Reasoning about Choreographic Programs

Luís Cruz-Filipe, Eva Graversen, Fabrizio Montesi, and Marco Peressotti

Department of Mathematics and Computer Science, University of Southern Denmark

Abstract. Choreographic programming is a paradigm where a concurrent or distributed system is developed in a top-down fashion. Programs, called choreographies, detail the desired interactions between processes, and can be compiled to distributed implementations based on message passing. Choreographic languages usually guarantee deadlock-freedom and provide an operational correspondence between choreographies and their compiled implementations, but until now little work has been done on verifying other properties.

This paper presents a Hoare-style logic for reasoning about the behaviour of choreographies, and illustrate its usage in representative examples. We show that this logic is sound and complete, and discuss decidability of its judgements. Using existing results from choreographic programming, we show that any functional correctness property proven for a choreography also holds for its compiled implementation.

1 Introduction

Programming communicating systems is hard, because of the challenge of ensuring that separate communication actions (like sending or receiving a message) executed by independent programs match each other correctly at runtime [21].

In the paradigm of *choreographic programming* [26], this challenge is tackled by providing high-level abstractions that allow programmers to express the desired flow of communications safely from a 'global' viewpoint [6,8,9,13,17,18,20, 23,27]. In a choreography program, or *choreography*, communication is expressed in some variation of the communication term from security protocol notation, $\mathsf{Alice} \rightarrow \mathsf{Bob}: M$, which reads "$\mathsf{Alice}$ communicates the message M to Bob" [29]. These terms can be composed in structured choreographies using common programming language constructs. Then, a compiler can automatically generate an executable distributed implementation [6, 13, 16], as depicted in Fig. 1.

So far, research on choreographic programming has mostly focused on improving the expressivity of choreographic programming languages, their implementation, and the formalisation of general properties about compilation. Theory of choreographic programming typically comes with proofs of correctness of the accompanying compilation procedure. A hallmark result is *deadlock-freedom by design*: since mismatched communication actions cannot be syntactically expressed in choreographies, the compiled code cannot incur deadlocks [6].

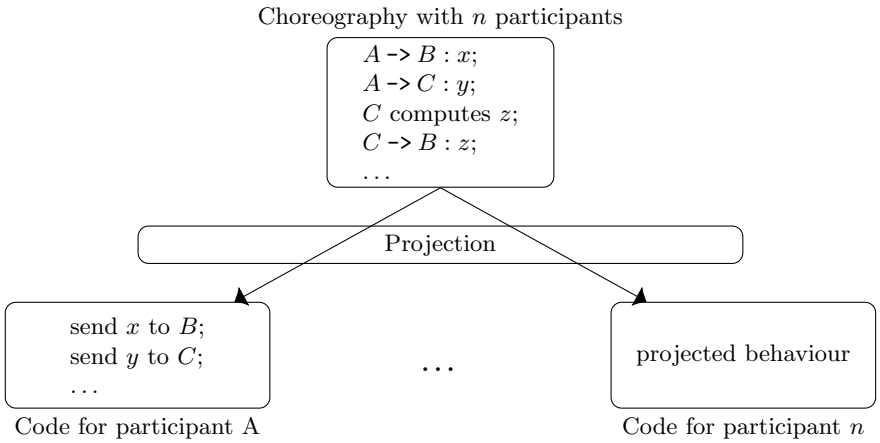

Fig. 1. Choreographic programming: the communication and computation behaviour of a system is defined in a choreography, which is then projected (compiled) to deadlock-free distributed code (adapted from [17]).

By contrast, little research has been done on general methods for proving functional correctness properties about choreographies. Yet choreographies codify distributed protocols, and reasoning about the effect that these protocols have on the states of participants is usually important.

This work. In this work, we present a Hoare logic for reasoning about choreographies. Hoare logic [2, 19] is a common way of reasoning about programs. A Hoare assertion is a triple, $\{\varphi\}P\{\psi\}$, where φ and ψ are formulas (respectively called the *precondition* and *postcondition*) and P is a program. This triple states that if P is executed from a state that satisfies φ and terminates, then the final state satisfies ψ. We develop a Hoare logic where programs are choreographies and formulas can talk about the states of multiple processes jointly.

Our framework is based on well-studied theories of choreographic programming [10,27], in particular on properties that have been formalised in Coq [11,12]. This helps with the generality and elegance of our development. For example, we leverage the property of confluence in metatheoretical proofs, and we rely on the compiler correctness results proven previously to transfer properties proven with our logic to distributed implementations compiled from choreographies.

Contribution. We define a Hoare logic for reasoning about choreographic programs expressed in standard ways, thanks to a modular design parametrised on the language of state formulas. We prove that our logic has the expected properties of a Hoare logic (soundness and partial completeness), and illustrate how it can be used to prove important properties of specific protocols encoded as choreographies.

Structure. We review the choreographic language from [10] in Section 2. In Section 3 we describe our logic and prove its soundness. Section 4 introduces weakest liberal preconditions, and uses them to show completeness and decidability results. Section 5 discusses additional related work. Illustrative examples are included throughout the text.

2 Language

In this section we recall the choreographic language from [10], which we will be reasoning about. This language models systems of independent processes (networks), which interact by means of synchronous communication. Each process is uniquely identified by a name, which is known by all other processes in the network, and can store values locally in memory referenced by variables. The set of variable names is assumed to be the same for all processes. The set of all processes is denoted by $\mathcal{P}$.

There are two kinds of messages that can be exchanged: *values* are results of evaluating *expressions* locally; and *selection labels* are special constants used to implement agreement on choices about alternative distributed behaviour.

The actual sets of expressions and labels are left unspecified, but we make some assumptions. *Labels* are taken from a (small) finite set. *Expressions* are freely generated from a (typed) signature Ξ and the set of process variables. Expressions that evaluate to a Boolean value are also called Boolean expressions.

2.1 Syntax

Formally, the syntax of choreographies is defined by the grammar

$$C ::= I; C \mid \mathsf{if}\ \mathsf{p}.b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \mid X \mid \lceil \vec{\mathsf{q}}, X \rfloor C \mid \mathbf{0}$$
$$I ::= \mathsf{p}.x := e \mid \mathsf{p}.e \to \mathsf{q}.x \mid \mathsf{p} \to \mathsf{q}[\mathrm{L}]$$

where C is a choreography, I is an instruction, p and q are processes names, e is an expression, v is a value, x is a variable, b is a Boolean expression, L is a selection label, and X is a procedure name.

Choreographies can be built as: an instruction I followed by a choreography; alternative composition of two choreographies C_1 and C_2; procedure calls; or the terminated choreography $\mathbf{0}$. There are two terms for procedure calls, corresponding to: (a) a procedure that has yet to be entered by any processes (X) or (b) one which has already started, annotated with the set of processes that still have to enter it ($\lceil \vec{\mathsf{q}}, X \rfloor C$).

There are three types of instructions: local assignment ($\mathsf{p}.x := e$), where p evaluates expression e and stores the result in its local variable x; value communication, where p evaluates e and sends the result to q, who stores it in variable x; and label selection, where p sends a label L to q (typically to communicate the result of a local choice – see below).

In a conditional, $\mathsf{if}\ \mathsf{p}.b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2$, process p evaluates the expression b to decide whether the choreography should continue as C_1 or C_2. Since only p

knows the result of the evaluation, the remaining processes need to be informed of how they should behave – this knowledge is typically propagated to other participants by means of label selections.[1]

Repetitive and iterative behaviour in this language is achieved by means of procedure calls. Calling a procedure X simply invokes the choreography corresponding to X, given in a separate mapping of procedure definitions $\mathscr{C}$. Since choreography execution is distributed, processes do not need to synchronise when entering a procedure. This requires a runtime term, $\lceil \vec{\mathsf{q}}, X \rfloor C$, to denote a procedure call that only some processes have entered. This term keeps track of both the set of processes $\vec{\mathsf{q}}$ that still need to enter X and the execution state of the choreography, C. As we show below, the semantics of choreographies allows for out-of-order execution, and consequently some processes may start executing their part of the procedure before others have entered it.

