Scientific Annals of Computer Science vol. 22 (1), 2012, pp. 5–60
DOI: 10.7561/SACS.2012.1.5

Contract-Oriented Computing in CO_2

Massimo BARTOLETTI[1], Emilio TUOSTO[2], Roberto ZUNINO[3]

Abstract

We present CO_2, a parametric calculus for contract-based computing in distributed systems. By abstracting from the actual contract language, our calculus generalises both the *contracts-as-processes* and *contracts-as-formulae* paradigms. The calculus features primitives for advertising contracts, for reaching agreements, and for querying the fulfilment of contracts. Coordination among participants happens via multi-party sessions, which are created once agreements are reached. We present two instances of our calculus, by modelling contracts as processes in a variant of CCS, and as formulae in a logic. We formally relate the two paradigms, through an encoding from contracts-as-formulae to contracts-as-processes which ensures that the promises deducible in the logical system are exactly those reachable by its encoding as a process. Finally, we present a coarse-grained taxonomy of possible misbehaviours in contract-oriented systems, and we illustrate them with the help of a variety of examples.

Keywords: contracts, concurrent constraint programming, multiparty sessions

1 Introduction

The increasing spread and complexity of distributed computing services is changing the way services are designed, implemented and exploited. A key problem is how to drive safe and fair interactions among distributed

[1]Dipartimento di Matematica e Informatica, Università degli Studi di Cagliari, via Ospedale 72, 09124 Cagliari, Italy. Email: bart@unica.it
[2]Department of Computer Science, University of Leicester, UK
[3]DISI, Università di Trento and COSBI, Italy

participants which are possibly mutually distrusted, and have possibly conflicting individual goals.

Typically, this problem is tackled quite unfairly: service providers fix some terms of usage (Service-Level Agreement, SLA), which have to be accepted by the client before using the service. SLAs are legal documents, written in natural language, normally so long and convoluted that they are accepted by default. If a provider infringes its SLA, clients could legally prosecute the provider, but the potential costs of a legal action discourage clients from taking legal steps, especially when the transaction involves small amounts of money.

Traditionally, SLAs are mostly meant to protect the service provider, while the client has to trust that the service correctly implements the required functionalities. The acceptance of the SLA has the purpose of freeing the service provider from responsibilities in case of hardware and/or software hazards, or to avoid costly or impossible tasks. On the other side, service providers are rather reluctant to propose SLAs which offer strong guarantees to clients. This is quite understandable, since it would require providers to certify that their services actually satisfy the promised properties — which most often is unfeasible. This situation becomes even more complex when the boundary between providers and clients is fuzzy, as it happens for services composed together from other services, where the client of a service may be the provider of another one.

Contract-oriented computing is a software design paradigm where the interaction between clients and services is disciplined by formal entities, named contracts. The life-cycle of a contract-oriented service can be thought as composed of three phases. In the first phase, contracts are used to negotiate the required and offered behaviour. At the end of this phase, contracts may be stipulated: when this happens, the terms of service they prescribe become legally binding. In the last phase, services execute their tasks. While doing that, they may inspect the contracts they have stipulated, to check e.g. what are their current duties, and what has to be expected by the other parties. In this framework, enforcing a contract through a legal authority has to be considered as a last resort. Typical situations would be resolved automatically by the service infrastructure itself, e.g. by providing suitable compensations to the injured party, and by inflicting the right punishment to the offending party.

Contracts have been investigated from a variety of perspectives and using a variety of different formalisms and analysis techniques, ranging from

c-semirings [8, 9, 18], to behavioural types [7, 12, 11, 13], to formulae in suitable logics [1, 4, 25], to categories [10], *etc.* This heterogeneous ecosystem of formalisms makes it difficult to understand the essence of those methods, and how they are related.

As a first step towards a unifying framework for reasoning about contracts we propose a generic calculus for COntract-Oriented computing (in short, CO_2). By abstracting away from the actual contract language, our calculus can encompass a variety of different contract paradigms. We provide a common set of primitives for computing with contracts: they allow for advertising and querying contracts, for reaching agreements, and for fulfilling them with the needed actions. All these primitives are independent from the chosen language of contracts, and they only pivot on some general requirements fixed in the *contract model* proposed here.

Synopsis. Our main contributions are centred around CO_2 which is designed around three main principles.

The first is the separation of concerns between the way contracts are modelled and the way they are used in distributed computations. Indeed, we abstract from the actual contract language by only imposing a few general requirements. In this way, we envisage our calculus as a generic framework which can be tuned by instantiating the contract model to concrete formalisations of contracts. In § 2 we present the abstract contract model, followed by two concretisations: in § 2.1 we adopt the contracts-as-processes paradigm whereby CCS-like processes represent contracts that drive the behaviour of distributed participants; in § 2.2 we embrace instead the contracts-as-formulae paradigm, by instantiating our calculus with contracts expressed in a suitable logic. We relate the two concrete models in § 2.3, first with the help of a few examples, and then by showing that contracts-as-formulae, expressed in a significant fragment of our logic, can be suitably encoded into contracts-as-processes (Theorem 2.7).

The second design principle of our calculus is that its primitives must be reasonably implementable in a distributed setting. To this purpose, we blend in § 3 a few primitives inspired by Concurrent Constraint Programming (CCP [27]) to other primitives inspired by session types [21]. The key notions around which our primitives are conceived are *participants* and *sessions*. The former represent distributed units of computation that can advertise contracts, execute the corresponding operations, and establish/check agreements. Each agreement corresponds to a fresh session, containing rights and

obligations of each stipulating party. Participants use sessions to coordinate with each other and fulfil their obligations. Also, sessions enable us to formulate a general notion of "misbehaviour" which paves the way for automatic verification. We suggest possible variants of our primitives and of the contract model (§ 3.7).

The third design principle of CO_2 is that it must allow for modelling a wide variety of misbehaviours, specific to contract-oriented systems. Remarkably enough, in our approach contracts are *not* supposed to be always respected after they have been stipulated. This motivates the fact that, in CO_2 , contracts are not discharged after they have been used to establish a session among a set of participants, as usually done e.g. in the approaches dealing with behavioural types. In our approach, contracts are also used to drive computations after sessions have been initiated, e.g. to detect if some violation has happened, and to identify the responsibles. In § 3.5 we sketch a concise taxonomy of what can go wrong in contract-oriented systems, and in § 3.6 we present a collection of relevant misbehaviours.

In § 4 we discuss some related work, and in § 5 we conclude. Appendix A contains all the proofs of our statements.

A simple example. We now give an overview of our proposal with the help of a simple example. To do that, we abstract away from the actual language for describing contracts: we just illustrate the main features of CO_2, and how contracts are exploited in our framework.

Consider an on-line marketplace where sellers advertise their products, and buyers can shop. In CO_2 we can specify a system composed by a seller A, a buyer B, and a marketplace M as follows:

$$\mathsf{A}[\cdots] \mid \mathsf{M}[\cdots] \mid \mathsf{B}[\cdots]$$

representing three distributed participants running in parallel. Within the square brackets, the ellipses stand for a process which describes the state and behaviour of each participant.

