Type-checking Availability in Choreographic Programming

Hugo A. López Flemming Nielson Hanne Riis Nielson
Technical University of Denmark
Kongens Lyngby, Denmark

Choreographic programming is a programming-language design approach that drives a safe protocol development in distributed systems. We study choreographic programming for loosely-coupled infrastructures, where the availability of components may change at runtime. We introduce a choreography language featuring novel operators for multiparty, partial and collective communications; we provide a type discipline that controls how partial communications refer only to available components; and we show that well-typed choreographies enjoy progress.

1 Introduction

Choreographies are an emerging paradigm in concurrent programming aiming at providing a *correct-by-construction* framework for distributed systems [1, 2]. Protocols, dubbed *choreographies*, provide a global vision on how the communications among different components in a distributed system shall be structured. Once defined, choreographies serve as stepping stones to guide the implementation of a system, both by automating the generation of deadlock-free code for each component involved (EndPoint Projection), by monitoring that the execution of a distributed system behaves according to a protocol, and even by healing anomalous executions of a distributed system.

Choreographic languages studied so far have assumed an underlying *tighly-coupled* infrastructure, meaning that all the components involved in a protocol are present along the entire life of a system. A protocol for a tighly-coupled infrastructure considers the absence of one of the components as a failure. The case of Cyber-Physical Systems (CPS) represents a challenge for choreographies. CPS run under *loosely-coupled* infrastructures, where the assumptions regarding the availability components require relaxation. For instance, in a CPS, sensors can become temporarily unavailable due to multiple aspects (battery failure, climatic conditions, sensor malfunction, distance of the RF transceiver, etc.) More the norm than the exception, practitioners in CPS must take availability into consideration, programming safety-critical applications in a *failure-aware* fashion. A component at risk of damage will be normally deployed in a redundant manner, to avoid that the system blocks in case such component fails. It is more conceivable then to move from point-to-point to collective communications, such as broadcast and reduce. In this way, one can model scenarios where decisions are based on the aggregate of the communcation received from redundant components, rather on a single unit. In the design of loosely-coupled systems, the specification should accept that some of the components may be unavailable, and work properly as long as a minimum set is. This assumption modify our correctness criteria, and poses the following question:

Can we ensure progress in distributed systems with variable availability conditions?

The study of communication protocols including collective communications has recently spawn different research directions (c.f. [4, 3, 7, 8, 9].) In this paper we introduce a choreography language featuring novel operators for multiparty, partial, collective communications. It is a generalization of the global calculus [2], restricted to limited asynchrony, furtherly enriched with quality predicates [10]. Our novel communication primitives impose softer requirements on component availability. The use of q as a *quality predicate* is common to all communication interactions. Its inclusion derives from the Quality

Submitted to:
PLACES 2016

$$t_1[S]\{X_1\}, t_2[S]\{X_2\}, t_3[S]\{X_3\} \textbf{ start } t_m[M]\{X_m\} : temperature(k); \quad (1)$$
$$t_m\{X_m;Y_m\} \twoheadrightarrow \&_{\mathbf{q_1}}(t_1\{X_1;Y_1\}, t_2\{X_2;Y_2\}, t_3\{X_3;Y_3\}) : k[measure]; \quad (2)$$
$$\&_{\mathbf{q_2}}(t_1\{Y_1;Z_1\}.\text{“}1^\circ\text{”}, t_2\{Y_2;Z_2\}.\text{“}-2^\circ\text{”}, t_3\{Y_3;Z_3\}.\text{“}5^\circ\text{”}) \twoheadrightarrow t_m\{Y_m;Z_m\} : x_m : \langle k, avg\rangle; 0 \quad (3)$$

Figure 1: Example: Sensor network choreography

Calculus [10], and allows one to describe interactions under a subset of *available* participants. Additionally, GC_q annotates threads with atomic formulae $\{X;Y\}$. One way of seeing them is as capabilities that must be achieved before a thread can continue. We provide an illustrative example of the language.

Example 1.1 (Sensor Networks (SN)). SN are commonly deployed in unreliable conditions, making their nodes easily unavailable. Measurements performed by SN do not depend on the reliability of one node, but they rather require readings from a reasonable subset of the nodes deployed. Figure 1 portrays a simple SN choreography for temperature measurement. The syntax in black font corresponds to classical choreographies [2] extended with collective communications [9]. Line 1 models a *session establishment* phase running between sensors t_1, t_2, t_3 and a new thread t_m in the monitor. In Line 2, t_m executes a *collective selection* of method *measure* in each sensor. Line 3 will *reduce* all the measurements in the sensors, and compute its average in the monitor. Quality predicates (in green font) $\mathbf{q_1}$ and $\mathbf{q_2}$ decree the availability requirements for the SN. For instance, $\mathbf{q_1} = \mathbf{q_2} = \forall$ only allows communications with all sensors in place, while $\mathbf{q_1} = \forall, \mathbf{q_2} = 2/3$ tolerates the absence of readings in one sensor. Finally, progress capabilities (in blue font) $\{X;Y\}$ define with X the required capability for a thread to be executed, and with Y the capability offered after its engagement. For session establishment, no capability is required.

In §2 we introduce the GC_q choreography language. The novel communication primitives in GC_q greatly affect progress guarantees for choreographies, since collective communications specified without due care of the quality predicates used will lead into systems that block due to unavailable components. In §3 we present a type system, orthogonal to session types, that ensures progress under variable availability conditions. Finally, §4 concludes the paper. In the Appendix we present full definitions and the proof sketches of the main results.

2 The Global Quality Calculus (GC_q)

In the sequel, C, C' are choreographies; $t, t', \ldots$ are threads and $A_1, A_2, \ldots$ are role annotations. Annotated threads range over $p_1, \ldots, p_n$. Variable a ranges over *service channels*, intuitively denoting the public identifier of a service, and k range over a finite, countable set of session (names) $\mathbf{N}$, created at runtime. Variables $x, x', \ldots$ range over variables local to each thread; Atomic formulae range over $X, Y, \ldots$; arithmetic and other first-order expressions excluding service and session channels are denoted by e, e'. Names m, n range over threads and session channels. Finally, q indicates a *quality predicate*, that determines when sufficient inputs/outputs are available. As an example, q can be $\exists$, meaning that one sender/receiver is required in the interaction, or it can be $\forall$ meaning that all of them are needed. We require q to be monotonic and satisfiable.

