Choreographies in the Wild[☆]

Massimo Bartoletti[a], Julien Lange[b,*], Alceste Scalas[a], Roberto Zunino[c]

[a] *Dipartimento di Matematica e Informatica, Università degli Studi di Cagliari, Italy*
[b] *Department of Computing, Imperial College London, UK*
[c] *Dipartimento di Matematica, Università degli Studi di Trento, Italy*

Abstract

We investigate the use of choreographies in distributed scenarios where, as in the real world, mutually distrusting (and possibly dishonest) participants may be unfaithful to their expected behaviour. In our model, each participant advertises its promised behaviour as a *contract*. Participants may interact through multiparty sessions, created when their contracts allow to synthesise a *choreography*. We show that systems of *honest* participants (which always adhere to their contracts) enjoy *progress* and *session fidelity*.

Keywords: contracts, choreographies, session types

1. Introduction

Designing reliable large-scale distributed systems is a difficult challenge. Several intrinsic issues (e.g. concurrency, physical distribution, non-reproducibility of bugs, *etc.*) make it extremely unlikely that any non-trivial distributed application actually works as its designers intended. To cope with this grand challenge, a lot of foundational and applied research has been carried over in the last 15 years.

One of the most notable approaches is that which studies *session types* [1, 2, 3] as an abstraction of the communication behaviour of distributed applications. In the *top-down* approach to session types, the designer of a distributed application specifies its overall communication behaviour through a *choreography*, which validates some global properties of the application (e.g. safety and deadlock-freedom). However, a distributed application is nothing more than the composition of a number services, running on different nodes in the network. In order to globally enjoy the properties validated by the choreography, all the components forming the application have to be verified; this can be done e.g. by projecting the choreography to *end-point types*, against which these services are actually type-checked.

The above-mentioned top-down approach is quite suitable in case the designers have complete control over the application, e.g., when they develop all the needed services. However, in many realistic situations this is not the case: for instance, one might need to integrate with already-existing third-party services, e.g., e-commerce facilities, maps, *etc.* In this case, the top-down approach might not work as expected, because it could be impossible to adapt the end-point type projected from the choreography to the actual interface of the third-party service. In the last few years, approaches mixing top-down and *bottom-up* composition have been studied to cope with this kind of situation [4, 5, 6].

Dropping the (unrealistic) hypothesis that a distributed application is built of services coming from a single provider has a further consequence. We can no longer suppose that services live in a "civil society" where they smoothly collaborate with each other; rather, they live in a "wild" environment (like Hobbes'

[☆] Work partially supported by Aut. Region of Sardinia under grants L.R.7/2007 CRP-17285 (TRICS) and P.I.A. 2010 project "Social Glue", MIUR PRIN 2010-11 project "Security Horizons", EU COST Action IC1201 (BETTY), and EPSRC EP/K034413/1.

[*] *Corresponding author.* Department of Computing, Imperial College London, 180 Queen's Gate, SW7 2AZ, London e-mail: `j.lange@imperial.ac.uk`

condition of mere nature [7]) where they must compete for resources, and possibly diverge from the intended behaviour in case they find it convenient to do so. Indeed, this situation is quite reasonable in inter-organisational scenarios, where mistrusted providers compete to achieve conflicting goals. Although some proposals like OASIS [8] try do deal with this issue by prescribing runtime monitoring and logging, no general techniques exist for checking that services actually adhere to their specifications, unless one controls the "bottom layer" of the software stack [9].

To overcome the state of nature, and move towards the "social contract" (as in Hobbes' Leviathan), some recent works have proposed to discipline the interaction of services through *contracts* [10, 11]. The idea is a contract-oriented, bottom-up composition, where only those services with *compliant* contracts can establish sessions through which they interact. To give some intuition about the underlying dynamics, let us assume three participants `A`, `B` and `K`, where the latter plays the role of a *contract broker*. A possible interaction among these participants could be the following:

1. `A` advertises its contract $c_{\mathtt{A}}$ to `K`;
2. `B` advertises its contract $c_{\mathtt{B}}$ to `K`;
3. `K` checks if $c_{\mathtt{A}}$ and $c_{\mathtt{B}}$ are compliant; if so, it creates a fresh session s between `A` and `B`;
4. at this point, `A` and `B` can start interacting through s, by performing the communications prescribed by their contracts. While doing that, they can advertise other contracts, so establishing other sessions (possibly, with different participants).

One key aspect of this contract-oriented approach is that the runtime behaviour of a service may diverge from the advertised one. For instance, participant `B` above may fail to perform some of the actions in $c_{\mathtt{B}}$ (either maliciously or accidentally); if `A` assumes that `B` respects its contract, then `A` may get stuck on session s, and consequently it may fail to fulfil its contracts in some other sessions (so possibly damaging itself or other participants). A notable question then arises: how can designers guarantee that their services are *honest*, in that they always respect the advertised contracts, despite the possible dishonesty of the services they may interact with?

In this paper we address the issue of reconciling the top-down and the bottom-up approaches to service composition in wild environments. To this purpose, we propose a combination of two approaches. Like in the contract-oriented approach [10], we assume that applications are built bottom-up, by composing services whose advertised contracts admit agreements. However, unlike in previous contract-oriented calculi, service contracts are *local session types*, in a form similar to those used in [12, 13]. A set of contracts admits an agreement whenever it is possible to synthesise from them a choreography — i.e. a global type — whose projections are the contracts themselves [6]. The existence of such a choreography ensures that, if all services behave honestly, the overall application enjoys *progress* and *session fidelity*. However, by the wilderness assumption, the actual behaviour of services may deviate from the promised one. In this case, progress no longer holds, but at least we can single out the services which are responsible for contract violations.

Contributions. We have combined notions and techniques from the session types literature to devise a model of distributed systems where services may be composed bottom-up according to their contracts, and they may deviate from the expected behaviour at runtime. Contracts are expressed as local session types, and multiparty agreements are constructed by adapting the synthesis technique for choreographies of [6]. This allows us to import some results from the session types literature — in particular, the existence of a choreography ensures that contractual agreements enjoy safety and progress (Theorem 3.9). Using choreographies as the basis for agreements is quite flexible: indeed, it allows us to impose further constraints before establishing sessions, e.g. on the number of involved participants, whether or not the session may terminate, *etc.*

We model services in the CO_2 calculus [14], which is adapted here to use (multiparty) local session types as contracts. We define when a service is *culpable* (i.e., responsible for the next move) in a session, and we show that at least one culpable participant exists in each non-terminated session (Theorem 4.10). If

$\mathtt{a}, \mathtt{b}, \ldots$	Participant identifiers, union of:	$u, v, \ldots$	Session identifiers, union of:
$\mathtt{A}, \mathtt{B}, \ldots \in \mathbb{P}$	Participant names	$s, s', t, \ldots \in \mathbb{S}$	Session names
$\mathtt{a}, \mathtt{b}, \ldots \in \mathcal{P}$	Participant variables	$x, y, \ldots \in \mathcal{S}$	Session variables
$\mathsf{e}, \mathsf{e}', \ldots$	Sorts	$\gamma, \gamma', \ldots \in \Gamma$	Unsigned contracts
$c, c', \ldots$	Contracts	$K, K', \ldots$	Latent contracts
$T, T', \ldots$	Systems of contracts	$P, P', \ldots$	Processes
$\mathcal{G}, \mathcal{G}', \ldots$	Global types	$S, S', \ldots$	Systems

Table 1: Summary of notation.

a system gets stuck, it is always possible to identify which participants violated their contracts. We then adapt the notion of *honesty* of [11]. We show that an honest service is always able to satisfy its contracts by firing some of its actions, even when the other services do not cooperate (Theorem 6.1). Furthermore, we show that systems of honest services preserve the properties enjoyed by choreographies: in particular, *session fidelity* (Theorem 6.2) ensures that no extra-contractual interactions are possible in a session, and that honest participants will eventually perform the actions required by their contracts.

We present a case study where the key features of our framework are put into practice. Multiparty sessions, honest participants, and choreography-based contractual agreements are used to model distributed systems where brokers let participants start a new session only if they are advertising contracts such that, as a whole, they implement a class of gossip protocols [15].

Synopsis. The rest of the paper is structured as follows. In Section 2, we sketch an example that we will use through the paper to illustrate our framework. In Section 3, we introduce a contract model based on multiparty session types, and we define contractual agreement as choreography synthesis. In Section 4 we present CO_2, suitably adapted to deal with the contracts of Section 3, and we highlight some of its main features. In Section 5, we formalise the notion of honesty. Our main technical results are presented in Section 6, where we show how global properties of choreographies are projected to (honest) CO_2 systems [14]. In Section 7 we test the suitability of our framework through a large case study. Finally, in Section 8 we discuss some related work, and in Section 9 draw some conclusions.

2. A Motivating Example

We introduce a running example which we will use through the paper to illustrate our framework. To make symbols lookup easier, we have summarised the syntactic categories and some notation in Table 1; some of the symbols defined therein will only be used in later sections.

An online store $\mathtt{A}$ allows two (still unknown) buyers $\mathtt{b}_1$ and $\mathtt{b}_2$ to make a joint purchase through a simple protocol: after they both request the same item, a quote is sent to $\mathtt{b}_1$, who is then expected to either place an order (order) or end the session (bye); the store also promises to notify $\mathtt{b}_2$ about whether the order was placed (ok) or cancelled (bye). The behaviour promised by $\mathtt{A}$ is described by the following contract:

$$c_{\mathtt{A}} \;=\; \mathtt{b}_1?\mathsf{req};\ \mathtt{b}_2?\mathsf{req};\ \mathtt{b}_1!\mathsf{quote};\ (\mathtt{b}_1?\mathsf{order};\ \mathtt{b}_2!\mathsf{ok} \;+\; \mathtt{b}_1?\mathsf{bye};\ \mathtt{b}_2!\mathsf{bye})$$

What kind of contracts would be compliant with $c_{\mathtt{A}}$? One answer consists in the following contracts, advertised by buyers $\mathtt{B}_1$ and $\mathtt{B}_2$:

$$c_{\mathtt{B}_1} \;=\; \mathtt{a}!\mathsf{req};\mathtt{a}?\mathsf{quote};\ (\mathtt{b}_2'!\mathsf{ok};\mathtt{a}!\mathsf{order} \oplus \mathtt{b}_2'!\mathsf{bye};\mathtt{a}!\mathsf{bye})$$
$$c_{\mathtt{B}_2} \;=\; \mathtt{a}'!\mathsf{req};\ (\mathtt{b}_1'?\mathsf{ok};\mathtt{a}'?\mathsf{ok} + \mathtt{b}_1'?\mathsf{bye};\mathtt{a}'?\mathsf{bye})$$

Here, $\mathtt{B}_1$ promises to send the request to a (still unknown) store $\mathtt{a}$, wait for the quote, and then notify the other buyer $\mathtt{b}_2'$ before accepting or rejecting the store offer; symmetrically, $\mathtt{B}_2$ promises to send the request to the store $\mathtt{a}'$, and then expects to receive the same notification (either ok or bye) from both the other buyer $\mathtt{b}_1'$ and the store itself. Each contract represents the local viewpoint of the participant who advertises

it: in particular, $c_{\mathtt{A}}$ represents the local viewpoint of the store, and thus it does not (and indeed, it cannot) capture the communications between $\mathtt{B}_1$ and $\mathtt{B}_2$.

An *agreement* among $c_{\mathtt{A}}$, $c_{\mathtt{B}_1}$ and $c_{\mathtt{B}_2}$ may be found after instantiating the participant variables (in each contract) to actual names, through the substitutions $\{^{\mathtt{A}}/_{\mathtt{a},\mathtt{a}'}\}$, $\{^{\mathtt{B}_1}/_{\mathtt{b}_1,\mathtt{b}'_1}\}$ and $\{^{\mathtt{B}_2}/_{\mathtt{b}_2,\mathtt{b}'_2}\}$. The contracts admit an agreement because a *choreography* exists which can be projected back to $c_{\mathtt{A}}$, $c_{\mathtt{B}_1}$ and $c_{\mathtt{B}_2}$. The existence of such a choreography guarantees progress and safety of the contractual agreement (Theorem 3.9). We can exploit the technique in [6] to synthesise a choreography from a set of contracts. In our example, we obtain:

$$\begin{array}{rcl} \mathcal{G}_{\mathtt{AB}_1\mathtt{B}_2} & = & \mathtt{B}_1 \to \mathtt{A} : \mathsf{req} \,;\, \mathtt{B}_2 \to \mathtt{A} : \mathsf{req} \,;\, \mathtt{A} \to \mathtt{B}_1 : \mathsf{quote} \,; \\ & & (\mathtt{B}_1 \to \mathtt{B}_2 : \mathsf{ok} \,;\, \mathtt{B}_1 \to \mathtt{A} : \mathsf{order} \,;\, \mathtt{A} \to \mathtt{B}_2 : \mathsf{ok} \quad + \quad \mathtt{B}_1 \to \mathtt{B}_2 : \mathsf{bye} \,;\, \mathtt{B}_1 \to \mathtt{A} : \mathsf{bye} \,;\, \mathtt{A} \to \mathtt{B}_2 : \mathsf{bye}) \end{array}$$

Global type $\mathcal{G}_{\mathtt{AB}_1\mathtt{B}_2}$ says that $\mathtt{B}_1$ and $\mathtt{B}_2$ send a req*uest* to $\mathtt{A}$, consecutively; then $\mathtt{A}$ replies to $\mathtt{B}_1$ with a quote. Subsequently, $\mathtt{B}_1$ chooses whether to accept or reject the quote. In the former case, $\mathtt{B}_1$ notifies $\mathtt{B}_2$ (ok message) then $\mathtt{A}$ (order message), who sends a confirmation to $\mathtt{B}_2$ (ok message). In the latter case, $\mathtt{B}_1$ sends bye to both $\mathtt{B}_2$ and $\mathtt{A}$, and $\mathtt{A}$ confirms the cancellation to $\mathtt{B}_2$ by sending bye as well.

However, in realistic distributed scenarios, the existence of a contractual agreement among participants does not guarantee that progress and safety will also hold at runtime: in fact, a participant may advertise a contract promising some behaviour, and then fail to respect it — either maliciously or accidentally. Such failures may then cascade on other participants, e.g., if they remain stuck waiting for a promised message that is never sent.

This sort of situations can be modelled using the CO_2 calculus [14]. For instance, a CO_2 system for the above scenario will have the following form:

$$\mathtt{A}[(x)(\mathtt{b}_1, \mathtt{b}_2)\, \mathsf{tell}_{\mathtt{A}}{\downarrow_x}\, c_{\mathtt{A}} \,.\, \mathsf{fuse} \,.\, P_{\mathtt{A}}] \;\mid\; \mathtt{B}_1[(y)(\mathtt{a}, \mathtt{b}'_2)\, \mathsf{tell}_{\mathtt{A}}{\downarrow_y}\, c_{\mathtt{B}_1} \,.\, P_{\mathtt{B}_1}] \;\mid\; \mathtt{B}_2[(z)(\mathtt{a}, \mathtt{b}'_1)\, \mathsf{tell}_{\mathtt{A}}{\downarrow_z}\, c_{\mathtt{B}_2} \,.\, P_{\mathtt{B}_2}]$$

Here, participant $\mathtt{A}$ advertises the contract $c_{\mathtt{A}}$ to itself via the primitive $\mathsf{tell}_{\mathtt{A}}{\downarrow_x}\, c_{\mathtt{A}}$, where x is used as a session handle for interacting with other participants. $\mathtt{B}_1$ and $\mathtt{B}_2$ advertise their respective contracts to $\mathtt{A}$ with a similar invocation.

In this example, $\mathtt{A}$ also plays the role of *contract broker*, which collects the advertised contracts, while looking for an agreement. Once an agreement is found, the fuse prefix is fired, and a new session is established among all the involved participants. At this point, the participants may realise the agreed contracts — or they may even choose not to. In fact, we will see that when the contracts are violated, the calculus allows for responsible participants to be always ruled out (Theorem 4.10). In systems where participants are *honest* (i.e. they always fulfil their contracts) the progress and safety of the contractual agreement are also reflected in the runtime behaviour of the CO_2 system (Theorem 6.1 and Theorem 6.2).

Other possible agreements. Our framework allows for modelling a wide variety of scenarios. For instance, a participant $\mathtt{B}_{12}$ may impersonate both buyers, and promise to always accept the store offer, by advertising the following contract:

$$c_{\mathtt{B}_{12}} \;=\; \mathtt{a}''!\mathsf{req};\mathtt{a}''!\mathsf{req};\mathtt{a}''?\mathsf{quote};\mathtt{a}''!\mathsf{order};\mathtt{a}''?\mathsf{ok}$$

where the request to the store $\mathtt{a}''$ is sent twice (i.e., once for each impersonated customer). In this case, if we instantiate the participant variables in $c_{\mathtt{A}}$ and $c_{\mathtt{B}_{12}}$ with substitutions $\{^{\mathtt{A}}/_{\mathtt{a}''}\}$, $\{^{\mathtt{B}_{12}}/_{\mathtt{b}_1,\mathtt{b}_2}\}$, we can find an agreement by synthesising the following choreography:

$$\mathcal{G}_{\mathtt{AB}_{12}} \quad = \quad \mathtt{B}_{12} \to \mathtt{A} : \mathsf{req} \,;\, \mathtt{B}_{12} \to \mathtt{A} : \mathsf{req} \,;\, \mathtt{A} \to \mathtt{B}_{12} : \mathsf{quote} \,;\, \mathtt{B}_{12} \to \mathtt{A} : \mathsf{order} \,;\, \mathtt{A} \to \mathtt{B}_{12} : \mathsf{ok}$$

This scenario may be modelled by a CO_2 system of the form:

$$S_2 \;=\; \mathtt{A}[(x)(\mathtt{b}_1, \mathtt{b}_2)\, \mathsf{tell}_{\mathtt{A}}{\downarrow_x}\, c_{\mathtt{A}} \,.\, \mathsf{fuse} \,.\, P_{\mathtt{A}}] \;\mid\; \mathtt{B}_{12}[(w)(\mathtt{a}'')\, \mathsf{tell}_{\mathtt{A}}{\downarrow_w}\, c_{\mathtt{B}_{12}} \,.\, P_{\mathtt{B}_{12}}]$$

where the fuse prefix can now create a session involving $\mathtt{A}$ and $\mathtt{B}_{12}$.

The participants in the CO_2 systems S_1 and S_2 may also be combined, so to obtain:

$$\begin{array}{rcl} S_{12} & = & \mathtt{A}[(x)(\mathtt{b}_1, \mathtt{b}_2)\, \mathsf{tell}_{\mathtt{A}}{\downarrow_x}\, c_{\mathtt{A}} \,.\, \mathsf{fuse} \,.\, P_{\mathtt{A}}] \;\mid\; \mathtt{B}_1[(y)(\mathtt{a}, \mathtt{b}'_2)\, \mathsf{tell}_{\mathtt{A}}{\downarrow_y}\, c_{\mathtt{B}_1} \,.\, P_{\mathtt{B}_1}] \\ & & \mid\; \mathtt{B}_2[(z)(\mathtt{a}, \mathtt{b}'_1)\, \mathsf{tell}_{\mathtt{A}}{\downarrow_z}\, c_{\mathtt{B}_2} \,.\, P_{\mathtt{B}_2}] \;\mid\; \mathtt{B}_{12}[(w)(\mathtt{a}'')\, \mathsf{tell}_{\mathtt{A}}{\downarrow_w}\, c_{\mathtt{B}_{12}} \,.\, P_{\mathtt{B}_{12}}] \end{array}$$

In this case, after all contracts have been advertised to $\mathtt{A}$, either a session corresponding to $\mathcal{G}_{\mathtt{AB}_1\mathtt{B}_2}$, or to $\mathcal{G}_{\mathtt{AB}_{12}}$ may take place, thus involving a different number of participants depending on which contracts are fused. As mentioned above, the honesty of $\mathtt{A}$ can not depend on which of the two agreements is chosen: to be honest, $\mathtt{A}$ must respect its contracts whatever agreement is found with the other participants.

3. A Choreography-Based Contract Model

In this section we exploit concepts and results from the session types literature to devise a multiparty contract model. In Sections 3.1 and 3.2, we present the syntax and semantics of contracts. In Section 3.3, we define the crucial notion of *agreement*. Intuitively, a set of contracts admits an agreement when there exists a choreography which can be projected back to such contracts. This choreography can be inferred through a type system, adapted from [16], which guarantees safety properties (Theorem 3.9). In Section 3.4, we introduce a notion of *culpability*. Theorem 3.12 ensures that, if a system of contracts admits an agreement, then at each step of its execution some participants will be "culpable", i.e. responsible for making the next step in a non-terminated system of contract.

3.1. Local session types as Contracts

We express contracts using the syntax of local session types, in the style of [12, 13]. Let $\mathbb{P}$ and $\mathcal{P}$ be disjoint sets of, respectively, *participant names* (ranged over by $\mathtt{A}, \mathtt{B}, \ldots$) and *participant variables* (ranged over by $\mathtt{a}, \mathtt{b}, \ldots$), and let $\underline{\mathtt{a}}, \underline{\mathtt{b}}$ range over $\mathbb{P} \cup \mathcal{P}$. We abstract from the actual data exchanged between participants by using *sorts* $\mathtt{e}$, $\mathtt{e}'$, ... We denote (individual) *contracts* with c, c', ..., and *systems* of contracts with T, T', ... We write $\mathcal{P}(T)$ for the set of participant names in T.

