Alma Mater Studiorum · Università di Bologna

Dipartimento di Informatica — Scienza e Ingegneria
Corso di Laurea Triennale in Informatica

Formalisation and Static Analysis for the Serverless Choreographic Programming Language FaaSChal

Relatore:
Chiar.mo Prof.
Saverio Giallorenzo

Presentata da:
Mauro Vergnani

IV Sessione 03 2026
Anno Accademico 2025/2026

Acknowledgements

I would like to extend my deepest gratitude to my supervisor, Professor Saverio Giallorenzo, for his invaluable advice and his availability, and for helping me find a thesis topic that I found both incredibly interesting and satisfying to work on. I would also like to thank my parents, for supporting me throughout the years, both morally and financially. Finally, I would like to thank all of my friends, and especially Riccardo, David, and Edo, for helping me get through times of stress, and for always being great company.

Sommario

Il calcolo moderno si basa più che mai su due aspetti: la concorrenza, cioè l'esecuzione di più compiti in contemporanea; e la distribuzione, cioè l'utilizzo di componenti che si trovano su dispositivi comunicanti separati. Questo è dovuto all'importanza crescente del World Wide Web, di Internet of Things, e delle telecomunicazioni, ed al maggiore utilizzo di edge computing e cloud computing, che portano alla crescita delle reti di dispositivi.

In particolare, il cloud computing è uno degli ambienti principali dove vengono eseguiti sistemi distribuiti, dato che consente maggiore scalabilità e una gestione delle risorse più semplice. Uno degli approcci di design più utilizzati con il cloud computing è il serverless computing, che consiste nel definire funzioni con inneschi specifici, lasciando che sia il cloud provider a istanziarle quando vengono innescate.

Per queste ragioni è stato proposto un nuovo linguaggio di programmazione, chiamato FaaSChal, con l'obiettivo di rendere più semplice specificare il comportamento di sistemi serverless, e di garantire l'assenza di alcuni tipi di errori di comunicazione. FaaSChal è un linguaggio coreografico, che significa che il comportamento dell'intero sistema è specificato in un singolo programma.

In questa tesi formalizziamo un calcolo coreografico, chiamato FaaSChal-Core, che contiene tutti gli aspetti chiave di FaaSChal, e definiamo una disciplina di analisi statica che, separando i programmi corretti da quelli errati, ci permette di dimostrare proprietà importanti sui programmi corretti, fra cui garantire che le funzioni serverless terminino correttamente prima che il sistema reggiunga la propria conclusione.

Contents

Chapter 1

Introduction

Modern computing relies more than ever on two aspects: concurrency, the performance of multiple tasks at the same time; and distribution, the usage of components located on different communicating devices. This is due to the rising importance of the World Wide Web, the Internet of Things, and telecommunications, and due to the rising popularity of edge computing and cloud computing, all of which lead to larger computer networks.

Cloud computing in particular is one of the main environments where distributed systems run, since it allows for greater scalability and easier management of resources. One of the main design approaches used with cloud computing is serverless computing, which consists of defining functions with specific triggers, letting the cloud provider handle their instantiation when they are triggered.

For these reasons, a new programming language, called FaaSChal[1], was proposed, with the aim of making it easier for programmers to specify the behaviour of serverless systems, and of guaranteeing freedom from certain types of communication errors. FaaSChal is a choreographic language, meaning that the behaviour of all processes that participate in the system is specified in a single program.

In this thesis we formalize a choreographic calculus, called FaaSChalCore, which contains all the key features of FaaSChal, and we define a static ana-

lysis discipline that, by separating correct and incorrect programs, allows us to prove important properties for the language on the correct programs, like guaranteeing that serverless functions terminate properly before the system ends execution.

In Chapter 2 we provide background information important to understand the thesis work. First we describe Choreographic Programming, then serverless computing, and finally we go into more detail regarding FaaSChal and its differences with FaaSChalCore. In Chapter 3 we formalize the core calculus for FaaSChalCore, by first providing the syntax and then the semantics of the language. In Chapter 4 we formalize the static analysis discipline, and provide formal definitions of correct (well-formed) programs. Finally, in Chapter 5, we prove some properties that emerge from the formal definitions in Chapters 3 and 4.

Chapter 2

Background

This Chapter provides a brief background on the essential concepts necessary to understand the thesis work.

2.1 Choreographic Programming

Choreographic Programming is a language paradigm where programs, called choreographies, specify the behaviour of communicating participants, called processes or roles.

2.1.1 Choreographies and Choreographic Languages

A choreography is a piece of software that specifies the behaviour of participants in a distributed concurrent system, and how they interact with each other, without requiring a central entity to coordinate them[4].

The languages used for describing choreographies are called choreographic languages, and feature specific primitives that represent interactions and communications between roles. These primitives usually take the form $\mathtt{p}.m \rightarrow \mathtt{q}.x$, denoting that $\mathtt{p}$ sends a message m to $\mathtt{q}$, which will save it in variable x.

When a choreographic language is expressive enough to allow the detailed definition of an executable distributed program, we say that it is a choreographic programming language.

Choreographic programming allows for a correctness-by-construction approach in the definition of concurrent and distributed systems, since the interactions between roles can be regulated through the primitives of the language and through static analysis.

2.1.2 Endpoint Projection

Most choreographic programming languages also define Endpoint Projection (EPP), a process which takes a choreography and outputs the corresponding code that each role, or endpoint, must execute to follow the specified behaviour[4].

EPP is essential when creating a compiler for a choreographic language, since it acts as a formal bridge between the original choreography and the behaviours of the individual participants, which creates a system that preserves the properties of the original language.

Although we do not define Endpoint Projection for FaaSChalCore, it is important to account for it when designing the language and its static analysis discipline. The most important property to account for is knowledge of choice.

2.1.3 Knowledge of Choice

We say that a choreography has the knowledge of choice property if, whenever a choice is made between alternative behaviours, all affected roles are made aware of that choice[4].

Since the guard of a conditional is evaluated by a single role, a choreographic language should define a primitive that allows roles to transmit knowledge of choice to other roles. This primitive is usually called a selection, and takes the form $\texttt{p} \rightarrow \texttt{q[L]}$, denoting that `p` sends the constant value `L` to `q`, who uses it to choose between different behaviours.

We provide an example of a choreography that does not have the knowledge of choice property, and an equivalent version that has the property, in

```
// Choreography without knowledge of choice
if p.coin_flip() then
    q.0 -> r.x;
else
    q.1 -> r.x;
// Choreography that has knowledge of choice
if p.coin_flip() then
    p -> q[OK];
    q.0 -> r.x;
else
    p -> q[KO];
    q.1 -> r.x;
```

Figure 2.1: Example of a choreography without knowledge of choice, and a possible correction.

Figure 2.1. In the first choreography, the behaviour of `q` depends on the evaluation made at role `p`, but `q` is never informed of the result. In the second, we add selections from `p` to `q` to propagate knowledge of choice. There is no need for a selection to `r` because its behaviour remains the same in both branches, i.e., it receives a message from `q` and saves it in `x`.

2.2 Serverless Computing and Architectures

Serverless is a cloud-native development model that allows developers to build and run applications without having to manage servers. This does not mean that there are no servers, just that the servers are abstracted away from application development, and the responsibility of managing their resources falls on their provider[3].

Serverless computing and serverless architecture are often used interchangeably, but they are different concepts. The former refers to the application development model, while the latter refers to the design approach of an application.

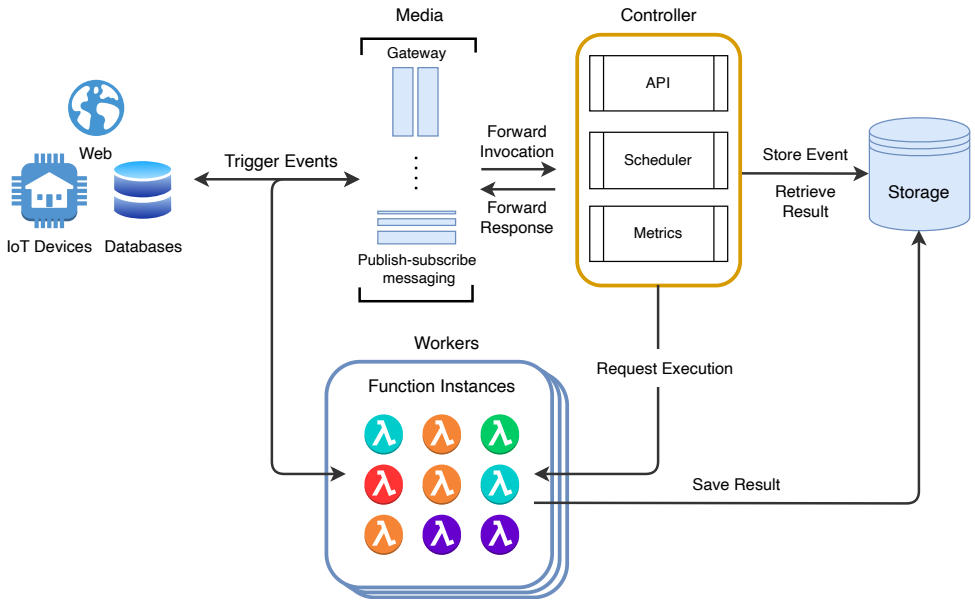

Figure 2.2: A typical serverless platform architecture.

There are various types of serverless computing, but the thesis work only requires understanding of Function-as-a-Service (FaaS).

2.2.1 Function-as-a-Service

When using Function-as-a-Service, developers make a serverless application out of software units, called functions, that run in short-lived containers, triggered by various kinds of events and managed by the cloud services provider. When a specified event occurs, the FaaS platform runs an instance of the function(s) linked to that event, each within a secure and isolated container. Each function instance is stateless, meaning that it has no memory of previous interactions, which simplifies data integration.

A typical FaaS architecture is shown in Figure 2.2, which we use to explain the execution of serverless functions. The main components of a FaaS platform are the Controller and the Workers. The Controller can receive requests to execute functions from various media, e.g., an HTTP gateway

or a publish-subscribe messaging service like the Simple Notification Service by AWS. These media expose endpoints which can be triggered by other entities, including running serverless functions, to invoke the execution of a serverless function. The Controller allocates the functions on Workers based on the latter's status, and handles both the storage of the invocation from the caller, and the retrieval of the result to the caller.

2.3 FaaSChal

FaaSChal is a proposed choreographic programming language, tailored for serverless Function-as-a-Service[1], for which we provide a formal core in this thesis. We provide an example of some of the relevant changes between the two languages in Figure 2.3.

FaaSChal features a distinction between stateful roles, which represent the standard roles which appear in all choreographic languages, and stateless roles, which represent serverless Workers executing a serverless function instance. In the original proposal, stateless role names are each paired with a serverless medium through which they are triggered, but in our version this coupling is removed, with the serverless media being declared separately.

It also features a primitive for creating stateless roles during execution and defining their behaviour, which we split into two separate primitives, one to create stateless roles, and one to conclude them and send their response.

We also remove iteration in favor of procedure calls, as it simplifies the definition of the semantics and allows for more effective static analysis, and we remove the concept of external services, because it is not as important to achieve the objective of the language.

```
// FaaSChal program
stateful: user
stateless: f{ Gateway }

def main(queries@user)
  queries@user <-Gateway-> f do | queries@f |
    results@f = handle(queries)@f;
  end with results@f
end

// FaaSChalCore program
media: Gateway;

def main(){
  queries@user = get_queries()@user;
  queries@user <-Gateway-> queries@f |> results@user;
  results@f = handle(queries)@f;
  end results@f -> user;
}
```

Figure 2.3: An example of a FaaSChal program, and an equivalent FaaSChal-Core program.

Chapter 3

Formalisation of FaaSChalCore

In this Chapter we define the fundamental structure of FaaSChalCore, extending the language defined in Chapter 7 of *Introduction to Choreographies*[4]. We first provide the syntax of the language, containing the most fundamental features of FaaSChal. Then, we define the semantics of the language, taking into consideration potential issues that will be handled by the static analysis discipline described in Chapter 4.

Before defining the language, we define a few basic sets which the rest of the definitions will rely upon:

- Let **Var** be the set of variable names, ranged over by `x`, `y`, `z`, representing variable identifiers.
- Let **Val** be the set of values, ranged over by v, u.
- Let **Media** be the set of serverless media, ranged over by `M`, `N`.
- Let **StatefulRoles** be the set of role names, ranged over by `p`, `q`, `r`, used for standard choreographic roles.
- Let **StatelessRoles** be the set of role names, ranged over by `f`, `g`, `h`, used for serverless function instances.
- Let **Roles**= **StatefulRoles** $\cup$ **StatelessRoles**, ranged over by `n`, `m`, `l`, used for any role. The distinction between stateless and stateful role

names is purely formal, and is made to simplify the static analysis of the language. When implenting the static analysis discipline, one only needs to check how the names are used the first time they appear, and that they are used consistently throughout the rest of the program.

3.1 Syntax

A program in FaaSChalCore begins with a declaration of the serverless media that can be used, as a simple list. This part is left intentionally vague since it is not particularly relevant to the rest of the language; we will assume that we can extrapolate a set of serverless media, $\mathscr{M} \subseteq \mathbf{Media}$, from this declaration.

The media declaration is followed by a series of procedure declarations, the syntax of which is explained last, for the sake of clarity. From these declarations, we can extrapolate a set of procedure definitions, $\mathscr{C}$, which will be explained in more detail with the syntax for declarations.

Finally, the main choreography is defined using the following syntax.