For instance, assume that the seller first checks that a payment has been made, and then ships some goods to the buyer, while the buyer simply pays. We informally describe the contracts of A and B as follows:

Seller contract:	c_{A} = "I promise to ship after the buyer has paid"
Buyer contract:	c_{B} = "I promise to pay"

The behaviour of A and B can be formalised in CO_2 as:

$$\begin{array}{rrl} \text{Seller behaviour:} & P & = (v)\,\mathsf{tell}_{\mathsf{M}}\downarrow_v c_{\mathsf{A}}.\ \mathsf{ask}_v\,\phi_{paid?}.\ \mathsf{do}_v\,\mathsf{ship} \\ \text{Buyer behaviour:} & Q & = (z)\,\mathsf{tell}_{\mathsf{M}}\downarrow_z c_{\mathsf{B}}.\ \mathsf{do}_z\,\mathsf{pay} \end{array}$$

In their first steps, both seller and buyer advertise their contracts (c_{A} and c_{B}, respectively) to the marketplace M. This is done through the primitive tell. The effect of $\mathsf{tell}_{\mathsf{M}}\downarrow_v c_{\mathsf{A}}$ is to move the *latent contract* $\downarrow_v c_{\mathsf{A}}$ to the environment of M. Once the two tell prefixes have been executed, the system $\mathsf{A}[P] \mid \mathsf{M}[\cdots] \mid \mathsf{B}[Q]$ becomes:

$$(v,z)\Big(\mathsf{A}[\mathsf{ask}_v\,\phi_{paid?}.\ \mathsf{do}_v\,\mathsf{ship}] \ \mid\ \mathsf{M}[\downarrow_v c_{\mathsf{A}}\ \mid\downarrow_z c_{\mathsf{B}} \mid \cdots] \ \mid\ \mathsf{B}[\mathsf{do}_z\,\mathsf{pay}]\Big) \qquad (1)$$

It is important to note that, as long as the contracts are *latent* (symbolically, represented by prefixing c_{A} and c_{B} with $\downarrow$), neither A nor B have any obligations. As typically done in process calculi, the scope of delimiters has been expanded in the system (1).

Let us now turn our attention to the marketplace M. In our scenario, M acts as a brokerage service between sellers and buyers. Indeed, the first step of M is to inspect the latent contracts in its environment, in order to find agreements among participants. This is modelled by the recursive process $X \stackrel{\text{def}}{=} (u)\,\mathsf{fuse}_u\,\phi.\,X$. The prefix $\mathsf{fuse}_u\,\phi$ looks for a set of latent contracts that ensure (at least) a given level of service agreement, described by the condition ϕ. The actual nature of such conditions, and when contracts satisfy them, depend on the contract language adopted (e.g. those described in § 2.1 and § 2.2). Here we assume that ϕ requires "A ships the goods and B pays", which is satisfied by the contracts c_{A} and c_{B}. Once the $\mathsf{fuse}_u\,\phi$ is executed, the system (1) (where the ellipses are replaced by X) evolves to:

$$(s)\Big(\mathsf{A}[\mathsf{ask}_s\,\phi_{paid?}.\ \mathsf{do}_s\,\mathsf{ship}] \ \mid\ \mathsf{M}[X] \ \mid\ \mathsf{B}[\mathsf{do}_s\,\mathsf{pay}] \ \mid\ s[c_{\mathsf{A}} \mid c_{\mathsf{B}}]\Big) \qquad (2)$$

Basically, the fuse has initiated a new session s among A, B and M, and it has moved the contracts c_{A} and c_{B} to the environment of s. Note that in (2) these contracts are no longer latent; also, the variables u, v and z involved in the latent contracts are now instantiated to the session name s. The shared name s allows A and B to coordinate, by accessing the environment of s through the primitives ask and do.

The next step is performed by B, which fires the prefix $\mathsf{do}_s\,\mathsf{pay}$. This makes the contracts in s adjust themselves to reflect the fact that B has

paid. Formally, this is modelled by an LTS on contracts such that $c_{\mathsf{A}} \mid c_{\mathsf{B}} \xrightarrow{\mathsf{B}\ \mathit{does}\ \mathsf{pay}} c$. The contract c represents a situation where A has to ship, while B has paid (and so, he has no further obligations). The system (2) evolves then to:

$$(s)\Big(\mathsf{A}[\mathsf{ask}_s\ \phi_{paid?}.\ \mathsf{do}_s\ \mathsf{ship}]\ \mid\ \mathsf{M}[X]\ \mid\ \mathsf{B}[\mathbf{0}]\ \mid\ s[c]\Big) \tag{3}$$

Assume that the condition $\phi_{paid?}$ requires that the goods have been paid to A. The prefix $\mathsf{ask}_s\ \phi_{paid?}$ in A can now be fired, because the contract c in s satisfies $\phi_{paid?}$. Therefore, the system (3) may evolve to:

$$(s)\Big(\mathsf{A}[\mathsf{do}_s\ \mathsf{ship}]\ \mid\ \mathsf{M}[X]\ \mid\ \mathsf{B}[\mathbf{0}]\ \mid\ s[c]\Big) \tag{4}$$

The seller can now perform her last action, i.e. shipping the goods. Assume that $c \xrightarrow{\mathsf{A}\ \mathit{does}\ \mathsf{ship}} c'$, where the contract c' prescribes no obligations. Then, the system (4) finally evolves to:

$$(s)\Big(\mathsf{A}[\mathbf{0}]\ \mid\ \mathsf{M}[X]\ \mid\ \mathsf{B}[\mathbf{0}]\ \mid\ s[c']\Big)$$

The example above shows some of the main features of CO_2 . However, to keep it small, we have not included in the example some distinguished features, which we briefly discuss below.

First, the primitive fuse allows for agreements which involve an arbitrary number of participants and contracts. Technically, this is obtained by stipulating a minimal set of contracts which satisfy the given condition, and then by opening a multiparty session among all the involved participants.

Second, CO_2 allows for stipulating contracts with multiple levels of compliance. Contracts can expose optional behaviours, and the condition ϕ in $\mathsf{fuse}\ \phi$ ensures a minimal level of "static" agreement. The actual obligations of a participant (contained in the contract) may be discovered later on in the execution, depending on the other contracts involved in the agreement, and possibly also on the actions performed in the session (see e.g. Example 3.4).

Third, participants are never considered trusted in CO_2 , i.e. we do not assume that a stipulated contract is always respected. This makes CO_2 a suitable language for modelling a variety of attacks in contract-oriented applications, as shown e.g. by Examples 3.10, 3.11, 3.6, 3.12, and 3.13. Since it is not possible, in general, to guarantee that the promises in a contract are followed by facts, a minimal safety requirement is that, if some promise has not been maintained, then it is possible to identify the responsibles.

Our framework allows for identifying, at each step of the execution, who is culpable of such violations. Technically, this is done through a relation $c \breve{\smile} \mathsf{A}$ between participants and contracts, which has to be specified in each instance of the abstract contract model. Intuitively, $c \breve{\smile} \mathsf{A}$ means that A has no obligations in the contract c. This allows us to specify (in Definition 3.5) when a participant is *honest* in a given context. Roughly, $\mathsf{A}[P]$ is honest in S when in all fair maximal computations of $\mathsf{A}[P] \mid S$, $c \breve{\smile} \mathsf{A}$ holds true for all contracts c stipulated by A in the system. The property of being honest in *all* possible contexts is a main design goal for contract-oriented systems, because when a participant satisfies it, she is guaranteed that no (honest) judge will ever deem her culpable of violations.