Definition 2.1 (GC_q syntax).

$$\textit{(Choreographies)} \quad C, C', C'' ::= \eta; C \mid C + C' \mid e@p?C' : C'' \mid 0 \qquad \textit{(Annotated threads)} \quad p ::= t[A]\{X;Y\}$$
$$\textit{(Interactions)} \quad \eta ::= \widetilde{p_r} \textbf{ start } \widetilde{p_s} : a(k) \mid p_r.e \twoheadrightarrow \&_{\mathbf{q}}(\widetilde{p_s : x_s}) : k \mid \&_{\mathbf{q}}(\widetilde{p_r.e_r}) \twoheadrightarrow p_s : x : \langle k, op\rangle \mid p_r \twoheadrightarrow \&_{\mathbf{q}}(\widetilde{p_s}) : k[l]$$

For simplicity of presentation, all models in the paper are finite. The addition of operators like recursion for writing infinite behaviors goes as expected. We will concentrate our discussion to novel interactions. Annotated threads $t[A]\{X;Y\}$ are built from a thread name t, a pair of atomic formulae $\{X;Y\}$ and a role annotation A. Intuitively, X and Y describe the capabilities required/offered in a thread. Interactions can take various shapes: first, **start** defines a (multiparty) *session initiation* between active annotated threads $\widetilde{p_r}$ and annotated service threads $\widetilde{p_s}$. Each active thread (resp. service thread) implements the behaviour of one of the roles in $\widetilde{A_r}$ (resp. $\widetilde{A_s}$), sharing a new session name k. We assume that a session is established with at least two participating processes, therefore $2 \leq |\widetilde{p_r}| + |\widetilde{p_s}|$, and that threads in $\widetilde{p_r} \cup \widetilde{p_s}$ are pairwise different. A *broadcast* takes the form $p_r.e \text{ -> } \&_{\mathbf{q}}(\widetilde{p_s : x_s}) : k$, where a session channel k is used to transfer the evaluation of expression e (located at p_r) to threads in $\widetilde{p_s}$, with the resulting binding of variable x_i at p_i, for each $p_i \in \widetilde{p_s}$. A *reduce* is written as $\&_{\mathbf{q}}(\widetilde{p_r.e_r}) \text{ -> } p_s : x : \langle k, op \rangle$, where each annotated thread p_i in $\widetilde{p_r}$ evaluates an expression e_i, and the aggregate of all receptions is evaluated using op (an operator respecting commutative, associativity and neutral elements such as max, min, etc.) Interaction $p_r \text{ -> } \&_{\mathbf{q}}(\widetilde{p_s}) : k[l]$ describes a collective label selection: p_r communicates the selection of label l to peers in $\widetilde{p_s}$ through session k. Explicit $x@p/e@p$ indicates the variable/boolean expression x/e is located at p. We often omit 0, empty vectors and atomic formulae $\{X;Y\}$ from annotated threads when unnecessary. The free term variables and atomic formulae are defined as usual. They are denoted by $\mathsf{fv}(C)$ and $\mathsf{fform}(C)$ respectively. An interaction η in $\eta; C$ can bind session channels, processes and variables. In **start** , variables $\{\widetilde{p_r}, a\}$ are free while variables $\{\widetilde{p_s}, k\}$ are bound (since they are freshly created). In broadcast, variables $\widetilde{x_s}$ are bound. A reduce comand binds $\{x\}$. Finally, we assume that all bound variables in an expression have been renamed apart from each other, and apart from any other free variables in the expression.

Expressivity The importance of roles is only crucial in a **start** interaction. Technically, one can infer the role of a given thread t used in an interaction η by looking at the **start** interactions preceding it in the AST. GC_q can still represent unicast message-passing patterns as in [1]. Unicast communication $p_1.e \text{ -> } p_2 : x : k$ can be encoded in multiple ways using broadcast/reduce operators. For instance, $p_1.e \text{ -> } \&_{\forall}(p_2 : x) : k$ and $\&_{\forall}(p_1.e) \text{ -> } p_2 : x : \langle id, k \rangle$ are just a couple of possible implementations.

2.1 Semantics

Choreographies are considered modulo standard structural and swapping congruence relations (resp. $\equiv$, $\simeq_C$) [2]. The swap congruence provides a way to reorder non-conflicting interactions, allowing for a restricted form of asynchronous behavior. Non-conflicting interactions are those involving sender-receiver actions that do not conform a control-flow dependency. For instance, $t_A.e_A \text{ -> } \&_{\mathbf{q_1}}(t_B : x_B) : k_1; t_C.e_C \text{ -> } \&_{\mathbf{q_2}}(t_D : x_D) : k_2 \simeq_C t_C.e_C \text{ -> } \&_{\mathbf{q_2}}(t_D : x_D) : k_2; t_A.e_A \text{ -> } \&_{\mathbf{q_1}}(t_B : x_B) : k_1$. Formally, the operational semantics is given in terms of labelled transition rules. A transition $(\nu \widetilde{m})\ \langle \sigma, C \rangle \xrightarrow{\lambda} (\nu \widetilde{n})\ \langle \sigma', C' \rangle$ says that a configuration $\langle \sigma, C \rangle$ with used names $\widetilde{m}$ fires an action λ and evolves into $\langle \sigma', C' \rangle$ with names $\widetilde{n}$. A state σ keeps track of the capabilities achieved by a thread in a session, and it is formally defined as set of maps $(t,k) \mapsto X$. We use standard state manipulation functions, including lookup $(\sigma(t,k))$ and update $(\sigma[\sigma'])$. The exchange function $[\![X;Y]\!]Z$ returns $(Y \backslash X) \cup Z$ if $X \subseteq Z$ and Z otherwise. Actions are defined as $\lambda ::= \{\tau, \eta, \lambda@p\}$, where η denotes interactions, τ represents an internal computation, and $\lambda@p$ refers to the thread p performing action λ. Relation $e@p \downarrow v$ describes the evaluation of a expression e (in p) to a value v.

Because of the introduction of quality predicates, a move from $\eta; C$ into C might leave some variables in η without proper values, as the participants involved were not available. We draw inspiration from

$$\frac{\eta = \&_{\mathbf{q}}(t_1[A_1]\{X_1;Y_1\}.e_1,\ldots,t_j[A_j]\{X_j;Y_j\}.e_j) \twoheadrightarrow t_B[B]\{X_B;Y_B\} : x : \langle k, op\rangle \quad e_i@t_i \downarrow v_i \quad X_i \subseteq \sigma(t_i,k) \quad \sigma' = \sigma[(t_i,k) \mapsto [\![X_i;Y_i]\!](\sigma(t_i,k))] \quad i\in\{1\ldots j\}}{\langle \sigma,\eta;C\rangle \xrightarrow{(t_i,k):\, X_i::Y_i} \twoheadrightarrow \langle \sigma', (\&_{\mathbf{q}}(t_1[A_1]\{X_1;Y_1\}.e_1,\ldots,t_i[A_i].\mathsf{some}(v_i),\ldots,t_j[A_j]\{X_j;Y_j\}.e_j) \twoheadrightarrow t_B[B]\{X_B;Y_B\} : x : \langle k,op\rangle);C\rangle}$$

$$\frac{}{p.e ::_{\mathtt{ff}} [\,]} \qquad \frac{}{p.\mathsf{some}(v) ::_{\mathtt{tt}} [(p,\mathsf{some}(v))]} \qquad \frac{sc_1 ::_{t_1} \theta_1 \quad \ldots \quad sc_n ::_{t_n} \theta_n}{\&_q(sc_1,\ldots,sc_n) ::_{q(t_1,\ldots,t_n)} \theta_1 \circ \ldots \circ \theta_n}$$