S	::=	`def main(){`C`}`	
C	::=	I; C	instruction sequencing
	|	**0**	teminated choreography, usually omitted when clear from context
I	::=	e`@p -> x@q`	`p` sends the evaluation of e to `q`, which assigns it to `x`
	|	`p -> q[L]`	`p` sends the constant `L` to `q`
	|	`x@n =` e`@n`	`n` assigns the evaluation of e to `x`
	|	e`@n -M-> x@f`	`n` sends a request to the server through the medium `M`, which creates a new stateless role `f`, which assigns the evaluation of e to `x`
	|	e`@n <-M-> x@f |> y@n`	`n` sends a request-response to the server, acting like the request instruction, additionally noting that `f` will respond through the medium `M` to `n`, which will assign the response from `f` to `y`
	|	`end f`	`f` terminates, without a response
	|	`end` e`@f->n`	`f` sends, through the noted `M`, the evaluation of e to `n`, which assigns it to the noted `y`. Then, `f` terminates
	|	`if` e`@n then` $\{C_1\}$ `else` $\{C_2\}$	`n` evaluates e and the choreography continues as C_1 or C_2 accordingly
	|	`X(`F`, nonterm` N`, term` T`)`	call procedure `X` with stateful parameters F, stateful and nonterminating stateless parameters N, and terminating stateless parameters T, associated with the roles they must respond to
	|	N_P`, term` T_P`:` `X(`F`, nonterm` N`, term` T`).` C'	(runtime term, the roles in N_P and T_P have yet to enter the call)

F	::=	$\texttt{p}, F \mid \epsilon$	list of stateful roles
N	::=	$\texttt{n}, N \mid \epsilon$	list of roles
T	::=	$\texttt{f}, T \mid \texttt{[f, n]}, T \mid \epsilon$	list of stateless roles, where `[f, n]` means that `f` was created by `n` expecting a response, and `f` alone means that `f` was created without expecting a response
e	::=	$v \mid \texttt{x} \mid f(e_1, \ldots, e_n)$	values, variables, and local function calls

Let **FaaSChalCore** be the set of all terms which can be derived from C, i.e., the language which contains main choreographies in FaaSChalCore programs.

In the remainder, we often treat F, N, and T as ordered sets, and apply to them operations that would normally apply to an ordered set. In particular, T requires some additional explanations. When we treat T as an ordered set, we consider it to be a set of ordered pairs (`f, s`), where `s` can be a generic role `n`, or the empty role `0`. The first element `f` can be `0` in T_P, which appears only in runtime terms, but only if the second element is not `0` as well. When (`0, n`) appears in T_P, the corresponding list representation is `[0, n]`.

We now define the syntax for procedure declarations, where the term derived from $S_{\texttt{X}}$ gives all the necessary information to add procedure `X` to the set of proceedure definitions, and where $F_{\texttt{X}}$, $N_{\texttt{X}}$, $T_{\texttt{X}}$, and $C_{\texttt{X}}$ are defined in the same way as the main choreography:

$$S_{\texttt{X}} ::= \texttt{def X(}F_{\texttt{X}}\texttt{, nonterm }N_{\texttt{X}}\texttt{, term }T_{\texttt{X}}\texttt{):(}O_{\texttt{X}}\texttt{)\{}C_{\texttt{X}}\texttt{\}}$$
$$O_{\texttt{X}} ::= \texttt{f <: g, }O_{\texttt{X}} \mid \epsilon$$

We additionally impose that for each `f` $<:$ `g` in $O_{\texttt{X}}$, `f` and `g` must appear as the first elements of some pairs in $T_{\texttt{X}}$, and must be different stateless roles.

$O_{\texttt{X}}$ is used together with $T_{\texttt{X}}$ to define a strict partial order relation between the terminating parameters, $<:_{\texttt{X}}$, which we call the termination order for `X`, and is used exclusively during static analysis for procedure calls.

```
def X(term [a, b], c){
    end c;
    end x@a -> b;
}
// We declare that a must terminate before c in Y
def Y(term [a, b], c):(a<:c){
    end x@a -> b;
    end c;
}
def main(){
  0@p -M-> x@f;
  x@f -M-> x@g |> y@f;
  // Incorrect call, f terminates before receiving
  // an answer from g
  X(term [g, f], f);
  // Calling Y instead, using the same parameters,
  // is correct, since we know that a terminates
  // before c in Y
}
```

Figure 3.1: Correct and incorrect use of parameters, based on procedure declaration.

As such, although formally the elements of $\mathscr{C}$ have the form

$$(\texttt{X(}F_{\texttt{X}}\texttt{, nonterm }N_{\texttt{X}}\texttt{, term }T_{\texttt{X}}\texttt{)} = C_{\texttt{X}}, <:_{\texttt{X}})$$

in this Chapter we implicitly drop the second element of each pair in $\mathscr{C}$, since we only need the association between parameters and the choreographic body of the procedure. The details of how $<:_{\texttt{X}}$ is obtained from $O_{\texttt{X}}$ and $T_{\texttt{X}}$ are covered formally in Chapter 4.

When writing the declaration for a procedure, the programmer should include all roles that appear in its body without being created there. If one of these roles uses communications or selections it should be listed in $F_{\texttt{X}}$. If it is terminated inside the procedure's body, it should be listed in $T_{\texttt{X}}$, alone if it does not send a response, and paired with its creator if it does. If neither

of these conditions apply, and the role appears in any instruction that is not `end` e`@f -> n`, then it should be listed in $N_{\mathtt{x}}$.

The programmer can specify the behaviour of the creator of one or more stateless roles by writing the creator in the appropriate terminating pairs and in either $F_{\mathtt{x}}$ or $N_{\mathtt{x}}$, as appropriate.

The programmer can also use $O_{\mathtt{x}}$ to specify that some stateless roles will terminate before others in the procedure's body, which allows more flexible use of the procedure, by allowing us to instantiate roles that would otherwise be unrelated as the creators of other parameters. We show an example of this in Figure 3.1.

These guidelines emerge from the static analysis discipline defined in Chapter 4.

3.2 Semantics

We define the behaviour of the choreography through Structured Operational Semantics, by providing a corresponding Labelled Transition System (LTS). First, we introduce a few concepts that will be used in the definition of the LTS.

3.2.1 Process Store and Choreographic Store

A process store σ is a function that represents the memory of a process, by mapping variable names to values. We write **PStore** for the set of all process stores.

$$\sigma : \mathbf{Var} \to \mathbf{Val}$$

We then define a notation for updating a process store, $\sigma[\mathtt{y} \to v]$, as follows:

$$\sigma[\mathtt{y} \mapsto v](\mathtt{x}) = \begin{cases} v & \text{if } \mathtt{x} = \mathtt{y} \\ \sigma(\mathtt{x}) & \text{otherwise} \end{cases}$$

A choreographic store Σ is a function that represents the memory of an entire running system, by mapping roles to process stores.

$$\Sigma : \textbf{Roles} \to \textbf{PStore}$$

We say that the stateless roles within the domain of a choreographic store are defined in that store; this is used in the LTS to determine if a given role name is currently being used. To allow for the creation of new stateless roles and to change the values of variables within roles, we define a notation for updating a choreographic store:

$$\Sigma[\mathtt{y@m} \mapsto v](\mathtt{n}) = \begin{cases} \Sigma(\mathtt{n})[\mathtt{y} \mapsto v] & \text{if } \mathtt{n} = \mathtt{m} \\ \Sigma(\mathtt{n}) & \text{otherwise} \end{cases}$$

We also define a notation for removing a stateless role from a choreographic store, to allow for the removal of a stateless role from the store when it terminates; $\Sigma \backslash \mathtt{f}$ behaves in the same way as Σ, but $\mathtt{f}$ is not in its domain.

Finally we introduce a notation to refer to the evaluation of an expression in a store: $\sigma \vdash e \downarrow v$ means that the process store σ evaluates the expression e to the value $v \in \textbf{Val}$, and consequently $\Sigma(\mathtt{n}) \vdash e \downarrow v$ means that the choreographic store Σ evaluates the expression e to the value $v \in \textbf{Val}$ in role $\mathtt{n}$. If $\mathtt{n} \notin \mathrm{dom}(\Sigma)$, we say that $\Sigma(\mathtt{n}) \vdash e \downarrow \mathtt{()}$, where $\mathtt{()}$ is the unit value, used as a placeholder for some value that does not carry any information, similar to `null` or `undefined` in other programming languages.

$$\frac{}{\sigma \vdash v \downarrow v}\ \text{VAL} \qquad \frac{}{\sigma \vdash \mathtt{x} \downarrow \sigma(\mathtt{x})}\ \text{VAR} \qquad \frac{\sigma \vdash e_1 \downarrow v_1 \cdots \sigma \vdash e_n \downarrow v_n \qquad \vdash f(v_1, \ldots, v_n) \downarrow v}{\sigma \vdash f(e_1, \ldots, e_n) \downarrow v}\ \text{CALL}$$

We assume that there is some system to derive expressions of the form $\vdash f(v_1, \ldots, v_n) \downarrow v$, which indicate that calling the local function f with arguments $v_1, \ldots, v_n$ evaluates to v, for all $v_1, \ldots, v_n$, without defining such a system.

3.2.2 Response Store

A response store Ψ is a function that stores information for the serverless roles in a choreography about which role they should send a response to, where this role should save the response, and which serverless medium should be used for the communication. Not all serverless roles are covered by a response store, because some roles are created without expecting a response, so they do not need to store this information.

$$\Psi : \textbf{StatelessRoles} \to \textbf{Roles} \times \textbf{Var} \times \textbf{Media}$$

We say that the stateless roles within the domain of a response store must respond according to that store, and that the stateless roles outside of the domain must not respond; as with choreographic stores, this is used in the LTS to determine the details for the execution of termination instructions. To handle the creation of new roles and their termination, we define a notation for updating response stores:

$$\Psi[\mathtt{g} \overset{\mathtt{M}}{\mapsto} \mathtt{y@n}](\mathtt{f}) = \begin{cases} \langle \mathtt{n}, \mathtt{y}, \mathtt{M} \rangle & \text{if } \mathtt{f} = \mathtt{g} \\ \Psi(\mathtt{f}) & \text{otherwise} \end{cases}$$

We introduce a more intuitive notation for $\Psi(\mathtt{f}) = \langle \mathtt{n}, \mathtt{y}, \mathtt{M} \rangle$, which denotes that role `f` must respond to role `n`, which will save the result in variable `y`, using medium `M`. From now on, this will be denoted by $\Psi(\mathtt{f}) \overset{\mathtt{M}}{\mapsto} \mathtt{y@n}$.

We write $\Psi \backslash \mathtt{f}$ to denote the response store which behaves like Ψ, but does not have `f` in its domain.

It should be noted that, since the role to which a stateless role should respond does not change during its execution, and the variable and medium used do not change either, the response store should only ever be updated by adding or removing roles from its domain, and never by changing the mapping of a role that is already within its domain.

3.2.3 Process Names

We want to refer to the roles which appear in an instruction or a choreography, and to the roles which appear in a transition label in the LTS. In order to do so, we define a function, pn, which maps instructions, choreographies, lists of parameters, and transition labels to the role names that appear within them. To be more precise, we define various functions with separate domains that share the same name, distinguishing them based on their parameters.

- pn for choreographies

$$\mathrm{pn}(\mathbf{0}) \triangleq \varnothing \qquad \mathrm{pn}(I;\ C) \triangleq \mathrm{pn}(I) \cup \mathrm{pn}(C)$$

- pn for instructions

$$\mathrm{pn}(e\texttt{@p -> x@q}) \triangleq \{\texttt{p}, \texttt{q}\} \qquad \mathrm{pn}(\texttt{p -> q[L]}) \triangleq \{\texttt{p}, \texttt{q}\}$$
$$\mathrm{pn}(\texttt{x@n =}\ e\texttt{@n}) \triangleq \{\texttt{n}\}$$
$$\mathrm{pn}(e\texttt{@n -M-> x@f}) \triangleq \{\texttt{n}, \texttt{f}\} \qquad \mathrm{pn}(e\texttt{@n <-M-> x@f |> y@n}) \triangleq \{\texttt{n}, \texttt{f}\}$$
$$\mathrm{pn}(\texttt{end f}) \triangleq \{\texttt{f}\} \qquad \mathrm{pn}(\texttt{end}\ e\texttt{@f -> n}) \triangleq \{\texttt{f}, \texttt{n}\}$$
$$\mathrm{pn}(\texttt{if}\ e\texttt{@n then}\ \{C_1\}\ \texttt{else}\ \{C_2\}) \triangleq \{n\} \cup \mathrm{pn}(C_1) \cup \mathrm{pn}(C_2)$$
$$\mathrm{pn}(\texttt{X(}F\texttt{, nonterm}\ N\texttt{, term}\ T\texttt{)}) \triangleq \mathrm{pn}(F) \cup \mathrm{pn}(N) \cup \mathrm{pn}(T)$$
$$\mathrm{pn}(N_P\texttt{, term}\ T_P\texttt{:X(}F\texttt{, nonterm}\ N\texttt{, term}\ T\texttt{)}) \triangleq \mathrm{pn}(N_P) \cup \mathrm{pn}(T_P)$$

- pn for transition labels

$$\mathrm{pn}(\tau\texttt{@n}) \triangleq \{\texttt{n}\} \qquad \mathrm{pn}(v\texttt{@p} \to \texttt{q}) \triangleq \{\texttt{p}, \texttt{q}\} \qquad \mathrm{pn}(\texttt{p} \to \texttt{q[L]}) \triangleq \{\texttt{p}, \texttt{q}\}$$
$$\mathrm{pn}(v\texttt{@n} \xrightarrow{\texttt{M}} \texttt{f}) \triangleq \{\texttt{n}, \texttt{f}\} \qquad \mathrm{pn}(v\texttt{@n} \xleftrightarrow{\texttt{M}} \texttt{f}) \triangleq \{\texttt{n}, \texttt{f}\}$$
$$\mathrm{pn}(\texttt{f} \xleftarrow{\texttt{M}} v\texttt{@n}) \triangleq \{\texttt{n}, \texttt{f}\}$$

- pn for parameter lists

$$\mathrm{pn}(\epsilon) \triangleq \varnothing$$
$$\mathrm{pn}(\texttt{p,}F) \triangleq \{\texttt{p}\} \cup \mathrm{pn}(F)$$
$$\mathrm{pn}(\texttt{n,}N) \triangleq \{\texttt{n}\} \cup \mathrm{pn}(N)$$
$$\mathrm{pn}(\texttt{f,}T) \triangleq \{\texttt{f}\} \cup \mathrm{pn}(T) \quad \mathrm{pn}(\texttt{[f, n],}T) \triangleq \{\texttt{f}, \texttt{n}\} \cup \mathrm{pn}(T)$$

3.2.4 Name Substitution

We finally define an operator to replace role names in choreographies and in lists of parameters, before moving on to the definition of the LTS. First we define name substitution for choreographies.