Example 1 (Diffie-Hellman). Consider the Diffie-Hellman key exchange protocol [14] which allows two parties, p and q, to establish a shared secret, s, that they can later use for symmetric encryption. To implement this protocol in our choreographic language we need only communication, local computation, and a language of expressions with modular exponentiation ($b^e \bmod m$) [16, 27]. The protocol assumes that participants have a private key each (a, b) and that they share a prime number m and a primitive root modulo m, g.

$$
\begin{array}{lll}
DH = & \mathsf{p}.(g^a \bmod m) \to \mathsf{q}.a; & \mathsf{p} \text{ computes its public key and sends it to } \mathsf{q} \\
& \mathsf{q}.(g^b \bmod m) \to \mathsf{p}.b; & \mathsf{q} \text{ computes its public key and sends it to } \mathsf{p} \\
& \mathsf{p}.s \coloneqq b^a \bmod m; & \mathsf{p} \text{ generates the shared secret} \\
& \mathsf{q}.s \coloneqq a^b \bmod m; & \mathsf{q} \text{ generates the shared secret} \\
& \mathbf{0} &
\end{array}
$$

◁

Example 2 (Zeros). Searching for a zero of a function is a common textbook example for program verification using Hoare-style logics [3]. In this example, we consider a version of the problem where p and q coordinate to find a zero of a function f over natural numbers: p is responsible for selecting the values to test and q for evaluating f and choosing whether to stop or continue searching. We capture this iterative protocol with the following recursive procedure.

$$
\begin{array}{l}
\mathscr{C}(Z) = \mathsf{p}.x \to \mathsf{q}.x; \\
\qquad \mathsf{if}\ \mathsf{q}.f(x) = 0\ \mathsf{then}\ \ (\mathsf{q} \to \mathsf{p}[\mathrm{L}]; \mathbf{0}) \\
\qquad\qquad\qquad \mathsf{else}\ \ (\mathsf{q} \to \mathsf{p}[\mathrm{R}]; \mathsf{p}.x \coloneqq 1 + x; Z)
\end{array}
$$

Then, to search the domain of f, we run the choreography $\mathsf{p}.x \coloneqq 0; Z$. ◁

We define a function pn that returns the set of processes involved in an instruction or choreography. This function is defined inductively in the natural

[1] For this reason, the set of labels is often fixed to be a two-element set, one for each branch of a choice.

way.

$$\begin{array}{ll} \mathsf{pn}(\mathsf{p}.x := e) = \{\mathsf{p}\} & \mathsf{pn}(\mathsf{p}.e \to \mathsf{q}.x) = \mathsf{pn}(\mathsf{p} \to \mathsf{q}[\mathrm{L}]) = \{\mathsf{p}, \mathsf{q}\} \\ \mathsf{pn}(I; C) = \mathsf{pn}(I) \cup \mathsf{pn}(C) & \mathsf{pn}(\mathsf{if}\ \mathsf{p}.b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2) = \{\mathsf{p}\} \cup \mathsf{pn}(C_1) \cup \mathsf{pn}(C_2) \\ \mathsf{pn}(X) = \mathcal{P} & \mathsf{pn}(\lceil \vec{\mathsf{q}}, X \rfloor C) = \vec{\mathsf{q}} \cup \mathsf{pn}(C) \end{array}$$

For simplicity we assume that all processes are involved in all procedures; an alternative is to annotate procedure names with the set of processes they use, see [12]. This does not affect the behaviour of any processes actually involved in the procedure, and semantically only means that a process which would otherwise be considered terminated may first have to enter some number of empty procedure calls.

2.2 Semantics

The semantics of choreographies uses a notion of *state*, which maps each variable at each process to the value it currently stores. It is convenient to define a *local* state as a mapping from variables to values (representing the memory state at one process), and a *global* state as a function Σ such that $\Sigma(\mathsf{p})$ is the local state at p.

To evaluate expressions, we assume that there is an evaluation function that takes a local state as parameter, evaluates variables to their value according to the state, and proceeds homeomorphically. In other words, evaluation maps each symbol in Ξ to a function from values to values. We assume that all choreographies and functions are well-typed, in the sense that the values stored in each variable match the types expected in the expressions in which they occur. Furthermore, we assume that evaluation always terminates, and write $e \downarrow_{\Sigma(\mathsf{p})} v$ to denote that e evaluates to v according to state $\Sigma(\mathsf{p})$ (local at p).

The formal semantics of choreographies is defined by means of a labelled transition system capturing the intuitions given above, whose rules are given in Fig. 2. Transitions are labelled by transition labels, which abstract from the possible choreography actions that can be observed: communications of values ($\mathsf{p}.v \to \mathsf{q}$) and labels ($\mathsf{p} \to \mathsf{q}[\mathrm{L}]$), or internal actions ($\tau@\mathsf{p}$). The function pn is naturally extended to these.

$$\mathsf{pn}(\tau@\mathsf{p}) = \{\mathsf{p}\} \qquad \mathsf{pn}(\mathsf{p}.v \to \mathsf{q}) = \mathsf{pn}(\mathsf{p} \to \mathsf{q}[\mathrm{L}]) = \{\mathsf{p}, \mathsf{q}\}$$

Rules C|Assign, C|Com, C|Sel, C|Then and C|Else capture the intuition behind the different choreographic primitives given earlier. The next three rules deal with procedure invocation: the procedure starts when one process decides to enter it, and all remaining processes are put on a "waiting list" (rule C|Call); whenever a new process enters it, it is removed from the set of waiting processes (rule C|Enter); and when the last process enters the call the set is removed (rule C|Finish).

The last three rules deal with out-of-order execution: processes can always execute what for them is the next action, regardless of what other processes are

$$\frac{e \downarrow_{\Sigma(\mathsf{p})} v}{\langle \mathsf{p}.x := e; C, \Sigma\rangle \xrightarrow{\tau@\mathsf{p}}_{\mathscr{C}} \langle C, \Sigma[\langle \mathsf{p}, x\rangle \mapsto v]\rangle}\ \text{C|Assign}$$

$$\frac{e \downarrow_{\Sigma(\mathsf{p})} v}{\langle \mathsf{p}.e \to \mathsf{q}.x; C, \Sigma\rangle \xrightarrow{\mathsf{p}.v \to \mathsf{q}}_{\mathscr{C}} \langle C, \Sigma[\langle \mathsf{q}, x\rangle \mapsto v]\rangle}\ \text{C|Com}$$

$$\frac{}{\langle \mathsf{p} \to \mathsf{q}[\text{L}]; C, \Sigma\rangle \xrightarrow{\mathsf{p} \to \mathsf{q}[\text{L}]}_{\mathscr{C}} \langle C, \Sigma\rangle}\ \text{C|Sel} \qquad \frac{b \downarrow_{\Sigma(\mathsf{p})} \mathsf{true}}{\langle \mathsf{if}\ \mathsf{p}.b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2, \Sigma\rangle \xrightarrow{\tau@\mathsf{p}}_{\mathscr{C}} \langle C_1, \Sigma\rangle}\ \text{C|Then}$$

$$\frac{b \downarrow_{\Sigma(\mathsf{p})} \mathsf{false}}{\langle \mathsf{if}\ \mathsf{p}.b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2, \Sigma\rangle \xrightarrow{\tau@\mathsf{p}}_{\mathscr{C}} \langle C_2, \Sigma\rangle}\ \text{C|Else}$$

$$\frac{\mathscr{C}(X) = C}{\langle X, \Sigma\rangle \xrightarrow{\tau@\mathsf{r}}_{\mathscr{C}} \langle \lceil \mathsf{pn}(C) \setminus \mathsf{r}, X \rfloor C, \Sigma\rangle}\ \text{C|Call}$$

$$\frac{\mathsf{r} \in \vec{\mathsf{q}} \quad \vec{\mathsf{q}} \setminus \mathsf{r} \neq \emptyset}{\langle \lceil \vec{\mathsf{q}}, X \rfloor C, \Sigma\rangle \xrightarrow{\tau@\mathsf{r}}_{\mathscr{C}} \langle \lceil \vec{\mathsf{q}} \setminus \mathsf{r}, X \rfloor C, \Sigma\rangle}\ \text{C|Enter} \qquad \frac{}{\langle \lceil \mathsf{q}, X \rfloor C, \Sigma\rangle \xrightarrow{\tau@\mathsf{q}}_{\mathscr{C}} \langle C, \Sigma\rangle}\ \text{C|Finish}$$

$$\frac{\langle C, \Sigma\rangle \xrightarrow{\mu}_{\mathscr{C}} \langle C', \Sigma'\rangle \quad \mathsf{pn}(I) \# \mathsf{pn}(\mu)}{\langle I; C, \Sigma\rangle \xrightarrow{\mu}_{\mathscr{C}} \langle I; C', \Sigma'\rangle}\ \text{C|DelayI}$$

$$\frac{\langle C_1, \Sigma\rangle \xrightarrow{\mu}_{\mathscr{C}} \langle C_1', \Sigma'\rangle \quad \langle C_2, \Sigma\rangle \xrightarrow{\mu}_{\mathscr{C}} \langle C_2', \Sigma'\rangle \quad \mathsf{p} \notin \mathsf{pn}(\mu)}{\langle \mathsf{if}\ \mathsf{p}.b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2, \Sigma\rangle \xrightarrow{\mu}_{\mathscr{C}} \langle \mathsf{if}\ \mathsf{p}.b\ \mathsf{then}\ C_1'\ \mathsf{else}\ C_2', \Sigma'\rangle}\ \text{C|DelayC}$$

$$\frac{\langle C, \Sigma\rangle \xrightarrow{\mu}_{\mathscr{C}} \langle C', \Sigma'\rangle \quad \vec{\mathsf{q}} \# \mathsf{pn}(\mu)}{\langle \lceil \vec{\mathsf{q}}, X \rfloor C, \Sigma\rangle \xrightarrow{\mu}_{\mathscr{C}} \langle \lceil \vec{\mathsf{q}}, X \rfloor C', \Sigma'\rangle}\ \text{C|DelayP}$$

Fig. 2. Semantics

doing. This is modelled by rules C|DelayI, C|DelayC and C|DelayP, which allow execution of an action that is not syntactically the first instruction, conditional or procedure entering, respectively. The side conditions in these rules state that the processes involved in the action being executed do not participate in the actions being skipped (we write $X \# Y$ for $X \cap Y = \emptyset$). Additionally, the action being performed in C|DelayC must be an action that can be made regardless of what p chooses.

