to denote the size of the collection messages. Notice that size will always be up-to-date with relation to the contents of messages. The **foreach** expression is an iterator (c.f. `foldl`) over collection messages with an accumulator y whose initial value is `0`, and that on each iteration, the value of y is increased by `1`.

We abstract the insertion of a message into collection messages, by means of a function

$$\textbf{def}\ \mathsf{post}{=}\lambda a.\lambda m.$$
$$\textbf{action}\ \{\textbf{insert}\ \left\{\begin{array}{ll} \mathit{id} & = \mathsf{size}, \\ \mathit{author} & = a, \\ \mathit{message} & = m, \\ \mathit{likes} & = 0 \end{array}\right\}\ \textbf{into}\ \mathsf{messages}\}$$

where parameter a denotes the author's name, and m denotes the message text. This function can then be used to define the behavior of a button in the user interface. Notice that the definition of function post uses name size to assign a fresh identifier to

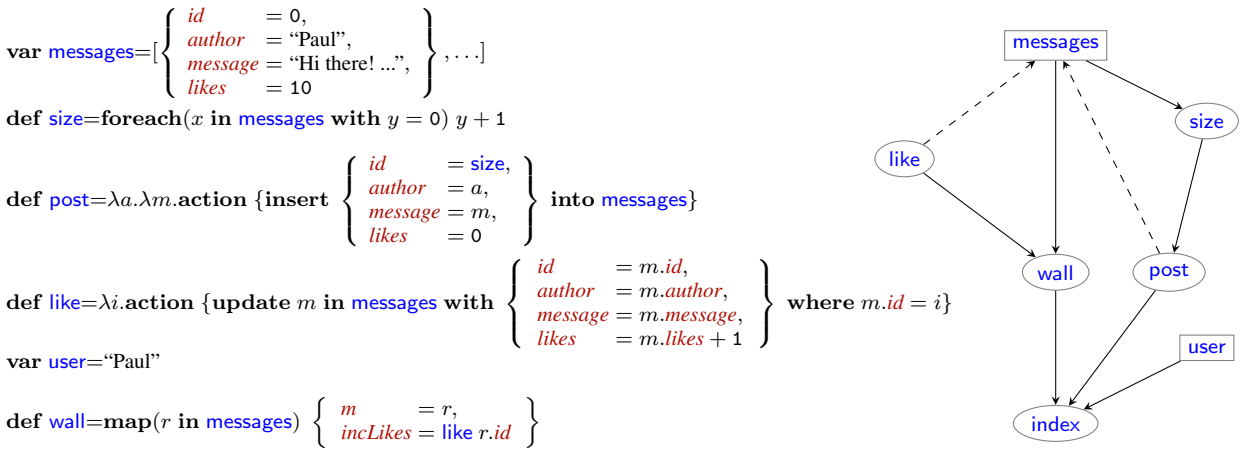


Figure 2. Bulletin Board.

the new message, which is always up-to-date with relation to the collection stored in messages.

To continue building the interface shown in Figure 1, we need to define the behavior of the "thumbs-up" icon, on each row. We start by defining a function that increments the *likes* counter of one particular message

```
def like=λi.action {update m in messages
  with { id = m.id, author = m.author, message = m.message, likes = m.likes + 1 } where m.id = i}
```

Function like takes a message identifier (i) and increments the number of *likes* of the specified message, through an **update** expression, with an appropriate filter, and assuming that the identifier of a message is unique.

Next, we define a state variable user (**var** user="Paul"), denoting the name of the current logged in user. Notice that, at this stage, we abstract the different levels of data persistence commonly present in web applications (e.g. sessions, database tables). In this case, messages would represent a database table, and user would be a session variable.

We can now gather all the required data, used in the UI shown in Figure 1, into a collection under the name wall.

```
def wall=map(r in messages) { m = r, incLikes = like r.id }
```

This collection contains the message record (m) and an **action** value for each row. Each action in field *incLikes* is a thunk containing the identifier of the message in the same row. In a data-oriented cloud application, the value of wall could be made available through a web-service and used to populate a client application. An alternative, is to use a template language, and produce a web page definition that corresponds to the user interface shown in Figure 1.

```
def index=
  map(r in wall){
    div {
      img ("/imgs/" + r.m.author + ".jpg")
      r.m.message
      button (img "/imgs/thumbsup.jpg") do r.incLikes
      "(" + r.m.likes + ")"
    }
  }
  div {
    img ("/imgs/" + user + ".jpg")
    textarea#message
    button "Post" do post user #message
  }
```

For the sake of simplicity, we omit any styling properties of page elements and only present the page skeleton. In the definition above, the **map** expression iterates over collection wall, and for each message generates a corresponding HTML `<div>` element. Notice that declared names can conceivably be accessed as resources through agreed URLs. For instance, name wall can be accessed using a GET request (e.g. on URL `/wall/`). The execution of a **do** *r.incLikes* operation, triggered by clicking the "thumbs-up" button, is linked to a POST request (e.g. on URL `/like/` with the actual parameters {"i":"1"}). Notice that *r.incLikes* denotes an action generated by name like for the respective message. The second **div** expression generates the bottom section of the UI presented in Figure 1, containing the current user profile picture, and a **textarea** whose content is bound to the local identifier `#message`. Button "Post" is linked to the execution of an action, and in this case is linked to a POST request. For instance, if user "Paul" enters the message "I'll pick you up at 6." and clicks the "Post" button, the following parameters will be used in an asynchronous request {"a":"Paul", "m":"I'll pick you up at 6."}. So, the list of messages shown in Figure 1 corresponds to accessing name index (e.g. on URL `/`). Whenever the state of the application is changed, the web page generated by accessing name index is updated, and the new values are pro-actively pushed to the user interface.

In summary, by adding a new message or clicking a "thumbs-up" icon, the state variable messages is updated, causing the propagation of changes to the names that depend on it (i.e. size, wall and index), thus refreshing them. The paths of change propagation are detailed in the graph of Figure 2. The application dependency

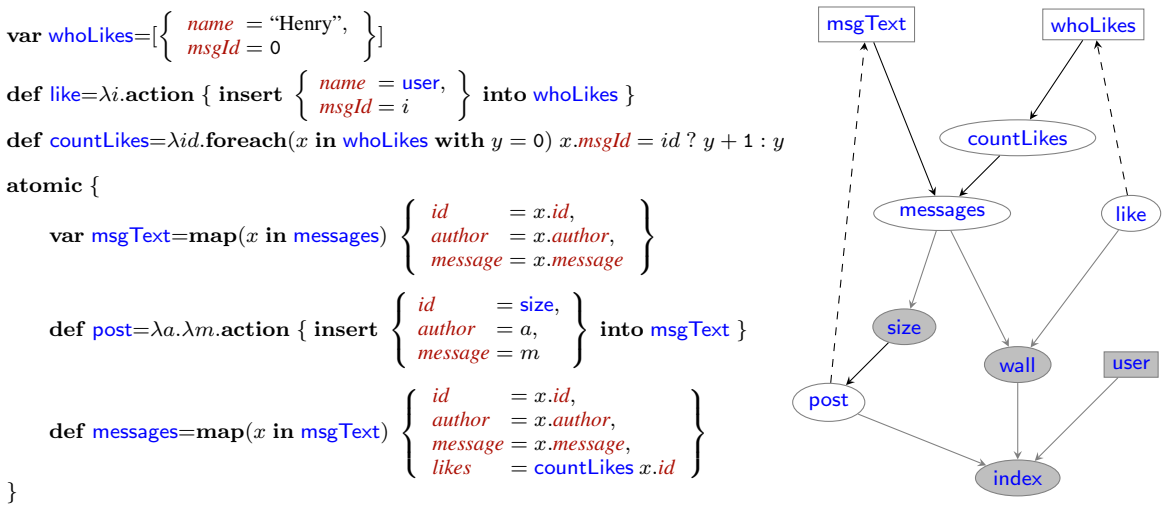


Figure 3. Bulletin Board Reconfiguration.

graph is incrementally built along with each operation, and kept up-to-date with relation to the name dependencies of data transformation expressions. Graph nodes correspond to named elements (square - **var** and ellipse - **def**), and solid edges denote dependencies between names, representing the direction of the propagation of changes. Dashed edges denote delayed action assignments, and therefore are not accounted in the propagation of changes, since actions are only explicitly executed with a **do** operation. In order to ensure that change propagation does not diverge, we must ensure that the dependency graph is acyclic. This is statically and gradually ensured by our type system every time a new construction operation is added to the system (presented in Section 3.1), and is one of our final results, expressed in Theorem 4.8 (Section 4). Notice that the expressiveness of the language is not limited by the acyclic property of the graph, because the application logic is mainly concentrated in the data transformation expressions, which is a parameter in our setting.

2.2 Evolving the System

Consider now a second sprint in the development of our running example where a new feature is introduced. We would like to store which users have "liked" each message, instead of simply counting the occurrences. By using the general scheme of (re)definition of names associated to state variables and data transformation expressions, one can extend and redefine parts of the running system. The language's type discipline ensures that the typed name dependencies in the system are not broken by each new (re)definitions. We also introduce a more flexible reconfiguration mechanism, that combines a sequence of construction operations into a single and atomic operation, and where transient inconsistent states can happen without impact on execution.

The evolution of the new feature in the existing system is attained by the operations listed in Figure 3, that build the dependency graph illustrated on the right-hand side of the figure. These operations will change the core of the application without the need to redefine important visible parts (index and wall), depicted in gray on the graph.

We start by declaring a new state variable to store the relation between messages and users (whoLikes). In this case, there was no previous information of which user liked each message, and for the sake of illustrating the structure of the collection, we include only one record stating that user "Henry" "liked" the message with *id* 0.

Next, we redefine function like to insert a record into the new collection (whoLikes) instead of incrementing the counter in the messages variable. Also, we define an auxiliary function(countLikes) that for a given message identifier id, iterates over the collection whoLikes to count the occurrences of that particular message. The following construction steps target the creation of a new collection that does not contain a counter (msgText), and the redefinition of name messages as a view (c.f. database views) to the pair of new state variables (msgText and whoLikes). Since the evolution of the system is interleaved with regular execution (the adding of new messages and "likes"), notice that we created a subtle effect with this code. The new "likes" will be accounted for, but will not be depicted in the interface, while the previously existing likes will be set to 0 (by a design choice in the example).

We next introduce the construction operation **atomic**, that allows to apply a set of operations to a running application in bulk, redefining several names in a *transaction*-like style. With this mechanism, we temporarily disallow interaction operations (**do**) and propagation of changes, which help avoiding effects such as the one described above (e.g. transient states). As we will see in Section 3.2 and Section 3.3, our runtime system explicitly schedules the execution of operations, and hence during a combined construction operation (**atomic**), interaction operations are postponed.

In this **atomic** block, we define a new collection to store messages (msgText), without a counter, and initialize it by iterating the existing messages. Function post is modified, so to add messages to msgText instead of messages. Finally, messages is redefined as a view to the new state variables msgText and whoLikes. Notice that messages represents essentially the same information as before, but with a different internal representation that can be used in further extensions.

A working prototype of a development environment based on this programming model can be found in [1]. In this live development environment it is possible to build and evolve an application, while at the same time interaction with the defined application is still possible. New application elements are seamlessly added to the running application, while redefinition of exiting names are reactively propagated into the UI, without the need for manual refresh. Using a REST interface as the one described above, it is possible to access and inspect all defined names through conventioned URLs. Also, using an HTML-like language, we are able to define web pages that are reactively and seamlessly refreshed, thus providing

o	::=	r	Construction Operations
	|	$\mathbf{do}\ e$	Interaction Operations
r	::=	$\mathbf{var}\ a{=}e$	State variables
	|	$\mathbf{def}\ a{=}e$	Data transformations
	|	$r \circledast r'$	Composition of Operations
e	::=	a	Names
	|	$\mathbf{b}$	Base values
	|	x	Variables
	|	$\lambda x.e$	Abstraction
	|	$e\ e'$	Application
	|	$\mathbf{action}\ \{a \leftarrow e\}$	Action
	|	$e\ ?\ e' : e''$	Conditionals
	|	$e\ \mathsf{op}\ e'$	Binary operations
	|	$[e_1, \ldots, e_n]$	Collection construction
	|	$\mathbf{foreach}(x\ \mathbf{in}\ e\ \mathbf{with}\ y = e')\ e''$	Collection iteration
	|	$\mathbf{match}\ e\ \mathbf{with}\ x{::}xs \rightarrow e' \mid [] \rightarrow e''$	Collection destruction

Figure 4. Programming Language Syntax.

an immediate feedback on the changes occurring in the application data.

In summary, all construction steps are seamlessly applied to the running system, while all elements visible in the user interface in Figure 1, namely index, continue to operate and be refreshed. This allows for a step-by-step construction style where the developer has immediate feedback about the validity and effect of the introduced changes.

3. Programming Language

We now present our core programming language, whose syntax is presented in Figure 4, and consists in top-level operations (o), comprising two distinct kinds – construction ($\mathbf{var}\ a{=}e$ and $\mathbf{def}\ a{=}e$) and interaction ($\mathbf{do}\ e$). Both kinds of operations rely on a λ-calculus-based functional core (e). Recall that our model is parametric in the language used for expressions.

We define an operational semantics where top-level operations (o) are evaluated with relation to a running application, that encloses both state and code. We assume that $a, b, c, \ldots$ range over a set of names $\mathcal{N}$, and that $x, y, z, \ldots$ range over a set of variables $\mathcal{V}$. For the sake of simplicity, names are global in the application domain. A modular structure where names are nested, and exported explicitly as services can be extrapolated from this language, but with no real immediate benefit to the focus of this work.

Construction operations (r) include the declaration of state variables ($\mathbf{var}\ a{=}e$) in the application's namespace, imperatively associating name a to the value denoted by expression e; and the construction operation for the declaration of pure data transformation elements ($\mathbf{def}\ a{=}e$) which associates name a to the value denoted by expression e. Expression defining names may use names previously defined in the application's namespace. Finally, we introduce the composition operation $r \circledast r'$, that combines construction operations into blocks of operations. In the example from Section 2.2, the composition operator was introduced as an **atomic** block. These blocks of operations are evaluated in a transactional-style, avoiding interferences with the modified state, and allowing for transient inconsistencies in the system.

The top-level operations denominated as interaction operations ($\mathbf{do}\ e$), represent the explicit execution of an action denoted by expression e, which updates the state of the application. Data updates are implicitly and automatically propagated through the definitions of data transformation elements, thus updating their computed values (cf. a data-driven operational semantics [16]).

The functional core of the language (e) includes names (a) that denote the corresponding values in the application state, basic values $\mathbf{b}$ (e.g. **true**, **false**), and variables x. Abstraction ($\lambda x.e$) and application ($e\ e'$) follow a call-by-value semantics. The expression $\mathbf{action}\ \{a \leftarrow e\}$ contains a delayed assignment statement to state variable a. We also include the ternary conditional operator ($?:$), and for the sake of simplicity, assume a set of binary operators (op) over base values. We extend the base λ-calculus with collections, with a constructor $[e_1, \ldots, e_n]$, and the corresponding concatenation ($@$) and append ($::$) binary operations (in op). The iterator $\mathbf{foreach}(x\ \mathbf{in}\ e\ \mathbf{with}\ y = e')\ e''$ denotes a fold-left operation on the collection denoted by expression e, and the iterated expression e''. Variable x denotes the current element in the collection, and variable y denotes either the value of expression e'' in the previous iteration, or the initial value given by expression e' in the first iteration. The destructor expression for collections $\mathbf{match}\ e\ \mathbf{with}\ x{::}xs \rightarrow e' \mid [] \rightarrow e''$ denotes one of two cases, depending on the value of expression e. In the case of a non-empty collection, the evaluation proceeds with expression e', where variable x denotes the head of the collection, and variable xs denotes its tail. In the case of an empty collection ($[]$), the result is denoted by expression e''.

In Section 2, we used a slightly more elaborate language that can be directly mapped into the syntax of the core language, in Figure 4, by the following abbreviations

$$\mathbf{insert}\ e\ \mathbf{into}\ \mathsf{a} \triangleq \mathsf{a} \leftarrow \mathsf{a}@[e]$$

$$\mathbf{update}\ x\ \mathbf{in}\ \mathsf{a}\ \mathbf{with}\ e\ \mathbf{where}\ e' \triangleq \mathsf{a} \leftarrow \mathbf{foreach}(x\ \mathbf{in}\ \mathsf{a}\ \mathbf{with}\ y = [])\ y@[e'\ ?\ e : x]$$

$$\mathbf{map}(x\ \mathbf{in}\ e)\ e' \triangleq \mathbf{foreach}(x\ \mathbf{in}\ e\ \mathbf{with}\ y = [])\ y@[e']$$

$$\mathbf{atomic}\ \{r_1, \ldots, r_n\} \triangleq r_1 \circledast \ldots \circledast r_n$$

We also use a template language to produce web pages that can be directly included into base values and binary operations, that extend the language orthogonally.

We next present the type system and operational semantics for the language. Since the construction and execution of applications are tightly intertwined, we start by describing the type language and type system, and then define the language's operational semantics.

3.1 Type System

We now define a type and effect system to discipline the construction of systems ((re)declaration of state variables and data transformation expressions) so that soundness of the type system implies that the resulting application is safe and responsive. To do so, we statically keep track of name dependencies to avoid the creation of unguarded cyclic dependencies. We say that a dependency cycle is guarded if it crosses an action value, and hence needs an explicit interaction operation to be activated. Any unguarded cycle would cause the propagation process to diverge. The absence of runtime errors and infinite propagation cycles are implied by the standard progress and preservation results (presented in Section 4). Our type language is defined as follows

$$\tau ::= \beta(\mathbf{b}) \mid \tau^* \mid \tau \xrightarrow{\delta} \tau' \mid \mathsf{Action}(\delta)$$

where we include a set of base types (e.g. Bool, Int) with $\beta(\mathbf{b})$ denoting the type of base value $\mathbf{b}$, types for homogeneous collections (τ^*), function ($\tau \xrightarrow{\delta} \tau'$), and action types ($\mathsf{Action}(\delta)$). Function and action types, specify the behavior of delayed expressions, and use δ to denote a set of typed dependencies on declared names. This bookkeeping of dependencies is crucial to determine the sound-

$$\frac{\tau = \lfloor \Delta(a) \rfloor \quad \tau \prec \delta(a)}{\Delta;\Gamma \overset{\delta}{\vdash} a : \tau}\ \text{(T-Name)} \qquad \frac{}{\Delta;\Gamma \overset{\delta}{\vdash} \mathbf{b} : \beta(\mathbf{b})}\ \text{(T-BaseValue)} \qquad \frac{}{\Delta;\Gamma, x:\tau \overset{\delta}{\vdash} x : \tau}\ \text{(T-ID)} \qquad \frac{\Delta;\Gamma, x:\tau \overset{\delta}{\vdash} e : \tau'}{\Delta;\Gamma \overset{\delta'}{\vdash} \lambda x.e : \tau \xrightarrow{\delta} \tau'}\ \text{(T-Abstraction)}$$

$$\frac{\Delta;\Gamma \overset{\delta}{\vdash} e : \tau \xrightarrow{\delta'} \tau' \quad \Delta;\Gamma \overset{\delta}{\vdash} e' : \tau \quad \delta' \sqsubseteq \delta}{\Delta;\Gamma \overset{\delta}{\vdash} e\ e' : \tau'}\ \text{(T-Application)} \qquad \frac{\Delta;\Gamma \overset{\delta}{\vdash} e : \tau \quad \Delta;\Gamma \overset{\delta}{\vdash} e' : \tau' \quad \mathsf{op} : \tau \to \tau' \to \tau''}{\Delta;\Gamma \overset{\delta}{\vdash} e\ \mathsf{op}\ e' : \tau''}\ \text{(T-BinaryOp)}$$

$$\frac{\Delta;\Gamma \overset{\delta}{\vdash} e : \mathsf{Bool} \quad \Delta;\Gamma \overset{\delta}{\vdash} e' : \tau \quad \Delta;\Gamma \overset{\delta}{\vdash} e'' : \tau}{\Delta;\Gamma \overset{\delta}{\vdash} (e\ ?\ e' : e'') : \tau}\ \text{(T-If)} \qquad \frac{\Delta;\Gamma \overset{\delta}{\vdash} e : \tau^* \quad \Delta;\Gamma \overset{\delta}{\vdash} e' : \tau' \quad \Delta;\Gamma, x:\tau, y:\tau' \overset{\delta}{\vdash} e'' : \tau'}{\Delta;\Gamma \overset{\delta}{\vdash} \mathbf{foreach}(x\ \mathbf{in}\ e\ \mathbf{with}\ y = e')\ e'' : \tau'}\ \text{(T-ForEach)}$$

$$\frac{\Delta;\Gamma \overset{\delta}{\vdash} e_i : \tau \quad i = 1,\ldots,n}{\Delta;\Gamma \overset{\delta}{\vdash} [e_1,\ldots,e_n] : \tau^*}\ \text{(T-Collection)} \qquad \frac{\Delta;\Gamma \overset{\delta}{\vdash} e : \tau^* \quad \Delta;\Gamma, x:\tau, xs:\tau^* \overset{\delta}{\vdash} e' : \tau' \quad \Delta;\Gamma \overset{\delta}{\vdash} e'' : \tau'}{\Delta;\Gamma \overset{\delta}{\vdash} (\mathbf{match}\ e\ \mathbf{with}\ x::xs \to e' \mid [] \to e'') : \tau'}\ \text{(T-Match)}$$

$$\frac{\Delta, a : \mathsf{var}_\delta(\tau);\Gamma \overset{\delta[a:\mathsf{var}(\tau)]}{\vdash} e : \tau}{\Delta, a : \mathsf{var}_\delta(\tau);\Gamma \overset{\delta'}{\vdash} \mathbf{action}\ \{a \leftarrow e\} : \mathsf{Action}(\delta[a : \mathsf{var}(\tau)])}\ \text{(T-Action)}$$

Figure 5. Typing Rules for Expressions.

ness of a names (re)definition, as described in detail in Section 3.3. Typed dependencies set are defined by

$$\delta ::= a : \tau, \delta \mid a : \mathsf{var}(\tau), \delta \mid \varepsilon$$

Typed dependencies sets capture information about the type of a used name, and wheter if it denotes a state variable. A typed dependency of the form $a{:}\tau$ means that name a is used with type τ. A typed dependency of the form $a{:}\mathsf{var}(\tau)$ means that name a is used as state variable in an assignment, with type τ.

Our type and effect system is defined by two layers of typing judgments that target construction operations and expressions.

$$\Delta \vdash o : \Delta' \quad \text{Operations}$$
$$\Delta;\Gamma \overset{\delta}{\vdash} e : \tau \quad \text{Expressions}$$

We register, in typing environments Δ, the type and usage information of names that can be defined and used in operations and expressions. Expressions type environments Γ map variables $x, y, z, \ldots$ to types. The typing judgment for operations registers the effect (Δ') of operations (o), and the judgment for expressions includes the typed dependencies of expressions towards names (δ), that establishes the effect of operations.

Typing environments Δ map names to type annotations (σ), which are designed to carry information about the names' denotation. Type annotations are defined by

$$\sigma ::= \mathsf{var}_\delta(\tau) \mid \mathsf{def}_\delta(\tau)$$

where δ denotes a typed dependencies set (as defined above), to denote all typed dependencies of the expression associated the annotated name (i.e. the direct dependencies of that name). Function types ($\tau \xrightarrow{\delta} \tau'$), conservatively signal the typed dependencies (δ) of the abstraction body. In action types ($\mathsf{Action}(\delta)$), the typed dependencies set (δ) denotes the dependencies used in the delayed assignment and corresponding expression ($\mathbf{action}\ \{a \leftarrow e\}$). In the case of type annotations (σ) in typing environments, typed dependency sets become essential to avoid unguarded cyclic definitions, which would cause infinite loops during propagation of changes. This allows to statically discipline the (re)definition of names. In Section 3.3, we show how to statically satisfy and preserve these conditions.

Now consider the following auxiliary definitions used in our type system, and essential to technically understand the typing rules for expressions.

Definition 3.1 (Type of Annotation)**.** We obtain the type of an annotation σ, as follows

$$\lfloor \mathsf{var}_\delta(\tau) \rfloor \triangleq \tau \qquad \lfloor \mathsf{def}_\delta(\tau) \rfloor \triangleq \tau$$

Definition 3.2 (Typed Dependency Coercion)**.** We define the type dependency coercion, as follows

$$\tau \prec \tau \qquad \tau \prec \mathsf{var}(\tau) \qquad \mathsf{var}(\tau) \prec \mathsf{var}(\tau)$$

The former allows the extraction of the actual type in a type annotation, and the latter defines a coercion mechanism for typed dependencies. Notably, we define that a *write* typed dependency ($\mathsf{var}(\tau)$) can be used whenever a simple dependency (τ) is required. This relation can also be lifted to define a relation between typed dependency sets $\delta \sqsubseteq \delta'$.

The typing relation on expressions, inductively defined by the rules in Figure 5, follows standard lines, except that they are extended with the invariant conditions about typed dependencies (δ). As expected, in the typing relation on expressions, all language values (T-BaseValue, T-ID, T-Abstraction and T-Action) are typed with all sets of typed dependencies. The most interesting cases in the typing rules for expressions are T-Name, T-Abstraction, T-Application and T-Action. In the case of rule T-Name, a name a is typed with relation to an environment and a typed dependencies set that agree in the type of a, modulo the typed dependency coercion of Definition 3.2, and the type of the annotation (Definition 3.1). An abstraction $\lambda x.e$ is typed (rule T-Abstraction) with any typed dependency set δ', given that the resulting function type is annotated with the typed dependency set (δ) that types the abstraction body e. This allows us to capture the set of dependencies of a closure, and defer its use to the time of its application. That can be observed in rule T-Application, where the set of typed dependencies associated with the abstraction type (δ') is required to be a subset (or equal) of the application typed dependencies set ($\delta' \sqsubseteq \delta$). Rule T-Action requires that in an assignment, the type of the expression (e) agrees with the type of the annotated name (a), and that name is a state variable ($\mathsf{var}_\delta(\tau)$). Expression e can refer to the old value of the assigned variable, hence its typed dependencies set is $\delta[a : \mathsf{var}(\tau)]$. The resulting action type keeps track of the use of name a as a state variable ($a : \mathsf{var}(\tau)$), therefore blocking all code changes that try to transform it into a data transformation expression ($\mathbf{def}$).

The type and effect system on operations is defined by a judgment of the form $\Delta \vdash o : \Delta'$ asserting that operation o produces the effect Δ' with relation to the typing environment Δ. The typing based on the effects of operations allows us to capture the incremental effects of each operation, and therefore verify that a given operation is compatible with the running system. In Section 3.3 we further detail this with Definition 3.11. The typing relation on operations is inductively defined by the rules in Figure 6. Notice that the type verification based on the effects is essential to dynamically accept new construction operations in a system, by allowing to predict if the system is sound after executing all steps that are in the

evaluation queue to be processed. We use the following auxiliary definition about the union of two typing environments, with precedence to the right-hand side.

Definition 3.3 (Typing Environment Union)**.** The union of typing environments Δ and Δ', written $\Delta \uplus \Delta'$, is derived by

$$\Delta \uplus \Delta' = \{a : \Delta(a) \mid a \in \mathsf{dom}(\Delta) - \mathsf{dom}(\Delta')\} \cup \Delta'$$

We next define the notion of well-formed type and well-formed typed dependencies set. The first (Definition 3.4) asserts that a given type is well-formed with relation to a typing environment Δ, and the most interesting cases are for function and action types, where we also assert that the typed dependencies sets δ are well-formed, according to Definition 3.5. Regarding Definition 3.5, it asserts that all typed dependencies agree in the type of the dependency name, and also that type associated with the name is well-formed, according to Definition 3.4.

Definition 3.4 (Well-Formed Type)**.** A type τ is well-formed wrt. typing environment Δ, if $\Delta \vdash \tau$ can be derived by

$$\Delta \vdash \beta(\mathbf{b}) \qquad \frac{\Delta \vdash \tau}{\Delta \vdash \tau^*} \qquad \frac{\Delta \vdash \delta}{\Delta \vdash \mathsf{Action}(\delta)}$$

$$\frac{\Delta \vdash \tau \qquad \Delta \vdash \tau' \qquad \Delta \vdash \delta}{\Delta \vdash \tau \xrightarrow{\delta} \tau'}$$

Definition 3.5 (Well-Formed Typed Dependencies)**.** A typed dependencies set δ is well-formed with relation to a typing environment Δ, if $\Delta \vdash \delta$ can be inductively derived by

$$\Delta \vdash \varepsilon \qquad \frac{\lfloor \Delta(a) \rfloor \prec \tau \qquad \Delta \vdash \tau \qquad \Delta \vdash \delta'}{\Delta \vdash \delta', a : \tau}$$

$$\frac{\Delta(a) = \mathsf{var}_\delta(\tau) \qquad \Delta \vdash \tau \qquad \Delta \vdash \delta'}{\Delta \vdash \delta', a : \mathsf{var}(\tau)}$$

The effect of the two construction operations **var** $a{=}e$ and **def** $a{=}e$ is specified by rules T-Var and T-Def (Figure 6). The defining expressions e are typed with relation to a set of typed dependencies (δ) and expression type (τ) that are then included in the registered effect ($\mathsf{def}_\delta(\tau)$ or $\mathsf{var}_\delta(\tau)$ respectively). Notice that a definition for name a cannot directly depend on the name itself, which is represented in the condition ($a \notin \mathsf{dom}(\delta)$). In a composition operation (T-Composition), the left operand is typed with relation to the initial typing environment Δ, while the right-hand side operation is typed in a typing environment obtained by combining the initial environment with the effect of the left-hand side operation ($\Delta \uplus \Delta'$). The effect of the composition is obtained by combining the effects of both operands ($\Delta' \uplus \Delta''$). The effect of interaction operations (**do** e) is empty in all cases (ε), i.e. the running application definition is not modified, only the state is modified at runtime.

Our typing relation on expressions has one invariant, $\Delta \vdash \delta$, stating that the typed dependencies set is always well-formed according to the typing environment Δ (Definition 3.5). Also, all types τ are well-formed according to Definition 3.4.