2 An Abstract Contract Model

Before providing a formal definition, we sketch the basic ingredients of a generic contract model.

We start by introducing some preliminary notions and definitions; some of them will only be used later on in § 3. *Participants* are those agents which may advertise contracts, establish agreements, and realise them. *Sessions* are created upon reaching an agreement, and provide the context in which participants can interact to fulfil their contracts. Let $\mathcal{N}$ and $\mathcal{V}$ be countably infinite, disjoint sets of names and variables, respectively. Assume $\mathcal{N}$ partitioned into two infinite sets $\mathcal{N}_P$ and $\mathcal{N}_S$, for names of participants and of sessions, respectively. Similarly, $\mathcal{V}$ is partitioned into two infinite sets $\mathcal{V}_P$ and $\mathcal{V}_S$, for variable identifiers of participants and sessions. A substitution is a partial map σ from $\mathcal{V}$ to $\mathcal{N}$; we write $u \in \operatorname{dom}\sigma$ when σ is defined at u, and require that σ maps $a \in \operatorname{dom}\sigma \cap \mathcal{V}_P$ to $\mathcal{N}_P$ and $s \in \operatorname{dom}\sigma \cap \mathcal{V}_S$ to $\mathcal{N}_S$.

Our main notational conventions are displayed in Table 1.

The first ingredient of our contract model is a set $\mathcal{A}$ of *atoms*. Intuitively, these will be the basic statements contained in contracts, like e.g. pay and ship mentioned in the example in § 1.

The second ingredient is a set $\mathcal{C}$ of *contracts*. We are quite liberal about it: we only require that (i) there exists a set Γ of *unsigned contracts* such that A *says* $\gamma \in \mathcal{C}$ for all participants A and $\gamma \in \Gamma$, and (ii) $c \mid c' \in \mathcal{C}$ for all $c, c' \in \mathcal{C}$. The contract A *says* γ can be thought of as "γ signed by A", while $c \mid c'$ can be thought as a contract comprising both c and c'.

$n, m, \ldots \in \mathcal{N}$	names, union of:
$\mathsf{A}, \mathsf{B}, \ldots \in \mathcal{N}_P$	participant names
$s, t, \ldots \in \mathcal{N}_S$	session names
$\mathcal{V}$	variables, union of:
$a, b, \ldots \in \mathcal{V}_P$	participant variables
$x, y, \ldots \in \mathcal{V}_S$	session variables
$u, v, \ldots \in \mathcal{N} \cup \mathcal{V}$	names or variables
$A, B, \ldots \in \mathcal{N}_P \cup \mathcal{V}_P$	participant names/variables
$\alpha, \alpha' \ldots \in \mathcal{A}$	atoms
$\gamma, \gamma', \ldots \in \Gamma$	unsigned contracts
$c, c', \ldots \in \mathcal{C}$	contracts (of the form A *says* γ)
$c \mid c'$	contract comprising c and c'
$\phi, \psi, \ldots \in \Phi$	observables
$c \xrightarrow{A \text{ does } \alpha}\!\!\!\!\rightarrow\, c'$	LTS on contracts
$A \breve{\smile} c$	A has no obligations in c
$c \vdash \phi$	c entails ϕ

Table 1: Notation

A labelled transition relation $\xrightarrow{A \text{ does } \alpha}\!\!\!\!\rightarrow$ on contracts models their evolution under the actions A *does* α performed by participants.

Two further ingredients are a set Φ of *observables* (properties of contracts) and an entailment relation $\vdash$ between contracts and observables. Note that we keep distinct contracts from observables. This has the same motivations as the traditional distinction between behaviours (systems) and their properties (formulae predicating on behaviours), which brought in plenty of advantages in the design/implementation of systems.

The last ingredient of our contract model is a relation $\breve{\smile}$ between participants and contracts. We write $A \breve{\smile} c$ to mean that the contract c prescribes no obligations to the participant A. This does not mean that A will remain without any obligations while c evolves. Possibly, some obligations will arise when some actions are performed by other participants involved in c. When $A \breve{\smile} c$ is false, we say that A is *culpable* in c. We require that $A \breve{\smile} c$ whenever c has no occurrences of A *says* $\cdots$, i.e. A is not culpable in contracts where she has not made any statement.

Definition 2.1 formalises the above concepts.

Definition 2.1. *A* contract model *is a tuple* $\langle \mathcal{A}, \mathcal{C}, \twoheadrightarrow, \Phi, \vdash, \smile \rangle$ *where*

- $\mathcal{A}$ *is a set of* atoms.
- $\mathcal{C}$ *is a* set of contracts*, such that* (i) $\exists \Gamma.\, \forall A \in \mathcal{N}_P, \gamma \in \Gamma.\, A$ *says* $\gamma \in \mathcal{C}$*,* (ii) $\forall c, c' \in \mathcal{C}.\, c \mid c' \in \mathcal{C}$*, and* (iii) *for some signature* Σ*,* $\mathcal{C}$ *forms a subalgebra of a term-algebra* $T_{\mathcal{V} \cup \mathcal{N}}(\Sigma)$*.*
- $c \xrightarrow{A \text{ does } \alpha} c'$ *is a labelled transition relation over contracts.*
- Φ *is a set of* observables*, forming a subalgebra of a term-algebra* $T_{\mathcal{V} \cup \mathcal{N}}(\Sigma')$ *for some signature* Σ'*.*
- $\vdash$ *is a* contract entailment *relation between* $\mathcal{C}$ *and* Φ*.*
- $\smile$ *is a* contract fulfilment *relation between* $\mathcal{C}$ *and participants, such that* $A \smile c$ *for all* c *without* A *says .*

We illustrate the contract model with the help of an informal example.

Example 2.1. *A seller* A *and a buyer* B *stipulate a contract* c_0*, which binds* A *to ship an item after* B *has paid. Let* pay *be the atom which models the action of paying. The transition* $c_0 \xrightarrow{\mathsf{B} \text{ does } \mathsf{pay}} c_1$ *models the evolution of* c_0 *into a contract* c_1 *where* A *is obliged to ship, while* B *has no more duties. Now, let* ϕ *be the observable "*A *must ship". Then, we would have* $c_0 \not\vdash \phi$*, because* A *does not have to ship anything yet, while* $c_1 \vdash \phi$*, because* B *has paid and so* A *must ship. It would not be the case that* $\mathsf{A} \smile c_1$*, because* B *has paid, while* A *has not yet fulfilled her obligation to ship.*

We remark that the use of term-algebras in Definition 2.1 allows us to smoothly apply variable substitutions to contracts and observables. Accordingly, we assume defined the sets $\mathrm{var}(c)$ and $\mathrm{var}(\phi)$ of variables of contracts and observables. Note that actions are not required to be in $\mathcal{C}$. Including them in $\mathcal{C}$ would allow for recording the history of the past actions *within* contracts, see e.g. § 2.2.

2.1 Contracts as Processes

The first instance of our contract model appeals to the contracts-as-processes paradigm. A contract is represented as a CCS-like process [24], the execution of which dictates obligations to participants.