$$\mathbf{0}[\mathtt{l}/\mathtt{m}] \triangleq \mathbf{0} \qquad (I;\ C)[\mathtt{l}/\mathtt{m}] \triangleq I[\mathtt{l}/\mathtt{m}];\ C[\mathtt{l}/\mathtt{m}]$$

$$(e\texttt{@p -> x@q})[\mathtt{l}/\mathtt{m}] \triangleq \begin{cases} e\texttt{@l -> x@q} & \text{if } \mathtt{m} = \mathtt{p} \\ e\texttt{@p -> x@l} & \text{if } \mathtt{m} = \mathtt{q} \\ e\texttt{@p -> x@q} & \text{otherwise} \end{cases}$$

$$(\texttt{p -> q[L]})[\mathtt{l}/\mathtt{m}] \triangleq \begin{cases} \texttt{l -> q[L]} & \text{if } \mathtt{m} = \mathtt{p} \\ \texttt{p -> l[L]} & \text{if } \mathtt{m} = \mathtt{q} \\ \texttt{p -> q[L]} & \text{otherwise} \end{cases}$$

$$(\texttt{x@n = } e\texttt{@n})[\mathtt{l}/\mathtt{m}] \triangleq \begin{cases} \texttt{x@l = } e\texttt{@l} & \text{if } \mathtt{m} = \mathtt{n} \\ \texttt{x@n = } e\texttt{@n} & \text{otherwise} \end{cases}$$

$$(e\texttt{@n -M-> x@f})[\mathtt{l}/\mathtt{m}] \triangleq \begin{cases} e\texttt{@l -M-> x@f} & \text{if } \mathtt{m} = \mathtt{n} \\ e\texttt{@n -M-> x@l} & \text{if } \mathtt{m} = \mathtt{f} \\ e\texttt{@n -M-> x@f} & \text{otherwise} \end{cases}$$

$$(e\texttt{@n <-M-> x@f |> y@n})[\mathtt{l}/\mathtt{m}] \triangleq \begin{cases} e\texttt{@l <-M-> x@f |> y@l} & \text{if } \mathtt{m} = \mathtt{n} \\ e\texttt{@n <-M-> x@l |> y@n} & \text{if } \mathtt{m} = \mathtt{f} \\ e\texttt{@n <-M-> x@f |> y@n} & \text{otherwise} \end{cases}$$

$$(\texttt{end f})[\mathtt{l}/\mathtt{m}] \triangleq \begin{cases} \texttt{end l} & \text{if } \mathtt{m} = \mathtt{f} \\ \texttt{end f} & \text{otherwise} \end{cases}$$

$$(\texttt{end } e\texttt{@f -> n})[\mathtt{l}/\mathtt{m}] \triangleq \begin{cases} \texttt{end } e\texttt{@f -> l} & \text{if } \mathtt{m} = \mathtt{n} \\ \texttt{end } e\texttt{@l -> n} & \text{if } \mathtt{m} = \mathtt{f} \\ \texttt{end } e\texttt{@f -> n} & \text{otherwise} \end{cases}$$

$$(\texttt{if } e\texttt{@n then } \{C_1\} \texttt{ else } \{C_2\})[\mathtt{l}/\mathtt{m}] \triangleq \begin{cases} \texttt{if } e\texttt{@l then } \{C_1[\mathtt{l}/\mathtt{m}]\} \texttt{ else } \{C_2[\mathtt{l}/\mathtt{m}]\} & \text{if } \mathtt{m} = \mathtt{n} \\ \texttt{if } e\texttt{@n then } \{C_1[\mathtt{l}/\mathtt{m}]\} \texttt{ else } \{C_2[\mathtt{l}/\mathtt{m}]\} & \text{otherwise} \end{cases}$$

$$(\texttt{X(}F\texttt{, nonterm }N\texttt{, term }T\texttt{)})[\texttt{l/m}] \triangleq \texttt{X(}F[\texttt{l/m}]\texttt{, nonterm }N[\texttt{l/m}]\texttt{, term }T[\texttt{l/m}]\texttt{)}$$

$$(N_P\texttt{, term }T_P\texttt{:X(}F\texttt{, nonterm }N\texttt{, term }T\texttt{)})[\texttt{l/m}] \triangleq$$
$$N_P[\texttt{l/m}]\texttt{, term }T_P[\texttt{l/m}]\texttt{:X(}F[\texttt{l/m}]\texttt{, nonterm }N[\texttt{l/m}]\texttt{, term }T[\texttt{l/m}]\texttt{)}$$

We now define name substitution for lists of parameters.

$$\epsilon[\texttt{l/m}] \triangleq \epsilon$$

$$(\texttt{p},F)[\texttt{l/m}] \triangleq \begin{cases} \texttt{l},(F[\texttt{l/m}]) & \text{if } \texttt{m} = \texttt{p} \\ \texttt{p},(F[\texttt{l/m}]) & \text{otherwise} \end{cases}$$

$$(\texttt{n},N)[\texttt{l/m}] \triangleq \begin{cases} \texttt{l},(N[\texttt{l/m}]) & \text{if } \texttt{m} = \texttt{n} \\ \texttt{n},(N[\texttt{l/m}]) & \text{otherwise} \end{cases}$$

$$(\texttt{f},T)[\texttt{l/m}] \triangleq \begin{cases} \texttt{l},(T[\texttt{l/m}]) & \text{if } \texttt{m} = \texttt{f} \\ \texttt{f},(T[\texttt{l/m}]) & \text{otherwise} \end{cases}$$

$$(\texttt{[f, n]},T)[\texttt{l/m}] \triangleq \begin{cases} \texttt{[l, n]},(T[\texttt{l/m}]) & \text{if } \texttt{m} = \texttt{f} \\ \texttt{[f, l]},(T[\texttt{l/m}]) & \text{if } \texttt{m} = \texttt{n} \\ \texttt{[f, n]},(T[\texttt{l/m}]) & \text{otherwise} \end{cases}$$

We also introduce a notation to use this operator with pairs of roles and with ordered sets of roles, instead of individual roles, where A represents a choreography or a list of parameters, and, for each i, both s_i and t_i are either a single element or an ordered pair. Since the ordered pairs can contain `0` instead of one of their roles, we place safeguards against it in the definition of this notation. This operator is left-associative.

$$A[(\texttt{g, m})/(\texttt{f, n})] \triangleq \begin{cases} A & \text{if } (\texttt{f} = \texttt{0} \ \vee \ \texttt{g} = \texttt{0}) \wedge \\ & \quad (\texttt{n} = \texttt{0} \ \vee \ \texttt{m} = \texttt{0}) \\ A[\texttt{g/f}] & \text{if } \texttt{n} = \texttt{0} \ \vee \ \texttt{m} = \texttt{0} \\ A[\texttt{m/n}] & \text{if } \texttt{f} = \texttt{0} \ \vee \ \texttt{g} = \texttt{0} \\ A[\texttt{g/f}][\texttt{m/n}] & \text{otherwise} \end{cases}$$

$$A[\langle s_1, s_2, \ldots, s_n\rangle/\langle t_1, t_2, \ldots, t_n\rangle] \triangleq A[s_1/t_1][\langle s_2, \ldots, s_n\rangle/\langle t_2, \ldots, t_n\rangle]$$
$$A[\langle s_1\rangle/\langle t_1\rangle] \triangleq A[s_1/t_1]$$

3.2.5 Labelled Transition System

The LTS for FaaSChalCore's Structured Operational Semantics is defined as follows:

- The set of states is the set of tuples $\langle C, \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle$, which we call choreographic configurations, where C is a choreography in FaaSChalCore, Σ is a choreographic store, Ψ is a response store, $\mathscr{C}$ is a set of procedure definitions, and $\mathscr{M} \subseteq \textbf{Media}$ is a set of serverless media.
- The set of transition labels is the set of all possible labels obtained by instantiating the variable names that appear in the inference system below. As with the syntax, `p` and `q` represent stateful roles, `f` represents a stateless role, and `n` represents either.
- The transition relation is the smallest relation that satisfies the inference rules defined below.

Rule LOC executes a local assignment, by updating the value of `x` in `n` to v, the evaluation of the expression e in `n`.

$$\frac{\Sigma(\texttt{n}) \vdash e \downarrow v}{\langle \texttt{x@g = } e\texttt{@n;} C, \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle \xrightarrow{\tau\texttt{@n}} \langle C, \Sigma[\texttt{x@n} \mapsto v], \Psi, \mathscr{C}, \mathscr{M} \rangle} \text{ LOC}$$

Rule COM executes a communication between stateful roles, by updating the value of `x` in `q` to v, which carries the evaluation of the expression e in `p`.

$$\frac{\Sigma(\texttt{p}) \vdash e \downarrow v}{\langle e\texttt{@p -> x@q;} C, \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle \xrightarrow{v\texttt{@p}\to\texttt{q}} \langle C, \Sigma[\texttt{x@q} \mapsto v], \Psi, \mathscr{C}, \mathscr{M} \rangle} \text{ COM}$$

Rule SEL executes a selection, a simple message which sends a constant, `L`, used to propagate knowledge of choice from one stateful role to another, without affecting the state of the choreography.

$$\frac{}{\langle \texttt{p -> q[L]}\,;C,\Sigma,\Psi,\mathscr{C},\mathscr{M}\rangle \xrightarrow{\texttt{p}\to\texttt{q[L]}} \langle C,\Sigma,\Psi,\mathscr{C},\mathscr{M}\rangle}\ \text{SEL}$$

Rule REQ executes a request for a service without a response, using the serverless medium `M`, which causes the creation of a new stateless role `f`. The choreographic store is updated by adding `f` to its domain, and setting the value of `x` in `f` to v, while the response store is left unchanged, since `f` will not send a response when it terminates.

$$\frac{\texttt{M} \in \mathscr{M} \qquad \Sigma(\texttt{n}) \vdash e \downarrow v}{\langle e\texttt{@n -M-> x@f}\,;C,\Sigma,\Psi,\mathscr{C},\mathscr{M}\rangle \xrightarrow{v\texttt{@n}\overset{\texttt{M}}{\longrightarrow}\texttt{f}} \langle C,\Sigma[\texttt{x@f} \mapsto v],\Psi,\mathscr{C},\mathscr{M}\rangle}\ \text{REQ}$$

Rule REQ-RES executes a request for a service with an expected response, using the serverless medium `M`, which causes the creation of a new stateless role `f`, as in rule REQ. The choreographic store is also updated in the same way as REQ, but the response store is updated to record that `f` must send its response to `y` in `n`, using the serverless medium `M`.

$$\frac{\texttt{M} \in \mathscr{M} \qquad \Sigma(\texttt{n}) \vdash e \downarrow v}{\langle e\texttt{@g <-M-> x@f |> y@n}\,;C,\Sigma,\Psi,\mathscr{C},\mathscr{M}\rangle \xrightarrow{v\texttt{@n}\overset{\texttt{M}}{\longleftrightarrow}\texttt{f}} \langle C,\Sigma[\texttt{x@f} \mapsto v],\Psi[\texttt{f} \overset{\texttt{M}}{\mapsto} \texttt{y@n}],\mathscr{C},\mathscr{M}\rangle}\ \text{REQ-RES}$$

Rule END executes the termination of the stateless role `f` without a response. This is done by simply removing `f` from the choreographic store's domain.

$$\frac{}{\langle \texttt{end f}\,;C,\Sigma,\Psi,\mathscr{C},\mathscr{M}\rangle \xrightarrow{\tau\texttt{@f}} \langle C,\Sigma\backslash\texttt{f},\Psi,\mathscr{C},\mathscr{M}\rangle}\ \text{END}$$

Rule END-RES executes the termination of the stateless role `f` with a response to role `n`, by sending v, the evaluation of the expression e in `f`, through the medium `M` to `n`, which saves this value in `y`; before removing `f` from the domains of the choreographic store and the response store.

$$\frac{\Psi(\texttt{f}) \overset{\texttt{M}}{\mapsto} \texttt{y@n} \qquad \Sigma(\texttt{f}) \vdash e \downarrow v}{\langle \texttt{end } e\texttt{@f -> n;} C, \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle \xrightarrow{\texttt{n} \overset{\texttt{M}}{\leftarrow} v\texttt{@f}} \langle C, \Sigma[\texttt{y@n} \mapsto v] \backslash \texttt{f}, \Psi \backslash \texttt{f}, \mathscr{C}, \mathscr{M}\rangle}\ \text{END-RES}$$

Rules COND-THEN and COND-ELSE evaluate the expression e at role `n`, and take the appropriate branch. The $\fatsemi$ operator concatenates choreographies by removing the additional $\mathbf{0}$ from the first of the concatenated choreographies

$$\frac{\Sigma(\texttt{n}) \vdash e \downarrow \texttt{true}}{\langle \texttt{if } e\texttt{@n then } \{C_1\} \texttt{ else } \{C_2\}; C, \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle \xrightarrow{\tau\texttt{@n}} \langle C_1 \fatsemi C, \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle}\ \text{COND-THEN}$$

$$\frac{\Sigma(\texttt{g}) \vdash e \downarrow v \qquad v \neq \texttt{true}}{\langle \texttt{if } e\texttt{@n then } \{C_1\} \texttt{ else } \{C_2\}; C, \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle \xrightarrow{\tau\texttt{@n}} \langle C_2 \fatsemi C, \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle}\ \text{COND-ELSE}$$

Rule CALL-ONLY executes a procedure call in the case where it has exactly one parameter, meaning that the runtime term for procedure calls is skipped entirely. The first two premises check that there is only one parameter and that $C_{\texttt{X}}$ is the body for procedure `X`, while the remaining premises prevent name capture when expanding the body of the procedure. The first of these premises finds the set of hidden role names for `X`, meaning the roles which appear in the procedure's body, but not in its parameters. These roles are necessarily stateless, so we create a new set of stateless roles, H, which is distinct from all currently existing roles and from the rest of the choreography, as shown by the premise $H \# \mathrm{dom}(\Sigma) \cup \mathrm{pn}(C) \cup \mathrm{pn}(C_{\texttt{X}})$, which is a shorthand for $H \cap (\mathrm{dom}(\Sigma) \cup \mathrm{pn}(C) \cup \mathrm{pn}(C_{\texttt{X}})) = \varnothing$. The transition expands the procedure's body, replacing the original hidden roles in $H_{\texttt{X}}$ with the new roles in H (with a small abuse of notation since these are not ordered sets), and then replacing the formal parameters with the actual parameters. We allow procedures with a single parameter because they can still contain more

than one role, using request instructions, and thus the corresponding behaviour cannot be expressed by local functions.