The reflexive and transitive closure of transition is denoted by $\to^*_{\mathscr{C}}$; we omit the sequente of transition labels, as this is immaterial for the current presentation.

For our proofs we also need the concept of *head transition*, which is the transition relation defined by the first 8 rules in Fig. 2 – that is, disallowing

out-of-order execution. We write $\langle C, \Sigma\rangle \overset{\mu}{\Rightarrow}_{\mathscr{C}} \langle C', \Sigma'\rangle$ to denote that C makes a head transition to C', and $\Rightarrow^*_{\mathscr{C}}$ for the reflexive and transitive closure of this relation.

3 A Hoare calculus for choreographies

In this section we introduce our formal calculus for proving semantic properties of choreographies based on Hoare logic. Our judgements are triples $\{\varphi\}C\{\psi\}$, interpreted as "if choreography C is executed from a state satisfying formula φ and execution terminates, then the final state satisfies formula ψ".

In this section we formally define the syntax and the semantics of this calculus, starting with the state logic – the language in which formulas φ and ψ are written.

3.1 State logic

State logics in Hoare calculi typically express properties as "variable x stores a value v", which are easily expressible in equational logic. We follow this tradition, and define our state logic to be an extension of equational logic. In order to deal with assignments, we need to be able to update formulas in a way that corresponds to the state update in rule C|Assign – but without computing values. This can be achieved by substituting the expression communicated in the original formula – but this means that expressions may suddenly refer to variables stored in different processes, so that they are no longer evaluated locally.

To deal with these issues, our state logic is parameterised on a set of expressions that is freely generated from the same signature Ξ, but using localised variables $\mathsf{p}.x$. We denote these expressions as $\mathcal{E}$, and extend evaluation to them in the natural way.

State formulas are defined as

$$\varphi, \psi ::= (\mathcal{E} = \mathcal{X}) \mid \delta \mid \varphi \wedge \varphi \mid \neg\varphi$$

where $\mathcal{X}$ is a (logical) variable and $\delta \in \mathfrak{D}$, where $\mathfrak{D}$ is a decidable theory whose terms include the logical variables. Parameterising the language on $\mathfrak{D}$ keeps the syntax of formulas simpler, while giving the user flexibility to define additional needed formulas. This is similar to our treatment of the local language. For example, if $\mathfrak{D}$ includes $\mathcal{X} > \mathcal{X}'$, then the state logic is able to express constraints such as $\mathsf{p}.x > \mathsf{q}.y$, assuming values are integers: this can be written as $\mathsf{p}.x = \mathcal{X} \wedge \mathsf{q}.y = \mathcal{Y} \wedge \mathcal{X} > \mathcal{Y}$. Disjunction and implication are defined as abbreviations in the usual way.

Given a state Σ, a formula φ and an assignment ρ from logical variables to values, we define $\Sigma \Vdash_\rho \varphi$, read "$\Sigma$ satisfies φ under ρ", by the rules

$$\frac{\mathcal{E} \downarrow_\Sigma \rho(\mathcal{X})}{\Sigma \Vdash_\rho \mathcal{E} = \mathcal{X}} \qquad \frac{\delta \in \mathfrak{D} \quad \varphi \text{ is true}}{\Sigma \Vdash_\rho \delta}$$

together with the usual rules for logical connectives.

As usual in Hoare logics, assignment is dealt with using substitution – for example, we expect to be able to prove something like

$$\overline{\{\varphi'\}\mathsf{p}.x \coloneqq e; \mathbf{0}\{\varphi\}}$$

where φ' is obtained by φ by substituting $\mathsf{p}.x$ with e. However, simply replacing every occurrence of $\mathsf{p}.x$ with e yields in general an invalid formula (due to the different variables in choreographies and state formulas). We define the *localisation of e at p*, $L(\mathsf{p}, e)$, as the (logical) expression obtained from e by replacing every (choreography) variable x with $\mathsf{p}.x$; and the *localised substitution* $\mathcal{E}[\mathsf{q}.x := \mathsf{p}.e]$ as the expression obtained from $\mathcal{E}$ by replacing every occurrence of $\mathsf{q}.x$ with $L(\mathsf{p}, e)$. (The rule for communication uses different values for p and q.) Observe that these operations can both be defined by structural recursion on expressions. Localised substitution extends to formulas in the natural way.

Example 3. Take φ to be the formula $\mathsf{p}.x > 3$ and e to be the expression $y - z$. Replacing $\mathsf{p}.x$ with $y - z$ in φ would yield the ill-formed formula $\mathsf{p}.(y - z) > 3$. Instead, replacing $\mathsf{p}.x$ with $L(\mathsf{p}, y - z) = \mathsf{p}.y - \mathsf{p}.z$ yields the right formula $\mathsf{p}.y - \mathsf{p}.z > 3$, and the above judgement becomes

$$\overline{\{\mathsf{p}.y - \mathsf{p}.z > 3\}\mathsf{p}.x := y - z; \mathbf{0}\{\mathsf{p}.x > 3\}}$$

which is syntactically well-formed. ◁

We now show that an expression that has been localised to p is interpreted as its original evaluation in p.

Lemma 1. *Let Σ be a state, v be a value, $\mathcal{X}$ be a logical variable and ρ be an assignment such that $\rho(\mathcal{X}) = v$. For any process p and expression e, $e \downarrow_{\Sigma(\mathsf{p})} v$ iff $\Sigma \Vdash_\rho L(\mathsf{p}, e) = \mathcal{X}$.*

Proof. Follows from induction on the structure of e. □

We then show that doing a localised substitution in a formula is equivalent to changing the value of that variable in the environment.

Corollary 1. *Let Σ be a state, p be a process, e be an expression and v be a value such that $e \downarrow_{\Sigma(\mathsf{p})} v$. For any formula φ and assignment ρ, $\Sigma[\langle \mathsf{p}, x\rangle \mapsto v] \Vdash_\rho \varphi$ iff $\Sigma \Vdash_\rho \varphi[\mathsf{q}.x := \mathsf{p}.e]$.*

Proof. By structural induction on φ. One of the base cases is simply Lemma 1, while the other is trivially empty (since formulas in $\mathfrak{D}$ are not affected by substitution). The two inductive cases follow directly by induction hypothesis. □

$$\frac{}{\vdash_{\mathfrak{C}} \{\varphi\}\mathbf{0}\{\varphi\}}\ \text{H|Nil} \qquad \frac{\vdash_{\mathfrak{C}} \{\varphi\}C\{\varphi'\}}{\vdash_{\mathfrak{C}} \{\varphi[\mathsf{p}.x := \mathsf{p}.e]\}\mathsf{p}.x := e; C\{\varphi'\}}\ \text{H|Assign}$$

$$\frac{\vdash_{\mathfrak{C}} \{\varphi\}C\{\varphi'\}}{\vdash_{\mathfrak{C}} \{\varphi[\mathsf{q}.x := \mathsf{p}.e]\}\mathsf{p}.e \to \mathsf{q}.x; C\{\varphi'\}}\ \text{H|Com} \qquad \frac{\vdash_{\mathfrak{C}} \{\varphi\}C\{\varphi'\}}{\vdash_{\mathfrak{C}} \{\varphi\}\mathsf{p} \to \mathsf{q}[\mathrm{L}]; C\{\varphi'\}}\ \text{H|Sel}$$

$$\frac{\vdash_{\mathfrak{C}} \{\varphi \wedge L(\mathsf{p}, b) \overset{\mathcal{X}}{=} \mathsf{true}\}C_1\{\psi\} \quad \vdash_{\mathfrak{C}} \{\varphi \wedge L(\mathsf{p}, b) \overset{\mathcal{X}}{=} \mathsf{false}\}C_2\{\psi\} \quad \mathcal{X} \text{ fresh}}{\vdash_{\mathfrak{C}} \{\varphi\}\mathsf{if}\ \mathsf{p}.b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\{\psi\}}\ \text{H|Cond}$$

$$\frac{\mathfrak{C}(X) = \langle \varphi, \psi \rangle}{\vdash_{\mathfrak{C}} \{\varphi\}X\{\psi\}}\ \text{H|Call} \qquad \frac{\vdash_{\mathfrak{C}} \{\varphi\}C\{\psi\}}{\vdash_{\mathfrak{C}} \{\varphi\}\lceil \vec{\mathsf{q}}, X \rfloor C\{\psi\}}\ \text{H|Call'}$$

$$\frac{\mathfrak{D} \models \varphi \to \varphi' \quad \vdash_{\mathfrak{C}} \{\varphi'\}C\{\psi'\} \quad \mathfrak{D} \models \psi' \to \psi}{\vdash_{\mathfrak{C}} \{\varphi\}C\{\psi\}}\ \text{H|Weak}$$