Figure 2: (2a) effects, and (2b) binding testing rules

$$\frac{\eta = \widetilde{t_r[A]\{Y_r\}}\ \mathbf{start}\ \widetilde{t_s[B]\{Y_s\}} : a(k) \qquad \sigma' = [(t_i,k) \mapsto Y_i]_{i=1}^{|\widetilde{t_r}|+|\widetilde{t_s}|} \qquad \widetilde{n} = \widetilde{t_s},\{k\} \qquad \widetilde{n}\ \#\ \widetilde{m}}{(\nu\widetilde{m})\ \langle\sigma,\eta;C\rangle \xrightarrow{\eta} (\nu\widetilde{m},\widetilde{n})\ \langle\sigma[\sigma'],C\rangle}\ \textsf{Init}$$

$$\frac{J \subseteq \widetilde{t_r} \quad q(J) \quad \forall i\in\{A\}\cup J : X_i \subseteq \sigma(t_i,k) \wedge \sigma'(t_i,k) = [\![X_i;Y_i]\!](\sigma(t_i,k)) \quad \forall i\in\widetilde{t_r} : \theta(x_i) = \begin{cases} \mathsf{some}(v) & i\in J \\ \mathsf{none} & \text{otherwise} \end{cases} \qquad e@t_A \downarrow v}{(\nu\widetilde{m})\ \left\langle\sigma, \left(t_A[A]\{X_A;Y_A\}.e \twoheadrightarrow \&_{\mathbf{q}}(\widetilde{t_r[B_r]\{X_r;Y_r\}} : x_r) : k\right);C\right\rangle \xrightarrow{\theta(\eta)} (\nu\widetilde{m})\ \langle\sigma[\sigma'],\theta(C)\rangle}\ \textsf{Bcast}$$

$$\frac{\langle\sigma,\eta;C\rangle \xrightarrow{\xi} \twoheadrightarrow \langle\sigma',\eta';C\rangle \qquad \eta' ::_{\mathtt{ff}} \theta}{(\nu\widetilde{m})\ \langle\sigma,\eta;C\rangle \xrightarrow{\tau} (\nu\widetilde{m})\ \langle\sigma',\eta';C\rangle}\ \textsf{RedD} \qquad \frac{\langle\sigma,\eta;C\rangle \xrightarrow{\xi} \twoheadrightarrow \langle\sigma',\eta';C\rangle \qquad \eta' ::_{\mathtt{tt}} \theta}{(\nu\widetilde{m})\ \langle\sigma,\eta;C\rangle \xrightarrow{\theta(\eta')} (\nu\widetilde{m})\ \langle\sigma',C[\mathsf{op}(\theta)/x@t_{j+1}]\rangle}\ \textsf{RedE}$$

Figure 3: SOS for GC_q (excerpt): η in (BCAST), (REDD) and (REDE) is defined as in (INIT).

[10], introducing *effect* rules describing how the evaluation of an expression in a reduce operation affects interactions. The relation $\twoheadrightarrow$ (Figure 2a) describes how evaluations are partially applied without affecting waiting threads. Label ξ records the substitutions of atomic formulae in each thread. We distinguish between data and optional data, much like the use of option data types in programming languages like Standard ML [6]. In the syntax we use terms t to denote data and expressions e to denote optional data; in particular, the expression $\mathsf{some}(t)$ signals the presence of some data t and none the absence of data. Finally, given $\phi \in \{\mathtt{tt},\mathtt{ff}\}$, the relation $\beta ::_\phi \theta$ tracks whether all required substitutions in β have been performed, as well as the values used θ. Binder β is defined in terms of partially evaluated outputs c:

$$sc ::= \quad p.e \quad \mid \quad p.\mathsf{some}(v) \qquad\qquad c ::= \quad \&_q(sc_1,\ldots,sc_n)$$

The rules specifying $\beta ::_\phi \theta$ are presented in Figure 2b. A composition of evaluations $\theta_1 \circ \theta_2(x)$ is defined as $\theta_1 \circ \theta_2(x) ::= \theta_1(\theta_2(x))$, and $q(t_1,\ldots,t_n) = \bigwedge_{i\in 1\le i\le n} t_i$ if $q = \forall$, $q(t_1,\ldots,t_n) = \bigvee_{i\in 1\le i\le n} t_i$ if $q = \exists$, and possible combinations therein.

We now have all the ingredients to understand the semantics of GC_q. The set of transition rules in $\xrightarrow{\lambda}$ is defined as the minimum relation on names, states, and choreographies satisfying the rules in Figure 7. We give some intuitions on the most representative ones. We use the shorthand notation $A \# B$ to denote set disjointness, $A \cap B = \emptyset$. Rule (INIT) models initial interactions: state σ is updated to account for the new threads in the session, updating the set of used names in the reductum. Rule (BCAST) models broadcast: given an expression evaluated at the sender, one needs to check that there are enough receivers ready to get a message ($q(J)$). The result will: 1. update the current state with the new states of each participant engaged in the broadcast, and 2. apply the partial substitution θ to the continuation C. The behaviour of a reduce operation is described using rules (REDD) and (REDE): each sender t_i evaluates an expression e_i, generating an effect in the overall interaction. If all required substitutions have been performed, then one

can proceed by applying evaluating the operator to the set of received values, binding variable x to its results and proceeding with the continuation, otherwise the choreography will wait until further inputs are bound, (i.e.: the continuation is delayed).
In contrast to previous works in multiparty sessions (e.g. [3]), we present an *early* semantics: it allows for transitions to match with distinct moves, depending on which participants are available first. The choice is motivated from our area of study. In CPSs, progress cannot rely on the availability of all components, but rather on a subset of them. CPS also motivates the asymmetry between broadcast and reduce: while a broadcast is a *non-blocking* operation that fires as long as enough receivers are ready to be engaged, a reduce is a *blocking* operation, and will delay the transition until there is enough senders.