$$\frac{\begin{array}{c}\mathrm{pn}(F \cup N \cup T) = \{\mathtt{n}\} \\ \mathtt{X(}F_{\mathtt{X}}\mathtt{,\ nonterm\ }N_{\mathtt{X}}\mathtt{,\ term\ }T_{\mathtt{X}}\mathtt{)} = C_{\mathtt{X}} \in \mathscr{C} \\ H_{\mathtt{X}} = \mathrm{pn}(C_{\mathtt{X}}) \backslash \mathrm{pn}(F_{\mathtt{X}} \cup N_{\mathtt{X}} \cup T_{\mathtt{X}}) \\ |H| = |H_{\mathtt{X}}| \qquad H \# \mathrm{dom}(\Sigma) \cup \mathrm{pn}(C) \cup \mathrm{pn}(C_{\mathtt{X}})\end{array}}{\begin{array}{l}\langle \mathtt{X(}F\mathtt{,\ nonterm\ }N\mathtt{,\ term\ }T\mathtt{);}\, C, \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle \xrightarrow{\tau@\mathtt{n}} \\ \langle C_{\mathtt{X}}[H, F, N, T/H_{\mathtt{X}}, F_{\mathtt{X}}, N_{\mathtt{X}}, T_{\mathtt{X}}] \fatsemi C, \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle\end{array}}\ \text{CALL-ONLY}$$

Rule CALL-FIRST executes the initial procedure call in the case where it has more than one parameter, acting similarly to rule CALL-ONLY. The main difference is the presence of the runtime term, where F and N are merged into N_P, since the distinction between stateful and stateless nonterminating roles is not relevant at runtime, and the role entering the procedure call is removed from the list of roles that have yet to enter the call. We abuse the notation when writing $T\backslash\mathtt{n}$, which means that for each $(\mathtt{f,\ n}) \in T$, we replace it with $(\mathtt{f,\ 0})$, for each $(\mathtt{n,\ m}) \in T$, we replace it with $(\mathtt{0,\ m})$, and for each $(\mathtt{n,\ 0}) \in T$ and $(\mathtt{0,\ n}) \in T$, we remove it. The suffix $.C$ at the end of runtime terms is not relevant to the semantics and is added to follow standard practice in the definition of semantics for choreographic calculi[4], to support the definition of Endpoint Projection for the language.

$$\frac{\begin{array}{c}\mathtt{X(}F_{\mathtt{X}}\mathtt{,\ nonterm\ }N_{\mathtt{X}}\mathtt{,\ term\ }T_{\mathtt{X}}\mathtt{)} = C_{\mathtt{X}} \in \mathscr{C} \\ \mathtt{n} \in \mathrm{pn}(F \cup N \cup T) \\ (F \cup N \cup T)\backslash\mathtt{n} \neq \varnothing \qquad H_{\mathtt{X}} = \mathrm{pn}(C_{\mathtt{X}}) \backslash \mathrm{pn}(F_{\mathtt{X}} \cup N_{\mathtt{X}} \cup T_{\mathtt{X}}) \\ |H| = |H_{\mathtt{X}}| \qquad H \# \mathrm{dom}(\Sigma) \cup \mathrm{pn}(C) \cup \mathrm{pn}(C_{\mathtt{X}})\end{array}}{\begin{array}{l}\langle \mathtt{X(}F\mathtt{,\ nonterm\ }N\mathtt{,\ term\ }T\mathtt{);}\, C, \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle \xrightarrow{\tau@\mathtt{n}} \\ \langle (F \cup N)\backslash\mathtt{n}, \mathtt{term\ }T\backslash\mathtt{n:X(}F\mathtt{,\ nonterm\ }N\mathtt{,\ term\ }T\mathtt{)}.C\mathtt{;} \\ \qquad C_{\mathtt{X}}[H, F, N, T/H_{\mathtt{X}}, F_{\mathtt{X}}, N_{\mathtt{X}}, T_{\mathtt{X}}] \fatsemi C, \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle\end{array}}\ \text{CALL-FIRST}$$

Rule CALL-ENTER executes the entrance of a role into the body of a procedure it is a parameter of, but has not yet entered, by simply removing it from the list of roles that have yet to enter.

$$\frac{\mathtt{n} \in \mathrm{pn}(N_P \cup T_P) \qquad (N_P \cup T_P)\backslash\mathtt{n} \neq \varnothing}{\langle N_P, \mathtt{term}\ T_P\mathtt{:X(}F\mathtt{,\ nonterm}\ N\mathtt{,\ term}\ T\mathtt{)}.\, C'; C, \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle \xrightarrow{\tau@\mathtt{n}} \langle N_P\backslash\mathtt{n}, \mathtt{term}\ T_P\backslash\mathtt{n:X(}F\mathtt{,\ nonterm}\ N\mathtt{,\ term}\ T\mathtt{)}.\, C'; C, \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle}\ \text{CALL-ENTER}$$

Rule CALL-LAST executes the entrance of the last parameter into a procedure call, by removing the runtime term.

$$\frac{\mathrm{pn}(N_P \cup T_P) = \{\mathtt{n}\}}{\langle N_P, \mathtt{term}\ T_P\mathtt{:X(}F\mathtt{,\ nonterm}\ N\mathtt{,\ term}\ T\mathtt{)}.\, C'; C, \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle \xrightarrow{\tau@\mathtt{n}} \langle C, \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle}\ \text{CALL-LAST}$$

Rule DELAY allows for out-of-order execution of instructions that do not mention roles that appear in preceding instructions. In other words it allows for interleaved parallel execution.

$$\frac{\langle C, \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle \xrightarrow{\mu} \langle C', \Sigma', \Psi', \mathscr{C}, \mathscr{M}\rangle \qquad \mathrm{pn}(I) \# \mathrm{pn}(\mu)}{\langle I\,; C, \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle \xrightarrow{\mu} \langle I\,; C', \Sigma', \Psi', \mathscr{C}, \mathscr{M}\rangle}\ \text{DELAY}$$

Rule DELAY-COND allows for out-of-order execution of instructions inside the branches of an `if` statement, granted they have the same transition label, which does not include the role in the guard of the conditional, and cause the same effect to the choreographic store and the response store.

$$\frac{\begin{array}{c}\langle C_1, \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle \xrightarrow{\mu} \langle C_1', \Sigma', \Psi', \mathscr{C}, \mathscr{M}\rangle \\ \langle C_2, \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle \xrightarrow{\mu} \langle C_2', \Sigma', \Psi', \mathscr{C}, \mathscr{M}\rangle \qquad \mathtt{n} \notin \mathrm{pn}(\mu)\end{array}}{\begin{array}{l}\langle \mathtt{if}\ e\mathtt{@n\ then}\ \{C_1\}\ \mathtt{else}\ \{C_2\}; C, \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle \xrightarrow{\mu} \\ \quad \langle \mathtt{if}\ e\mathtt{@n\ then}\ \{C_1'\}\ \mathtt{else}\ \{C_2'\}; C, \Sigma', \Psi', \mathscr{C}, \mathscr{M}\rangle\end{array}}\ \text{DELAY-COND}$$

Chapter 4

Static analysis

In this Chapter, we define the static analysis discipline for FaaSChalCore, extending the discipline defined in Chapter 8 of *Introduction to Choreographies*[4]. We formally define the concept of well-formedness of a choreographic body, then we use it to define the well-formedness of a program in FaaSChalCore. We introduce the concept of runtime well-formedness, which is a property maintained by the transitions defined in the semantics of the language. Well-formedness requires that there are no stateless roles present, which means that it is violated as soon as a role creation instruction is executed.

We start by introducing two functions that allow us to handle sets of tuples while maintaining readability. The first is the canonical projection.

$$\begin{aligned} \pi_i : S_1 \times \cdots \times S_i \times \cdots \times S_n &\to S_n \\ (x_1, \ldots, x_i, \ldots, x_n) &\mapsto x_i \end{aligned}$$

We often use this function with a set as an argument, meaning that, given $D \subseteq S_1 \times \cdots \times S_i \times \cdots \times S_n$, we have that

$$\pi_i(D) = \{x_i \mid \exists\, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n.(x_1, \ldots, x_n) \in D\}$$

Given $D \subseteq S_1 \times \cdots \times S_i \times \cdots \times S_n$, we also define the preimage of π_i when restricting the domain to D.

$$\begin{aligned} \pi^{-1}_{D,i} : S_i &\to \mathscr{P}(D) \\ x_i &\mapsto \{(x_1, \ldots, x_i, \ldots, x_n) \in D\} \end{aligned}$$

In both cases we do not specify $S_1, \ldots, S_n$ when they are clear from the context.

4.1 Well-formedness

We introduce the notation $\langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term}, <:, R, C \rangle \checkmark$, which means that C is well-formed in the context of the set of stateless media $\mathscr{M}$ and the set of procedure definitions $\mathscr{C}$, and may include the stateful roles in Ful, the roles in NonT, and the terminating roles in Term, with the termination order given by $<:$ and the set of stateless roles R in its scope.

Then, we define an inference system to derive these statements. We assume to have the following invariants, and we design the inference rules to maintain them for sub-trees:

- $\mathscr{M} \subseteq \mathbf{Media}$
- $\mathit{Ful} \subseteq \mathbf{StatefulRoles}$;
- $\mathit{NonT} \subseteq \mathbf{Roles} \backslash \mathit{Ful}$;
- $\mathit{Term} \subseteq (\mathbf{StatelessRoles} \backslash \mathit{NonT}) \times (\mathbf{Roles} \cup \{\mathtt{0}\})$, where a pair in Term is ranged over by $(\mathtt{f}, \mathtt{s}), (\mathtt{g}, \mathtt{t})$;
- $R \supseteq \mathit{NonT} \cup \pi_1(\mathit{Term})$;
- $\forall (\mathtt{f}, \mathtt{s}) \in \mathit{Term}.\mathtt{f} \neq \mathtt{s}$, which means that a stateless role cannot be its own creator;

- $\pi_2(\mathit{Term}) \subseteq (\mathit{Ful} \cup \mathit{NonT} \cup \pi_1(\mathit{Term}) \cup \{\mathtt{0}\})$, which means that the second element in each terminating pair can either be $\mathtt{0}$, or a role which is mentioned in Ful, NonT, or as the first element in a terminating pair;
- $\forall(\mathtt{f},\mathtt{s}),(\mathtt{g},\mathtt{t}) \in \mathit{Term}.(\mathtt{s} \neq \mathtt{t} \Rightarrow \mathtt{f} \neq \mathtt{g})$, which means that two different terminating pairs in Term must have different first elements;
- $<: \subseteq (\mathbf{StatelessRoles} \backslash \mathit{NonT}) \times \mathbf{Roles}$;
- $<: \supseteq (\mathit{Term} \backslash \pi^{-1}_{\mathit{Term},2}(\mathtt{0}))$, which means that $<:$ must include all terminating pairs that do not have $\mathtt{0}$ as the second element;
- $\pi_1(<:) \subseteq \pi_1(\mathit{Term} \backslash \pi^{-1}_{\mathit{Term},2}(\mathtt{0}))$, which means that for each $(\mathtt{f},\mathtt{n}) \in <:$, there must be some $\mathtt{m} \in \mathbf{Roles}$ such that $(\mathtt{f},\mathtt{m}) \in \mathit{Term}$. Combined with the previous invariant, we also obtain that $\pi_1(<:) = \pi_1(\mathit{Term} \backslash \pi^{-1}_{\mathit{Term},2}(\mathtt{0}))$;
- $\pi_2(<:) \subseteq (\mathit{Ful} \cup \mathit{NonT} \cup \pi_1(\mathit{Term}))$;
- $\forall(\mathtt{f},\mathtt{n}) \in (\mathit{Term} \backslash \pi^{-1}_{\mathit{Term},2}(\mathtt{0})).\ \mathtt{n} \not<: \mathtt{f}$, which means that no role should terminate before its creator;
- $<:$ is a strict partial order.

The inference system is defined as follows.

Rule WF-CLOSE states that the terminated choreography is well-formed when there are no terminating roles, i.e., all roles which had to terminate have actually terminated. We allow nonterminating stateless roles to exist, to allow this inference system to be used in the definition of well-formedness for procedure definitions.

$$\frac{\mathit{Term} = \varnothing}{\langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term}, <:, R, \mathbf{0} \rangle \checkmark}\ \text{WF-CLOSE}$$

Rule WF-COM states that a communication is well-formed if it involves two different stateful roles that can be included in the choreography, and the continuation is well-formed.

$$\frac{\mathtt{p} \neq \mathtt{q} \qquad \mathtt{p}, \mathtt{q} \in \mathit{Ful} \qquad \langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term}, <:, R, C \rangle \checkmark}{\langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term}, <:, R, e\texttt{@p -> x@q;}\, C \rangle \checkmark} \text{ WF-COM}$$

Rule WF-SEL states that a selection is well-formed if it involves two different stateful roles that can be included in the choreography, and the continuation is well-formed.

$$\frac{\mathtt{p} \neq \mathtt{q} \qquad \mathtt{p}, \mathtt{q} \in \mathit{Ful} \qquad \langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term}, <:, R, C \rangle \checkmark}{\langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term}, <:, R, \mathtt{p -> q[L];}\, C \rangle \checkmark} \text{ WF-SEL}$$

Rule WF-LOC states that a local assignment is well-formed if it involves a defined role, and the continuation is well-formed. We use def(n) as a shorthand for $\mathtt{n} \in \mathit{Ful} \cup \mathit{NonT} \cup \pi_1(\mathit{Term})$, meaning that a role is defined if it is a stateful or nonterminating role which the choreography can include, or if it is the first element of a terminating pair.

$$\frac{\text{def}(\mathtt{n}) \qquad \langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term}, <:, R, C \rangle \checkmark}{\langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term}, <:, R, \mathtt{x@n = }\, e\texttt{@n;}\, C \rangle \checkmark} \text{ WF-LOC}$$

Rule WF-REQ states that a request instruction is well-formed if n is defined, f is not in the current scope, M is a valid stateless medium in the current context, and the continuation is well-formed, given the additional nonterminating pair (f, 0).

$$\frac{\begin{array}{c}\text{def}(\mathtt{n}) \qquad \mathtt{f} \notin R \qquad \mathtt{M} \in \mathscr{M} \\ \langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term} \cup \{(\mathtt{f}, \mathtt{0})\}, <:, R \cup \{\mathtt{f}\}, C \rangle \checkmark\end{array}}{\langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term}, <:, R, e\texttt{@n -> x@f;}\, C \rangle \checkmark} \text{ WF-REQ}$$

Rule WF-REQ-RES states that a request-response instruction is well-formed if `n` is defined, `f` is not in the current scope, `M` is a valid stateless medium in the current context, and the continuation is well-formed, given the additional nonterminating pair (`f`, `n`), and the extended termination order obtained by adding (`f`, `n`) to $<:$ and taking its transitive closure, denoted by $<:^{+}$. We observe that our invariants, combined with the fact that `f` is not in the current scope, tell us that the extended termination order is also a strict partial order, since `f` does not appear in $<:$, meaning that all the elements added by the transitive closure have the form (`f`, `m`), where $\texttt{n} <: \texttt{m}$.