Fig. 3. Inference rules

3.2 Hoare logic

We are now ready to introduce the rules for our calculus, which are depicted in Fig. 3. To deal with procedure definitions, we need additional information about their effect on states. This is achieved by the *procedure specification map* $\mathfrak{C}$, which maps each procedure name to a pair $\langle \varphi, \psi \rangle$ with intended meaning that the judgement $\{\varphi\}C\{\psi\}$ should hold, where C is the definition of X.

The rule for assignment H|Assign has already been motivated earlier, and is similar to the rule in standard Hoare calculi for imperative programs; likewise, rules H|Nil and H|Cond are also standard. The notation $L(\mathsf{p}, b) \overset{\mathcal{X}}{=} \mathsf{true}$ in rule H|Cond abbreviates the conjunction $L(\mathsf{p}, b) = \mathcal{X} \wedge \mathcal{X} = \mathsf{true}$.

Rule H|Weak is a weakening rule, which allows us to include reasoning in the state logic. The notation $\mathfrak{D} \models \varphi$ stands for "φ is a valid formula".

Rules H|Com and H|Sel adapt the intuitions behind those rules to our choreography actions — a communication is essentially an assignment of a variable located at a different process, while selection does not affect the state.

Rule H|Call deals with unexpanded procedure calls by reading the corresponding judgement from the specification map, while H|Call' reflects the fact that the current state of the expanded procedure is explicitly given and a process entering a procedure does not affect the state.

These rules only make sense if the specification map is consistent with the procedure definitions in the following sense.

Definition 1. *A procedure specification map* $\mathfrak{C}$ *is* consistent *with a set of procedure definitions* $\mathscr{C}$ *if* $\vdash_{\mathfrak{C}} \{\mathsf{fst}(\mathfrak{C}(X))\}\mathscr{C}(X)\{\mathsf{snd}(\mathfrak{C}(X))\}$ *for every* X*, where* fst *and* snd *are the standard projection operators for pairs.*

This notion plays a similar role to the more usual concept of "being a loop invariant" in Hoare logics for languages with while-loops, stating that $\mathsf{fst}(\mathfrak{C}(X))$ always holds whenever X is called.

Example 4 (Diffie-Hellman, functional correctness). Consider Example 1, and assume $\mathfrak{D}$ is a theory for deciding equality of arithmetic expressions with modular exponentiation. Functional correctness for the Diffie-Hellman protocol, states if p and q have the same modulus m and base g then they will share the same secret s once the protocol terminates. These pre- and postconditions are captured by the following state formulas $\varphi = (\mathsf{p}.g \overset{\mathcal{G}}{=} \mathsf{q}.g \wedge \mathsf{p}.m \overset{\mathcal{M}}{=} \mathsf{q}.m)$ and $\psi = \mathsf{p}.s \overset{\mathcal{S}}{=} \mathsf{q}.s$. Thus, we can show the correctness of DH by deriving $\vdash \{\varphi\}DH\{\psi\}$:

$$\dfrac{\mathfrak{D} \models \varphi \to \varphi_1 \quad \dfrac{\vdash \{\varphi_1\}\mathsf{p}.(g^a \bmod m) \to \mathsf{q}.a; \ldots\{\psi\}}{\dfrac{\vdash \{\varphi_2\}\mathsf{q}.(g^b \bmod m) \to \mathsf{p}.b; \ldots\{\psi\}}{\dfrac{\vdash \{\varphi_3\}\mathsf{p}.s := b^a \bmod m; \ldots\{\psi\}}{\dfrac{\vdash \{\varphi_4\}\mathsf{q}.s := a^b \bmod m; \mathbf{0}\{\psi\}}{\dfrac{}{\vdash \{\psi\}\mathbf{0}\{\psi\}}\,\mathrm{H|Nil}}\,\mathrm{H|Assign}}\,\mathrm{H|Assign}}\,\mathrm{H|Com}}\,\mathrm{H|Com}}{\vdash \{\varphi\}DH\{\psi\}}\,\mathrm{H|Weak}$$

where:

$$\begin{aligned}
\varphi_1 &= \varphi_2[\mathsf{q}.a := \mathsf{p}.g^a \bmod m] \\
&= (\mathsf{q}.g^{\mathsf{q}.b} \bmod \mathsf{q}.m)^{\mathsf{p}.a} \bmod \mathsf{p}.m \overset{\mathcal{S}}{=} (\mathsf{p}.g^{\mathsf{p}.a} \bmod \mathsf{p}.m)^{\mathsf{q}.b} \bmod \mathsf{q}.m \\
\varphi_2 &= \varphi_3[\mathsf{p}.b := \mathsf{q}.g^b \bmod m] = (\mathsf{q}.b^{\mathsf{q}.b} \bmod \mathsf{q}.m)^{\mathsf{p}.a} \bmod \mathsf{p}.m \overset{\mathcal{S}}{=} \mathsf{q}.a^{\mathsf{q}.b} \bmod \mathsf{q}.m \\
\varphi_3 &= \varphi_4[\mathsf{p}.s := \mathsf{p}.b^a \bmod m] = \mathsf{p}.b^{\mathsf{p}.a} \bmod \mathsf{p}.m \overset{\mathcal{S}}{=} \mathsf{q}.a^{\mathsf{q}.b} \bmod \mathsf{q}.m \\
\varphi_4 &= \psi[\mathsf{q}.s := \mathsf{q}.a^b \bmod m] = \mathsf{p}.s \overset{\mathcal{S}}{=} \mathsf{q}.a^{\mathsf{q}.b} \bmod \mathsf{q}.m
\end{aligned}$$ ◁

We can now show that this calculus is sound, in the sense that it only derives valid judgements. Given confluence of the transition system for the semantics of choreographies [12], it suffices to show that this holds for head transitions: if execution terminates, any path of execution must lead to the same final state.

Lemma 2. *Assume that $\mathfrak{C}$ is consistent with $\mathscr{C}$ and that $\vdash_{\mathfrak{C}} \{\varphi\}C\{\psi\}$. For every state Σ and assignment ρ, if $\Sigma \Vdash_\rho \varphi$ and $\langle C, \Sigma\rangle \Rightarrow^*_{\mathscr{C}} \langle \mathbf{0}, \Sigma'\rangle$, then $\Sigma' \Vdash_\rho \psi$.*

Proof. The proof is by induction on the number of transitions from $\langle C, \Sigma\rangle$ to $\langle \mathbf{0}, \Sigma'\rangle$. Within each case, we use induction on the size of the derivation of $\vdash_{\mathfrak{C}} \{\varphi\}C\{\psi\}$. We include some representative cases.

- If the number of transitions is 0, then $C = \mathbf{0}$ and $\Sigma = \Sigma'$. The derivation of $\vdash_{\mathfrak{C}} \{\varphi\}\mathbf{0}\{\psi\}$ must then end with an application of H|Nil – which implies that $\psi = \varphi$, establishing the thesis – or of H|Weak – and the induction hypothesis together with soundness of $\mathfrak{D}$ establishes the thesis.