Definition 2.2 (Deadlock Freedom). *C is deadlock-free if $\exists C', \sigma', \lambda$ s.t. $\langle \sigma, C\rangle \xrightarrow{\lambda} \langle \sigma', C'\rangle$, for all σ.*

3 Type-checking progress

One of the challenges regarding the use of partial collective operations concerns the possibility of getting into runs with locking states. Consider a variant of Example 1.1 with $\mathbf{q_1} = \exists$ and $\mathbf{q_2} = \forall$. Such a choice may lead to blocked configurations. The system may block, since the collective selection in line (2) only requires one receiver to be engaged, while the reduce operator in line (3) requires all senders to be ready. One can also reach a blocking situation if the participant dependencies among interactive behaviour are not preserved. For instance, by replacing Line (3) in the example with:

$$\&_{\exists}(t_1\{Y_1;Z_1\}.\text{“1°”}, t_3\{Y_3;Z_3\}.\text{“5°”}) \text{ -> } t_m\{Y_m;Z_m\} : x_m : \langle k, avg\rangle; 0 \tag{3.1}$$

The choreography will block if $\mathbf{q_1} = \exists$, as the choice operator in line (2) can execute a communication over t_2, blocking the reduce in the next step.
We introduce a type system to ensure progress on variable availability conditions. A judgment is written as $\Psi \vdash C$, where Ψ is a formula in intutionistic linear logic (ILL) [5]. The type environment Ψ is formed by formulae in ILL. Intuitively, $\Psi \vdash C$ is read as *the formulae in Ψ describes the program point immediately before C*. Ψ can take the following shapes: $\Psi ::= \mathtt{tt} \mid p : k[A] \triangleright X \mid \Psi \otimes \Psi \mid \Psi \oplus \Psi \mid \Psi \multimap \Psi \mid \exists x.\Psi$, where $p : k[A] \triangleright X$ is an *ownership type*, asserting that p behaves as the role A in session k with atomic formula X, and the other operators are standard ILL quantifiers and connectives. Moreover, we require Ψ to be formulae free of linear implications in $\Psi \vdash C$ [1].

Figure 4 presents the typing rules GC_q (see Appendix A.4 for its full definition). Rule (TINIT) types new sessions: Ψ is extended with function $\mathbf{init}(\widetilde{t_p[A]\{X\}}, k)$, which returns a set of ownership types $\{t_q : k[B] \triangleright X \mid t_q[B] \in \widetilde{t_p[A]}\}$. The condition $\{\widetilde{t_s}, k\} \# (\mathbf{T}(\Psi) \cup \mathbf{K}(\Psi))$ ensure that new names do not exist neither in the threads nor in the used keys in Ψ. The typing rules for broadcast, reduce and selection are analogous, so we focus our explanation in (TBCAST). Here we abuse of the notation, writing $\Psi \vdash C$ to denote type checking, and $\Psi \vdash \psi$ to denote formula entailment. A broadcast is typable if Ψ contains the capabilities for the sender and a subset of the receivers. Because of the variable availability conditions, we lesser the typing requirements governing all receivers, to typing all the subsets of receivers s.t. the evaluation of the quality predicate holds. Formula Ψ must contain subformulae for senders, receivers and concurrent sessions. Finally, the type of the continuation will consume the resources required to type the broadcast, updating them with new capabilities for the threads engaged.

Example 3.1. In Example 1.1, $\mathtt{tt} \vdash C$ if $\mathbf{q_1} = \mathbf{q_2} = \forall$. In the case $\mathbf{q_1} = \exists, \mathbf{q_2} = \forall$, the same typing fails. Similarly, $\mathtt{tt} \nvdash C$ if $\mathbf{q_1} = \exists$, for the variant of the example updated with eq. 3.1.

[1] We do, however, use the full set of operators when performing proof search

$$\frac{\Psi \otimes \mathbf{init}(\widetilde{t_r[A]\{Y_r\}}, \widetilde{t_s[B]\{Y_s\}}, k) \vdash C \quad \{\widetilde{t_s}, k\} \# (\mathbf{T}(\Psi) \cup \mathbf{K}(\Psi))}{\Psi \vdash \widetilde{t_r[A_r]\{Y_r\}}\ \mathbf{start}\ \widetilde{t_s[B_r]\{Y_s\}} : a(k); C}\ \textsf{Tinit} \qquad \frac{}{\Psi \vdash 0}\ \textsf{Tinact}$$

$$\frac{\begin{array}{c}\forall J.\ s.t. (J \subseteq \widetilde{B} \wedge q(J)) \qquad \Psi = \bigotimes_{j\in J}(\psi_j) \otimes \psi_A \otimes \Psi' \qquad \phi = t_A : k[A] \triangleright X_A \bigotimes_{j\in J}(t_j : k\left[B_j\right] \triangleright X_j) \\ \phi' = t_A : k[A] \triangleright Y_A \bigotimes_{j\in J}\left(t_j : k\left[B_j\right] \triangleright Y_j\right) \quad \left(\bigotimes_{j\in J}\psi_j\right) \otimes (\phi \multimap \phi') \vdash \phi' \quad (t_A : k[A] \triangleright Y_A) \bigotimes_{j\in J}\left(t_j : k\left[B_j\right] \triangleright Y_j\right) \otimes \Psi' \vdash C\end{array}}{\Psi \vdash \left(t_A[A]\{X_A; Y_A\}.e \mathrel{->} \&_{\mathbf{q}}(\widetilde{t_r[B_r]\{X_r; Y_r\}} : x_r) : k\right); C}\ \textsf{Tbcast}$$

$$\frac{\begin{array}{c}\forall J.\ s.t. (J \subseteq \widetilde{A} \wedge q(J)) \qquad \Psi = \bigotimes_{j\in J}(\psi_j) \otimes \psi_B \otimes \Psi' \qquad \phi = t_B : k[B] \triangleright X_B \bigotimes_{j\in J}(t_j : k\left[A_j\right] \triangleright X_j) \\ \phi' = t_B : k[B] \triangleright Y_B \bigotimes_{j\in J}\left(t_j : k\left[A_j\right] \triangleright Y_j\right) \quad \left(\bigotimes_{j\in J}\psi_j\right) \otimes (\phi \multimap \phi') \vdash \phi' \quad (t_B : k[B] \triangleright Y_B) \bigotimes_{j\in J}\left(t_j : k\left[A_j\right] \triangleright Y_j\right) \otimes \Psi' \vdash C\end{array}}{\Psi \vdash \left(\&_{\mathbf{q}}(\widetilde{t_r[A_r]\{X_r; Y_r\}}.e_r) \mathrel{->} t_B[B]\{X_B; Y_B\} : x : \langle k, op\rangle\right); C}\ \textsf{Tred}$$

Figure 4: GC_q: Type checking rules (excerpt)

A type preservation theorem must consider the interplay between the state and formulae in Ψ. We write $\sigma \models \Psi$ to say that the tuples in σ entail the formulae in Ψ.