3.2 Operational Semantics

We define a reactive operational semantics by means of two layers of small-step reduction relations, one for construction operations and another for expressions. The semantics of construction operations is defined on runtime configurations that represent a complete live system. Runtime configurations are of the form $(\Delta; \mathcal{S}; \mathcal{Q})$, where Δ is a typing environment that describes the current system type specification, $\mathcal{S}$ is a state mapping names (a) to tuples with the form (e, ν), with e an expression, whose free names range over the domain of $\mathcal{S}$, and ν the current denotation of the name a, that can be either a computed value (v), or the special denotation *undefined* ($\square$). We write $\mathcal{S}[a \mapsto (e, \nu)]$ to denote a state obtained from

$$\frac{\Delta; \varepsilon \overset{\delta}{\vdash} e : \tau \quad a \notin \mathsf{dom}(\delta)}{\Delta \vdash \mathbf{var}\ a{=}e : \{a : \mathsf{var}_\delta(\tau)\}}\ \text{(T-Var)} \qquad \frac{\Delta; \varepsilon \overset{\delta}{\vdash} e : \tau \quad a \notin \mathsf{dom}(\delta)}{\Delta \vdash \mathbf{def}\ a{=}e : \{a : \mathsf{def}_\delta(\tau)\}}\ \text{(T-Def)}$$

$$\frac{\Delta \vdash r : \Delta' \qquad \Delta \uplus \Delta' \vdash r' : \Delta''}{\Delta \vdash r \circledast r' : \Delta' \uplus \Delta''}\ \text{(T-Composition)}$$

$$\frac{\Delta; \varepsilon \overset{\delta'}{\vdash} e : \mathsf{Action}(\delta)}{\Delta \vdash \mathbf{do}\ e : \varepsilon}\ \text{(T-Do)}$$

Figure 6. Typing Rules for Operations.

state $\mathcal{S}$ and associating a to (e, ν). The language runtime values are defined by

$$v ::= \mathbf{b} \mid \lambda x.e \mid \mathbf{action}\ \{a \leftarrow e\} \mid [v_0, \ldots, v_n]$$

and includes base values $\mathbf{b}$ (e.g. **true**, **false**), λ-abstractions, actions, and collections of values. The third, and last element of a runtime configuration, $\mathcal{Q}$, denotes a queue of construction operations, whose syntax is defined by

$$\begin{array}{lcl} \mathcal{Q} & ::= & \mathbf{do}\ e; \mathcal{Q} \mid a; \mathcal{Q} \mid a\langle e \rangle; \mathcal{Q} \mid [q]^{\Delta}_{\mathcal{S}}; \mathcal{Q} \mid \varepsilon \\ q & ::= & r; q \mid a; q \mid a\langle e \rangle; q \mid \varepsilon \end{array}$$

The queue disciplines the evaluation of a set of related events. Queued events have one of the following sorts: interaction operations (**do** e), names to be refreshed (a), names currently being refreshed ($a\langle e \rangle$), or ongoing construction blocks ($[q]^{\Delta}_{\mathcal{S}}$). Construction blocks are annotated with a typing environment Δ and a state $\mathcal{S}$, that describe its partial effect with relation to the initial state. A construction block is defined over a queue of selected events q, that exclude the execution of actions (**do** e). Blocks can contain **var** $a{=}e$ or **def** $a{=}e$ construction operations (r), names queued to be refreshed (a), and names being evaluated ($a\langle e \rangle$).

We first consider some extra definitions, auxiliary in the reduction relation on operations. We define the set of subscribers of a name a, as all data transformation expressions (**def**) that depend directly on name a. This definition is crucial to setup the propagation process, that follow the data transformation expressions and does not go into expressions that initialize state variables.

Definition 3.6 (Subscribers)**.** The set of subscribers of name a, with relation to Δ, written $\mathsf{subscribers}_\Delta(a)$, is defined by

$$\mathsf{subscribers}_\Delta(a) = \{\ b \mid \Delta(b) = \mathsf{def}_\delta(\tau) \wedge a \in \mathsf{dom}(\delta)\ \}$$

We also define the union of two states as the combination of two states, giving preference to the right-hand side operand (similar to Definition 3.3 on typing environments).

Definition 3.7 (State Union)**.** We define the union of two states $\mathcal{S}$ and $\mathcal{S}'$, written $\mathcal{S} \uplus \mathcal{S}'$, as follows:

$$\mathcal{S} \uplus \mathcal{S}' = \{a \mapsto \mathcal{S}(a) \mid a \in \mathsf{dom}(\mathcal{S}) - \mathsf{dom}(\mathcal{S}')\} \cup \mathcal{S}'$$

Given these auxiliary definitions, the reduction relation on operations is inductively defined by the rules in Figure 7, following the general structure $\Delta; \mathcal{S}; \mathcal{Q} \rightarrow \Delta'; \mathcal{S}'; \mathcal{Q}'$ where operations are executed according to the discipline imposed by queue $\mathcal{Q}$, thus modifying the typing environment Δ and state $\mathcal{S}$ accordingly.

We divide the description of the operational semantics, and start by the rules that specify the handling of queued events. Section 3.3 introduces runtime rules that allow to insert new operations into the queue.

Queued construction operations reduce through rules S-RVar and S-RDef. Notice that the syntax of queues always places construction operations (r) inside construction blocks ($[q]^{\Delta'}_{\mathcal{S}'}$), so, the effect of these operations is computed with relation to the block's

$$\frac{\Delta \uplus \Delta' \vdash \textbf{var}\ a{=}e : \Delta''}{\Delta; \mathcal{S}; ([\textbf{var}\ a{=}e; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}) \to \Delta; \mathcal{S}; ([a; q]^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q})}\ \text{(S-RVar)} \qquad \frac{\Delta \uplus \Delta' \vdash \textbf{def}\ a{=}e : \Delta''}{\Delta; \mathcal{S}; ([\textbf{def}\ a{=}e; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}) \to \Delta; \mathcal{S}; ([a; q]^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q})}\ \text{(S-RDef)}$$

$$\frac{\mathcal{S}(a) = (e, v)}{\Delta; \mathcal{S}; (a; \mathcal{Q}) \to \Delta; \mathcal{S}; (a\langle e\rangle; \mathcal{Q})}\ \text{(S-Queue)} \qquad \frac{(\mathcal{S} \uplus \mathcal{S}')(a) = (e, \nu)}{\Delta; \mathcal{S}; ([a; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}) \to \Delta; \mathcal{S}; ([a\langle e\rangle; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q})}\ \text{(S-RQueue)}$$

$$\frac{\mathcal{S}; e \to e'}{\Delta; \mathcal{S}; (a\langle e\rangle; \mathcal{Q}) \to \Delta; \mathcal{S}; (a\langle e'\rangle; \mathcal{Q})}\ \text{(S-Step)} \qquad \frac{\mathcal{S} \uplus \mathcal{S}'; e \to e'}{\Delta; \mathcal{S}; ([a\langle e\rangle; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}) \to \Delta; \mathcal{S}; ([a\langle e'\rangle; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q})}\ \text{(S-RStep)}$$

$$\frac{s = \mathsf{subscribers}_{\Delta}(a)}{\Delta; \mathcal{S}[a \mapsto (e, v')]; (a\langle v\rangle; \mathcal{Q}) \to \Delta; \mathcal{S}[a \mapsto (e, v)]; (s; \mathcal{Q})}\ \text{(S-Value)} \qquad \frac{(\mathcal{S} \uplus \mathcal{S}')(a) = (e, \nu)}{\Delta; \mathcal{S}; ([a\langle v\rangle; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}) \to \Delta; \mathcal{S}; ([q]^{\Delta'}_{\mathcal{S}'[a \mapsto (e, v)]}; \mathcal{Q})}\ \text{(S-RValue)}$$

$$\frac{s = \mathsf{subscribers}_{\Delta \uplus \Delta'}(\mathsf{dom}(\Delta'))}{\Delta; \mathcal{S}; ([\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}) \to \Delta \uplus \Delta'; \mathcal{S} \uplus \mathcal{S}'; (s; \mathcal{Q})}\ \text{(S-REmpty)} \qquad \frac{\mathcal{S}; e \to e'}{\Delta; \mathcal{S}; (\textbf{do}\ e; \mathcal{Q}) \to \Delta; \mathcal{S}; (\textbf{do}\ e'; \mathcal{Q})}\ \text{(S-Do)}$$

$$\Delta; \mathcal{S}[a \mapsto (e', v)]; (\textbf{do action}\ \{a \leftarrow e\}; \mathcal{Q}) \to \Delta; \mathcal{S}[a \mapsto (e, v)]; (a; \mathcal{Q})\ \text{(S-Do-Action)}$$

Figure 7. Operations Operational Semantics.

partial typing environment and state (Δ' and $\mathcal{S}'$). The partial typing environment (Δ') is updated with the effect of the operation (Δ''), and the state is updated to associate name a to the defined expression e, and an undefined value ($\square$). The name a is placed in the queue to be evaluated. This is the general schema to evaluate expressions, through the queue, and with precedence towards other queued operations.

The evaluation of the expression associated to a name is specified by rules S-Queue and S-RQueue that reduces the event a in the queue to the event $a\langle e\rangle$, according to the current state. Rule S-Queue works with relation to $\mathcal{S}$, and rule S-RQueue works on the partially constructed state ($\mathcal{S} \uplus \mathcal{S}'$). The remaining pair of rules work according to the same convention over states and partially constructed states. Rule S-Step (S-RStep) reduces expression e associated to a name a in the queue. This step relies on the reduction relation for expressions defined below. The evaluation of a terminated queued name (i.e. it refers to a value) is reduced by rule S-Value (S-RValue) that updates the state, by replacing the old value v' (ν) associated to name a by the new value v. In the case of a value update outside a construction block (rule S-Value), all subscribers of the updated name (Definition 3.6) are placed at the beginning of the queue. This ensures that we immediately propagate changes into other defined names, thus giving the reactive behavior to our semantics. In the case of a construction block, this procedure is delayed until the end of the block, and names are placed in bulk in the queue (rule S-REmpty). Rule S-REmpty uses a lifted definition of the subscribers set (Definition 3.6) for sets of names. Notice that in the case of S-RValue, we do not place the subscribers in the queue, hence propagation does not occur during the execution of composed operations (construction blocks). At the end of a construction block (rule S-REmpty) the partially computed typing environment and state are combined with the main typing environment and state, as if committing changes. This is a key point, where our static type analysis ensures that the soundness of the system (that can be transient in a construction block) is restored, and propagation can carry on.

Finally, rules S-Do and S-Do-Action handle the interaction operation (**do** e). Rule S-Do reduces the inner expression e, based on the reduction relation for expressions. The reduction of an action value (S-Do-Action) proceeds by updating the expression (e') associated to the assigned name (a) in the state. The general evaluation strategy then follows; name a is added to the queue, is evaluated and its denotation is placed in the state afterwards (rules S-Step and S-Value respectively).

The reduction relation on configurations depends on a reduction relation for expressions with relation to a state. We follow the structure $\mathcal{S}; e \to e'$ that specifies that expression e reduces to e' with relation to state $\mathcal{S}$. Notice that evaluation of expressions does not change the state. The reduction relation for expressions is inductively defined by the rules in Figure 8. We use the general notion of evaluation contexts (Definition 3.8) and the corresponding evaluation rule E-Ctxt.

Definition 3.8 (Expressions Contexts)**.** Evaluation contexts for expressions, $\mathcal{E}$, are defined by

$$\begin{array}{rcl} \mathcal{E} & ::= & \cdot \mid \mathcal{E}\ e \mid (\lambda x.e)\ \mathcal{E} \mid \mathcal{E}\ \mathsf{op}\ e \mid v\ \mathsf{op}\ \mathcal{E} \mid \mathcal{E}\ ?\ e : e' \\ & \mid & [v_1, \ldots, \mathcal{E}, \ldots, e_n] \mid \textbf{foreach}(x\ \textbf{in}\ \mathcal{E}\ \textbf{with}\ y = e)\ e' \\ & \mid & \textbf{foreach}(x\ \textbf{in}\ v\ \textbf{with}\ y = \mathcal{E})\ e \\ & \mid & \textbf{match}\ \mathcal{E}\ \textbf{with}\ x{::}xs \to e \mid [] \to e' \end{array}$$

Notably, rule E-Name specifies that a name is implicitly dereferenced to the corresponding value in state $\mathcal{S}$. The ternary conditional operator (E-True and E-False) follows the usual semantics, reducing to the first or second branch depending on the conditional value. Reduction of binary operations (E-BOp) is abstracted from the system, and rule E-Application implements the usual reduction of a call-by-value application. The semantics of collection iteration (E-ForEachNil and E-ForEach), and collection destruction (E-MatchNil and E-Match) also follow straightforward semantics.

This description concludes the operational semantics regarding the execution of the system. We now describe how a system may evolve by adding new construction operations to the evaluation queue.

3.3 Incremental Construction

We next describe how our runtime system can dynamically evolve by combining the presented type system (Section 3.1) and operational semantics (Section 3.2). For that purpose, we need to first consider some auxiliary definitions that enable our runtime system to dynamically check conditions that ensure the soundness of the system at all times.

Definition 3.9 (Typed Dependencies Expansion)**.** The expansion of typed dependencies δ, in a given names environment Δ, written $\sqcup_{\Delta}(\delta)$, is defined by

$$\sqcup_{\Delta}(\delta) \triangleq \mathsf{dom}(\delta) \cup \bigcup_{a \in \mathsf{dom}(\delta)} \{\ \sqcup_{\Delta}(\delta') \mid \Delta(a) = \mathsf{def}_{\delta'}(\tau)\ \}$$

Computing the expansion of the typed dependencies set is essential to statically avoid circular dependencies, and ensuring that the propagation of changes does not diverge. Notice that state variables

$$\text{(E-Ctxt)}\ \frac{\mathcal{S};e \to e'}{\mathcal{S};\mathcal{E}[e] \to \mathcal{E}[e']} \qquad \text{(E-Name)}\ \mathcal{S}[a \mapsto (e,v)];a \to v \qquad \text{(E-True)}\ \mathcal{S};\mathbf{true}\ ?\ e : e' \to e \qquad \text{(E-False)}\ \mathcal{S};\mathbf{false}\ ?\ e : e' \to e'$$

$$\text{(E-BOp)}\ \frac{v'' = \langle v\ \mathsf{op}\ v' \rangle}{\mathcal{S};v\ \mathsf{op}\ v' \to v''} \qquad \text{(E-Application)}\ \mathcal{S};(\lambda x.e)\ v \to e\{v/x\}$$

$$\text{(E-ForEachNil)}\ \mathcal{S};\mathbf{foreach}(x\ \mathbf{in}\ []\ \mathbf{with}\ y = v)\ e \to v \qquad \text{(E-ForEach)}\ \mathcal{S};\mathbf{foreach}(x\ \mathbf{in}\ v{::}vs\ \mathbf{with}\ y = v')\ e \to \mathbf{foreach}(x\ \mathbf{in}\ vs\ \mathbf{with}\ y = e\{v/x\}\{v'/y\})\ e$$

$$\text{(E-MatchNil)}\ \mathcal{S};(\mathbf{match}\ []\ \mathbf{with}\ x{::}xs \to e \mid [] \to e') \to e' \qquad \text{(E-Match)}\ \mathcal{S};(\mathbf{match}\ v{::}vs\ \mathbf{with}\ x{::}xs \to e \mid [] \to e') \to e\{v/x\}\{vs/xs\}$$

Figure 8. Expressions Operational Semantics.

in δ are not expanded, this is sound and corresponds with the restriction on change propagation into the initializer expressions of state variables. One important invariant condition on runtime configurations is the absence of unguarded dependency cycles. We next define a (static) property on typing environments, that directly implies this invariant.

Definition 3.10 (Acyclic)**.** A typing environment Δ is acyclic if, $\Delta\downarrow$ is derivable by the rules

$$\frac{\forall a \in \mathsf{dom}(\Delta).\Delta\downarrow_a}{\Delta\downarrow}$$

$$\frac{\Delta(a) = \mathsf{var}_\delta(\tau) \qquad \Delta \vdash \delta, a : \tau}{\Delta\downarrow_a}$$

$$\frac{\Delta(a) = \mathsf{def}_\delta(\tau) \qquad a \notin \sqcup_\Delta(\delta) \qquad \Delta \vdash \delta, a : \tau}{\Delta\downarrow_a}$$

We define the notion of compatibility between two typing environments Δ and Δ' in Definition 3.11.

Definition 3.11 (Compatible)**.** We define compatibility of typing environments Δ and Δ', written $\Delta \vDash \Delta'$, if $\Delta \uplus \Delta'\downarrow$.

Verifying that a typing environment does not contain unguarded cycles, and that two typing environments are compatible (without cycles after being combined) is a crucial step when adding operations to a running system. The notion of compatibility between two typing environments ensures that for the combined typing environment $\Delta \uplus \Delta'$ (Definition 3.3), all names are acyclic ($a \notin \sqcup_\Delta(\delta)$), and their typed dependencies set is well-formed ($\Delta \vdash \delta, a : \tau$) according to the previously introduced Definition 3.5. This enables us to guarantee that either type changes, or a name redefined from a state variable into a pure data transformation element (or vice-versa) does not break type safety.

Since our runtime queue may contain unprocessed construction blocks, we define the notion of queue effects (Definition 3.12). This allows to compute the effects of an unprocessed queue, thus allowing to add new construction operations at the end of the queue, in a verified way.

Definition 3.12 (Queue Effects)**.** We define the effects of a queue $\mathcal{Q}$, wrt. a typing environment Δ, written $\Delta \vdash \mathcal{Q} \dashv \Delta'$, producing the effects typing environment Δ', as follows

$$\frac{\Delta \vdash \mathcal{Q} \dashv \Delta'}{\Delta \vdash a;\mathcal{Q} \dashv \Delta'} \qquad \frac{\Delta \vdash [q]^{\Delta'}_{\mathcal{S}'};\mathcal{Q} \dashv \Delta''}{\Delta \vdash [a;q]^{\Delta'}_{\mathcal{S}'};\mathcal{Q} \dashv \Delta''}$$

$$\frac{\Delta \vdash \mathcal{Q} \dashv \Delta'}{\Delta \vdash a\langle e\rangle;\mathcal{Q} \dashv \Delta'} \qquad \frac{\Delta \vdash [q]^{\Delta'}_{\mathcal{S}'};\mathcal{Q} \dashv \Delta''}{\Delta \vdash [a\langle e\rangle;q]^{\Delta'}_{\mathcal{S}'};\mathcal{Q} \dashv \Delta''}$$

$$\frac{}{\Delta \vdash \varepsilon \dashv \varepsilon} \qquad \frac{\Delta \uplus \Delta' \vdash \mathcal{Q} \dashv \Delta''}{\Delta \vdash [\varepsilon]^{\Delta'}_{\mathcal{S}'};\mathcal{Q} \dashv \Delta' \uplus \Delta''}$$

$$\frac{\Delta \vdash \mathbf{do}\ e : \varepsilon \qquad \Delta \vdash \mathcal{Q} \dashv \Delta'}{\Delta \vdash \mathbf{do}\ e;\mathcal{Q} \dashv \Delta'}$$

$$\frac{\Delta \uplus \Delta' \vdash r : \Delta'' \qquad \Delta \uplus \Delta' \vdash [q]^{\varepsilon}_{\varepsilon};\mathcal{Q} \dashv \Delta'''}{\Delta \vdash [r;q]^{\Delta'}_{\mathcal{S}'};\mathcal{Q} \dashv \Delta' \uplus \Delta'' \uplus \Delta'''}$$

With these auxiliary definitions, we now introduce how interaction operations are added to the running application. The following rule enables the addition of an interaction operation $\mathbf{do}\ e$ to the end of the queue

$$\text{(Interaction)}\ \frac{\Delta \vdash \mathcal{Q} \dashv \Delta' \qquad \Delta \uplus \Delta' \vdash \mathbf{do}\ e : \varepsilon}{\Delta;\mathcal{S};\mathcal{Q} \xrightarrow{\mathbf{do}\ e} \Delta;\mathcal{S};(\mathcal{Q};\mathbf{do}\ e)}$$

Notice that the operation $\mathbf{do}\ e$ is typed under the typing environment $\Delta \uplus \Delta'$, corresponding to the combination of the current system typing environment Δ with the typing environment Δ' that will be produced by the queue $\mathcal{Q}$. This ensures that the new operation takes into account all pending events still queued in $\mathcal{Q}$. Also, recall that interactions operations do not change the running system definition, hence the resulting typing environment ε.

In order to add construction operations, we need one more auxiliary definition that, given a construction operation, produces a queue construction block.

Definition 3.13 (Construction Block Operations)**.** We define the transformation of a construction operation r into a construction block as follows

$$[\![\mathbf{var}\ a{=}e]\!] = [\mathbf{var}\ a{=}e] \qquad [\![r \circledast r']\!] = [\![r]\!] +\!\!+ [\![r']\!]$$
$$[\![\mathbf{def}\ a{=}e]\!] = [\mathbf{def}\ a{=}e] \qquad [q] +\!\!+ [q'] \triangleq [q;q']$$

Next, we present the rule that enqueues a construction operation r at the end of the queue

$$\text{(Construction)}\ \frac{\Delta \vdash \mathcal{Q} \dashv \Delta' \qquad \Delta \uplus \Delta' \vdash r : \Delta'' \qquad \Delta \uplus \Delta' \vDash \Delta''}{\Delta;\mathcal{S};\mathcal{Q} \xrightarrow{r} \Delta;\mathcal{S};(\mathcal{Q};[\![r]\!]^{\varepsilon}_{\varepsilon})}$$

Similarly, we gather the remaining queue effects Δ', and the new operation r is typed under the combined typing environment $\Delta \uplus \Delta'$. Since, in this case, we are reconfiguring the system, we must check that the new operation is compatible with the currently running system, therefore $\Delta \uplus \Delta' \vDash \Delta''$. This extra verification ensures that the queued operation r will not break type safety, and prevent a divergence in the propagation of changes. Notice that the construction block initial typing environment and state are both empty (ε). These will later be constructed by the operational semantics rules that reduce the construction block (S-RVar, S-RDef, S-RQueue, S-RStep and S-RValue).

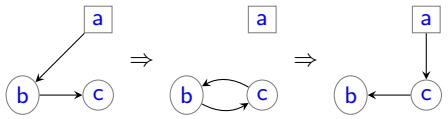
With the notion of compatibility (Definition 3.11), we allow for a composition construction operation to introduce transient inconsistencies in-between *sub*-operations (e.g. cyclic definitions). However, at the end of the composition operation we ensure that all those inconsistencies are solved. To illustrate this effect, consider the following example

```
var a=1
def b=a + 2
def c=b + 3
```

To swap the definitions of names b and c, one would write

$$\textbf{def } \mathsf{b}{=}\mathsf{c}+3 \circledast \textbf{def } \mathsf{c}{=}\mathsf{a}+2$$

which corresponds to the following data dependency graphs



The first graph depicts the state immediately before the composition ($\circledast$) above. After the first operation (redefinition of name b), dependencies are changed and the dependency graph contains a cycle. In the final step, we redefine name c, and the cycle is removed. The right most graph depicts the final state. Notice that this is only possible due to our compatibility notion (Definition 3.11), that ensures that at the end of the construction operation the graph remains acyclic. Without the composition operator, the above would not be possible in a simple way. The intermediate graph illustrates a program containing unguarded cyclic dependencies, hence is not typable in our type system, and would diverge. However, in this example, the re-definition of name c re-establishes the soundness of the dependency graph. In conclusion, our composition operator enables the (re)definition of several defined names that introduce intermediate inconsistencies, while ensuring that in the end the running application is sound. In Section 4 we present our main results that include type safety, and the convergence of propagation of changes.

4. Type Safety

In this section we present the soundness results for our language, that are based on proving safety (progress and preservation), and convergence of the propagation of changes. Besides common typing errors, our system also statically ensures that the intended reactive behavior is kept sound even when the system is reconfigured. Our functional core can be orthogonally extended, e.g. to include general recursion. The results presented here follow the syntactic approach of [27]. We state our results and provide full proofs in a companion technical report available online [10]. All results are based on the typing of runtime configurations, which in turn relies on a well-formed relation on queues, and well-typed relations on states and expressions.

The typing judgment for states (Definition 4.1), written $\Delta \vdash \varepsilon * \mathcal{S}$, asserts that names in state $\mathcal{S}$ and their respective expressions and values are well-typed with relation to the typing environment Δ. The typing relation imposes that the domains of the typing environment and state are equal ($\mathsf{dom}(\Delta) = \mathsf{dom}(\mathcal{S})$), and that all names in the state $\mathcal{S}$ are inspected by moving them from the right-hand side of the $*$ to the left-hand side. For each name, we conclude that its corresponding expression is well-typed under the same typing environment Δ, with the typed dependencies set δ of the same name, according to the typing rules for expressions defined in Figure 5. For names containing a value in the state we also check that the value is well-typed.

Definition 4.1 (Well-Typed State)**.** A state $\mathcal{S}$ is well-typed with relation to a typing environment Δ, if $\Delta \vdash \varepsilon * \mathcal{S}$ is derivable by the following rules

$$\frac{}{\Delta \vdash \mathcal{S} * \varepsilon \ (\mathsf{dom}(\Delta) = \mathsf{dom}(\mathcal{S}))}\ (\textsc{WTSEmpty})$$

$$\frac{\Delta(a) = \mathsf{var}_\delta(\tau) \quad \Delta; \varepsilon \overset{\delta[a:\mathsf{var}(\tau)]}{\vdash} e : \tau \quad \Delta; \varepsilon \overset{\delta[a:\mathsf{var}(\tau)]}{\vdash} v : \tau \quad \Delta \vdash \mathcal{S}, a \mapsto (e, \nu) * \mathcal{S}'}{\Delta \vdash \mathcal{S} * \mathcal{S}', a \mapsto (e, v)}\ (\textsc{WTSVarValue})$$

$$\frac{\Delta(a) = \mathsf{var}_\delta(\tau) \quad \Delta; \varepsilon \overset{\delta}{\vdash} e : \tau \quad \Delta \vdash \mathcal{S}, a \mapsto (e, \square) * \mathcal{S}'}{\Delta \vdash \mathcal{S} * \mathcal{S}', a \mapsto (e, \square)}\ (\textsc{WTSVar})$$

$$\frac{\Delta(a) = \mathsf{def}_\delta(\tau) \quad \Delta; \varepsilon \overset{\delta}{\vdash} e : \tau \quad \Delta; \varepsilon \overset{\delta}{\vdash} v : \tau \quad \Delta \vdash \mathcal{S}, a \mapsto (e, \nu) * \mathcal{S}'}{\Delta \vdash \mathcal{S} * \mathcal{S}', a \mapsto (e, v)}\ (\textsc{WTSDefValue})$$

$$\frac{\Delta(a) = \mathsf{def}_\delta(\tau) \quad \Delta; \varepsilon \overset{\delta}{\vdash} e : \tau \quad \Delta \vdash \mathcal{S}, a \mapsto (e, \square) * \mathcal{S}'}{\Delta \vdash \mathcal{S} * \mathcal{S}', a \mapsto (e, \square)}\ (\textsc{WTSDef})$$

We also define a well-typed expression configuration (Definition 4.2), that builds on the definitions of well-typed state (Definition 4.1), typing rules for expressions (Figure 5), and well-formed typed dependencies (Definition 3.5).

Definition 4.2 (Well-Typed Expression Configuration)**.** An expression configuration, written $(\mathcal{S}; e)$, is well-typed wrt. an environment Δ, if $\Delta \vdash (\mathcal{S}; e)$ is derivable by the rule

$$\frac{\Delta \vdash \varepsilon * \mathcal{S} \quad \Delta; \varepsilon \overset{\delta}{\vdash} e : \tau \quad \Delta \vdash \delta}{\Delta \vdash (\mathcal{S}; e)}$$

The notion of well-formed queue (Definition 4.3) with relation to a typing environment and state, written $\Delta; \mathcal{S} \vdash \mathcal{Q}$, asserts that all queued operations are well-typed. This notion relies on the typing rules for operations and expressions (Section 3.1), well-typed expression configuration (Definition 4.2), and well-typed state (Definition 4.1).

Definition 4.3 (Well-Formed Queue)**.** A queue $\mathcal{Q}$ is well-formed, wrt. an environment Δ and a state $\mathcal{S}$, if $\Delta; \mathcal{S} \vdash \mathcal{Q}$ can be inductively derived by the rules in Figure 9.

We establish that the empty queue is always well-formed. For the other cases, we inductively check the queue. Construction blocks $([q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q})$ are sequentially inspected, and in the case of an empty construction block we assure that the resulting construction block typing environment (Δ') is compatible with the system (according to Definition 3.11), and that the combined state ($\mathcal{S} \uplus \mathcal{S}'$) is well-typed with relation to the combined typing environment ($\Delta \uplus \Delta'$).

Finally, we define a well-typed runtime configuration (Definition 4.4), relying on the previous definitions of well-typed state (Definition 4.1), well-formed queue (Definition 4.3), and compatibility (Definition 3.11). With this definition we assert, via $\varepsilon \vDash \Delta$, that all names and respective typed dependencies sets are well-typed, are there are no cyclic definitions, which is essential to ensure that the propagation of changes does not diverge.

Definition 4.4 (Well-Typed Runtime Configuration)**.** A runtime configuration, written $(\Delta; \mathcal{S}; \mathcal{Q})$, is well-typed if $\vdash (\Delta; \mathcal{S}; \mathcal{Q})$ is derivable by the following rule

$$\frac{\Delta \vdash \varepsilon * \mathcal{S} \quad \Delta; \mathcal{S} \vdash \mathcal{Q} \quad \varepsilon \vDash \Delta}{\vdash (\Delta; \mathcal{S}; \mathcal{Q})}$$

Based on these auxiliary definitions, we establish type safety, and present here the main results: progress (Theorem 4.5), type preservation (Theorem 4.6). These results are based on intermediate results for expressions (progress and preservation). For full details and proofs, please refer to [10].

We can now establish the runtime progress (Theorem 4.5), where we ensure a stronger invariant of the reduction relation, stating that all names in the current state have a value associated ($\mathcal{S}_\mathsf{v}(a) = v$).