Definition 3.1 (Contracts). *The syntax of contracts is given below (we use the colour blue for contracts and green for* systems *of contracts).*

c	::=	$\bigoplus_{i\in I} \underline{\mathtt{a}}_i!\mathtt{e}_i; c_i$	*where* $\forall i \neq j \in I : (\underline{\mathtt{a}}_i, \mathtt{e}_i) \neq (\underline{\mathtt{a}}_j, \mathtt{e}_j)$	(Internal choice)
	$\mid$	$\sum_{i\in I} \underline{\mathtt{a}}?\mathtt{e}_i; c_i$	*where* $\forall i \neq j \in I : \mathtt{e}_i \neq \mathtt{e}_j$	(External choice)
	$\mid$	$\mu\, \mathbf{x}.c$		(Recursion)
	$\mid$	$\mathbf{x}$		(Recursion variable)
	$\mid$	$\mathbf{0}$		(End)
T	::=	$T \mid T'$	*where* $\mathcal{P}(T) \cap \mathcal{P}(T') = \varnothing$	(System composition)
	$\mid$	$\mathtt{A}\langle c \rangle$		(Participant's contract)
	$\mid$	$(\mathtt{AB}) : \rho$	*where* $\mathtt{A} \neq \mathtt{B}$ *and* ρ *is a sequence of sorts*	(Queue)
	$\mid$	$\mathbf{0}$		(End)

Intuitively, in an internal choice, after sending the message $\mathtt{e}_i$ to participant $\underline{\mathtt{a}}_i$, behaviour c_i takes place; whereas in an external choice if a message of sort $\mathtt{e}_i$ is received from $\underline{\mathtt{a}}$, then behaviour c_i takes place. We write $\mathrm{fv}(c)$ for the (free) participant variables in c. We identify empty internal and external sums, i.e. $\mathbf{0} = \bigoplus_{i\in\varnothing} \underline{\mathtt{a}}_i!\mathtt{e}_i; c_i = \sum_{i\in\varnothing} \underline{\mathtt{a}}?\mathtt{e}_i; c_i$.

A system of contracts T may be either: (i) a parallel composition of systems $T \mid T'$; or (ii) a *named* contract $\mathtt{A}\langle c \rangle$, saying that participant $\mathtt{A}$ promises to behave according to c; (iii) a *queue* $(\mathtt{AB}) : \rho$ of messages from $\mathtt{A}$ to $\mathtt{B}$; we write [] for the empty sequence of message. In a system T, we assume that there are exactly two uni-directional queues per pair of participants; that is, two participants $\mathtt{A}$ and $\mathtt{B}$ can only communicate via queues named $\mathtt{AB}$ (from $\mathtt{A}$ to $\mathtt{B}$) and $\mathtt{BA}$ (from $\mathtt{B}$ to $\mathtt{A}$). Whenever we refer to a system T, we assume that it respects the conditions in Definition 3.1 as well as those above.

$$\mathtt{A}\langle \mathtt{B}!\mathtt{e};c_0 \oplus c_1\rangle \mid (\mathtt{AB}):\rho \xrightarrow{\mathtt{A}\to\mathtt{B}:\mathtt{e}}\!\!\!\!\!\rightarrow \mathtt{A}\langle c_0\rangle \mid (\mathtt{AB}):\rho\cdot\mathtt{e}$$

$$\mathtt{A}\langle \mathtt{B}?\mathtt{e};c_0 + c_1\rangle \mid (\mathtt{BA}):\mathtt{e}\cdot\rho \xrightarrow{\mathtt{A}\leftarrow\mathtt{B}:\mathtt{e}}\!\!\!\!\!\rightarrow \mathtt{A}\langle c_0\rangle \mid (\mathtt{BA}):\rho$$

$$\mu\,\mathbf{x}.c \equiv c\,\{{}^{\mu\,\mathbf{x}.c}/_{\mathbf{x}}\} \qquad \mathtt{A}\langle \mathbf{0}\rangle \equiv \mathbf{0} \qquad \frac{T_1 \xrightarrow{\lambda}\!\!\!\!\!\rightarrow T_1'}{T_1 \mid T_2 \xrightarrow{\lambda}\!\!\!\!\!\rightarrow T_1' \mid T_2}$$

commutative monoidal laws for $\mid$, $\oplus$ and $+$

Figure 1: Semantics of systems of contracts.

3.2. Semantics

The semantics of systems of contracts is given in Figure 1, where labels λ have either the form $\mathtt{A}\to\mathtt{B}:\mathtt{e}$ or $\mathtt{A}\leftarrow\mathtt{B}:\mathtt{e}$. The label $\mathtt{A}\to\mathtt{B}:\mathtt{e}$ means that a message of sort $\mathtt{e}$ is sent by participant $\mathtt{A}$ to $\mathtt{B}$; instead, the label $\mathtt{A}\leftarrow\mathtt{B}:\mathtt{e}$ means that a message of sort $\mathtt{e}$ is received by $\mathtt{A}$ from $\mathtt{B}$. The first rule in Figure 1 says that, after an internal choice, $\mathtt{A}$ puts a message $\mathtt{e}$ on its queue for participant $\mathtt{B}$. The second rule says that, in an external choice, $\mathtt{A}$ can receive a message (of the expected sort) from an input queue $\mathtt{BA}$. The semantics is assumed up-to the congruence relation $\equiv$ defined in Figure 1.

Notation 3.2. *Hereafter, we will use the following notation. We write:*

- $T \xrightarrow{\mathtt{A}\leftrightharpoons\mathtt{B}:\mathtt{e}}\!\!\!\!\!\rightarrow T'$ *when either* $T \xrightarrow{\mathtt{A}\to\mathtt{B}:\mathtt{e}}\!\!\!\!\!\rightarrow T'$ *or* $T \xrightarrow{\mathtt{A}\leftarrow\mathtt{B}:\mathtt{e}}\!\!\!\!\!\rightarrow T'$.
- $T \twoheadrightarrow T'$ *when* $T \xrightarrow{\mathtt{A}\leftrightharpoons\mathtt{B}:\mathtt{e}}\!\!\!\!\!\rightarrow T'$, *and the label is unimportant.*
- $\mathcal{P}(T)$ *(resp.* $\mathcal{C}(T)$*) for the set of participant (resp. queue) names in* T.
- $\mathrm{Q}(T)$ *for the parallel composition of the empty queues connecting all distinct pairs of participants appearing in* T.
- $T(\mathtt{A})$ *for the contract of* $\mathtt{A}$ *in* T, *and* $T[\mathtt{AB}]$ *for the content of the queue* $\mathtt{AB}$, *i.e.:*

$$T(\mathtt{A}) = \begin{cases} c & \text{if } T \equiv \mathtt{A}\langle c\rangle \mid T' \\ \bot & \text{otherwise} \end{cases} \qquad T[\mathtt{AB}] = \begin{cases} \rho & \text{if } T \equiv (\mathtt{AB}):\rho \mid T' \\ \bot & \text{otherwise} \end{cases}$$

Example 3.3. *From the example in Section 2, consider the instantiated contracts of the store* $\mathtt{A}$ *and its customer* $\mathtt{B}_{12}$. *We illustrate the system, and how it progresses (hereafter, for the sake of readability, we often omit empty queues).*

$$\begin{array}{rcl}
T_{\mathtt{AB}_{12}Q} & = & \mathtt{A}\langle c_{\mathtt{A}}\,\{{}^{\mathtt{A}}/_{\mathtt{a}''}\}\,\{{}^{\mathtt{B}_{12}}/_{\mathtt{b}_1,\mathtt{b}_2}\}\rangle \mid \mathtt{B}_{12}\langle c_{\mathtt{B}_{12}}\,\{{}^{\mathtt{A}}/_{\mathtt{a}''}\}\,\{{}^{\mathtt{B}_{12}}/_{\mathtt{b}_1,\mathtt{b}_2}\}\rangle \mid (\mathtt{B}_{12}\mathtt{A}):[] \\
 & = & \mathtt{A}\langle \mathtt{B}_{12}?\mathtt{req};\mathtt{B}_{12}?\mathtt{req};\ldots\rangle \mid \mathtt{B}_{12}\langle \mathtt{A}!\mathtt{req};\mathtt{A}!\mathtt{req};\ldots\rangle \mid (\mathtt{B}_{12}\mathtt{A}):[] \\
 & \xrightarrow{\mathtt{B}_{12}\to\mathtt{A}:\mathtt{req}}\!\!\!\!\!\rightarrow & \mathtt{A}\langle \mathtt{B}_{12}?\mathtt{req};\mathtt{B}_{12}?\mathtt{req};\ldots\rangle \mid \mathtt{B}_{12}\langle \mathtt{A}!\mathtt{req};\ldots\rangle \mid (\mathtt{B}_{12}\mathtt{A}):\mathtt{req} \\
 & \xrightarrow{\mathtt{A}\leftarrow\mathtt{B}_{12}:\mathtt{req}}\!\!\!\!\!\rightarrow & \mathtt{A}\langle \mathtt{B}_{12}?\mathtt{req};\ldots\rangle \mid \mathtt{B}_{12}\langle \mathtt{A}!\mathtt{req};\ldots\rangle \mid (\mathtt{B}_{12}\mathtt{A}):[]
\end{array}$$

3.3. Choreography Synthesis as Agreement

We introduce in Definition 3.6 below the agreement relation, that tells whether some contracts can be combined to describe a correct interaction. Roughly, a set of contracts admits an agreement if it can be assigned a choreography. To do that, we borrow some results from [6, 16]: in particular, we exploit a typing system which assigns a (unique) choreography to a set of contracts.

Definition 3.4. *The syntax of choreographies* $\mathcal{G}$ *is defined as follows:*

$$\mathcal{G} \quad ::= \quad \mathtt{A}\to\mathtt{B}:\mathtt{e};\mathcal{G} \;\mid\; \mathcal{G}+\mathcal{G}' \;\mid\; \mathcal{G} \mid \mathcal{G}' \;\mid\; \mu\,\mathbf{x}.\mathcal{G} \;\mid\; \mathbf{x} \;\mid\; \mathbf{0}$$

The first production means that $\mathtt{A}$ sends a message of sort $\mathtt{e}$ to $\mathtt{B}$, then interactions in $\mathcal{G}$ take place; $\mathcal{G}+\mathcal{G}'$ means that either interactions in $\mathcal{G}$ or in $\mathcal{G}'$ take place; $\mathcal{G} \mid \mathcal{G}'$ means that interactions in $\mathcal{G}$ and $\mathcal{G}'$ are executed concurrently; the rest of the productions are for recursive interactions, and termination ($\mathbf{0}$).

To synthesise a choreography $\mathcal{G}$ from a system of contracts T, we analyse what actions can occur at each possible state of T. These actions are contained in the set $\mathrm{TRS}(T)$ defined below. The auxiliary predicate $T\updownarrow$ holds whenever two participants are able to synchronise (i.e., $\mathtt{B}$ may receive the message sent by $\mathtt{A}$ as soon as it appears on the queue).

Definition 3.5 (System of contracts Ready Set). *We define the* ready set *of a system as follows:*

$$\mathrm{TRS}(T) = \begin{cases} \{\mathtt{A}\leftarrow\mathtt{B}\} \cup \mathrm{TRS}(T') & \text{if } T \equiv \mathtt{A}\langle \textstyle\sum_{i\in I} \mathtt{B}?\mathtt{e}_i;c_i\rangle \mid T' \\ \{\mathtt{A}\rightarrow\mathtt{B}_i \mid i \in I\} \cup \mathrm{TRS}(T') & \text{if } T \equiv \mathtt{A}\langle \bigoplus_{i\in I} \mathtt{B}_i!\mathtt{e}_i;c_i\rangle \mid T' \\ \varnothing & \text{if } T \equiv \mathbf{0} \end{cases}$$

We write $T\updownarrow$ when $\exists\, \mathtt{A},\mathtt{B} : \mathtt{A}\rightarrow\mathtt{B} \in \mathrm{TRS}(T) \wedge \mathtt{B}\leftarrow\mathtt{A} \in \mathrm{TRS}(T)$. We write $T\not\updownarrow$ if $T\updownarrow$ does not hold.

In Figure 2 we present the synthesis of choreographies from systems of contracts (this is a slight adaptation of the synthesis in [16]). Typing judgements have the form $\Gamma \vdash T \blacktriangleright \mathcal{G}$, where Γ is an environment which keeps track of recursion variables. Roughly, Γ maps pairs $(\mathtt{A},\mathbf{x})$ to $\mathbf{x}'$, where $\mathbf{x}$ is the local recursion variable of participant $\mathtt{A}$ and $\mathbf{x}'$ is a global recursion variable. A typing judgement $\Gamma \vdash T \blacktriangleright \mathcal{G}$ says that, under hypothesis Γ, the system T is assigned to global type $\mathcal{G}$.

We write $\mathcal{P}(\mathcal{G})$ (resp. $\mathcal{C}(\mathcal{G})$) for the set of participant (resp. queue) names in $\mathcal{G}$.

Definition 3.6 (Agreement). *We say that a system of contracts T* admits an agreement *when $\vdash T \blacktriangleright \mathcal{G}$ can be inferred from the rules in Figure 2.*

Essentially, the synthesis rules allow to execute a set of contracts step-by-step, while keeping track of the structure of the interactions in a global type. The rules are driven by the ready set of T and the structure of its processes. We briefly describe them below.

Sequential interactions are dealt with by rule [;]. The rule validates prefixes provided that the continuation is typable and that no other interactions are possible in T. For instance, rule [;] does not apply to

$$\mathtt{A}_1\langle \mathtt{B}_1!\mathtt{e};c_1\rangle \mid \mathtt{B}_1\langle \mathtt{A}_1?\mathtt{e};c_1'\rangle \mid \mathtt{A}_2\langle \mathtt{B}_2!\mathtt{e}_1;c_2\rangle \mid \mathtt{B}_2\langle \mathtt{A}_2?\mathtt{e}_1;c_2'\rangle \qquad \text{✘}$$

because there is no ordering relation between the actions of $\mathtt{A}_1$ and $\mathtt{B}_1$ on one hand, and the actions of $\mathtt{A}_2$ and $\mathtt{B}_2$ on the other hand. In such cases, the rule [|] must be used. In fact, the premise $T\not\updownarrow$ guarantees that a synthesised global type is unique (up-to structural congruence), cf. [16, Theorem 3.3].

Concurrent branches are introduced by rule [|]. The rule validates concurrent branches when they can be validated using a partition of the system being considered (recall that $\mathcal{P}(T)\cap\mathcal{P}(T') = \varnothing$ by assumption).

Choice in a global type is dealt with by rules [⊕] and [+]. Rule [⊕] introduces the global type choice operator, it requires both branches to be typable and that no other interactions are possible in T — for the same reason as in rule [;]. Rule [+] discharges a branch of an external choice. This allows, e.g., to assign a type to systems of the form:

$$\mathtt{A}\langle \mathtt{B}!\mathtt{e} \oplus \mathtt{B}!\mathtt{e}_1\rangle \mid \mathtt{B}\langle \mathtt{A}?\mathtt{e} + \mathtt{A}?\mathtt{e}_1 + \mathtt{A}?\mathtt{e}_2\rangle \qquad \text{✔}$$

Recursion is handled by rules [μ] and [$\mathbf{x}$]. The former rule "guesses" the participants involved in a recursive behaviour. If two of them interact, [μ] validates the recursion provided that the system can be typed when such participants are associated to the global recursion variable $\mathbf{x}$ – assuming $\mathbf{x}$ is fresh. Rule [$\mathbf{x}$] checks that all the participants in the recursion have reached a local recursion variable corresponding to the global recursion. The termination $\mathbf{0}$ is introduced by rule [$\mathbf{0}$], which only applies when all the participants in T are terminated. Rule [≡] validates a system up-to structural congruence, so allowing recursive behaviour to be unfolded.

$$[;]\ \frac{\Gamma \vdash \mathtt{A}\langle c\rangle \mid \mathtt{B}\langle c'\rangle \mid T \blacktriangleright \mathcal{G} \qquad T \not\updownarrow}{\Gamma \vdash \mathtt{A}\langle \mathtt{B}!\mathtt{e};c\rangle \mid \mathtt{B}\langle \mathtt{A}?\mathtt{e};c'\rangle \mid T \blacktriangleright \mathtt{A}\to\mathtt{B}:\mathtt{e};\mathcal{G}} \qquad [\,|\,]\ \frac{\vdash T \blacktriangleright \mathcal{G} \qquad \vdash T' \blacktriangleright \mathcal{G}'}{\Gamma \vdash T \mid T' \blacktriangleright \mathcal{G} \mid \mathcal{G}'}$$

$$[\oplus]\ \frac{\Gamma \vdash \mathtt{A}\langle c\rangle \mid T \blacktriangleright \mathcal{G} \qquad \Gamma \vdash \mathtt{A}\langle c'\rangle \mid T \blacktriangleright \mathcal{G}' \qquad T \not\updownarrow}{\Gamma \vdash \mathtt{A}\langle c \oplus c'\rangle \mid T \blacktriangleright \mathcal{G} + \mathcal{G}'} \qquad [+]\ \frac{\Gamma \vdash \mathtt{B}\langle c\rangle \mid T \blacktriangleright \mathcal{G}}{\Gamma \vdash \mathtt{B}\langle c + c'\rangle \mid T \blacktriangleright \mathcal{G}}$$

$$[\mu]\ \frac{\exists\, 1 \le i,j \le k\colon (\mathtt{A}_i\langle c_i\rangle \mid \mathtt{A}_j\langle c_j\rangle)\updownarrow \qquad \Gamma \cdot (\mathtt{A}_1,\mathbf{x}_1) : \mathbf{x},\ldots,(\mathtt{A}_k,\mathbf{x}_k) : \mathbf{x} \vdash \mathtt{A}_1\langle c_1\rangle \mid \cdots \mid \mathtt{A}_k\langle c_k\rangle \blacktriangleright \mathcal{G}}{\Gamma \vdash \mathtt{A}_1\langle \mu\,\mathbf{x}_1.c_1\rangle \mid \cdots \mid \mathtt{A}_k\langle \mu\,\mathbf{x}_k.c_k\rangle \blacktriangleright \mu\,\mathbf{x}.\mathcal{G}}$$

$$[\mathbf{x}]\ \frac{\forall 1 \le i \le k\colon \Gamma(\mathtt{A}_i,\mathbf{x}_i) = \mathbf{x}}{\Gamma \vdash \mathtt{A}_1\langle \mathbf{x}_1\rangle \mid \cdots \mid \mathtt{A}_k\langle \mathbf{x}_k\rangle \blacktriangleright \mathbf{x}} \qquad [\equiv]\ \frac{T \equiv T' \qquad \Gamma \vdash T' \blacktriangleright \mathcal{G}}{\Gamma \vdash T \blacktriangleright \mathcal{G}} \qquad [\mathbf{0}]\ \frac{\forall \mathtt{A} \in \mathcal{P}(T)\colon T(\mathtt{A}) = \mathbf{0}}{\Gamma \vdash T \blacktriangleright \mathbf{0}}$$

Figure 2: Inference of global types.

The main properties that we are interested in — and that are guaranteed by the synthesis — are (*i*) that the inferred global type is projectable back to the original contracts and (*ii*) that if a system of contract T is typable, then T is "safe", cf. Theorem 3.9 below. The synthesised global types also satisfy other well-formedness properties which we discuss briefly in Section 8.2.

Example 3.7. *Recall the buyer-seller scenario from Section 2. By combining the contract of store* $\mathtt{A}$ *with those of customers* $\mathtt{B}_1$ *and* $\mathtt{B}_2$*, we obtain the system:*

$$T_{\mathtt{AB}_1\mathtt{B}_2} \;=\; \mathtt{A}\langle c_{\mathtt{A}}\,\{{}^{\mathtt{B}_1}\!/_{\mathtt{b}_1}\}\,\{{}^{\mathtt{B}_2}\!/_{\mathtt{b}_2}\}\rangle \;\mid\; \mathtt{B}_1\langle c_{\mathtt{B}_1}\,\{{}^{\mathtt{A}}\!/_{\mathtt{a}}\}\,\{{}^{\mathtt{B}_2}\!/_{\mathtt{b}'_2}\}\rangle \;\mid\; \mathtt{B}_2\langle c_{\mathtt{B}_2}\,\{{}^{\mathtt{A}}\!/_{\mathtt{a}'}\}\,\{{}^{\mathtt{B}_1}\!/_{\mathtt{b}'_1}\}\rangle$$

The system $T_{\mathtt{AB}_1\mathtt{B}_2}$ *admits an agreement: indeed, we have that* $\vdash T_{\mathtt{AB}_1\mathtt{B}_2} \blacktriangleright \mathcal{G}_{\mathtt{AB}_1\mathtt{B}_2}$ *(where the choreography* $\mathcal{G}_{\mathtt{AB}_1\mathtt{B}_2}$ *is that defined at page 4). Similarly, if we combine the store* $\mathtt{A}$ *with* $\mathtt{B}_{12}$*, we have that* $T_{\mathtt{AB}_{12}}$ *admits an agreement, since* $\vdash T_{\mathtt{AB}_{12}} \blacktriangleright \mathcal{G}_{\mathtt{AB}_{12}}$ *(where* $\mathcal{G}_{\mathtt{AB}_{12}}$ *is the choreography at page 4).*

Theorem 3.9 below establishes some properties of systems of contracts which do admit an agreement. We first define some *undesirable* properties of systems. A system is a *deadlock* if all its queues are empty, there is one participant expecting a message, and all participants are either terminated or waiting for an input. An *unspecified reception configuration* is a system for which there is a participant $\mathtt{B}$ who is permanently unable to read any messages from its queues, i.e., there are *unexpected* messages in $\mathtt{B}$'s buffers. A system has a *self-interaction* if some participant is expected to send a message to himself (or to receive a message from himself). Definition 3.8 formalises these concepts.