Theorem 3.2 (Type Preservation). *If* $(\nu\widetilde{m})\ \langle \sigma, C\rangle \xrightarrow{\lambda} (\nu\widetilde{n})\ \langle \sigma', C'\rangle$, $\sigma \models \Psi$, *and* $\Psi \vdash C$, *then* $\exists \Psi'.\Psi' \vdash C'$ *and* $\sigma' \models \Psi'$.

Theorem 3.3 (Progress). *If* $\Psi \vdash C$, $\sigma \models \Psi$ *and* $C \not\equiv 0$, *then* C *is deadlock-free.*

4 Final Remarks

We have presented a language for the description of protocols with variable availability conditions, as well as a type system to ensure its progress. The contributions here developed constitute the first step towards the development of a methodology for the safe development of communication protocols in CPS. The analysis presented is orthogonal to existing type systems for choreographies (c.f. session types [2].) The integration of our analysis techniques and generation of distributed implementations (e.g. EndPoint Projection [1]) constitutes our next research step.

Our most immediate plans include the modification of the type theory to capture a large set of assynchronous behaviour (c.f. Rule $\lfloor {}^{\mathrm{C}}|_{\mathrm{ASYNCH}}\rceil$ in [2]) and recursion. Our types are computationally expensive, because for each global communication primitive one must perform the analysis on each subset of participants. The situation will be critical once recursion is considered. We believe that the efficiency of type checking can be improved by modifying the theory so it generates one formulae for all subsets. Finally, we believe that many of the atomic formulae can be automatically inferred, which could greatly simplify the models here presented.

References

[1] M. Carbone, K. Honda & N. Yoshida (2007): *Structured communication-centred programming for web services*. In: *ESOP*, pp. 2–17.

[2] M. Carbone & F. Montesi (2013): *Deadlock-freedom-by-design: Multiparty Asynchronous Global Programming*. In: *POPL*, pp. 263–274, doi:10.1145/2429069.2429101.

[3] G. Castagna, M. Dezani-Ciancaglini & L. Padovani (2011): *On global types and multi-party sessions*. In: *FORTE*, Springer, pp. 1–28.

[4] P.M. Denielou & N. Yoshida (2012): *Multiparty Session Types Meet Communicating Automata*. In: *ESOP*, 7211, Springer Science & Business Media, p. 194.

$$\frac{\mathbf{T}(\eta)\,\#\,\mathbf{T}(\eta')}{\eta;(\eta';C)\simeq_C \eta';(\eta;C)} \qquad \frac{t[A]\notin\mathbf{T}(\eta)}{e@t[A]?\eta;C_1:\eta;C_2\simeq_C \eta;e@t[A]?C_1:C_2}$$

$$\frac{t_1[A]\neq t_2[B]}{e@t_1[A]?(e'@q?C_1:C_2):(e'@t_2[B]?C_1':C_2')\simeq_C e'@t_2[B]?(e@t_1[A]?C_1:C_1'):(e@t_1[A]?C_2:C_2')}$$

Figure 5: Swap congruence relation, $\simeq_C$

[5] Jean-Yves Girard (1987): *Linear Logic*. *Theor. Comp. Sci.* 50, pp. 1–102.

[6] R. Harper (2013): *Programming in Standard ML*. Working Draft. Available at `http://www.cs.cmu.edu/~rwh/smlbook/book.pdf`.

[7] H. Hüttel & N. Pratas (2015): *Broadcast and aggregation in BBC*. In Simon Gay & Jade Alglave, editors: *PLACES'15*, pp. 51–62.

[8] D. Kouzapas, R. Gutkovas & S. J. Gay (2014): *Session types for broadcasting*. In Alastair F. Donaldson & Vasco T. Vasconcelos, editors: *PLACES*, *EPTCS* 155, Grenoble, France, pp. 25–31.

[9] H. A. López, E. R. B. Marques, F. Martins, N. Ng, C. Santos, V. T. Vasconcelos & N. Yoshida (2015): *Protocol-based verification of message-passing parallel programs*. In: *OOPSLA*, pp. 280–298, doi:10.1145/2814270.2814302.

[10] H. R. Nielson, F. Nielson & R. Vigo (2013): *A calculus for quality*. In: *FACS*, Springer, pp. 188–204.

A Additional Definitions

A.1 Structural Congruence

Definition A.1 (Structural Congruence). *Choreographies are considered modulo structural congruence, defined as the least congruence relation on C supporting $\alpha-$renaming, such that $(C,0,+)$ is an abelian monoid.*

A.2 Swap Congruence

Let $\mathbf{T}(C)$ the set of threads in C, defined inductively as $\mathbf{T}(\eta;C) \stackrel{\texttt{def}}{=} \mathbf{T}(\eta)\cup\mathbf{T}(C)$, and $\mathbf{T}(\eta) \stackrel{\texttt{def}}{=} \bigcup_{i=\{1..j\}} t_i$ if $\eta = t_1[A_1].e \texttt{->} \&_{\mathbf{q}}(t_1[A_2]:x_2,\ldots,t_j[A_j]:x_j):k$ (similarly for (START), (REDUCE) and (CHOICE), and standardly for the other process constructs in C). The swapping congruence rules are presented in Figure 5.

A.3 Operational Semantics

Definition A.2 (Store Manipulation). *The rules in Figure 6 define store manipulation operations, including store update ($\sigma[\sigma']$), and lookup ($\sigma(t,k)$) :*

$$\frac{Y=\begin{cases} X & (t,k,X)\in\sigma \\ \emptyset & \text{otherwise}\end{cases}}{\sigma(t,k)=Y} \qquad \frac{\sigma\,\#\,\sigma'}{\sigma[\sigma']=\sigma,\sigma'} \qquad \frac{\delta=\{(t,k,X)\mid (t,k,X)\in\sigma\wedge(t,k,Y)\in\sigma'\}}{\sigma[\sigma']=(\sigma\backslash\delta),\sigma'}$$