Theorem 4.5 (Runtime Progress)**.** For all runtime configurations $(\Delta; \mathcal{S}; \mathcal{Q})$, if $\vdash (\Delta; \mathcal{S}; \mathcal{Q})$ and $\mathcal{Q} \neq \varepsilon$ and $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_\mathsf{v}(a) =$

$$\frac{}{\Delta;\mathcal{S}\vdash\varepsilon}\ (\text{WFQEMPTY})$$

$$\frac{\Delta\vdash\mathbf{do}\ e:\varepsilon\quad\Delta\vdash(\mathcal{S};e)\quad\Delta;\mathcal{S}\vdash\mathcal{Q}}{\Delta;\mathcal{S}\vdash\mathbf{do}\ e;\mathcal{Q}}\ (\text{WFQDO})$$

$$\frac{\Delta(a)=\sigma\quad\mathcal{S}(a)=(e,v)\quad\Delta\vdash(\mathcal{S};e)\quad\Delta\vdash(\mathcal{S};v)\quad\Delta;\mathcal{S}\vdash\mathcal{Q}}{\Delta;\mathcal{S}\vdash a;\mathcal{Q}}\ (\text{WFQNAME})$$

$$\frac{\Delta(a)=\mathsf{var}_\delta(\tau)\quad\Delta;\varepsilon\vdash^{\delta[a:\mathsf{var}(\tau)]}e:\tau\quad\Delta\vdash(\mathcal{S};e)\quad\Delta;\mathcal{S}\vdash\mathcal{Q}}{\Delta;\mathcal{S}\vdash a\langle e\rangle;\mathcal{Q}}\ (\text{WFQVAREVAL})$$

$$\frac{\Delta(a)=\mathsf{def}_\delta(\tau)\quad\Delta;\varepsilon\vdash^{\delta}e:\tau\quad\Delta\vdash(\mathcal{S};e)\quad\Delta;\mathcal{S}\vdash\mathcal{Q}}{\Delta;\mathcal{S}\vdash a\langle e\rangle;\mathcal{Q}}\ (\text{WFQDEFEVAL})$$

$$\frac{\Delta\uplus\Delta'\vdash\varepsilon*(\mathcal{S}\uplus\mathcal{S}')\quad\Delta\vDash\Delta'\quad\Delta\uplus\Delta';\mathcal{S}\uplus\mathcal{S}'\vdash\mathcal{Q}}{\Delta;\mathcal{S}\vdash[\varepsilon]^{\Delta'}_{\mathcal{S}'};\mathcal{Q}}\ (\text{WFQREMPTY})$$

$$\frac{\Delta\uplus\Delta'\vdash\mathbf{var}\ a{=}e:\Delta''\quad\Delta\uplus\Delta'\vdash(\mathcal{S}\uplus\mathcal{S}';e)\quad\Delta;\mathcal{S}\vdash[q]^{\Delta'\uplus\Delta''}_{\mathcal{S}'[a\mapsto(e,\square)]};\mathcal{Q}'}{\Delta;\mathcal{S}\vdash[\mathbf{var}\ a{=}e;q]^{\Delta'}_{\mathcal{S}'};\mathcal{Q}'}\ (\text{WFQRVAR})$$

$$\frac{(\Delta\uplus\Delta')(a)=\mathsf{var}_\delta(\tau)\quad(\mathcal{S}\uplus\mathcal{S}')(a)=(e,\nu)\quad\Delta\uplus\Delta';\varepsilon\vdash^{\delta}e:\tau\quad\Delta\uplus\Delta'\vdash(\mathcal{S}\uplus\mathcal{S}';e)\quad\Delta;\mathcal{S}\vdash[q]^{\Delta'}_{\mathcal{S}'};\mathcal{Q}'}{\Delta;\mathcal{S}\vdash[a;q]^{\Delta'}_{\mathcal{S}'};\mathcal{Q}'}\ (\text{WFQRVARNAME})$$

$$\frac{\Delta\uplus\Delta'\vdash\mathbf{def}\ a{=}e:\Delta''\quad\Delta\uplus\Delta'\vdash(\mathcal{S}\uplus\mathcal{S}';e)\quad\Delta;\mathcal{S}\vdash[q]^{\Delta'\uplus\Delta''}_{\mathcal{S}'[a\mapsto(e,\square)]};\mathcal{Q}'}{\Delta;\mathcal{S}\vdash[\mathbf{def}\ a{=}e;q]^{\Delta'}_{\mathcal{S}'};\mathcal{Q}'}\ (\text{WFQRDEF})$$

$$\frac{(\Delta\uplus\Delta')(a)=\mathsf{def}_\delta(\tau)\quad(\mathcal{S}\uplus\mathcal{S}')(a)=(e,\nu)\quad\Delta\uplus\Delta';\varepsilon\vdash^{\delta}e:\tau\quad\Delta\uplus\Delta'\vdash(\mathcal{S}\uplus\mathcal{S}';e)\quad\Delta;\mathcal{S}\vdash[q]^{\Delta'}_{\mathcal{S}'};\mathcal{Q}'}{\Delta;\mathcal{S}\vdash[a;q]^{\Delta'}_{\mathcal{S}'};\mathcal{Q}'}\ (\text{WFQRDEFNAME})$$

$$\frac{(\Delta\uplus\Delta')(a)=\mathsf{var}_\delta(\tau)\quad\Delta\uplus\Delta';\varepsilon\vdash^{\delta}e:\tau\quad\Delta\uplus\Delta'\vdash(\mathcal{S}\uplus\mathcal{S}';e)\quad\Delta;\mathcal{S}\vdash[q]^{\Delta'}_{\mathcal{S}'};\mathcal{Q}'}{\Delta;\mathcal{S}\vdash[a\langle e\rangle;q]^{\Delta'}_{\mathcal{S}'};\mathcal{Q}'}\ (\text{WFQRVAREVAL})$$

$$\frac{(\Delta\uplus\Delta')(a)=\mathsf{def}_\delta(\tau)\quad\Delta\uplus\Delta';\varepsilon\vdash^{\delta}e:\tau\quad\Delta\uplus\Delta'\vdash(\mathcal{S}\uplus\mathcal{S}';e)\quad\Delta;\mathcal{S}\vdash[q]^{\Delta'}_{\mathcal{S}'};\mathcal{Q}'}{\Delta;\mathcal{S}\vdash[a\langle e\rangle;q]^{\Delta'}_{\mathcal{S}'};\mathcal{Q}'}\ (\text{WFQRVAREVAL})$$

Figure 9. Well-Formed Queue.

v then, there is a program configuration $(\Delta';\mathcal{S}';\mathcal{Q}')$ such that $\Delta;\mathcal{S};\mathcal{Q}\to\Delta';\mathcal{S}';\mathcal{Q}'$ and $\forall a\in\mathsf{dom}(\mathcal{S}').\mathcal{S}'_\nu(a)=v$.

Runtime type preservation (Theorem 4.6) follows standard lines, and is proven by induction on the length of the reduction $\Delta;\mathcal{S};\mathcal{Q}\to\Delta';\mathcal{S}';\mathcal{Q}'$.

Theorem 4.6 (Runtime Type Preservation)**.** For all runtime configurations $(\Delta;\mathcal{S};\mathcal{Q})$ and $(\Delta';\mathcal{S}';\mathcal{Q}')$, if $\vdash\Delta;\mathcal{S};\mathcal{Q}$, and $\Delta;\mathcal{S};\mathcal{Q}\to\Delta';\mathcal{S}';\mathcal{Q}'$ then, $\vdash\Delta';\mathcal{S}';\mathcal{Q}'$.

With progress and preservation, we now introduce some auxiliary definitions that will allow us to prove the convergence of the propagation of changes (Theorem 4.8). We define the length of a queue $\mathcal{Q}$, written $\#_\Delta(\mathcal{Q})$, with relation to a typing environment Δ, as the sum of the lengths of all its operations. Intuitively, the length of a queue represents the total number of computational steps needed to execute all queued operations and propagate changes throughout the subscribers. The formal definition of the length of a queue can be found in [10]. Based on this measure, we next provide our main queue result.

Lemma 4.7 (Bound Queue Length)**.** If $\vdash(\Delta;\mathcal{S};\mathcal{Q})$ and $\Delta;\mathcal{S};\mathcal{Q}\to\Delta';\mathcal{S}';\mathcal{Q}'$ then one of the following holds

i. $\mathcal{Q}'=\mathbf{do}\ e;\mathcal{Q}''$ and $\#_\Delta(\mathcal{Q})=\#_{\Delta'}(\mathcal{Q}')$
ii. $\mathcal{Q}'=a\langle e\rangle;\mathcal{Q}''$ and $\#_\Delta(\mathcal{Q})=\#_{\Delta'}(\mathcal{Q}')$
iii. $\mathcal{Q}'=[a\langle e\rangle;q'']^{\Delta''}_{\mathcal{S}''};\mathcal{Q}'''$ and $\#_\Delta(\mathcal{Q})=\#_{\Delta'}(\mathcal{Q}')$
iv. $\#_\Delta(\mathcal{Q})>\#_{\Delta'}(\mathcal{Q}')$

The first three cases of Lemma 4.7 depend on the reduction of expressions, therefore the queue length remains the same ($\#_\Delta(\mathcal{Q})=\#_{\Delta'}(\mathcal{Q}')$). Notice that in these cases, if the expression reduction terminates, we reach a value, and then the queue length decreases (S-DO-ACTION, S-VALUE and S-RVALUE for each case respectively). In the fourth case, we prove that the queue length strictly decreases ($\#_\Delta(\mathcal{Q})>\#_{\Delta'}(\mathcal{Q}')$). Based on a strong normalizing result for λ-calculus (c.f. [25]), we are able to establish an even stronger result, where all well-typed runtime configurations with a non-empty queue, reach a runtime configuration with an empty queue after a finite number of steps.

Theorem 4.8 (Convergence)**.** For all runtime configurations $(\Delta;\mathcal{S};\mathcal{Q})$, if $\vdash(\Delta;\mathcal{S};\mathcal{Q})$ then $\Delta;\mathcal{S};\mathcal{Q}\to^*\Delta';\mathcal{S}';\varepsilon$.

Notice that this result is based (and parametric) on the termination of the functional core, which in our case is a λ-calculus extended with collections. Using any expression language free of side-effects, for which we can prove termination (e.g. [3, 18]), then the result also holds. With the termination of the expression language then the propagation of changes also terminates. Since the queue length never increases when the expression reduction reaches a value, the whole reaction process eventually terminates.

In summary, the main results we extract from our operational semantics and type system are type safety (Theorems 4.5 and 4.6), and convergence of the change propagation (Theorem 4.8). These results allow us to ensure that the automatic propagation of changes happens without causing infinite propagation loops, and dynamic reconfiguration of both code and data does not break type safety.

5. Related Work

Several works on dynamic reconfiguration and incremental computation of software systems have already been proposed [14, 24]. These solutions are mostly designed for imperative programming languages. For instance, DSU [14] provides dynamic updates in programs by explicitly defining points where updates may occur. When the execution of a program reaches an update point, if a dynamic update is available, it is applied. Up to some point, our approach is similar. We do not explicitly define update points, but we interpolate interaction operations with construction operations of the system. By combining reactivity with dynamic reconfiguration we are able to maintain a dependency graph that changes dynamically, while maintaining the typing invariants. In this paper

we combine dynamic reconfiguration with a imperative and reactive programming language.

Data-flow programming languages have been extensively studied in the past, and summarized in [16]. The pure data-flow model considers that every node is pure and without side effects. Likewise, in our language, a node defined by an operation (**var** or **def**) does not have side effects. Many data-flow languages also explore parallel computation by computing non-interfering nodes simultaneously. We focus on the core evolution model, but believe that a notion of separation based on name dependencies, although orthogonal, can be directly applied to our scheduling discipline.

Adaptive Functional Programming (AFP) [2] uses an underlying dependency graph and a change propagation algorithm to adapt the output when input changes. Based on the dependency graph, the algorithm builds a priority queue of nodes, and executes until the queue is emptied. AFP encodes this algorithm into traces in the operational semantics. We use a similar technical approach, where propagation is also encoded into operational semantics by the queue. Our approach also goes further by providing a reactive framework that allows to safely dynamically reconfigure and evolve applications.

Reactive systems [11, 17, 20, 21] maintain an ongoing interaction with their environment. Our incremental style of programming also maintains an ongoing interaction with the environment, where reconfigurations and execution of actions can change the application state and code. Imperative reactive languages [5, 6, 12] usually require explicit constructs to enable reactivity. Several Functional Reactive Programming approaches have been proposed [17, 21, 26]. However, neither of these approaches combines reactivity with a statically verified reconfiguration mechanism as the one presented in this paper. In FrTime [9], reactivity is embedded into a call-by-value dynamic programming language. In principle, an untyped fragment of our language can be encoded into the FrTime's framework. However, with such untyped fragment, it would be possible to define programs with transient and inconsistent states causing an infinite propagation of changes. Since, in our approach, we statically verify programs, we ensure that the propagation of changes does not diverge, and that all reconfigurations are safe, thus making it impossible to define unguarded (infinite) propagation cycles.

TouchDevelop [7] provides a programming environment with immediate feedback in the development of user interfaces and small applications. It provides a statically verified approach that separates presentation and application code. The edit-compile-run cycle is tightened by allowing the display code to be refreshed without restarting programs. Although related in their goals, both approach target different kind of scenarios. We target the development of web and cloud applications where the correct evolution of state is of major importance.

6. Final Remarks

We have introduced a core language, its type system and operational semantics, that is suitable for safe and incremental construction of live software systems. Our language is the first approach that is uniform, statically type safe, integrated, and supporting the dynamic evolution mechanism for both code and data. Our main results include the soundness (progress and type preservation) of the whole system, and the convergence of the propagation of changes. We also presented a working prototype of a live development environment, available at [1], that allows to build and evolve web applications using the techniques presented in this paper.

We identify some future challenges in our framework, both in the formal and in the pragmatic domains. On the pragmatic side, we may extend the language expressiveness by parameterizing the model with a full-fledged functional or imperative language; extend the data manipulation layer to interact with known database management systems; and use realistic web design languages. All these extensions are being pursued by developments in our prototype [1]. On the formal side, conspicuous extensions include the concurrent and parallel scheduling of the queue operations and corresponding static type discipline based on notions of separation. Another interesting topic relates to the representation of different shades of data persistence, e.g. session data, that contrasts with the globally shared data represented here.

References

[1] Prototype. http://tiny.cc/prototype.

[2] U. A. Acar, G. E. Blelloch, and R. Harper. Adaptive Functional Programming. *ACM TOPLAS*, 28(6), 2006.

[3] G. Barthe, M. J. Frade, E. Giménez, L. Pinto, and T. Uustalu. Type-based termination of recursive definitions. *Mathematical. Structures in Computer Science*, 2004.

[4] P. Bhattacharya and I. Neamtiu. Dynamic Updates for Web and Cloud Applications. In *Proceedings of APLWACA*, 2010.

[5] F. Boussinot. Reactive C: an extension of C to program reactive systems. *Software: Practice and Experience*, 21(4), 1991.

[6] F. Boussinot and J.-F. Susini. The SugarCubes tool box: a reactive Java framework. *Software: Practice and Experience*, 28(14), 1998.

[7] S. Burckhardt, M. Fahndrich, P. de Halleux, S. McDirmid, M. Moskal, N. Tillmann, and J. Kato. It's Alive! Continuous Feedback in UI Programming. In *Proceedings of PLDI*, 2013.

[8] Y. Chen, J. Dunfield, and U. A. Acar. Type-directed automatic incrementalization. In *Proceedings of PLDI*, 2012.

[9] G. H. Cooper and S. Krishnamurthi. Embedding dynamic dataflow in a call-by-value language. In *Proceedings of ESOP*, 2006.

[10] M. Domingues and J. C. Seco. Type Safe Evolution of Live Systems. Technical report, NOVA-LINCS – UNL, 2015. http://miguel.domingues.pt.

[11] N. Halbwachs. Synchronous Programming of Reactive Systems. In *Proceedings of CAV*, 1998.

[12] M. A. Hammer, U. A. Acar, and Y. Chen. CEAL: a C-based language for self-adjusting computation. In *Proceedings of PLDI*, 2009.

[13] C. M. Hancock. *Real-time programming and the big ideas of computational literacy*. PhD thesis, Massachusetts Institute of Technology, 2003.

[14] M. Hicks and S. Nettles. Dynamic Software updating. *ACM TOPLAS*, 27(6), 2005.

[15] M. Hofmann, B. C. Pierce, and D. Wagner. Symmetric lenses. In *Proceedings of POPL*, 2011.

[16] W. M. Johnston, J. R. P. Hanna, and R. J. Millar. Advances in Dataflow Programming Languages. *ACM Computing Surveys*, 36(1), 2004.

[17] L. Mandel and M. Pouzet. ReactiveML: a reactive extension to ML. In *Proceedings of PPDP*, 2005.

[18] D. McAllester and K. Arkoudas. Walther recursion. In *Automated Deduction—Cade-13*, pages 643–657. Springer, 1996.

[19] S. McDirmid. Usable live programming. In *SPLASH Onward!*, 2013.

[20] H. Nilsson, A. Courtney, and J. Peterson. Functional Reactive Programming, Continued. In *Proceedings of Workshop on Haskell*, 2002.

[21] R. R. Pucella. Reactive Programming in Standard ML. In *Proceedings of ICCL*, 1998.

[22] G. Ramalingam and T. Reps. A categorized Bibliography on Incremental Computation. In *Proceedings of POPL*, 1993.

[23] P. Sewell. Modules, abstract types, and distributed versioning. In *Proceedings of POPL*, 2001.

[24] G. Stoyle, M. Hicks, G. Bierman, P. Sewell, and I. Neamtiu. Mutatis Mutandis: Safe and Predictable Dynamic Software Updating. *ACM TOPLAS*, 29(4), 2007.

[25] W. W. Tait. Intensional interpretations of functionals of finite type i. *Journal of Symbolic Logic*, 32(2), 1967.

[26] Z. Wan, W. Taha, and P. Hudak. Real-time FRP. In *Proceedings of ICFP*, 2001.

[27] A. Wright and M. Felleisen. A syntactic approach to type soundness. *Information and Computation*, 1994.

[28] D. Yellin and R. Strom. INC: a language for incremental computations. In *Proceedings of PLDI*, 1988.

A. Extra Definitions

Definition A.1 (Substitution)**.** Given an expression e, a value v and an identifier x, the substitution relation for expression, written $e\{v/x\}$, replaces all occurrences of x by the value v in the expression e. We define the substitution relation for expressions as follows

$$
\begin{array}{lcl}
a\{v/x\} & = & a \\
\mathbf{b}\{v/x\} & = & \mathbf{b} \\
x\{v/x\} & = & v \\
y\{v/x\} & = & y \\
(\lambda y.e)\{v/x\} & = & \lambda y.(e\{v/x\}) \\
(e\ e')\{v/x\} & = & (e\{v/x\}\ )(e'\{v/x\}) \\
(e\ \mathsf{op}\ e')\{v/x\} & = & e\{v/x\}\ \mathsf{op}\ e'\{v/x\} \\
(e\ ?\ e' : e'')\{v/x\} & = & e\{v/x\}\ ?\ e'\{v/x\} : e''\{v/x\} \\
[e_0, \ldots, e_n]\{v/x\} & = & [e_0\{v/x\}, \ldots, e_n\{v/x\}] \\
(\mathbf{iter}(e, e', y.z.e''))\{v/x\} & = & \mathbf{iter}(e\{v/x\}, e'\{v/x\}, y.z.e''\{v/x\}) \\
\mathbf{action}\ \{a \leftarrow e\}\{v/x\} & = & \mathbf{action}\ \{a \leftarrow e\{v/x\}\} \\
\mathbf{match}\ e\ \mathbf{with}\ x{::}xs \rightarrow e' \mid [] \rightarrow e''\{v/x\} & = & \mathbf{match}\ e\{v/x\}\ \mathbf{with}\ y{::}ys \rightarrow e'\{v/x\} \mid [] \rightarrow e''\{v/x\}
\end{array}
$$

Definition A.2 (State Value)**.** The value of name a, with relation to state $\mathcal{S}$ such that $a \in \mathsf{dom}(\mathcal{S})$, written $\mathcal{S}_{\mathsf{v}}(a)$, is defined by

$$\mathcal{S}_{\mathsf{v}}(a) = \{\ \nu \quad \mathcal{S}(a) = (e, \nu)$$

Definition A.3 (State Expression)**.** The state value of name a, with relation to state $\mathcal{S}$ such that $a \in \mathsf{dom}(\mathcal{S})$, written $\mathcal{S}_{\mathsf{e}}(a)$, is defined by

$$\mathcal{S}_{\mathsf{e}}(a) = \{\ e \quad \mathcal{S}(a) = (e, \nu)$$

Definition A.4 (Typed Dependencies Subset)**.** We define the type dependency subset of two type dependencies sets δ' and δ, written $\delta' \sqsubseteq \delta$, as follows:

$$\frac{}{\varepsilon \sqsubseteq \delta}\ (\textsc{DepsSubsetEmpty})$$

$$\frac{\tau \prec \delta'(a) \qquad \delta \sqsubseteq \delta'}{\delta, a : \tau \sqsubseteq \delta'}\ (\textsc{DepsSubsetTau})$$

$$\frac{\mathsf{var}(\tau) \prec \delta'(a) \qquad \delta \sqsubseteq \delta'}{\delta, a : \mathsf{var}(\tau) \sqsubseteq \delta'}\ (\textsc{DepsSubsetVar})$$

Definition A.5 (Queue Length)**.** We define the length of a queue $\mathcal{Q}$, with relation to a typing environment Δ, written $\#_\Delta(\mathcal{Q})$, as follows:

$$
\begin{array}{lcl}
\#_\Delta(\varepsilon) & = & 0 \\
\#_\Delta(a; \mathcal{Q}) & = & 1 + \#_\Delta(\mathsf{subscribers}_\Delta(a); \mathcal{Q}) \\
\#_\Delta(a\langle e\rangle; \mathcal{Q}) & = & \#_\Delta(a; \mathcal{Q}) \\
\#_\Delta(\mathbf{do}\ e; \mathcal{Q}) & = & 1 + \#^{\mathsf{max}}_\Delta + \#_\Delta(\mathcal{Q}) \\
\#_\Delta([\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') = 1 + \#_{\Delta \uplus \Delta'}(\mathsf{subscribers}_{\Delta \uplus \Delta'}(\mathsf{dom}(\Delta')); \mathcal{Q}') & & \\
\#_\Delta([a; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') & = & 1 + \#_\Delta([q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') \\
\#_\Delta([a\langle e\rangle; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') & = & \#_\Delta([a; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') \\
\#_\Delta([\mathbf{var}\ a{=}e; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') & = & 1 + \#_\Delta([a; q]^{\Delta''}_{\mathcal{S}''}; \mathcal{Q}') \\
 & & \qquad \Delta \vdash [\mathbf{var}\ a{=}e]^{\Delta'}_{\mathcal{S}'} \dashv \Delta'' \\
\#_\Delta([\mathbf{def}\ a{=}e; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') & = & 1 + \#_\Delta([a; q]^{\Delta''}_{\mathcal{S}''}; \mathcal{Q}') \\
 & & \qquad \Delta \vdash [\mathbf{def}\ a{=}e]^{\Delta'}_{\mathcal{S}'} \dashv \Delta''
\end{array}
$$

Definition A.6 (Maximum Name Length)**.** The maximum name length, with relation to typing environment Δ, written $\#^{\mathsf{max}}_\Delta$, is defined by

$$\#^{\mathsf{max}}_\Delta = \mathsf{max}\,(\{\#_\Delta(a) \mid a \in \mathsf{dom}(\Delta)\})$$

B. Results

Corollary B.1 (Well-Type State Domain)**.** For all well-typed states $\Delta \vdash \varepsilon * \mathcal{S}$, then $\mathsf{dom}(\Delta) = \mathsf{dom}(\mathcal{S})$.

Corollary B.2 (Well-Type Dependencies Domain)**.** For all well-typed dependencies $\Delta \vdash \delta$, then $\mathsf{dom}(\delta) \subseteq \mathsf{dom}(\Delta)$.

B.1 Expressions

Lemma B.3 (Dependencies Weakening)**.** For all typing environments Δ and Γ and expressions e and dependencies environment δ and δ, such that $\Delta; \Gamma \overset{\delta}{\vdash} e : \tau$ and $\Delta \vdash \delta'$ and $\delta \sqsubseteq \delta'$, and a dependencies environment δ then $\Delta; \Gamma \overset{\delta'}{\vdash} e : \tau$.

Proof. By induction on the derivation $\Delta; \Gamma \overset{\delta}{\vdash} e : \tau$.

1. T-Name

(H1) $\Delta; \Gamma \overset{\delta}{\vdash} a : \tau$
(H2) $\Delta \vdash \delta'$
(H3) $\delta \sqsubseteq \delta'$
(4) $\lfloor \Delta(a) \rfloor \prec \delta(a)$ — by inv. T-Name in (H1)
(5) $\delta'(a) = \delta(a)$ — by Definition 3.5 and (H2, H3)
(6) $\lfloor \Delta a \rfloor \prec \delta'(a)$ — from (4, 5)
$\star$ $\Delta; \Gamma \overset{\delta'}{\vdash} a : \tau$ — by T-Name with (6)

2. T-ID

(H1) $\Delta; \Gamma, x : \tau \overset{\delta}{\vdash} x : \tau$
(H2) $\Delta \vdash \delta'$
(H3) $\delta \sqsubseteq \delta'$
$\star$ $\Delta; \Gamma, x : \tau \overset{\delta'}{\vdash} x : \tau$ — by T-ID

3. T-BaseValue

(H1) $\Delta; \Gamma \overset{\delta}{\vdash} \mathbf{b} : \beta(\mathbf{b})$
(H2) $\Delta \vdash \delta'$
(H3) $\delta \sqsubseteq \delta'$
$\star$ $\Delta; \Gamma \overset{\delta'}{\vdash} \mathbf{b} : \beta(\mathbf{b})$ — by T-BaseValue

4. T-Abstraction

(H1) $\Delta; \Gamma \overset{\delta}{\vdash} \lambda x.e : \tau \xrightarrow{\delta''} \tau'$
(H2) $\Delta \vdash \delta'$
(H3) $\delta \sqsubseteq \delta'$
$\star$ $\Delta; \Gamma \overset{\delta'}{\vdash} \lambda x.e : \tau \xrightarrow{\delta''} \tau'$ — by T-Abstraction

5. T-Application

(H1) $\Delta; \Gamma \overset{\delta}{\vdash} e\ e' : \tau'$
(H2) $\Delta \vdash \delta'$
(H3) $\delta \sqsubseteq \delta'$
(4) $\Delta; \Gamma \overset{\delta}{\vdash} e : \tau \xrightarrow{\delta''} \tau'$ — by inv. T-Application in (H1)
(5) $\Delta; \Gamma \overset{\delta}{\vdash} e' : \tau$ — by inv. T-Application in (H1)
(6) $\Delta; \Gamma \overset{\delta'}{\vdash} e : \tau \xrightarrow{\delta''} \tau'$ — by I.H. with (4, H2, H3)
(7) $\Delta; \Gamma \overset{\delta'}{\vdash} e' : \tau$ — by I.H. with (5, H2, H3)
$\star$ $\Delta; \Gamma \overset{\delta'}{\vdash} e\ e' : \tau'$ — by T-Application with (6, 7)

6. T-BinaryOp

(H1) $\Delta; \Gamma \overset{\delta}{\vdash} e\ \mathsf{op}\ e' : \tau''$
(H2) $\Delta \vdash \delta'$
(H3) $\delta \sqsubseteq \delta'$
(4) $\Delta; \Gamma \overset{\delta}{\vdash} e : \tau^{\delta}$ — by inv. T-BinaryOp in (H1)
(5) $\Delta; \Gamma \overset{\delta}{\vdash} e' : \tau'$ — by inv. T-BinaryOp in (H1)
(6) $\mathsf{op}: \tau \to \tau' \to \tau''$ — by inv. T-BinaryOp in (H1)
(7) $\Delta; \Gamma \overset{\delta'}{\vdash} e : \tau$ — by I.H. with (4, H2, H3)
(8) $\Delta; \Gamma \overset{\delta'}{\vdash} e' : \tau'^{\delta'}$ — by I.H. with (5, H2, H3)
$\star$ $\Delta; \Gamma \overset{\delta'}{\vdash} e\ \mathsf{op}\ e' : \tau''$ — by T-BinaryOp with (7, 8, 6)

7. T-If

(H1) $\Delta; \Gamma \overset{\delta}{\vdash} e\ ?\ e' : e'' : \tau$
(H2) $\Delta \vdash \delta'$
(H3) $\delta \sqsubseteq \delta'$
(4) $\Delta; \Gamma \overset{\delta}{\vdash} e : \mathsf{Bool}$ — by inv. T-If in (H1)
(5) $\Delta; \Gamma \overset{\delta}{\vdash} e : \tau$ — by inv. T-If in (H1)
(6) $\Delta; \Gamma \overset{\delta}{\vdash} e : \tau$ — by inv. T-If in (H1)
(7) $\Delta; \Gamma \overset{\delta'}{\vdash} e : \mathsf{Bool}$ — by I.H. with (4, H2, H3)
(8) $\Delta; \Gamma \overset{\delta'}{\vdash} e' : \tau$ — by I.H. with (4, H2, H3)
(9) $\Delta; \Gamma \overset{\delta'}{\vdash} e' : \tau$ — by I.H. with (4, H2, H3)

⋆ $\Delta; \Gamma \vdash^{\delta'} e\ ?\ e' : e'' : \tau$ by T-If with (7, 8, 9)

8. T-ForEach

(H1) $\Delta; \Gamma \vdash^{\delta} \mathbf{iter}(e, e', x.y.e'') : \tau'$
(H2) $\Delta \vdash \delta'$
(H3) $\delta \sqsubseteq \delta'$
(4) $\Delta; \Gamma \vdash^{\delta} e : \tau^*$ by inv. T-ForEach in (H1)
(5) $\Delta; \Gamma \vdash^{\delta} e' : \tau'$ by inv. T-ForEach in (H1)
(6) $\Delta; \Gamma, x : \tau, y : \tau' \vdash^{\delta} e'' : \tau'$ by inv. T-ForEach in (H1)
(7) $\Delta; \Gamma \vdash^{\delta'} e : \tau^*$ by I.H. with (4, H2, H3)
(8) $\Delta; \Gamma \vdash^{\delta'} e' : \tau'$ by I.H. with (5, H2, H3)
(9) $\Delta; \Gamma, x : \tau, y : \tau' \vdash^{\delta'} e'' : \tau'$ by I.H. with (6, H2, H3)
⋆ $\Delta; \Gamma \vdash^{\delta'} \mathbf{iter}(e, e', x.y.e'') : \tau'$ by T-ForEach with (7, 8, 9)

9. T-Match

(H1) $\Delta; \Gamma \vdash^{\delta} \mathbf{match}\ e\ \mathbf{with}\ x{::}xs \to e' \mid [] \to e'' : \tau'$
(H2) $\Delta \vdash \delta'$
(H3) $\delta \sqsubseteq \delta'$
(4) $\Delta; \Gamma \vdash^{\delta} e : \tau^*$ by inv. T-Match in (H1)
(5) $\Delta; \Gamma, x : \tau, xs : \tau^* \vdash^{\delta} e' : \tau'$ by inv. T-Match in (H1)
(6) $\Delta; \Gamma \vdash^{\delta} e'' : \tau'$ by inv. T-Match in (H1)
(7) $\Delta; \Gamma \vdash^{\delta'} e : \tau^*$ by I.H. with (4, H2, H3)
(8) $\Delta; \Gamma, x : \tau, xs : \tau^* \vdash^{\delta'} e' : \tau'$ by I.H. with (5, H2, H3)
(9) $\Delta; \Gamma \vdash^{\delta'} e'' : \tau'$ by I.H. with (6, H2, H3)
⋆ $\Delta; \Gamma \vdash^{\delta'} \mathbf{match}\ e\ \mathbf{with}\ x{::}xs \to e' \mid [] \to e'' : \tau'$ by T-Match with (7, 8, 9)

10. T-Action

(H1) $\Delta, a : \mathsf{var}_{\delta''}(\tau); \Gamma \vdash^{\delta} \mathbf{action}\ \{a \leftarrow e\} : \mathsf{Action}(\delta''[a : \mathsf{var}(\tau)])$
(H2) $\Delta \vdash \delta'$
(H3) $\delta \sqsubseteq \delta'$
⋆ $\Delta, a : \mathsf{var}_{\delta''}(\tau); \Gamma \vdash^{\delta'} \mathbf{action}\ \{a \leftarrow e\} : \mathsf{Action}(\delta''[a : \mathsf{var}(\tau)])$ by T-Action
□

Lemma B.4 (Substitution)**.** For all expressions e and typing environments Δ and Γ, if $\Delta; \Gamma, x : \tau' \vdash^{\delta} e : \tau$ and a value v such that $\Delta; \Gamma \vdash^{\delta} v : \tau'$ then $\Delta; \Gamma \vdash^{\delta} e\{v/x\} : \tau$.

Proof. By induction on the derivation $\Delta; \Gamma, x : \tau' \vdash^{\delta} e : \tau$.