Definition 3.8 (Deadlock/unspecified reception). *For a system of contracts* T*, we say that:*

- T *is a deadlock if the following conditions hold:*

$$\begin{aligned}
&\forall\, \mathtt{AB} \in \mathcal{C}(T)\colon T[\mathtt{AB}] = []\\
&\exists\, \mathtt{C} \in \mathcal{P}(T)\colon T(\mathtt{C}) \equiv \mathtt{A}?\mathtt{e};c + c'\\
&\forall\, \mathtt{C} \in \mathcal{P}(T)\colon T(\mathtt{C}) \equiv \mathtt{A}?\mathtt{e};c + c' \;\vee\; T(\mathtt{C}) \equiv \mathbf{0}
\end{aligned}$$

- T *is an unspecified reception configuration if* $\exists\, \mathtt{B} \in \mathcal{P}(T)$:

$$T(\mathtt{B}) \equiv \mathtt{A}'?\mathtt{e}';c_1 + c_2 \;\wedge\; (\forall \mathtt{e},\mathtt{A} \in \mathcal{P}(T)\colon T(\mathtt{B}) \equiv \mathtt{A}?\mathtt{e};c + c' \implies T[\mathtt{AB}] \neq [] \wedge T[\mathtt{AB}] \neq \mathtt{e}\cdot\rho)$$

- T *has a self-interaction if there exists* $\mathtt{A} \in \mathcal{P}(T)$ *such that* $T'(\mathtt{A}) \equiv \mathtt{A}!\mathtt{e};c \oplus c'$ *or* $T'(\mathtt{A}) \equiv \mathtt{A}?\mathtt{e};c + c'$*.*

Theorem 3.9 (Agreement). *If T admits an agreement, then for all T' such that $T \mid \mathrm{Q}(T) \twoheadrightarrow^* T'$:*

1. *T' is not a deadlock.*
2. *T' is not an unspecified reception configuration.*
3. *T' has no self-interactions.*
4. *Either there is T'' such that $T' \twoheadrightarrow T''$, or*

$$\forall\, \mathtt{AB} \in \mathcal{C}(T') \colon T'[\mathtt{AB}] = [] \quad \textit{and} \quad \forall\, \mathtt{C} \in \mathcal{P}(T') \colon T'(\mathtt{C}) \equiv \mathbf{0}$$

Proof. Items (1),(2) and (4) follow from Theorem 3.1 in [16]. Item (3) is a direct consequence of Theorem 3.9 and of the assumption that for all $\mathtt{AB} \in \mathcal{C}(T) \colon \mathtt{A} \neq \mathtt{B}$. □

3.4. Culpability

We introduce a notion of culpability in systems of contracts. Intuitively, a participant is culpable in a system if it is able to send or receive some message. This condition is not necessarily permanent: a culpable participant may exculpate itself by sending/receiving the required messages.

Definition 3.10 (Contract culpability). $\mathtt{A}$ *is* culpable in T *when* $T \xrightarrow{\mathtt{A} \leftrightharpoons \mathtt{B}:\mathtt{e}} \twoheadrightarrow$, *for some* $\mathtt{B}$ *and* $\mathtt{e}$.

As expected, a participant $\mathtt{A}$ is not culpable in a system of contracts not including any contracts of $\mathtt{A}$.

Lemma 3.11. *For all system T, if $\mathtt{A} \notin \mathcal{P}(T)$, then $\mathtt{A}$ is not culpable in T.*

The following theorem ensures that, if a system of contracts admits an agreement, then at each step of its execution (until reaching the final state $\mathbf{0}$) some participant is culpable. This guarantees that, unless some participants want to stay culpable forever, the system of contracts enjoys progress.

Theorem 3.12 (Presence of culpability in contracts). *If T admits an agreement and $T \mid \mathrm{Q}(T) \twoheadrightarrow^* T' \not\equiv \mathbf{0}$, then there exists at least one culpable participant in T'.*

Proof. Since $T' \not\equiv \mathbf{0}$ then, for some $\mathtt{A}, \mathtt{B}, \mathtt{e}, c, c'$, either (a) $T' \equiv \mathtt{A}\langle \mathtt{B}!\mathtt{e}; c \oplus c' \rangle \mid T''$, or (b) $T' \equiv \mathtt{A}\langle \mathtt{B}?\mathtt{e}; c + c' \rangle$. Case (a) follows directly, since a send action can always be fired. Case (b) follows from Theorem 3.9, i.e., there must be a message for $\mathtt{A}$ in a queue $\mathtt{BA}$, otherwise we would have a deadlock or an unspecified reception configuration. □

4. Introducing Choreographies to the Wild

We present a variant of the CO_2 calculus (for COntract-Oriented computing) [14] which we adapt to deal with the multiparty contracts and sessions introduced in Section 3. In Sections 4.1 and 4.2 we give its syntax and semantics, while in Section 4.3 we explain in detail how new sessions are established, providing several examples. In Section 4.4 we formalise when participants in CO_2 systems are "culpable", and show that a non-terminated session always implies the (possibly transient) culpability of some participant (Theorem 4.10). Finally, in Section 4.5 we discuss how the session creation mechanics can be adjusted in order to enforce some desired properties on the resulting interactions.

commutative monoidal laws for $+$ and $|$, with $\mathbf{0}$ as identity element

$$\mathtt{A}[(u)P] \equiv (u)\mathtt{A}[P] \qquad (u)Z \equiv Z \ \text{ if } u \notin \mathrm{fnv}(Z)$$

$$(u)(v)Z \equiv (v)(u)Z \qquad Z \mid (u)Z' \equiv (u)(Z \mid Z') \ \text{ if } u \notin \mathrm{fnv}(Z)$$

$$\mathtt{A}[K] \mid \mathtt{A}[K'] \equiv \mathtt{A}[K \mid K']$$

Figure 3: Structural congruence rules for CO_2.

4.1. Syntax

Let $\mathbb{S}$ and $\mathcal{S}$ be disjoint sets of, respectively, *session names* (ranged over by $s, s', t, \ldots$) and *session variables* (ranged over by $x, y, z \ldots$). Let $u, v, \ldots$ range over $\mathbb{S} \cup \mathcal{S}$. The syntax of CO_2 is given as follows:

$$\begin{array}{lrcl}
\text{(Processes)} & P, Q & ::= & \textstyle\sum_{i\in I} p_i \,.\, P_i \;\mid\; P \mid Q \;\mid\; (\vec{u})P \;\mid\; (\vec{\mathtt{a}})P \;\mid\; X(\vec{u}, \vec{\underline{\mathtt{a}}}) \\
\text{(Prefixes)} & p & ::= & \tau \;\mid\; \mathsf{tell}_{\underline{\mathtt{a}}}\!\downarrow_u c \;\mid\; \mathsf{fuse} \;\mid\; \mathsf{do}^{u}_{\underline{\mathtt{a}}}\,\mathtt{e} \\
\text{(Latent contracts)} & K & ::= & \downarrow_u \mathtt{A}\ \mathit{says}\ c \;\mid\; K \mid K \\
\text{(Systems)} & S & ::= & \mathtt{A}[P] \;\mid\; \mathtt{A}[K] \;\mid\; s[T] \;\mid\; S \mid S \;\mid\; (\vec{u}, \vec{\mathtt{a}})S \;\mid\; \mathbf{0}
\end{array}$$

CO_2 features CCS-style processes, equipped with branching $\sum$ (not to be confused with the choice operator used in contracts), parallel composition $|$, restrictions of session and participant variables, and named process invocation. As usual, we denote the empty sum with $\mathbf{0}$, and omit its trailing occurrences. We assume that each process identifier X has a unique defining equation $X(y_1, \ldots, y_n, \mathtt{a}_1, \ldots, \mathtt{a}_m) := P$ such that $\mathrm{fv}(P) \subseteq \{y_1, \ldots, y_n, \mathtt{a}_1, \ldots, \mathtt{a}_m\}$ and each occurrence of process identifiers in P is prefix-guarded. We shall take the liberty of omitting the arguments of $X(\vec{y}, \vec{\mathtt{a}})$ when they are clear from the context. The prefixes are for internal action (τ), contract advertisement (tell), session creation upon contractual agreement (fuse), and execution of contractual actions (do).

A latent contract of the form $\downarrow_x \mathtt{A}\ \mathit{says}\ c$ represents the promise of participant $\mathtt{A}$ to fulfil c by executing do-actions, once x has been instantiated with a session name.

CO_2 systems may be parallel compositions of *participants* $\mathtt{A}[P]$ (where P is the process being executed by $\mathtt{A}$), collections of *latent contracts* $\mathtt{A}[K]$ (where $\mathtt{A}$ is the participant to which the contracts in K have been advertised), and *sessions* $s[T]$ (where s is a session name, and T is a system of contracts as in Section 3). We assume *well-formed* systems where each participant name $\mathtt{A}$ can appear with at most one process — i.e., we rule out systems like $\mathtt{A}[P] \mid \mathtt{A}[Q]$.

Note that CO_2 process and system productions allow to delimit both session names/variables ($\vec{u}$) and participant variables ($\vec{\mathtt{a}}$), but *not* participant names: this is because participant names are used to uniquely (and statically) identify participants in a network, and therefore we do not want these names to be α-converted.

4.2. Semantics

The semantics of CO_2 (Figure 4) is defined up-to a congruence relation between CO_2 systems (Figure 3). The rules defining $\equiv$ are mostly standard. In Figure 3, Z, Z', Z'' range over processes, systems, or latent contracts; all rules involving delimited session names/variables u, v also hold for delimited participant variables $\mathtt{a}, \mathtt{b}$ (e.g., $\mathtt{A}[(\mathtt{a})P] \equiv (\mathtt{a})\mathtt{A}[P]$). Rule $\mathtt{A}[(u)P] \equiv (u)\mathtt{A}[P]$ allows to move delimitations between CO_2 systems and processes; the last rule ($\mathtt{A}[K] \mid \mathtt{A}[K'] \equiv \mathtt{A}[K \mid K']$) will be discussed in Section 4.3 and in Example 4.4.

Definition 4.1 (Semantics of CO_2). *$\xrightarrow{\ell}$ is the smallest relation between CO_2 systems closed under the rules of Figure 4, which are to be considered up-to the structural congruence relation $\equiv$ in Figure 3. The labels ℓ have the form $\mathtt{A} : p$, where $\mathtt{A}$ is the name of the participant firing prefix p.*

We briefly discuss the rules in Figure 4. [CO_2-Tau] allows a participant $\mathtt{A}$ to fire an internal action. [CO_2-Tell] allows $\mathtt{A}$ to advertise a contract c to $\mathtt{B}$; as a result, a new *latent* contract is created, recording the fact that

$$[\text{CO}_2\text{-Tau}] \quad \mathtt{A}[\tau \,.\, P + P' \mid Q] \xrightarrow{\mathtt{A}:\,\tau} \mathtt{A}[P \mid Q]$$

$$[\text{CO}_2\text{-Tell}] \quad \mathtt{A}[\mathsf{tell}_{\mathtt{B}}\downarrow_x c \,.\, P + P' \mid Q] \xrightarrow{\mathtt{A}:\,\mathsf{tell}_{\mathtt{B}}\downarrow_x c} \mathtt{A}[P \mid Q] \mid \mathtt{B}[\downarrow_x \mathtt{A}\ \mathit{says}\ c]$$

$$[\text{CO}_2\text{-Fuse}] \quad \frac{K \rhd^{\sigma}_{\pi} T \qquad \vec{\mathtt{a}} = \mathrm{dom}(\pi) \qquad \vec{u} = \mathrm{dom}(\sigma) \qquad \mathrm{img}(\sigma) = \{s\} \qquad s \text{ fresh}}{(\vec{u})(\vec{\mathtt{a}})\big(\mathtt{A}\big[\mathsf{fuse} \,.\, P + P' \mid Q\big] \mid \mathtt{A}[K] \mid S\big) \xrightarrow{\mathtt{A}:\,\mathsf{fuse}} (s)(\mathtt{A}[P \mid Q]\,\sigma\pi \mid s[T \mid \mathrm{Q}(T)] \mid S\sigma\pi)}$$

$$[\text{CO}_2\text{-Do}] \quad \frac{T \xrightarrow{\mathtt{A} \rightleftharpoons \mathtt{B}:\mathsf{e}} T'}{s[T] \mid \mathtt{A}\big[\mathsf{do}^{s}_{\mathtt{B}}\,\mathsf{e} \,.\, P + P' \mid Q\big] \xrightarrow{\mathtt{A}:\,\mathsf{do}^{s}_{\mathtt{B}}\,\mathsf{e}} s\big[T'\big] \mid \mathtt{A}[P \mid Q]}$$

$$[\text{CO}_2\text{-Def}] \quad \frac{X(\vec{y}, \vec{\mathtt{a}}) := P \qquad (\vec{z})(\vec{c})\,\big(\mathtt{A}\big[P\,\{\vec{v}/\vec{y}\}\,\{\vec{\mathtt{b}}/\vec{\mathtt{a}}\} \mid Q\big] \mid S\big) \xrightarrow{\ell} S'}{(\vec{z})(\vec{c})\,\big(\mathtt{A}\big[X(\vec{v}, \vec{\mathtt{b}}) \mid Q\big] \mid S\big) \xrightarrow{\ell} S'}$$

$$[\text{CO}_2\text{-Par}] \quad \frac{S \xrightarrow{\ell} S'}{S \mid S'' \xrightarrow{\ell} S' \mid S''}$$

$$[\text{CO}_2\text{-Del}] \quad \frac{S \xrightarrow{\mathtt{A}:\,p} S'}{(\vec{u})(\vec{\mathtt{a}})S \xrightarrow{\mathtt{A}:\,\mathrm{del}(p)} (\vec{u})(\vec{\mathtt{a}})S'} \qquad \text{where } \mathrm{del}(p) = \begin{cases} p & \text{if } \mathrm{fnv}(p) \cap (\vec{u} \cup \vec{\mathtt{a}}) = \varnothing \\ \tau & \text{otherwise} \end{cases}$$

Figure 4: Reduction semantics of CO_2.

it was promised by A. [CO_2-Fuse] establishes a new session, when an agreement is found among the latent contracts held in $\mathtt{A}[K]$. Such an agreement is reached only if the relation $K \rhd^{\sigma}_{\pi} T$ holds (cf. Definition 4.3). Intuitively, the relation holds if the contracts in K can be synthesised into a well-formed choreography. When an agreement exists, a fresh session name s and the names of the involved participants are shared (via substitutions σ and π, respectively); the initial state of the new session consists in the system of contracts T (constructed from K) with the communication queues connecting its participants. Rule [CO_2-Do] allows A to perform an input/output action e towards B on session s, provided that T (i.e., the current state of the contracts at s) allows for it. The other CO_2 rules are standard. In rule [CO_2-Del], the prefix p fired in the premise becomes τ in the conclusions, when p contains a delimited name/variable. For instance, consider the following reduction: $(x)\mathtt{A}[\mathsf{tell}_{\mathtt{B}}\downarrow_x c \,.\, P] \xrightarrow{\mathtt{A}:\,\tau} (x)\,(\mathtt{A}[P] \mid \mathtt{B}[\downarrow_x \mathtt{A}\ \mathit{says}\ c])$; one would not use the label $\mathtt{A} : \mathsf{tell}_{\mathtt{B}}\downarrow_x c$ instead of $\mathtt{A} : \tau$, since the variable x is not visible outside the process of A.

Example 4.2. *Consider the CO_2 system:*

$$S \;=\; \mathtt{A}[\mathsf{do}^{s}_{\mathtt{B}}\,\mathsf{int} + \mathsf{do}^{s}_{\mathtt{B}}\,\mathsf{bool}] \quad \mid \quad s[\mathtt{A}\langle \mathtt{B}!\mathsf{int}\rangle \mid \mathtt{B}\langle \mathtt{A}?\mathsf{int}\rangle \mid (\mathtt{AB}) : []] \quad \mid \quad \mathtt{B}[\mathsf{do}^{s}_{\mathtt{A}}\,\mathsf{int}]$$

Participant A *is willing to perform an action towards* B *on session s, with either a message of sort* int *or* bool*. However,* A*'s contract in s only specifies that* A *should send a message of sort* int *to* B*: therefore, according to rule* [CO_2-Do]*, only the first branch of* A *may be chosen, and the system reduces as follows.*

$$\begin{array}{lcccccc} S & \xrightarrow{\mathtt{A}:\,\mathsf{do}^{s}_{\mathtt{B}}\,\mathsf{int}} & \mathtt{A}[\mathbf{0}] & \mid & s[\mathtt{A}\langle \mathbf{0}\rangle \mid \mathtt{B}\langle \mathtt{A}?\mathsf{int}\rangle \mid (\mathtt{AB}) : \mathsf{int}] & \mid & \mathtt{B}[\mathsf{do}^{s}_{\mathtt{A}}\,\mathsf{int}] \\ & \xrightarrow{\mathtt{B}:\,\mathsf{do}^{s}_{\mathtt{A}}\,\mathsf{int}} & \mathtt{A}[\mathbf{0}] & \mid & s[\mathtt{A}\langle \mathbf{0}\rangle \mid \mathtt{B}\langle \mathbf{0}\rangle \mid (\mathtt{AB}) : []] & \mid & \mathtt{B}[\mathbf{0}] \end{array}$$

4.3. Reaching Agreements

We now discuss the mechanics underlying the [CO_2-Fuse] rule, with a focus on the agreement relation which regulates session establishment.

Definition 4.3 (Agreement relation $K \rhd_\pi^\sigma T$). *Let* $K \equiv \big|_{i\in I} \downarrow_{x_i} \mathtt{A}_i$ *says* c_i*, and let* $\pi\colon \mathcal{P} \to \mathbb{P}$ *and* $\sigma\colon \mathcal{S} \to \mathbb{S}$ *be two substitutions mapping participant variables to participant names, and session variables to session names, respectively. Also, let* $T \equiv \big|_{i\in I} \mathtt{A}_i\langle c_i\rangle\pi$*. Then,* $K \rhd_\pi^\sigma T$ *holds iff:*

1. $\mathrm{dom}(\sigma) = \bigcup_{i\in I} \{x_i\}$;
2. $\mathrm{dom}(\pi) = \bigcup_{i\in I} \mathrm{fv}(c_i)$;
3. T *admits an agreement — i.e.,* $\exists \mathcal{G}\colon \vdash T \blacktriangleright \mathcal{G}$.

Conditions 1 and 2, on σ and π, guarantee that all the session and participant variables are indeed instantiated. Condition 3 holds whenever a judgment $\vdash T \blacktriangleright \mathcal{G}$ can be inferred, for some $\mathcal{G}$ (see Definition 3.6).

We discuss two safety issues that Definition 4.3 rules out:

1. our assumptions on a system T imply that each participant may have at most one contract in K, cf. Section 3.2. This restriction allows us to avoid problematic scenarios arising during session establishment. For instance, consider a process $\mathtt{A}[P]$ advertising two contracts c and c' using session variables x and y, respectively. Were we not preventing both contracts to appear in a same session s, c and c' could be fused together in s. Therefore, by rule [CO₂-Fuse], both x and y would be replaced by s. Hence, each prefixes of the form do^x and do^y in P would be turned into do^s, possibly mixing the actions originally intended for one contract with those intended for the other. In such situations, one cannot enforce safety properties such as deadlock freedom, since contracts c and c' are independent in T, while there might be causal dependencies in P;
2. by item (3) of Theorem 3.9, it cannot be the case that, within a contract c_i belonging to $\mathtt{A}_i$, a free participant variable is substituted by $\mathtt{A}_i$ itself.

When searching for possible agreements, the congruence rule $\mathtt{A}[K] \mid \mathtt{A}[K'] \equiv \mathtt{A}[K \mid K']$ in Figure 3 allows for rearranging collections of latent contracts, in order to select the compliant ones that satisfy the premises of [CO₂-Fuse] rule, cf. Example 4.4 below.

Example 4.4 (Agreements and structural congruence). *Consider the system:*

$$S \;=\; (x,y,z)\big(\mathtt{B}[\mathsf{fuse}\,.\,P \mid Q] \;\mid\; \mathtt{B}[K] \;\mid\; \mathtt{A}_1[\ldots] \;\mid\; \mathtt{A}_2[\ldots] \;\mid\; \mathtt{A}_3[\ldots]\big)$$

where $K = \downarrow_x \mathtt{A}_1$ *says* $\mathtt{A}_2!\mathsf{int} \mid \downarrow_y \mathtt{A}_2$ *says* $\mathtt{A}_1?\mathsf{int} \mid \downarrow_z \mathtt{A}_3$ *says* $\mathtt{B}?\mathsf{bool}$.

The fuse *prefix of participant* $\mathtt{B}$ *cannot be immediately fired: no contract matches* $\mathtt{A}_3$*'s, and thus the three latent contracts cannot be assigned a global type. Therefore, there is no* T, σ, π *such that* $K \rhd_\pi^\sigma T$*, as required by rule* [CO₂-Fuse]*. However, we can rearrange the latent contracts via structural congruence:*

$$S \;\equiv\; S' \;=\; (z)(x,y)\big(\mathtt{B}[\mathsf{fuse}\,.\,P \mid Q] \;\mid\; \mathtt{B}[K'] \;\mid\; \mathtt{B}[\downarrow_z \mathtt{A}_3 \text{ says } \mathtt{B}?\mathsf{bool}] \;\mid\; \mathtt{A}_1[\ldots] \;\mid\; \mathtt{A}_2[\ldots] \;\mid\; \mathtt{A}_3[\ldots]\big)$$

where $K' = \downarrow_x \mathtt{A}_1$ *says* $\mathtt{A}_2!\mathsf{int} \mid \downarrow_y \mathtt{A}_2$ *says* $\mathtt{A}_1?\mathsf{int}$.