A contract A *says* γ represents a participant A committed to the behaviour specified by the process γ. A *pending message* $A\rangle\mathsf{a}\rangle B$ represents a message sent on channel a by A, but not received by B yet. A process γ is a (possibly recursive) composition of prefix-guarded sums of processes. The prefixes are the atoms, and they represent silent, input, or output actions. Unlike in the standard CCS, input and output prefixes declare not only the communication port, but also the intended partner. In an output prefix $\overline{\mathsf{a}}\rangle A$, the participant A denotes the receiver of the message, while in an input prefix $B\rangle\mathsf{a}$, B stands for the expected sender.

Definition 2.2. *We define a* contracts-as-processes *language as follows*

- $\mathcal{A}$ *is the union of three disjoint sets: the inputs* $A\rangle\mathsf{a}$*, the outputs* $\overline{\mathsf{a}}\rangle A$*, and the internal activity* τ*, where* a *is a channel name.*
- $\mathcal{C}$ *is the set of process terms generated by the non-terminal* c *(the set of terms* Γ *is generated by* γ*):*

$$\begin{array}{rcl|l|l|l} \alpha & ::= & \tau & A\rangle\mathsf{a} & \overline{\mathsf{a}}\rangle B & \\ \gamma & ::= & \sum_i \alpha_i.\gamma_i & \gamma \mid \gamma & X & \\ c & ::= & 0 & A \text{ says } \gamma & c \mid c & A\rangle\mathsf{a}\rangle B \end{array}$$

  where we assume variables X *to be defined through (prefix-guarded recursive) equations* $X \triangleq \gamma$*. We also denote with* 0 *the empty sum; trailing occurrences of* 0 *may be omitted. The signature* Σ *corresponds to the syntax above.*

- $\twoheadrightarrow$ *is the least relation closed under the rules in Figure 1 and structural equivalence* $\equiv$ *(defined with the usual CCS rules and* A *says* $0 \equiv 0$*).*
- Φ *is a set of terms on a signature* Σ'*, including one sort interpreted on atoms* $\mathcal{A}$ *(e.g. LTL or CTL formulae).*
- $c \vdash \phi$ *is a decidable relation between the LTS of* c *and* $\phi \in \Phi$*.*
- $A \smile c$ *holds iff* $\nexists c', \alpha.\ c \xrightarrow{A \text{ does } \alpha} c'$*.*

We briefly comment on the rules in Figure 1.

The first row defines an LTS $\xrightarrow{\alpha}$ for γ-processes. The rules are quite standard: (Pref) allows for firing an atom, (Par) for making two γ-processes evolve in parallel, and (Def) for expanding a process definition.

$$\alpha.\gamma + \gamma' \xrightarrow{\alpha} \gamma \ \text{(PREF)} \qquad \frac{\gamma_1 \xrightarrow{\alpha} \gamma_1'}{\gamma_1 \mid \gamma_2 \xrightarrow{\alpha} \gamma_1' \mid \gamma_2} \ \text{(PAR)} \qquad \frac{X \stackrel{\text{def}}{=} \gamma \qquad \gamma \xrightarrow{\alpha} \gamma'}{X \xrightarrow{\alpha} \gamma'} \ \text{(DEF)}$$

$$\frac{\gamma \xrightarrow{\overline{\mathsf{a}}\rangle B} \gamma'}{A \ \mathit{says}\ \gamma \ \xrightarrow{A \ \mathit{does}\ \overline{\mathsf{a}}\rangle B}\!\!\!\!\rightarrow \ A \ \mathit{says}\ \gamma' \mid A\rangle\mathsf{a}\rangle B} \ \text{(OUT)}$$

$$\frac{\gamma \xrightarrow{A\rangle\mathsf{a}} \gamma'}{B \ \mathit{says}\ \gamma \ \mid \ A\rangle\mathsf{a}\rangle B \ \xrightarrow{B \ \mathit{does}\ A\rangle\mathsf{a}}\!\!\!\!\rightarrow \ B \ \mathit{says}\ \gamma'} \ \text{(IN)}$$

$$\frac{\gamma \xrightarrow{\tau} \gamma'}{A \ \mathit{says}\ \gamma \ \xrightarrow{A \ \mathit{does}\ \tau}\!\!\!\!\rightarrow \ A \ \mathit{says}\ \gamma'} \ \text{(TAU)} \qquad \frac{c_1 \xrightarrow{A \ \mathit{does}\ \alpha}\!\!\!\!\rightarrow \ c_1'}{c_1 \mid c_2 \ \xrightarrow{A \ \mathit{does}\ \alpha}\!\!\!\!\rightarrow \ c_1' \mid c_2} \ \text{(CPAR)}$$

Figure 1: Labelled transition relation of contract-as-processes

The other rules define the LTS $\twoheadrightarrow$ of contracts, as prescribed by Definition 2.1. The rule (OUT) implements an asynchronous output: when the contract of A is willing to send a message on channel a to B, the *pending message* $A\rangle\mathsf{a}\rangle B$ is put in parallel with the continuation of the contract of A. The rule (IN) allows the participant B to perform an input, when a suitable pending message is available. Note that the order of output messages is not preserved in general. By rule (TAU), a contract willing to perform an internal action τ can do so and exhibit the label A *does* τ. Rule (CPAR) makes two contracts evolve in parallel.

In this paper the actual choice of the set Φ and of the relation $\vdash$ is almost immaterial. For the sake of the examples we choose for Φ the set of LTL formulae [17], where the atomic formulae are the atoms in $\mathcal{A}$, and $\vdash$ is $\models_{LTL}$ where, for all traces η of c, the semantics of atoms is defined as: $\eta \models \alpha \iff \exists A, \eta'.\ \eta = (A \ \mathit{does}\ \alpha)\ \eta'$. Decidability of $\vdash$ is ensured by restricting contracts to finite control processes [16], i.e. processes where parallel composition does not appear under recursion.

Concerning the fulfillment relation $\smile$, note that a process with an output at the toplevel is always culpable, while an input at the toplevel makes a process culpable only if the corresponding pending message is available.

Example 2.2. *Recall the buyer-seller scenario from Example 2.1. The seller* A *first requires the buyer* B *to pay, then promises to ship some goods*

to B. *The buyer* B *promises to pay* A*, and then to receive the item. Let the contracts of* A *and* B *be defined as follows:*

$$c_{\mathsf{A}} = \mathsf{A}\ \mathit{says}\ \mathsf{B}\rangle\mathsf{pay}.\ \overline{\mathsf{ship}}\rangle\mathsf{B}$$
$$c_{\mathsf{B}} = \mathsf{B}\ \mathit{says}\ \overline{\mathsf{pay}}\rangle\mathsf{A}.\ \mathsf{A}\rangle\mathsf{ship}$$

A possible evolution of the contract $c_{\mathsf{A}} \mid c_{\mathsf{B}}$ *is then:*

$$\begin{aligned} c_{\mathsf{A}} \mid c_{\mathsf{B}} &\xrightarrow{\mathsf{B}\ \mathit{does}\ \overline{\mathsf{pay}}\rangle\mathsf{A}}\!\!\!\!\rightarrow\ c_{\mathsf{A}} \mid \mathsf{B}\ \mathit{says}\ \mathsf{A}\rangle\mathsf{ship} \mid \mathsf{B}\rangle\mathsf{pay}\rangle\mathsf{A} \\ &\xrightarrow{\mathsf{A}\ \mathit{does}\ \mathsf{B}\rangle\mathsf{pay}}\!\!\!\!\rightarrow\ \mathsf{A}\ \mathit{says}\ \overline{\mathsf{ship}}\rangle\mathsf{B} \mid \mathsf{B}\ \mathit{says}\ \mathsf{A}\rangle\mathsf{ship} \\ &\xrightarrow{\mathsf{A}\ \mathit{does}\ \overline{\mathsf{ship}}\rangle\mathsf{B}}\!\!\!\!\rightarrow\ \mathsf{A}\rangle\mathsf{ship}\rangle\mathsf{B} \mid \mathsf{B}\ \mathit{says}\ \mathsf{A}\rangle\mathsf{ship} \\ &\xrightarrow{\mathsf{B}\ \mathit{does}\ \mathsf{A}\rangle\mathsf{ship}}\!\!\!\!\rightarrow\ 0 \end{aligned}$$

Note that that the buyer contract c_{B} in Example 2.2 is subject to attacks. Indeed, A can use a different contract c_{A} allowing her to receive the payment without having to perform any other action (cf. Example 3.12). In that case, she can legitimately choose not to ship, and still be regarded as non culpable.