Figure 6: Store update rules

$$\frac{\sigma' = [(t_i,k) \mapsto Y_i]_{i=1}^{|\widetilde{t_r}|+|\widetilde{t_s}|} \quad \widetilde{n} = \widetilde{t_s},\{k\} \quad \widetilde{n} \# \widetilde{m}}{(\nu\widetilde{m})\ \left\langle \sigma, t_r\widetilde{[A]\{Y_r\}}\ \mathbf{start}\ t_s\widetilde{[B]\{Y_s\}} : a(k); C \right\rangle \xrightarrow{t_r[A]\ \mathbf{start}\ t_s[B]:a(k)} (\nu\widetilde{m},\widetilde{n})\ \langle \sigma[\sigma'], C\rangle}\ \textsf{Init}$$

$$\frac{(\nu\widetilde{m})\ \langle \sigma, C_i\rangle \xrightarrow{\lambda} (\nu\widetilde{m})\ \langle \sigma', C'\rangle \quad i\in\{1,2\}}{(\nu\widetilde{m})\ \langle \sigma, C_1 + C_2\rangle \xrightarrow{\lambda} (\nu\widetilde{m})\ \langle \sigma', C'\rangle}\ \textsf{Sum}$$

$$\frac{J \subseteq \widetilde{t_r} \quad q(J) \quad \forall i\in\{A\}\cup J : X_i \subseteq \sigma(t_i,k) \wedge \sigma'(t_i,k) = [\![X_i;Y_i]\!](\sigma(t_i,k)) \quad e@t_A \downarrow v \quad \forall i\in\widetilde{t_r} : \theta(x_i) = \begin{cases} \mathsf{some}(v) & i \in J \\ \mathsf{none} & \text{otherwise} \end{cases}}{(\nu\widetilde{m})\ \left\langle \sigma, \left(t_A[A]\{X_A,Y_A\}.e \twoheadrightarrow \&_{\mathbf{q}}(t_r[B_r]\widetilde{\{X_r;Y_r\}} : x_r) : k\right); C \right\rangle \xrightarrow{\theta(t_A[A]\{X_A,Y_A\}.e \gg \&_{\mathbf{q}}(t_r[B_r]\widetilde{\{X_r;Y_r\}}:x_r):k)} (\nu\widetilde{m})\ \langle \sigma[\sigma'], \theta(C)\rangle}\ \textsf{Bcast}$$

$$\frac{J \subseteq \widetilde{t_r} \quad q(J) \quad \forall i\in\{A\}\cup J : X_i \subseteq \sigma(t_i,k) \wedge \sigma'(t_i,k) = [\![X_i;Y_i]\!](\sigma(t_i,k))}{(\nu\widetilde{m})\ \left\langle \sigma, \left(t_A[A]\{X_A;Y_A\} \twoheadrightarrow \&_{\mathbf{q}}(t_r[B_r]\widetilde{\{X_r;Y_r\}}) : k[l_h]\right); C \right\rangle \xrightarrow{t_A[A]\{X_A;Y_A\} \gg \&_{\mathbf{q}}(t_r[B_r]\widetilde{\{X_r;Y_r\}}):k[l_h]} (\nu\widetilde{m})\ \langle \sigma[\sigma'], C\rangle}\ \textsf{Sel}$$

$$\frac{\eta = \&_{\mathbf{q}}(t_r[A_r]\widetilde{\{X_r;Y_r\}}.e_r) \twoheadrightarrow t_B[B]\{X_B;Y_B\} : x : \langle k, op\rangle \quad \langle \sigma,\eta; C\rangle \xrightarrow{\xi}\!\!\!\!\rightarrow \langle \sigma',\eta'; C\rangle \quad \eta' ::_{\mathtt{ff}} \theta}{(\nu\widetilde{m})\ \langle \sigma,\eta; C\rangle \xrightarrow{\tau} (\nu\widetilde{m})\ \langle \sigma',\eta'; C\rangle}\ \textsf{RedD}$$

$$\frac{\eta = \&_{\mathbf{q}}(t_r[A_r]\widetilde{\{X_r;Y_r\}}.e_r) \twoheadrightarrow t_B[B]\{X_B;Y_B\} : x : \langle k, op\rangle \quad \langle \sigma,\eta; C\rangle \xrightarrow{\xi}\!\!\!\!\rightarrow \langle \sigma',\eta'; C\rangle \quad \eta' ::_{\mathtt{tt}} \theta}{(\nu\widetilde{m})\ \langle \sigma,\eta; C\rangle \xrightarrow{\theta(\eta')} (\nu\widetilde{m})\ \langle \sigma', C[\mathsf{op}(\theta)/x@t_{j+1}]\rangle}\ \textsf{RedE}$$

$$\frac{C \mathscr{R} C' \quad (\nu\widetilde{m})\ \langle \sigma, C'\rangle \xrightarrow{\lambda} (\nu\widetilde{n})\ \langle \sigma', C''\rangle \quad C'' \mathscr{R} C''' \quad \mathscr{R} \in \{\equiv, \simeq_C\}}{(\nu\widetilde{m})\ \langle \sigma, C\rangle \xrightarrow{\lambda} (\nu\widetilde{n})\ \langle \sigma', C'''\rangle}\ \textsf{Cong}$$

$$\frac{i = 1 \text{ if } e@t \downarrow \mathtt{tt}, \quad i = 2 \text{ otherwise}}{(\nu\widetilde{m})\ \langle \sigma, e@t ? C_1 : C_2\rangle \xrightarrow{\tau@t} (\nu\widetilde{m})\ \langle \sigma, C_i\rangle}\ \textsf{If}$$

Figure 7: GC_q: Operational Semantics - Complete rules

Figure 7 presents the complete operational semantics for GC_q.

A.4 Type System

Figure 8 presents the complete type system for GC_q.

Definition A.3 (State satisfaction). *The entailment relation between a state σ and a formula ψ (written $\sigma \models \psi$) is inductively defined as follows:*

$$\begin{array}{ll}
\sigma \models \mathtt{tt} & \iff \sigma \textit{ is defined} \\
\sigma \models t : k\,[A] \triangleright X & \iff (t,k,X) \in \sigma \\
\sigma \models \psi_1 \otimes \psi_2 & \iff \sigma = \sigma', \sigma'' \mid \sigma' \models \psi_1 \ \wedge\ \sigma'' \models \psi_2 \\
\sigma \models \psi_1 \oplus \psi_2 & \iff \sigma \models \psi_1 \textit{ or } \sigma \models \psi_2 \\
\sigma \models \psi \backslash \delta & \iff \exists \sigma' \textit{ s.t. } \sigma' \models \psi \ \wedge\ \sigma = \sigma' \backslash \delta \\
\sigma \models \exists x.\psi & \iff \sigma \models \psi[w/x] \textit{ (for some appropriate } w\textit{)}
\end{array}$$

B Proofs

B.1 Results related to states

Lemma B.1 (Validity: States). *If $\sigma \models \Psi$, then Ψ:* **formula** *and σ:* **state**.

Proof. By rule induction on the first hypothesis. □

Lemma B.2 (Weakening: States). *If $\sigma \models \Psi$, then $\sigma, (t\,[A]\,,k,X) \models \Psi \otimes t : k\,[A] \triangleright X$*

Proof. It follows by induction on the hypothesis. □

We write $(t,k,X) \in \mathsf{sf}(\Psi)$ when $t : k\,[A] \triangleright X$ is a subformulae of Ψ for any A.