If we take $\sigma = \{{}^{s}/_{x,y}\}$ *(with* s *fresh) and* $T = \mathtt{A}_1\langle \mathtt{A}_2!\mathsf{int}\rangle \mid \mathtt{A}_2\langle \mathtt{A}_1?\mathsf{int}\rangle$*, we now have* $K' \rhd_\varnothing^\sigma T$*. Indeed,* T *is composed by* $\mathtt{A}_1$*'s and* $\mathtt{A}_2$*'s contracts, and* $\vdash T \blacktriangleright \mathtt{A}_1 \to \mathtt{A}_2 : \mathsf{int}$*; furthermore, the domains of* σ *and* $\pi = \varnothing$ *comply with the conditions in Definition 4.3.*

Since all the premises of [CO₂-Fuse] *are now satisfied, it is possible to create a new session* s*, involving* $\mathtt{A}_1$ *and* $\mathtt{A}_2$*, whose initial state is* $T \mid \mathrm{Q}(T)$*, and we obtain the system:*

$$S' \xrightarrow{\mathtt{B}:\,\mathsf{fuse}} (z)(s)\big(\mathtt{B}[P \mid Q]\,\sigma \;\mid\; s[T \mid \mathrm{Q}(T)] \;\mid\; \mathtt{B}[\downarrow_z \mathtt{A}_3 \text{ says } \mathtt{B}?\mathsf{bool}]\,\sigma \;\mid\; \mathtt{A}_1[\ldots]\,\sigma \;\mid\; \mathtt{A}_2[\ldots]\,\sigma \;\mid\; \mathtt{A}_3[\ldots]\,\sigma\big)$$

Note that $\mathtt{A}_3$*'s contract remains latent, and thus it may be* fused *later on.*

Example 4.5. *Going back to our running example, take* $c_{\mathtt{A}}$, $c_{\mathtt{B}_1}$, $c_{\mathtt{B}_2}$ *from Section 2, and* $T_{\mathtt{AB}_1\mathtt{B}_2}, \mathcal{G}_{\mathtt{AB}_1\mathtt{B}_2}$ *from Example 3.7. Consider the following CO₂ system, where* $P_{\mathtt{A}}, P_{\mathtt{B}_1}, P_{\mathtt{B}_2}$ *are given in Example 5.6,*

$$S = \mathtt{A}[(x)(\mathsf{b}_1,\mathsf{b}_2)\,\mathsf{tell}_{\mathtt{A}}\downarrow_x c_{\mathtt{A}}\,.\,\mathsf{fuse}\,.\,P_{\mathtt{A}}] \mid \mathtt{B}_1[(y)(\mathsf{a},\mathsf{b}_2')\,\mathsf{tell}_{\mathtt{A}}\downarrow_y c_{\mathtt{B}_1}\,.\,P_{\mathtt{B}_1}] \mid \mathtt{B}_2[(z)(\mathsf{a},\mathsf{b}_1')\,\mathsf{tell}_{\mathtt{A}}\downarrow_z c_{\mathtt{B}_2}\,.\,P_{\mathtt{B}_2}]$$
$$\rightarrow\rightarrow\rightarrow$$
$$(x,y,z)(\mathsf{a},\mathsf{a}',\mathsf{b}_1,\mathsf{b}_1',\mathsf{b}_2,\mathsf{b}_2')\big(\mathtt{A}[\mathsf{fuse}\,.\,P_{\mathtt{A}}] \mid \mathtt{B}_1[P_{\mathtt{B}_1}] \mid \mathtt{B}_2[P_{\mathtt{B}_2}]\mathtt{A}[\downarrow_x \mathtt{A}\ \mathit{says}\ c_{\mathtt{A}} \mid\downarrow_y \mathtt{B}_1\ \mathit{says}\ c_{\mathtt{B}_1} \mid\downarrow_z \mathtt{B}_2\ \mathit{says}\ c_{\mathtt{B}_2}]\big)$$
$$\xrightarrow{\mathtt{A}:\,\mathsf{fuse}}$$
$$(s)S' = (s)\big(\mathtt{A}[P_{\mathtt{A}}\sigma\pi] \mid s[T_{\mathtt{AB}_1\mathtt{B}_2} \mid \mathrm{Q}(T_{\mathtt{AB}_1\mathtt{B}_2})] \mid \mathtt{B}_1[P_{\mathtt{B}_1}\sigma\pi] \mid \mathtt{B}_2[P_{\mathtt{B}_2}\sigma\pi]\big)$$
where $\sigma = \{{}^{s}/_{x,y,z}\}$ *(*s *fresh) and* $\pi = \{{}^{\mathtt{A}}/_{\mathsf{a},\mathsf{a}'}, {}^{\mathtt{B}_1}/_{\mathsf{b}_1,\mathsf{b}_1'}, {}^{\mathtt{B}_2}/_{\mathsf{b}_2,\mathsf{b}_2'}\}$

System S *is the one already considered in Section 2, where all the participants are willing to advertise their respective contracts to the store* A*, by using a* tell *primitive. Firing those prefixes (first 3 steps of the computation above) has the effect of creating corresponding latent contracts, collected by* A*. At the* 4^{th} *step, these contracts are* fused*. Given* σ *and* π *as above, Definition 4.3 is indeed applicable: the domains of* σ *and* π *comply with the definition premises, and as shown in Example 3.7, a system* $T_{\mathtt{AB}_1\mathtt{B}_2}$ *consisting of contracts* $c_{\mathtt{A}}\pi$, $c_{\mathtt{B}_1}\pi$ *and* $c_{\mathtt{B}_2}\pi$ *may be assigned a global type* $\mathcal{G}_{\mathtt{AB}_1\mathtt{B}_2}$*. Hence, a new session* s *is created, based on* $T_{\mathtt{AB}_1\mathtt{B}_2}$*, plus the queues connecting all pairs of participants. The session variables of the latent contracts being* fused *(i.e.,* x *for participant* A*,* y *for* $\mathtt{B}_1$*, and* z *for* $\mathtt{B}_2$*) are replaced by the fresh session name* s *in the processes* $P_{\mathtt{A}}$*,* $P_{\mathtt{B}_1}$ *and* $P_{\mathtt{B}_2}$*, via* σ*. Similarly for participant variables which are replaced by participant names, via* π*.*

The next example illustrates a case where advertising a contract with *both* participant names and variables allows for enforcing specific properties on new sessions — in particular, making sure that a certain participant will be involved.

Example 4.6. *A seller* S *wants to sell an item to a* specific *buyer* B*, via* any *shipping company that provides a package tracking system. The contract of* S *is:*

$$\mathsf{B!price};\,(\mathsf{B?pay};\mathsf{c!request};\mathsf{c?tracking};\mathsf{B!tracking}\ +\ \mathsf{B?cancel};\mathsf{c!cancel})$$

saying that the seller S *promises to send a* price *to the buyer* B*. Then,* B *can either* pay*, or* cancel *the transaction: in the first case,* S *must send a shipping* request *to a (still unknown) shipping company* c*, who in turn must send back a* tracking *number, which is then forwarded to* B*; otherwise, if* B cancel*s, then* S *will also* cancel *the transaction with* c*. This contract admits an agreement only if the role of the buyer is played by* B*; the role of shipping company* c*, instead, may be played by any participant offering a compliant contract (such as* $\mathtt{A}?\mathsf{request};\mathtt{A}!\mathsf{tracking} + \mathtt{A}?\mathsf{cancel}$*).*

Example 4.7 shows how a participant can use participant variables to enforce some property along *different* sessions — in particular, to ensure the presence of some participant that was involved in a previous agreement.

Example 4.7. *A seller* S *wants to find a shipping company; then,* S *wants to use that very same shipping company for every new transaction with some buyer.*

The shipping company can be found with a "service discovery" contract such as $c_D = \mathsf{c!find\_ship}$*: when a session is established on such a contract,* c *will be replaced with the name of the participant playing the role of the shipper. For each selling transaction,* S *advertises a contract of the form* $c_S = \mathsf{b!price};\ldots\mathsf{c!address}$*, where the seller sends a* price *to a (still unknown) buyer* b*, and after some interaction (here omitted) it sends a shipping* address *to the shipping company* c*.*

The seller can be implemented with the following CO₂ process, where K *is a contract broker:*

$$P \;=\; (x)(\mathsf{c})\,\mathsf{tell}_{\mathtt{K}}\downarrow_x c_D\,.\,\mathsf{do}^x_{\mathsf{c}}\,\mathsf{find\_ship}\,.\,X(\mathsf{c})$$
$$\text{where } X(\mathsf{c}) := (y)(\mathsf{b})\,(\mathsf{tell}_{\mathtt{K}}\downarrow_y c_S \ldots X(\mathsf{c})) \qquad (c_S\text{-related actions omitted})$$

Note that c_S *may be advertised multiple times, using different session variables, inside the recursive process* X*.*

An extended version of this example is available in the appendix [17].

4.4. On Culpability

In this section we formalise the notion of culpability in CO_2 systems. Intuitively, a CO_2 participant is "culpable" when it is expected to send or receive some message in a session — and may exculpate itself by performing the required interaction.

We begin by noticing that, according to the CO_2 semantics, each do prefix exposed by a participant, say $\mathtt{A}[P]$, is driven by the contract that A promised to abide by. In a sense, CO_2 is "culpability-driven", according to Definition 4.8 below: when a participant is culpable, it has the duty of making the session progress according to its contract. The notion of culpability in CO_2 systems is an extension of the one on contracts (Definition 3.10), covering established sessions.

Definition 4.8 (Process culpability). *We say that* A is culpable at s in S *iff* $S \equiv s[T] \mid S'$ *and* A *is culpable in* T.
We say that A is culpable in S *iff, for some* s *and* u, $S \equiv (u)S_0$ *and* A *is culpable at* s *in* S_0.

According to Definition 4.8, when e.g. A is culpable at s in S, then necessarily $s \in \mathrm{fv}(S)$; furthermore, A is also culpable in $(s)S$ (i.e., culpability is preserved when s is closed and the corresponding session is hidden).

Example 4.9. *Consider the system* S' *from Example 4.5, having the form* $S' \equiv s[T] \mid \cdots$ *where* $T = T_{\mathtt{AB_1B_2}} \mid \mathrm{Q}(T_{\mathtt{AB_1B_2}})$. *We have that, by Definition 3.10, both* $\mathtt{B}_1$ *and* $\mathtt{B}_2$ *are culpable in* T*: in fact,* $T \xrightarrow{\mathtt{B}_1 \leftharpoonup \mathtt{A:req}} \twoheadrightarrow$ *and* $T \xrightarrow{\mathtt{B}_2 \leftharpoonup \mathtt{A:req}} \twoheadrightarrow$. *Therefore, by Definition 4.8, we also have that* $\mathtt{B}_1$ *and* $\mathtt{B}_2$ *are culpable at* s *in* S'*; also, we have that* $\mathtt{B}_1$ *and* $\mathtt{B}_2$ *are culpable both in* S' *and in* $(s)S'$.

A culpable participant can overcome its culpability status by firing its do prefixes, according to [CO_2-Do], until another participant becomes culpable or session s terminates. Hence, as long as a culpable participant A does not enable a do-prefix matching a contractual obligation, A will remain culpable. Note that when a participant is involved in multiple sessions, it may result culpable in more than one of them.

Theorem 4.10 (Presence of culpability in CO_2 systems). *Given a* CO_2 *system* S, *whenever* $S \to^* (s)S'$ *such that* $S' \equiv s[T] \mid \cdots$ *with* $T \not\equiv \mathbf{0}$, *then there exists at least one culpable participant at* s *in* S'.

Proof. By Theorem 3.12, we have that there exists at least one culpable participant in T — and by Definition 4.8, such a participant is also culpable at s in S'. □

Theorem 4.10 says that in an active session established through a fuse reduction, there is always *at least* one participant $\mathtt{A}[P]$ who leads the next interaction. Thus, if a corresponding $\mathsf{do}^{s}_{\mathtt{B}}\mathsf{e}$ prefix is not in P, S may get stuck, and A is culpable.

4.5. Finer-grained Forms of Agreement

In our framework, the participants firing fuse prefixes are playing the role of contract brokers. In some situations, they may want to guarantee some additional requirements before establishing a new session, in order to obtain finer levels of agreement.

In this section, we discuss some scenarios which can be addressed with simple fuse variations. The first one addresses the (possible) issue of participants playing multiple roles in a contract. The other examples add constraints to the synthesised choreography, in order to enforce *global* properties on sessions.

Example 4.11 (Single-role enforcement). *Consider the last example of Section 2: it shows a participant* $\mathtt{B}_{12}$ *that, by advertising the contract* $c_{\mathtt{B}_{12}}$, *plays the roles of two buyers in the contract advertised by the seller* A. *This is allowed since Definition 4.3 is quite liberal with respect to the substitution* π *mapping participant variables to participant names. However,* A *may want to ensure that the two buyers are actually distinct participants: i.e.,* π *must not map two participant variables to the same participant name. Since* A *is also a contract broker, it could enforce this requirement through its* fuse *prefixes, e.g., by adding the following condition to Definition 4.3:*

$$\forall i \in I \colon \forall \mathsf{a}, \mathsf{b} \in \mathrm{fv}(c_i) \colon \mathsf{a} \neq \mathsf{b} \implies \pi(\mathsf{a}) \neq \pi(\mathsf{b})$$

With this restriction in place, A *disregards the possible agreement involving* $c_{\mathtt{B}_{12}}$.

Using the same approach as in Example 4.11 (i.e., extending the requirements of Definition 4.3) we can parameterise fuse with a predicate ϕ on global types. Such a predicate is checked on the global type $\mathcal{G}$ (synthesised from of the set of latent contracts): when $\phi(\mathcal{G})$ holds for all $\mathcal{G}$, we have the standard fuse behaviour decribed in Section 4.3; in general, fuse^{ϕ} can enforce global properties emerging from the composition of individual contracts, as illustrated in Examples 4.12 and 4.13 below.

Example 4.12 (Avoiding infinite sessions). *A contract broker wants to start new sessions only if they are guaranteed to terminate within a finite number of interactions. This is useful e.g. when a system has a short life expectancy due to an approaching scheduled maintenance, and the brokers are instructed to put limitations on new sessions.*

We obtain such a behaviour by letting $\phi(\mathcal{G}) \iff \mathrm{bv}(\mathcal{G}) = \varnothing$*, where* $\mathrm{bv}(\mathcal{G})$ *is the set of bound variables in* $\mathcal{G}$*. When* $\phi(\mathcal{G})$ *holds, we have that new sessions will be created only if there are no recursion variables in* $\mathcal{G}$ *— and therefore, the choreography does not admit recursive behaviours.*

Note that this property cannot *be decided by looking at the individual contracts, only. For instance, consider:*

$$\mathtt{A}\langle \mathtt{B}!\mathtt{e}\rangle \;\mid\; \mathtt{B}\langle \mu\, \mathbf{x}.\, (\mathtt{A}?\mathtt{e} + \mathtt{A}?\mathtt{e}';\mathbf{x})\rangle$$

Here, B*'s contract is recursive; however,* A*'s contract always chooses its* non-*recursive branch. Indeed, when such contracts are composed, their global type is* $\mathtt{A} \to \mathtt{B}:\mathtt{e}$*, i.e., a non-recursive choreography.*

Example 4.13 (Requiring infinite sessions). *A contract broker wants to start only non-terminating sessions, where the participants can perpetually interact. To do that, we define* $\phi(\mathcal{G})$ *so that it holds only if* $\mathbf{0}$ *is not a sub-term of* $\mathcal{G}$*. This ensures that the choreography underlying the session does not admit terminating behaviours.*

As in Example 4.12, this property needs to be checked at the global (choreography) level; otherwise, contracts with a terminating branch would always be discarded, even when composed with other contracts that always select recursive branches.

In Section 7 we present a case study which takes advantage of the parameterised fuse primitive.

5. Domesticating Wild Choreographies via Honesty

In this section we discuss the notion of *honesty* [11], providing several examples.

Intuitively, a participant is honest when it is always able to fulfil its contractual obligations, in any context, no matter what other participants do. This is a desirable property in a distributed contract-oriented scenario: in the simplest case, honesty ensures that a participant will never violate its own contracts due to "trivial" bugs, e.g. missing do prefixes (see Example 5.6). In more complex cases, i.e., when a participant A handles multiple sessions simultaneously, honesty ensures that the erratic behaviour of some participants in a session will not cause A to remain stuck, and unable to fulfil its obligations in other sessions (see Example 6.4).

Honesty is especially important when contract-based interactions have some economic relevance: a non-honest (but not necessarily malicious) participant may be unable to reach its goals (i.e., make its contract progress), or may damage other participants if its session obligations remain unfulfilled. Since the state of CO_2 sessions allows to single out which participants are responsible for session advancement (by Definition 3.10 and Theorem 4.10), such a state could be used as "proof" for sanctioning contractual violations, possibly in an automatic way. In this scenario, a non-honest participant A may be vulnerable to attacks: other participants may induce a situation in which A cannot fulfil its contract, causing it to be punished. For all these reasons, before deploying a service, its developers might want to ensure that its implementation is honest.

Formally, the definition of honesty relies on that of *contract ready sets* (Definition 5.1) and *process ready sets* (Definition 5.3).

Definition 5.1 (Contract Ready Sets). *For a contract c, we define the set* $\mathrm{CRS}(c)$ *as follows:*

$$\mathrm{CRS}(c) = \begin{cases} \mathrm{CRS}(c') & \text{if } \ c = \mu\, \mathbf{x}.c' \\ \{\{(\mathtt{A}, \mathsf{e}_i) \mid i \in I\}\} & \text{if } \ c = \sum_{i\in I} \mathtt{A}?\mathsf{e}_i;c_i \\ \{\{(\mathtt{A}_i, \mathsf{e}_i)\} \mid i \in I\} & \text{if } \ c = \bigoplus_{i\in I} \mathtt{A}_i!\mathsf{e}_i;c_i \quad \text{and} \quad I \neq \varnothing \end{cases}$$

and we call each element of $\mathrm{CRS}(c)$ *a* contract ready set of c.

Intuitively, contract ready sets represent the immediate obligations in a contract: if $\mathtt{A}$ advertises c, then $\mathtt{A}$'s process should also offer the interaction capabilities of one of the ready sets in $\mathrm{CRS}(c)$. Each interaction is a pair $(\mathtt{A}, \mathsf{e})$, where $\mathtt{A}$ is the other party involved in the interaction, and e is the sort of the exchanged message. An external choice results in a single ready set, meaning that $\mathtt{A}$ is expected to offer all the interactions therein; instead, an internal choice results in one singleton ready set for each branch, meaning that $\mathtt{A}$ should pick one of its choice and offer the corresponding interaction (cf. Example 5.2 below). Note that Definition 5.1 above extends the notion of ready set in [11] (where only bilateral sessions are considered) to suit our multiparty contract model.