1. T-Name

(H1) $\Delta; \Gamma, x : \tau' \vdash^{\delta} a : \tau$
(H2) $\Delta; \Gamma \vdash^{\delta} v : \tau'$
(3) $a\{v/x\} = a$ by Definition A.1
⋆ $\Delta; \Gamma \vdash^{\delta} a\{v/x\} : \tau$ by T-Name and (3)

2. T-ID

(H1) $\Delta; \Gamma, x : \tau' \vdash^{\delta} y : \tau$
(H2) $\Delta; \Gamma \vdash^{\delta} v : \tau'$

(3) Case: $x \neq y$
(4) $y\{v/x\} = y$ by Definition A.1 with (3)
⋆ $\Delta; \Gamma \vdash^{\delta} y : \tau$ by T-ID and (4)

(6) Case: $x = y$
(7) $\Delta; \Gamma, x : \tau' \vdash^{\delta} x : \tau$ from (6)
(8) $\tau = \tau'$ from (7)
(9) $x\{v/x\} = v$ by Definition A.1
⋆ $\Delta; \Gamma \vdash^{\delta} x\{v/x\} : \tau$ from (H2, 8, 9)

3. T-BaseValue

(H1) $\Delta; \Gamma, x : \tau' \vdash^{\delta} \mathbf{b} : \beta(\mathbf{b})$
(H2) $\Delta; \Gamma \vdash^{\delta} v : \tau'$
(3) $\mathbf{b}\{v/x\} = \mathbf{b}$

$\star$ $\Delta; \Gamma \vdash^{\delta} \mathbf{b}\{v/x\} : \beta(\mathbf{b})$ by T-BaseValue and (3)

4. T-Abstraction

(H1) $\Delta; \Gamma, x : \tau'' \vdash^{\delta} \lambda y.e : \tau \xrightarrow{\delta'} \tau'$
(H2) $\Delta; \Gamma \vdash^{\delta} v : \tau''$
(3) $\Delta; \Gamma, x : \tau'', y : \tau \vdash^{\delta'} e : \tau'$ by inv. T-Abstraction in (H1)
(4) $\Delta; \Gamma \vdash^{\delta'} v : \tau''$ from T-ID, T-BaseValue, T-Abstraction, T-Action and (H2)
(5) $\Delta; \Gamma, y : \tau \vdash^{\delta'} e\{v/x\} : \tau'$ by I.H. with (3, 4)
(6) $\Delta; \Gamma \vdash^{\delta} \lambda y.(e\{v/x\}) : \tau \xrightarrow{\delta'} \tau'$ by T-Abstraction with (5)
(7) $(\lambda y.e)\{v/x\} = \lambda y.(e\{v/x\})$ by Definition A.1
$\star$ $\Delta; \Gamma \vdash^{\delta} (\lambda y.e)\{v/x\} : \tau \xrightarrow{\delta'} \tau'$ from (6, 7)

5. T-Application

(H1) $\Delta; \Gamma, x : \tau'' \vdash^{\delta} e\ e' : \tau'$
(H2) $\Delta; \Gamma \vdash^{\delta} v : \tau''$
(3) $\Delta; \Gamma, x : \tau'' \vdash^{\delta} e : \tau \xrightarrow{\delta'} \tau'$ by inv. T-Application in (H1)
(4) $\Delta; \Gamma, x : \tau'' \vdash^{\delta} e' : \tau$ by inv. T-Application in (H1)
(5) $\delta' \sqsubseteq \delta$ by inv. T-Application in (H1)
(6) $\Delta; \Gamma \vdash^{\delta} e\{v/x\} : \tau \xrightarrow{\delta'} \tau'$ by I.H. with (3, H2)
(7) $\Delta; \Gamma \vdash^{\delta} e'\{v/x\} : \tau$ by I.H. with (4, H2)
(8) $\Delta; \Gamma, x : \tau'' \vdash^{\delta} (e\{v/x\})\ (e'\{v/x\}) : \tau'$ by T-Application with (6, 7, 5)
(9) $(e\{v/x\})\ (e'\{v/x\}) = (e\ e')\{v/x\}$ by Definition A.1
$\star$ $\Delta; \Gamma, x : \tau'' \vdash^{\delta} (e\ e')\{v/x\} : \tau'$ from (8, 9)

6. T-BinaryOp

(H1) $\Delta; \Gamma, x : \tau''' \vdash^{\delta} e\ \mathsf{op}\ e' : \tau''$
(H2) $\Delta; \Gamma \vdash^{\delta} v : \tau'''$
(3) $\Delta; \Gamma, x : \tau''' \vdash^{\delta} e : \tau$ by inv. T-Application in (H1)
(4) $\Delta; \Gamma, x : \tau''' \vdash^{\delta} e' : \tau'$ by inv. T-Application in (H1)
(5) op: $\tau \to \tau' \to \tau''$ by inv. T-Application in (H1)
(6) $\Delta; \Gamma \vdash^{\delta} e\{v/x\} : \tau$ by I.H. with (3, H2)
(7) $\Delta; \Gamma \vdash^{\delta} e'\{v/x\} : \tau$ by I.H. with (4, H2)
(8) $\Delta; \Gamma \vdash^{\delta} e\{v/x\}\ \mathsf{op}\ e'\{v/x\} : \tau''$ by T-BinaryOp with (6, 7, 5)
(9) $e\{v/x\}\ \mathsf{op}\ e'\{v/x\} = (e\ \mathsf{op}\ e')\{v/x\}$ Definition A.1
$\star$ $\Delta; \Gamma \vdash^{\delta} (e\ \mathsf{op}\ e')\{v/x\} : \tau''$ from (8, 9)

7. T-If

(H1) $\Delta; \Gamma, x : \tau' \vdash^{\delta} e\ ?\ e' : e'' : \tau$
(H2) $\Delta; \Gamma \vdash^{\delta} v : \tau'$
(3) $\Delta; \Gamma, x : \tau' \vdash^{\delta} e : \mathsf{Bool}$ by inv. T-If in (H1)
(4) $\Delta; \Gamma, x : \tau' \vdash^{\delta} e' : \tau$ by inv. T-If in (H1)
(5) $\Delta; \Gamma, x : \tau' \vdash^{\delta} e'' : \tau$ by inv. T-If in (H1)
(6) $\Delta; \Gamma \vdash^{\delta} e\{v/x\} : \mathsf{Bool}$ by I.H. with (3, H2)
(7) $\Delta; \Gamma \vdash^{\delta} e'\{v/x\} : \tau$ by I.H. with (4, H2)
(8) $\Delta; \Gamma \vdash^{\delta} e''\{v/x\} : \tau$ by I.H. with (5, H2)
(9) $\Delta; \Gamma \vdash^{\delta} (e\{v/x\})\ ?\ (e'\{v/x\}) : e''\{v/x\} : \tau$ by T-If with (6, 7, 8)
(10) $(e\{v/x\})\ ?\ (e'\{v/x\}) : e''\{v/x\} = (e\ ?\ e' : e'')\{v/x\}$ by Definition A.1
$\star$ $\Delta; \Gamma \vdash^{\delta} (e\ ?\ e' : e'')\{v/x\} : \tau$ from (9, 10)

8. T-ForEach

(H1) $\Delta; \Gamma, x : \tau'' \vdash^{\delta} \mathbf{iter}(e, e', y.z.e'') : \tau'$
(H2) $\Delta; \Gamma \vdash^{\delta} v : \tau''$
(3) $\Delta; \Gamma, x : \tau'' \vdash^{\delta} e : \tau^*$ by inv. T-ForEach in (H1)
(4) $\Delta; \Gamma, x : \tau'' \vdash^{\delta} e' : \tau'$ by inv. T-ForEach in (H1)
(5) $\Delta; \Gamma, x : \tau'', y : \tau, z : \tau' \vdash^{\delta} e'' : \tau'$ by inv. T-ForEach in (H1)
(6) $\Delta; \Gamma \vdash^{\delta} e\{v/x\} : \tau^*$ by I.H. with (3, H2)
(7) $\Delta; \Gamma \vdash^{\delta} e'\{v/x\} : \tau'$ by I.H. with (4, H2)
(8) $\Delta; \Gamma, y : \tau, z : \tau' \vdash^{\delta} e''\{v/x\} : \tau'$ by I.H. with (5, H2)
(9) $\Delta; \Gamma \vdash^{\delta} \mathbf{iter}(e\{v/x\}, e'\{v/x\}, y.z.e''\{v/x\}) : \tau'$ by T-ForEach with (6, 7, 8)

(10) $\mathbf{iter}(e\{v/x\}, e'\{v/x\}, y.z.e''\{v/x\}) = (\mathbf{iter}(e, e', y.z.e''))\{v/x\}$ by Definition A.1
⋆ $\Delta; \Gamma \overset{\delta}{\vdash} (\mathbf{iter}(e, e', y.z.e''))\{v/x\} : \tau'$ from (9, 10)

9. T-MATCH
(H1) $\Delta; \Gamma, x : \tau'' \overset{\delta}{\vdash} \mathbf{match}\ e\ \mathbf{with}\ y{::}ys \to e' \mid [] \to e'' : \tau'$
(H2) $\Delta; \Gamma \overset{\delta}{\vdash} v : \tau''$
(3) $\Delta; \Gamma, x : \tau'' \overset{\delta}{\vdash} e : \tau^*$ by inv. T-MATCH in (H1)
(4) $\Delta; \Gamma, x : \tau'', y : \tau, ys : \tau^* \overset{\delta}{\vdash} e' : \tau'$ by inv. T-MATCH in (H1)
(5) $\Delta; \Gamma, x : \tau'' \overset{\delta}{\vdash} e' : \tau'$ by inv. T-MATCH in (H1)
(6) $\Delta; \Gamma \overset{\delta}{\vdash} e\{v/x\} : \tau^*$ by I.H. with (3, H2)
(7) $\Delta; \Gamma \overset{\delta}{\vdash} e'\{v/x\} : \tau'$ by I.H. with (4, H2)
(8) $\Delta; \Gamma \overset{\delta}{\vdash} e''\{v/x\} : \tau'$ by I.H. with (5, H2)
(9) $\Delta; \Gamma \overset{\delta}{\vdash} \mathbf{match}\ e\{v/x\}\ \mathbf{with}\ y{::}ys \to e'\{v/x\} \mid [] \to e''\{v/x\} : \tau'$ by T-MATCH with (6, 7, 8)
(10) $\mathbf{match}\ e\{v/x\}\ \mathbf{with}\ y{::}ys \to e'\{v/x\} \mid [] \to e''\{v/x\}$
$= (\mathbf{match}\ e\ \mathbf{with}\ y{::}ys \to e' \mid [] \to e'')\{v/x\}$ by Definition A.1
⋆ $\Delta; \Gamma \overset{\delta}{\vdash} (\mathbf{match}\ e\ \mathbf{with}\ y{::}ys \to e' \mid [] \to e'')\{v/x\} : \tau'$ from (9, 10)

10. T-ACTION
(H1) $\Delta, a : \mathsf{var}_\delta(\tau); \Gamma, x : \tau'' \overset{\delta'}{\vdash} \mathbf{action}\ \{a \leftarrow e\} : \mathsf{Action}(\delta[a : \mathsf{var}(\tau)])$
(H2) $\Delta; \Gamma \overset{\delta'}{\vdash} v : \tau''$
(3) $\Delta, a : \mathsf{var}_\delta(\tau); \Gamma, x : \tau'' \overset{\delta[a:\mathsf{var}(\tau)}{\vdash} e : \tau$ by inv. T-ACTION
(4) $\Delta; \Gamma \overset{\delta[a:\mathsf{var}(\tau)}{\vdash} v : \tau''$ from T-ID, T-BASEVALUE, T-ABSTRACTION, T-ACTION and (H2)
(5) $\Delta, a : \mathsf{var}_\delta(\tau); \Gamma \overset{\delta[a:\mathsf{var}(\tau)}{\vdash} e\{v/x\} : \tau$ by I.H. with (3, 4)
(6) $\Delta, a : \mathsf{var}_\delta(\tau); \Gamma \overset{\delta'}{\vdash} \mathbf{action}\ \{a \leftarrow e\{v/x\}\} : \mathsf{Action}(\delta[a : \mathsf{var}(\tau)])$ by T-ACTION with (5)
(7) $\mathbf{action}\ \{a \leftarrow e\{v/x\}\} = (\mathbf{action}\ \{a \leftarrow e\})\{v/x\}$ by Definition A.1
⋆ $\Delta, a : \mathsf{var}_\delta(\tau); \Gamma \overset{\delta'}{\vdash} (\mathbf{action}\ \{a \leftarrow e\})\{v/x\} : \mathsf{Action}(\delta[a : \mathsf{var}(\tau)])$ from (6, 7)
□

Lemma B.5 (Expression Context)**.** For all expression configurations e and typing environments of names Δ, and typing environments of identifiers Γ, if $\Delta; \Gamma \overset{\delta}{\vdash} \mathcal{E}[e] : \tau$ and $\Delta; \Gamma \overset{\delta}{\vdash} e : \tau'$ and $\Delta; \Gamma \overset{\delta}{\vdash} e' : \tau'$ then $\Delta; \Gamma \overset{\delta}{\vdash} \mathcal{E}[e'] : \tau$.

Proof. Trivially, by induction on the derivation $\Delta; \Gamma \overset{\delta}{\vdash} \mathcal{E}[e] : \tau$.
□

Lemma B.6 (Expression Progress)**.** For all well-typed expression configurations $\Delta \vdash (\mathcal{S}; e)$, and typing environments of identifiers Γ, if $\Delta; \Gamma \overset{\delta}{\vdash} e : \tau$ and $\forall a \in \mathsf{dom}(\delta).\mathcal{S}_\mathsf{v}(a) = v$ then either e is a value, or there is an expression e' such that $\mathcal{S}; e \to e'$.

Proof. By induction on the derivation $\Delta; \Gamma, x : \tau \overset{\delta}{\vdash} e : \tau$.

1. T-NAME
(H1) $\Delta \vdash (\mathcal{S}; a)$
(H2) $\Delta; \Gamma \overset{\delta}{\vdash} a : \tau$
(H3) $\forall a \in \mathsf{dom}(\delta).\mathcal{S}_\mathsf{v}(a) = v$
(4) $\lfloor \Delta(a) \rfloor \prec \delta(a)$ by inv. T-NAME in (H2)
(5) $a \in \mathsf{dom}(\delta)$ from (4)
(6) $\mathcal{S}_\mathsf{v}(a) = v$ from (H3, 5)
⋆ $\mathcal{S}; a \to v$ by E-NAME with (6)

2. T-ID
(H1) $\Delta \vdash (\mathcal{S}; x)$
(H2) $\Delta; \Gamma, x : \tau \overset{\delta}{\vdash} x : \tau$
(H3) $\forall a \in \mathsf{dom}(\delta).\mathcal{S}_\mathsf{v}(a) = v$
⋆ x is a value

3. T-BASEVALUE
(H1) $\Delta \vdash (\mathcal{S}; \mathbf{b})$
(H2) $\Delta; \Gamma \overset{\delta}{\vdash} \mathbf{b} : \beta(\mathbf{b})$
(H3) $\forall a \in \mathsf{dom}(\delta).\mathcal{S}_\mathsf{v}(a) = v$
⋆ $\mathbf{b}$ is a value

4. T-ABSTRACTION

(H1) $\Delta \vdash (\mathcal{S}; \lambda x.e)$

(H2) $\Delta; \Gamma \overset{\delta'}{\vdash} \lambda x.e : \tau \overset{\delta}{\to} \tau'$

(H3) $\forall a \in \mathsf{dom}(\delta').\mathcal{S}_{\mathsf{v}}(a) = v$

$\star$ $\lambda x.e$ is a value

5. T-APPLICATION

(H1) $\Delta \vdash (\mathcal{S}; e\ e')$

(H2) $\Delta; \Gamma \overset{\delta}{\vdash} e\ e' : \tau'$

(H3) $\forall a \in \mathsf{dom}(\delta).\mathcal{S}_{\mathsf{v}}(a) = v$

(4) $\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.2 in (H1)

(5) $\Delta \vdash \delta$ by Definition 4.2 in (H1)

(6) $\Delta; \Gamma \overset{\delta}{\vdash} e : \tau \overset{\delta'}{\to} \tau'$ by inv. T-APPLICATION in (H2)

(7) $\Delta; \Gamma \overset{\delta}{\vdash} e' : \tau$ by inv. T-APPLICATION in (H2)

$\delta' \sqsubseteq \delta$ by inv. T-APPLICATION in (H2)

(8) $\Delta \vdash (\mathcal{S}; e)$ by Definition 4.2 with (4, 6, 5)

$\mathcal{S}; e \to e'' \vee e$ is a value by I.H. with (8, 6, H3)

(9) CASE: $\mathcal{S}; e \to e''$

$\star$ $\mathcal{S}; e\ e' \to e''\ e'$ by E-CTXT with (9)

(11) CASE: e is a value

(12) $\Delta \vdash (\mathcal{S}; e')$ by Definition 4.2 with (4, 7, 5)

$\mathcal{S}; e' \to e'' \vee e'$ is a value by I.H. with (12, 7, H3)

(13) SCASE: $\mathcal{S}; e' \to e''$

$\star$ $\mathcal{S}; e\ e' \to e\ e''$ by E-CTXT with (13)

(15) SCASE: e' is a value

$e = \lambda x.e''$ from (6, 11)

$\star$ $\mathcal{S}; (\lambda x.e'')\ e' \to e''\{e'/x\}$ by E-APPLICATION

6. T-BINARYOP

(H1) $\Delta \vdash (\mathcal{S}; e\ \mathsf{op}\ e')$

(H2) $\Delta; \Gamma \overset{\delta}{\vdash} e\ \mathsf{op}\ e' : \tau''$

(H3) $\forall a \in \mathsf{dom}(\delta).\mathcal{S}_{\mathsf{v}}(a) = v$

(4) $\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.2 in (H1)

(5) $\Delta \vdash \delta$ by Definition 4.2 in (H1)

(6) $\Delta; \Gamma \overset{\delta}{\vdash} e : \tau$ by inv. T-BINARYOP in (H2)

(7) $\Delta; \Gamma \overset{\delta}{\vdash} e' : \tau'$ by inv. T-BINARYOP in (H2)

(8) $\mathsf{op}: \tau \to \tau' \to \tau''$ by inv. T-BINARYOP in (H2)

(9) $\Delta \vdash (\mathcal{S}; e)$ by Definition 4.2 with (4, 6, 5)

$\mathcal{S}; e \to e'' \vee e$ is a value by I.H. with (9, 6, H3)

(10) CASE: $\mathcal{S}; e \to e''$

$\star$ $\mathcal{S}; e\ e' \to e''\ e'$ by E-CTXT with (10)

(12) CASE: e is a value

(13) $\Delta \vdash (\mathcal{S}; e')$ by Definition 4.2 with (4, 7, 5)

$\mathcal{S}; e' \to e'' \vee e'$ is a value by I.H. with (13, 7, H3)

(14) SCASE: $\mathcal{S}; e' \to e''$

$\star$ $\mathcal{S}; e\ e' \to e\ e''$ by E-CTXT with (14)

(16) SCASE: e' is a value

$v = e\ \mathsf{op}\ e'$ from (12, 16)

$\star$ $\mathcal{S}; e\ \mathsf{op}\ e' \to v$ by E-BOP

7. T-IF

(H1) $\Delta \vdash (\mathcal{S}; e\ ?\ e' : e'')$

(H2) $\Delta; \Gamma \overset{\delta}{\vdash} (e\ ?\ e' : e'') : \tau$

(H3) $\forall a \in \mathsf{dom}(\delta).\mathcal{S}_{\mathsf{v}}(a) = v$

(4) $\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.2 in (H1)

(5) $\Delta \vdash \delta$ by Definition 4.2 in (H1)

(6) $\Delta; \Gamma \overset{\delta}{\vdash} e : \mathsf{Bool}$ by T-IF in (H2)

$\Delta; \Gamma \overset{\delta}{\vdash} e' : \tau$ by T-IF in (H2)

$\Delta; \Gamma \overset{\delta}{\vdash} e'' : \tau$ by T-IF in (H2)

(7) $\Delta \vdash (\mathcal{S}; e)$ by Definition 4.2 with (4, 6, 5)
$\mathcal{S}; e \to e_2 \vee e$ is a value by I.H. with (7, 6, H3)

(8) CASE: $\mathcal{S}; e \to e_2$
⋆ $\mathcal{S}; e\ ?\ e' : e'' \to e_2\ ?\ e' : e''$ by E-CTXT with (8)

(10) CASE: e is a value
$e = \mathbf{true} \vee e = \mathbf{false}$ from (6, 10)

SCASE: $e = \mathbf{true}$
⋆ $\mathcal{S}; \mathbf{true}\ ?\ e' : e'' \to e'$ by E-TRUE

SCASE: $e = \mathbf{false}$
⋆ $\mathcal{S}; \mathbf{false}\ ?\ e' : e'' \to e''$ by E-FALSE

8. T-FOREACH
(H1) $\Delta \vdash (\mathcal{S}; \mathbf{iter}(e, e', x.y.e''))$
(H2) $\Delta; \Gamma \overset{\delta}{\vdash} \mathbf{iter}(e, e', x.y.e'') : \tau'$
(H3) $\forall a \in \mathsf{dom}(\delta). \mathcal{S}_\mathsf{v}(a) = v$
(4) $\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.2 in (H1)
(5) $\Delta \vdash \delta$ by Definition 4.2 in (H1)
(6) $\Delta; \Gamma \overset{\delta}{\vdash} e : \tau^*$ by inv. T-FOREACH in (H2)
(7) $\Delta; \Gamma \overset{\delta}{\vdash} e' : \tau'$ by inv. T-FOREACH in (H2)
$\Delta; \Gamma, x : \tau, y : \tau' \overset{\delta}{\vdash} e'' : \tau'$ by inv. T-FOREACH in (H2)
(8) $\Delta \vdash (\mathcal{S}; e)$ by Definition 4.2 with (4, 6, 5)
$\mathcal{S}; e \to e_2 \vee e$ is a value by I.H. with (8, 6, H3)

(9) CASE: $\mathcal{S}; e \to e_2$
⋆ $\mathcal{S}; \mathbf{iter}(e, e', x.y.e'') \to \mathbf{iter}(e_2, e', x.y.e'')$ by E-CTXT with (9)

(11) CASE: e is a value
$e = v{::}vs \vee e = []$ from (11, 6)

(12) SCASE: $e = v{::}vs$
(13) $\Delta \vdash (\mathcal{S}; e')$ by Definition 4.2 with (4, 7, H3)
$\mathcal{S}; e' \to e'_2 \vee e'$ is a value by I.H. with (13, 6, H3)

(14) SSCASE: $\mathcal{S}; e' \to e'_2$
⋆ $\mathcal{S}; \mathbf{iter}(e, e', x.y.e'') \to \mathbf{iter}(e, e'_2, x.y.e'')$ by E-CTXT with (9)

(16) SSCASE: e' is a value
$v' = e'$ from (16)
⋆ $\mathcal{S}; \mathbf{iter}(v{::}vs, v', x.y.e'') \to \mathbf{iter}(vs, e''\{v/x\}\{v'/y\}, x.y.e'')$
by E-FOREACH and (11, 12, 16)

(18) SCASE: $e = []$
(19) $\Delta \vdash (\mathcal{S}; e')$ by Definition 4.2 with (4, 7, H3)
$\mathcal{S}; e' \to e'_2 \vee e'$ is a value by I.H. with (19, 6, H3)

(20) SSCASE: $\mathcal{S}; e' \to e'_2$
⋆ $\mathcal{S}; \mathbf{iter}([], e', x.y.e'') \to \mathbf{iter}([], e'_2, x.y.e'')$ by E-CTXT with (18, 20)

(22) SSCASE: e' is a value
$v = e'$ from (22)
⋆ $\mathcal{S}; \mathbf{iter}([], v, x.y.e'') \to v$ by E-FOREACHNIL

9. T-MATCH
(H1) $\Delta \vdash (\mathcal{S}; \mathbf{match}\ e\ \mathbf{with}\ x{::}xs \to e' \mid [] \to e'')$
(H2) $\Delta; \Gamma \overset{\delta}{\vdash} \mathbf{match}\ e\ \mathbf{with}\ x{::}xs \to e' \mid [] \to e'' : \tau'$
(H3) $\forall a \in \mathsf{dom}(\delta). \mathcal{S}_\mathsf{v}(a) = v$
(4) $\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.2 in (H1)
(5) $\Delta \vdash \delta$ by Definition 4.2 in (H1)
(6) $\Delta; \Gamma \overset{\delta}{\vdash} e : \tau^*$ by inv. T-MATCH in (H2)
$\Delta; \Gamma, x : \tau, xs : \tau^* \overset{\delta}{\vdash} e' : \tau'$ by inv. T-MATCH in (H2)
$\Delta; \Gamma \overset{\delta}{\vdash} e'' : \tau'$ by inv. T-MATCH in (H2)
(7) $\Delta \vdash (\mathcal{S}; e)$ by Definition 4.2 with (4, 6, 5)
$\mathcal{S}; e \to e_2 \vee e$ is a value by I.H. with (7, 6, H3)

(8) CASE: $\mathcal{S}; e \to e_2$
⋆ $\mathcal{S}; (\mathbf{match}\ e\ \mathbf{with}\ x{::}xs \to e' \mid [] \to e'') \to$

$\to (\textbf{match}\ e_2\ \textbf{with}\ x::xs \to e' \mid [] \to e'')$ by E-CTXT with (8)

(10) CASE: e is a value
$e = v::vs \vee e = []$ from (10, 6)

(11) SCASE: $e = v::vs$
$\star$ $\mathcal{S}; \textbf{match}\ v::vs\ \textbf{with}\ x::xs \to e' \mid [] \to e'' \to e'$ by E-MATCH

(13) SCASE: $e = []$
$\star$ $\mathcal{S}; \textbf{match}\ []\ \textbf{with}\ x::xs \to e' \mid [] \to e'' \to e'$ by E-MATCHNIL

10. T-ACTION
(H1) $\Delta \vdash (\mathcal{S}; \textbf{action}\ \{a \leftarrow e\})$
(H2) $\Delta; \Gamma \vdash^{\delta'} \textbf{action}\ \{a \leftarrow e\} : \mathsf{Action}(\delta)$
(H3) $\forall a \in \mathsf{dom}(\delta').\mathcal{S}_\mathsf{v}(a) = v$
$\star$ $\textbf{action}\ \{a \leftarrow e\}$ is a value
□

Lemma B.7 (Expression Type Preservation)**.** For all expression configurations $(\mathcal{S}; e)$ and typing environments of names Δ, and typing environments of identifiers Γ, if $\Delta \vdash (\mathcal{S}; e)$ and $\Delta; \Gamma \vdash^{\delta} e : \tau$ and $\mathcal{S}; e \to e'$, then $\Delta; \Gamma \vdash^{\delta} e' : \tau$.

Proof. By induction on the length of the reduction $\mathcal{S}; e \to e'$.

1. E-CTXT
(H1) $\Delta \vdash (\mathcal{S}; \mathcal{E}[e])$
(H2) $\Delta; \Gamma \vdash^{\delta} \mathcal{E}[e] : \tau$
(H3) $\mathcal{S}; \mathcal{E}[e] \to \mathcal{E}[e']$
(4) $\mathcal{S}; e \to e'$ by inv. E-CTXT in (H2)
(5) $\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.2 in (H1)
(6) $\Delta \vdash \delta$ by Definition 4.2 in (H1)
(7) $\Delta; \Gamma \vdash^{\delta} e : \tau'$ from (H2)
(8) $\Delta \vdash (\mathcal{S}; e)$ by Definition 4.2 with (5, 7, 6)
(9) $\Delta; \Gamma \vdash^{\delta} e' : \tau'$ by I.H. with (8, 7, 4)
$\star$ $\Delta; \Gamma \vdash^{\delta} \mathcal{E}[e'] : \tau$ by Lemma B.5 with (H2, 7, 9)

2. E-NAME
(H1) $\Delta \vdash (\mathcal{S}; a)$
(H2) $\Delta; \Gamma \vdash^{\delta} a : \tau$
(H3) $\mathcal{S}; a \to v$
(4) $\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.2 in (H1)
$\Delta \vdash \delta$ by Definition 4.2 in (H1)
(5) $\tau = \lfloor \Delta(a) \rfloor$ by inv. T-NAME in (H2)
$\tau \prec \delta(a)$ by inv. T-NAME in (H2)
(6) $\mathcal{S}_\mathsf{v}(a) = v$ by inv. E-NAME in (H3)
(7) $\mathsf{dom}(\Delta) = \mathsf{dom}(\mathcal{S})$ by Corollary B.1 with (4)
$a \in \mathsf{dom}(\Delta)$ from (6, 7)
$\Delta(a) = \mathsf{var}_{\delta'}(\tau) \vee \Delta(a) = \mathsf{def}_{\delta'}(\tau)$ from (5)
$\mathcal{S} = \mathcal{S}', a \mapsto (e, v)$ from (6)
$\Delta \vdash a \mapsto (e, \nu) * \mathcal{S}'$ by Definition 4.1 (WTSVARVALUE, WTSDEFVALUE) in (4)

(8) CASE: $\Delta(a) = \mathsf{var}_\delta(\tau)$
$\Delta; \varepsilon \vdash^{\delta'[a:\mathsf{var}(\tau)]} e : \tau$ by Definition 4.1 (WTSVARVALUE) in (4)
(9) $\Delta; \varepsilon \vdash^{\delta'[a:\mathsf{var}(\tau)]} v : \tau$ by Definition 4.1 (WTSVARVALUE) in (4)
$\star$ $\Delta; \Gamma \vdash^{\delta} v : \tau$ from T-ID, T-BASEVALUE, T-ABSTRACTION, T-ACTION and (9)

(11) CASE: $\Delta(a) = \mathsf{def}_\delta(\tau)$
$\Delta; \varepsilon \vdash^{\delta} e : \tau$ by Definition 4.1 (WTSDEFVALUE) in (4)
(12) $\Delta; \varepsilon \vdash^{\delta]} v : \tau$ by Definition 4.1 (WTSDEFVALUE) in (4)
$\star$ $\Delta; \Gamma \vdash^{\delta} v : \tau$ from T-ID, T-BASEVALUE, T-ABSTRACTION, T-ACTION and (12)

3. E-TRUE
(H1) $\Delta \vdash (\mathcal{S}; \textbf{true}\ ?\ e : e')$
(H2) $\Delta; \Gamma \vdash^{\delta} (\textbf{true}\ ?\ e : e') : \tau$
(H3) $\mathcal{S}; \textbf{true}\ ?\ e : e' \to e$

⋆ $\Delta; \Gamma \vdash^{\delta} e : \tau$ by inv. T-If in (H2)

4. E-False

(H1) $\Delta \vdash (\mathcal{S}; \mathbf{false}\ ?\ e : e')$
(H2) $\Delta; \Gamma \vdash^{\delta} (\mathbf{false}\ ?\ e : e') : \tau$
(H3) $\mathcal{S}; \mathbf{false}\ ?\ e : e' \to e'$
⋆ $\Delta; \Gamma \vdash^{\delta} e' : \tau$ by inv. T-If in (H2)

5. E-Application

(H1) $\Delta \vdash (\mathcal{S}; \lambda x.e\ v)$
(H2) $\Delta; \Gamma \vdash^{\delta} \lambda x.e\ v : \tau'$
(H3) $\mathcal{S}; \lambda x.e\ v \to e\{v/x\}$
$\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.2 in (H1)
(4) $\Delta \vdash \delta$ by Definition 4.2 in (H1)
(5) $\Delta; \Gamma \vdash^{\delta} \lambda x.e : \tau \xrightarrow{\delta'} \tau'$ by inv. T-Application in (H2)
(6) $\Delta; \Gamma \vdash^{\delta} v : \tau$ by inv. T-Application in (H2)
(7) $\delta' \sqsubseteq \delta$ by inv. T-Application in (H2)
(8) $\Delta; \Gamma, x : \tau \vdash^{\delta'} e : \tau'$ by inv. T-Abstraction in (5)
(9) $\Delta; \Gamma, x : \tau \vdash^{\delta} e : \tau'$ by Lemma B.3 with (8, 4, 7)
⋆ $\Delta; \Gamma \vdash^{\delta} e\{v/x\} : \tau'$ by Lemma B.4 with (9, 6)

6. E-BOp

(H1) $\Delta \vdash (\mathcal{S}; v\ \mathsf{op}\ v')$
(H2) $\Delta; \Gamma \vdash^{\delta} v\ \mathsf{op}\ v' : \tau''$
(H3) $\mathcal{S}; v\ \mathsf{op}\ v' \to v''$
⋆ $\Delta; \Gamma \vdash^{\delta} v'' : \tau''$ by T-BinaryOp