Further examples of contracts-as-processes will be provided in § 3.

2.2 Contracts as Formulae

For the second specialization of our generic model, we choose the contract logic PCL [4]. A comprehensive presentation of PCL is beyond the scope of this paper, so we give here just a brief overview, and we refer the reader to [4, 3] for more details.

PCL extends intuitionistic propositional logic IPC [28] with the connective $\twoheadrightarrow$, called *contractual implication*. Differently from IPC, a contract $\mathsf{b} \twoheadrightarrow \mathsf{a}$ implies a not only when b is true, like IPC implication, but also in the case that a "compatible" contract, e.g. $\mathsf{a} \twoheadrightarrow \mathsf{b}$, holds. So, PCL allows for a sort of "circular" assume-guarantee reasoning, summarized by the theorem $\vdash (\mathsf{b} \twoheadrightarrow \mathsf{a}) \ \wedge\ (\mathsf{a} \twoheadrightarrow \mathsf{b}) \ \rightarrow\ \mathsf{a} \wedge \mathsf{b}$. Also, PCL is equipped with an indexed lax modality $\_\ \mathit{says}\ \_$, similarly to [19]. The proof system of PCL extends that of IPC with the axioms in Figure 2, while remaining decidable.

Following Definition 2.1, we now define a contract-as-formulae language which builds upon PCL.

$$\top \twoheadrightarrow \top$$
$$(\phi \twoheadrightarrow \phi) \to \phi$$
$$(\phi' \to \phi) \to (\phi \twoheadrightarrow \psi) \to (\psi \to \psi') \to (\phi' \twoheadrightarrow \psi')$$
$$\phi \to (A \textit{ says } \phi)$$
$$(A \textit{ says } A \textit{ says } \phi) \to A \textit{ says } \phi$$
$$(\phi \to \psi) \to (A \textit{ says } \phi) \to (A \textit{ says } \psi)$$

Figure 2: Hilbert-style proof system of PCL (IPC axioms omitted)

Definition 2.3. *We define a* contracts-as-formulae *language as follows:*

- $\mathcal{A}$ *is partitioned in* promises*, written as* a*, and* facts*, written as* $!\mathsf{a}$*.*
- $\mathcal{C} = \Gamma$ *is the set of* PCL *formulae, with the following syntax:*

  $$c ::= \bot \mid \alpha \mid \neg p \mid p \vee p \mid p \wedge p \mid p \to p \mid p \twoheadrightarrow p \mid A \textit{ says } c$$

  We let $c \mid c'$ *be syntactic sugar for* $c \wedge c'$*. The signature* Σ *comprises the atoms* $\mathcal{A}$*, all the connectives of* PCL*, and the* $\_$ *says* $\_$ *modality.*
- *The labelled relation* $\twoheadrightarrow$ *is defined by the rule:*

  $$c \xrightarrow{A \textit{ does } \mathsf{a}} \twoheadrightarrow \quad c \mid A \textit{ says } \mathsf{a} \mid A \textit{ says } !\mathsf{a}$$
- $\Phi = \mathcal{C}$*, and* $\Sigma' = \Sigma$*.*
- $\vdash$ *is the provability relation of* PCL*.*
- $A \smile c$ *holds iff* $c \vdash A \textit{ says } \mathsf{a}$ *implies* $c \vdash A \textit{ says } !\mathsf{a}$*, for all promises* a*.*

Note that the definition of $\twoheadrightarrow$ allows participants to perform any actions: the result is that c is augmented with the corresponding fact $!\mathsf{a}$. We include the promise a as well, following the intuition that a fact may safely imply the corresponding promise. A participant A fulfils a contract c when all the promises a made by A have been followed by the fact $!\mathsf{a}$.

Example 2.3. *The contracts of seller* A *and buyer* B *from Example 2.1 can be modelled by the following contract-as-formulae:*

$$c_{\mathsf{A}} = \mathsf{A} \textit{ says } ((\mathsf{B} \textit{ says } \mathsf{pay}) \to \mathsf{ship}) \qquad c_{\mathsf{B}} = \mathsf{B} \textit{ says } \mathsf{pay}$$

By the proof system of PCL*, we have that:*

$$c_{\mathsf{A}} \mid c_{\mathsf{B}} \vdash (\mathsf{A} \textit{ says } \mathsf{ship}) \wedge (\mathsf{B} \textit{ says } \mathsf{pay})$$

Example 2.4. *Consider the following variant for the contracts of buyer and seller introduced in Example 2.3.*

$$c_{\mathsf{A}} = \mathsf{A}\ \mathit{says}\ ((\mathsf{B}\ \mathit{says}\ \mathsf{pay}) \twoheadrightarrow \mathsf{ship})$$
$$c_{\mathsf{B}} = \mathsf{B}\ \mathit{says}\ ((\mathsf{A}\ \mathit{says}\ \mathsf{ship}) \twoheadrightarrow \mathsf{pay})$$

By the proof system of PCL*, we have that:*

$$c_{\mathsf{A}} \mid c_{\mathsf{B}} \ \vdash\ (\mathsf{A}\ \mathit{says}\ \mathsf{ship}) \wedge (\mathsf{B}\ \mathit{says}\ \mathsf{pay})$$

2.3 On Contracts-as-Processes *vs.* Contracts-as-Formulae

We now compare contracts-as-processes with contracts-as formulae. Our main technical result is Theorem 2.7, where we show that contracts-as-formulae, expressed in a significant fragment of PCL, can be encoded into contracts-as-processes. Finally, we further discuss the differences between the two contract models in some specific examples.

Concretely, we consider a fragment of PCL contracts comprising atoms, conjunctions, says, and non-nested (contractual) implications. We call this fragment $1N$-PCL , after "PCL with 1 level of Nesting of connectives". Then, we provide an encoding of $1N$-PCL into contract-as-processes (Definition 2.5) and formally relate them.