Example 5.2. *Consider the system of contracts* $T_{\mathtt{AB_1B_2}}$ *from Example 3.7, and in particular, the stipulated contracts therein, using the substitution* $\pi = \{{}^{\mathtt{A}}/_{\mathtt{a},\mathtt{a}'}, {}^{\mathtt{B_1}}/_{\mathtt{b}_1,\mathtt{b}'_1}, {}^{\mathtt{B_2}}/_{\mathtt{b}_2,\mathtt{b}'_2}\}$ *from Example 4.5:*

$$\begin{array}{rcccl} \tilde{c_{\mathtt{A}}} & = & c_{\mathtt{A}}\pi & = & \mathsf{B_1}?\mathsf{req};\mathsf{B_2}?\mathsf{req};\mathsf{B_1}!\mathsf{quote};\, (\mathsf{B_1}?\mathsf{order};\mathsf{B_2}!\mathsf{ok} + \mathsf{B_1}?\mathsf{bye};\mathsf{B_2}!\mathsf{bye}) \\ \tilde{c_{\mathtt{B_1}}} & = & c_{\mathtt{B_1}}\pi & = & \mathtt{A}!\mathsf{req};\mathtt{A}?\mathsf{quote};\, (\mathsf{B_2}!\mathsf{ok};\mathtt{A}!\mathsf{order} \oplus \mathsf{B_2}!\mathsf{bye};\mathtt{A}!\mathsf{bye}) \\ \tilde{c_{\mathtt{B_2}}} & = & c_{\mathtt{B_2}}\pi & = & \mathtt{A}!\mathsf{req};\, (\mathsf{B_1}?\mathsf{ok};\mathtt{A}?\mathsf{ok} + \mathsf{B_1}?\mathsf{bye};\mathtt{A}?\mathsf{bye}) \end{array}$$

We have $\mathrm{CRS}(\tilde{c_{\mathtt{A}}}) = \{\{(\mathsf{B_1}, \mathsf{req})\}\}$*: in other words, at this point of the contract, an interaction is expected between* $\mathtt{A}$ *and* $\mathtt{B_1}$ *(since* $\mathtt{A}$ *is waiting for* req*), while no interaction is expected between* $\mathtt{A}$ *and* $\mathtt{B_2}$*. Furthermore, since we just have a single ready set,* $\mathtt{A}$ *does not have other interactions to choose from.*

Let us now equip $T_{\mathtt{AB_1B_2}}$ *with one queue between each pair of participants, and let it perform the* req*uest exchange between* $\mathtt{B_1}$ *and* $\mathtt{A}$*, with the transitions:*

$$T_{\mathtt{AB_1B_2}} \mid \mathrm{Q}(T_{\mathtt{AB_1B_2}}) \xrightarrow{\mathtt{B_1}\to\mathtt{A}:\mathsf{req}} \xrightarrow{\mathtt{A}\leftarrow\mathtt{B_1}:\mathsf{req}} T'_{\mathtt{AB_1B_2}} \mid \mathrm{Q}(T_{\mathtt{AB_1B_2}})$$

We have that $\tilde{c_{\mathtt{A}}}$ *in* $T'_{\mathtt{AB_1B_2}}$ *is now reduced to:*

$$\tilde{c_{\mathtt{A}}}' = \mathsf{B_2}?\mathsf{req};\mathsf{B_1}!\mathsf{quote};\, (\mathsf{B_1}?\mathsf{order};\mathsf{B_2}!\mathsf{ok} + \mathsf{B_1}?\mathsf{bye};\mathsf{B_2}!\mathsf{bye})$$

and thus we have $\mathrm{CRS}(\tilde{c_{\mathtt{A}}}') = \{\{(\mathsf{B_2}, \mathsf{req})\}\}$*, i.e.,* $\mathtt{A}$ *is now waiting for a* req*uest from* $\mathtt{B_2}$*.*

If we let the system reduce further, $\tilde{c_{\mathtt{A}}}'$ *reaches its external choice:*

$$\tilde{c_{\mathtt{A}}}'' = \mathsf{B_1}?\mathsf{order};\mathsf{B_2}!\mathsf{ok} + \mathsf{B_1}?\mathsf{bye};\mathsf{B_2}!\mathsf{bye}$$

We now have a single ready set with two elements: $\mathrm{CRS}(\tilde{c_{\mathtt{A}}}'') = \{\{(\mathsf{B_1}, \mathsf{order}), (\mathsf{B_1}, \mathsf{bye})\}\}$*, i.e.,* $\mathtt{A}$ *must handle* both *answers from* $\mathtt{B_1}$*. Instead, when* $\tilde{c_{\mathtt{B_1}}}$ *reduces to its internal choice, we have:*

$$\tilde{c_{\mathtt{B_1}}}'' = \mathsf{B_2}!\mathsf{ok};\mathtt{A}!\mathsf{order} \oplus \mathsf{B_2}!\mathsf{bye};\mathtt{A}!\mathsf{bye}$$

We now have two *ready sets, each one containing one interaction:* $\mathrm{CRS}(\tilde{c_{\mathtt{B_1}}}'') = \{\{(\mathsf{B_2}, \mathsf{ok})\}, \{(\mathsf{B_2}, \mathsf{bye})\}\}$*.* $\mathtt{B_1}$ *is expected to fulfil either of them, and is free to choose which one.*

Example 5.2 shows that, when a *contract* c of a participant $\mathtt{A}$ evolves within a system T, its ready sets change accordingly. Now we need to define the counterpart of contract ready sets for CO_2 processes, i.e., the *process ready sets*.

Definition 5.3 (Process Ready Set). *For all CO_2 systems S, all participants $\mathtt{A}$ and sessions s, we define the set of pairs:*[1]

$$\mathrm{PRS}^s_{\mathtt{A}}(S) = \{(\mathtt{B}, \mathsf{e}) \mid \exists \vec{v}, \vec{\mathsf{a}} : S \equiv (\vec{v}, \vec{\mathsf{a}})\,(\mathtt{A}[\mathsf{do}^s_{\mathtt{B}}\,\mathsf{e} \,.\, \cdots + \cdots \mid \cdots] \mid \cdots) \wedge s \notin \vec{v}\}$$

Intuitively, Definition 5.3 says that the process ready set of $\mathtt{A}$ over a session s in a system S contains the interactions that $\mathtt{A}$ is immediately willing to perform with other participants through its do^s prefixes. Just like contract ready sets, the interactions are represented by participant/sort pairs. Definition 5.3 is adapted from [18] to suit our multiparty contract model.

Next, we want to characterise a weaker notion of the process ready set, so that it only takes into account the first actions on a specific session that a participant is willing to make.

Definition 5.4 (Weak Process Ready Set). *We write $S \xrightarrow{\neq(\mathtt{A}:\,\mathsf{do}^s)} S'$ iff:*

$$\exists\, \mathtt{B}, p\colon S \xrightarrow{\mathtt{B}:p} S' \;\wedge\; \big(\mathtt{A} \neq \mathtt{B} \;\vee\; \big(\forall\, \mathsf{e}\colon \forall\, \mathtt{C} : \; p = \mathsf{do}^t_{\mathtt{C}}\,\mathsf{e} \implies s \neq t\big)\big)$$

We then define the set of pairs $\mathrm{WPRS}^s_{\mathtt{A}}(S)$ *as:*

$$\mathrm{WPRS}^s_{\mathtt{A}}(S) \;=\; \left\{(\mathtt{B}, \mathsf{e}) \;\middle|\; \exists S'\colon S \xrightarrow{\neq(\mathtt{A}:\,\mathsf{do}^s)}{}^{*}\; S' \wedge (\mathtt{B}, \mathsf{e}) \in \mathrm{PRS}^s_{\mathtt{A}}(S')\right\}$$

In Definition 5.4, the transition relation $\xrightarrow{\neq(\mathtt{A}:\,\mathsf{do}^s)}$ abstracts from the reductions of S which are *not* due to $\mathtt{A}$'s actions on session s. In other words, such a relation allows a system S with a participant $\mathtt{A}[P]$ to reduce freely, until a reduct S' containing $\mathtt{A}[P']$ is reached, such that a do^s prefix is at top-level in P'. Note that, since Definition 5.4 builds upon Definition 5.3, if $s \notin \mathrm{fv}(S)$, then necessarily $\mathrm{WPRS}^s_{\mathtt{A}}(S) = \varnothing$: i.e., observing the weak process ready set of a session s is only meaningful after s has been established, and it is not delimited (and thus it cannot be α-converted).

We now introduce the final ingredient required to define honesty, that is the *readiness* of a participant.

Definition 5.5 (Readiness). *We say that $\mathtt{A}$ is ready in S iff, whenever $S \equiv (\vec{u})(\vec{\mathtt{b}})S_0$ for some $\vec{u}, \vec{\mathtt{b}}$ and $S_0 \equiv s[\mathtt{A}\langle c\rangle \mid \cdots] \mid \cdots$, the following holds:*

$$\exists \mathcal{X} \in \mathrm{CRS}(c) \;:\; \mathcal{X} \subseteq \mathrm{WPRS}^s_{\mathtt{A}}(S_0)$$

Definition 5.5 says that a participant $\mathtt{A}$ is *ready* in a system S when, for all its established sessions s, S can reduce in such a way that $\mathtt{A}$'s process will provide at s *all* the interactions expected by one of the ready sets of $\mathtt{A}$'s contract at s. In other words, when $\mathtt{A}$ is "ready", then, for all its contracts c already involved in a session, $\mathtt{A}$ has a way to fulfil at least its immediate obligations in c.

Example 5.6. *Consider the system S in Example 4.5. We have seen that, after its latent contracts have been* fused*, we obtain:*

$$(s)S' \quad\equiv\quad (s)\big(\mathtt{A}[P_{\mathtt{A}}\sigma\pi] \quad\mid\quad s[T_{\mathtt{AB_1B_2}} \mid \mathrm{Q}(T_{\mathtt{AB_1B_2}})] \quad\mid\quad \mathtt{B}_1[P_{\mathtt{B}_1}\sigma\pi] \quad\mid\quad \mathtt{B}_2[P_{\mathtt{B}_2}\sigma\pi]\big)$$

where the substitutions σ and π are also from Example 4.5. Let us define the processes (after substitutions):

$$\begin{array}{rcl} P_{\mathtt{A}}\sigma\pi & = & \mathsf{do}^s_{\mathtt{B}_1}\,\mathsf{req} \,.\, \mathsf{do}^s_{\mathtt{B}_2}\,\mathsf{req} \,.\, \mathsf{do}^s_{\mathtt{B}_1}\,\mathsf{quote} \,.\, \big(\mathsf{do}^s_{\mathtt{B}_1}\,\mathsf{order} \,.\, \mathsf{do}^s_{\mathtt{B}_2}\,\mathsf{ok} \;+\; \mathsf{do}^s_{\mathtt{B}_1}\,\mathsf{bye} \,.\, \mathsf{do}^s_{\mathtt{B}_2}\,\mathsf{bye}\big) \\ P_{\mathtt{B}_1}\sigma\pi & = & \tau \,.\, \mathsf{do}^s_{\mathtt{A}}\,\mathsf{req} \,.\, \mathsf{do}^s_{\mathtt{A}}\,\mathsf{quote} \,.\, \mathsf{do}^s_{\mathtt{A}}\,\mathsf{order} \\ P_{\mathtt{B}_2}\sigma\pi & = & \mathsf{do}^s_{\mathtt{A}}\,\mathsf{req} \,.\, \big(\mathsf{do}^s_{\mathtt{B}_1}\,\mathsf{ok} \,.\, \mathsf{do}^s_{\mathtt{A}}\,\mathsf{ok} \;+\; \mathsf{do}^s_{\mathtt{B}_1}\,\mathsf{bye} \,.\, \mathsf{do}^s_{\mathtt{A}}\,\mathsf{bye}\big) \end{array}$$

[1] The side condition "$s \notin \vec{v}$" of Definition 5.3 deals with cases like $S_0 = (s)\,(\mathtt{A}[\mathsf{do}^s_{\mathtt{B}}\,\mathsf{int}])$ and $S = S_0 \mid s[\mathtt{A}\langle \mathtt{B}!\mathsf{int}\rangle \mid \ldots] \mid \ldots$: without the side condition, $\mathrm{PRS}^s_{\mathtt{A}}(S_0) = \{\{(\mathtt{B}, \mathsf{int})\}\}$ — hence, by Definition 5.5, $\mathtt{A}$ would result to be ready in S.

We have:

$$\begin{array}{lcccl} \mathrm{PRS}^{s}_{\mathtt{A}}(S') & = & \{(\mathtt{B}_1, \mathsf{req})\} & = & \mathrm{WPRS}^{s}_{\mathtt{A}}(S') \\ \mathrm{PRS}^{s}_{\mathtt{B}_1}(S') & = & \varnothing \neq \{(\mathtt{A}, \mathsf{req})\} & = & \mathrm{WPRS}^{s}_{\mathtt{B}_1}(S') \\ \mathrm{PRS}^{s}_{\mathtt{B}_2}(S') & = & \{(\mathtt{A}, \mathsf{req})\} & = & \mathrm{WPRS}^{s}_{\mathtt{B}_2}(S') \end{array}$$

Note that the τ prefix in $P_{\mathtt{B}_1}$ prevents $\mathtt{B}_1$ from interacting immediately with $\mathtt{A}$ on session s, although it is "weakly ready" to do so. Hence, considering that the weak process ready sets of each participant in S'_1 match their respective contract ready sets in $T_{\mathtt{AB}_1\mathtt{B}_2}$ (Example 5.2) according to Definition 5.5 we have that participants $\mathtt{A}$, $\mathtt{B}_1$ and $\mathtt{B}_2$ are all ready in $(s)S'_1$.

Before defining honesty formally, we need to characterise the class of systems for which this concept is meaningful, — i.e., those systems where a participant is not (yet) involved in latent contracts nor active sessions, and whose process has no free variables.

Definition 5.7 (Initial System). *A CO_2 system S is* $\mathtt{A}$-initial *if S has no sub-term of the form $\downarrow_{\_}\ \mathtt{A}$ says c or $\mathtt{A}\langle c\rangle$ with $c \not\equiv \mathbf{0}$, and whenever S has a subterm of the form $\mathtt{A}[P]$, then P is closed. A CO_2 system S is* initial *when it is $\mathtt{A}$-initial for all participants $\mathtt{A}$ in S.*

Definition 5.8 (Honesty). *We say that $\mathtt{A}[P]$ is* honest *iff, for all $\mathtt{A}$-initial $S \equiv \mathtt{A}[P] \mid S_0$ s.t. $S \to^* S'$, $\mathtt{A}$ is ready in S'.*

A process $\mathtt{A}[P]$ is said to be honest when, for all contexts and reductions that $\mathtt{A}[P]$ may be engaged in, $\mathtt{A}$ is persistently ready. In other words, for all context S_0 with which $\mathtt{A}[P]$ may be composed, and for all systems S' resulting from the continuation of such a composition, there must be a correspondence between the interactions required by $\mathtt{A}$'s stipulated contracts and $\mathtt{A}$'s (weak) process ready set. The definition requires an $\mathtt{A}$-initial system to rule out contexts with latent/stipulated contracts of $\mathtt{A}$ — otherwise, $\mathtt{A}$ could be made trivially dishonest, e.g. by inserting a latent contract $\downarrow_x\ \mathtt{A}$ *says* c that $\mathtt{A}$ cannot fulfil. Notice that $\mathtt{A}[P]$ is trivially honest when P advertises no contracts.

Example 5.9. *Consider the process $\mathtt{B}_1[\mathsf{tell}_{\mathtt{A}}\downarrow_y c_{\mathtt{B}_1} . P_{\mathtt{B}_1}]$ of system S, as defined in Examples 4.5 and 5.6. We show that this process is* not *honest. In fact, S can reduce as $S \to^* (s)S' \to^* (s)S''$, where:*

$$\begin{aligned} (s)S'' \;=\; (s)\Big(&\mathtt{A}\big[\mathsf{do}^{s}_{\mathtt{B}_1}\ \mathsf{order} \,.\, \mathsf{do}^{s}_{\mathtt{B}_2}\ \mathsf{ok} \;+\; \mathsf{do}^{s}_{\mathtt{B}_1}\ \mathsf{bye} \,.\, \mathsf{do}^{s}_{\mathtt{B}_2}\ \mathsf{bye}\big] \\ &\mid\; s\big[\mathtt{A}\langle \mathtt{B}_1?\mathsf{order};\mathtt{B}_2!\mathsf{ok} + \mathtt{B}_1?\mathsf{bye};\mathtt{B}_2!\mathsf{bye}\rangle \\ &\qquad \mid \mathtt{B}_1\langle \mathtt{B}_2!\mathsf{ok};\mathtt{A}!\mathsf{order} \oplus \mathtt{B}_2!\mathsf{bye};\mathtt{A}!\mathsf{bye}\rangle \\ &\qquad \mid \mathtt{B}_2\langle \mathtt{B}_1?\mathsf{ok};\mathtt{A}?\mathsf{ok} + \mathtt{B}_1?\mathsf{bye};\mathtt{A}?\mathsf{bye}\rangle\ \big] \\ &\mid\; \mathtt{B}_1\big[\mathsf{do}^{s}_{\mathtt{A}}\ \mathsf{order}\big] \;\mid\; \mathtt{B}_2\big[\mathsf{do}^{s}_{\mathtt{B}_1}\ \mathsf{ok} \,.\, \mathsf{do}^{s}_{\mathtt{A}}\ \mathsf{ok} \;+\; \mathsf{do}^{s}_{\mathtt{B}_1}\ \mathsf{bye} \,.\, \mathsf{do}^{s}_{\mathtt{A}}\ \mathsf{bye}\big]\Big) \end{aligned}$$

At this point, we see that there is a problem in the implementation of $\mathtt{B}_1$: it does not notify the other buyer before making an order. In fact, $\mathtt{B}_1$'s process is trying to perform $\mathsf{do}^{s}_{\mathtt{A}}\ \mathsf{order}$, but its contract requires that $\mathsf{do}^{s}_{\mathtt{B}_2}\ \mathsf{ok}$ is performed first (or $\mathsf{do}^{s}_{\mathtt{B}_2}\ \mathsf{bye}$, if the quote is rejected). This is reflected by the mismatch between $\mathtt{B}_1$'s process ready set in S'', and its contract ready sets, in session s:

$$\begin{array}{rcl} \mathrm{PRS}^{s}_{\mathtt{B}_1}(S'') & = & \{\{(\mathtt{A}, \mathsf{order})\}\} \\ \mathrm{CRS}(\mathtt{B}_2!\mathsf{ok};\mathtt{A}!\mathsf{order} \oplus \mathtt{B}_2!\mathsf{bye};\mathtt{A}!\mathsf{bye}) & = & \{\{(\mathtt{B}_2, \mathsf{ok})\}, \{(\mathtt{B}_2, \mathsf{bye})\}\} \end{array}$$

In terms of the definitions above, we can say that there exists a system S — containing $\mathtt{B}_1[\mathsf{tell}_{\mathtt{A}}\downarrow_y c_{\mathtt{B}_1} . P_{\mathtt{B}_1}]$ — that reduces to a $(s)S''$ where $\mathtt{B}_1$ is not ready (Definition 5.5). Therefore, $\mathtt{B}_1[\mathsf{tell}_{\mathtt{A}}\downarrow_y c_{\mathtt{B}_1} . P_{\mathtt{B}_1}]$ is not honest. In fact, $\mathtt{B}_1$ is culpable in $(s)S''$, according to Definition 4.8.

As in [18], the definition of honesty subsumes a *fair* scheduler, eventually allowing participants to fire persistently (weakly) enabled do actions.

6. Properties of CO_2 Systems

We now give the main properties of our framework. We start by proving that honest participants can always perform their contractual obligations without relying on the context (Theorem 6.1). Then, we formalise a link between the honesty of participants, and two key properties borrowed from the session types literature: Theorem 6.2 introduces session fidelity in CO_2, while Corollary 6.3 (descending from Theorem 6.1) introduces a notion of progress of CO_2 systems.

Theorem 6.1 below descends from the definition of honesty, formalising the fact that honest participants can always fulfil their contractual obligations, at all sessions s they are involved in, by eventually firing their do^s prefixes. Most importantly, these do^s prefixes can be fired even if the context does not always cooperate — in particular, when other participants fail to respect their own contracts. In this situation, honest participants will never be stuck in a culpable condition: there always exists a way for them to reduce by only relying on their own prefixes.

Theorem 6.1 (Session progress). *Let S_0 be an $\mathtt{A}$-initial CO_2 system with $\mathtt{A}[P_0]$ honest. If $S_0 \to^* (s)S$, where $S \equiv s[T] \mid \mathtt{A}[P] \mid \cdots$ and $\mathtt{A}$ is culpable at s in S, then:*

$$S \xrightarrow{\mathtt{A}:\, p_1} \cdots \xrightarrow{\mathtt{A}:\, p_n} \xrightarrow{\mathtt{A}:\, \mathsf{do}_s^{\mathtt{B}}\, \mathsf{e}}$$

for some $\mathtt{B}$, e, and $p_1, \ldots, p_n$ such that $n \geq 0$, and $p_i \neq \mathsf{fuse}$ for all $i \in 1..n$.

Proof. By assumption, $\mathtt{A}$ is culpable at s in S. By Definition 5.8, $\mathtt{A}$ is *ready* in any reduction of S_0 — in particular in $(s)S$. Therefore, by Definition 5.5, there exists $\mathcal{X} \in \mathrm{CRS}(c)$ such that $\mathcal{X} \subseteq \mathrm{WPRS}^s_{\mathtt{A}}(S)$. From these premises, we have that there must exist some $\mathtt{B}, \mathsf{e}$ such that $T \xrightarrow{\mathtt{A} \Rightarrow \mathtt{B}:\mathsf{e}}\!\!\!\!\rightarrow$ (due to culpability) and $(\mathtt{B}, \mathsf{e}) \in \mathrm{WPRS}^s_{\mathtt{A}}(S)$ (due to readiness). By Definition 5.4, it means:

$$\exists S' \colon S \xrightarrow{\neq(\mathtt{A}:\, \mathsf{do}^s)}{}^{*}\ S' \ \wedge\ (\mathtt{B}, \mathsf{e}) \in \mathrm{PRS}^s_{\mathtt{A}}(S') \tag{6.1}$$

and therefore, by Definition 5.3, we must have that $\mathtt{A}[\mathsf{do}^s_{\mathtt{B}}\, \mathsf{e} \,.\, P' + P'' \mid Q]$ is a participant of S'. Thus, we have to show that $\mathtt{A}[P]$ in S may reduce to such a process, with $\mathsf{do}^s_{\mathtt{B}}\, \mathsf{e}$ enabled, by firing its own prefixes $p_1, \ldots, p_n$ *only* — except fuse.

By contradiction, assume that *none* of the traces from S to S' (following (6.1)) satisfies the thesis. We show that, in this case, $\mathtt{A}[P_0]$ would *not* be honest: in fact, there would exist a system $\tilde{S}_0 \equiv \mathtt{A}[P_0] \mid \cdots$ with a reduct $(s)\tilde{S}$ such that $\tilde{S} \equiv s[T] \mid \mathtt{A}[P] \mid \cdots$, where $\mathtt{A}$ would *not* be ready. Our goal is to create such a system $\tilde{S}_0$.

Before proceeding, we define some auxiliary functions.