7. E-Match

(H1) $\Delta \vdash (\mathcal{S}; \mathbf{match}\ v{::}vs\ \mathbf{with}\ x{::}xs \to e \mid [] \to e')$
(H2) $\Delta; \Gamma \vdash^{\delta} \mathbf{match}\ v{::}vs\ \mathbf{with}\ x{::}xs \to e \mid [] \to e' : \tau'$
(H3) $\mathcal{S}; (\mathbf{match}\ v{::}vs\ \mathbf{with}\ x{::}xs \to e \mid [] \to e') \to e\{v/x\}\{vs/xs\}$
(4) $\Delta; \Gamma \vdash^{\delta} v{::}vs : \tau^*$ by inv. T-Match in (H2)
(5) $\Delta; \Gamma, x : \tau, xs : \tau^* \vdash^{\delta} e : \tau'$ by inv. T-Match in (H2)
$\Delta; \Gamma \vdash^{\delta} e' : \tau'$ by inv. T-Match in (H2)
(6) $\Delta; \Gamma \vdash^{\delta} v : \tau$ by T-BinaryOp in (4)
(7) $\Delta; \Gamma \vdash^{\delta} vs : \tau^*$ by T-BinaryOp in (4)
(8) $\Delta; \Gamma, xs : \tau^* \vdash^{\delta} e\{v/x\} : \tau'$ by Lemma B.4 with (5, 6)
⋆ $\Delta; \Gamma \vdash^{\delta} e\{v/x\}\{vs/xs\} : \tau'$ by Lemma B.4 with (8, 7)

8. E-MatchNil

(H1) $\Delta \vdash (\mathcal{S}; \mathbf{match}\ []\ \mathbf{with}\ x{::}xs \to e \mid [] \to e')$
(H2) $\Delta; \Gamma \vdash^{\delta} \mathbf{match}\ []\ \mathbf{with}\ x{::}xs \to e \mid [] \to e' : \tau'$
(H3) $\mathcal{S}; (\mathbf{match}\ []\ \mathbf{with}\ x{::}xs \to e \mid [] \to e') \to e'$
$\Delta; \Gamma \vdash^{\delta} [] : \tau^*$ by inv. T-Match in (H2)
$\Delta; \Gamma, x : \tau, xs : \tau^* \vdash^{\delta} e : \tau'$ by inv. T-Match in (H2)
⋆ $\Delta; \Gamma \vdash^{\delta} e' : \tau'$ by inv. T-Match in (H2)

9. E-ForEach

(H1) $\Delta \vdash (\mathcal{S}; \mathbf{iter}(v{::}vs, v', x.y.e))$
(H2) $\Delta; \Gamma \vdash^{\delta} \mathbf{iter}(v{::}vs, v', x.y.e) : \tau'$
(H3) $\mathcal{S}; \mathbf{iter}(v{::}vs, v', x.y.e) \to \mathbf{iter}(vs, e\{v/x\}\{v'/y\}, x.y.e)$
(4) $\Delta; \Gamma \vdash^{\delta} v{::}vs : \tau^*$ by inv. T-ForEach in (H2)
(5) $\Delta; \Gamma \vdash^{\delta} v' : \tau'$ by inv. T-ForEach in (H2)
(6) $\Delta; \Gamma, x : \tau, y : \tau' \vdash^{\delta} e : \tau'$ by inv. T-ForEach in (H2)
(7) $\Delta; \Gamma \vdash^{\delta} v : \tau$ by inv. T-BinaryOp in (4)
(8) $\Delta; \Gamma \vdash^{\delta} vs : \tau^*$ by inv. T-BinaryOp in (4)
(9) $\Delta; \Gamma, y : \tau' \vdash^{\delta} e\{x/v\} : \tau'$ by Lemma B.4 with (6, 7)
(10) $\Delta; \Gamma \vdash^{\delta} e\{v/x\}\{v'/y\} : \tau'$ by Lemma B.4 with (9, 5)
⋆ $\Delta; \Gamma \vdash^{\delta} \mathbf{iter}(vs, e\{v/x\}\{v'/y\}, x.y.e)$ by T-ForEach with (8, 10, 6)

10. E-ForEachNil

(H1) $\Delta \vdash (\mathcal{S}; \mathbf{iter}([], v, x.y.e))$

(H2) $\Delta; \Gamma \vdash^{\delta} \mathbf{iter}([], v, x.y.e) : \tau'$

(H3) $\mathcal{S}; \mathbf{iter}([], v, x.y.e) \to v$

⋆ $\Delta; \Gamma \vdash^{\delta} v : \tau'$ by inv. T-ForEach in (H2)

□

B.2 Operations

Lemma B.8 (Reconfiguration Progress)**.** For all runtime configurations $(\Delta; \mathcal{S}; ([q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}))$, such that $\vdash (\Delta; \mathcal{S}; ([q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}))$ if $q \neq \varepsilon$ and $\forall a \in \mathsf{dom}(\mathcal{S}').\mathcal{S}'_{\mathsf{v}}(a) = \square \Rightarrow (q = a; q_1 \vee q = a\langle e\rangle; q_1)$ then, there is a program configuration $(\Delta; \mathcal{S}; ([q']^{\Delta''}_{\mathcal{S}''}; \mathcal{Q}))$ such that $\Delta; \mathcal{S}; ([q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}) \to \Delta; \mathcal{S}; ([q']^{\Delta''}_{\mathcal{S}''}; \mathcal{Q})$ and $\forall a \in \mathsf{dom}(\mathcal{S}'').\mathcal{S}''_{\mathsf{v}}(a) = \square \Rightarrow (q' = a; q_2 \vee q' = a\langle e\rangle; q_2)$

Proof. By induction on the derivation $\vdash \Delta; \mathcal{S}; ([q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}')$.

1. $q = (\mathbf{var}\ a{=}e; q_1)$

(H1) $\vdash \Delta; \mathcal{S}; ([\mathbf{var}\ a{=}e; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}')$

(H2) $\forall b \in \mathsf{dom}(\mathcal{S}').\mathcal{S}'_{\mathsf{v}}(b) = \square \Rightarrow (q = b; q_1 \vee q = b\langle e\rangle; q_1)$

$\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.4 in (H1)

(3) $\Delta; \mathcal{S} \vdash [\mathbf{var}\ a{=}e; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ by Definition 4.4 in (H1)

$\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)

(4) $\Delta \uplus \Delta' \vdash \mathbf{var}\ a{=}e : \Delta''$ by Definition 4.3 (WFQRVar) in (3)

$\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'; e)$ by Definition 4.3 (WFQRVar) in (3)

$\Delta; \mathcal{S} \vdash [q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ by Definition 4.3 (WFQRVar) in (3)

⋆ $\Delta; \mathcal{S}; ([\mathbf{var}\ a{=}e; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') \to \Delta; \mathcal{S}; ([a; q_1]^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q}')$ by S-RVar with (4)

(6) $\forall b \in \mathsf{dom}(\mathcal{S}').\mathcal{S}'_{\mathsf{v}}(b) = v$ from (H1, H2)

(7) $\mathcal{S}'' = \mathcal{S}'[a \mapsto (e, \square)]$

⋆ $\forall b \in \mathsf{dom}((\mathcal{S}'')).(\mathcal{S}'')]_{\mathsf{v}}(b) = \square \Rightarrow (q'' = b; q_2 \vee q'' = b\langle e\rangle; q_2)$ from (6, 7)

2. $q = (\mathbf{def}\ a{=}e; q_1)$

(H1) $\vdash \Delta; \mathcal{S}; ([\mathbf{def}\ a{=}e; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}')$

(H2) $\forall b \in \mathsf{dom}(\mathcal{S}').\mathcal{S}'_{\mathsf{v}}(b) = \square \Rightarrow (q = b; q_1 \vee q = b\langle e\rangle; q_1)$

$\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.4 in (H1)

(3) $\Delta; \mathcal{S} \vdash [\mathbf{def}\ a{=}e; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ by Definition 4.4 in (H1)

$\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)

(4) $\Delta \uplus \Delta' \vdash \mathbf{def}\ a{=}e : \Delta''$ by Definition 4.3 (WFQRDef) in (3)

$\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'; e)$ by Definition 4.3 (WFQRDef) in (3)

$\Delta; \mathcal{S} \vdash [q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ by Definition 4.3 (WFQRDef) in (3)

(5) $\forall b \in \mathsf{dom}(\mathcal{S}').\mathcal{S}'_{\mathsf{v}}(b) = v$ from (H1, H2)

(6) $\mathcal{S}'' = \mathcal{S}'[a \mapsto (e, \square)]$

⋆ $\Delta; \mathcal{S}; ([\mathbf{var}\ a{=}e; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') \to \Delta; \mathcal{S}; ([a; q_1]^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q}')$ by S-RDef with (4)

⋆ $\forall b \in \mathsf{dom}((\mathcal{S}'')).(\mathcal{S}'')]_{\mathsf{v}}(b) = \square \Rightarrow (q'' = b; q_2 \vee q'' = b\langle e\rangle; q_2)$ from (5, 6)

3. $q = a; q_1$

(H1) $\vdash \Delta; \mathcal{S}; ([a; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}')$

(H2) $\forall b \in \mathsf{dom}(\mathcal{S}').\mathcal{S}'_{\mathsf{v}}(b) = \square \Rightarrow (q = b; q_1 \vee q = b\langle e\rangle; q_1)$

$\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.4 in (H1)

(3) $\Delta; \mathcal{S} \vdash [a; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ by Definition 4.4 in (H1)

$\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)

$(\Delta \uplus \Delta')(a) = \mathsf{var}_{\delta}(\tau) \vee (\Delta \uplus \Delta')(a) = \mathsf{def}_{\delta}(\tau)$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (3)

(4) $(\mathcal{S} \uplus \mathcal{S}')(a) = (e, \nu)$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (3)

$\Delta \uplus \Delta'; \varepsilon \vdash^{\delta} e : \tau$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (3)

$\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'; e)$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (3)

$\Delta; \mathcal{S} \vdash [q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (3)

⋆ $\Delta; \mathcal{S}; ([a; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') \to \Delta; \mathcal{S}; ([a\langle e\rangle; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}')$ by S-RQueue with (4)

⋆ $\forall b \in \mathsf{dom}(\mathcal{S}').\mathcal{S}'_{\mathsf{v}}(b) = \square \Rightarrow (q'' = b; q_2 \vee q'' = b\langle e\rangle; q_2)$ from (H2)

4. $q = a\langle e\rangle; q_1$

(H1) $\vdash \Delta; \mathcal{S}; ([a\langle e\rangle; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}')$

(H2) $\forall b \in \mathsf{dom}(\mathcal{S}').\mathcal{S}'_{\mathsf{v}}(b) = \square \Rightarrow (q = b; q_1 \vee q = b\langle e\rangle; q_1)$

$\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.4 in (H1)

(3) $\Delta; \mathcal{S} \vdash [a\langle e\rangle; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ by Definition 4.4 in (H1)

$\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)

(4) $(\Delta \uplus \Delta')(a) = \mathsf{var}_\delta(\tau) \vee (\Delta \uplus \Delta')(a) = \mathsf{def}_\delta(\tau)$ by Definition 4.3 (WFQRVarEval, WFQRVarEval) in (3)

(5) $\Delta \uplus \Delta'; \varepsilon \overset{\delta}{\vdash} e : \tau$ by Definition 4.3 (WFQRVarEval, WFQRVarEval) in (3)

(6) $\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'; e)$ by Definition 4.3 (WFQRVarEval, WFQRVarEval) in (3)

$\Delta; \mathcal{S} \vdash [q_1]^{\Delta'}_{\mathcal{S}'}; q'$ by Definition 4.3 (WFQRVarEval, WFQRVarEval) in (3)

(7) $\Delta \uplus \Delta' \vdash \varepsilon * (\mathcal{S} \uplus \mathcal{S}')$ by Definition 4.2 in (6)

(8) $\Delta \uplus \Delta' \vdash \delta$ by Definition 4.2 in (6)

(9) $\mathsf{dom}(\delta) \subseteq \mathsf{dom}(\Delta \uplus \Delta')$ by Corollary B.2 with (8)

(10) $\mathsf{dom}(\Delta \uplus \Delta') = \mathsf{dom}(\mathcal{S} \uplus \mathcal{S}')$ by Corollary B.1 with (7)

(11) $\mathsf{dom}(\delta) \subseteq \mathsf{dom}(\mathcal{S} \uplus \mathcal{S}')$ from (9, 10)

(12) $a \notin \mathsf{dom}(\delta)$ by T-Var, T-Def and (4)

(13) $\forall a \in \mathsf{dom}(\delta).(\mathcal{S} \uplus \mathcal{S}')_{\mathsf{v}}(a) = v$ from (H2, 11, 12)

$\mathcal{S} \uplus \mathcal{S}'; e \to e' \vee e$ is a value by Lemma B.6 with (6, 5, 13)

(14) Case: $\mathcal{S} \uplus \mathcal{S}'; e \to e'$

- $\Delta; \mathcal{S}; ([a\langle e\rangle; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') \to \Delta; \mathcal{S}; ([a\langle e'\rangle; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}')$ by S-RStep with (14)
- $\forall b \in \mathsf{dom}(\mathcal{S}').\mathcal{S}'_{\mathsf{v}}(b) = \square \Rightarrow (q'' = b; q_2 \vee q'' = b\langle e\rangle; q_2)$ from (H2)

(17) Case: $e = v$ is a value

(18) $(\mathcal{S} \uplus \mathcal{S}')(a) = (e'', \nu)$

(19) $\mathcal{S}'' = \mathcal{S}'[a \mapsto (e'', v)]$

(20) $\forall b \in \mathsf{dom}(\mathcal{S}'').\mathcal{S}''_{\mathsf{v}}(b) = v$ from (H2, 19)

- $\Delta; \mathcal{S}; ([a\langle e\rangle; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') \to \Delta; \mathcal{S}; ([q_1]^{\Delta'}_{\mathcal{S}'[a\mapsto(e'',v)]}; \mathcal{Q}')$ by S-RValue with (18)
- $\forall b \in \mathsf{dom}(\mathcal{S}').\mathcal{S}'_{\mathsf{v}}(b) = \square \Rightarrow (q'' = b; q_2 \vee q'' = b\langle e\rangle; q_2)$ from (20)

□

Lemma B.9 (Empty Reconfiguration Values)**.** For all runtime configurations $(\Delta; \mathcal{S}; ([\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'))$, if $\vdash (\Delta; \mathcal{S}; ([\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'))$ then $\forall a \in \mathsf{dom}(\mathcal{S}').\mathcal{S}'_{\mathsf{v}}(a) = v$

Proof. Follows directly from Lemma B.8.

□

Lemma B.10 (Reconfiguration Name Eval)**.** For all runtime configurations $(\Delta; \mathcal{S}; ([a\langle e\rangle; \mathcal{Q}]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'))$, if $\vdash (\Delta; \mathcal{S}; ([a\langle e\rangle; \mathcal{Q}]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'))$ then, $\forall b \in \mathsf{dom}(\mathcal{S}') \wedge b \neq a.\mathcal{S}'_{\mathsf{v}}(b) = v$.

Proof. Follows directly from Lemma B.8.

□

Lemma B.11 (State Join Left-Value Update)**.** For all states $\mathcal{S}$ and $\mathcal{S}'$ such that $\Delta \vdash \varepsilon * (\mathcal{S}[a \mapsto (e, v)] \uplus \mathcal{S}')$, if $\Delta; \varepsilon \overset{\delta}{\vdash} e : \tau$ and $\Delta; \varepsilon \overset{\delta}{\vdash} v' : \tau$ then, $\Delta \vdash \varepsilon * (\mathcal{S}[a \mapsto (e, v')] \uplus \mathcal{S}')$.

Proof. By case analysis of the $\mathsf{dom}(\mathcal{S}')$**1.** $a \in \mathsf{dom}(\mathcal{S}')$

(H1) $\Delta \vdash \varepsilon * (\mathcal{S}[a \mapsto (e, v)] \uplus \mathcal{S}')$

(H2) $\Delta; \varepsilon \overset{\delta}{\vdash} e : \tau$

(H3) $\Delta; \varepsilon \overset{\delta}{\vdash} v' : \tau$

(4) $(\mathcal{S}[a \mapsto (e, v)] \uplus \mathcal{S}')(a) = (e'', v'')$

(5) $\mathcal{S}'(a) = (e'', v'')$

$\mathcal{S}[a \mapsto (e, v')] \uplus \mathcal{S}' = \mathcal{S}[a \mapsto (e, v)] \uplus \mathcal{S}'$ from (4, 5)

- $\Delta \vdash \varepsilon * (\mathcal{S}[a \mapsto (e, v')] \uplus \mathcal{S}')$

2. $a \notin \mathsf{dom}(\mathcal{S}')$

(H1) $\Delta \vdash \varepsilon * (\mathcal{S}[a \mapsto (e, v)] \uplus \mathcal{S}')$

(H2) $\Delta; \varepsilon \overset{\delta}{\vdash} e : \tau$

(H3) $\Delta; \varepsilon \overset{\delta}{\vdash} v' : \tau$

$(\mathcal{S}[a \mapsto (e, v)] \uplus \mathcal{S}')(a) = (e, v)$

(4) $\mathcal{S}[a \mapsto (e, v)] = \mathcal{S}'', a \mapsto (e, v)$

(5) $\Delta \vdash a \mapsto (e, \nu) * (\mathcal{S}'' \uplus \mathcal{S}')$ by Definition 4.1 (WTSVarValue, WTSDefValue) in (H1) and (4)

(6) $\Delta(a) = \mathsf{var}_\delta(\tau) \vee \Delta(a) = \mathsf{def}_\delta(\tau)$ by Definition 4.1 (WTSVarValue, WTSDefValue) in (H1) and (4)

(7) Case: $\Delta(a) = \mathsf{var}_\delta(\tau)$

$\Delta; \varepsilon \overset{\delta[a:\mathsf{var}(\tau)]}{\vdash} v : \tau$ by Definition 4.1 (WTSVarValue) in (H1) and (4)

(8) $\Delta; \varepsilon \vdash^{\delta[a:\mathsf{var}(\tau)]} e : \tau$ — by Lemma B.3 with (H2)

(9) $\Delta; \varepsilon \vdash^{\delta[a:\mathsf{var}(\tau)]} v : \tau$ — by Lemma B.3 with (H3)

(10) $\Delta \vdash \varepsilon * (\mathcal{S}'' \uplus \mathcal{S}'), a \mapsto (e, v')$ — by Definition 4.1 (WTSVarValue) with (6, 8, 9, 5)

(11) $(\mathcal{S}'' \uplus \mathcal{S}'), a \mapsto (e, v') = \mathcal{S}[a \mapsto (e, v')] \uplus \mathcal{S}'$

⋆ $\Delta \vdash \varepsilon * (\mathcal{S}[a \mapsto (e, v')] \uplus \mathcal{S}')$ — from (10, 11)

(13) Case: $\Delta(a) = \mathsf{var}_\delta(\tau)$

$\Delta; \varepsilon \vdash^{\delta} v : \tau$ — by Definition 4.1 (WTSDefValue) in (H1) and (4)

(14) $\Delta \vdash \varepsilon * (\mathcal{S}'' \uplus \mathcal{S}'), a \mapsto (e, v')$ — by Definition 4.1 (WTSDefValue) with (6, H2, H3, 5)

(15) $(\mathcal{S}'' \uplus \mathcal{S}'), a \mapsto (e, v') = \mathcal{S}[a \mapsto (e, v')] \uplus \mathcal{S}'$

⋆ $\Delta \vdash \varepsilon * (\mathcal{S}[a \mapsto (e, v')] \uplus \mathcal{S}')$ — from (14, 15)

□

Lemma B.12 (State Value Update)**.** For all runtime configurations $(\Delta; \mathcal{S}[a \mapsto (e, v)]; \mathcal{Q})$ such that $\vdash \Delta; \mathcal{S}[a \mapsto (e, v)]; \mathcal{Q}$, if $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e, v')]$ and $\Delta; \varepsilon \vdash e : \tau^\delta$ and $\Delta; \varepsilon \vdash v' : \tau^\delta$ then, $\vdash \Delta; \mathcal{S}[a \mapsto (e, v')]; \mathcal{Q}$.

Proof. By induction on the derivation $\vdash \Delta; \mathcal{S}; \mathcal{Q}$.

1. $\mathcal{Q} = \varepsilon$

(H1) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v)]; \varepsilon$

(H2) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e', v')]$

(H3) $\Delta; \varepsilon \vdash^{\delta} e' : \tau$

(H4) $\Delta; \varepsilon \vdash^{\delta} v' : \tau$

(5) $\Delta; \mathcal{S}[a \mapsto (e', v')] \vdash \varepsilon$ — by Definition 4.3 (WFQEmpty)

(6) $\varepsilon \vDash \Delta$ — by Definition 4.4 in (H1)

⋆ $\vdash \Delta; \mathcal{S}[a \mapsto (e', v')]; \varepsilon$ — by Definition 4.4 with (H2, 5, 6)

2. $\mathcal{Q} = \mathbf{do}\ e; \mathcal{Q}'$

(H1) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v)]; (\mathbf{do}\ e; \mathcal{Q}')$

(H2) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e', v')]$

(H3) $\Delta; \varepsilon \vdash^{\delta} e' : \tau$

(H4) $\Delta; \varepsilon \vdash^{\delta} v' : \tau$

(5) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e', v)]$ — by Definition 4.4 in (H1)

(6) $\Delta; \mathcal{S}[a \mapsto (e', v)] \vdash \mathbf{do}\ e; \mathcal{Q}'$ — by Definition 4.4 in (H1)

(7) $\varepsilon \vDash \Delta$ — by Definition 4.4 in (H1)

(8) $\Delta \vdash \mathbf{do}\ e : \varepsilon$ — by Definition 4.3 (WFQDo) in (6)

(9) $\Delta \vdash (\mathcal{S}[a \mapsto (e', v)]; e)$ — by Definition 4.3 (WFQDo) in (6)

(10) $\Delta; \mathcal{S}[a \mapsto (e', v)] \vdash \mathcal{Q}'$ — by Definition 4.3 (WFQDo) in (6)

(11) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v)]; \mathcal{Q}'$ — by Definition 4.4 with (5, 10, 7)

(12) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v')]; \mathcal{Q}'$ — by I.H. with (11, H2, H3, H4)

(13) $\Delta; \mathcal{S}[a \mapsto (e', v')] \vdash \mathcal{Q}'$ — by Definition 4.4 in (12)

(14) $\Delta; \varepsilon \vdash^{\delta'} e : \tau'$ — by Definition 4.2 in (9)

(15) $\Delta \vdash \delta'$ — by Definition 4.2 in (9)

(16) $\Delta \vdash (\mathcal{S}[a \mapsto (e', v')]; e)$ — by Definition 4.2 with (H2, 14, 15)

(17) $\Delta; \mathcal{S}[a \mapsto (e', v')] \vdash \mathbf{do}\ e; \mathcal{Q}'$ — by Definition 4.3 (WFQDo) with (8, 16, 13)

⋆ $\vdash \Delta; \mathcal{S}[a \mapsto (e', v')]; \mathbf{do}\ e; \mathcal{Q}'$ — by Definition 4.4 with (H2, 17, 7)

3. $\mathcal{Q} = b; \mathcal{Q}'$

(H1) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v)]; (b; \mathcal{Q}')$

(H2) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e', v')]$

(H3) $\Delta; \varepsilon \vdash^{\delta} e' : \tau$

(H4) $\Delta; \varepsilon \vdash^{\delta} v' : \tau$

(5) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e', v')]$ — by Definition 4.4 in (H1)

(6) $\Delta; \mathcal{S}[a \mapsto (e', v)] \vdash b; \mathcal{Q}'$ — by Definition 4.4 in (H1)

(7) $\varepsilon \vDash \Delta$ — by Definition 4.4 in (H1)

(8) $\Delta(b) = \sigma$ — by Definition 4.3 (WFQName) in (6)

(9) $\mathcal{S}[a \mapsto (e', v)](b) = (e'', v'')$ — by Definition 4.3 (WFQName) in (6)

(10) $\Delta \vdash (\mathcal{S}[a \mapsto (e', v)]; e'')$ — by Definition 4.3 (WFQName) in (6)

(11) $\Delta \vdash (\mathcal{S}[a \mapsto (e', v)]; v'')$ — by Definition 4.3 (WFQName) in (6)

(12) $\Delta; \mathcal{S}[a \mapsto (e', v)] \vdash \mathcal{Q}'$ — by Definition 4.3 (WFQName) in (6)

(13) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v)]; \mathcal{Q}'$ — by Definition 4.4 with (5, 12, 7)

(14) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v')]; \mathcal{Q}'$ — by I.H. with (13, H2, H3, H4)

(15) $\Delta; \mathcal{S}[a \mapsto (e', v')] \vdash \mathcal{Q}'$ — by Definition 4.4 in (14)

(16) $\mathcal{S}[a \mapsto (e', v')](b) = (e'', v'')$ — from (9)

(17) $\Delta; \varepsilon \overset{\delta'}{\vdash} e'' : \tau'$ by Definition 4.2 in (10)
(18) $\Delta \vdash \delta'$ by Definition 4.2 in (10)
(19) $\Delta \vdash (\mathcal{S}[a \mapsto (e', v')]; e'')$ by Definition 4.2 with (5, 17, 18)
(20) $\Delta; \varepsilon \overset{\delta''}{\vdash} v'' : \tau''$ by Definition 4.2 in (11)
(21) $\Delta \vdash \delta''$ by Definition 4.2 in (11)
(22) $\Delta \vdash (\mathcal{S}[a \mapsto (e', v')]; v'')$ by Definition 4.2 with (5, 20, 21)
(23) $\Delta; \mathcal{S}[a \mapsto (e', v')] \vdash b; \mathcal{Q}'$ by Definition 4.3 (WFQNAME) with (8, 16, 19, 22, 15)
⋆ $\vdash \Delta; \mathcal{S}[a \mapsto (e', v')]; (b; \mathcal{Q}')$ by Definition 4.4 with (H2, 23, 7)

4. $\mathcal{Q} = b\langle e \rangle; \mathcal{Q}'$
(H1) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v)]; (b\langle e \rangle; \mathcal{Q}')$
(H2) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e', v')]$
(H3) $\Delta; \varepsilon \overset{\delta}{\vdash} e' : \tau$
(H4) $\Delta; \varepsilon \overset{\delta}{\vdash} v' : \tau$
(5) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e', v)]$ by Definition 4.4 in (H1)
(6) $\Delta; \mathcal{S}[a \mapsto (e', v)] \vdash b\langle e \rangle; \mathcal{Q}'$ by Definition 4.4 in (H1)
(7) $\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)
(8) $\Delta(b) = \mathsf{var}_{\delta'}(\tau) \vee \Delta(b) = \mathsf{def}_{\delta'}(\tau')$
(9) $\Delta; \varepsilon \overset{\delta'[a:\mathsf{var}(\tau)]}{\vdash} e : \tau' \vee \Delta; \varepsilon \overset{\delta'}{\vdash} e : \tau'$ by Definition 4.3 (WFQVAREVAL, WFQDEFEVAL) in (6)
(10) $\Delta \vdash (\mathcal{S}[a \mapsto (e', v)]; e)$ by Definition 4.3 (WFQVAREVAL, WFQDEFEVAL) in (6)
(11) $\Delta; \mathcal{S}[a \mapsto (e', v)] \vdash \mathcal{Q}'$ by Definition 4.3 (WFQVAREVAL, WFQDEFEVAL) in (6)
(12) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v)]; \mathcal{Q}'$ by Definition 4.4 with (5, 11, 7)
(13) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v')]; \mathcal{Q}'$ by I.H. with (12, H2, H3, H4)
(14) $\Delta; \mathcal{S}[a \mapsto (e', v')] \vdash \mathcal{Q}'$ by Definition 4.4 in (13)
(15) $\Delta \vdash \delta'$ by Definition 4.2 in (10)
(16) $\Delta \vdash (\mathcal{S}[a \mapsto (e', v')]; e)$ by Definition 4.2 with (H2, 10, 15)
(17) $\Delta; \mathcal{S}[a \mapsto (e', v')] \vdash b\langle e \rangle; \mathcal{Q}'$ by Definition 4.3 (WFQVAREVAL, WFQDEFEVAL) with (8, 9, 16, 14)
⋆ $\vdash \Delta; \mathcal{S}[a \mapsto (e', v')]; (b\langle e \rangle; \mathcal{Q}')$ by Definition 4.4 with (H2, 17, 7)

5. $\mathcal{Q} = [\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$
(H1) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v)]; ([\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}')$
(H2) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e', v')]$
(H3) $\Delta; \varepsilon \overset{\delta}{\vdash} e' : \tau$
(H4) $\Delta; \varepsilon \overset{\delta}{\vdash} v' : \tau$
$\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e', v)]$ by Definition 4.4 in (H1)
(5) $\Delta; \mathcal{S}[a \mapsto (e', v)] \vdash [\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ by Definition 4.4 in (H1)
(6) $\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)
(7) $\Delta \uplus \Delta' \vdash \varepsilon * (\mathcal{S}[a \mapsto (e', v)] \uplus \mathcal{S}')$ by Definition 4.3 (WFQREMPTY) in (5)
(8) $\Delta \vDash \Delta'$ by Definition 4.3 (WFQREMPTY) in (5)
(9) $\Delta \uplus \Delta'; \mathcal{S}[a \mapsto (e', v)] \uplus \mathcal{S}' \vdash \mathcal{Q}'$ by Definition 4.3 (WFQREMPTY) in (5)
(10) $\Delta \uplus \Delta' \vdash \varepsilon * (\mathcal{S}[a \mapsto (e', v')] \uplus \mathcal{S}')$ by Lemma B.11 with (7, H3, H4)

(11) CASE: $a \in \mathsf{dom}(\mathcal{S}')$
(12) $\mathcal{S}[a \mapsto (e', v)] \uplus \mathcal{S}' = \mathcal{S}[a \mapsto (e', v')] \uplus \mathcal{S}'$ by Definition 3.7 and (11)
(13) $\Delta \uplus \Delta'; \mathcal{S}[a \mapsto (e', v')] \uplus \mathcal{S}' \vdash \mathcal{Q}'$ from (12)
⋆ $\Delta; \mathcal{S}[a \mapsto (e', v')] \vdash [\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ by Definition 4.3 (WFQREMPTY) with (10, 8, 13)

(15) CASE: $a \notin \mathsf{dom}(\mathcal{S}')$
$\mathcal{S}[a \mapsto (e', v)] \uplus \mathcal{S}' = (\mathcal{S} \uplus \mathcal{S}')[a \mapsto (e', v)]$ by Definition 3.7 and (15)
(16) $\varepsilon \vDash \Delta \uplus \Delta'$ by Definition 3.11 with (6, 8)
(17) $\vdash \Delta \uplus \Delta'; (\mathcal{S} \uplus \mathcal{S}')[a \mapsto (e', v)]; \mathcal{Q}'$ by Definition 4.4 with (7, 9, 16)
(18) $\vdash \Delta \uplus \Delta'; (\mathcal{S} \uplus \mathcal{S}')[a \mapsto (e', v')]; \mathcal{Q}'$ by I.H. with (17, 10, H3, H4)
(19) $\mathcal{S}[a \mapsto (e', v')] \uplus \mathcal{S}' = (\mathcal{S} \uplus \mathcal{S}')[a \mapsto (e', v')]$ by Definition 3.7 and (15)
(20) $\vdash \Delta \uplus \Delta'; \mathcal{S}[a \mapsto (e', v')] \uplus \mathcal{S}'; \mathcal{Q}'$ from (18, 19)
(21) $\Delta \uplus \Delta'; \mathcal{S}[a \mapsto (e', v')] \uplus \mathcal{S}' \vdash \mathcal{Q}'$ by Definition 4.4 in (20)
(22) $\Delta; \mathcal{S}[a \mapsto (e', v')] \vdash [\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ by Definition 4.3 (WFQREMPTY) with (10, 8, 21)
⋆ $\vdash \Delta; \mathcal{S}[a \mapsto (e', v')]; [\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ by Definition 4.4 with (10, 22, 6)