Definition 2.4. *We define* $1N$-PCL *as the fragment of* PCL *where formulae* c *have the following syntax:*

$$\begin{array}{rcl} c & ::= & \bigwedge_{i\in\mathcal{I}}\ A_i\ \mathit{says}\ \gamma_i \\ \gamma & ::= & \bigwedge_{j\in\mathcal{J}}\ \mathsf{a}_j \ \mid\ (\bigwedge_{i\in\mathcal{I}}\ B_i\ \mathit{says}\ \mathsf{b}_i) \to \bigwedge_{j\in\mathcal{J}}\mathsf{a}_j \\ & & \mid\ (\bigwedge_{i\in\mathcal{I}}\ B_i\ \mathit{says}\ \mathsf{b}_i) \twoheadrightarrow \bigwedge_{j\in\mathcal{J}}\mathsf{a}_j \end{array}$$

Definition 2.5. *For all* $1N$-PCL *contracts* c *and set of participant names* $\mathcal{P}$*, we define the contract-as-process* $[c]_{\mathcal{P}}$ *by the rules in Figure 3.*

We now comment on the equations in Figure 3. All the rules are parameterized by a set $\mathcal{P}$ of participant names/variables, which is used to keep track of the participants mentioned in the contract. The encoding of a signed contract $A\ \mathit{says}\ \gamma$ is just the encoding $[\gamma]$, prefixed with the signature $A\ \mathit{says}$. The encoding of a conjunction of signed contracts c_i, with $i \in \mathcal{I}$, is the parallel composition of the encodings $[c_i]$.

$$\Big[\bigwedge_{i\in\mathcal{I}} A_i \ says\ \gamma_i\Big]_{\mathcal{P}} = \big|\big|_{i\in\mathcal{I}}\ A_i \ says\ [\gamma_i]_{\mathcal{P}}$$

$$\Big[\bigwedge_{j\in\mathcal{J}} \mathsf{a}_j\Big]_{\mathcal{P}} = \big|\big|_{j\in\mathcal{J}}\ OUT_{\mathcal{P}}(\mathsf{a}_j)$$

$$\Big[(\bigwedge_{i\in 1..n} B_i \ says\ \mathsf{b}_i) \to \bigwedge_{j\in\mathcal{J}} \mathsf{a}_j\Big]_{\mathcal{P}} = B_1\rangle\mathsf{b}_1.\ \cdots\ .B_n\rangle\mathsf{b}_n.\ \big(||_{j\in\mathcal{J}}\ OUT_{\mathcal{P}}(\mathsf{a}_j)\big)$$

$$\Big[(\bigwedge_{i\in\mathcal{I}} B_i \ says\ \mathsf{b}_i) \twoheadrightarrow \bigwedge_{j\in\mathcal{J}} \mathsf{a}_j\Big]_{\mathcal{P}} = \tau.0 + \tau.\big(||_{i\in\mathcal{I}} DEBT(B_i,\mathsf{b}_i)\ \big|\ ||_{j\in\mathcal{J}}\ OUT_{\mathcal{P}}(\mathsf{a}_j)\big)$$

$$OUT_{\mathcal{P}}(\mathsf{a}) = \tau.0\ +\ \sum_{A\in\mathcal{P}} \overline{\mathsf{a}}\rangle A.OUT_{\mathcal{P}}(\mathsf{a})$$

$$DEBT(A,\mathsf{a}) = \tau.DEBT(A,\mathsf{a}) + A\rangle\mathsf{a}.0$$

Figure 3: Mapping from $1N$-PCL contracts to contracts-as-processes.

The encoding of an (unsigned) atom a is the process $OUT_{\mathcal{P}}(\mathsf{a})$. This process can either perform an internal action and then terminate, or perform an output $\overline{\mathsf{a}}\rangle A$ and then return to the initial state. The set $\mathcal{P}$ is used to send a (non-deterministically) to each participant mentioned in the contract. This is to accomodate the explicit receivers of contracts-as-processes with the fact that, in $1N$-PCL , actions are "broadcasted" to all participants. Note that $OUT(\mathsf{a})$ is a recursive process, where an output of a can be performed at each recursion. This reconciles the fact that PCL has a non-linear interpretation of actions (i.e. once an a is true, it can be "consumed" an arbitrary number of times), while contracts-as-processes interpret actions linearly.

The encoding of an implication $(\bigwedge_{i\in 1..n} B_i \ says\ \mathsf{b}_i) \to \bigwedge_{j\in\mathcal{J}} \mathsf{a}_j$ is a process that first inputs all the premises $\mathsf{b}_1,\ldots,\mathsf{b}_n$ from the respective senders, and then outputs (in parallel) all the consequences a_j. This is done through the recursive process $OUT(\mathsf{a}_j)$ discussed above. Note that the order in which the inputs are put is arbitrary. An alternative mapping could be:

$$\Big[(\bigwedge_{i\in\mathcal{I}} B_i \ says\ \mathsf{b}_i) \to \bigwedge_{j\in\mathcal{J}} \mathsf{a}_j\Big]_{\mathcal{P}} = IN(\{B_i\rangle\mathsf{b}_i\}_{i\in\mathcal{I}},\ \{\mathsf{a}_j\}_{j\in\mathcal{J}})$$

$$IN(\mathcal{X},\mathcal{O}) = \begin{cases} \sum_{\pi\in\mathcal{X}} \pi.\ IN(\mathcal{X}\setminus\{\pi\},\mathcal{O}) & \text{if } \mathcal{X}\neq\emptyset \\ ||_{\mathsf{a}\in\mathcal{O}}\ OUT_{\mathcal{P}}(\mathsf{a}) & \text{if } \mathcal{X}=\emptyset \end{cases}$$

We opted for the simpler encoding, because the order in which inputs are executed is immaterial. Indeed, (i) all outputs are asynchronous and

replicated, and (*ii*) what matters is that all the inputs have a matching output. In other words, it is not important if a blocked input prefix occurs before or after another input prefix; the outputs in the consequence of $\to$ will not be fired anyway.

The encoding of a contractual implication $(\bigwedge_{i\in\mathcal{I}} B_i \ says\ \mathsf{b}_i) \twoheadrightarrow \bigwedge_{j\in\mathcal{J}} \mathsf{a}_j$ is a process which can either perform an internal action and terminate, or perform an internal action and continue with the process $||_{i\in\mathcal{I}} DEBT(B_i, \mathsf{b}_i)$, in parallel with $||_{j\in\mathcal{J}}\ OUT_{\mathcal{P}}(\mathsf{a}_j)$. The outputs have the same behaviour as in the case of standard implication. The process $DEBT(B_i, \mathsf{b}_i)$ waits for an input of b_i, but while doing so, it may perform internal actions. Therefore, the participant who runs a $DEBT(B_i, \mathsf{b}_i)$ is culpable as long as he keeps iterating. When the process eventually chooses to perform the input, it is no longer culpable.

Theorem 2.7 below establishes a correspondence between contracts-as-formulae and contracts-as-processes. It states that the formulae $A\ says\ \mathsf{a}$ logically entailed by a $1N$-PCL contract c exactly correspond to those actions labelling the traces of $[c]$ which lead to states where no participants are culpable.

A technical issue is related to "ambiguous" contracts where some atom is associated with more than one participant. We rule out this case by restricting to non-ambiguous contracts, where instead each atom is said by a unique participant. This requires to analyse the contract at hand in order to tell, for each atom contained therein, the participant name which "binds" it. Technically, this is modelled by the relation $\pi(c)$ between atoms and participants, introduced in Definition 2.6 below. We shall say that c is *non-ambiguous* iff $\pi(c)$ is a function.