Given two CO_2 systems S_A, S_B such that there exists a transition $S_A \xrightarrow{\ell} S_B$, $\mathrm{inLab}(S_A, S_B)$ gives the "innermost" label in the derivation inducing such a transition. Intuitively, it pierces through delimitations, and returns the underlying CO_2 prefix before it is turned into τ by an application of rule [CO_2-Del]:

$$\mathrm{inLab}(S_A, S_B) \;=\; \begin{cases} \ell_0 & \text{if} \quad [\mathrm{R}]\ \dfrac{\vdots}{S_A \xrightarrow{\ell_0} S_B} \quad \text{and} \quad [\mathrm{R}] \notin \{[\mathrm{CO_2\text{-}Del}], [\mathrm{CO_2\text{-}Par}]\} \\[2ex] \mathrm{inLab}(S'_A, S'_B) & \text{if} \quad [\mathrm{R}]\ \dfrac{S'_A \xrightarrow{\ell_1} S'_B}{S_A \xrightarrow{\ell_0} S_B} \quad \text{and} \quad [\mathrm{R}] \in \{[\mathrm{CO_2\text{-}Del}], [\mathrm{CO_2\text{-}Par}]\} \end{cases}$$

Given a sequence of CO_2 systems $\vec{\mathcal{S}}_0$ such that each element is a reduct of the preceding one, $\mathrm{inTr}\left(\vec{\mathcal{S}}_0\right)$ gives the trace composed by the "innermost" labels of each reduction:

$$\begin{aligned} \mathrm{inTr}(S_A \cdot \epsilon) &= \epsilon \qquad (\text{where } \epsilon \text{ is the empty sequence}) \\ \mathrm{inTr}\left(S_A \cdot S_B \cdot \vec{\mathcal{S}}_0\right) &= \mathrm{inLab}(S_A, S_B) \cdot \mathrm{inTr}\left(S_B \cdot \vec{\mathcal{S}}_0\right) \end{aligned}$$

Let us now consider an execution going from the $\mathtt{A}$-initial S_0 to $(s)S$ in the hypotheses, and let $\vec{\mathcal{S}}$ be the sequence of CO_2 system configurations composing such an execution; furthermore, let us define the "inner" trace $\vec{\mathcal{T}} = \mathrm{inTr}\left(\vec{\mathcal{S}}\right)$.

With this information at hand, we can build $\tilde{S}_0$ so that:

1. $\tilde{S}_0 \equiv \mathtt{A}[P_0] \mid \tilde{S}'_0$;
2. $\tilde{S}'_0$ contains:
   (a) *all* the sessions appearing in S;
   (b) all the latent contracts appearing in S_0, *except* those that are *not* fused along $\vec{\mathcal{S}}$ and thus appear in $(s)S$;
   (c) participants $\mathtt{B}_1[P_1], \ldots, \mathtt{B}_n[P_n]$, where $\mathtt{B}_1, \ldots, \mathtt{B}_n \neq \mathtt{A}$ also appear in S, and $P_1, \ldots, P_n$ are written so that:
      i. *all* the "innermost" prefixes in $\vec{\mathcal{T}}$ fired by $\mathtt{B}_1, \ldots, \mathtt{B}_n$ can still be fired in the same order — *with the exception of the* tell *prefixes corresponding to latent contracts in* $(s)S$, which are neglected;
      ii. after the prefixes in $\vec{\mathcal{T}}$ are fired, all $\mathtt{B}_1[P_1] \ldots, \mathtt{B}_n[P_n]$ reduce to $\mathtt{B}_1[\mathbf{0}], \ldots, \mathtt{B}_n[\mathbf{0}]$.

As a result, we have that $\tilde{S}_0$ defined above can reduce to a system $(s)\tilde{S}$ such that $\tilde{S} \equiv s[T] \mid \mathtt{A}[P] \mid \cdots$ through a "inner" trace $\vec{\mathcal{T}}'$ whose prefixes match those in $\vec{\mathcal{T}}$, *except* for the lack of tell prefixes fired by any $\mathtt{B} \neq \mathtt{A}$ (by item 2(c)i above). Furthermore, we have that:

1. no latent contracts appear in $\tilde{S}$ (unless they have been advertised by $\mathtt{A}$);
2. all established sessions in S also appear in $\tilde{S}$, and *vice versa*;
3. except for $\mathtt{A}[P]$, all participants in $\tilde{S}$ have the form $\mathtt{B}[\mathbf{0}]$.

Hence, no participant in $\tilde{S}$ can fire a fuse prefix, and no participant $\mathtt{B} \neq \mathtt{A}$ in $\tilde{S}$ has any prefix enabled. Now, recall that we assumed (by contradiction) that Equation (6.1) cannot be satisfied with a trace consisting only of $\mathtt{A}[P]$'s prefixes (except fuse). Therefore:

$$\nexists \tilde{S}' \colon \tilde{S} \xrightarrow{\neq(\mathtt{A}\colon \mathsf{do}^s)}{}^{*}\ \tilde{S}' \ \wedge\ (\mathtt{B}, \mathsf{e}) \in \mathrm{PRS}^s_{\mathtt{A}}\left(\tilde{S}'\right)$$

But then, there exists an $\mathtt{A}$-initial system $\tilde{S}_0 \equiv \mathtt{A}[P] \mid \cdots$ such that $\tilde{S}_0 \to^* (s)\tilde{S}$, and $\mathtt{A}$ is *not* ready in $(s)\tilde{S}$. Therefore, we must conclude that $\mathtt{A}[P]$ is *not* honest — thus violating the theorem hypothesis. □

Theorem 6.2 below says that each (honest) participant will strictly adhere to its contracts, once they have been fused in a session s: no extra-contractual interactions are possible on s, and honest participants will eventually perform do^s actions, as long as it is required by their contracts.

Theorem 6.2 (Session fidelity). *Given a CO_2 system S_0 containing $\mathtt{A}[P_0]$, whenever $S \to^* (s)S$ with $S \equiv s[T] \mid \cdots$, then:*

1. $S \xrightarrow{\neq(\mathtt{A}\colon \mathsf{do}^s)}{}^{*} \xrightarrow{\mathtt{A}\colon \mathsf{do}^s_{\mathtt{B}}\ \mathsf{e}} \implies \exists\, n \geq 0 \colon T \xrightarrow{\lambda_1}\!\!\!\!\rightarrow \cdots \xrightarrow{\lambda n}\!\!\!\!\rightarrow \xrightarrow{\mathtt{A} \leftrightharpoons \mathtt{B}:\mathsf{e}}\!\!\!\!\rightarrow$ *where* $\forall \mathtt{C}, \mathsf{e}', i = 1..n \colon \lambda_i \neq \mathtt{A} \leftrightharpoons \mathtt{C} : \mathsf{e}'$;
2. *If S is $\mathtt{A}$-initial, $\mathtt{A}[P_0]$ honest, and $\mathcal{Y} = \left\{(\mathtt{B}, \mathsf{e}) \mid T \xrightarrow{\mathtt{A} \leftrightharpoons \mathtt{B}:\mathsf{e}}\!\!\!\!\rightarrow\right\} \neq \varnothing$, then*

$$\exists (\mathtt{B}, \mathsf{e}) \in \mathcal{Y} \text{ such that } S \xrightarrow{\neq(\mathtt{A}\colon \mathsf{do}^s)}{}^{*} \xrightarrow{\mathtt{A}\colon \mathsf{do}^s_{\mathtt{B}}\ \mathsf{e}}$$

Recall that, as in Definition 5.4, $\xrightarrow{\neq(\mathtt{A}\colon \mathsf{do}^s)}{}^{*}$ intuitively stands for any reduction not *involving $\mathtt{A}$ on session s.*

Proof. Item 1 follows directly from the semantics of CO_2 (in particular, rule [CO_2-DO]): if $\mathtt{A}$ is able to fire a $\mathsf{do}^s_{\mathtt{B}}\,\mathsf{e}$ prefix, then its contract must necessarily allow for sending/receiving e to/from $\mathtt{B}$.

Item 2 follows from the honesty hypothesis for $\mathtt{A}[P_0]$, and from Definition 5.5 (readiness). Let us fix $n \geq 1$ enumerating *all* enabled interactions in T driven by $\mathtt{A}$ — i.e., let n be the cardinality of $\mathcal{Y}$ in the theorem statement. Let us consider $c = T(\mathtt{A})$, i.e., $\mathtt{A}$'s contract in T. Due to $\mathtt{A}$'s readiness, c is such that, either:

1. $\mathrm{CRS}(c) = \left\{ \{(\mathtt{B}_j, \mathsf{e}_j)\}_{j=1..n} \right\}$; or,
2. $\mathrm{CRS}(c) = \{\{(\mathtt{B}_j, \mathsf{e}_j)\}\}_{j=1..n}$.

where $\{(\mathtt{B}_j, \mathsf{e}_j)\}_{j=1..n} = \mathcal{Y}$ — i.e., $\mathcal{Y}$ contains exactly the $(\mathtt{B}, \mathsf{e})$ pairs appearing in $\mathrm{CRS}(c)$. Recall that, since S_0 is $\mathtt{A}$-initial and $\mathtt{A}$ is honest, then $\mathtt{A}$ must be *ready* in all possible reducts of S_0 — and in particular, it must be ready in $(s)(S)$. Therefore, we have that $\exists \mathcal{X} \in \mathrm{CRS}(c)$ such that $\mathcal{X} \subseteq \mathrm{WPRS}^s_{\mathtt{A}}(S')$, i.e.:

$$\exists \mathcal{X} = \{(\mathtt{B}, \mathsf{e}), \cdots\} \in \mathrm{CRS}(c) \text{ such that } S' \xrightarrow{\neq(\mathtt{A}:\, \mathsf{do}^s)}{}^{*} \xrightarrow{\mathtt{A}:\, \mathsf{do}^s_{\mathtt{B}}\,\mathsf{e}} \tag{6.2}$$

Since the $(\mathtt{B}, \mathsf{e})$ pair satisfying Equation 6.2 also belongs to $\mathcal{Y}$, the thesis holds. □

Note that, by Theorem 6.1, the conclusion of Item 2 of Theorem 6.2 above may be stricter: i.e., the $\mathsf{do}^s_{\mathtt{B}}\,\mathsf{e}$ transition may be reached by firing $\mathtt{A}$'s prefixes only (excluding fuse).

Corollary 6.3 below introduces a notion of "global progress" in CO_2. This notion is only meaningful *after* a session has been established, and thus a culpable participant exists. In fact, a system without sessions may not progress — either because a set of compliant contracts cannot be found, or because a fuse prefix is not enabled. In both cases, no participant may be deemed culpable, and thus responsible for the next move. However, the system could progress again if it is composed with another system consisting of (honest) participants, allowing for a new session to be established.

Corollary 6.3. *Given an initial CO_2 system S_0 with only honest participants, if $S_0 \to^* S \equiv (s)\,(s[T] \mid \cdots)$ with $T \not\equiv \mathbf{0}$, then $S \to$.*

Proof. By contradiction. From [6], we know that T is deadlock free, thus if the CO_2 system cannot make further reductions, it must be because one of the participant $\mathtt{A}$ cannot meet its obligations. This is in contradiction with Theorem 6.1, since all participants are honest in S_0. □

This result also holds for systems where a process takes part in multiple sessions: the honesty of all participants guarantees that all sessions will (eventually) progress.

Example 6.4. *We now give a simple example of a system with multiple sessions. We show how a seemingly honest process $\mathtt{B}$ could be deemed culpable due to the unexpected behaviour of other participants, and how honest participants guarantee progress of the whole system. Consider:*

$$\begin{aligned} S = (x, y, z, w)\big(\ & \mathtt{A}[\mathsf{tell}_{\mathtt{A}}\downarrow_x (\mathtt{B}!\mathsf{int})\,.\,\mathsf{fuse}\,.\,\mathsf{fuse}] \\ \mid\ & \mathtt{B}[\mathsf{tell}_{\mathtt{A}}\downarrow_y (\mathtt{A}?\mathsf{int})\,.\,\mathsf{tell}_{\mathtt{A}}\downarrow_z (\mathtt{C}!\mathsf{bool})\,.\,\mathsf{do}^y_{\mathtt{A}}\,\mathsf{int}\,.\,\mathsf{do}^z_{\mathtt{C}}\,\mathsf{bool}] \\ \mid\ & \mathtt{C}[\mathsf{tell}_{\mathtt{A}}\downarrow_w (\mathtt{B}?\mathsf{bool})\,.\,\mathsf{do}^w_{\mathtt{B}}\,\mathsf{bool}]\ \big) \end{aligned}$$

After all four contracts have been advertised to $\mathtt{A}$ and fused, the system reduces to:

$$\begin{aligned} S' = (s_1, s_2)\big(\ & \mathtt{A}[\mathbf{0}] \quad\mid\quad \mathtt{B}[\mathsf{do}^{s_1}_{\mathtt{A}}\,\mathsf{int}\,.\,\mathsf{do}^{s_2}_{\mathtt{C}}\,\mathsf{bool}] \quad\mid\quad \mathtt{C}[\mathsf{do}^{s_2}_{\mathtt{B}}\,\mathsf{bool}] \\ \mid\ & s_1[\mathtt{A}\langle \mathtt{B}?\mathsf{int}\rangle \mid \mathtt{B}\langle \mathtt{A}!\mathsf{int}\rangle] \\ \mid\ & s_2[\mathtt{B}\langle \mathtt{C}!\mathsf{bool}\rangle \mid \mathtt{C}\langle \mathtt{B}?\mathsf{bool}\rangle]\ \big) \end{aligned}$$

Even if both sessions s_1 and s_2 enjoy contractual progress, S' is stuck: $\mathtt{A}$ does not perform the promised action, thus remaining culpable in s_1; $\mathtt{B}$ is stuck waiting in s_1, thus remaining culpable in s_2.[2] *Indeed,*

[2] In this case, $\mathtt{B}$ is deemed culpable in s_2 because its implementation did not expect $\mathtt{A}$ to misbehave.

neither A *nor* B *are ready in* S'*, and thus they are not honest in* S*. Hence, global progress is not guaranteed. Let us now consider the following variant of* S*, where all participants are honest:*

$$\begin{aligned}\hat{S} = (x,y,z,w)\big(\ \mathtt{A}[&(\mathsf{tell}_{\mathtt{A}}\downarrow_x \ (\mathtt{B}!\mathsf{int})\ .\ \mathsf{do}^{x}_{\mathtt{B}}\ \mathsf{int})\mid \mathsf{fuse}\mid \mathsf{fuse}]\\ \mid\ &\mathtt{C}[\mathsf{tell}_{\mathtt{A}}\downarrow_w \ (\mathtt{B}?\mathsf{bool})\ .\ \mathsf{do}^{w}_{\mathtt{B}'}\ \mathsf{bool}]\\ \mid\ &\mathtt{B}[\mathsf{tell}_{\mathtt{A}}\downarrow_y \ (\mathtt{A}?\mathsf{int})\ .\ \mathsf{tell}_{\mathtt{A}}\downarrow_z \ (\mathtt{C}!\mathsf{bool})\ .(\mathsf{do}^{y}_{\mathtt{A}}\ \mathsf{int}\ .\ \mathsf{do}^{z}_{\mathtt{C}}\ \mathsf{bool}\\ &\qquad\qquad + \tau\ .\ (\mathsf{do}^{y}_{\mathtt{A}}\ \mathsf{int}\mid \mathsf{do}^{z}_{\mathtt{C}}\ \mathsf{bool}))]\ \big)\end{aligned}$$

In this case, A *will respect its contractual duties, while* B *will be ready to fulfil its contracts on both sessions — even if one is not activated, or remains stuck (here,* τ *represents an internal action, e.g., a timeout: if the first* $\mathsf{do}^{y}_{\mathtt{A}}\ \mathsf{int}$ *cannot reduce,* B *falls back to running the two sessions in parallel). The honesty of all participants in* $\hat{S}$ *guarantees that, once a session is active, it will reach its completion.*

7. Case Study: Sharing Knowledge

We illustrate the suitability of our framework to reason about distributed systems where contract brokers are willing to offer specific guarantees to their clients. In particular, we illustrate how three key components of our framework, i.e.,(*i*) multiparty sessions, (*ii*) the notion of honesty, and (*iii*) contractual agreements based on choreographies, address complex challenges encountered in the design and specification of distributed systems.

We present a case study where a contract broker wants to provide a specific service to its clients: guaranteeing that participants will interact according to a class of *gossip protocols* [15]. Gossip protocols are used to spread a piece of information among a group of participants, ensuring that each one of them will eventually acquire such information. In particular, we focus on *"knowledge sharing"* protocols, where each participant owns some *initial knowledge*, which must be shared across all participants before the participants terminate, i.e., everyone must eventually receive each other's initial knowledge.

For instance, assuming that k is a message sort representing the current knowledge of a participant, the system of contracts on the left below implements a sharing knowledge protocol, while the one on the right does not. Indeed, on the right-hand side, A does not acquire B's knowledge.

$$\mathtt{A}\langle \mathtt{B}!\mathsf{k};\mathtt{B}?\mathsf{k}\rangle \mid \mathtt{B}\langle \mathtt{A}?\mathsf{k};\mathtt{A}!\mathsf{k}\rangle \quad \checkmark \qquad\qquad \mathtt{A}\langle \mathtt{B}!\mathsf{k}\rangle \mid \mathtt{B}\langle \mathtt{A}?\mathsf{k}\rangle \quad \times$$

We distinguish three challenges that one needs to address to ensure that a broker provides effectively a knowledge sharing service. (*a*) Intuitively, for a set of participants to share knowledge, they may interact either via a single (multiparty) session or via several binary sessions. However, guaranteeing safety and liveness properties across several, possibly interleaved, sessions is a notoriously difficult problem (see [19] for instance) and it is even harder to check for semantic properties such as knowledge sharing. Instead, intra-sessions properties are much easier to attain [3]; and in our case a broker is able to take a well-informed decision before starting a session by analysing its choreography. (*b*) In real world applications, e.g., in cloud computing [20], one cannot assume that each component will behave as expected. In our framework, it is always possible to discover which participants are not fulfilling their contract (culpability); and the honesty of a participant guarantees that it will always fulfil its contract. Therefore, if an honest participant joins a knowledge sharing session, it will always share all its knowledge. (*c*) Properties of any contract model that offers a good level of expressiveness (e.g., one based on FIFO queues like ours) take the risk of being hard to analyse (or even undecidable [21]). In such a case, one needs to find sound and efficient ways to decide whether a given set of contracts satisfy certain properties. Below, we give an effective decision procedure that guarantees knowledge sharing of a set of contracts by parsing only a global type.

We begin this case study by introducing a formal model for transmission and acquisition of knowledge (Section 7.1); then we show how knowledge is transmitted/acquired in systems of contracts and CO_2 systems (Section 7.2). Finally, in Section 7.3 we introduce a variant of the parameterised fuse^{ϕ} primitive (cf. Section 4.5) which guarantees that new sessions satisfy the knowledge sharing property. For the sake of readability, some auxiliary technical details are relegated to in the appendix [17].

7.1. Semantic knowledge sharing

We introduce a formalization of knowledge transmission and acquisition among a set of participants through labelled transition systems (LTS), which may be induced by both system of contracts and CO_2 systems.

Definition 7.1 (Knowledge of a participant). *The* knowledge of a participant *is a pair* $(\mathcal{A}, \mathcal{Q})$*, where* $\mathcal{A}$ *(*acquired knowledge*) is a set of participant names, and* $\mathcal{Q}$ *(*pending knowledge*) is a map from participant names to queues of the form* $\mathcal{A}' \cdot \mathcal{A}'' \cdot \ldots$*, where* $\mathcal{A}', \mathcal{A}''$ *are sets of participant names. We write* $\varnothing$ *when* $\mathcal{Q}$ *maps each participant to the empty queue.*

If the knowledge of $\mathtt{B}$ is $(\{\mathtt{B}\}, \varnothing)$, then $\mathtt{B}$ has only its own initial knowledge. The knowledge of $\mathtt{B}$ may increase as it interacts with other participant. For instance, if the knowledge of $\mathtt{B}$ is $(\{\mathtt{B},\mathtt{E}\}, \{\mathtt{A}:\{\mathtt{A}\}, \mathtt{C}:\{\mathtt{C}\} \cdot \{\mathtt{C},\mathtt{D}\}\})$; then $\mathtt{B}$ has acquired $\mathtt{E}$'s initial knowledge, while some knowledge is pending, i.e., the initial knowledge of $\mathtt{A}$ (transmitted by $\mathtt{A}$ itself) and the initial knowledge of $\mathtt{C}$, followed by $\mathtt{C}$'s initial knowledge and $\mathtt{D}$'s initial knowledge.

We introduce an LTS for knowledge transmission/acquisition in Definition 7.2 below. We use two operators: $\mathrm{push}_{\mathtt{B}}((\mathcal{A}, \mathcal{Q}), (\mathcal{A}', \mathcal{Q}'))$ which augments the knowledge of a participant with the knowledge transmitted by $\mathtt{B}$, i.e., the knowledge $\mathcal{A}'$ sent by $\mathtt{B}$ is appended to $\mathcal{Q}$; and $\mathrm{pop}_{\mathtt{B}}(\mathcal{A}, \mathcal{Q})$ moves the first pending knowledge sent by $\mathtt{B}$ to the acquired knowledge $\mathcal{A}$. For instance, we have:

$$\begin{aligned} \mathrm{push}_{\mathtt{B}}((\{\mathtt{A}\}, \{\mathtt{C}:\{\mathtt{C}\}, \mathtt{B}:\{\mathtt{B}\}\}) \ , \ (\{\mathtt{A},\mathtt{B}\}, \{\mathtt{D}:\{\mathtt{D}\}\})) &= (\{\mathtt{A}\}, \{\mathtt{C}:\{\mathtt{C}\}, \mathtt{B}:\{\mathtt{B}\} \cdot \{\mathtt{A},\mathtt{B}\}\}) \\ \mathrm{pop}_{\mathtt{B}}(\{\mathtt{A}\}, \{\mathtt{B}:\{\mathtt{B}\}, \mathtt{C}:\{\mathtt{C}\}\}) &= (\{\mathtt{A},\mathtt{B}\}, \{\mathtt{C}:\{\mathtt{C}\}\}) \end{aligned}$$

Observe that push ignores the pending knowledge $\mathcal{Q}'$ of the second parameter. The formal definition of these operators is given in the appendix [17].