$$\frac{\mathrm{def}(\texttt{n}) \qquad \texttt{f} \notin R \qquad \texttt{M} \in \mathscr{M} \qquad \langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term} \cup \{(\texttt{f},\texttt{n})\}, (<: \cup \{(\texttt{f},\texttt{n})\})^{+}, R \cup \{\texttt{f}\}, C\rangle\checkmark}{\langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term}, <:, R, e\texttt{@n -> x@f |> y@n;}\, C\rangle\checkmark}\ \text{WF-REQ-RES}$$

Rule WF-END states that a termination without a response is well-formed if (`f`, `0`) is a terminating pair, `f` is free to terminate, and the continuation is well-formed, with (`f`, `0`) removed from the terminating pairs. We use free(`f`) as a shorthand for $\texttt{f} \notin \pi_2(<:)$, which means that `f` is not waiting for a response from any terminating role, and it is allowed to terminate. We impose this restriction because there is no way to transmit knowledge of choice to a stateless role, since selections are restricted to stateful roles. This check prevents cases such as the creator of a stateless role that terminates without first receiving a response from the former; in such a situation the stateless role would try to answer anyway and cause a communication error, since there would be no role to receive the response. This rule only applies when the stateless role is created with a request-response instruction, which is why we only add elements to $<:$ in WF-REQ-RES, and not in WF-REQ.

$$\frac{\texttt{free(f)} \qquad (\texttt{f},\texttt{0}) \in \mathit{Term} \qquad \langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term} \backslash \{(\texttt{f},\texttt{0})\}, <:, R, C\rangle\checkmark}{\langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term}, <:, R, \texttt{end f;}\, C\rangle\checkmark}\ \text{WF-END}$$

Rule WF-END-RES states that a termination with a response is well-formed if (`f`, `n`) is a terminating pair, `f` is free to terminate, and the continuation is well-formed, with (`f`, `n`) removed from the terminating pairs, and with all (`f`, `m`) removed from $<:$. We observe that, since `f` is free to terminate and $(\texttt{f},\texttt{n}) \in \mathit{Term}$, we have that (`f`, `n`) appears in $<:$, and `f` only appears as the first element of any pair in $<:$, meaning that $<:$ remains a strict partial order after removing all pairs that start with `f` from it. We also have that `f` no longer appears in the continuation's terminating pairs.

$$\frac{(\texttt{f},\texttt{n}) \in \mathit{Term} \qquad \texttt{free(f)} \qquad \langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term} \backslash \{(\texttt{f},\texttt{n})\}, <: \backslash \pi^{-1}_{<:,1}(\texttt{f}), R, C\rangle\checkmark}{\langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term}, <:, R, \texttt{end } e\texttt{@f -> n;} C\rangle\checkmark}\ \text{WF-END-RES}$$

Rule WF-COND states that a conditional is well-formed if the guard role is defined, and if a set $D \subseteq \mathit{Term}$ exists such that all terminating pairs outside of D terminate inside both branches, and none of the terminating pairs in D terminate inside either branch, in such a way that does not cause issues. More precisely, the third premise checks that the termination order is not violated by D. The fourth and fifth premises check that each branch is well-formed when treating only the pairs outside of D as terminating, while treating the first element of each pair in D as nonterminating, and removing pairs from $<:$ accordingly. Removing these pairs preserves all invariants and only removes termination order information for roles which are treated as nonterminating inside the branches, which means it does not cause issues. Finally, the sixth and seventh premises check that the continuation is well-formed without the terminating pairs outside of D, and with all the stateless roles which appeared in the branches added to its scope. We impose that the same set of roles terminates in both branches to ensure that the continuation always has the same set of roles left to terminate, irrespective of the branch that was chosen. Without this restriction, knowledge of choice would need to be propagated outside of the branches of the conditional statement, which

would cause issues.

$$\frac{\begin{array}{c}\mathrm{def}(\mathtt{n}) \\ D \subseteq \mathit{Term} \qquad \forall\, \mathtt{f} \in \pi_1(D).\, \forall\, \mathtt{g} \in \pi_1(\mathit{Term} \backslash D).\, \mathtt{f} \not<: \mathtt{g} \\ \langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT} \cup \pi_1(D), \mathit{Term} \backslash D, <: \backslash \pi^{-1}_{<:,1}(\pi_1(D)), R, C_1 \rangle \checkmark \\ \langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT} \cup \pi_1(D), \mathit{Term} \backslash D, <: \backslash \pi^{-1}_{<:,1}(\pi_1(D)), R, C_2 \rangle \checkmark \\ R' = (\mathrm{pn}(C_1) \cup \mathrm{pn}(C_2)) \cap \mathbf{StatelessRoles} \\ \langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, D, <: \backslash \pi^{-1}_{<:,1}(\pi_1(\mathit{Term} \backslash D)), R \cup R', C \rangle \checkmark\end{array}}{\langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term}, <:, R, \mathtt{if}\ e\mathtt{@n}\ \mathtt{then}\ \{C_1\}\ \mathtt{else}\ \{C_2\}; C \rangle \checkmark}\ \text{WF-COND}$$

Rule WF-CALL states the conditions under which a call instruction is well-formed. Here, and in rule WF-RT-CALL, we assume that $F = \langle \mathtt{p}_1, \ldots, \mathtt{p}_n \rangle$, $N = \langle \mathtt{n}_1, \ldots, \mathtt{n}_m \rangle$, and $T = \langle (\mathtt{f}_1, \mathtt{s}_1), \ldots, (\mathtt{f}_l, \mathtt{s}_l) \rangle$, with $n, m, l \geq 0$, to express properties which involve single parameters more easily. We make the same assumption for $F_{\mathtt{X}}$, $N_{\mathtt{X}}$, $T_{\mathtt{X}}$, N_P,and T_P, distinguishing between them with a superscript $\mathtt{X}$ or P. The first three premises state that the parameter lists must be included in the current sets of mentionable roles, and T specifically must contain terminating pairs. The following four premises state that all parameters must be distinct, with the only exceptions being that the second element of different terminating pairs can be the same, and that stateful or nonterminating parameters can be the same as the second element of one or more nonterminating pairs. The eighth premise states that procedure $\mathtt{X}$ must be in the set of procedure definitions, with its stated termination order $<:_{\mathtt{X}}$. The next three premises state that the number of actual parameters and formal parameters must be the same. The twelfth premise checks that all terminating pairs in T match the corresponding formal parameters. The next three premises state that when two formal parameters are the same, the corresponding actual parameters must be the same. Note that this can only happen when at least one of the parameters involved is the second element of a terminating pair. The following three premises check that termination order is not violated by the procedure call. The first of these premises checks that T does not divide terminating roles in a way that violates $<:$. The other

two premises check that the body of the procedure terminates the parameters in the correct order, according to the declared termination order $<:_{\mathtt{X}}$. The last premise checks that the continuation is well-formed without the terminating pairs in T. We do not add roles to R because, when the call is first executed, all hidden roles in the procedure's body are replaced with roles that are not being used at that time and do not appear in the rest of the choreography, meaning that they never interfere with the rest of the program.

The second of the three termination order premises, $\forall\, i, j.\mathtt{s}_i = \mathtt{f}_j \Rightarrow \mathtt{f}_i^{\mathtt{X}} <:_{\mathtt{X}} \mathtt{f}_j^{\mathtt{X}}$, requires further explanation. Intuitively, we should also require that $\mathtt{s}_i <: \mathtt{f}_j \Rightarrow \mathtt{f}_i^{\mathtt{X}} <:_{\mathtt{X}} \mathtt{f}_j^{\mathtt{X}}$, but this is not necessary, because it comes as a consequence of the chosen termination order premises. If the call is well-formed and we have i and j such that $\mathtt{s}_i <: \mathtt{f}_j$, then there must be some k such that $\mathtt{f}_k = \mathtt{s}_i$, otherwise the first premise would be violated. This means that, by the second premise, we have $\mathtt{f}_i^{\mathtt{X}} <:_{\mathtt{X}} \mathtt{f}_k^{\mathtt{X}}$. Since $\mathtt{s}_i = \mathtt{f}_k$, we also have $\mathtt{f}_k <: \mathtt{f}_j$, and by the third premise we get that $\mathtt{f}_k^{\mathtt{X}} <:_{\mathtt{X}} \mathtt{f}_j^{\mathtt{X}}$. Finally, in a well-formed $\mathscr{C}$, which we define further ahead, we have that $<:_{\mathtt{X}}$ is a strict partial order, which means that it is transitive, which gives us $\mathtt{f}_i^{\mathtt{X}} <:_{\mathtt{X}} \mathtt{f}_j^{\mathtt{X}}$.

$$\frac{\begin{array}{c} F \subseteq \mathit{Ful} \qquad N \subseteq \mathit{NonT} \cup \mathit{Ful} \cup \pi_1(\mathit{Term}) \\ T \subseteq \mathit{Term} \qquad \forall\, i \neq j.\mathtt{p}_i \neq \mathtt{p}_j \qquad \forall\, i \neq j.\mathtt{n}_i \neq \mathtt{n}_j \\ \forall\, i \neq j.\mathtt{f}_i \neq \mathtt{f}_j \qquad \forall\, i, j, k.\mathtt{p}_i \neq \mathtt{n}_j \neq \mathtt{f}_k \neq \mathtt{s}_k \\ (\mathtt{X(}F_{\mathtt{X}}\mathtt{,\ nonterm\ }N_{\mathtt{X}}\mathtt{,\ term\ }T_{\mathtt{X}}\mathtt{)} = C_{\mathtt{X}}, <:_{\mathtt{X}}) \in \mathscr{C} \\ |F| = |F_{\mathtt{X}}| \qquad |N| = |N_{\mathtt{X}}| \\ |T| = |T_{\mathtt{X}}| \qquad \forall\, i.\mathtt{s}_i = \mathtt{0} \Leftrightarrow \mathtt{s}_i^{\mathtt{X}} = \mathtt{0} \qquad \forall\, i, j.\mathtt{p}_i^{\mathtt{X}} = \mathtt{s}_j^{\mathtt{X}} \Rightarrow \mathtt{p}_i = \mathtt{s}_j \\ \forall\, i, j.\mathtt{n}_i^{\mathtt{X}} = \mathtt{s}_j^{\mathtt{X}} \Rightarrow \mathtt{n}_i = \mathtt{s}_j \qquad \forall\, i, j.\mathtt{s}_i^{\mathtt{X}} = \mathtt{s}_j^{\mathtt{X}} \Rightarrow \mathtt{s}_i = \mathtt{s}_j \\ \forall\, \mathtt{f} \in \pi_1(\mathit{Term} \backslash T).\, \forall\, \mathtt{g} \in \pi_1(T).\mathtt{f} \not<: \mathtt{g} \\ \forall\, i, j.\mathtt{s}_i = \mathtt{f}_j \Rightarrow \mathtt{f}_i^{\mathtt{X}} <:_{\mathtt{X}} \mathtt{f}_j^{\mathtt{X}} \qquad \forall\, i, j.\mathtt{f}_i <: \mathtt{f}_j \Rightarrow \mathtt{f}_i^{\mathtt{X}} <:_{\mathtt{X}} \mathtt{f}_j^{\mathtt{X}} \\ \langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term} \backslash T, <: \backslash \pi_{<:,1}^{-1}(\pi_1(T)), R, C \rangle \checkmark \end{array}}{\langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term}, <:, R, \mathtt{X(}F\mathtt{,\ nonterm\ }N\mathtt{,\ term\ }T\mathtt{);}\, C \rangle \checkmark}\ \text{WF-CALL}$$

```
1   def X(a, b, nonterm c, term [d, e]){
2     x@a -> x@b;
3     end x@d -> e;
4     x@c = (x+1)@c;
5   }
6   def main(){
7     0@q -> x@p;
8     0@p -M-> x@f;
9     0@f <-M-> x@g |> y@f;
10    X(p, q, nonterm f, term [g, f]);
11    end f;
12  }
```

Figure 4.1: Procedure call where all stateless parameters can terminate before the stateful roles enter the call.

Rule WF-RT-CALL states the conditions under which a runtime term for a procedure call is well-formed. Since many of the premises are the same as WF-CALL, we only discuss the premises that are different. We start with T, which is no longer restricted to being a subset of *Term*, because some of the terminating roles that were originally part of the procedure call may have been terminated in the expanded procedure body, before the other roles entered the call. N is similarly extended to allow for any stateless role, since a terminating role that was treated as nonterminating in the call could still terminate after completing its part in the procedure's body. We show an example of a program that can lead to these situations in Figure 4.1. The eighth and ninth premises check that N_P only includes roles which were nonterminating or stateful in the original call, and that can be included according to the current context. The tenth and eleventh premises are similar, but for T_P. We write $\subseteq^*$ in these two premises to denote an improper subset relation, which is defined formally below. The twelfth premise checks that at least one role has entered the call. The second-to-last premise checks that the original continuation after the original procedure call is well-formed without the terminating pairs in T, while the last premise checks that the actual con-

tinuation, which already contains the expansion of the procedure's body, is well-formed. We do not include a corresponding premise to the last premise of rule WF-RT-CALL in the standard definition of well-formedness[4], since we do not define Endpoint Projection for FaaSChalCore. A corresponding premise must be added once EPP is defined for the language.