Definition 2.6. *For all c, we define the relation $\pi(c) \in \mathcal{A} \times \mathcal{N}_P \cup \mathcal{V}_P$ as follows, where $\circ \in \{\to, \twoheadrightarrow\}$:*

$$\pi(\bigwedge_{i\in\mathcal{I}} A_i\ says\ \gamma_i) = \bigcup_{i\in\mathcal{I}} \pi_{A_i}(\gamma_i)$$

$$\pi_A(\bigwedge_{j\in\mathcal{J}} \mathsf{a}_j) = \{(\mathsf{a}_j, A))\ \mid\ j \in \mathcal{J}\}$$

$$\pi_A((\bigwedge_{i\in\mathcal{I}} B_i\ says\ \mathsf{b}_i) \circ \bigwedge_{j\in\mathcal{J}} \mathsf{a}_j) = \{(\mathsf{b}_i, B_i)\ \mid\ i \in \mathcal{I}\} \cup \{(\mathsf{a}_j, A))\ \mid\ j \in \mathcal{J}\}$$

We can now state Theorem 2.7. Its proof is contained in the appendix. Note that the non-ambiguity of c is only needed in the "only if" direction.

Theorem 2.7. *Let c be a non-ambiguous $1N$-PCL contract, and let the set $\mathcal{P}$ include the participant names occurring in c. Then, $c \vdash A$ says a iff:*

$$\exists \eta, d, B.\ \Big([c]_{\mathcal{P}} \xrightarrow{\eta} d \ \wedge\ (A\ \mathit{does}\ \overline{\mathsf{a}}\rangle B) \in \eta \ \wedge\ \forall B' \in \mathcal{P}.\ B' \breve{\cup} d \Big)$$

Example 2.5. *Recall the contracts from Example 2.4. According to Definition 2.5, we have that:*

$$\begin{array}{rcl} [c_{\mathsf{A}} \mid c_{\mathsf{B}}] & = & \mathsf{A}\ \mathit{says}\ \tau.0 + \tau.\big(\mathit{DEBT}(\mathsf{B},\mathsf{pay}) \mid \mathit{OUT}(\mathsf{ship})\big) \mid \\ & & \mathsf{B}\ \mathit{says}\ \tau.0 + \tau.\big(\mathit{DEBT}(\mathsf{A},\mathsf{ship}) \mid \mathit{OUT}(\mathsf{pay})\big) \\ & \xrightarrow{\eta}\!\!\!\!\rightarrow & 0 \end{array}$$

where $\eta = (\mathsf{A}\ \mathit{does}\ \tau)\ (\mathsf{B}\ \mathit{does}\ \tau)\ (\mathsf{A}\ \mathit{does}\ \overline{\mathsf{ship}}\rangle\mathsf{B})\ (\mathsf{B}\ \mathit{does}\ \overline{\mathsf{pay}}\rangle\mathsf{A})\ (\mathsf{A}\ \mathit{does}\ \mathsf{B}\rangle\mathsf{pay})\ (\mathsf{B}\ \mathit{does}\ \mathsf{A}\rangle\mathsf{ship})\ (\mathsf{A}\ \mathit{does}\ \tau)\ (\mathsf{B}\ \mathit{does}\ \tau)$. This agrees with Theorem 2.7, since $\mathsf{A} \breve{\cup} 0$ and $\mathsf{B} \breve{\cup} 0$.

Remark 2.1. *The contracts-as-processes used in Example 2.2 may suggest a simpler encoding of $\twoheadrightarrow$ than the one given in Figure 3. For instance, suppose that — along that direction — we naïvely encode the formula B says $(\mathsf{A}$ says $\mathsf{ship}) \twoheadrightarrow \mathsf{pay}$ as the following process c_{B}*

$$c_{\mathsf{B}} = \mathsf{B}\ \mathit{says}\ \overline{\mathsf{pay}}\rangle\mathsf{A}.\ \mathsf{A}\rangle\mathsf{ship}$$

Assume that the context of B is empty, hence it never performs a $\overline{\mathsf{ship}}$ output. Then, we have that

$$c_{\mathsf{B}} \xrightarrow{\mathsf{B}\ \mathit{does}\ \overline{\mathsf{pay}}\rangle\mathsf{A}}\!\!\!\!\rightarrow c'_{\mathsf{B}} = \mathsf{B}\ \mathit{says}\ \mathsf{A}\rangle\mathsf{ship}$$

Note that no participant is culpable in c'_{B}. The simplified encoding would then lead to a counterexample to Theorem 2.7. Indeed, such theorem would imply

$$\mathsf{B}\ \mathit{says}\ (\mathsf{A}\ \mathit{says}\ \mathsf{ship}) \twoheadrightarrow \mathsf{pay} \ \vdash\ \mathsf{B}\ \mathit{says}\ \mathsf{pay}$$

which does not hold in PCL*. To solve this problem, the encoding in Definition 2.5 guarantees that some participant is culpable until the $\overline{\mathsf{ship}}$ output has been performed.*

3 A Calculus for Contract-Oriented Computing

We now introduce the syntax and semantics of CO_2. It generalises the contract calculus in [4] by making it independent of the actual contract language. In fact, CO_2 assumes the abstract model of contracts introduced in § 2, which can be instantiated to a variety of actual contract languages.

$$
\begin{array}{rcccccccc}
S & ::= & \mathbf{0} & \mid & \mathsf{A}[P] & \mid & s[c] & \mid & S \mid S & \mid & (u)S \\
P & ::= & \downarrow_u c & \mid & \sum_i \pi_i.P_i & \mid & P \mid P & \mid & (u)P & \mid & X(\vec{u}) \\
\pi & ::= & \tau & \mid & \mathsf{do}_u\, \alpha & \mid & \mathsf{tell}_u \downarrow_v \gamma & \mid & \mathsf{ask}_{u,\vec{v}}\, \phi & \mid & \mathsf{fuse}_u\, \phi
\end{array}
$$

Figure 4: Syntax of CO_2

3.1 Syntax

First, let us define the syntax of CO_2.

Definition 3.1. *The abstract syntax of* CO_2 *is given by the productions in Figure 4, where* S *stands for* systems*,* P *for* processes*, and* π *for* prefixes*. Also, we write* $||_{i\in I}\, P_i$ *(resp.* $||_{i\in I}\, S_i$*) for the parallel composition of processes* $(P_i)_{i\in I}$ *(resp. systems* $(S_i)_{i\in I}$*). Further, we stipulate that the following conditions always hold: in a system* $||_{i\in I}\, n_i[P_i]$*,* $n_i \neq n_j$ *for each* $i \neq j \in I$*; in a process* $(u)P$ *and a system* $(u)S$*,* $u \notin \mathcal{N}_P$*.*

We distinguish between *processes* and *systems*. Systems S consist of a set of *agents* $\mathsf{A}[P]$ and of *sessions* $s[c]$, composed in parallel. Note that A and s are *names* of participants and sessions, respectively.