Definition 7.2 (Knowledge LTS). *An LTS* $L = (\Sigma \times \Delta, \Lambda, \rightarrow, k_0^\delta)$ *is a* knowledge LTS *(abbreviated KLTS) iff:*

1. Σ *is a set of total mappings from participants names to their respective knowledge (i.e.,* $(\mathcal{A}, \mathcal{Q}$ *pairs by Definition 7.1);*
2. Δ *is any set, called* decorating set*, with elements ranged over by* δ, δ'*, etc. For brevity, we write* k^δ *instead of* $(k, \delta) \in \Sigma \times \Delta$*;*
3. k_0^δ *is the* initial state*, such that for all* $\mathtt{A} \in \mathbb{P} : k_0(\mathtt{A}) = (\{\mathtt{A}\}, \varnothing)$*;*
4. Λ *contains* labels *of the form* $\mathtt{B!A}$, $\mathtt{A?B}$, τ*;*
5. *the* labelled transition relation $\rightarrow \ \subseteq \ (\Sigma \times \Delta) \times \Lambda \times (\Sigma \times \Delta)$ *has the following properties, for all* $k_1^\delta, k_2^{\delta'} \in \Sigma \times \Delta$*:*
   - *(a)* $k_1^\delta \xrightarrow{\mathtt{B!A}} k_2^{\delta'} \implies k_2 = k_1\{^{\mathrm{push}_{\mathtt{B}}(k_1(\mathtt{A}),k_1(\mathtt{B}))}/_{\mathtt{A}}\}$*;*
   - *(b)* $k_1^\delta \xrightarrow{\mathtt{A?B}} k_2^{\delta'} \implies k_2 = k_1\{^{\mathrm{pop}_{\mathtt{B}}(k_1(\mathtt{A}))}/_{\mathtt{A}}\}$*;*
   - *(c)* $k_1^\delta \xrightarrow{\tau} k_2^{\delta'} \implies k_1 = k_2$*.*

Item 1 says that each state of a KLTS maps a participant to its current knowledge, e.g., $k^\delta(\mathtt{A}) = (\{\mathtt{A},\mathtt{B}\}, \{\mathtt{D}:\{\mathtt{C},\mathtt{D}\}\})$. The decorations in item 2 allows to distinguish states with a same knowledge mapping. By item 3, each participant has only its own knowledge in the initial state. The transition labels in 4 represent either knowledge transmission ($\mathtt{B!A}$), knowledge acquisition ($\mathtt{A?B}$), or and silent moves (τ). Transitions and knowledge mapping are related as follows

- by (5a), when $\mathtt{B}$ sends its knowledge to $\mathtt{A}$, $\mathtt{A}$'s knowledge is updated by pushing $\mathtt{B}$'s knowledge to the corresponding (pending) knowledge queue.

- by (5b), when $\mathtt{A}$ acquires the knowledge from $\mathtt{B}$, $\mathtt{A}$'s knowledge is updated by dequeueing the first element of the knowledge transmitted by $\mathtt{B}$;
- by (5c), silent moves have no effects on $k(\_)$.

Hereafter, we often write $k, k_1, \ldots$ for KLTS states (omitting decorations δ when unimportant).

Definition 7.3 (Knowledge sharing). *Let $L = (\Sigma, \Lambda, \to, k_0)$ be a KLTS. We say that a state $k^\delta \in \Sigma \times \Delta$* shares knowledge among participants $\mathcal{A} \subseteq \mathbb{P}$ *iff, for all $\mathtt{B} \in \mathcal{A}$, $\exists \mathcal{A}'$ s.t. $k(\mathtt{B}) = (\mathcal{A}', \Omega)$ and $\mathcal{A} \subseteq \mathcal{A}'$.*
We say that L shares knowledge among $\mathcal{A}$ *iff, whenever $k_0^\delta \to^* k_1^{\delta'}$, then there exists $k_2^{\delta''}$ such that $k_1^{\delta'} \to^* k_2^{\delta''}$ and $k_2^{\delta''}$ shares knowledge among $\mathcal{A}$.*

A KLTS state k^δ shares knowledge among a set of participants $\mathcal{A}$ when, in this state, each participant has acquired the knowledge of everybody else in $\mathcal{A}$. L shares knowledge among $\mathcal{A}$ iff *all* traces in a KLTS L eventually reach a state that shares knowledge among $\mathcal{A}$.

Example 7.4. *Let L_1 be a KLTS with a unique trace $k_0 \xrightarrow{\mathtt{A!B}} k_1 \xrightarrow{\mathtt{B?A}} k_2$: L_1 does* not *share knowledge among $\mathtt{A}$ and $\mathtt{B}$. Let L_2 be a KLTS with a unique trace $k_0 \xrightarrow{\mathtt{A!B}} k_1 \xrightarrow{\mathtt{B?A}} k_2 \xrightarrow{\mathtt{B!A}} k_3 \xrightarrow{\mathtt{A?B}} k_4$: L_2 shares knowledge among $\mathtt{A}$ and $\mathtt{B}$.*

7.2. Knowledge sharing in contracts and CO_2

A KLTS L^T can be easily induced from the semantics of a systems of contracts T. Given a system of contracts T, whenever $\mathtt{A} \in \mathcal{P}(T)$ receives the special "knowledge message" k from $\mathtt{B} \in \mathcal{P}(T)$, the knowledge of $\mathtt{A}$ is augmented with that of $\mathtt{B}$. Subsequently, if $\mathtt{A}$ sends k to $\mathtt{C} \in \mathcal{P}(T)$, then $\mathtt{C}$ will receive the knowledge of both $\mathtt{A}$ and $\mathtt{B}$. Technically, each time k is sent (resp. received) by a participant in T, a corresponding transition is induced in L^T, so that knowledge is transmitted (resp. acquired). Instead, if T reduces by exchanging a message sort $\mathsf{e} \neq \mathsf{k}$, then a τ transition is induced in L^T. Each configuration of the system of contracts is used as decoration in L^T, so that the LTS of T and L^T are isomorphic.

Example 7.5. *Let $T_1 = \mathtt{A}\langle \mathtt{B!k} \rangle \mid \mathtt{B}\langle \mathtt{A?k} \rangle$. L^{T_1} does* not *share knowledge among $\mathtt{A}$ and $\mathtt{B}$, and thus T_1 does* not *share knowledge among $\mathtt{A}$ and $\mathtt{B}$. Instead, $T_2 = \mathtt{A}\langle \mathtt{B!k;B?k} \rangle \mid \mathtt{B}\langle \mathtt{A?k;A!k} \rangle$ shares knowledge among $\mathtt{A}$ and $\mathtt{B}$.*

Analogously, a KLTS L^S can be induced from the semantics of a CO_2 system S. Intuitively, a participant $\mathtt{A}$ in S transmits (resp. acquires) its knowledge to (resp. from) $\mathtt{B}$ only when $\mathtt{A}$ fires a $\mathsf{do}^s_{\mathtt{B}}\,\mathsf{k}$ prefix on some session $s[T]$, where $\mathtt{A}$'s contract in T sends (resp. receives) k accordingly. In this case, each state of L^S is decorated with its corresponding CO_2 system so to keep both the KLTS and the semantic LTS of S isomorphic.

In the example below, we show how two participants $\mathtt{A}$ and $\mathtt{B}$ share knowledge by interacting via two sessions — even though none of systems of contracts in these sessions share knowledge among $\{\mathtt{A}, \mathtt{B}\}$.

Example 7.6 (Sharing knowledge via two sessions). *Consider the CO_2 system:*

$$S_1 \;=\; \mathtt{A}[(x,y)\mathsf{tell}_{\mathsf{K}}{\downarrow_x}\, c_{\mathtt{A1}} \mid \mathsf{tell}_{\mathsf{K}}{\downarrow_y}\, c_{\mathtt{A2}} \mid \mathsf{do}^x_{\mathtt{B}}\,\mathsf{k} \,.\, \mathsf{do}^y_{\mathtt{B}}\,\mathsf{k}] \mid \mathtt{B}[(w,z)\mathsf{tell}_{\mathsf{K}}{\downarrow_w}\, c_{\mathtt{B1}} \mid \mathsf{tell}_{\mathsf{K}}{\downarrow_z}\, c_{\mathtt{B2}} \mid \mathsf{do}^w_{\mathtt{A}}\,\mathsf{k} \,.\, \mathsf{do}^z_{\mathtt{A}}\,\mathsf{k}] \mid \mathtt{K}[\mathsf{fuse} \,.\, \mathsf{fuse}]$$

with contracts $c_{\mathtt{A1}} = \mathtt{B!k}$, $c_{\mathtt{A2}} = \mathtt{B?k}$, $c_{\mathtt{B1}} = \mathtt{A?k}$, $c_{\mathtt{B2}} = \mathtt{A!k}$. The system S_1 reduces as follows:

$$\begin{aligned} S_1 \to^* \;& (x,y,w,z)\big(\mathtt{A}[\mathsf{do}^x_{\mathtt{B}}\,\mathsf{k} \,.\, \mathsf{do}^y_{\mathtt{B}}\,\mathsf{k}] \mid \mathtt{B}[\mathsf{do}^w_{\mathtt{A}}\,\mathsf{k} \,.\, \mathsf{do}^z_{\mathtt{A}}\,\mathsf{k}] \\ & \qquad \mid \mathtt{K}[{\downarrow_x}\,\mathtt{A}\;\mathit{says}\;c_{\mathtt{A1}} \mid {\downarrow_w}\,\mathtt{B}\;\mathit{says}\;c_{\mathtt{B1}}] \mid \mathtt{K}[{\downarrow_y}\,\mathtt{A}\;\mathit{says}\;c_{\mathtt{A2}} \mid {\downarrow_z}\,\mathtt{B}\;\mathit{says}\;c_{\mathtt{B2}}] \mid \mathtt{K}[\mathsf{fuse} \,.\, \mathsf{fuse}]\big) \\ \xrightarrow{\mathtt{K}:\,\mathsf{fuse}}\xrightarrow{\mathtt{K}:\,\mathsf{fuse}} \;& S' = (s,t)\mathtt{A}\big[\mathsf{do}^s_{\mathtt{B}}\,\mathsf{k} \,.\, \mathsf{do}^t_{\mathtt{B}}\,\mathsf{k}\big] \mid \mathtt{B}\big[\mathsf{do}^s_{\mathtt{A}}\,\mathsf{k} \,.\, \mathsf{do}^t_{\mathtt{A}}\,\mathsf{k}\big] \mid \mathtt{K}[\mathbf{0}] \mid s[\mathtt{A}\langle \mathtt{B!k} \rangle \mid \mathtt{B}\langle \mathtt{A?k} \rangle] \mid t[\mathtt{A}\langle \mathtt{B?k} \rangle \mid \mathtt{B}\langle \mathtt{A!k} \rangle] \end{aligned}$$

Participants $\mathtt{A}$ and $\mathtt{B}$ share knowledge in S' since they both exchange two k messages: in session s, $\mathtt{A}$ sends its knowledge to $\mathtt{B}$, while in session t, $\mathtt{B}$ sends its knowledge to $\mathtt{A}$.

Note however that neither $\mathtt{A}$ *nor* $\mathtt{B}$ *is honest: they both perform consecutive* do *actions on different sessions, and the second prefix may become unreachable if the first one does not succeed. For instance,* $\mathtt{B}$ *will become persistently culpable if the process of* $\mathtt{A}$ *in* S *is replaced by:*

$$\mathtt{A}[(y)\mathsf{tell}_{\mathtt{K}}{\downarrow_y}\, c_{\mathtt{A2}} \mid \mathsf{do}^{y}_{\mathtt{B}}\, \mathsf{k}] \qquad \textit{or} \qquad \mathtt{A}[(x,y)\mathsf{tell}_{\mathtt{K}}{\downarrow_x}\, c_{\mathtt{A1}} \mid \mathsf{tell}_{\mathtt{K}}{\downarrow_y}\, c_{\mathtt{A2}} \mid \mathsf{do}^{y}_{\mathtt{B}}\, \mathsf{k}]$$

i.e., $\mathtt{A}$ *does not advertise* $c_{\mathtt{A1}}$*, or it lacks the expected* $\mathsf{do}^{x}_{\mathtt{B}}\,\mathsf{k}$ *prefix. In our terminology, there are* CO_2 *systems (those in* (7.6) *above) such that* $\mathtt{B}$ *is not ready once it is composed with them.*

The lack of honesty can be addressed by changing $\mathtt{A}$ *and* $\mathtt{B}$*'s processes so that they execute their* do *actions in parallel, e.g.,* $\mathtt{A}$ *and* $\mathtt{B}$ *become*

$$\mathtt{A}[(x,y)\mathsf{tell}_{\mathtt{K}}{\downarrow_x}\, c_{\mathtt{A1}} \mid \mathsf{tell}_{\mathtt{K}}{\downarrow_y}\, c_{\mathtt{A2}} \mid \mathsf{do}^{x}_{\mathtt{B}}\, \mathsf{k} \mid \mathsf{do}^{y}_{\mathtt{B}}\, \mathsf{k}] \qquad \mathtt{B}[(w,z)\mathsf{tell}_{\mathtt{K}}{\downarrow_w}\, c_{\mathtt{B1}} \mid \mathsf{tell}_{\mathtt{K}}{\downarrow_z}\, c_{\mathtt{B2}} \mid \mathsf{do}^{w}_{\mathtt{A}}\, \mathsf{k} \mid \mathsf{do}^{z}_{\mathtt{A}}\, \mathsf{k}]$$

Example 7.6 shows that it is possible to achieve knowledge sharing through multiple sessions. If a broker were to guarantee knowledge sharing via, e.g., two sessions as above, it would have to wait for both sessions to be ready before starting them, thus ensuring that both session, as a whole, will implement a sharing knowledge protocol. In fact, this implies that these two sessions are not independent since they can only be started together, and, as discussed above, guaranteeing safety and liveness properties for inter-dependent sessions is not trivial.

Additionally, for a set of n *honest* participants to implement a sharing knowledge protocol via binary sessions, $n \times (n-1)$ sessions would be necessary. Multiparty sessions offer a more elegant solution here, as we show in Example 7.7 below.

Example 7.7. *Consider the* CO_2 *system:*

$$S_3 \quad = \quad \mathtt{A}[(x,y)\mathsf{tell}_{\mathtt{K}}{\downarrow_x}\, c_{\mathtt{A}} \mid \mathsf{do}^{x}_{\mathtt{B}}\, \mathsf{k} \,.\, \mathsf{do}^{y}_{\mathtt{B}}\, \mathsf{k}] \mid \mathtt{B}[(w,z)\mathsf{tell}_{\mathtt{K}}{\downarrow_w}\, c_{\mathtt{B}} \mid \mathsf{do}^{w}_{\mathtt{A}}\, \mathsf{k} \,.\, \mathsf{do}^{z}_{\mathtt{A}}\, \mathsf{k}] \mid \mathtt{K}[\mathsf{fuse}]$$

with contracts $c_{\mathtt{A}} = \mathtt{B?k;B!k}$ *and* $c_{\mathtt{B}} = \mathtt{A!k;A?k}$*. In this case, both* $\mathtt{A}$ *and* $\mathtt{B}$ *are honest and share knowledge in a single session.*

7.3. Knowledge sharing choreographies

Our contract model allows for system of contracts that have infinite traces (due to the use of FIFO queues, notably). It is therefore crucial to have an effective and sound procedure to decide whether a given system T shares knowledge among $\mathcal{P}(T)$. Definition 7.8 below gives such a procedure, by analysing a global type $\mathcal{G}$.

We first introduce a map $\mathtt{K} : \mathbb{P} \to 2^{\mathbb{P}}$ that, given a participant name $\mathtt{C}$, returns the set of names representing the participants from whom $\mathtt{C}$ acquired knowledge. Initially, we have that $\mathtt{K}(\mathtt{C}) = \{\mathtt{C}\}$ for each participant $\mathtt{C}$, i.e., a participant has only access to its own knowledge. Given a global type $\mathcal{G}$, we want that

$$\forall\, \mathtt{C} \in \mathcal{P}(\mathcal{G}) \colon \mathtt{K}(\mathtt{C}) = \mathcal{P}(\mathcal{G})$$

holds at the end of the session embodied by $\mathcal{G}$.

We define the boolean function $\hat{\mathtt{K}}(\mathcal{G},N,\mathtt{K})$ which returns true if all the participants in $N \supseteq \mathcal{P}(\mathcal{G})$ share the same knowledge (with respect to $\mathtt{K}$) at the end of the session.

Definition 7.8. *Let* $\hat{\mathtt{K}}(\mathcal{G}) \stackrel{def}{=} \hat{\mathtt{K}}(\mathcal{G},\mathcal{P}(\mathcal{G}),\mathtt{K}_0)$*, where* $\mathtt{K}_0$ *is the initial knowledge map, i.e.* $\forall\, \mathtt{C} \in \mathcal{P}(\mathcal{G}) \colon \mathtt{K}_0(\mathtt{C}) = \{\mathtt{C}\}$ *and*

$$\hat{\mathtt{K}}(\mathcal{G},N,\mathtt{K}) \stackrel{def}{=} \begin{cases} \hat{\mathtt{K}}(\mathcal{G}',N,\mathtt{K}') & \textit{if } \mathcal{G} = \mathtt{A}\to\mathtt{B}:\mathtt{k}\,;\mathcal{G}' \ \wedge\ \mathtt{K}'=\mathtt{K}\{^{\mathtt{K}(\mathtt{B})\cup\mathtt{K}(\mathtt{A})}/_{\mathtt{B}}\} \\ \hat{\mathtt{K}}(\mathcal{G}',N,\mathtt{K}) & \textit{if } \mathcal{G} = \mathtt{A}\to\mathtt{B}:\mathtt{e}\,;\mathcal{G}' \ \wedge\ \mathtt{e}\neq\mathtt{k} \\ \hat{\mathtt{K}}(\mathcal{G}_1,N,\mathtt{K}) \wedge \hat{\mathtt{K}}(\mathcal{G}_2,N,\mathtt{K}) & \textit{if } \mathcal{G} = \mathcal{G}_1 + \mathcal{G}_2 \\ \hat{\mathtt{K}}(\mathcal{G}_1,N,\mathtt{K}_1) \wedge \hat{\mathtt{K}}(\mathcal{G}_2,N,\mathtt{K}_2) & \textit{if } \mathcal{G} = \mathcal{G}_1 \mid \mathcal{G}_2 \ \wedge\ \mathtt{K} = \mathtt{K}_1 \sqcup \mathtt{K}_2 \\ & \quad \textit{and } \mathcal{P}(\mathcal{G}_i) \subseteq \mathrm{dom}(\mathtt{K}_i) \textit{ with } i \in \{1,2\} \\ \forall \mathtt{C} \in \mathrm{dom}(\mathtt{K}) \colon \mathtt{K}(\mathtt{C}) = N & \textit{if } \mathcal{G} = \mathbf{0} \\ \boldsymbol{false} & \textit{otherwise} \end{cases}$$

We write $\sqcup$ *for the disjoint union of maps.*

The first case of Definition 7.8 handles knowledge sharing interactions: $\mathtt{B}$ acquires all the current knowledge of $\mathtt{A}$, while the second case deals with interactions that do not represent a transmission of knowledge. The case of choice simply requires that the condition is satisfied in both branches. In the case of concurrent branches, we need to partition the knowledge map in two – since the sets of participants in the concurrent branches of $\mathcal{G}$ are disjoint. In the case of $\mathcal{G} = \mathbf{0}$, we check that all participants share the same knowledge. For the sake of simplicity, the function returns **false** if the global type is recursive.

We write fuse_K for the variant of fuse^{ϕ} (as introduced in Section 4.5) where $\phi(\mathcal{G}) \iff \hat{\mathtt{K}}(\mathcal{G})$. We illustrate Definition 7.8 in Example 7.10 below.

The proposition below (whose proof is in the appendix [17]) formalises the relationship between *local* and *global* knowledge sharing.

Proposition 7.9. *Let T by a system of contracts such that $\vdash T \blacktriangleright \mathcal{G}$. If $\hat{\mathtt{K}}(\mathcal{G})$, then T shares knowledge among $\mathcal{P}(T)$.*

We conclude this case study with an example that combine: (i) multiparty sessions, (ii) honest participants, and (iii) contractual agreements based on fuse_K.