Lemma B.3 (Strengthening: States). *If $\sigma, (t\,[A],k,X) \models \Psi$ and $t : k\,[A] \triangleright X \notin \mathsf{sf}(\Psi)$, then $\sigma \models \Psi$.*

Choreography Formation ($\Psi \vdash C$),

$$\frac{\Psi \otimes \mathbf{init}(t_r[\widetilde{A_r}]\{Y_r\}, t_s[\widetilde{B_r}]\{Y_s\}, k) \vdash C \quad \{\widetilde{t_s}, k\} \# (\mathbf{T}(\Psi) \cup \mathbf{K}(\Psi))}{\Psi \vdash t_r[\widetilde{A_r}]\{Y_r\} \ \mathbf{start}\ t_s[\widetilde{B_s}]\{Y_s\} : a(k); C}\ \textsf{Tinit} \qquad \frac{\Psi \vdash C \qquad \Psi' \vdash C'}{\Psi \oplus \Psi' \vdash C + C'}\ \textsf{Tsum}$$

$$\frac{\begin{array}{c}\forall J.\ s.t.(J \subseteq \widetilde{B} \wedge q(J)) \qquad \Psi = \bigotimes_{j\in J}(\psi_j) \otimes (\psi_A) \otimes \Psi' \qquad \phi = t_A : k[A] \triangleright X_A \bigotimes_{j\in J}(t_j : k[B_j] \triangleright X_j) \\ \phi' = t_A : k[A] \triangleright Y_A \bigotimes_{j\in J}(t_j : k[B_j] \triangleright Y_j) \quad (\bigotimes_{j\in J}\psi_j) \otimes (\phi \multimap \phi') \vdash \phi' \quad (t_A : k[A] \triangleright Y_A) \bigotimes_{j\in J}(t_j : k[B_j] \triangleright Y_j) \otimes \Psi' \vdash C\end{array}}{\Psi \vdash \left(t_A[A]\{X_A; Y_A\}.e \twoheadrightarrow \&_{\mathbf{q}}(t_r[B_r]\widetilde{\{X_r; Y_r\}} : x_r) : k\right); C}\ \textsf{Tbcast}$$

$$\frac{\begin{array}{c}\forall J.\ s.t.(J \subseteq \widetilde{A} \wedge q(J)) \qquad \Psi = \bigotimes_{j\in J}(\psi_j) \otimes (\psi_B) \otimes \Psi' \qquad \phi = t_B : k[B] \triangleright X_B \bigotimes_{j\in J}(t_j : k[A_j] \triangleright X_j) \\ \phi' = t_B : k[B] \triangleright Y_B \bigotimes_{j\in J}(t_j : k[A_j] \triangleright Y_j) \quad (\bigotimes_{j\in J}\psi_j) \otimes (\phi \multimap \phi') \vdash \phi' \quad (t_B : k[B] \triangleright Y_B) \bigotimes_{j\in J}(t_j : k[A_j] \triangleright Y_j) \otimes \Psi' \vdash C\end{array}}{\Psi \vdash \left(\&_{\mathbf{q}}(t_r[A_r]\widetilde{\{X_r; Y_r\}}.e_r) \twoheadrightarrow t_B[B]\{X_B; Y_B\} : x : \langle k, op\rangle\right); C}\ \textsf{Tred}$$

$$\frac{((\text{same as } (\textsc{Tbcast})))}{\Psi \vdash \left(t_1[A]\{X_A; Y_A\} \twoheadrightarrow \&_{\mathbf{q}}(t_r[\widetilde{B_r}]\{X_r; Y_r\}) : k[l_h]\right); C}\ \textsf{Tsel} \qquad \frac{}{\Psi \vdash 0}\ \textsf{Tinact} \qquad \frac{\Psi \vdash C_1 \quad \Psi \vdash C_2}{\Psi \vdash e@t?C_1 : C_2}\ \textsf{Tcond}$$

State Formation ($\sigma : \mathbf{state}$),

$$\frac{}{\emptyset : \mathbf{state}}\ \textsf{TS1} \qquad \frac{\sigma : \mathbf{state} \quad \sigma(t[A], k) = \emptyset \quad X \in dom(\Sigma)}{\sigma, (t[A], k, X) : \mathbf{state}}\ \textsf{TS2} \qquad \frac{\sigma : \mathbf{state} \quad (t[A], k, X) \in \sigma \quad Y \in dom(\Sigma)}{[\![X; Y]\!](\sigma(t, k)) : \mathbf{state}}\ \textsf{TS3} \qquad \frac{\sigma : \mathbf{state} \quad \delta : \mathbf{state}}{\sigma \backslash \delta : \mathbf{state}}\ \textsf{TS4}$$

Formulae Formation ($\Psi : \mathbf{formula}$),

$$\frac{}{\mathtt{tt} : \mathbf{formula}}\ \textsf{TF1} \qquad \frac{}{t : k[A] \triangleright X : \mathbf{formula}}\ \textsf{TF2} \qquad \frac{\psi : \mathbf{formula} \quad \psi' : \mathbf{formula} \quad \circ \in \{\otimes, \oplus\}}{\psi \circ \psi' : \mathbf{formula}}\ \textsf{TF3} \qquad \frac{\psi : \mathbf{formula} \quad \delta : \mathbf{state}}{\psi \backslash \delta : \mathbf{formula}}\ \textsf{TF4}$$

Figure 8: GC_q: Type checking - Complete rules

Proof. It follows by induction on the first hypothesis. □

Lemma B.4 (Update: States). *If $\sigma \models \Psi$ and $\sigma' \models \Psi'$, then $\sigma[\sigma'] \models (\Psi \backslash \delta) \otimes \Psi'$, where $\delta = \{(t,k,X) \mid (t,k,X) \in \sigma \wedge (t,k,Y) \in \sigma'\}$.*

Proof. It follows directly from the hypotheses and definitions A.2 and A.3. □

B.2 Results related to choreographies

Lemma B.5 (Weakening: choreographies). *Let $\psi : \mathbf{formula}$. If $\Psi \vdash C$, then $\Psi \otimes \psi \vdash C$.*

Proof. By rule induction on the hypothesis. □

Lemma B.6 (Strengthening: choreographies). *If $\Psi \otimes t : k[A] \triangleright X \vdash C$ and $X \notin \mathsf{fform}(C)$, then $\Psi \vdash C$.*

Proof. By rule induction on the first hypothesis. □

Lemma B.7 (Substitution). *Let t be a term, if $\Psi \vdash C$, then $\Psi \vdash C[t/x]$.*

Proof. By rule induction on the first hypothesis. Note that $x \notin \Psi$. □

Lemma B.8 (Subject Congruence). *If $C \equiv C'$ and $\Psi \vdash C$, then $\Psi \vdash C'$.*