– Assume that $\langle C, \Sigma\rangle \xrightarrow{\tau@\mathsf{p}}_{\mathscr{C}} \langle C', \Sigma'\rangle \to^*_{\mathscr{C}} \langle C'', \Sigma''\rangle$ and that the first transition is derived by rule C|Assign. Then C has the form $\mathsf{p}.x \coloneqq e; C'$, $e \downarrow_{\Sigma(\mathsf{p})} v$, and $\Sigma' = \Sigma[\langle \mathsf{p}, x\rangle \mapsto v]$. There are two cases, depending on the last rule applied in the derivation of $\vdash_{\mathfrak{C}} \{\varphi\}C\{\psi\}$.
  If the derivation terminates with an application of H|Assign, then φ is $\varphi'[\mathsf{p}.x := \mathsf{p}.e]$ for some formula φ' such that $\vdash_{\mathfrak{C}} \{\varphi'\}C'\{\psi\}$. By Corollary 1 it follows that $\Sigma' \Vdash_\rho \varphi'$, and the induction hypothesis applied to C' establishes the thesis.
  If the derivation terminates with an application of H|Weak, then the thesis is established by the induction hypothesis over the derivation, as in the base case.
– Assume that $\langle C, \Sigma\rangle \xrightarrow{\tau@\mathsf{p}}_{\mathscr{C}} \langle C', \Sigma'\rangle \to^*_{\mathscr{C}} \langle C'', \Sigma''\rangle$ and that the first transition is derived by rule C|Call. Then C has the form $X, \lceil \mathsf{pn}(C) \setminus \mathsf{r}, X\rfloor \mathscr{C}(X)$ and $\Sigma' = \Sigma$. Again there are two cases, depending on the last rule applied in the derivation of $\vdash_{\mathfrak{C}} \{\varphi\}C\{\psi\}$.
  If the derivation terminates with an application of H|Call, then by consistency of $\mathfrak{C}$ and $\mathscr{C}$ we know that $\vdash_{\mathfrak{C}} \{\varphi\}\mathscr{C}(X)\{\psi\}$, from which we can infer (using H|Call') that also $\vdash_{\mathfrak{C}} \{\varphi\}\lceil \mathsf{pn}(C) \setminus \mathsf{r}, X\rfloor \mathscr{C}(X)\{\psi\}$. The induction hypothesis applies to this choreography to establish the thesis.
  If the derivation terminates with an application of H|Weak, then the thesis is established as in the previous cases. □

Theorem 1 (Soundness). *Assume that $\mathfrak{C}$ is consistent with $\mathscr{C}$ and we have $\vdash_{\mathfrak{C}} \{\varphi\}C\{\psi\}$. For every state Σ and assignment ρ, if $\Sigma \Vdash_\rho \varphi$ and $\langle C, \Sigma\rangle \to^*_{\mathscr{C}} \langle \mathbf{0}, \Sigma'\rangle$, then $\Sigma' \Vdash_\rho \psi$.*

Proof. By the results in [12], if $\langle C, \Sigma\rangle \to^*_{\mathscr{C}} \langle \mathbf{0}, \Sigma'\rangle$ then also $\langle C, \Sigma\rangle \Rightarrow^*_{\mathscr{C}} \langle \mathbf{0}, \Sigma'\rangle$ (combining deadlock-freedom with confluence). Lemma 2 then establishes the thesis. □

Example 5 (Zeros, functional correctness). Correctness for the program from Example 2 requires that if f has a zero, the program terminates finding it or, equivalently, that the postcondition $\psi = ((f(\mathsf{p}.x) = 0) \overset{\mathcal{Z}}{=} \mathsf{true})$ holds. Since there are no hypothesis on the initial state, we can use as a precondition ϕ any tautology (preferably one without occurrences of variables used in the program) e.g., $\varphi = (\mathsf{true} \overset{\mathcal{T}}{=} \mathsf{true})$. The following derivation shows that the procedure specification map $\mathfrak{C}(Z) = \langle \varphi, \psi\rangle$ is consistent with $\mathscr{C}$ from Example 2:

$$\dfrac{\mathfrak{D} \models \varphi \to \varphi_1 \qquad \dfrac{\vdash_{\mathfrak{C}} \{\varphi_1\}\mathsf{p}.x \to \mathsf{q}.x; \ldots\{\psi\} \qquad \dfrac{\vdash_{\mathfrak{C}} \{\varphi_2\}\mathsf{if}\ \mathsf{q}.f(x) = 0\ \mathsf{then} \ldots \mathsf{else} \ldots\{\psi\} \qquad \dfrac{\dfrac{\dfrac{}{\vdash_{\mathfrak{C}} \{\psi\}\mathbf{0}\{\psi\}}\text{H|Nil}}{\vdash_{\mathfrak{C}} \{\psi\}\mathsf{q} \to \mathsf{p}[\mathrm{L}]; \mathbf{0}\{\psi\}}\text{H|Sel} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\mathfrak{C}(Z) = \langle \varphi, \psi\rangle}{\vdash_{\mathfrak{C}} \{\varphi\}Z\{\psi\}}\text{H|Call}}{\vdash_{\mathfrak{C}} \{\varphi\}\mathsf{p}.x \coloneqq x + 1; Z\{\psi\}}\text{H|Assign}}{\vdash_{\mathfrak{C}} \{\varphi\}\mathsf{q} \to \mathsf{p}[\mathrm{R}]; \ldots\{\psi\}}\text{H|Sel}}\text{H|Cond}}{\vdash_{\mathfrak{C}} \{\varphi_2\}\mathsf{if}\ \mathsf{q}.f(x) = 0\ \mathsf{then} \ldots \mathsf{else} \ldots\{\psi\}}\text{H|Com}}{\vdash_{\mathfrak{C}} \{\varphi\}\mathscr{C}(Z)\{\psi\}}\text{H|Weak}$$

where:

$$\varphi_1 = ((f(\mathsf{p}.x) = 0) \overset{Z}{=} \mathsf{true} \to \psi) \wedge ((f(\mathsf{p}.x) = 0) \overset{Z}{=} \mathsf{false} \to \varphi)$$
$$\varphi_2 = ((f(\mathsf{q}.x) = 0) \overset{Z}{=} \mathsf{true} \to \psi) \wedge ((f(\mathsf{q}.x) = 0) \overset{Z}{=} \mathsf{false} \to \varphi)$$

The same pre- and postconditions hold for the whole program:

$$\dfrac{\dfrac{\mathfrak{C}(Z) = \langle \varphi, \psi \rangle}{\vdash_{\mathfrak{C}} \{\varphi\} Z \{\psi\}}\ \mathrm{H|CALL}}{\vdash_{\mathfrak{C}} \{\varphi\} \mathsf{p}.x \coloneqq 0; Z\{\psi\}}\ \mathrm{H|ASSIGN}$$

If follows from soundness, that any terminating execution ends in a state Σ s.t., $f(x) = 0 \downarrow_{\Sigma(\mathsf{p})} \mathsf{true}$. Termination follows by observing that p scans natural numbers starting from 0 proceeding by single increments and thus, if f has any zero, p will eventually send the first of them to q which in turn will choose to terminate the search. ◁

4 Completeness of the Hoare calculus

To establish a completeness result for our calculus, we follow standard techniques from the literature, by using a notion of *weakest liberal precondition* – the weakest assertion φ, given $\mathfrak{C}$, C and ψ, such that $\vdash_{\mathfrak{C}} \{\varphi\} C \{\psi\}$.

4.1 Weakest liberal preconditions

In this section we define the weakest liberal precondition operator and show that it satisfies the expected properties.

Definition 2. *Let C be a choreography, ψ be a formula and $\mathfrak{C}$ be a procedure specification map. The* weakest liberal precondition *for C and ψ under $\mathfrak{C}$, $\mathsf{wlp}_{\mathfrak{C}}(C, \psi)$, is defined as follows.*

$$\begin{aligned}
\mathsf{wlp}_{\mathfrak{C}}((\mathsf{p}.x \coloneqq e; C), \psi) &= \mathsf{wlp}_{\mathfrak{C}}(C, \psi)[\mathsf{p}.x := \mathsf{p}.e] \\
\mathsf{wlp}_{\mathfrak{C}}((\mathsf{p}.e \to \mathsf{q}.x; C), \psi) &= \mathsf{wlp}_{\mathfrak{C}}(C, \psi)[\mathsf{q}.x := \mathsf{p}.e] \\
\mathsf{wlp}_{\mathfrak{C}}((\mathsf{p} \to \mathsf{q}[\mathrm{L}]; C), \psi) &= \mathsf{wlp}_{\mathfrak{C}}(C, \psi) \\
\mathsf{wlp}_{\mathfrak{C}}(\mathsf{if}\ \mathsf{p}.b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2, \psi) &= (L(\mathsf{p}, b) \overset{\mathcal{X}}{=} \mathsf{true} \to \mathsf{wlp}_{\mathfrak{C}}(C_1, \psi)) \\
&\quad \wedge (L(\mathsf{p}, b) \overset{\mathcal{X}}{=} \mathsf{false} \to \mathsf{wlp}_{\mathfrak{C}}(C_2, \psi)) \\
\mathsf{wlp}_{\mathfrak{C}}(X, \psi) &= \mathsf{fst}(\mathfrak{C}(X)) \\
\mathsf{wlp}_{\mathfrak{C}}(\lceil \vec{\mathsf{q}}, X \rfloor C, \psi) &= \mathsf{wlp}_{\mathfrak{C}}(C, \psi) \\
\mathsf{wlp}_{\mathfrak{C}}(\mathbf{0}, \psi) &= \psi
\end{aligned}$$

This operator is essentially mimicking the rules from Figure 3. In the clause for conditionals, $\mathcal{X}$ is fresh. The only potentially surprising item is the definition

of $\mathsf{wlp}_{\mathfrak{C}}(X, \psi)$, which ignores the actual formula ψ: this is again due to the fact that our results require an additional condition on $\mathfrak{C}$ (namely, that the conditions given are compatible with the definition of $\mathsf{wlp}_{\mathfrak{C}}$), which indirectly ensures that ψ is also considered.