Example 7.10. *Consider the following contracts*

$$\begin{array}{rcl} c_{\mathtt{A}} & = & \mathtt{B}_1?\mathtt{k};\mathtt{B}_2?\mathtt{k};\mathtt{B}_3?\mathtt{k};\ \mathtt{B}_1!\mathtt{k};\mathtt{B}_2!\mathtt{k};\mathtt{B}_3!\mathtt{k} \\ c_{\mathtt{B}_i} & = & \mathtt{A}!\mathtt{k};\mathtt{A}?\mathtt{k} \qquad \mathtt{i} \in \{1,2,3\} \end{array}$$

Let $T = \mathtt{A}\langle c_{\mathtt{A}}\rangle \mid \mathtt{B}_1\langle c_{\mathtt{B}_1}\rangle \mid \mathtt{B}_2\langle c_{\mathtt{B}_2}\rangle \mid \mathtt{B}_3\langle c_{\mathtt{B}_3}\rangle$, *we have* $T \vdash \mathcal{G} \blacktriangleright$, *with*

$$\begin{array}{rcl} \mathcal{G} & = & \mathtt{B}_1 \to \mathtt{A}:\mathtt{k}\,;\ \mathtt{B}_2 \to \mathtt{A}:\mathtt{k}\,;\ \mathtt{B}_3 \to \mathtt{A}:\mathtt{k}\,; \\ & & \mathtt{A} \to \mathtt{B}_1:\mathtt{k}\,;\ \mathtt{A} \to \mathtt{B}_2:\mathtt{k}\,;\ \mathtt{A} \to \mathtt{B}_3:\mathtt{k}\,;\mathbf{0} \end{array}$$

$\hat{\mathtt{K}}(\mathcal{G})$ *holds since at the end of the first line of* $\mathcal{G}$, *we have* $\mathtt{K}(\mathtt{A}) = \{\mathtt{A},\mathtt{B}_1,\mathtt{B}_2,\mathtt{B}_3\}$ *and, in the second line,* $\mathtt{A}$ *shares its acquired knowledge with all the other participants.*

Consider the CO₂ system below which uses the contracts above, where $\mathtt{K}$ *is a "knowledge sharing" broker using primitive* fuse_K*:*

$$S \;=\; \mathtt{A}[P_{\mathtt{A}}] \mid \mathtt{B}_1[P_{\mathtt{B}_1}] \mid \mathtt{B}_2[P_{\mathtt{B}_2}] \mid \mathtt{B}_3[P_{\mathtt{B}_3}] \mid \mathtt{K}[\mathsf{fuse}_K]$$

with

$$\begin{array}{rcl} P_{\mathtt{A}} & = & (w)\,(\mathsf{tell}_{\mathtt{K}}{\downarrow_w}\ c_{\mathtt{A}}\ .\ \mathsf{do}^w_{\mathtt{B}_1}\,\mathtt{k}\ .\ \mathsf{do}^w_{\mathtt{B}_2}\,\mathtt{k}\ .\ \mathsf{do}^w_{\mathtt{B}_3}\,\mathtt{k}\ .\ \mathsf{do}^w_{\mathtt{B}_1}\,\mathtt{k}\ .\ \mathsf{do}^w_{\mathtt{B}_2}\,\mathtt{k}\ .\ \mathsf{do}^w_{\mathtt{B}_3}\,\mathtt{k}) \\ P_{\mathtt{B}_i} & = & (x)\,(\mathsf{tell}_{\mathtt{K}}{\downarrow_x}\ c_{\mathtt{B}_i}\ .\ \mathsf{do}^x_{\mathtt{a}}\,\mathtt{k}\ .\ \mathsf{do}^x_{\mathtt{a}}\,\mathtt{k}) \qquad \mathtt{i} \in \{1,2,3\} \end{array}$$

Each of the participants above is honest and $\mathtt{K}$ *will only start knowledge sharing sessions.*

vi

8. Related Work

8.1. CO₂ and Verification of Honesty

The origin of CO_2 goes back to [10], where a first calculus of contracting processes was introduced. The calculus was then generalised in [22, 14] to suit different contract models (e.g., contracts as processes or logic formulæ). In [11], it has been instantiated to a theory of bilateral contracts inspired by [23]. The main differences between our adaptation of CO_2 and the original presentation in [22] lie in the specification of session establishment and the introduction of participant variables.

The notion of honesty has been introduced in [11], where a participant is considered honest when, in all possible contexts, it can always exculpate itself by a sequence of its own moves. Here, instead, we require that the participant is *ready* (Definition 5.5) in all possible contexts, similarly to [18]. The two notions are equivalent in the contract model based on binary session types considered in [18]. In the present paper, a similar correspondence is highlighted by Theorem 6.1, which states that an honest participant $\mathtt{A}$ who is

culpable at some session s can always fire some do_s after a sequence of its own moves. Note however that $\mathtt{A}$ may still be culpable after such do_s, e.g. when its contract requires to perform more outputs.

A negative result of [11] is that honesty is not decidable: hence, verification techniques can only aim at safely approximating it. Two different approaches have been proposed in [18, 24], where CO_2 is instantiated with *binary* session types. In [18], a type system for CO_2 processes is presented, associating behavioural types to all session variables used in a processe; honesty is then verified by model checking these behavioural types. The technique introduced in [24] is rather different, and more closely based on the semantic definition of honesty: it first devises an abstraction of the contexts wherein $\mathtt{A}[P]$ may run, and then safely approximates the honesty of $\mathtt{A}$ by model checking such an abstraction. Basically, the abstraction only maintains the process of $\mathtt{A}$ and its contracts; the rest of the system is discarded, by over-approximating its moves; then, model checking is used to search the abstracted state space of $\mathtt{A}[P]$ for states where $\mathtt{A}$ is *not* ready. If the search fails, then $\mathtt{A}$ is honest. Such a search for non-ready states is decidable for finite state processes, i.e. those without delimitation/parallel under process definitions.

We expect that both techniques above can be adapted to cope with the contracts and the calculus studied in this paper. In particular, we now sketch one way to adapt the model checking technique of [24] to the present work.

- First, we develop a safe approximation of honesty, called *1-bounded honesty*. We begin by observing that for all synthesizable T, the culpability states of participants along the reductions of T can be faithfully captured by only considering "1-bounded" configurations (i.e., where each queue contains at most one message). Furthermore, the local and global types in the present work are a subset of those in [4]. Therefore, they enjoy a key property from [4]: for all synthesizable T, each reachable 1-bounded configurations is reachable via a "1-bounded" computation (i.e., an execution such that *each step* has 1-bounded queues). We can then define "1-bounded" honesty as in Definition 5.8, but using the 1-bounded contract semantics, instead of the unbounded one; the observations above can be used to prove that 1-bounded honesty implies honesty. Note that the 1-bounded contract semantics is finite-state.

- Second, we devise an abstraction of 1-bounded systems of contracts wrt. a participant $\mathtt{A}$, whose honesty we want to verify. This abstraction must preserve all the moves of $\mathtt{A}$, while over-approximating the moves of other possible participants. This is done similarly to [24]. Such an abstraction also has a finite-state semantics.

- Finally, we design an abstraction for CO_2 systems which preserves the moves of a participant $\mathtt{A}$ while over-approximating those of the context (similarly to the abstraction for contracts). Again, this abstraction can be obtained by slightly adapting the one in [24], using the abstract contracts from the previous item. Given any $\mathtt{A}[P]$ with no delimitation/parallel under process definitions, such an abstraction has a finite-state semantics.

Summing up, we have a finite-state abstraction of $\mathtt{A}[P]$ that, as in [24], can be searched for states where $\mathtt{A}$ is *not* ready. If the search fails, we say that $\mathtt{A}[P]$ is *abstractly honest*. From the observations above, abstract honesty implies 1-bounded honesty, which in turn implies honesty.

8.2. Choreographies and Compliance among Contracts

Several papers have addressed the problem of defining notions of *compliance* (also called *agreement*, or *compatibility*) among contracts, both in the biparty and in the multiparty case. Very roughly, compliance relates contracts that, when composed, give rise to "correct" interactions.

Our notion of compliance (Definition 3.6) exploits the approach introduced in [6, 16], which gives a typing system allowing to synthesise a choreography (i.e., a global type) from a set of local types, and gives a characterisation of the global types that can be synthesised. To the best of our knowledge, our approach is the first formal model of distributed systems where contractual agreement is based on the existence of a choreography. Deniélou and Yoshida [4] have given a characterisation of finite-state communicating machines from which it is possible to synthesise a global type, as well as an algorithm to build global types

from such machines. Their results could also be used as a foundation for an agreement relation similar the one presented here. However, we remark that the global type obtained from the algorithm in [4] is *not* unique in general, notably due to the lack of special treatment of independent interactions (i.e., $\mathcal{G} \mid \mathcal{G}'$, in our notation). Practically, this means that specific fuse relations (such as the ones discussed in Section 4.3) based on this algorithm may become non-deterministic.

In [25] contracts are considered compliant when their composition may always reach success (so guaranteeing progress until success). In our work, progress is provided by the synthesis of a global type. In [26], compliance requires that the composition of contracts adheres to a predetermined choreography; in our framework, however, no choreography is assumed beforehand. In [27], a contract language has been proposed for processes, where, as in the π-calculus, communications are synchronous and channel names can be passed around. The notion of compliance in [27] is based on a testing approach: contracts are compliant if, essentially, their composition has progress. In the same line of work, [28] proposes to "project" processes into session types (akin to the contracts of [27] but without name passing capabilities). The author gives an adapted notion of compliance, called completeness, which permits to characterise when a set of session types is able to reduce further. The semantic correspondence between global specifications (choreographies) and local specification (local types) has been investigated in [29, 12]. In the context of *binary* session types, several notions of *duality* (i.e., compliance between two complementary session types) are studied and compared in [30], including the higher-order case. A syntax-independent notion of compliance, defined over generic LTSs with input/output actions, and encompassing first-order binary session types with either synchronous or asynchronous semantics, is introduced in [31].

Choreographies have been studied with different types of syntax and several well-formedness conditions (ensuring properties of their projections). The syntax of our choreographies is close to the ones used in [12, 3, 32] and satisfy similar properties such as knowledge of choice (or local choice); other formalisms include automata-based syntax (e.g., [33]), Petri nets [34] and graphical representation such as BPMN [35]. Each formalism generally comes with a set of conditions ensuring good properties on the system specified by the choreography — such as realisability [36] or connectedness [29] which ensure a correspondence between the order of messages exchanged by the projections and the order specified by the choreography. In the present work, we only require that a choreography specifies a *safe* system of contracts (which is guaranteed by the synthesis).

8.3. Conformance between Processes and Contracts

The seminal top-down approach of multiparty session types was first presented in [3]. The methodology it advocates is as follows: a team of programmers designs a global view of the interactions to be implemented (i.e., a choreography or global type), then the choreography is *projected* onto each role, or participant, so to obtain local types; finally each program implementing one or more roles in the choreography is *type-checked* against its corresponding projection(s). The approach permits, if the choreography is "well-formed", to guarantee safety properties such as deadlock freedom, and to ensure that an implementation abides by the specified protocol (session fidelity).

As in most works on session types (e.g., [3, 32, 12]), our contract model does not feature concurrency at the local level, i.e., it does not allow for a multi-threaded single contract. However, our results could be easily extended to support local concurrency, by using a choreography synthesis framework which does cater for it: for instance, the one in [37], which offers the same safety guarantees as the one in this work.

Honesty ensures that a CO_2 process will interact in a session according to some specific contract. A similar problem is tackled in the session types literature. However, session type based systems do not cater for a notion of "culpability", and are generally unable to guarantee progress for processes interacting on multiple interleaved sessions without strong assumptions on their execution context. The problem of guaranteeing progress in a composition of processes interacting through several interleaved sessions is tackled in [19, 32] via a complex typing system. Their results include that a well-typed compositions of processes can interact on multiple interleaved sessions without deadlocking — and thus, in our setting, without remaining persistently culpable. In the present paper, a similar progress result is provided for systems of honest participants (Corollary 6.3): a main difference wrt. [19, 32] is that the honesty of each participant is checked "in isolation", without assumptions on the runtime context.

Another related approach in the context of multiparty session types is that of run-time *monitoring* [38, 39]. Basically, each monitor is constructed from a local type, and mediates the inputs/outputs of a process in a multiparty session; run-time safety and session fidelity properties are enforced by forbidding interactions not specified in the originating local type. This approach is similar to that of CO_2 in the sense that local types are used to "guard" the processes implementing the protocols, as required by the [CO_2-Do] rule premises (Figure 4). The framework in [38], however, requires a global type to be available beforehand in order to project process monitors — while in our case, global types are synthesised dynamically from the local types advertised by each participant.

9. Concluding Remarks

In this work, we investigated the combination of the contract-oriented calculus CO_2 with a contract model that fulfils two basic design requirements: (*i*) it supports multiparty agreements, and (*ii*) it provides an explicit description of the choreography that embodies each agreement. These requirements prompted us towards the well-established results from the session types setting — in particular, regarding the interplay between a well-formed global type and its corresponding local behaviours. We introduced the concepts of global progress and session fidelity in CO_2, also inspired from the analogous concepts in theories based on session types. We built our framework upon a simple version of session types, and yet it turns out to be quite flexible, allowing for sessions where the number of participants is not known beforehand.

This article is the full version of the extended abstract published in [40]. Here we streamlined several aspects of the theory, introduced additional results, and extended definitions, explanations and proofs. In addition, we present a large case study whereby our approach is put into practice.

Future work. We plan to extend our work so to offer even more flexibility — for example, by analysing further parameterised fuse^{ϕ} variants implementing different session establishment criteria. We also plan to study the possibility for a participant to be involved in a session under multiple contracts, e.g., a bank advertising two services, and a customer publishing a contract which uses both of them in a well-formed choreography.

Another research direction is the concept of "group honesty". In fact, the current definition of honesty is quite strict: it basically verifies each participant in isolation, thus providing a sufficient (but not necessary) condition for progress. Consider, for example, a CO_2 system like:

$$S \quad = \quad (x,y)\big(\, \mathtt{A}[\mathsf{tell}_{\mathtt{A}}{\downarrow_x}\ (\mathtt{B}!\mathsf{int} \oplus \mathtt{B}!\mathsf{bool})\ .\ \mathsf{fuse}\ .\ \mathsf{do}^{x}_{\mathtt{B}}\ \mathsf{int}] \ \mid\ \mathtt{B}[\mathsf{tell}_{\mathtt{A}}{\downarrow_y}\ (\mathtt{A}?\mathsf{int} + \mathtt{A}?\mathsf{bool})\ .\ \mathsf{do}^{y}_{\mathtt{A}}\ \mathsf{int}]\,\big)$$

`B` is not honest, since it is not ready for the bool branch of its contract. However, the system S has progress: when `B` establishes a session with `A`, the latter will never take the bool branch; hence, `B` will not remain culpable. This kind of "group honesty" may be used to validate (sub-)systems of participants developed by the same organization: it would ensure that they never "cheat each other", and are collectively honest when deployed in any context. Furthermore, the group honesty of all participants in a system S may turn out to be a necessary condition for the global progress of S.

Acknowledgements. We would like to thank Emilio Tuosto for his (in)valuable advice, and the anonymous reviewers of ICE 2013 for their comments and suggestions.

References

[1] K. Honda, V. T. Vasconcelos, M. Kubo, Language primitives and type discipline for structured communication-based programming, in: Proc. ESOP, 1998, pp. 122–138.
[2] K. Takeuchi, K. Honda, M. Kubo, An interaction-based language and its typing system, in: Proc. PARLE'94 Parallel Architectures and Languages Europe, Vol. 817 of LNCS, Springer, 1994, pp. 398–413. doi:10.1007/3-540-58184-7_118.
[3] K. Honda, N. Yoshida, M. Carbone, Multiparty asynchronous session types, in: Proc. POPL, 2008, pp. 273–284.
[4] P.-M. Deniélou, N. Yoshida, Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types, in: Proc. ICALP, 2013, pp. 174–186.

[5] F. Montesi, N. Yoshida, Compositional choreographies, in: Proc. CONCUR, 2013, pp. 425–439.
[6] J. Lange, E. Tuosto, Synthesising choreographies from local session types, in: Proc. CONCUR, 2012, pp. 225–239.
[7] T. Hobbes, The Leviathan, 1651, chapter XIV.
[8] OASIS, Reference architecture foundation for service oriented architecture, comm. Spec. 01, v.1.0. Available at http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/cs01/soa-ra-v1.0-cs01.html (December 2012).
[9] M. Armbrust, et al., A view of cloud computing, Comm. ACM 53 (4) (2010) 50–58. doi:10.1145/1721654.1721672.
[10] M. Bartoletti, R. Zunino, A calculus of contracting processes, in: Proc. LICS, 2010, pp. 332–341.
[11] M. Bartoletti, E. Tuosto, R. Zunino, On the realizability of contracts in dishonest systems, in: Proc. COORDINATION, 2012, pp. 245–260.
[12] G. Castagna, M. Dezani-Ciancaglini, L. Padovani, On global types and multi-party session, Logical Methods in Comp. Sci. 8 (1).
[13] P.-M. Deniélou, N. Yoshida, Multiparty session types meet communicating automata, in: Proc. ESOP, 2012, pp. 194–213.
[14] M. Bartoletti, E. Tuosto, R. Zunino, Contract-oriented computing in CO_2, Scientific Annals in Comp. Sci. 22 (1) (2012) 5–60.
[15] S. M. Hedetniemi, S. T. Hedetniemi, A. L. Liestman, A survey of gossiping and broadcasting in communication networks, Networks 18 (4) (1988) 319–349.
[16] J. Lange, On the synthesis of choreographies, Ph.D. thesis, University of Leicester (2013).
[17] Appendix to this paper, http://tcs.unica.it/publications.
[18] M. Bartoletti, A. Scalas, E. Tuosto, R. Zunino, Honesty by typing, in: Proc. FMOODS/FORTE, 2013, pp. 305–320.
[19] M. Coppo, M. Dezani-Ciancaglini, L. Padovani, N. Yoshida, Inference of global progress properties for dynamically interleaved multiparty sessions, in: R. De Nicola, C. Julien (Eds.), Proc. COORDINATION, Vol. 7890 of Lecture Notes in Computer Science, Springer, 2013, pp. 45–59.
[20] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, Commun. ACM 53 (4) (2010) 50–58.
[21] D. Brand, P. Zafiropulo, On communicating finite-state machines, JACM 30 (2) (1983) 323–342.
[22] M. Bartoletti, E. Tuosto, R. Zunino, Contracts in distributed systems, in: Proc. ICE, 2011, pp. 130–147.
[23] G. Castagna, N. Gesbert, L. Padovani, A theory of contracts for web services, ACM Trans. on Prog. Lang. and Sys. 31 (5).
[24] M. Bartoletti, M. Murgia, A. Scalas, R. Zunino, Modelling and verifying contract-oriented systems in Maude, in: Rewriting Logic and Its Applications (WRLA), Lecture Notes in Computer Science, Springer, 2014, pp. 130–146. doi:10.1007/978-3-319-12904-4_7.
[25] M. Bravetti, G. Zavattaro, A foundational theory of contracts for multi-party service composition, Fundam. Inform. 89 (4) (2008) 451–478.
[26] M. Bravetti, I. Lanese, G. Zavattaro, Contract-driven implementation of choreographies, in: TGC, Vol. 5474 of LNCS, Springer, 2009, pp. 1–18.
[27] G. Castagna, L. Padovani, Contracts for mobile processes, in: Proc. CONCUR, 2009, pp. 211–228.
[28] L. Padovani, On projecting processes into session types, MSCS 22 (2012) 237–289.
[29] I. Lanese, C. Guidi, F. Montesi, G. Zavattaro, Bridging the gap between interaction- and process-oriented choreographies, in: Proc. SEFM, 2008, pp. 323–332.
[30] G. Bernardi, O. Dardha, S. J. Gay, D. Kouzapas, On duality relations for session types, in: TGC, 2014, to appear.
[31] M. Bartoletti, A. Scalas, R. Zunino, A semantic deconstruction of session types, in: CONCUR, 2014, pp. 402–418. doi:10.1007/978-3-662-44584-6_28.
[32] L. Bettini, M. Coppo, L. D'Antoni, M. D. Luca, M. Dezani-Ciancaglini, N. Yoshida, Global progress in dynamically interleaved multiparty sessions, in: F. van Breugel, M. Chechik (Eds.), CONCUR 2008 - Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19-22, 2008. Proceedings, Vol. 5201 of Lecture Notes in Computer Science, Springer, 2008, pp. 418–433. doi:10.1007/978-3-540-85361-9_33.
URL http://dx.doi.org/10.1007/978-3-540-85361-9_33
[33] X. Fu, T. Bultan, J. Su, Conversation protocols: a formalism for specification and verification of reactive electronic services, Theor. Comput. Sci. 328 (1-2) (2004) 19–37. doi:10.1016/j.tcs.2004.07.004.
URL http://dx.doi.org/10.1016/j.tcs.2004.07.004
[34] L. Fossati, R. Hu, N. Yoshida, Multiparty session nets, in: TGC, 2014, to appear.
[35] Business Process Model and Notation 2.0 Choreography, http://en.bpmn-community.org/tutorials/34/ (2012).
[36] S. Basu, T. Bultan, M. Ouederni, Deciding choreography realizability, in: J. Field, M. Hicks (Eds.), Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012, ACM, 2012, pp. 191–202. doi:10.1145/2103656.2103680.
URL http://doi.acm.org/10.1145/2103656.2103680
[37] J. Lange, E. Tuosto, N. Yoshida, From communicating machines to graphical choreographies, in: POPL, 2015, to appear, available at http://www.doc.ic.ac.uk/research/technicalreports/2014/DTR14-4.pdf.
[38] L. Bocchi, T.-C. Chen, R. Demangeon, K. Honda, N. Yoshida, Monitoring networks through multiparty session types, in: Proc. FMOODS/FORTE, Vol. 7892 of Lecture Notes in Computer Science, Springer, 2013, pp. 50–65.
[39] T.-C. Chen, L. Bocchi, P.-M. Deniélou, K. Honda, N. Yoshida, Asynchronous distributed monitoring for multiparty session enforcement, in: Proc. TGC, Vol. 7173 of Lecture Notes in Computer Science, Springer, 2011, pp. 25–45.
[40] J. Lange, A. Scalas, Choreography synthesis as contract agreement, in: M. Carbone, I. Lanese, A. Lluch-Lafuente, A. Sokolova (Eds.), ICE, Vol. 131 of EPTCS, 2013, pp. 52–67.