We say that $T_1 \subseteq^* T_2$ if the following statements are true:

- $\forall(\mathtt{f},\mathtt{n}) \in T_1.(\mathtt{f},\mathtt{n}) \in T_2$
- $\forall(\mathtt{0},\mathtt{n}) \in T_1.\mathtt{n} \in \pi_2(T_2 \backslash \{(\mathtt{g},\mathtt{m}) \in T_1\})$
- $\forall(\mathtt{f},\mathtt{0}) \in T_1.\mathtt{f} \in \pi_1(T_2 \backslash \{(\mathtt{g},\mathtt{m}) \in T_1\})$

$$\frac{\begin{array}{c} F \subseteq \mathit{Ful} \qquad N \subseteq \mathit{NonT} \cup \mathit{Ful} \cup \mathbf{StatelessRoles} \\ T \subseteq (\mathbf{StatelessRoles} \backslash \mathit{NonT}) \times (\mathbf{Roles} \cup \{\mathtt{0}\}) \qquad \forall\, i \neq j.\mathtt{p}_i \neq \mathtt{p}_j \\ \forall\, i \neq j.\mathtt{n}_i \neq \mathtt{n}_j \qquad \forall\, i \neq j.\mathtt{f}_i \neq \mathtt{f}_j \qquad \forall\, i,j,k.\mathtt{p}_i \neq \mathtt{n}_j \neq \mathtt{f}_k \neq \mathtt{s}_k \\ N_P \subseteq F \cup N \qquad N_P \subseteq \mathit{NonT} \cup \mathit{Ful} \cup \pi_1(\mathit{Term}) \\ T_P \subseteq^* T \qquad T_P \subseteq^* \mathit{Term} \qquad \mathrm{pn}(N_P \cup T_P) \subset \mathrm{pn}(F \cup N \cup T) \\ (\mathtt{X(}F_\mathtt{X}\mathtt{,\ nonterm\ }N_\mathtt{X}\mathtt{,\ term\ }T_\mathtt{X}\mathtt{)} = C_\mathtt{X}, <:_\mathtt{X}) \in \mathscr{C} \\ |F| = |F_\mathtt{X}| \qquad |N| = |N_\mathtt{X}| \qquad |T| = |T_\mathtt{X}| \qquad \forall\, i.\mathtt{s}_i = \mathtt{0} \Leftrightarrow \mathtt{s}_i^\mathtt{X} = \mathtt{0} \\ \forall\, i,j.\mathtt{p}_i^\mathtt{X} = \mathtt{s}_j^\mathtt{X} \Rightarrow \mathtt{p}_i = \mathtt{s}_j \qquad \forall\, i,j.\mathtt{n}_i^\mathtt{X} = \mathtt{s}_j^\mathtt{X} \Rightarrow \mathtt{n}_i = \mathtt{s}_j \\ \forall\, i,j.\mathtt{s}_i^\mathtt{X} = \mathtt{s}_j^\mathtt{X} \Rightarrow \mathtt{s}_i = \mathtt{s}_j \qquad \forall\, \mathtt{f} \in \pi_1(\mathit{Term} \backslash T).\forall\, \mathtt{g} \in \pi_1(T).\mathtt{f} \not<: \mathtt{g} \\ \forall\, i,j.\mathtt{s}_i = \mathtt{f}_j \Rightarrow \mathtt{f}_i^\mathtt{X} <:_\mathtt{X} \mathtt{f}_j^\mathtt{X} \qquad \forall\, i,j.\mathtt{f}_i <: \mathtt{f}_j \Rightarrow \mathtt{f}_i^\mathtt{X} <:_\mathtt{X} \mathtt{f}_j^\mathtt{X} \\ \langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term} \backslash T, <: \backslash \pi_{<:,1}^{-1}(\pi_1(T)), R, C' \rangle \checkmark \\ \langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term}, <:, R, C \rangle \checkmark \end{array}}{\langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT}, \mathit{Term}, <:, R, N_P\mathtt{,\ term\ }T_P\mathtt{:X(}F\mathtt{,\ nonterm\ }N\mathtt{,\ term\ }T\mathtt{)}.C'; C \rangle \checkmark}\ \text{WF-RT-CALL}$$

4.2 Well-formedness of Programs

We now use the statements defined in the previous section to provide the definition of well-formedness of a FaaSChalCore program.

Before doing so, we formally describe how

$$(\texttt{X(}F_\texttt{X}\texttt{, nonterm }N_\texttt{X}\texttt{, term }T_\texttt{X}\texttt{)} = C_\texttt{X}, <:_\texttt{X})$$

is obtained from the corresponding procedure declaration.

Let $\texttt{def X(}F_\texttt{X}\texttt{, nonterm }N_\texttt{X}\texttt{, term }T_\texttt{X}\texttt{):(}O_\texttt{X}\texttt{)\{}C_\texttt{X}\texttt{\}}$ be a procedure declaration, we define $<:_\texttt{X}$ as the transitive closure of the union of all pairs in $O_\texttt{X}$ and all pairs in $T_\texttt{X}$ that do not have $\texttt{0}$ as a second element, or, more formally, $<:_\texttt{X} = (O_\texttt{X} \cup (T_\texttt{X} \backslash \pi^{-1}_{T_\texttt{X},2}(\texttt{0})))^+$.

Definition 4.1 (Runtime Well-formedness of Choreographies)**.** Let $\mathscr{C}$ be a set of procedure definitions, $\mathscr{M} \subseteq$ **Media** be a set of stateless media,

$$NonT \subseteq \textbf{Roles},$$

$$Term \subseteq (\textbf{StatelessRoles} \backslash NonT) \times (\textbf{Roles} \cup \{\texttt{0}\}), \text{ and}$$

$$R \supseteq NonT \cup \pi_1(Term)$$

we say that $C \in$ **FaaSChalCore** is runtime well-formed in the context of $\mathscr{M}, \mathscr{C}, NonT$, $Term$, and R if some $Ful \subseteq$ **StatefulRoles** exists, such that

$$\langle \mathscr{M}, \mathscr{C}, Ful, NonT, Term, (Term \backslash \pi^{-1}_{Term,2}(\texttt{0}))^+, R, C \rangle \checkmark \text{ is derivable,}$$

and the invariants we assumed when defining well-formedness are respected.

Definition 4.2 (Well-formedness of Procedures)**.** Let $\mathscr{M} \subseteq$ **Media** be a set of stateless media and $\mathscr{C}$ be a set of procedure definitions. We say that $\mathscr{C}$ is well-formed in the context of $\mathscr{M}$ if,

$$\forall(\texttt{X(}F_\texttt{X}\texttt{, nonterm }N_\texttt{X}\texttt{, term }T_\texttt{X}\texttt{)} = C_\texttt{X}, <:_\texttt{X}) \in \mathscr{C}.$$

the following statements are true:

- $\langle \mathscr{M}, \mathscr{C}, F_\texttt{X}, N_\texttt{X}, T_\texttt{X}, <:_\texttt{X}, N_\texttt{X} \cup \pi_1(T_\texttt{X}), C_\texttt{X}\rangle\checkmark$ is derivable;
- $(\forall\, i \neq j . \texttt{p}_i^\texttt{X} \neq \texttt{p}_j^\texttt{X}) \wedge (\forall\, i \neq j . \texttt{n}_i^\texttt{X} \neq \texttt{n}_j^\texttt{X}) \wedge (\forall\, i \neq j . \texttt{f}_i^\texttt{X} \neq \texttt{f}_j^\texttt{X})$;
- $\forall\, i, j, k . \texttt{p}_i^\texttt{X} \neq \texttt{n}_j^\texttt{X} \neq \texttt{f}_k^\texttt{X} \neq \texttt{s}_k^\texttt{X}$;
- $<:_\texttt{X}$ is a strict partial order;
- $\forall\, i . \texttt{s}_i^\texttt{X} \not<:_\texttt{X} \texttt{f}_i^\texttt{X}$, meaning that no creator of a role terminates before the role it created;
- $|\text{pn}(F_\texttt{X} \cup N_\texttt{X} \cup T_\texttt{X})| \geq 1$, meaning that there must be at least one formal parameter;
- $|\text{pn}(C_\texttt{X})| \geq 2$, meaning that the body of the procedure must contain at least two roles, otherwise its behaviour can be expressed using a local function;
- $\text{pn}(F_\texttt{X} \cup N_\texttt{X} \cup T_\texttt{X}) \subseteq \text{pn}(C_\texttt{X})$, meaning that the procedure's body must mention all formal parameters.

We observe that these premises mean that $C_\texttt{X}$ is runtime well-formed in the context of $\mathscr{M}, \mathscr{C}, N_\texttt{X}$, $T_\texttt{X}$, and $R = N_\texttt{X} \cup \pi_1(T_\texttt{X})$, since $<:_\texttt{X}$ is more specific than the termination order resulting from $T_\texttt{X}$, and the other premises guarantee that $F_\texttt{X}$ does not violate the invariants required in Definition 4.1.

Definition 4.3 (Runtime well-formedness of Choreographic Configurations)**.** Given $NonT \subseteq \mathbf{Roles}$ and

$$Term \subseteq (\mathbf{StatelessRoles} \backslash NonT) \times (\mathbf{Roles} \cup \{\mathtt{0}\}),$$

we say that the choreographic configuration $\langle C, \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle$ is runtime well-formed in the context of $NonT$ and $Term$ if:

- $\mathscr{C}$ is well-formed in the context of $\mathscr{M}$;
- C is runtime well-formed in the context of $\mathscr{M}, \mathscr{C}, NonT, Term$, and $R = \mathrm{dom}(\Sigma)$;
- $\mathrm{dom}(\Sigma) = NonT \cup \pi_1(Term)$, which means that the choreographic store must be consistent with the sets of nonterminating and terminating stateless roles;
- $\forall(\mathtt{f}, \mathtt{n}) \in Term \backslash \pi^{-1}_{Term,2}(\mathtt{0}).\, \exists\, \mathtt{M} \in \mathscr{M}, \mathtt{x} \in \mathbf{Var}.\, \Psi(\mathtt{f}) \overset{\mathtt{M}}{\mapsto} \mathtt{x@n}$, which means that the response store must contain all responses prescribed by $Term$;
- $\forall\, \mathtt{f} \in \pi_1(\pi^{-1}_{Term,2}(\mathtt{0}))\, .\, \mathtt{f} \notin \mathrm{dom}(\Psi)$, which means that the response store must not contain response information for roles that will not respond when they terminate, according to $Term$;
- $\mathrm{dom}(\Psi) \subseteq R$.

Definition 4.4 (Well-formedness of Choreographic Configurations)**.** We say that a choreographic configuration $\langle C, \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle$ is well-formed if it is runtime well-formed in the context of $NonT = \varnothing$ and $Term = \varnothing$. We observe that, by Definition 4.3, this also means that $\mathrm{dom}(\Sigma) \subseteq$ **StatefulRoles** and $\mathrm{dom}(\Psi) = \varnothing$.

Definition 4.5 (Well-formedness of FaaSChalCore Programs)**.** Let $\mathscr{M}$ and $\mathscr{C}$ respectively be the set of stateless media and the set of procedure declarations obtained from a FaaSChalCore program, and let C be the main choreography of that program. We say that the program is well-formed if there exist Σ and Ψ such that $\langle C, \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle$ is a well-formed choreographic configuration.

Chapter 5

Properties of FaaSChalCore

In this Chapter we formulate interesting properties that arise from well-formedness, and outline a sketch of the proofs of these properties.

Proposition 5.1. Let $\langle C, \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle$ be a runtime well-formed configuration in the context of $NonT$ and $Term$, let $<: = (Term \backslash \pi^{-1}_{Term,2}(\mathtt{0}))^+$ be the termination order resulting from $Term$, and let

$$<:' = (\{(\mathtt{f}, \mathtt{n}) \mid \mathtt{f} \in \pi_1(Term) \wedge \exists\, \mathtt{M} \in \mathscr{M}, \mathtt{x} \in \mathbf{Var}. \Psi(\mathtt{f}) \overset{\mathtt{M}}{\mapsto} \mathtt{x@n}\})^+$$

be the termination order resulting from Ψ, then we have that $<: = <:'$.

Proof. This is an immediate consequence of Definition 4.3, which gives us that

$$\forall (\mathtt{f}, \mathtt{n}) \in Term \backslash \pi^{-1}_{Term,2}(\mathtt{0}). \exists\, \mathtt{M} \in \mathscr{M}, \mathtt{x} \in \mathbf{Var}.\ \Psi(\mathtt{f}) \overset{\mathtt{M}}{\mapsto} \mathtt{x@n}$$

and that

$$\forall\, \mathtt{f} \in \pi_1(\pi^{-1}_{Term,2}(\mathtt{0})).\ \mathtt{f} \notin \mathrm{dom}(\Psi)$$

□

Lemma 5.2. Let $\mathit{Term} = T \cup T'$, let $\langle C, \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle$ be a runtime well-formed configuration in the context of $\mathit{NonT} \cup \pi_1(T')$ and T, and let $\langle C', \Sigma', \Psi', \mathscr{C}, \mathscr{M} \rangle$ be a runtime well-formed configuration in the context of NonT and T', such that

- $\pi_1(T) \# \pi_1(T')$;
- $\mathrm{pn}(C) \backslash (\mathit{NonT} \cup \pi_1(T') \cup \mathbf{StatefulRoles}) \# \mathrm{pn}(C')$, i.e., the roles that appear in C and end within C must not appear in C';
- $\forall (\mathtt{f}, \mathtt{n}) \in T' \backslash \pi^{-1}_{T',2}(\mathtt{0}).\, \exists\, \mathtt{M} \in \mathscr{M}, \mathtt{x} \in \mathbf{Var}.\, \Psi(\mathtt{f}) \overset{\mathtt{M}}{\mapsto} \mathtt{x@n}$;
- $\forall\, \mathtt{f} \in \pi_1(\pi^{-1}_{T',2}(\mathtt{0}))\,.\, \mathtt{f} \notin \mathrm{dom}(\Psi)$;
- $\forall\, \mathtt{f} \in \pi_1(T').\Psi'(\mathtt{f}) = \Psi(\mathtt{f})$.

Let $<: = (T \backslash \pi^{-1}_{T,2}(\mathtt{0}))^+$ and $<:' = (T' \backslash \pi^{-1}_{T',2}(\mathtt{0}))^+$ be the termination orders obtained from T and T' respectively, and let $<:_{\mathit{Term}} = (\mathit{Term} \backslash \pi^{-1}_{\mathit{Term},2}(\mathtt{0}))^+$ be the termination order resulting from their union. If $<:_{\mathit{Term}}$ is a strict partial order and $\forall\, \mathtt{f} \in \pi_1(T').\, \forall\, \mathtt{g} \in \pi_1(T).\mathtt{f} \not<:_{\mathit{Term}} \mathtt{g}$, then $\langle C \fatsemi C', \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle$ is a runtime well-formed configuration in the context of NonT and Term

Proof. We proceed by induction on the structure of C

$\Diamond$ Case $C = \mathbf{0}$

By Definitions 4.3 and 4.1 we get that the following statement must be derivable, with $R = \mathrm{dom}(\Sigma)$

$$\langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT} \cup \pi_1(T'),\, T, <:, R, \mathbf{0} \rangle \checkmark$$

This is only possible using rule WF-CLOSE, which requires that $T = \varnothing$

$$\frac{T = \varnothing}{\langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT} \cup \pi_1(T'),\, T, <:, R, \mathbf{0} \rangle \checkmark}\ \text{WF-CLOSE}$$

$T = \varnothing$, combined with Definitions 4.3 and 4.1, gives us that

- $\mathrm{dom}(\Sigma) = \mathit{NonT} \cup \pi_1(T') = \mathrm{dom}(\Sigma')$;
- $\mathrm{dom}(\Psi) \subseteq R = \mathrm{dom}(\Sigma')$.