6. $\mathcal{Q} = [\mathbf{var}\ b{=}e; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}''$
(H1) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v)]; ([\mathbf{var}\ b{=}e; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$
(H2) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e', v')]$
(H3) $\Delta; \varepsilon \overset{\delta}{\vdash} e' : \tau$
(H4) $\Delta; \varepsilon \overset{\delta}{\vdash} v' : \tau$

(5) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e', v)]$ by Definition 4.4 in (H1)
(6) $\Delta; \mathcal{S}[a \mapsto (e', v)] \vdash [\mathbf{var}\ b{=}e; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}''$ by Definition 4.4 in (H1)
(7) $\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)
(8) $\Delta \uplus \Delta' \vdash \mathbf{var}\ b{=}e : \Delta''$ by Definition 4.3 (WFQRVar) in (6)
(9) $\Delta \uplus \Delta' \vdash (\mathcal{S}[a \mapsto (e', v)] \uplus \mathcal{S}'; e)$ by Definition 4.3 (WFQRVar) in (6)
(10) $\Delta; \mathcal{S}[a \mapsto (e', v)] \vdash [q']^{\Delta' \uplus \Delta''}_{\mathcal{S}'[b \mapsto (e, \square)]}; \mathcal{Q}''$ by Definition 4.3 (WFQRVar) in (6)
(11) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v)]; [q']^{\Delta' \uplus \Delta''}_{\mathcal{S}'[b \mapsto (e, \square)]}; \mathcal{Q}''$ by Definition 4.4 with (5, 10, 7)
(12) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v')]; ([q']^{\Delta' \uplus \Delta''}_{\mathcal{S}'[b \mapsto (e, \square)]}; \mathcal{Q}'')$ by I.H. with (11, H2, H3, H4)
(13) $\Delta; \mathcal{S}[a \mapsto (e', v')] \vdash [q']^{\Delta' \uplus \Delta''}_{\mathcal{S}'[b \mapsto (e, \square)]}; \mathcal{Q}''$ by Definition 4.4 in (12)
(14) $\Delta \uplus \Delta'; \varepsilon \overset{\delta'}{\vdash} e : \tau'$ by Definition 4.2 in (9)
(15) $\Delta \uplus \Delta' \vdash \delta'$ by Definition 4.2 in (9)
(16) $\Delta \uplus \Delta' \vdash \varepsilon * (\mathcal{S}[a \mapsto (e', v)] \uplus \mathcal{S}')$ by Definition 4.2 in (9)
(17) $\Delta \uplus \Delta' \vdash \varepsilon * (\mathcal{S}[a \mapsto (e', v')] \uplus \mathcal{S}')$ by Lemma B.11 with (16, H3, H4)
(18) $\Delta \uplus \Delta' \vdash (\mathcal{S}[a \mapsto (e', v')] \uplus \mathcal{S}'; e)$ by Definition 4.2 with (17, 14, 15)
(19) $\Delta; \mathcal{S}[a \mapsto (e', v')] \vdash [\mathbf{var}\ a{=}e; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}''$ by Definition 4.3 (WFQRVar) with (8, 18, 13)
$\star$ $\vdash \Delta; \mathcal{S}[a \mapsto (e', v')]; ([\mathbf{var}\ a{=}e; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$ by Definition 4.4 with (18, 19, 7)

7. $\mathcal{Q} = [\mathbf{def}\ b{=}e; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}''$

(H1) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v)]; ([\mathbf{def}\ b{=}e; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$
(H2) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e', v')]$
(H3) $\Delta; \varepsilon \overset{\delta}{\vdash} e' : \tau$
(H4) $\Delta; \varepsilon \overset{\delta}{\vdash} v' : \tau$
(5) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e', v)]$ by Definition 4.4 in (H1)
(6) $\Delta; \mathcal{S}[a \mapsto (e', v)] \vdash [\mathbf{def}\ b{=}e; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}''$ by Definition 4.4 in (H1)
(7) $\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)
(8) $\Delta \uplus \Delta' \vdash \mathbf{def}\ b{=}e : \Delta''$ by Definition 4.3 (WFQRDef) in (6)
(9) $\Delta \uplus \Delta' \vdash (\mathcal{S}[a \mapsto (e', v)] \uplus \mathcal{S}'; e)$ by Definition 4.3 (WFQRDef) in (6)
(10) $\Delta; \mathcal{S}[a \mapsto (e', v)] \vdash [q']^{\Delta' \uplus \Delta''}_{\mathcal{S}'[b \mapsto (e, \square)]}; \mathcal{Q}''$ by Definition 4.3 (WFQRDef) in (6)
(11) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v)]; [q']^{\Delta' \uplus \Delta''}_{\mathcal{S}'[b \mapsto (e, \square)]}; \mathcal{Q}''$ by Definition 4.4 with (5, 10, 7)
(12) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v')]; ([q']^{\Delta' \uplus \Delta''}_{\mathcal{S}'[b \mapsto (e, \square)]}; \mathcal{Q}'')$ by I.H. with (11, H2, H3, H4)
(13) $\Delta; \mathcal{S}[a \mapsto (e', v')] \vdash [q']^{\Delta' \uplus \Delta''}_{\mathcal{S}'[b \mapsto (e, \square)]}; \mathcal{Q}''$ by Definition 4.4 in (12)
(14) $\Delta \uplus \Delta'; \varepsilon \overset{\delta'}{\vdash} e : \tau'$ by Definition 4.2 in (9)
(15) $\Delta \uplus \Delta' \vdash \delta'$ by Definition 4.2 in (9)
(16) $\Delta \uplus \Delta' \vdash \varepsilon * (\mathcal{S}[a \mapsto (e', v)] \uplus \mathcal{S}')$ by Definition 4.2 in (9)
(17) $\Delta \uplus \Delta' \vdash \varepsilon * (\mathcal{S}[a \mapsto (e', v')] \uplus \mathcal{S}')$ by Lemma B.11 with (16, H3, H4)
(18) $\Delta \uplus \Delta' \vdash (\mathcal{S}[a \mapsto (e', v')] \uplus \mathcal{S}'; e)$ by Definition 4.2 with (17, 14, 15)
(19) $\Delta; \mathcal{S}[a \mapsto (e', v')] \vdash [\mathbf{def}\ a{=}e; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}''$ by Definition 4.3 (WFQRVar) with (8, 18, 13)
$\star$ $\vdash \Delta; \mathcal{S}[a \mapsto (e', v')]; ([\mathbf{def}\ a{=}e; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$ by Definition 4.4 with (18, 19, 7)

8. $\mathcal{Q} = [b; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}''$

(H1) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v)]; ([b; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$
(H2) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e', v')]$
(H3) $\Delta; \varepsilon \overset{\delta}{\vdash} e' : \tau$
(H4) $\Delta; \varepsilon \overset{\delta}{\vdash} v' : \tau$
(5) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e', v)]$ by Definition 4.4 in (H1)
(6) $\Delta; \mathcal{S}[a \mapsto (e', v)] \vdash [b; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}''$ by Definition 4.4 in (H1)
(7) $\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)
(8) $(\Delta \uplus \Delta')(b) = \mathsf{var}_{\delta'}(\tau') \vee (\Delta \uplus \Delta')(b) = \mathsf{def}_{\delta'}(\tau')$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (6)
(9) $(\mathcal{S}[a \mapsto (e', v)] \uplus \mathcal{S}')(b) = (e'', \nu)$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (6)
(10) $\Delta \uplus \Delta'; \varepsilon \overset{\delta'}{\vdash} e'' : \tau'$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (6)
(11) $\Delta \uplus \Delta' \vdash (\mathcal{S}[a \mapsto (e', v)] \uplus \mathcal{S}'; e'')$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (6)
(12) $\Delta; \mathcal{S}[a \mapsto (e', v)] \vdash [q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}''$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (6)
(13) $\Delta \uplus \Delta' \vdash \varepsilon * (\mathcal{S}[a \mapsto (e', v)] \uplus \mathcal{S}')$ by Definition 4.2 in (11)
(14) $\Delta \uplus \Delta' \vdash \delta'$ by Definition 4.2 in (11)
(15) $\Delta \uplus \Delta' \vdash \varepsilon * (\mathcal{S}[a \mapsto (e', v')] \uplus \mathcal{S}')$ by Lemma B.11 with (13, H3, H4)
(16) $\Delta \uplus \Delta' \vdash (\mathcal{S}[a \mapsto (e', v')] \uplus \mathcal{S}'; e'')$ by Definition 4.2 with (15, 10, 14)

(17) $(\mathcal{S}[a \mapsto (e', v')] \uplus \mathcal{S}')(b) = (e'', \nu)$ — from (9)

(18) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v)]; ([q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$ — by Definition 4.4 with (5, 12, 7)

(19) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v')]; ([q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$ — by I.H. with (18, H2, H3, H4)

(20) $\Delta \uplus \Delta'; \mathcal{S}[a \mapsto (e', v')] \uplus \mathcal{S}' \vdash [q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}''$ — by Definition 4.4 in (19)

(21) $\Delta; \mathcal{S} \vdash [b; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}''$ — by Definition 4.3 (WFQRVarName, WFQRDefName) with (8, 17, 10, 16, 20)

$\star$ $\vdash \Delta; \mathcal{S}[a \mapsto (e', v')]; ([b; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$ — by Definition 4.4 with (15, 21, 7)

9. $\mathcal{Q} = [b\langle e\rangle; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}''$

(H1) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v)]; ([b\langle e\rangle; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$

(H2) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e', v')]$

(H3) $\Delta; \varepsilon \vdash^{\delta} e' : \tau$

(H4) $\Delta; \varepsilon \vdash^{\delta} v' : \tau$

(5) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e', v)]$ — by Definition 4.4 in (H1)

(6) $\Delta; \mathcal{S}[a \mapsto (e', v)] \vdash [b\langle e\rangle; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}''$ — by Definition 4.4 in (H1)

(7) $\varepsilon \vDash \Delta$ — by Definition 4.4 in (H1)

(8) $(\Delta \uplus \Delta')(b) = \mathsf{var}_{\delta'}(\tau) \vee (\Delta \uplus \Delta')(b) = \mathsf{def}_{\delta'}(\tau)$ — by Definition 4.3 (WFQRVarEval, WFQRVarEval) in (6)

(9) $\Delta \uplus \Delta'; \varepsilon \vdash^{\delta'} e : \tau$ — by Definition 4.3 (WFQRVarEval, WFQRVarEval) in (6)

(10) $\Delta \uplus \Delta' \vdash (\mathcal{S}[a \mapsto (e', v)] \uplus \mathcal{S}'; e)$ — by Definition 4.3 (WFQRVarEval, WFQRVarEval) in (6)

(11) $\Delta; \mathcal{S}[a \mapsto (e', v)] \vdash [q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}''$ — by Definition 4.3 (WFQRVarEval, WFQRVarEval) in (6)

(12) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v)]; [q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}''$ — by Definition 4.4 with (5, 11, 7)

(13) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v')]; [q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}''$ — by I.H. with (12, H2, H3, H4)

(14) $\Delta; \mathcal{S}[a \mapsto (e', v')] \vdash [q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}''$ — by Definition 4.4 in (13)

(15) $\Delta \uplus \Delta' \vdash \delta'$ — by Definition 4.2 in (10)

(16) $\Delta \uplus \Delta' \vdash (\mathcal{S}[a \mapsto (e', v')] \uplus \mathcal{S}'; e)$ — by Definition 4.2 with (H2, 9, 15)

(17) $\Delta; \mathcal{S}[a \mapsto (e', v')] \vdash [b\langle e\rangle; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}''$ — by Definition 4.3 (WFQRVarEval, WFQRVarEval) with (8, 9, 16, 14)

$\star$ $\vdash \Delta; \mathcal{S}[a \mapsto (e', v')]; ([b\langle e\rangle; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$ — by Definition 4.4 with (H2, 17, 7)

□

Lemma B.13 (Reconfiguration State Value Update)**.** For all runtime configurations, if $\vdash \Delta; \mathcal{S}; ([q]^{\Delta'}_{\mathcal{S}'[a\mapsto(e,\nu)]}; \mathcal{Q}')$ and $\Delta \uplus \Delta'; \varepsilon \vdash^{\delta} e : \tau$ and $\Delta \uplus \Delta'; \varepsilon \vdash^{\delta} v : \tau$ and $\Delta \uplus \Delta' \vdash \varepsilon * \mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, v)]$ then, $\vdash \Delta; \mathcal{S}; ([q]^{\Delta'}_{\mathcal{S}'[a\mapsto(e,v)]}; \mathcal{Q}')$.

Proof. By induction on the derivation $\vdash \Delta; \mathcal{S}; ([q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}')$.

1. $q = \varepsilon$

(H1) $\vdash \Delta; \mathcal{S}; ([\varepsilon]^{\Delta'}_{\mathcal{S}'[a\mapsto(e,\nu)]}; \mathcal{Q}')$

(H2) $\Delta \uplus \Delta'; \varepsilon \vdash^{\delta} e : \tau$

(H3) $\Delta \uplus \Delta'; \varepsilon \vdash^{\delta} v : \tau$

(H4) $\Delta \uplus \Delta' \vdash \varepsilon * \mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, v)]$

(5) $\Delta \vdash \varepsilon * \mathcal{S}$ — by Definition 4.4 in (H1)

(6) $\Delta; \mathcal{S} \vdash [\varepsilon]^{\Delta'}_{\mathcal{S}'[a\mapsto(e,\nu)]}; \mathcal{Q}'$ — by Definition 4.4 in (H1)

(7) $\varepsilon \vDash \Delta$ — by Definition 4.4 in (H1)

(8) $\Delta \uplus \Delta' \vdash \varepsilon * (\mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, \nu)])$ — by Definition 4.3 (WFQREmpty) in (6)

(9) $\Delta \vDash \Delta'$ — by Definition 4.3 (WFQREmpty) in (6)

(10) $\Delta \uplus \Delta'; \mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, \nu)] \vdash \mathcal{Q}'$ — by Definition 4.3 (WFQREmpty) in (6)

(11) $\varepsilon \vDash \Delta \uplus \Delta'$ — by Definition 3.11 with (7, 9)

(12) $\vdash \Delta \uplus \Delta'; \mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, \nu)]; \mathcal{Q}'$ — by Definition 4.4 with (8, 10, 11)

(13) $(\mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, \nu)])(a) = (e, v')$ — by Lemma B.9 in (12)

(14) $\vdash \Delta \uplus \Delta'; \mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, v)]; \mathcal{Q}'$ — by Lemma B.12 with (12, 13, H4, H2, H3)

(15) $\Delta \uplus \Delta'; \mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, v)] \vdash \mathcal{Q}'$ — by Definition 4.4 in (14)

(16) $\Delta; \mathcal{S} \vdash [\varepsilon]^{\Delta'}_{\mathcal{S}'[a\mapsto(e,v)]}; \mathcal{Q}'$ — by Definition 4.3 (WFQREmpty) with (H4, 9, 15)

$\star$ $\vdash \Delta; \mathcal{S}; [\varepsilon]^{\Delta'}_{\mathcal{S}'[a\mapsto(e,v)]}; \mathcal{Q}'$ — by Definition 4.4 with (5, 16, 7)

2. $q = (\mathbf{var}\ b{=}e'; \mathcal{Q}'')$

(H1) $\vdash \Delta; \mathcal{S}; ([\mathbf{var}\ b{=}e'; q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,\nu)]}; \mathcal{Q}')$

(H2) $\Delta \uplus \Delta'; \varepsilon \vdash^{\delta} e : \tau$

(H3) $\Delta \uplus \Delta'; \varepsilon \overset{\delta}{\vdash} v : \tau$
(H4) $\Delta \uplus \Delta' \vdash \varepsilon * \mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, v)]$
(5) $\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.4 in (H1)
(6) $\Delta; \mathcal{S} \vdash [\mathbf{var}\ b{=}e'; q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,\nu)]}; \mathcal{Q}'$ by Definition 4.4 in (H1)
(7) $\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)
(8) $\Delta \uplus \Delta' \vdash \mathbf{var}\ b{=}e' : \Delta''$ by Definition 4.3 (WFQRVar) in (6)
(9) $\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, \nu)]; e')$ by Definition 4.3 (WFQRVar) in (6)
(10) $\Delta; \mathcal{S} \vdash [q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,\nu),b\mapsto(e',\square)]}; \mathcal{Q}'$ by Definition 4.3 (WFQRVar) in (6)
(11) $\vdash \Delta; \mathcal{S}; ([q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,\nu),b\mapsto(e',\square)]}; \mathcal{Q}')$ by Definition 4.4 with (5, 10, 7)
(12) $\vdash \Delta; \mathcal{S}; ([q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,v),b\mapsto(e',\square)]}; \mathcal{Q}')$ by I.H. with (11, H2, H3, H4)
(13) $\Delta; \mathcal{S} \vdash [q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,v),b\mapsto(e',\square)]}; \mathcal{Q}'$ by Definition 4.4 in (12)
$\Delta \uplus \Delta' \vdash \varepsilon * (\mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, \nu)])$ by Definition 4.2 in (9)
(14) $\Delta \uplus \Delta'; \varepsilon \overset{\delta'}{\vdash} e' : \tau'$ by Definition 4.2 in (9)
(15) $\Delta \uplus \Delta' \vdash \delta'$ by Definition 4.2 in (9)
(16) $\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, v)]; e')$ by Definition 4.2 with (H4, 14, 15)
(17) $\Delta; \mathcal{S} \vdash [\mathbf{var}\ b{=}e'; q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,v)]}; \mathcal{Q}'$ by Definition 4.3 (WFQRVar) with (8, 16, 13)
$\star$ $\vdash \Delta; \mathcal{S}; [\mathbf{var}\ b{=}e'; q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,v)]}; \mathcal{Q}'$ by Definition 4.4 with (H4, 17, 7)

3. $q = (\mathbf{def}\ b{=}e'; q'')$

(H1) $\vdash \Delta; \mathcal{S}; ([\mathbf{def}\ b{=}e'; q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,\nu)]}; \mathcal{Q}')$
(H2) $\Delta \uplus \Delta'; \varepsilon \overset{\delta}{\vdash} e : \tau$
(H3) $\Delta \uplus \Delta'; \varepsilon \overset{\delta}{\vdash} v : \tau$
(H4) $\Delta \uplus \Delta' \vdash \varepsilon * \mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, v)]$
(5) $\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.4 in (H1)
(6) $\Delta; \mathcal{S} \vdash [\mathbf{def}\ b{=}e'; q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,\nu)]}; \mathcal{Q}'$ by Definition 4.4 in (H1)
(7) $\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)
(8) $\Delta \uplus \Delta' \vdash \mathbf{def}\ b{=}e' : \Delta''$ by Definition 4.3 (WFQRDef) in (6)
(9) $\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, \nu)]; e')$ by Definition 4.3 (WFQRDef) in (6)
(10) $\Delta; \mathcal{S} \vdash [q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,\nu),b\mapsto(e',\square)]}; \mathcal{Q}'$ by Definition 4.3 (WFQRDef) in (6)
(11) $\vdash \Delta; \mathcal{S}; ([q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,\nu),b\mapsto(e',\square)]}; \mathcal{Q}')$ by Definition 4.4 with (5, 10)
(12) $\vdash \Delta; \mathcal{S}; ([q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,v),b\mapsto(e',\square)]}; \mathcal{Q}')$ by I.H. with (11, H2, H3, H4)
(13) $\Delta; \mathcal{S} \vdash [q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,v),b\mapsto(e',\square)]}; \mathcal{Q}'$ by Definition 4.4 in (12)
$\Delta \uplus \Delta' \vdash \varepsilon * (\mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, \nu)])$ by Definition 4.2 in (9)
(14) $\Delta \uplus \Delta'; \varepsilon \overset{\delta'}{\vdash} e' : \tau'$ by Definition 4.2 in (9)
(15) $\Delta \uplus \Delta' \vdash \delta'$ by Definition 4.2 in (9)
(16) $\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, v)]; e')$ by Definition 4.2 with (H4, 14, 15)
(17) $\Delta; \mathcal{S} \vdash [\mathbf{def}\ b{=}e'; q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,v)]}; \mathcal{Q}'$ by Definition 4.3 (WFQRDef) with (8, 16, 13)
$\star$ $\vdash \Delta; \mathcal{S}; [\mathbf{def}\ b{=}e'; q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,v)]}; \mathcal{Q}'$ by Definition 4.4 with (H4, 17, 7)

4. $q = b; q''$

(H1) $\vdash \Delta; \mathcal{S}; ([b; q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,\nu)]}; \mathcal{Q}')$
(H2) $\Delta \uplus \Delta'; \varepsilon \overset{\delta}{\vdash} e : \tau$
(H3) $\Delta \uplus \Delta'; \varepsilon \overset{\delta}{\vdash} v : \tau$
(H4) $\Delta \uplus \Delta' \vdash \varepsilon * \mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, v)]$
(5) $\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.4 in (H1)
(6) $\Delta; \mathcal{S} \vdash [b; q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,\nu)]}; \mathcal{Q}'$ by Definition 4.4 in (H1)
(7) $\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)
(8) $(\Delta \uplus \Delta')(a) = \mathsf{var}_{\delta'}(\tau') \vee (\Delta \uplus \Delta')(a) = \mathsf{def}_{\delta'}(\tau')$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (6)
(9) $(\mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, \nu)])(b) = (e', \nu)$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (6)
(10) $\Delta \uplus \Delta'; \varepsilon \overset{\delta'}{\vdash} e' : \tau'$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (6)
(11) $\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, \nu)]; e')$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (6)
(12) $\Delta; \mathcal{S} \vdash [q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,\nu)]}; \mathcal{Q}'$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (6)
(13) $\vdash \Delta; \mathcal{S}; [q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,\nu)]}; \mathcal{Q}'$ by Definition 4.4 with (5, 12, 7)
(14) $\vdash \Delta; \mathcal{S}; [q'']^{\Delta'}_{\mathcal{S}'[a\mapsto(e,v)]}; \mathcal{Q}'$ by I.H. with (13, H2, H3, H4)

(15) $(\mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, v)])(b) = (e', \nu)$ from (9)
(16) $\Delta \uplus \Delta' \vdash \delta'$ by Definition 4.2 in (11)
(17) $\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, v)]; e')$ by Definition 4.2 with (H4, 10, 16)
(18) $\Delta; \mathcal{S} \vdash [q'']^{\Delta'}_{\mathcal{S}'[a \mapsto (e,v)]}; \mathcal{Q}'$ by Definition 4.4 in (14)
(19) $\Delta; \mathcal{S} \vdash [b; q'']^{\Delta'}_{\mathcal{S}'[a \mapsto (e,v)]}; \mathcal{Q}'$ by Definition 4.3 (WFQRVarName, WFQRDefName) with (8, 15, 10, 17, 18)
⋆ $\vdash \Delta; \mathcal{S}; ([b; q'']^{\Delta'}_{\mathcal{S}'[a \mapsto (e,v)]}; \mathcal{Q}')$ by Definition 4.4 with (H4, 19, 7)

5. $q = b\langle e' \rangle; q''$
(H1) $\vdash \Delta; \mathcal{S}; ([b\langle e' \rangle; q'']^{\Delta'}_{\mathcal{S}'[a \mapsto (e,\nu)]}; \mathcal{Q}')$
(H2) $\Delta \uplus \Delta'; \varepsilon \vdash^{\delta} e : \tau$
(H3) $\Delta \uplus \Delta'; \varepsilon \vdash^{\delta} v : \tau$
(H4) $\Delta \uplus \Delta' \vdash \varepsilon * \mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, v)]$
(5) $\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.4 in (H1)
(6) $\Delta; \mathcal{S} \vdash [b\langle e' \rangle; q'']^{\Delta'}_{\mathcal{S}'[a \mapsto (e,\nu)]}; \mathcal{Q}'$ by Definition 4.4 in (H1)
(7) $\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)
(8) $(\Delta \uplus \Delta')(a) = \mathsf{var}_{\delta'}(\tau') \vee (\Delta \uplus \Delta')(a) = \mathsf{def}_{\delta'}(\tau')$ by Definition 4.3 (WFQRVarEval, WFQRVarEval) in (6)
(9) $\Delta \uplus \Delta'; \varepsilon \vdash^{\delta'} e' : \tau'$ by Definition 4.3 (WFQRVarEval, WFQRVarEval) in (6)
(10) $\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, \nu)]; e')$ by Definition 4.3 (WFQRVarEval, WFQRVarEval) in (6)
(11) $\Delta; \mathcal{S} \vdash [q'']^{\Delta'}_{\mathcal{S}'[a \mapsto (e,\nu)]}; \mathcal{Q}'$ by Definition 4.3 (WFQRVarEval, WFQRVarEval) in (6)
(12) $\vdash \Delta; \mathcal{S}; [q'']^{\Delta'}_{\mathcal{S}'[a \mapsto (e,\nu)]}; \mathcal{Q}'$ by Definition 4.4 with (5, 11, 7)
(13) $\vdash \Delta; \mathcal{S}; [q'']^{\Delta'}_{\mathcal{S}'[a \mapsto (e,v)]}; \mathcal{Q}'$ by I.H. with (12, H2, H3, H4)
(14) $\Delta \uplus \Delta' \vdash \delta'$ by Definition 4.2 in (10)
(15) $\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'[a \mapsto (e, v)]; e')$ by Definition 4.2 with (H4, 9, 14)
(16) $\Delta; \mathcal{S} \vdash [q'']^{\Delta'}_{\mathcal{S}'[a \mapsto (e,v)]}; \mathcal{Q}'$ by Definition 4.4 in (13)
(17) $\Delta; \mathcal{S} \vdash [b\langle e' \rangle; q'']^{\Delta'}_{\mathcal{S}'[a \mapsto (e,v)]}; \mathcal{Q}'$ by Definition 4.3 (WFQRVarEval, WFQRVarEval) with (8, 9, 15, 16)
⋆ $\vdash \Delta; \mathcal{S}; ([b\langle e' \rangle; q'']^{\Delta'}_{\mathcal{S}'[a \mapsto (e,v)]}; \mathcal{Q}')$ by Definition 4.4 with (H4, 17, 7)
□

Theorem 4.5 (Runtime Progress)**.** For all runtime configurations $(\Delta; \mathcal{S}; \mathcal{Q})$, if $\vdash (\Delta; \mathcal{S}; \mathcal{Q})$ and $\mathcal{Q} \neq \varepsilon$ and $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_{\mathsf{v}}(a) = v$ then, there is a program configuration $(\Delta'; \mathcal{S}'; \mathcal{Q}')$ such that $\Delta; \mathcal{S}; \mathcal{Q} \rightarrow \Delta'; \mathcal{S}'; \mathcal{Q}'$ and $\forall a \in \mathsf{dom}(\mathcal{S}').\mathcal{S}'_{\mathsf{v}}(a) = v$.

Proof. By induction on the derivation $\vdash \Delta; \mathcal{S}; \mathcal{Q}$.