Example 6 (Diffie-Hellman, WLP). Consider the choreography DH from Example 1 and the postcondition $\psi = (\mathsf{p}.s \stackrel{\mathcal{S}}{=} \mathsf{q}.s)$ from Example 4, $\mathsf{wlp}(DH, \psi)$ is the formula φ_1 from Example 4. ◁

Definition 3. *A procedure specification map $\mathfrak{C}$ is* adequate for *ψ given a set of procedure definitions $\mathscr{C}$ if, for any procedure name X, $\mathsf{fst}(\mathfrak{C}(X))$ is logically equivalent to $\mathsf{wlp}_{\mathfrak{C}}(\mathscr{C}(X), \psi)$ and $\mathsf{snd}(\mathfrak{C}(X)) = \psi$.*

In other words, for each ψ we are interested in a mapping $\mathfrak{C}$ that, for each procedure, includes the right precondition that ensures that ψ will hold if that procedure terminates.

Example 7 (Zeros, WLP). The procedure specification map $\mathfrak{C}$ from Example 5 is adequate for the postcondition from the same example given the set of procedure definitions $\mathscr{C}$ from Example 2. In fact, $\mathsf{wlp}_{\mathfrak{C}}(\mathscr{C}(Z), f(\mathsf{p}.x) = 0 \stackrel{\mathcal{Z}}{=} \mathsf{true})$ is the formula φ_1 from Example 5, which is logically equivalent to $\mathsf{fst}(\mathfrak{C}(Z))$. ◁

The next results show that $\mathsf{wlp}_{\mathfrak{C}}(C, \psi)$ precisely characterises the set of states from which execution of C guarantees ψ.

Lemma 3. *Assume that $\mathfrak{C}$ is adequate for ψ given $\mathscr{C}$. Then, for every choreography C, $\vdash_{\mathfrak{C}} \{\mathsf{wlp}_{\mathfrak{C}}(C, \psi)\} C \{\psi\}$.*

Proof. By structural induction on C. Most cases immediately follow from the definition of $\mathsf{wlp}_{\mathfrak{C}}$ together with the induction hypothesis. We detail the only nontrivial ones.

- If C is $\mathsf{if}\ \mathsf{p}.b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2$, we observe that $\vdash_{\mathfrak{C}} \{\mathsf{wlp}_{\mathfrak{C}}(C_1, \psi)\} C_1 \{\psi\}$. Since

$$(\mathsf{wlp}_{\mathfrak{C}}(\mathsf{if}\ \mathsf{p}.b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2, \psi) \wedge L(\mathsf{p}, b) \stackrel{\mathcal{X}}{=} \mathsf{true}) \rightarrow \mathsf{wlp}_{\mathfrak{C}}(C_1, \psi)$$

  is a valid propositional formula, we can apply rule H|Weak to derive $\vdash_{\mathfrak{C}} \{\mathsf{wlp}_{\mathfrak{C}}(\mathsf{if}\ \mathsf{p}.b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2, \psi) \wedge L(\mathsf{p}, b) \stackrel{\mathcal{X}}{=} \mathsf{true}\} C_1 \{\psi\}$. A similar reasoning applied to C_2 derives the other hypothesis for rule H|Cond, and combining them establishes the thesis.
- If C is X, then the thesis follows from the assumption that $\mathsf{snd}(\mathfrak{C}(X)) = \psi$.

□

Corollary 2. *If $\mathfrak{C}$ is adequate for ψ given $\mathscr{C}$, then $\mathfrak{C}$ is consistent with $\mathscr{C}$.*

Corollary 3. *Assume that $\mathfrak{C}$ is adequate for ψ given $\mathscr{C}$. For every choreography C, state Σ, and assignment ρ, if $\Sigma \Vdash_{\rho} \mathsf{wlp}_{\mathfrak{C}}(C, \psi)$ and $\langle C, \Sigma \rangle \rightarrow^*_{\mathscr{C}} \langle \mathbf{0}, \Sigma' \rangle$ for some state Σ', then $\Sigma' \Vdash_{\rho} \psi$.*

Proof. By Lemma 3, $\vdash_{\mathfrak{C}} \{\mathsf{wlp}_{\mathfrak{C}}(C,\psi)\}C\{\psi\}$. By Corollary 2, $\mathfrak{C}$ is consistent with $\mathscr{C}$. The thesis then follows by Theorem 1. □

Lemma 4. *Assume that $\mathfrak{C}$ is adequate for ψ given $\mathscr{C}$. Let C be a choreography, Σ and Σ' be states, and ρ be an assignment. If $\langle C,\Sigma\rangle \Rightarrow^*_{\mathscr{C}} \langle \mathbf{0},\Sigma'\rangle$ and $\Sigma' \Vdash_\rho \psi$, then $\Sigma \Vdash_\rho \mathsf{wlp}_{\mathfrak{C}}(C,\psi)$.*

Proof. By induction on the number of transitions from C to $\mathbf{0}$. If this number is 0, then C is $\mathbf{0}$ and the thesis trivially follows. Otherwise, we detail some representative cases. We do case analysis on C to determine the first transition.

- If C is $\mathsf{p}.x \coloneqq e; C''$, then $\langle C,\Sigma\rangle \xrightarrow{\tau@\mathsf{p}}_{\mathscr{C}} \langle C'',\Sigma''\rangle \Rightarrow^*_{\mathscr{C}} \langle \mathbf{0},\Sigma'\rangle$, and $\Sigma'' \Vdash_\rho \mathsf{wlp}_{\mathfrak{C}}(C'',\psi)$ by induction hypothesis. But $\Sigma'' = \Sigma[\langle \mathsf{p},x\rangle \mapsto v]$ where $e \downarrow_{\Sigma(\mathsf{p})} v$, hence $\Sigma \Vdash_\rho \mathsf{wlp}_{\mathfrak{C}}(C'',\psi)[\mathsf{p}.x \coloneqq \mathsf{p}.e]$ by Corollary 1, establishing the thesis.
- If C is $\mathsf{if}\ \mathsf{p}.b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2$, then there are two cases. Assume wlog that $b \downarrow_{\Sigma(\mathsf{p})} \mathsf{true}$. Then $\langle \mathsf{if}\ \mathsf{p}.b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2,\Sigma\rangle \xrightarrow{\tau@\mathsf{p}}_{\mathscr{C}} \langle C_1,\Sigma\rangle \Rightarrow^*_{\mathscr{C}} \langle \mathbf{0},\Sigma'\rangle$, and $\Sigma \Vdash_\rho \mathsf{wlp}_{\mathfrak{C}}(C_1,\psi)$ by induction hypothesis. The only nontrivial case is when $\rho(\mathcal{X}) = \mathsf{true}$ – otherwise the antecedents of both implications in $\mathsf{wlp}_{\mathfrak{C}}(C,\psi)$ are false and the thesis trivially holds. If $\rho(\mathcal{X}) = \mathsf{true}$, then $\Sigma \vdash_\rho L(\mathsf{p},b) = \mathcal{X}$ by Lemma 1, and again both implications in $\mathsf{wlp}_{\mathfrak{C}}(C,\psi)$ are true (the first one has true premise and conclusion, while the premise in the second one is false). The case where $b \downarrow_{\Sigma(\mathsf{p})} \mathsf{false}$ is analogous.
- If C is X, then $\langle X,\Sigma\rangle \Rightarrow^*_{\mathscr{C}} \langle \mathscr{C}(X),\Sigma\rangle \Rightarrow^*_{\mathscr{C}} \langle \mathbf{0},\Sigma'\rangle$ by applying rules C|Call, C|Enter and C|Finish until all processes have entered X. By adequacy, $\mathsf{fst}(\mathfrak{C}(X)) = \mathsf{wlp}_{\mathfrak{C}}(\mathscr{C}(X),\psi)$, and the induction hypothesis establishes the thesis. □

Corollary 4. *Assume that $\mathfrak{C}$ is adequate for ψ given $\mathscr{C}$. Let C be a choreography, Σ and Σ' be states, and ρ be an assignment. If $\langle C,\Sigma\rangle \rightarrow^*_{\mathscr{C}} \langle \mathbf{0},\Sigma'\rangle$ and $\Sigma' \Vdash_\rho \psi$, then $\Sigma \Vdash_\rho \mathsf{wlp}_{\mathfrak{C}}(C,\psi)$.*

Proof. Combining Lemma 4 with deadlock-freedom and confluence of the semantics, as in the proof of Theorem 1. □

4.2 Completeness

Combining the results in the previous section, we obtain a completeness result for our calculus.