These results, combined with our hypotheses, give us that $\langle C', \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle$ is a runtime well-formed configuration in the context of NonT and Term, which concludes case $C = \mathbf{0}$, since $C \fatsemi C' = \mathbf{0} \fatsemi C' = C'$ by definition of $\fatsemi$. ⧫

◊ Case $C = I;\ C''$

By Definitions 4.3 and 4.1 we get that C must be runtime well-formed in the context of $\mathscr{M}, \mathscr{C}, \mathit{NonT} \cup \pi_1(T')$, T, and $R = \mathrm{dom}(\Sigma)$, so there must be some $\mathit{Ful} \subseteq \mathbf{StatefulRoles}$ such that

$$\langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT} \cup \pi_1(T'), T, <:, R, I;\ C'' \rangle \checkmark \text{ is derivable}$$

where $<: = (\mathit{Term} \backslash \pi^{-1}_{\mathit{Term},2}(\mathit{Term}))^+$. All well-formedness rules that could apply to $I;\ C''$, i.e., all except WF-CLOSE, also require that

$$\langle \mathscr{M}, \mathscr{C}, \mathit{Ful}, \mathit{NonT} \cup \pi_1(T'), T'', (T'' \backslash \pi^{-1}_{T'',2}(\mathtt{0}))^+, R'', C'' \rangle \checkmark \text{ is derivable}$$

for some $T'' \subseteq (\mathbf{StatelessRoles} \backslash \mathit{NonT}) \times (\mathbf{Roles} \cup \{\mathtt{0}\})$, which depends on the specific rule, and $R'' = R \cup \pi_1(T'')$. We observe that in all rules, we have that either $T'' \subseteq T$, or $T'' = T \cup \{(\mathtt{f}, \mathtt{s})\}$, and the added or removed pairs only contain roles that appear in $\mathrm{pn}(I)$. In all cases, it is possible to add or remove elements from Σ and Ψ to obtain Σ'' and Ψ'', and to construct them in such a way that

$$\langle C'', \Sigma'', \Psi'', \mathscr{C}, \mathscr{M} \rangle$$

is a runtime well-formed configuration in the context of $\mathit{NonT} \cup \pi_1(T')$ and T'', and all premises of this theorem are true for T'', Σ'', and Ψ''. In par-

ticular we know that $\pi_1(T'')\#\pi_1(T')$ because $\pi_1(T') \subseteq R$, meaning that if T'' contains a new role, that role must be in $\pi_1(T')$. We can then apply the induction hypothesis to C'', obtaining that

$$\langle C'' \fatsemi C', \Sigma'', \Psi'', \mathscr{C}, \mathscr{M} \rangle$$

is a runtime well-formed configuration in the context of $NonT$ and $T' \cup T''$.

Now we use the fact that $(I;\ C'') \fatsemi C' = I;\ (C'' \fatsemi C')$ to prove that

$$\langle C \fatsemi C', \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle$$

is a runtime well-formed configuration in the context of $NonT$ and $Term$.

We start by proving that $I;\ (C'' \fatsemi C')$ is runtime well-formed in the context of $\mathscr{M}, \mathscr{C}, NonT$, $Term$, and $R = \mathrm{dom}(\Sigma)$. To do this, we must prove there is some Ful such that

$$\langle \mathscr{M}, \mathscr{C}, Ful, NonT, Term, <:_{Term}, R, I;\ (C'' \fatsemi C') \rangle \checkmark \text{ is derivable}$$

and that the sets involved in the statement respect the invariants required in Definition 4.1. Any derivation for this statement must end with the same rule used to obtain

$$\langle \mathscr{M}, \mathscr{C}, Ful, NonT \cup \pi_1(T'), T, <:, R, I;\ C'' \rangle \checkmark \text{ is derivable}$$

meaning that it requires that

$$\langle \mathscr{M}, \mathscr{C}, Ful, NonT, T''', <:''', R''', C'' \fatsemi C' \rangle \checkmark \text{ is derivable}$$

We know from the statement on C'' that $\mathrm{pn}(I) \subseteq \mathrm{pn}(T)$, and that T''' is obtained from $Term$ by applying the same modifications that got us T'' from T. Since $\pi_1(T)\#\pi_1(T')$, this means that $T''' = T' \cup T''$, $<:''' = (T' \cup T'')^+$, $R''' = R' \cup R''$, and that the required statement is obtained from the fact

that

$$\langle C'' \fatsemi C', \Sigma'', \Psi'', \mathscr{C}, \mathscr{M} \rangle$$

is a runtime well-formed configuration in the context of $NonT$ and $T' \cup T''$.

This also allows us to prove the other requirements for Definitions 4.1 and 4.3, concluding case $C = I;\ C''$. ♦ □

Lemma 5.3. Let $C = \texttt{if}\ e\texttt{@n then}\ \{C_1\}\ \texttt{else}\ \{C_2\}; C'$. If $\langle C, \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle$ is a runtime well-formed configuration in the context of $NonT$ and $Term$, then we have that there exists some $D \subseteq Term$ such that

- $\forall\, \texttt{f} \in \pi_1(D). \forall\, \texttt{g} \in \pi_1(Term \backslash D). \texttt{f} \not<: \texttt{g}$;
- $\langle C_1, \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle$ and $\langle C_2, \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle$ are runtime well-formed configurations in the context of $NonT \cup \pi_1(D)$ and $Term \backslash D$;
- $\langle C', \Sigma \backslash \pi_1(Term \backslash D), \Psi \backslash \pi_1(Term \backslash D), \mathscr{C}, \mathscr{M} \rangle$ is runtime well-formed in the context of $NonT$ and D;
- $\mathrm{pn}(C_1) \backslash (NonT \cup \pi_1(D) \cup \mathbf{StatefulRoles}) \# \mathrm{pn}(C')$;
- $\mathrm{pn}(C_2) \backslash (NonT \cup \pi_1(D) \cup \mathbf{StatefulRoles}) \# \mathrm{pn}(C')$.

This means that both $C_1 \fatsemi C'$ and $C_2 \fatsemi C'$ satisfy the requirements for Lemma 5.2. When combined with Theorem 5.7, we also obtain that both branches of a runtime well-formed conditional cause the exact same set of terminating stateless roles to terminate, and each branch also terminates any new stateless roles it might create.

Proof. This is a direct consequence of Definitions 4.3 and 4.1, and of the premises for rule WF-COND. □

Lemma 5.4. Let $C = \texttt{X(}F\texttt{, nonterm }N\texttt{, term }T\texttt{)}\,;C'$. If $\langle C, \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle$ is a runtime well-formed configuration in the context of $NonT$ and $Term$, then there must be some $(\texttt{X(}F_{\texttt{X}}\texttt{, nonterm }N_{\texttt{X}}\texttt{, term }T_{\texttt{X}}\texttt{)} = C_{\texttt{X}}, <:_{\texttt{X}}) \in \mathscr{C}$, and there exists $H \subseteq$ **StatelessRoles** such that

- $H \# \mathrm{dom}(\Sigma) \cup \mathrm{pn}(C')$;
- $|H| = |H_{\texttt{X}}|$, where $H_{\texttt{X}} = \mathrm{pn}(C_{\texttt{X}}) \backslash \mathrm{pn}(F_{\texttt{X}} \cup N_{\texttt{X}} \cup T_{\texttt{X}})$;
- $\langle C_{\texttt{X}}[H, F, N, T/H_{\texttt{X}}, F_{\texttt{X}}, N_{\texttt{X}}, T_{\texttt{X}}], \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle$ is a runtime well-formed configuration in the context of $NonT \cup \pi_1(Term \backslash T)$ and T;
- $\langle C', \Sigma \backslash \pi_1(T), \Psi \backslash \pi_1(T), \mathscr{C}, \mathscr{M}\rangle$ is runtime well-formed in the context of $NonT$ and $Term \backslash T$;
- $\mathrm{pn}(C_{\texttt{X}}[H, F, N, T/H_{\texttt{X}}, F_{\texttt{X}}, N_{\texttt{X}}, T_{\texttt{X}}]) \backslash (NonT \cup \pi_1(Term \backslash T) \cup$ **StatefulRoles**$) \# \mathrm{pn}(C')$.

This means that $C_{\texttt{X}}[H, F, N, T/H_{\texttt{X}}, F_{\texttt{X}}, N_{\texttt{X}}, T_{\texttt{X}}]\,{}^{\S}_{9}\,C'$ satisfies the requirements for Lemma 5.2. When combined with Theorem 5.7, we obtain that the expanded body of the call, with appropriate replacement of hidden roles, causes the terminating parameters to terminate, and also terminates any new stateless roles it might create.

Proof. This is a direct consequence of Definitions 4.3 and 4.1, and of the premises of rule WF-CALL. □

Theorem 5.5. Let $\langle C, \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle$ be a runtime well-formed configuration in the context of $NonT$ and $Term$. If $\langle C, \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle \xrightarrow{\mu} \langle C', \Sigma', \Psi', \mathscr{C}, \mathscr{M} \rangle$, then we have that $\langle C', \Sigma', \Psi', \mathscr{C}, \mathscr{M} \rangle$ is a runtime well-formed configuration in the context of $NonT$ and $Term'$, where $Term'$ is either equal to $Term$, when $\mathrm{dom}(\Sigma) = \mathrm{dom}(\Sigma')$, or is obtained by adding or removing an element from $Term$ otherwise. If $\mathrm{dom}(\Sigma') = \mathrm{dom}(\Sigma) \backslash \mathtt{f}$, then $Term' = Term \backslash \pi^{-1}_{Term,1}(\mathtt{f})$, which only has one less element. If $\mathrm{dom}(\Sigma') = \mathrm{dom}(\Sigma) \cup \{\mathtt{f}\}$, then $Term' = Term \cup \{(\mathtt{f}, \mathtt{s})\}$, where $\mathtt{s} = \mathtt{0}$ if $\mathtt{f} \notin \mathrm{dom}(\Psi')$, and $\mathtt{s} = \mathtt{n}$ if $\Psi'(\mathtt{f}) \overset{\mathtt{M}}{\mapsto} \mathtt{x@n}$ for some $\mathtt{M} \in \mathscr{M}, \mathtt{x} \in \mathbf{Var}$.

Proof. We proceed by induction on the last rule applied in the derivation of $\langle C, \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle \xrightarrow{\mu} \langle C', \Sigma', \Psi', \mathscr{C}, \mathscr{M} \rangle$, and thus on the height of the corresponding derivation tree, showing the relevant cases.

◊ Case REQ
The proofs for cases LOC, COM, SEL, REQ-RES, END, END-RES, CALL-ENTER, and CALL-LAST follow a similar structure.

$$\frac{\mathtt{M} \in \mathscr{M} \qquad \Sigma(\mathtt{n}) \vdash e \downarrow v}{\langle e\mathtt{@n\ -M->\ x@f;}\, C', \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle \xrightarrow{v\mathtt{@n} \overset{\mathtt{M}}{\longrightarrow} \mathtt{f}} \langle C', \Sigma[\mathtt{x@f} \mapsto v], \Psi, \mathscr{C}, \mathscr{M} \rangle}\ \text{REQ}$$

We have that the height of the derivation tree is 1, $C = e\mathtt{@n\ -M->\ x@f;}\, C'$, $\Sigma' = \Sigma[\mathtt{x@f} \mapsto v]$, and $\Psi' = \Psi$, which means that $\mathrm{dom}(\Sigma') = \mathrm{dom}(\Sigma) \cup \{\mathtt{f}\}$, and then that $Term' = Term \cup \{(\mathtt{f}, \mathtt{0})\}$.

We must prove that $\langle C', \Sigma[\mathtt{x@f} \mapsto v], \Psi, \mathscr{C}, \mathscr{M} \rangle$ is runtime well-formed in the context of $NonT$ and $Term'$. By hypothesis and Definition 4.3 we have that

$$\mathrm{dom}(\Sigma) = NonT \cup \pi_1(Term)$$

Combined with $\mathrm{dom}(\Sigma') = \mathrm{dom}(\Sigma) \cup \{\mathtt{f}\}$ and $Term' = Term \cup \{(\mathtt{f}, \mathtt{0})\}$, we

obtain

$$\mathrm{dom}(\Sigma') = NonT \cup \pi_1(Term')$$

We similarly prove that

- $\forall(\mathtt{g},\mathtt{m}) \in Term' \backslash \pi^{-1}_{Term',2}(\mathtt{0}).\, \exists\, \mathtt{N} \in \mathscr{M}, \mathtt{y} \in \mathbf{Var}.\, \Psi'(\mathtt{g}) \overset{\mathtt{N}}{\mapsto} \mathtt{y@m}$;
- $\forall\, \mathtt{g} \in \pi_1(\pi^{-1}_{Term',2}(\mathtt{0})).\, \mathtt{g} \notin \mathrm{dom}(\Psi')$;
- $\mathrm{dom}(\Psi') \subseteq R' = \mathrm{dom}(\Sigma')$.