1. $\mathcal{Q} = (a; \mathcal{Q}_1)$
(H1) $\vdash \Delta; \mathcal{S}; (a; \mathcal{Q}_1)$
(H2) $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_{\mathsf{v}}(a) = v$
$\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.4 in (H1)
(3) $\Delta; \mathcal{S} \vdash (a; \mathcal{Q}_1)$ by Definition 4.4 in (H1)
$\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)
$\Delta(a) = \sigma$ by Definition 4.3 (WFQName) in (3)
(4) $\mathcal{S}(a) = (e, v)$ by Definition 4.3 (WFQName) in (3)
$\Delta \vdash (\mathcal{S}; e)$ by Definition 4.3 (WFQName) in (3)
$\Delta \vdash (\mathcal{S}; v)$ by Definition 4.3 (WFQName) in (3)
$\Delta; \mathcal{S} \vdash \mathcal{Q}$ by Definition 4.3 (WFQName) in (3)
⋆ $\Delta; \mathcal{S}; (a; \mathcal{Q}_1) \rightarrow \Delta; \mathcal{S}; (a\langle e \rangle; \mathcal{Q}_1)$ by S-Queue with (4)
⋆ $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_{\mathsf{v}}(a) = v$ from (H2)

2. $\mathcal{Q} = (a\langle e \rangle; \mathcal{Q}_1)$
(H1) $\vdash \Delta; \mathcal{S}; (a\langle e \rangle; \mathcal{Q}_1)$
(H2) $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_{\mathsf{v}}(a) = v$
(3) $\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.4 in (H1)
(4) $\Delta; \mathcal{S} \vdash (a\langle e \rangle; \mathcal{Q}_1)$ by Definition 4.4 in (H1)
$\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)
$\Delta(a) = \mathsf{var}_{\delta}(\tau) \vee \Delta(a) = \mathsf{def}_{\delta}(\tau)$ by Definition 4.3 (WFQVarEval, WFQDefEval) in (4)
(5) $\Delta; \varepsilon \vdash^{\delta[a:\mathsf{var}(\tau)]} e : \tau \vee \Delta; \varepsilon \vdash^{\delta} e : \tau$ by Definition 4.3 (WFQVarEval, WFQDefEval) in (4)
$\Delta; \mathcal{S} \vdash \mathcal{Q}_1$ by Definition 4.3 (WFQVarEval, WFQDefEval) in (4)
(6) $\Delta \vdash (\mathcal{S}; e)$ by Definition 4.3 (WFQVarEval, WFQDefEval) in (4)
(7) $\mathsf{dom}(\Delta) = \mathsf{dom}(\mathcal{S})$ by Corollary B.1 with (3)

(8) $\Delta \vdash \delta$ — by Definition 4.2 in (6)
(9) $\mathsf{dom}(\delta) \subseteq \mathsf{dom}(\Delta)$ — by Corollary B.2 with (8)
(10) $\mathsf{dom}(\delta) \subseteq \mathsf{dom}(\mathcal{S})$ — from (9, 7)
(11) $\forall a \in \mathsf{dom}(\delta).\mathcal{S}_{\mathsf{v}}(a) = v$ — from (H2, 10)
$\mathcal{S}; e \to e' \vee e$ is a value — by Lemma B.6 with (6, 5, 11)

(12) CASE: $\mathcal{S}; e \to e'$
⋆ $\Delta; \mathcal{S}; (a\langle e\rangle; \mathcal{Q}_1) \to \Delta; \mathcal{S}; (a\langle e'\rangle; \mathcal{Q}_1)$ — by S-STEP with (12)
⋆ $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_{\mathsf{v}}(a) = v$ — from (H2)

CASE: $e = v$ is a value
(16) $s = \mathsf{subscribers}_{\Delta}(a)$
(17) $\mathcal{S}(a) = (e', v')$ — (H2)
⋆ $\Delta; \mathcal{S}; (a\langle v\rangle; \mathcal{Q}_1) \to \Delta; \mathcal{S}[a \mapsto (e', v)]; (\mathcal{Q}_1; s)$ — by S-VALUE with (16, 17)
⋆ $\forall a \in \mathsf{dom}(\mathcal{S}[a \mapsto (e', v)]).\mathcal{S}[a \mapsto (e', v)]_{\mathsf{v}}(a) = v$ — from (H2)

3. $\mathcal{Q} = (a\langle v\rangle; \mathcal{Q}_1)$
(H1) $\vdash \Delta; \mathcal{S}; (a\langle v\rangle; \mathcal{Q}_1)$
(H2) $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_{\mathsf{v}}(a) = v$
(3) $\Delta \vdash \varepsilon * \mathcal{S}$ — by Definition 4.4 in (H1)
(4) $\Delta; \mathcal{S} \vdash (a\langle v\rangle; \mathcal{Q}_1)$ — by Definition 4.4 in (H2)
$\varepsilon \vDash \Delta$ — by Definition 4.4 in (H1)
$\Delta; \mathcal{S} \vdash \mathcal{Q}$ — by Definition 4.3 (WFQVAREVAL, WFQDEFEVAL) in (4)
$\Delta \vdash (\mathcal{S}; v)$ — by Definition 4.3 (WFQVAREVAL, WFQDEFEVAL) in (4)
$\Delta; \varepsilon \vdash^{\delta[a:\mathsf{var}(\tau)]} v : \tau \vee \Delta; \varepsilon \vdash^{\delta} v : \tau$ — by Definition 4.3 (WFQVAREVAL, WFQDEFEVAL) in (4)
(5) $\Delta(a) = \mathsf{var}_{\delta}(\tau) \vee \Delta(a) = \mathsf{def}_{\delta}(\tau)$ — by Definition 4.3 (WFQVAREVAL, WFQDEFEVAL) in (4)
(6) $\mathsf{dom}(\Delta) = \mathsf{dom}(\mathcal{S})$ — by Corollary B.1 with (3)
(7) $\mathcal{S}(a) = (e, v')$ — from (5, 6, H2)
(8) $\mathcal{S}' = \mathcal{S}[a \mapsto (e, v)]$ — from (7)
(9) $s = \mathsf{subscribers}_{\Delta}(a)$
⋆ $\Delta; \mathcal{S}; (a\langle v\rangle; \mathcal{Q}_1) \to \Delta; \mathcal{S}'; (s; \mathcal{Q}_1)$ — by S-VALUE with (8, 9)
⋆ $\forall a \in \mathsf{dom}(\mathcal{S}').\mathcal{S}'_{\mathsf{v}}(a) = v$ — from (H2, 8)

4. $\mathcal{Q} = (\mathbf{do}\ e; \mathcal{Q}_1)$
(H1) $\vdash \Delta; \mathcal{S}; (\mathbf{do}\ e; \mathcal{Q}_1)$
(H2) $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_{\mathsf{v}}(a) = v$
(3) $\Delta \vdash \varepsilon * \mathcal{S}$ — by Definition 4.4 in (H1)
(4) $\Delta; \mathcal{S} \vdash \mathbf{do}\ e; \mathcal{Q}_1$ — by Definition 4.4 in (H1)
$\varepsilon \vDash \Delta$ — by Definition 4.4 in (H1)
(5) $\Delta \vdash \mathbf{do}\ e : \varepsilon$ — by Definition 4.3 (WFQDO) in (4)
(6) $\Delta \vdash (\mathcal{S}; e)$ — by Definition 4.3 (WFQDO) in (4)
$\Delta; \mathcal{S} \vdash \mathcal{Q}$ — by Definition 4.3 (WFQDO) in (4)
(7) $\Delta; \varepsilon \vdash^{\delta'} e : \mathsf{Action}(\delta)$ — by inv. of T-DO in (5)
$\Delta \vdash \delta$ — by inv. of T-DO in (5)
(8) $\Delta \vdash \delta'$ — by inv. of T-DO in (5)
(9) $\mathsf{dom}(\delta') \subseteq \mathsf{dom}(\Delta)$ — by Corollary B.2 with (8)
(10) $\mathsf{dom}(\mathcal{S}) = \mathsf{dom}(\Delta)$ — by Corollary B.1 with (3)
(11) $\forall a \in \mathsf{dom}(\delta').\mathcal{S}_{\mathsf{v}}(a) = v$ — form (H2, 9, 10)
$\mathcal{S}; e \to e' \vee e$ is a value — by Lemma B.6 with (6, 7, 11)

(12) CASE: $\mathcal{S}; e \to e'$
⋆ $\Delta; \mathcal{S}; (\mathbf{do}\ e; \mathcal{Q}_1) \to \Delta; \mathcal{S}; (\mathbf{do}\ e'; \mathcal{Q}_1)$ — by S-DO with (12)
⋆ $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_{\mathsf{v}}(a) = v$ — from (H2)

(15) CASE: $e = v$ is a value
$e = \mathbf{action}\ \{a \leftarrow e''\}$ — form (15, 7)
$\mathcal{S}(a) = (e', v')$
⋆ $\Delta; \mathcal{S}; (\mathbf{do}\ e; \mathcal{Q}_1) \to \Delta; \mathcal{S}[a \mapsto (e'', v')]; (a; \mathcal{Q}_1)$ — by S-DO-ACTION
⋆ $\forall a \in \mathsf{dom}(\mathcal{S}[a \mapsto (e'', v')]).\mathcal{S}[a \mapsto (e'', v')]_{\mathsf{v}}(a) = v$ — from (H2)

5. $\mathcal{Q} = ([\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_1)$
(H1) $\vdash \Delta; \mathcal{S}; ([\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_1)$
(H2) $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_{\mathsf{v}}(a) = v$
$\Delta \vdash \varepsilon * \mathcal{S}$ — by Definition 4.4 in (H1)
(3) $\Delta; \mathcal{S} \vdash ([\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_1)$ — by Definition 4.4 in (H1)
$\varepsilon \vDash \Delta$ — by Definition 4.4 in (H1)

$\Delta \uplus \Delta' \vdash \varepsilon * (\mathcal{S} \uplus \mathcal{S}')$ by Definition 4.3 (WFQREmpty) in (3)
$\Delta \vDash \Delta'$ by Definition 4.3 (WFQREmpty) in (3)
$\Delta \uplus \Delta'; \mathcal{S} \uplus \mathcal{S}' \vdash \mathcal{Q}_1$ by Definition 4.3 (WFQREmpty) in (3)
(4) $\forall a \in \mathsf{dom}(\mathcal{S}').\mathcal{S}'_{\mathsf{v}}(a) = v$ by Lemma B.9 with (H1)
(5) $s = \mathsf{subscribers}_{\Delta \uplus \Delta'}(\mathsf{dom}(\Delta'))$ by Definition 3.6
⋆ $\Delta; \mathcal{S}; ([\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_1) \to \Delta \uplus \Delta'; \mathcal{S} \uplus \mathcal{S}'; (s; \mathcal{Q}_1)$ by S-REmpty with (5)
⋆ $\forall a \in \mathsf{dom}(\mathcal{S} \uplus \mathcal{S}').(\mathcal{S} \uplus \mathcal{S}')_{\mathsf{v}}(a) = v$ from Definition 3.7 and (H2, 4)

6. $\mathcal{Q} = ([\mathbf{var}\ a{=}e; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2)$
(H1) $\vdash \Delta; \mathcal{S}; ([\mathbf{var}\ a{=}e; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2)$
(H2) $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_{\mathsf{v}}(a) = v$
$\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.4 in (H1)
(3) $\Delta; \mathcal{S} \vdash ([\mathbf{var}\ a{=}e; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2)$ by Definition 4.4 in (H1)
$\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)
(4) $\Delta \uplus \Delta' \vdash \mathbf{var}\ a{=}e \dashv \Delta''$ by Definition 4.3 (WFQRVar) in (3)
$\Delta; \mathcal{S} \vdash [q_1]^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q}_2$ by Definition 4.3 (WFQRVar) in (3)
$\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'; e)$ by Definition 4.3 (WFQRVar) in (3)
⋆ $\Delta; \mathcal{S}; ([\mathbf{var}\ a{=}e; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2) \to \Delta; \mathcal{S}; ([a; q_1]^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q}_2)$ by S-RVar with (4)
⋆ $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_{\mathsf{v}}(a) = v$ from (H2)

7. $\mathcal{Q} = ([\mathbf{def}\ a{=}e; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2)$
(H1) $\vdash \Delta; \mathcal{S}; ([\mathbf{def}\ a{=}e; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2)$
(H2) $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_{\mathsf{v}}(a) = v$
$\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.4 in (H1)
(3) $\Delta; \mathcal{S} \vdash ([\mathbf{def}\ a{=}e; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2)$ by Definition 4.4 in (H1)
$\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)
(4) $\Delta \uplus \Delta' \vdash \mathbf{def}\ a{=}e \dashv \Delta''$ by Definition 4.3 (WFQRDef) in (3)
$\Delta; \mathcal{S} \vdash [q_1]^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q}_2$ by Definition 4.3 (WFQRDef) in (3)
$\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'; e)$ by Definition 4.3 (WFQRDef) in (3)
⋆ $\Delta; \mathcal{S}; ([\mathbf{def}\ a{=}e; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2) \to \Delta; \mathcal{S}; ([a; q_1]^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q}_2)$ by S-RDef with (4)
⋆ $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_{\mathsf{v}}(a) = v$ from (H2)

8. $\mathcal{Q} = ([a; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2)$
(H1) $\vdash \Delta; \mathcal{S}; ([a; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2)$
(H2) $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_{\mathsf{v}}(a) = v$
$\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.4 in (H1)
(3) $\Delta; \mathcal{S} \vdash ([a; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2)$ by Definition 4.4 in (H1)
$\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)
$(\Delta \uplus \Delta')(a) = \mathsf{var}_\delta(\tau) \vee (\Delta \uplus \Delta')(a) = \mathsf{def}_\delta(\tau)$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (3)
(4) $(\mathcal{S} \uplus \mathcal{S}')(a) = (e, \nu)$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (3)
$\Delta; \varepsilon \overset{\delta}{\vdash} e : \tau$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (3)
$\Delta; \mathcal{S} \vdash [q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (3)
⋆ $\Delta; \mathcal{S}; ([a; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2) \to \Delta; \mathcal{S}; ([a\langle e\rangle; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2)$ by S-RQueue with (4)
⋆ $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_{\mathsf{v}}(a) = v$ from (H2)

9. $\mathcal{Q} = ([a\langle e\rangle; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2)$
(H1) $\vdash \Delta; \mathcal{S}; ([a\langle e\rangle; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2)$
(H2) $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_{\mathsf{v}}(a) = v$
$\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.4 in (H1)
(3) $\Delta; \mathcal{S} \vdash ([a\langle e\rangle; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2)$ by Definition 4.4 in (H1)
$\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)
(4) $(\Delta \uplus \Delta')(a) = \mathsf{var}_\delta(\tau) \vee (\Delta \uplus \Delta')(a) = \mathsf{def}_\delta(\tau)$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (3)
(5) $\Delta \uplus \Delta'; \varepsilon \overset{\delta}{\vdash} e : \tau$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (3)
(6) $\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'; e)$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (3)
$\Delta; \mathcal{S} \vdash [q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2$ by Definition 4.3 (WFQRVarName, WFQRDefName) in (3)
(7) $\Delta \uplus \Delta' \vdash \delta$ by Definition 4.2 in (6)
(8) $\forall b \in \mathsf{dom}(\mathcal{S}') \wedge b \neq a.\mathcal{S}'_{\mathsf{v}}(b) = v$ by Lemma B.10 with (H1)
(9) $\forall b \in \mathsf{dom}(\mathcal{S} \uplus \mathcal{S}') \wedge b \neq a.(\mathcal{S} \uplus \mathcal{S}')_{\mathsf{v}}(b) = v$ from (H2, 8)

(10) $\mathsf{dom}(\delta) \subseteq \mathsf{dom}(\Delta \uplus \Delta')$ by Corollary B.2 in (7)
(11) $\Delta \uplus \Delta' \vdash \varepsilon * (\mathcal{S} \uplus \mathcal{S}')$ by Definition 4.2 in (6)
(12) $\mathsf{dom}(\Delta \uplus \Delta') = \mathsf{dom}(\mathcal{S} \uplus \mathcal{S}')$ by Corollary B.1 in (11)
(13) $\mathsf{dom}(\delta) \subseteq \mathsf{dom}(\mathcal{S} \uplus \mathcal{S}')$ from (10, 12)
(14) $a \notin \mathsf{dom}(\delta)$ by T-Var and T-Def in (4)
(15) $\forall b \in \mathsf{dom}(\delta) \wedge .(\mathcal{S} \uplus \mathcal{S}')_{\mathsf{v}}(b) = v$ from (9, 13, 14)
$\mathcal{S} \uplus \mathcal{S}'; e \to e' \vee e$ is a value by Lemma B.6 with (6, 5, 15)

(16) Case: $\mathcal{S} \uplus \mathcal{S}'; e \to e'$
- $\Delta; \mathcal{S}; ([a\langle e\rangle; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2) \to \Delta; \mathcal{S}; ([a\langle e'\rangle; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2)$ by S-RStep with (16)
- $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_{\mathsf{v}}(a) = v$ from (H2)

Case: $e = v$
(20) $a \in \mathsf{dom}(\mathcal{S} \uplus \mathcal{S}')$ from (4, 12)
(21) $\mathcal{S}'' = \mathcal{S}'[a \mapsto ((\mathcal{S} \uplus \mathcal{S}')_{\mathsf{e}}(a), v)]$
- $\Delta; \mathcal{S}; ([a\langle e\rangle; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2) \to \Delta; \mathcal{S}; ([q_1]^{\Delta'}_{\mathcal{S}''}; \mathcal{Q}_2)$ by S-RValue with (21)
- $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_{\mathsf{v}}(a) = v$ from (H2)

10. $\mathcal{Q} = ([a\langle v\rangle; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2)$
(H1) $\vdash \Delta; \mathcal{S}; ([a\langle v\rangle; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2)$
(H2) $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_{\mathsf{v}}(a) = v$
$\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.4 in (H1)
(3) $\Delta; \mathcal{S} \vdash ([a\langle v\rangle; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2)$ by Definition 4.4 in (H1)
$\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)
(4) $(\Delta \uplus \Delta')(a) = \mathsf{var}_{\vee}(\tau^{\delta})(\Delta \uplus \Delta')(a) = \mathsf{def}_{\delta'}(\tau^{\delta})$ by Definition 4.3 (WFQRVarEval, WFQRVarEval) in (3)
$\Delta \uplus \Delta'; \varepsilon \overset{\delta}{\vdash} v : \tau$ by Definition 4.3 (WFQRVarEval, WFQRVarEval) in (3)
(5) $\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'; v)$ by Definition 4.3 (WFQRVarEval, WFQRVarEval) in (3)
$\Delta; \mathcal{S} \vdash [q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2$ by Definition 4.3 (WFQRVarEval, WFQRVarEval) in (3)
(6) $\Delta \uplus \Delta' \vdash \varepsilon * (\mathcal{S} \uplus \mathcal{S}')$ by Definition 4.2 in (5)
(7) $\mathsf{dom}(\Delta \uplus \Delta') = \mathsf{dom}(\mathcal{S} \uplus \mathcal{S}')$ by Corollary B.1 in (6)
(8) $a \in \mathsf{dom}(\mathcal{S} \uplus \mathcal{S}')$ from (4, 7)
(9) $\mathcal{S}'' = \mathcal{S}'[a \mapsto ((\mathcal{S} \uplus \mathcal{S}')_{\mathsf{e}}(a), v)]$ from Definition A.3 with (8)
- $\Delta; \mathcal{S}; ([a\langle e\rangle; q_1]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}_2) \to \Delta; \mathcal{S}; ([q_1]^{\Delta'}_{\mathcal{S}''}; \mathcal{Q}_2)$ by S-RValue with (9)
- $\forall a \in \mathsf{dom}(\mathcal{S}).\mathcal{S}_{\mathsf{v}}(a) = v$ from (H2)

□

Theorem 4.6 (Runtime Type Preservation)**.** For all runtime configurations $(\Delta; \mathcal{S}; \mathcal{Q})$ and $(\Delta'; \mathcal{S}'; \mathcal{Q}')$, if $\vdash \Delta; \mathcal{S}; \mathcal{Q}$, and $\Delta; \mathcal{S}; \mathcal{Q} \to \Delta'; \mathcal{S}'; \mathcal{Q}'$ then, $\vdash \Delta'; \mathcal{S}'; \mathcal{Q}'$.

Proof. By induction on the length of the reduction $\Delta; \mathcal{S}; \mathcal{Q} \to \Delta'; \mathcal{S}'; \mathcal{Q}'$.
1. S-Do-Action
(H1) $\vdash \Delta; \mathcal{S}[a \mapsto (e, v)]; (\mathbf{do\ action}\ \{a \leftarrow e'\}; \mathcal{Q})$
(H2) $\Delta; \mathcal{S}[a \mapsto (e, v)]; (\mathbf{do\ action}\ \{a \leftarrow e'\}; \mathcal{Q}) \to \Delta; \mathcal{S}[a \mapsto (e', v)]; (a; \mathcal{Q})$
(3) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e, v)]$ by Definition 4.4 in (H1)
(4) $\Delta; \mathcal{S}[a \mapsto (e, v)] \vdash \mathbf{do\ action}\ \{a \leftarrow e'\}; \mathcal{Q}$ by Definition 4.4 in (H1)
(5) $\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)
(6) $\Delta(a) = \mathsf{var}_{\delta}(\tau)$ by Definition 4.1 (WTSVarValue) in (3)
$\Delta; \varepsilon \overset{\delta[a:\mathsf{var}(\tau)]}{\vdash} e : \tau$ by Definition 4.1 (WTSVarValue) in (3)
(7) $\Delta; \varepsilon \overset{\delta[a:\mathsf{var}(\tau)]}{\vdash} v : \tau$ by Definition 4.1 (WTSVarValue) in (3)
(8) $\Delta \vdash a \mapsto (e, \nu) * \mathcal{S}$ by Definition 4.1 (WTSVarValue) in (3)
(9) $\Delta \vdash \mathbf{do\ action}\ \{a \leftarrow e'\} : \varepsilon$ by Definition 4.3 (WFQDo) in (4)
(10) $\Delta \vdash (\mathcal{S}[a \mapsto (e, v)]; \mathbf{action}\ \{a \leftarrow e'\})$ by Definition 4.3 (WFQDo) in (4)
(11) $\Delta; \mathcal{S} \vdash \mathcal{Q}$ by Definition 4.3 (WFQDo) in (4)
(12) $\Delta; \varepsilon \overset{\delta'}{\vdash} \mathbf{action}\ \{a \leftarrow e'\} : \mathsf{Action}(\delta[a : \mathsf{var}(\tau)])$ by inv. of T-Do in (9)
(13) $\Delta \vdash \delta[a : \mathsf{var}(\tau)]$ by inv. of T-Do in (9)
(14) $\Delta; \varepsilon \overset{\delta[a:\mathsf{var}(\tau)]}{\vdash} e' : \tau$ by inv. of T-Action in (12)
(15) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e', v)]$ by Definition 4.1 (WTSVarValue) with (6, 13, 7, 8)
(16) $\mathcal{S}[a \mapsto (e', v)](a) = (e', v)$ from (15)
(17) $\Delta \vdash (\mathcal{S}[a \mapsto (e', v)]; e')$ by Definition 4.2 with (15, 14, 13)
(18) $\Delta \vdash (\mathcal{S}[a \mapsto (e', v)]; v)$ by Definition 4.2 with (15, 7, 13)
(19) $\Delta; \mathcal{S}[a \mapsto (e', v)] \vdash a; \mathcal{Q}$ by Definition 4.3 (WFQName) with (6, 16, 11, 17, 18)

$\star$ $\vdash \Delta; \mathcal{S}[a \mapsto (e', v)]; (a; \mathcal{Q})$ — Definition 4.4 with (15, 19, 5)

2. S-Do

(H1) $\vdash \Delta; \mathcal{S}; (\mathbf{do}\ e; \mathcal{Q})$
(H2) $\Delta; \mathcal{S}; (\mathbf{do}\ e; \mathcal{Q}) \rightarrow \Delta; \mathcal{S}; (\mathbf{do}\ e'; \mathcal{Q})$
(3) $\Delta \vdash \varepsilon * \mathcal{S}$ — by Definition 4.4 in (H1)
$\Delta; \mathcal{S} \vdash \mathbf{do}\ e; \mathcal{Q}$ — by Definition 4.4 in (H1)
(4) $\varepsilon \vDash \Delta$ — by Definition 4.4 in (H1)
(5) $\Delta \vdash \mathbf{do}\ e : \varepsilon$ — by Definition 4.3 (WFQDo) in (H1)
(6) $\Delta \vdash (\mathcal{S}; e)$ — by Definition 4.3 (WFQDo) in (H1)
(7) $\Delta; \mathcal{S} \vdash \mathcal{Q}$ — by Definition 4.3 (WFQDo) in (H1)
(8) $\mathcal{S}; e \rightarrow e'$ — by S-Do in (H2)
(9) $\Delta; \varepsilon \vdash^{\delta'} e : \mathsf{Action}(\delta)$ — by inv. T-Do in (5)
(10) $\Delta \vdash \delta$ — by inv. T-Do in (5)
(11) $\Delta \vdash \delta'$ — by inv. T-Do in (5)
(12) $\Delta; \varepsilon \vdash^{\delta'} e' : \mathsf{Action}(\delta)$ — Lemma B.7 with (6, 9, 8)
(13) $\Delta \vdash \mathbf{do}\ e' : \varepsilon$ — by T-Do with (12, 10, 11)
(14) $\Delta \vdash (\mathcal{S}; e')$ — by Definition 4.2 with (3, 12, 10)
(15) $\Delta; \mathcal{S} \vdash \mathbf{do}\ e'; \mathcal{Q}$ — by Definition 4.3 (WFQDo) with (13, 14, 7)
$\star$ $\vdash \Delta; \mathcal{S}; (\mathbf{do}\ e'; \mathcal{Q})$ — by Definition 4.4 with (3, 15, 4)

3. S-Queue

(H1) $\vdash \Delta; \mathcal{S}; (a; \mathcal{Q})$
(H2) $\Delta; \mathcal{S}; (a; \mathcal{Q}) \rightarrow \Delta; \mathcal{S}; (a\langle e\rangle; \mathcal{Q})$
(3) $\Delta \vdash \varepsilon * \mathcal{S}$ — by Definition 4.4 in (H1)
(4) $\Delta; \mathcal{S} \vdash a; \mathcal{Q}$ — by Definition 4.4 in (H1)
(5) $\varepsilon \vDash \Delta$ — by Definition 4.4 in (H1)
(6) $\Delta(a) = \sigma$ — by Definition 4.3 (WFQName) in (4)
(7) $\mathcal{S}(a) = (e, v)$ — by Definition 4.3 (WFQName) in (4)
(8) $\Delta; \mathcal{S} \vdash \mathcal{Q}$ — by Definition 4.3 (WFQName) in (4)
(9) $\Delta \vdash (\mathcal{S}; e)$ — by Definition 4.3 (WFQName) in (4)
$\Delta \vdash (\mathcal{S}; v)$ — by Definition 4.3 (WFQName) in (4)
(10) $\mathcal{S} = \mathcal{S}', a \mapsto (e, v)$ — from (7)
(11) $\sigma = \mathsf{var}_\delta(\tau) \vee \sigma = \mathsf{def}_\delta(\tau)$
$\Delta \vdash a \mapsto (e, \nu) * \mathcal{S}'$ — by Definition 4.1 (WTSVarValue, WTSDefValue) in (3, 7)
(12) Case: $\Delta(a) = \mathsf{var}_\delta(\tau)$
(13) $\Delta; \varepsilon \vdash^{\delta[a:\mathsf{var}(\tau)]} e : \tau$ — by Definition 4.1 (WTSVarValue) in (3, 7)
$\Delta; \varepsilon \vdash^{\delta[a:\mathsf{var}(\tau)]} v : \tau$ — by Definition 4.1 (WTSVarValue) in (3, 7)
(14) $\Delta; \mathcal{S} \vdash a\langle e\rangle; \mathcal{Q}$ — by Definition 4.3 (WFQVarEval) with (11, 13, 8, 9)
$\star$ $\vdash \Delta; \mathcal{S}; (a\langle e\rangle; \mathcal{Q})$ — by Definition 4.4 with (3, 14, 5)
(16) Case: $\Delta(a) = \mathsf{def}_\delta(\tau)$
(17) $\Delta; \varepsilon \vdash^{\delta} e : \tau$ — by Definition 4.1 (WTSDefValue) in (3, 7)
$\Delta; \varepsilon \vdash^{\delta} v : \tau$ — by Definition 4.1 (WTSDefValue) in (3, 7)
(18) $\Delta; \mathcal{S} \vdash a\langle e\rangle; \mathcal{Q}$ — by Definition 4.3 (WFQDefEval) with (11, 17, 8, 9)
$\star$ $\vdash \Delta; \mathcal{S}; (a\langle e\rangle; \mathcal{Q})$ — by Definition 4.4 with (3, 18, 5)

4. S-Step

(H1) $\vdash \Delta; \mathcal{S}; (a\langle e\rangle; \mathcal{Q})$
(H2) $\Delta; \mathcal{S}; (a\langle e\rangle; \mathcal{Q}) \rightarrow \Delta; \mathcal{S}; (a\langle e'\rangle; \mathcal{Q})$
(3) $\Delta \vdash \varepsilon * \mathcal{S}$ — by Definition 4.4 in (H1)
(4) $\Delta; \mathcal{S} \vdash a\langle e\rangle; \mathcal{Q}$ — by Definition 4.4 in (H1)
(5) $\varepsilon \vDash \Delta$ — by Definition 4.4 in (H1)
(6) $\mathcal{S}; e \rightarrow e'$ — by inv. of S-Step in (H2)
(7) $\Delta(a) = \mathsf{var}_\delta(\tau) \vee \Delta(a) = \mathsf{def}_\delta(\tau)$ — by Definition 4.3 (WFQVarEval, WFQDefEval) in (4)
(8) $\Delta; \mathcal{S} \vdash \mathcal{Q}$ — by Definition 4.3 (WFQVarEval, WFQDefEval) in (4)
(9) $\Delta \vdash (\mathcal{S}; e)$ — by Definition 4.3 (WFQVarEval, WFQDefEval) in (4)
(10) Case: $\Delta(a) = \mathsf{var}_\delta(\tau)$
(11) $\Delta; \varepsilon \vdash^{\delta[a:\mathsf{var}(\tau)]} e : \tau$ — by Definition 4.3 (WFQVarEval) in (4)
(12) $\Delta \vdash \delta$ — by Definition 4.2 in (9)
(13) $\Delta; \Delta \vdash^{\delta[a:\mathsf{var}(\tau)]} e' : \tau$ — by Lemma B.7 with (9, 11, 6)
(14) $\Delta \vdash (\mathcal{S}; e')$ — by Definition 4.2 with (3, 13, 12)
(15) $\Delta; \mathcal{S} \vdash a\langle e'\rangle; \mathcal{Q}$ — by Definition 4.3 (WFQVarEval) with (7, 13, 8, 14)

$\star$ $\vdash \Delta; \mathcal{S}; (a\langle e' \rangle; \mathcal{Q})$ by Definition 4.4 with (3, 15, 5)

(17) CASE: $\Delta(a) = \mathsf{def}_\delta(\tau)$

(18) $\Delta; \varepsilon \overset{\delta}{\vdash} e : \tau$ by Definition 4.3 (WFQDEFEVAL) in (4)

(19) $\Delta \vdash \delta$ by Definition 4.2 in (9)

(20) $\Delta; \Delta \overset{\delta}{\vdash} e' : \tau$ by Lemma B.7 with (9, 18, 6)

(21) $\Delta \vdash (\mathcal{S}; e')$ by Definition 4.2 with (3, 20, 19)

(22) $\Delta; \mathcal{S} \vdash a\langle e' \rangle; \mathcal{Q}$ by Definition 4.3 (WFQDEFEVAL) with (7, 20, 8, 21)

$\star$ $\vdash \Delta; \mathcal{S}; (a\langle e' \rangle; \mathcal{Q})$ by Definition 4.4 with (3, 22, 5)

5. S-VALUE

(H1) $\vdash \Delta; \mathcal{S}[a \mapsto (e, v')]; (a\langle v \rangle; \mathcal{Q})$

(H2) $\Delta; \mathcal{S}[a \mapsto (e, v')]; (a\langle v \rangle; \mathcal{Q}) \to \Delta; \mathcal{S}[a \mapsto (e, v)]; (s; \mathcal{Q})$

(3) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e, v')]$ by Definition 4.4 in (H1)

(4) $\Delta; \mathcal{S}[a \mapsto (e, v')] \vdash a\langle v \rangle; \mathcal{Q}$ by Definition 4.4 in (H1)

(5) $\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)

(6) $s = \mathsf{subscribers}_\Delta(a)$ by inv. of S-VALUE in (H2)

(7) $\mathcal{S}[a \mapsto (e, v')] = \mathcal{S}', a \mapsto (e, v')$

(8) $\mathsf{dom}(\mathcal{S}[a \mapsto (e, v')]) = \mathsf{dom}(\Delta)$ by Corollary B.1 in (3)

(9) $s \subseteq \mathsf{dom}(\Delta)$ by Definition 3.6 in (6)

(10) $s \subseteq \mathsf{dom}(\mathcal{S}[a \mapsto (e, v')])$ from (8, 9)

(11) $\mathcal{S}[a \mapsto (e, v')](s_i) = (e_i, v_i)$ from (10)

(12) $\Delta(s_i) = \mathsf{var}_{\delta_i}(\tau) \vee \Delta(s_i) = \mathsf{def}_{\delta_i}(\tau)$ from (9)

(13) $\Delta \vdash \delta_i$ by Definition 3.11 with (5) and (9)

(14) $\Delta; \varepsilon \overset{\delta_i}{\vdash} e_i : \tau$ by Definition 4.1 (WTSVARVALUE, WTSDEFVALUE) in (3)

(15) $\Delta; \varepsilon \overset{\delta_i}{\vdash} v_i : \tau$ by Definition 4.1 (WTSVARVALUE, WTSDEFVALUE) in (3)

(16) $\Delta \vdash (\mathcal{S}[a \mapsto (e, v')]; e_i)$ by Definition 4.2 with (3, 14, 13)

(17) $\Delta \vdash (\mathcal{S}[a \mapsto (e, v')]; v_i)$ by Definition 4.2 with (3, 15, 13)

(18) $\Delta \vdash a \mapsto (e, \nu) * \mathcal{S}'$ by Definition 4.1 (WTSVARVALUE, WTSDEFVALUE) in (3)

(19) $\Delta(a) = \mathsf{var}_\delta(\tau) \vee \Delta(a) = \mathsf{def}_\delta(\tau)$ by Definition 4.1 (WTSVARVALUE, WTSDEFVALUE) in (3)

CASE: $\Delta(a) = \mathsf{var}_\delta(\tau)$

(21) $\Delta; \varepsilon \overset{\delta[a:\mathsf{var}(\delta)]}{\vdash} e : \tau$ by Definition 4.1 (WTSVARVALUE) in (3)

$\Delta; \varepsilon \overset{\delta[a:\mathsf{var}(\delta)]}{\vdash} v' : \tau$ by Definition 4.1 (WTSVARVALUE) in (3)

(22) $\Delta; \varepsilon \overset{\delta[a:\mathsf{var}(\delta)]}{\vdash} v : \tau$ by Definition 4.3 (WFQVAREVAL) in (4)

(23) $\Delta; \mathcal{S}[a \mapsto (e, v')] \vdash \mathcal{Q}$ by Definition 4.3 (WFQVAREVAL) in (4)