Theorem 2 (Partial completeness). *Let C be a choreography, φ and ψ be formulas, and assume that $\mathfrak{C}$ is adequate for ψ given $\mathscr{C}$. Assume that, for all states Σ and Σ' and assignment ρ, if $\Sigma \Vdash_\rho \varphi$ and $\langle C,\Sigma\rangle \rightarrow^*_{\mathscr{C}} \langle \mathbf{0},\Sigma'\rangle$, then $\Sigma' \Vdash_\rho \psi$. Then $\vdash_{\mathfrak{C}} \{\varphi\}C\{\psi\}$.*

Proof. Let Σ be a state such that $\langle C,\Sigma\rangle \rightarrow^*_{\mathscr{C}} \langle \mathbf{0},\Sigma'\rangle$, implies $\Sigma' \Vdash_\rho \psi$. Then $\Sigma \Vdash_\rho \mathsf{wlp}_{\mathfrak{C}}(C,\psi)$ by Corollary 4. Since this is the case for all states Σ such that $\Sigma \Vdash_\rho \varphi$, it follows that $\mathfrak{D} \Vdash \varphi \rightarrow \mathsf{wlp}_{\mathfrak{C}}(C,\psi)$. But $\vdash_{\mathfrak{C}} \{\mathsf{wlp}_{\mathfrak{C}}(C,\psi)\}C\{\psi\}$ by Lemma 3, whence by H|Weak the thesis holds. □

Theorems 1 and 2 can be combined with the EPP theorem from [12], which relates the behaviour of choreographies with the behaviour of their projections, to yield results on execution of distributed implementations generated by choreographies. This means that properties of these implementations can be analysed at the choreographic level, which is arguably simple, without the need for a specialised Hoare calculus for process languages.

4.3 Decidability

Finally we establish some decidability results for the Hoare calculus. We start by pointing out that we assume $\mathfrak{D}$ is decidable; since propositional logic is decidable and evaluation converges, the judgments of the form $\mathfrak{D} \models \varphi$ that appear on the premises of rule H|Weak are also decidable.

Lemma 5. *The judgement $\vdash_{\mathfrak{C}} \{\varphi\}C\{\psi\}$ is decidable.*

Proof. Assume that $\vdash_{\mathfrak{C}} \{\varphi\}C\{\psi\}$. By Theorem 1, for every state Σ and assignment ρ such that $\Sigma \Vdash_\rho \varphi$ it is the case that: if $\langle C, \Sigma\rangle \to^*_{\mathscr{C}} \langle \mathbf{0}, \Sigma'\rangle$, then $\Sigma' \Vdash_\rho \psi$. By Corollary 4, this means that $\Sigma \Vdash_\rho \mathsf{wlp}_{\mathfrak{C}}(C, \psi)$, and therefore $\mathfrak{D} \models \varphi \to \mathsf{wlp}_{\mathfrak{C}}(C, \psi)$.

Conversely, if $\mathfrak{D} \models \varphi \to \mathsf{wlp}_{\mathfrak{C}}(C, \psi)$, then for every state Σ and assignment ρ such that $\Sigma \Vdash_\rho \varphi$ it is the case that $\Sigma \Vdash_\rho \mathsf{wlp}_{\mathfrak{C}}(C, \psi)$, and therefore if $\langle C, \Sigma\rangle \to^*_{\mathscr{C}} \langle \mathbf{0}, \Sigma'\rangle$ it must hold that $\Sigma' \Vdash_\rho$ by Corollary 3. By Theorem 2 this means that $\vdash_{\mathfrak{C}} \{\varphi\}C\{\psi\}$.

This shows that $\vdash_{\mathfrak{C}} \{\varphi\}C\{\psi\}$ iff $\mathfrak{D} \models \varphi \to \mathsf{wlp}_{\mathfrak{C}}(C, \psi)$. Since $\mathsf{wlp}_{\mathfrak{C}}$ is computable and validity is decidable, it follows that $\vdash_{\mathfrak{C}} \{\varphi\}C\{\psi\}$ is decidable. □

Although the set of procedure names can in principle be infinite, most practical applications only use a finite subset of them.[2] In this case, consistency and adequacy also become decidable.

Corollary 5. *If the set of procedure names is finite, then consistency between a procedure specification map $\mathfrak{C}$ and a set of procedure definitions $\mathscr{C}$ is decidable.*

Lemma 6. *If the set of procedure names is finite, then adequacy of a procedure specification map for a formula and set of procedure definitions is decidable.*

Proof. Immediate from the definition. □

We end this section with a negative result: it is not possible to compute an adequate procedure specification map.

Lemma 7. *There is no algorithm that, given a set of procedure definitions $\mathscr{C}$ and a formula ψ, always returns a procedure specification map $\mathfrak{C}$ that is adequate for ψ given $\mathscr{C}$.*

[2] This disallows choreographies where e.g. each procedure X_i calls procedure X_{i+1}, which do not occur in practice.

Proof. Consider the formula $\psi = \bot$, which never holds. For any choreography C and satisfiable formula φ, the judgement $\{\varphi\}C\{\bot\}$ holds iff C never terminates from a state that satisfies φ.

This means that, if $\mathfrak{C}$ is adequate for $\bot$ given $\mathscr{C}$, then $\mathsf{wlp}_{\mathfrak{C}}(C, \bot)$ characterises the set of states from which execution of C diverges. In particular, C never terminates if $\mathsf{wlp}_{\mathfrak{C}}(C, \bot)$ is logically equivalent to $\top$ – which is decidable in our state logic. But Rice's Theorem implies that the class of choreographies that always diverge is undecidable, therefore $\mathfrak{C}$ cannot be computable. □

Although this result states that adequate procedure specification maps are in general not computable, there is still the possibility that they can be shown to exist always. Such a result would entail that our calculus is strongly complete. We plan to investigate this issue in future work.

5 Related Work

The work nearest to ours is [20], where the authors propose a system for functional correctness of choreographies aimed at reasoning about distributed choices. While they also propose a Hoare calculus for choreographies, there are some key differences wrt our work.

Firstly, they introduce a new choreographic language with significant differences from common practice in choreographic programming, e.g., they require every choice to involve every process regardless of their involvement in the branches in the condition. By contrast, we used an existing language with standard constructs.

Secondly, the logic used in [20] is fixed and used in the choreography language for Boolean expressions. This coupling compromises the generality of the development, because the logic and the syntax of choreographies are not standalone. Instead, we follow the standard two-layered approach for Hoare logic [2, 19], and define a state logic that is parametric on both the language of expressions in the choreographies and the theory for reasoning about them.

As a consequence, our development is more readily applicable and adaptable to other existing choreographic languages.

The only other work combining choreographies and logic is Linear Compositional Choreographies (LCC) [7], a proof theory based on linear logic for reasoning about programs that modularly combine compositional choreographies [28] with processes. This was inspired by previous work on the correspondence between linear propositions and session types [5]. LCC, however, is not aimed at functional correctness: propositions represent communication behaviour rather than assertions about states.

Design-by-Contract [25] is a framework where each protocol or function is given a contract specifying its allowed input and resulting output, similar to the pre- and postconditions of Hoare logic, which has been used to reason about distributed programs from a global level. The first work in this line [4] defined a framework for specifying contracts for multiparty sessions. Being based on

session types, this work more focussed on specifying properties of communicated values than ours, which lets them specify more properties than us, but also requires adding annotations to the language being reasoned about. An extension of this idea [24] describes chaperone contracts for higher-order binary sessions, which lets contracts update dynamically at runtime. Design-by-Contract has also been applied to microservices in the form of Whip [31]. Like our work, Whip is language-agnostic with regard to the local language, though it uses global contracts to reason directly on the local language; unlike our logic, Whip is designed for monitoring communications at runtime.