Now we must prove that C' is runtime well-formed in the context of $\mathscr{M}$, $\mathscr{C}$, $NonT$, $Term'$, and R'. By hypothesis and Definition 4.3 we have that C is runtime well-formed in the context of $\mathscr{M}$, $\mathscr{C}$, $NonT$, $Term$, and R, where $R = \mathrm{dom}(\Sigma)$. By Definition 4.1, we obtain that there is some $Ful \subseteq$ **StatefulRoles** such that the sets in the following statement respect the invariants referenced in the Definition, and that

$$\langle \mathscr{M}, \mathscr{C}, Ful, NonT, Term, <:= (Term \backslash \pi^{-1}_{Term,2}(\mathtt{0}))^+, R, C\rangle\checkmark \text{ is derivable}$$

Since $C = e\texttt{@n -M-> x@f;}\, C''$, the derivation of the statement must end with rule WF-REQ

$$\frac{\mathrm{def}(\mathtt{n}) \qquad \mathtt{f} \notin R \qquad \mathtt{M} \in \mathscr{M} \qquad \langle \mathscr{M}, \mathscr{C}, Ful, NonT, Term \cup \{(\mathtt{f},\mathtt{0})\}, <:, R \cup \{\mathtt{f}\}, C'\rangle\checkmark}{\langle \mathscr{M}, \mathscr{C}, Ful, NonT, Term, <:, R, e\texttt{@n -> x@f;}\, C'\rangle\checkmark}\ \text{WF-REQ}$$

Which gives us that

$$\langle \mathscr{M}, \mathscr{C}, Ful, NonT, Term', <:, R', C'\rangle\checkmark \text{ is derivable}$$

The proof that this statement respects the invariants required by Definition 4.1 is trivial, since we update $Term'$ to match Σ' and Ψ. This proves that C' is runtime well-formed in the context of $\mathscr{M}, \mathscr{C}, NonT$, $Term'$, and R', and

thus concludes case REQ.♦

◊ Case COND-THEN and COND-ELSE
We focus on case COND-THEN, case COND-ELSE follows the same structure.

$$\frac{\Sigma(\mathtt{n}) \vdash e \downarrow \mathtt{true}}{\langle \mathtt{if}\ e\mathtt{@n}\ \mathtt{then}\ \{C_1\}\ \mathtt{else}\ \{C_2\}\,;C'', \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle \xrightarrow{\tau@\mathtt{n}} \langle C_1 \fatsemi C'', \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle}\ \text{COND-THEN}$$

We have that the height of the derivation tree is 1, $C = \mathtt{if}\ e\mathtt{@n}\ \mathtt{then}\ \{C_1\}\ \mathtt{else}\ \{C_2\}\,;C''$ and $C' = C_1 \fatsemi C''$, and we must prove that $\langle C_1 \fatsemi C'', \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle$ is runtime well-formed in the context of $NonT$ and $Term' = Term$. We have that

$$\langle \mathtt{if}\ e\mathtt{@n}\ \mathtt{then}\ \{C_1\}\ \mathtt{else}\ \{C_2\}\,;C'', \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle$$

is runtime well-formed in the context of $NonT$ and $Term$ by hypothesis, and by Lemma 5.3 and Lemma 5.2 we get that $\langle C_1 \fatsemi C'', \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle$ is runtime well-formed in the context of $NonT$ and $Term$, concluding Case COND-THEN. ♦

◊ Cases CALL-ONLY and CALL-FIRST follow the same structure as case COND-THEN, using Lemma 5.4 instead of Lemma 5.3.

◊ Cases DELAY and DELAY-COND
We focus on case DELAY, case DELAY-COND follows a similar structure.

$$\frac{\langle C'', \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle \xrightarrow{\mu} \langle C''', \Sigma', \Psi', \mathscr{C}, \mathscr{M}\rangle \qquad \mathrm{pn}(I)\#\mathrm{pn}(\mu)}{\langle I\,;C'', \Sigma, \Psi, \mathscr{C}, \mathscr{M}\rangle \xrightarrow{\mu} \langle I\,;C''', \Sigma', \Psi', \mathscr{C}, \mathscr{M}\rangle}\ \text{DELAY}$$

We have that the height of the derivation tree is $n+1$, $C = I;\ C''$ and $C' = I;\ C'''$, and we must prove that $\langle I\,;C''', \Sigma', \Psi', \mathscr{C}, \mathscr{M}\rangle$ is runtime well-formed in the context of $NonT$ and $Term'$. We have that C must be runtime well-formed in the context of $\mathscr{M}, \mathscr{C}, NonT$, $Term$, and $R = \mathrm{dom}(\Sigma)$, so there

must be some $Ful \subseteq \mathbf{StatefulRoles}$ such that

$$\langle \mathscr{M}, \mathscr{C}, Ful, NonT, Term, <:, R, I;\ C'' \rangle \checkmark \text{ is derivable}$$

where $<: = (Term \backslash \pi^{-1}_{Term,2}(Term))^+$. All well-formedness rules that could apply to $I;\ C''$, i.e., all except WF-CLOSE, also require that

$$\langle \mathscr{M}, \mathscr{C}, Ful, NonT, T, (T \backslash \pi^{-1}_{T,2}(\mathtt{0}))^+, R'', C'' \rangle \checkmark \text{ is derivable}$$

for some $T \subseteq (\mathbf{StatelessRoles} \backslash NonT) \times (\mathbf{Roles} \cup \{\mathtt{0}\})$, which depends on the specific rule, and $R'' = R \cup \pi_1(T)$. We observe that, in all cases where $T \neq Term$, the set of elements which appear in one but not the other, which we call $Diff$, is always included in $\mathrm{pn}(I)$, and we have that either $T \subseteq Term$, or $Term \subseteq T$, meaning that $Diff = Term \backslash T$, or $Diff = T \backslash Term$, respectively. Let Σ_{mod} and Ψ_{mod} be obtained by adding or removing elements to Σ and Ψ according to $Diff$, this gives us that

$$\langle C'', \Sigma_{\mathrm{mod}}, \Psi_{\mathrm{mod}}, \mathscr{C}, \mathscr{M} \rangle \xrightarrow{\mu} \langle C''', \Sigma'_{\mathrm{mod}}, \Psi'_{\mathrm{mod}}, \mathscr{C}, \mathscr{M} \rangle$$

is derivable with the same derivation tree as

$$\langle C'', \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle \xrightarrow{\mu} \langle C''', \Sigma', \Psi', \mathscr{C}, \mathscr{M} \rangle$$

since the execution of the corresponding instruction does not depend on the values of Σ_{mod} and Ψ_{mod} in the added or removed elements of their domain. Σ'_{mod} and Ψ'_{mod} are obtained by comparing Σ, Ψ with Σ', Ψ', and applying the same change to Σ_{mod} and Ψ_{mod}.

The way we constructed Σ_{mod} and Ψ_{mod} also gives us that

$$\langle C'', \Sigma_{\mathrm{mod}}, \Psi_{\mathrm{mod}}, \mathscr{C}, \mathscr{M} \rangle$$

is runtime well-formed in the context of $NonT$ and T, which, combined with the fact that the derivation tree for the corresponding transition has height

equal to n, allows us to apply our induction hypothesis to obtain that

$$\langle C''', \Sigma'_{\text{mod}}, \Psi'_{\text{mod}}, \mathscr{C}, \mathscr{M} \rangle$$

is runtime well-formed in the context of $NonT$ and T'.

Now we must prove that $\langle I\,;C''', \Sigma', \Psi', \mathscr{C}, \mathscr{M} \rangle$ is runtime well-formed in the context of $NonT$ and $Term'$.

We start by proving that C' is runtime well-formed in the context of $\mathscr{M}, \mathscr{C}, NonT, Term'$, and $R' = \text{dom}(\Sigma')$, which requires that

$$\langle \mathscr{M}, \mathscr{C}, Ful, NonT, Term', <:', R', I;\ C'''\rangle\checkmark \text{ is derivable}$$

Any derivation for this statement must end with the same rule used for C, meaning that it requires that

$$\langle \mathscr{M}, \mathscr{C}, Ful, NonT, T', (T'\backslash\pi^{-1}_{T',2}(\mathsf{0}))^+, R''', C'''\rangle\checkmark \text{ is derivable}$$

We obtain this from the fact that $\langle C''', \Sigma'_{\text{mod}}, \Psi'_{\text{mod}}, \mathscr{C}, \mathscr{M} \rangle$ is runtime well-formed in the context of $NonT$ and T', combined with Definitions 4.3 and 4.1 and our construction of $\Sigma'_{\text{mod}}, \Psi'_{\text{mod}}$. We can verify that the other requirements for the rule are maintained, and that the requirements for Definition 4.1 are maintained as well, meaning that we have proven that C' is runtime well-formed in the desired context.

Now we must prove that the other requirements for Definition 4.3 are met for $\langle I\,;C''', \Sigma', \Psi', \mathscr{C}, \mathscr{M} \rangle$ in the context of $NonT$ and $Term'$. We do this by using the fact that

$$\langle C''', \Sigma'_{\text{mod}}, \Psi'_{\text{mod}}, \mathscr{C}, \mathscr{M} \rangle$$

is runtime well-formed in the context of $NonT$ and T', combined with Definition 4.3 and our construction of $\Sigma'_{\text{mod}}, \Psi'_{\text{mod}}$. This concludes case DELAY.

♦ □

Corollary 5.6. Combining Theorem 5.5, Lemma 5.4, and rules CALL-ONLY and CALL-FIRST gives us that, in a runtime well-formed configuration, the expansion of a procedure body never causes a stateless role to be created with the same name as a stateless role currently in execution or that could be created during the rest of the choreography, i.e., it never causes name capture. Since name substitution only happens when expanding a procedure's body, this also means that, in a runtime well-formed configuration, name capture never happens.

Proof. The high level reasoning behind this informal Corollary centers around Lemma 5.4, which states, among other results, that the roles that terminate in the expanded body of the procedure never appear in the original continuation, and that the roles which did not appear as formal parameters are replaced with role names that are also not currently being used. This means that any role name which could cause name capture is replaced with a role name that is not used anywhere before adding the expanded body to the choreography. □

Theorem 5.7. Let $\langle C, \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle$ be a runtime well-formed configuration in the context of $NonT$ and $Term$. If

$$\langle C, \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle \xrightarrow{\vec{\mu}}\!\!\!\!\rightarrow \langle \mathbf{0}, \Sigma', \Psi', \mathscr{C}, \mathscr{M} \rangle,$$

we have that $\mathrm{dom}(\Sigma') = NonT \cup F$, where $F \subseteq \mathbf{StatefulRoles}$, and $\mathrm{dom}(\Psi') \subseteq NonT$, meaning that a runtime well-formed configuration terminates all terminating stateless roles it contains before reaching its conclusion. We additionally note that $\xrightarrow{\vec{\mu}}\!\!\!\!\rightarrow$ is the reflexive and transitive closure of $\xrightarrow{\vec{\mu}}$

Proof. We proceed by induction on the structure of $\vec{\mu}$

◊ Case $\vec{\mu} = \epsilon$

We have that $C = \mathbf{0}$, $\Sigma = \Sigma'$, and $\Psi = \Psi'$. By Definition 4.3 and 4.1

we have that there must be some $Ful \subseteq \textbf{StatefulRoles}$ such that

$$\langle \mathscr{M}, \mathscr{C}, Ful, NonT, Term, (Term \backslash \pi^{-1}_{Term,2}(\mathsf{0}))^+, C \rangle \checkmark \text{ is derivable}$$

Since $C = \mathbf{0}$, $\Sigma = \Sigma'$, we have that the derivation for this statement must be

$$\frac{Term = \varnothing}{\langle \mathscr{M}, \mathscr{C}, Ful, NonT, Term, <:, \mathbf{0} \rangle \checkmark} \text{ WF-CLOSE}$$

which gives us that $Term = \varnothing$, which, combined with Definition 4.3, gives us that $\mathrm{dom}(\Sigma') = NonT \cup F$, where $F \subseteq Ful \subseteq \textbf{StatefulRoles}$, and $\mathrm{dom}(\Psi') \subseteq NonT$, which concludes the case. ♦

◊ Case $\vec{\mu} = \mu', \vec{\mu}''$

We have that $C = I;\ C'$ and

$$\langle C, \Sigma, \Psi, \mathscr{C}, \mathscr{M} \rangle \xrightarrow{\mu'} \langle C'', \Sigma'', \Psi'', \mathscr{C}, \mathscr{M} \rangle \xrightarrow{\vec{\mu}''}\!\!\!\!\!\rightarrow \langle \mathbf{0}, \Sigma', \Psi', \mathscr{C}, \mathscr{M} \rangle,$$

which combined with Theorem 5.5 gives us that $\langle C'', \Sigma'', \Psi'', \mathscr{C}, \mathscr{M} \rangle$ is runtime well-formed in the context of $NonT$ and $Term'$, coherently with Σ'' and Ψ''. We apply the induction hypothesis to

$$\langle C'', \Sigma'', \Psi'', \mathscr{C}, \mathscr{M} \rangle \xrightarrow{\vec{\mu}''}\!\!\!\!\!\rightarrow \langle \mathbf{0}, \Sigma', \Psi', \mathscr{C}, \mathscr{M} \rangle$$

and we obtain our thesis, concluding case $\vec{\mu} = \mu', \vec{\mu}''$. ♦ □

Corollary 5.8. If a well-formed FaaSChalCore program reaches its conclusion, it does not leave any stateless roles in execution.

Proof. This is a direct consequence of Definition 4.5 and Theorem 5.7 □

Chapter 6

Conclusion

In this thesis, we formalised FaaSChalCore, a core choreographic calculus tailored for serverless computing, based on FaaSChal[1]. We defined FaaSChalCore's syntax and semantics by extending the formal language defined in Chapter 7 of *Introduction to Choreographies*[4] to handle serverless function instances, formalised as stateless roles in the language. We then defined a static analysis discipline, by giving definitions of well-formedness and runtime well-formedness of a choreographic configuration, extending the definition of well-formedness found in Chapter 8 of *Introduction to Choreographies*[4]. We then proved that runtime well-formedness is maintained when executing instructions, and that a well-formed program does not leave any stateless roles in execution when it terminates, i.e., all serverless function instances terminate properly before the choreography terminates. We also informally conjectured that, in a runtime well-formed configuration, name capture of stateless roles never happens.

Future development of FaaSChal can start with the definition of Endpoint Projection (EPP) for FaaSChalCore, and proving its correctness. An interesting target language for this purpose could be SKC[2], which is a calculus that formalises serverless computing, or a language based on it.

Another interesting direction for FaaSChal could be allowing stateless roles to perform communications and selections, which would remove the

need to differentiate between stateful and stateless nonterminating roles in procedure parameters, among other effects, or even allowing stateless roles to delegate responding to other stateless roles, although the latter would likely require channel mobility, like in the π-calculus[5].

Bibliography

[1] Giuseppe De Palma et al. "Towards a Function-as-a-Service Choreographic Programming Language: Examples and Applications". Presented at 1st International Workshop on Choreographic Programming, CP2024, Copenhagen, Denmark, 24 June 2024. 2024.

[2] Saverio Giallorenzo et al. "The Servers of Serverless Computing: A Formal Revisitation of Functions as a Service". In: *Recent Developments in the Design and Implementation of Programming Languages*. Ed. by Frank S. de Boer and Jacopo Mauro. Vol. 86. Open Access Series in Informatics (OASIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020, 5:1–5:21. ISBN: 978-3-95977-171-9. DOI: 10.4230/OASIcs.Gabbrielli.5. URL: https://drops.dagstuhl.de/entities/document/10.4230/OASIcs.Gabbrielli.5.

[3] Red Hat. *What is serverless?* 2025. URL: https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless.

[4] Fabrizio Montesi. *Introduction to Choreographies*. Cambridge University Press, 2023. DOI: 10.1017/9781108981491.

[5] Davide Sangiorgi and David Walker. *The Pi-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.