$\Delta \vdash (\mathcal{S}[a \mapsto (e, v')]; v)$ by Definition 4.3 (WFQVAREVAL) in (4)

(24) $\vdash \Delta; \mathcal{S}[a \mapsto (e, v')]; \mathcal{Q}$ by Definition 4.4 with (3, 23, 5)

(25) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e, v)]$ by Definition 4.1 (WTSVARVALUE) with (19, 21, 22, 18)

(26) $\vdash \Delta; \mathcal{S}[a \mapsto (e, v)]; \mathcal{Q}$ by Lemma B.12 with (24, 25, 21, 22)

(27) $\Delta; \mathcal{S}[a \mapsto (e, v)] \vdash \mathcal{Q}$ by Definition 4.4 in (26)

(28) $\Delta; \mathcal{S}[a \mapsto (e, v)] \vdash s; \mathcal{Q}$ by Definition 4.3 (WFQNAME) with (12, 11, 27, 16, 17)

$\star$ $\vdash \Delta; \mathcal{S}[a \mapsto (e, v)]; (s; \mathcal{Q})$ by Definition 4.4 with (27, 28, 5)

CASE: $\Delta(a) = \mathsf{def}_\delta(\tau)$

(31) $\Delta; \varepsilon \overset{\delta}{\vdash} e : \tau$ by Definition 4.1 (WTSDEFVALUE) in (3)

$\Delta; \varepsilon \overset{\delta}{\vdash} v' : \tau$ by Definition 4.1 (WTSDEFVALUE) in (3)

(32) $\Delta; \varepsilon \overset{\delta}{\vdash} v : \tau$ by Definition 4.3 (WFQDEFEVAL) in (4)

(33) $\Delta; \mathcal{S}[a \mapsto (e, v')] \vdash \mathcal{Q}$ by Definition 4.3 (WFQDEFEVAL) in (4)

$\Delta \vdash (\mathcal{S}[a \mapsto (e, v')]; v)$ by Definition 4.3 (WFQDEFEVAL) in (4)

(34) $\vdash \Delta; \mathcal{S}[a \mapsto (e, v')]; \mathcal{Q}$ by Definition 4.4 with (3, 33, 5)

(35) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e, v)]$ by Definition 4.1 (WTSDEFVALUE) with (19, 31, 32, 18)

(36) $\vdash \Delta; \mathcal{S}[a \mapsto (e, v)]; \mathcal{Q}$ by Lemma B.12 with (34, 35, 31, 32)

(37) $\Delta; \mathcal{S}[a \mapsto (e, v)] \vdash \mathcal{Q}$ by Definition 4.4 in (36)

(38) $\Delta; \mathcal{S}[a \mapsto (e, v)] \vdash s; \mathcal{Q}$ by Definition 4.3 (WFQNAME) with (12, 11, 37, 16, 17)

$\star$ $\vdash \Delta; \mathcal{S}[a \mapsto (e, v)]; (s; \mathcal{Q})$ by Definition 4.4 with (37, 38, 5)

6. S-REMPTY

(H1) $\vdash \Delta; \mathcal{S}; ([\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q})$

(H2) $\Delta; \mathcal{S}; ([\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}) \to \Delta \uplus \Delta'; \mathcal{S} \uplus \mathcal{S}'; (s; \mathcal{Q})$

$\Delta \vdash \varepsilon * \mathcal{S}$ by Definition 4.4 in (H1)

(3) $\Delta; \mathcal{S} \vdash [\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}$ by Definition 4.4 in (H1)

(4) $\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)

(5) $\Delta \uplus \Delta' \vdash \varepsilon * (\mathcal{S} \uplus \mathcal{S}')$ by Definition 4.3 (WFQREMPTY) in (3)

(6) $\Delta \vDash \Delta'$ — by Definition 4.3 (WFQREmpty) in (3)

(7) $\Delta \uplus \Delta'; \mathcal{S} \uplus \mathcal{S}' \vdash \mathcal{Q}$ — by Definition 4.3 (WFQREmpty) in (3)

(8) $s = \mathsf{subscribers}_{\Delta \uplus \Delta'}(\mathsf{dom}(\Delta'))$ — by inv. S-REmpty in (H2)

(9) $s \subseteq \mathsf{dom}(\Delta \uplus \Delta')$ — from (8)

(10) $\mathsf{dom}(\Delta \uplus \Delta') = \mathsf{dom}(\mathcal{S} \uplus \mathcal{S}')$ — by Corollary B.1 with (5)

(11) $s \subseteq \mathsf{dom}(\mathcal{S} \uplus \mathcal{S}')$ — from (9, 10)

(12) $(\mathcal{S} \uplus \mathcal{S}')(s_i) = (e_i, v_i)$ — from (11) and Lemma B.9 with (H1)

(13) $(\Delta \uplus \Delta')(s_i) = \mathsf{var}_{\delta_i}(\tau) \vee (\Delta \uplus \Delta')(s_i) = \mathsf{def}_{\delta_i}(\tau)$ — from (9)

(14) $\Delta \uplus \Delta' \vdash \delta_i$ — by Definition 3.11 in (4, 6)

(15) $\Delta \uplus \Delta'; \varepsilon \vdash^{\delta_i} e_i : \tau$ — by Definition 4.1 (WTSVarValue, WTSDefValue) in (5)

(16) $\Delta \uplus \Delta'; \varepsilon \vdash^{\delta_i} v_i : \tau$ — by Definition 4.1 (WTSVarValue, WTSDefValue) in (5)

(17) $\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'; e_i)$ — by Definition 4.2 with (5, 15, 14)

(18) $\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'; v_i)$ — by Definition 4.2 with (5, 16, 14)

(19) $\Delta \uplus \Delta'; \mathcal{S} \uplus \mathcal{S}' \vdash s; \mathcal{Q}$ — by Definition 4.3 (WFQName) with (13, 12, 17, 18, 7)

(20) $\varepsilon \vDash \Delta \uplus \Delta'$ — by Definition 3.11 with (4, 6)

$\star$ $\vdash \Delta \uplus \Delta'; \mathcal{S} \uplus \mathcal{S}'; \mathcal{Q}$ — by Definition 4.4 with (5, 19, 20)

7. S-RVar

(H1) $\vdash \Delta; \mathcal{S}; ([\mathbf{var}\ a{=}e; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}')$

(H2) $\Delta; \mathcal{S}; ([\mathbf{var}\ a{=}e; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') \to \Delta; \mathcal{S}; ([a; q]^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q}')$

(3) $\Delta \vdash \varepsilon * \mathcal{S}$ — by Definition 4.4 in (H1)

(4) $\Delta; \mathcal{S} \vdash [\mathbf{var}\ a{=}e; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ — by Definition 4.4 in (H1)

(5) $\varepsilon \vDash \Delta$ — by Definition 4.4 in (H1)

(6) $\Delta \uplus \Delta' \vdash \mathbf{var}\ a{=}e \dashv \Delta''$ — by Definition 4.4 in (H2)

(7) $\Delta; \mathcal{S} \vdash [q]^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q}'$ — by Definition 4.3 (WFQRVar) in (4)

(8) $\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'; e)$ — by Definition 4.3 (WFQRVar) in (4)

(9) $\mathcal{S}'' = \mathcal{S}'[a \mapsto (e, \square)]$

(10) $\mathcal{S}''(a) = (e, \square)$ — from (9)

(11) $(\mathcal{S} \uplus \mathcal{S}'')(a) = (e, \square)$ — by Definition 3.7 and (10)

(12) $\Delta'' = \{a : \mathsf{var}_\delta(\tau)\}$ — by Definition 3.12 in (6)

(13) $(\Delta \uplus \Delta' \uplus \Delta'')(a) = \mathsf{var}_\delta(\tau)$ — by Definition 3.3 and (12)

(14) $\Delta \uplus \Delta' \vdash \mathbf{var}\ a{=}e : \Delta''$ — by Definition 3.12 in (6)

$\Delta \uplus \Delta' \vdash \varepsilon \dashv \varepsilon$ — by Definition 3.12 in (6)

(15) $\Delta \uplus \Delta'; \varepsilon \vdash^{\delta} e : \tau$ — by inv. T-Var in (14) and (12)

$\Delta \uplus \Delta' \vdash \delta$ — by inv. T-Var in (14) and (12)

$a \notin \mathsf{dom}(\delta)$ — by inv. T-Var in (14) and (12)

(16) $\Delta; \mathcal{S} \vdash [a; q]^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q}'$ — by Definition 4.3 (WFQRVarName) with (13, 8, 11, 15, 7)

$\star$ $\vdash \Delta; \mathcal{S}; ([a; q]^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q}')$ — by Definition 4.4 with (3, 16, 5)

8. S-RDef

(H1) $\vdash \Delta; \mathcal{S}; ([\mathbf{def}\ a{=}e; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}')$

(H2) $\Delta; \mathcal{S}; ([\mathbf{def}\ a{=}e; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') \to \Delta; \mathcal{S}; ([a; q]^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q}')$

(3) $\Delta \vdash \varepsilon * \mathcal{S}$ — by Definition 4.4 in (H1)

(4) $\Delta; \mathcal{S} \vdash [\mathbf{def}\ a{=}e; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ — by Definition 4.4 in (H1)

(5) $\varepsilon \vDash \Delta$ — by Definition 4.4 in (H1)

(6) $\Delta \uplus \Delta' \vdash \mathbf{def}\ a{=}e \dashv \Delta''$ — by Definition 4.4 in (H2)

(7) $\Delta; \mathcal{S} \vdash [q]^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q}'$ — by Definition 4.3 (WFQRDef) in (4)

(8) $\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'; e)$ — by Definition 4.3 (WFQRDef) in (4)

(9) $\mathcal{S}'' = \mathcal{S}'[a \mapsto (e, \square)]$

(10) $\mathcal{S}''(a) = (e, \square)$ — from (9)

(11) $(\mathcal{S} \uplus \mathcal{S}'')(a) = (e, \square)$ — by Definition 3.7 and (10)

(12) $\Delta'' = \{a : \mathsf{def}_\delta(\tau)\}$ — by Definition 3.12 in (6)

(13) $(\Delta \uplus \Delta' \uplus \Delta'')(a) = \mathsf{def}_\delta(\tau)$ — by Definition 3.3 and (12)

(14) $\Delta \uplus \Delta' \vdash \mathbf{def}\ a{=}e : \Delta''$ — by Definition 3.12 in (6)

$\Delta \uplus \Delta' \vdash \varepsilon \dashv \varepsilon$ — by Definition 3.12 in (6)

(15) $\Delta \uplus \Delta'; \varepsilon \vdash^{\delta} e : \tau$ — by inv. T-Def in (14) and (12)

$\Delta \uplus \Delta' \vdash \delta$ — by inv. T-Def in (14) and (12)

$a \notin \mathsf{dom}(\delta)$ — by inv. T-Def in (14) and (12)

(16) $\Delta; \mathcal{S} \vdash [a; q]^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q}'$ — by Definition 4.3 (WFQRDefName) with (13, 8, 11, 15, 7)

$\star$ $\vdash \Delta; \mathcal{S}; ([a; q]^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q}')$ — by Definition 4.4 with (3, 16, 5)

9. S-RQUEUE

(H1) $\vdash \Delta; \mathcal{S}; ([a; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}')$

(H2) $\Delta; \mathcal{S}; ([a; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') \to \Delta; \mathcal{S}; ([a\langle e\rangle; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}')$

(3) $\Delta \vdash \varepsilon * \mathcal{S}$ — by Definition 4.4 in (H1)

(4) $\Delta; \mathcal{S} \vdash [a; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ — by Definition 4.4 in (H1)

(5) $\varepsilon \vDash \Delta$ — by Definition 4.4 in (H1)

$e = \mathcal{S} \uplus \mathcal{S}'_{\mathsf{e}}(a)$ — by inv. S-RQUEUE in (H2)

(6) $\Delta(a) = \mathsf{var}_\delta(\tau) \vee \Delta(a) = \mathsf{def}_\delta(\tau)$ — by Definition 4.3 (WFQRVAR, WFQRDEF) in (4)

$(\mathcal{S} \uplus \mathcal{S}')(a) = (e, \nu)$ — by Definition 4.3 (WFQRVAR, WFQRDEF) in (4)

(7) $\Delta \uplus \Delta'; \varepsilon \overset{\delta}{\vdash} e : \tau$ — by Definition 4.3 (WFQRVAR, WFQRDEF) in (4)

(8) $\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'; e)$ — by Definition 4.3 (WFQRVAR, WFQRDEF) in (4)

(9) $\Delta; \mathcal{S} \vdash [q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ — by Definition 4.3 (WFQRVAR, WFQRDEF) in (4)

(10) $\Delta; \mathcal{S} \vdash [a\langle e\rangle; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ — by Definition 4.3 (WFQRVAR, WFQRDEF) with (6, 7, 8, 9)

$\star$ $\vdash \Delta; \mathcal{S}; ([a\langle e\rangle; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}')$ — by Definition 4.4 with (3, 10, 5)

10. S-RSTEP

(H1) $\vdash \Delta; \mathcal{S}; ([a\langle e\rangle; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}')$

(H2) $\Delta; \mathcal{S}; ([a\langle e\rangle; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') \to \Delta; \mathcal{S}; ([a\langle e'\rangle; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}')$

(3) $\Delta \vdash \varepsilon * \mathcal{S}$ — by Definition 4.4 in (H1)

(4) $\Delta; \mathcal{S} \vdash [a\langle e\rangle; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ — by Definition 4.4 in (H1)

(5) $\varepsilon \vDash \Delta$ — by Definition 4.4 in (H1)

(6) $\mathcal{S} \uplus \mathcal{S}'; e \to e'$ — by inv. S-RSTEP in (H2)

(7) $(\Delta \uplus \Delta')(a) = \mathsf{var}_\delta(\tau) \vee (\Delta \uplus \Delta')(a) = \mathsf{def}_\delta(\tau)$ — by Definition 4.3 (WFQRVAR, WFQRDEF) in (4)

(8) $\Delta \uplus \Delta'; \varepsilon \overset{\delta}{\vdash} e : \tau$ — by Definition 4.3 (WFQRVAR, WFQRDEF) in (4)

(9) $\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}; e)$ — by Definition 4.3 (WFQRVAR, WFQRDEF) in (4)

(10) $\Delta; \mathcal{S} \vdash [q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ — by Definition 4.3 (WFQRVAR, WFQRDEF) in (4)

(11) $\Delta \uplus \Delta'; \varepsilon \overset{\delta}{\vdash} e' : \tau$ — by Lemma B.7 with (9, 8, 6)

(12) $\Delta \uplus \Delta' \vdash \varepsilon * (\mathcal{S} \uplus \mathcal{S}')$ — by Definition 4.2 in (9)

(13) $\Delta \uplus \Delta' \vdash \delta$ — by Definition 4.2 in (9)

(14) $\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'; e')$ — by Definition 4.2 with (12, 11, 13)

(15) $\Delta; \mathcal{S} \vdash [a\langle e'\rangle; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ — by Definition 4.3 (WFQRVAR, WFQRDEF) with (7, 11, 14, 10)

$\star$ $\vdash \Delta; \mathcal{S}; [a\langle e'\rangle; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ — by Definition 4.4 with (3, 15)

11. S-RVALUE

(H1) $\vdash \Delta; \mathcal{S}; ([a\langle v\rangle; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}')$

(H2) $\Delta; \mathcal{S}; ([a\langle v\rangle; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') \to \Delta; \mathcal{S}; ([q]^{\Delta'}_{\mathcal{S}''}; \mathcal{Q}')$

(3) $\Delta \vdash \varepsilon * \mathcal{S}$ — by Definition 4.4 in (H1)

(4) $\Delta; \mathcal{S} \vdash [a\langle v\rangle; q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ — by Definition 4.4 in (H1)

(5) $\varepsilon \vDash \Delta$ — by Definition 4.4 in (H1)

(6) $\mathcal{S}'' = \mathcal{S}'[a \mapsto ((\mathcal{S} \uplus \mathcal{S}')_{\mathsf{e}}(a), v)]$ — by inv. S-RVALUE in (H2)

(7) $\mathcal{S}'(a) = (e, \nu)$ — from (6)

$e = (\mathcal{S} \uplus \mathcal{S}')_{\mathsf{e}}(a)$

(8) $(\Delta \uplus \Delta')(a) = \mathsf{var}_\delta(\tau) \vee (\Delta \uplus \Delta')(a) = \mathsf{def}_\delta(\tau)$ — by Definition 4.3 (WFQRVARNAME, WFQRDEFNAME) in (4)

(9) $\Delta \uplus \Delta'; \varepsilon \overset{\delta}{\vdash} v : \tau$ — by Definition 4.3 (WFQRVARNAME, WFQRDEFNAME) in (4)

(10) $\Delta \uplus \Delta' \vdash (\mathcal{S} \uplus \mathcal{S}'; v)$ — by Definition 4.3 (WFQRVARNAME, WFQRDEFNAME) in (4)

(11) $\Delta; \mathcal{S} \vdash [q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'$ — by Definition 4.3 (WFQRVARNAME, WFQRDEFNAME) in (4)

(12) $\Delta \uplus \Delta' \vdash \varepsilon * (\mathcal{S} \uplus \mathcal{S}')$ — by Definition 4.2 in (10)

(13) $\Delta \uplus \Delta' \vdash \delta$ — by Definition 4.2 in (10)

(14) $\Delta \uplus \Delta'; \varepsilon \overset{\delta}{\vdash} e : \tau$ — by Definition 4.1 (WTSVAR, WTSVARVALUE, WTSDEF, WTSDEFVALUE) in (12)

$\mathcal{S}' = \mathcal{S}'_2, a \mapsto (e, \nu)$

(15) $\Delta \uplus \Delta' \vdash a \mapsto (e, \nu) * (\mathcal{S} \uplus \mathcal{S}'_2)$ — by Definition 4.1 (WTSVAR, WTSVARVALUE, WTSDEF, WTSDEFVALUE) in (12)

(16) $\Delta \uplus \Delta' \vdash \varepsilon * (\mathcal{S} \uplus \mathcal{S}'')$ — by Definition 4.1 (WTSVARVALUE, WTSDEFVALUE) with (8, 14, 9, 15)

(17) $\vdash \Delta; \mathcal{S}; ([q]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}')$ — by Definition 4.4 with (3, 11, 5)

$\star$ $\vdash \Delta; \mathcal{S}; ([q]^{\Delta'}_{\mathcal{S}''}; \mathcal{Q}')$ — by Lemma B.13 with (17, 7, 14, 9, 16)

□

Lemma 4.7 (Bound Queue Length)**.** If $\vdash (\Delta; \mathcal{S}; \mathcal{Q})$ and $\Delta; \mathcal{S}; \mathcal{Q} \to \Delta'; \mathcal{S}'; \mathcal{Q}'$ then one of the following holds

i. $\mathcal{Q}' = \mathbf{do}\ e; \mathcal{Q}''$ and $\#_\Delta(\mathcal{Q}) = \#_{\Delta'}(\mathcal{Q}')$
ii. $\mathcal{Q}' = a\langle e\rangle; \mathcal{Q}''$ and $\#_\Delta(\mathcal{Q}) = \#_{\Delta'}(\mathcal{Q}')$
iii. $\mathcal{Q}' = [a\langle e\rangle; q'']^{\Delta''}_{\mathcal{S}''}; \mathcal{Q}'''$ and $\#_\Delta(\mathcal{Q}) = \#_{\Delta'}(\mathcal{Q}')$
iv. $\#_\Delta(\mathcal{Q}) > \#_{\Delta'}(\mathcal{Q}')$

Proof. By induction on the length of the reduction $\Delta; \mathcal{S}; \mathcal{Q} \to \Delta'; \mathcal{S}'; \mathcal{Q}'$.

1. S-Do-Action

(H1) $\vdash \Delta; \mathcal{S}[a \mapsto (e', v)]; (\mathbf{do\ action}\ \{a \leftarrow e\}; \mathcal{Q}')$
(H2) $\Delta; \mathcal{S}[a \mapsto (e', v)]; (\mathbf{do\ action}\ \{a \leftarrow e\}; \mathcal{Q}') \to \Delta; \mathcal{S}[a \mapsto (e, v)]; (a; \mathcal{Q}')$
(3) $\Delta \vdash \varepsilon * \mathcal{S}[a \mapsto (e', v)]$ by Definition 4.4 in (H1)
$\Delta; \mathcal{S}[a \mapsto (e', v)] \vdash \mathbf{do\ action}\ \{a \leftarrow e\}; \mathcal{Q}'$ by Definition 4.4 in (H1)
$\varepsilon \vDash \Delta$ by Definition 4.4 in (H1)
(4) $\#_\Delta(\mathbf{do\ action}\ \{a \leftarrow e\}; \mathcal{Q}') = 1 + \#^{\mathsf{max}}_\Delta + \#_\Delta(\mathcal{Q}')$ by Definition A.5
(5) $\mathsf{dom}(\Delta) = \mathsf{dom}(\mathcal{S}[a \mapsto (e', v)])$ by Corollary B.1 in (3)
(6) $a \in \mathsf{dom}(\mathcal{S}[a \mapsto (e', v)])$
(7) $a \in \mathsf{dom}(\Delta)$ from (5, 6)
(8) $\#^{\mathsf{max}}_\Delta \geq \#_\Delta(a)$ by Definition A.6 and (7)
(9) $\#_\Delta(a) = 1 + \#_\Delta(\mathsf{subscribers}_\Delta(a))$ by Definition A.5
(10) $\#^{\mathsf{max}}_\Delta \geq 1 + \#_\Delta(\mathsf{subscribers}_\Delta(a))$ from (8, 9)
(11) $\#^{\mathsf{max}}_\Delta + \#_\Delta(\mathcal{Q}') \geq 1 + \#_\Delta(\mathsf{subscribers}_\Delta(a)) + \#_\Delta(\mathcal{Q}')$ from (10)
(12) $\#^{\mathsf{max}}_\Delta + \#_\Delta(\mathcal{Q}') \geq 1 + \#_\Delta(\mathsf{subscribers}_\Delta(a); \mathcal{Q}')$ from (11)
(13) $1 + \#^{\mathsf{max}}_\Delta + \#_\Delta(\mathcal{Q}') \geq 1 + \#_\Delta(\mathsf{subscribers}_\Delta(a); \mathcal{Q}')$ from (12)
(14) $\#_\Delta(a; \mathcal{Q}') = 1 + \#_\Delta(\mathsf{subscribers}_\Delta(a); \mathcal{Q}')$ by Definition A.5
⋆ $\#_\Delta(\mathbf{do\ action}\ \{a \leftarrow e\}; \mathcal{Q}') \geq \#_\Delta(a; \mathcal{Q}')$ from (4, 13, 14)

2. S-Do

(H1) $\vdash \Delta; \mathcal{S}; (\mathbf{do}\ e; \mathcal{Q}')$
(H2) $\Delta; \mathcal{S}; (\mathbf{do}\ e; \mathcal{Q}') \to \Delta; \mathcal{S}; (\mathbf{do}\ e'; \mathcal{Q}')$
(3) $\#_\Delta(\mathbf{do}\ e; \mathcal{Q}') = 1 + \#^{\mathsf{max}}_\Delta + \#_\Delta(\mathcal{Q}')$ by Definition A.5
(4) $\#_\Delta(\mathbf{do}\ e'; \mathcal{Q}') = 1 + \#^{\mathsf{max}}_\Delta + \#_\Delta(\mathcal{Q}')$ by Definition A.5
⋆ $\#_\Delta(\mathbf{do}\ e; \mathcal{Q}') = \#_\Delta(\mathbf{do}\ e'; \mathcal{Q}')$ from (3, 4)

3. S-Queue

(H1) $\vdash \Delta; \mathcal{S}; (a; \mathcal{Q}')$
(H2) $\Delta; \mathcal{S}; (a; \mathcal{Q}') \to \Delta; \mathcal{S}; (a\langle e\rangle; \mathcal{Q}')$
⋆ $\#_\Delta(a\langle e\rangle; \mathcal{Q}') = \#_\Delta(a; \mathcal{Q}')$ by Definition A.5

4. S-Step

(H1) $\vdash \Delta; \mathcal{S}; (a\langle e\rangle; \mathcal{Q}')$
(H2) $\Delta; \mathcal{S}; (a\langle e\rangle; \mathcal{Q}') \to \Delta; \mathcal{S}; (a\langle e'\rangle; \mathcal{Q}')$
(3) $\#_\Delta(a\langle e\rangle; \mathcal{Q}') = \#_\Delta(a; \mathcal{Q}')$ by Definition A.5
(4) $\#_\Delta(a\langle e'\rangle; \mathcal{Q}') = \#_\Delta(a; \mathcal{Q}')$ by Definition A.5
⋆ $\#_\Delta(a\langle e\rangle; \mathcal{Q}') = \#_\Delta(a\langle e'\rangle; \mathcal{Q}')$ from (3, 4)

5. S-Value

(H1) $\vdash \Delta; \mathcal{S}[a \mapsto (e, v')]; (a\langle e\rangle; \mathcal{Q}')$
(H2) $\Delta; \mathcal{S}[a \mapsto (e, v')]; (a\langle v\rangle; \mathcal{Q}') \to \Delta; \mathcal{S}[a \mapsto (e, v)]; (s; \mathcal{Q}')$
$\#_\Delta(a\langle v\rangle; \mathcal{Q}') = \#_\Delta(a; \mathcal{Q}')$ by Definition A.5
(3) $\#_\Delta(a; \mathcal{Q}') = 1 + \#_\Delta(\mathsf{subscribers}_\Delta(a); \mathcal{Q}')$ by Definition A.5
$s = \mathsf{subscribers}_\Delta(a)$ by inv. of S-Value in (H2)
(4) $\#_\Delta(s; \mathcal{Q}') = \#_\Delta(\mathsf{subscribers}_\Delta(a); \mathcal{Q}')$
(5) $1 + \#_\Delta(\mathsf{subscribers}_\Delta(a); \mathcal{Q}') \geq \#_\Delta(\mathsf{subscribers}_\Delta(a); \mathcal{Q}')$
⋆ $\#_\Delta(a\langle v\rangle; \mathcal{Q}') \geq \#_\Delta(s; \mathcal{Q}')$ from (3, 4, 5)

6. S-REmpty

(H1) $\vdash \Delta; \mathcal{S}; ([\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}')$
(H2) $\Delta; \mathcal{S}; ([\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') \to \Delta \uplus \Delta'; \mathcal{S} \uplus \mathcal{S}'; (s; \mathcal{Q}')$
(3) $\#_\Delta([\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') = 1 + \#_{\Delta \uplus \Delta'}(\mathsf{subscribers}_{\Delta \uplus \Delta'}(\mathsf{dom}(\Delta')); \mathcal{Q}')$ by Definition A.5
(4) $s = \mathsf{subscribers}_{\Delta \uplus \Delta'}(\mathsf{dom}(\Delta'))$ by inv. S-REmpty in (H2)
(5) $\#_\Delta([\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') = 1 + \#_{\Delta \uplus \Delta'}(s; \mathcal{Q}')$ from (3, 4)
⋆ $\#_\Delta([\varepsilon]^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}') \geq \#_{\Delta \uplus \Delta'}(s; \mathcal{Q}')$ from (5)

7. S-RVar

(H1) $\vdash \Delta; \mathcal{S}; ([\mathbf{var}\ a{=}e; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$

(H2) $\Delta; \mathcal{S}; ([\mathbf{var}\ a{=}e; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'') \to \Delta; \mathcal{S}; ([a; q']^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q}'')$
$\Delta \uplus \Delta' \vdash \mathbf{var}\ a{=}e : \Delta''$ by inv. S-RVAR in (H2)
(3) $\#_\Delta([\mathbf{var}\ a{=}e; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'') = 1 + \#_\Delta([a; q']^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q}'')$ by Definition A.5
(4) $\Delta \vdash [\mathbf{var}\ a{=}e]^{\Delta'}_{\mathcal{S}'} \dashv \Delta' \uplus \Delta''$ by Definition A.5
$\Delta \uplus \Delta' \vdash \mathbf{var}\ a{=}e \dashv \Delta''$ by Definition 3.12 in (4)
⋆ $\#_\Delta([\mathbf{var}\ a{=}e; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'') \geq \#_\Delta([a; q']^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q}'')$ from (3)

8. S-RDEF
(H1) $\vdash \Delta; \mathcal{S}; ([\mathbf{def}\ a{=}e; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$
(H2) $\Delta; \mathcal{S}; ([\mathbf{def}\ a{=}e; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'') \to \Delta; \mathcal{S}; ([a; q']^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q}'')$
$\Delta \uplus \Delta' \vdash \mathbf{def}\ a{=}e : \Delta''$ by inv. S-RDEF in (H2)
(3) $\#_\Delta([\mathbf{def}\ a{=}e; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'') = 1 + \#_\Delta([a; q']^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q}'')$ by Definition A.5
(4) $\Delta \vdash [\mathbf{def}\ a{=}e]^{\Delta'}_{\mathcal{S}'} \dashv \Delta' \uplus \Delta''$ by Definition A.5
$\Delta \uplus \Delta' \vdash \mathbf{def}\ a{=}e \dashv \Delta''$ by Definition 3.12 in (4)
⋆ $\#_\Delta([\mathbf{def}\ a{=}e; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'') \geq \#_\Delta([a; q']^{\Delta' \uplus \Delta''}_{\mathcal{S}'[a \mapsto (e, \square)]}; \mathcal{Q}'')$ from (3)

9. S-RQUEUE
(H1) $\vdash \Delta; \mathcal{S}; ([a; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$
(H2) $\Delta; \mathcal{S}; ([a; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'') \to \Delta; \mathcal{S}; ([a\langle e\rangle; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$
$\#_\Delta([a; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'') = 1 + \#_\Delta([q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$ by Definition A.5
⋆ $\#_\Delta([a\langle e\rangle; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'') = \#_\Delta([a; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$ by Definition A.5

10. S-RSTEP
(H1) $\vdash \Delta; \mathcal{S}; ([a\langle e\rangle; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$
(H2) $\Delta; \mathcal{S}; ([a\langle e\rangle; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'') \to \Delta; \mathcal{S}; ([a\langle e'\rangle; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$
(3) $\#_\Delta([a\langle e\rangle; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'') = \#_\Delta([a; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$ by Definition A.5
(4) $\#_\Delta([a\langle e'\rangle; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'') = \#_\Delta([a; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$ by Definition A.5
⋆ $\#_\Delta([a\langle e\rangle; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'') = \#_\Delta([a\langle e'\rangle; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$ form (3, 4)

11. S-RVALUE
(H1) $\vdash \Delta; \mathcal{S}; ([a\langle v\rangle; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$
(H2) $\Delta; \mathcal{S}; ([a\langle e\rangle; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'') \to \Delta; \mathcal{S}; ([q']^{\Delta'}_{\mathcal{S}'[a \mapsto (e, v)]}; \mathcal{Q}'')$
$\#_\Delta([a\langle e\rangle; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'') = 1 + \#_\Delta([a; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$ by Definition A.5
(3) $\#_\Delta([a\langle e\rangle; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'') = 1 + 1 + \#_\Delta([q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'')$ by Definition A.5
(4) $\#_\Delta([q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'') = \#_\Delta([q']^{\Delta'}_{\mathcal{S}'[a \mapsto (e, v)]}; \mathcal{Q}'')$ by Definition A.5
⋆ $\#_\Delta([a\langle e\rangle; q']^{\Delta'}_{\mathcal{S}'}; \mathcal{Q}'') \geq \#_\Delta([q']^{\Delta'}_{\mathcal{S}'[a \mapsto (e, v)]}; \mathcal{Q}'')$ from (3, 4)
□