Proof. It proceeds by induction on the depth of the first premise. □

Lemma B.9 (Inversion Lemma). *Let $\Psi \vdash C$ then either:*

- $C = \eta; C'$, *and:*
  - $\eta = t_r[\widetilde{A_r}]\{\widetilde{Y_r}\}$ **start** $t_s[\widetilde{B_s}]\{\widetilde{Y_s}\} : a(k)$ *and* $\Psi \otimes \mathbf{init}(t_r[\widetilde{A_r}]\{\widetilde{Y_r}\}, t_s[\widetilde{B_r}]\{\widetilde{Y_s}\}, k) \vdash C'$, *and* $\{\widetilde{t_s}, k\} \# (\mathbf{T}(\Psi) \cup \mathbf{K}(\Psi))$, *or*
  - $\eta = t_A[A]\{X_A; Y_A\}.e \to \&_{\mathbf{q}}(t_r[B_r]\widetilde{\{X_r; Y_r\}} : x_r) : k$ *and* $\forall J.\ s.t. (J \subseteq \widetilde{B} \wedge q(J))$, $\Psi = \bigotimes_{j \in J}(\psi_j) \otimes (\psi_A) \otimes \Psi'$, *and* $\phi = t_A : k[A] \triangleright X_A \bigotimes_{j \in J}(t_j : k[B_j] \triangleright X_j)$, *and* $\phi' = t_A : k[A] \triangleright Y_A \bigotimes_{j \in J}(t_j : k[B_j] \triangleright Y_j)$, *and* $(\bigotimes_{j \in J} \psi_j) \otimes (\phi \multimap \phi') \vdash \phi'$, *and* $(t_A : k[A] \triangleright Y_A) \bigotimes_{j \in J}(t_j : k[B_j] \triangleright Y_j) \otimes \Psi' \vdash C'$, *or*
  - $\eta = \&_{\mathbf{q}}(t_r[A_r]\widetilde{\{X_r; Y_r\}}.e_r) \to t_B[B]\{X_B; Y_B\} : x : \langle k, op \rangle$, *and* $\forall J.\ s.t. (J \subseteq \widetilde{A} \wedge q(J))$, *and* $\Psi = \bigotimes_{j \in J}(\psi_j) \otimes (\psi_B) \otimes \Psi'$, *and* $\phi = t_B : k[B] \triangleright X_B \bigotimes_{j \in J}(t_j : k[A_j] \triangleright X_j)$, *and* $\phi' = t_B : k[B] \triangleright Y_B \bigotimes_{j \in J}(t_j : k[A_j] \triangleright Y_j)$, *and* $(\bigotimes_{j \in J} \psi_j) \otimes (\phi \multimap \phi') \vdash \phi'$, *and* $(t_B : k[B] \triangleright Y_B) \bigotimes_{j \in J}(t_j : k[A_j] \triangleright Y_j) \otimes \Psi' \vdash C'$, *or*
  - $\eta = t_A[A]\{X_A; Y_A\}.e \to \&_{\mathbf{q}}(t_r[B_r]\widetilde{\{X_r; Y_r\}} : x_r) : k$ *and* $\forall J.\ s.t. (J \subseteq \widetilde{B} \wedge q(J))$, $\Psi = \bigotimes_{j \in J}(\psi_j) \otimes (\psi_A) \otimes \Psi'$, *and* $\phi = t_A : k[A] \triangleright X_A \bigotimes_{j \in J}(t_j : k[B_j] \triangleright X_j)$, *and* $\phi' = t_A : k[A] \triangleright Y_A \bigotimes_{j \in J}(t_j : k[B_j] \triangleright Y_j)$, *and* $(\bigotimes_{j \in J} \psi_j) \otimes (\phi \multimap \phi') \vdash \phi'$, *and* $(t_A : k[A] \triangleright Y_A) \bigotimes_{j \in J}(t_j : k[B_j] \triangleright Y_j) \otimes \Psi' \vdash C'$, *or*
- $C = C' + C''$, *and* $\Psi = \Psi' \oplus \Psi'$, *and* $\Psi' \vdash C'$ *and* $\Psi'' \vdash C''$, *or*
- $C = e@t?C_1 : C_2$, *and* $\Psi \vdash C_1$, *and* $\Psi \vdash C_2$, *or*
- $C = 0$.

Proof. By case analysis on the type formation rules. □

Lemma B.10 (Subject Swap). *If* $C \simeq_C C'$ *and* $\Psi \vdash C$, *then* $\Psi \vdash C'$.

Proof. It proceeds by induction on the depth of the first premise. Most of the cases are straightforward except $\eta; (\eta'; C) \simeq_C \eta'; (\eta; C)$, which requires the application of Lemma B.9 and case analysis. □

Let $\xi(\Psi)$ the update function in Ψ, defined as $[\![X; Y]\!](t : k[A] \triangleright X)$ if $\Psi = t : k[A] \triangleright X$ and inductively for any other case.

Lemma B.11 (Subject Effect). *Let* $\xi = (t, k) : X :: Y$, *if* $\langle \sigma, \eta; C \rangle \xrightarrow{\xi}\!\!\!\!\rightarrow \langle \sigma', \eta'; C \rangle$, $\Psi \vdash \eta; C$ *and* $\sigma \models \Psi$, *then* $\exists \Psi' = \xi(\Psi).\ \Psi' \vdash \eta'; C$, *and* $\sigma' \models \Psi'$.

Proof. It follows by rule induction on the first hypothesis. □

Theorem 3.2 (Type Preservation). Statement on page 6.

Proof. It follows by rule induction on the first hypothesis. We have eight cases.
Case (Init) rule: Standard inversion/formation rules. It requires the state update lemma (Lemma B.4).
Case (BCast), (Sel) rules: Standard Inversion/formation rules. Process typing requires substituion (Lemma B.7) and state typing require state update (Lemma B.4).
Case (RedD), (RedE) rules: State and process typing of the transition rely on Lemma B.11. Moreover, process typing requires the use of Lemma B.7.
Case (Cong) rule: One case for each congruence relation. $\equiv$ requires subject congruence (Lemma B.8), and $\simeq_C$ requires subject swap (Lemma B.10).
Case (If),(Sum) rules: they follow a standard induction. □

Theorem 3.3 (Progress). Statement on page 6.

Proof. Proof by contradiction. Let us assume that $\Psi \vdash C$, $\sigma \models \Psi$ and $C \not\equiv 0$ and $(\nu \widetilde{m})\ \langle \sigma, C \rangle \overset{\lambda}{\not\rightarrow}$. We proceed by case analysis on the structure of C to show that such C does not exists. □