Another way of reasoning about session types is combining them with dependent types [30]. Like the work of [4], dependent types can be used to reason about the values being communicated, but unlike our work they are not intended to reason about pre- and postconditions.

Hoare logic has also been used to reason directly about systems of communicating processes [1,22]. This is far more complex than reasoning about choreographies, as it requires independently considering properties of each participant's protocol and how they are combined in the global system.

6 Conclusions

We have presented a novel Hoare calculus for reasoning about choreographic programs. Our logic allows for a great deal of flexibility, since it is parametric on both the local language of the choreographic language and a decidable theory defined by the user.

We have proven that the standard properties of Hoare logics hold for our language. Using the operational correspondence theorems for choreographies and their projections, we also showed that any properties that our logic can prove for a choreography also hold for the distributed implementation automatically generated from that choreography.

Our section on decidability left open the question of whether there always exists an adequate procedure specification map for any target formula, which we plan to investigate in future work. We also want to look further into the issue of how our decidability results can be used to implement interesting algorithms, e.g. for proof automation.

Our formalism only gives us guarantees for terminating execution paths, which means that we cannot infer any properties of non-terminating choreographies. However, an inspection of the proofs of soundness and completeness (in particular, Lemmas 2 and 3) shows that these results actually guarantee something stronger, namely that the invariants described in $\mathfrak{C}$ must hold whenever the choreography reaches a procedure call. We plan to use this observation as a starting point for an investigation about how our calculus can be used to assert properties of non-terminating executions of choreographies.

Acknowledgements. This work was partially supported by Villum Fonden, grant nr 29518.

References

1. Apt, K.R., Francez, N., de Roever, W.P.: A proof system for communicating sequential processes. ACM Trans. Program. Lang. Syst. **2**(3), 359—-385 (jul 1980). https://doi.org/10.1145/357103.357110
2. Apt, K.R., Olderog, E.: Fifty years of Hoare's logic. CoRR **abs/1904.03917** (2019)
3. Apt, K.R., Olderog, E.R., Apt, K.: Verification of sequential and concurrent programs, vol. 2. Springer (2009)
4. Bocchi, L., Honda, K., Tuosto, E., Yoshida, N.: A theory of design-by-contract for distributed multiparty interactions. In: Gastin and Laroussinie [15], pp. 162–176. https://doi.org/10.1007/978-3-642-15375-4_12
5. Caires, L., Pfenning, F.: Session types as intuitionistic linear propositions. In: Gastin and Laroussinie [15], pp. 222–236. https://doi.org/10.1007/978-3-642-15375-4_16
6. Carbone, M., Montesi, F.: Deadlock-freedom-by-design: multiparty asynchronous global programming. In: Giacobazzi, R., Cousot, R. (eds.) Procs. POPL. pp. 263–274. ACM (2013). https://doi.org/10.1145/2429069.2429101
7. Carbone, M., Montesi, F., Schürmann, C.: Choreographies, logically. Distributed Comput. **31**(1), 51–67 (2018). https://doi.org/10.1007/s00446-017-0295-1
8. Cruz-Filipe, L., Graversen, E., Lugovic, L., Montesi, F., Peressotti, M.: Functional choreographic programming. In: Seidl, H., Liu, Z., Pasareanu, C.S. (eds.) Procs. ICTAC. Lecture Notes in Computer Science, vol. 13572, pp. 212–237. Springer (2022). https://doi.org/10.1007/978-3-031-17715-6_15
9. Cruz-Filipe, L., Montesi, F.: Procedural choreographic programming. In: Bouajjani, A., Silva, A. (eds.) Procs. FORTE. Lecture Notes in Computer Science, vol. 10321, pp. 92–107. Springer (2017). https://doi.org/10.1007/978-3-319-60225-7_7
10. Cruz-Filipe, L., Montesi, F.: A core model for choreographic programming. Theor. Comput. Sci. **802**, 38–66 (2020). https://doi.org/10.1016/j.tcs.2019.07.005
11. Cruz-Filipe, L., Montesi, F., Peressotti, M.: Certifying choreography compilation. In: Cerone, A., Ölveczky, P.C. (eds.) Procs. ICTAC. LNCS, vol. 12819, pp. 115–133. Springer (2021). https://doi.org/10.1007/978-3-030-85315-0_8
12. Cruz-Filipe, L., Montesi, F., Peressotti, M.: Formalising a Turing-complete choreographic language in Coq. In: Cohen, L., Kaliszyk, C. (eds.) Procs. ITP. LIPIcs, vol. 193, pp. 15:1–15:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPIcs.ITP.2021.15
13. Dalla Preda, M., Gabbrielli, M., Giallorenzo, S., Lanese, I., Mauro, J.: Dynamic choreographies: Theory and implementation. Log. Methods Comput. Sci. **13**(2) (2017). https://doi.org/10.23638/LMCS-13(2:1)2017
14. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Trans. Inf. Theory **22**(6), 644–654 (1976). https://doi.org/10.1109/TIT.1976.1055638
15. Gastin, P., Laroussinie, F. (eds.): CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings, Lecture Notes in Computer Science, vol. 6269. Springer (2010)
16. Giallorenzo, S., Montesi, F., Peressotti, M.: Choreographies as objects. CoRR **abs/2005.09520** (2020), https://arxiv.org/abs/2005.09520
17. Giallorenzo, S., Montesi, F., Peressotti, M., Richter, D., Salvaneschi, G., Weisenburger, P.: Multiparty languages: The choreographic and multitier cases (pearl). In: Møller, A., Sridharan, M. (eds.) Procs. ECOOP. LIPIcs, vol. 194, pp. 22:1–22:27. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPIcs.ECOOP.2021.22

18. Hirsch, A.K., Garg, D.: Pirouette: higher-order typed functional choreographies. Proc. ACM Program. Lang. **6**(POPL), 1–27 (2022). https://doi.org/10.1145/3498684
19. Hoare, C.: An axiomatic basis for computer programming. Commun. ACM **12**(10), 576–580 (1969). https://doi.org/10.1145/363235.363259
20. Jongmans, S., van den Bos, P.: A predicate transformer for choreographies – computing preconditions in choreographic programming. In: Sergey, I. (ed.) Procs. ESOP. Lecture Notes in Computer Science, vol. 13240, pp. 520–547. Springer (2022). https://doi.org/10.1007/978-3-030-99336-8_19
21. Leesatapornwongsa, T., Lukman, J.F., Lu, S., Gunawi, H.S.: Taxdc: A taxonomy of non-deterministic concurrency bugs in datacenter distributed systems. In: Conte, T., Zhou, Y. (eds.) Procs. ASPLOS. pp. 517–530. ACM (2016). https://doi.org/10.1145/2872362.2872374
22. Levin, G., Gries, D.: A proof technique for communicating sequential processes. Acta Informatica **15**, 281–302 (1981). https://doi.org/10.1007/BF00289266
23. López, H.A., Nielson, F., Nielson, H.R.: Enforcing availability in failure-aware communicating systems. In: Albert, E., Lanese, I. (eds.) Procs. FORTE. Lecture Notes in Computer Science, vol. 9688, pp. 195–211. Springer (2016). https://doi.org/10.1007/978-3-319-39570-8_13
24. Melgratti, H.C., Padovani, L.: Chaperone contracts for higher-order sessions. Proc. ACM Program. Lang. **1**(ICFP), 35:1–35:29 (2017). https://doi.org/10.1145/3110279
25. Meyer, B.: Applying "design by contract". Computer **25**(10), 40–51 (1992). https://doi.org/10.1109/2.161279
26. Montesi, F.: Choreographic Programming. Ph.D. Thesis, IT University of Copenhagen (2013)
27. Montesi, F.: Introduction to Choreographies. Cambridge University Press (2023)
28. Montesi, F., Yoshida, N.: Compositional choreographies. In: D'Argenio, P.R., Melgratti, H.C. (eds.) Procs. CONCUR. Lecture Notes in Computer Science, vol. 8052, pp. 425–439. Springer (2013). https://doi.org/10.1007/978-3-642-40184-8_30
29. Needham, R.M., Schroeder, M.D.: Using encryption for authentication in large networks of computers. Commun. ACM **21**(12), 993–999 (1978). https://doi.org/10.1145/359657.359659
30. Toninho, B., Caires, L., Pfenning, F.: Dependent session types via intuitionistic linear type theory. In: Schneider-Kamp, P., Hanus, M. (eds.) Procs. PPDP. pp. 161–172. ACM (2011). https://doi.org/10.1145/2003476.2003499
31. Waye, L., Chong, S., Dimoulas, C.: Whip: higher-order contracts for modern services. Proc. ACM Program. Lang. **1**(ICFP), 36:1–36:28 (2017). https://doi.org/10.1145/3110280