Processes P comprise latent contracts, guarded summation, parallel composition, scope delimitation, and process identifiers. A *latent contract* $\downarrow_x c$ represents a contract c which has not been stipulated yet; upon stipulation, the variable x will be bound to a fresh session name. We allow for finite prefix-guarded sums of processes; as usual, we write $\pi_1.P_1 + \pi_2.P_2$ for $\sum_{i=1,2} \pi_i.P_i$. The empty system and the empty sum are both denoted by $\mathbf{0}$. As usual, we may omit trailing occurrences of $\mathbf{0}$ in processes. Processes can be composed in parallel, and can be put under the scope of binders $(u)\_$. We use process identifiers X to express recursive processes, and we assume that each identifier has a corresponding defining equation $X(u_1, \ldots, u_j) \stackrel{\text{def}}{=} P$ such that $\mathrm{var}(P) \subseteq \{u_1, \ldots, u_j\} \subseteq \mathcal{V}$ and each occurrence of process identifiers Y in P is prefix-guarded. Free/bound occurrences of variables/names are defined as expected. Variables and session names can be bound; in $(u)S$ all free occurrences of u in S are bound. Note that participant names cannot be bound, but can be communicated if permitted by the contract language. A system S is *closed* when it has no free variables.

Prefixes include the silent prefix τ, action execution $\mathsf{do}_u\, \alpha$, contract advertisement $\mathsf{tell}_u \downarrow_v \gamma$, contract query $\mathsf{ask}_{u,\vec{v}}\, \phi$, and contract stipulation

commutative monoidal laws for $|$ on processes and systems

$$u[(v)P] \equiv (v)u[P] \quad \text{if } u \neq v \qquad Z \mid (u)Z' \equiv (u)(Z \mid Z') \quad \text{if } u \notin \text{var}(Z) \cup \text{fn}(Z)$$

$$(u)(v)Z \equiv (v)(u)Z \qquad (u)Z \equiv Z \quad \text{if } u \notin \text{var}(Z) \cup \text{fn}(Z)$$

$$\downarrow_n c \equiv \mathbf{0} \qquad \mathsf{tell}_u \downarrow_n \gamma.P \equiv \mathbf{0} \qquad \mathsf{ask}_{u,\vec{v}}\, \phi.P \equiv \mathbf{0} \quad \text{if } \vec{v} \cap \mathcal{N} \neq \emptyset \qquad \mathsf{fuse}_n\, \phi.P \equiv \mathbf{0}$$

Figure 5: Structural equivalence (Z, Z' range over systems or processes)

$\mathsf{fuse}_u\, \phi$. Note that the contract γ in a $\mathsf{tell}\, \gamma$ is unsigned: the correct signature A *says* $\cdots$ will be attached by the semantics. The index u in tell_u /ask_u indicates the target agent/session where the prefix will be fired; in the case of fuse_u , it refers to the session where the contracts will be stipulated.

We shall refer to the set of latent contracts within an agent $\mathsf{A}[P]$ as its *environment*; similarly, in a session $s[c]$ we shall refer to c as the environment of s. Note that the environment of agents can only contain latent contracts, advertised through the primitive tell, while sessions can only contain (non-latent) contracts, obtained either upon reaching an agreement with fuse, or possibly upon participants performing some do prefixes.

3.2 Semantics

The semantics of CO_2 is formalised by a reduction relation on systems which relies on the structural congruence laws in Figure 5. Only the last row in Figure 5 contains non-standard laws: they allow for collecting garbage terms which may possibly arise after variable substitutions.

Definition 3.2. *The binary relation $\rightarrow$ on closed systems is the smallest relation closed under structural congruence and under the rules in Figure 6, where the agreement relation $K \rhd_x^\sigma \phi$ in (*Fuse*) will be introduced in Definition 3.3, and $\uparrow K$ is obtained by removing all the $\downarrow_-$ from K, i.e. if $K = ||_{i \in I} \downarrow_{x_i} c_i$, then $\uparrow K = ||_{i \in I} c_i$.*

We now briefly comment on the rules in Figure 6.

Rules (Tau), (Par), (Del), and (Def) are standard. Axioms (Tell_1) and (Tell_2) state that a participant A can advertise a latent contract $\downarrow_x \gamma$ either in her own environment, or in a remote one. When the contract

$$\mathsf{A}[\tau.P + P' \mid Q] \to \mathsf{A}[P \mid Q] \tag{Tau}$$

$$\mathsf{A}[\mathsf{tell}_{\mathsf{A}} \downarrow_x \gamma.P + P' \mid Q] \to \mathsf{A}[\downarrow_x \mathsf{A}\ \mathit{says}\ \gamma \mid P \mid Q] \tag{Tell$_1$}$$

$$\mathsf{A}[\mathsf{tell}_{\mathsf{B}} \downarrow_x \gamma.P + P' \mid Q] \ \mid\ \mathsf{B}[R] \to \mathsf{A}[P \mid Q] \ \mid\ \mathsf{B}[R \mid \downarrow_x \mathsf{A}\ \mathit{says}\ \gamma] \tag{Tell$_2$}$$

$$\frac{c \xrightarrow{\mathsf{A}\ \mathit{does}\ \alpha}\!\!\!\!\rightarrow\ c'}{\mathsf{A}[\mathsf{do}_s\ \alpha.P + P' \mid Q] \mid s[c] \to \mathsf{A}[P \mid Q] \mid s[c']} \tag{Do}$$

$$\frac{\mathrm{dom}\,\sigma = \vec{u} \subseteq \mathcal{V} \qquad c\sigma \vdash \phi\sigma}{(\vec{u})(\mathsf{A}[\mathsf{ask}_{s,\vec{u}}\ \phi.P + P' \mid Q] \mid s[c] \mid S) \to \mathsf{A}[P \mid Q]\sigma \ \mid\ s[c]\sigma \mid S\sigma} \tag{Ask}$$

$$\frac{K \rhd_x^\sigma \phi \qquad \vec{u} = \mathrm{dom}\,\sigma \subseteq \mathcal{V} \qquad s = \sigma(x)\ \ \text{fresh}}{(\vec{u})(\mathsf{A}[\mathsf{fuse}_x\ \phi.P + P' \mid K \mid Q] \mid S) \to (s)(\mathsf{A}[P \mid Q]\sigma \ \mid\ s[\uparrow K]\sigma \mid S\sigma)} \tag{Fuse}$$

$$\frac{X(\vec{u}) \stackrel{\text{def}}{=} P \qquad P\{^{\vec{v}}/_{\vec{u}}\} \to P'}{X(\vec{v}) \to P'} \tag{Def}$$

$$\frac{S \to S'}{S \mid S'' \to S' \mid S''} \tag{Par}$$

$$\frac{S \to S'}{(u)S \to (u)S'} \tag{Del}$$

Figure 6: Reduction semantics of CO_2

is advertised, the correct signature A *says* $\cdots$ is attached. In (Do), the participants A may perform the atom α in session s, provided that the action A *does* α is permitted on the contract in s. Rule (Ask) allows A to check if an observable ϕ is entailed by the contracts c in session s; notice that the entailment is subject to an instantiation σ (possibly empty) of the variables mentioned in the ask prefix. Rule (Fuse) establishes a multi-party session s among all the parties that reach an agreement on the latent contracts K. Roughly, the *agreement* relation $K \rhd_x^\sigma \phi$ (Definition 3.3) holds when, upon some substitution σ, the latent contracts K entail ϕ.

The simplest typical usage of these primitives is as follows. First, a group of participants exchanges latent contracts using tell, hence sharing their intentions. Then, one of them opens a new session using the fuse primitive. Once this happens, each involved participant A can inspect the session using ask, hence discovering her actual duties within that session: in general, these depend not only on the contract advertised by A, but also on those of the other participants (see e.g. Example 3.4). Finally, the primitive do is used to actually perform the duties.

Example 3.1. *The sale scenario between seller* A *and buyer* B *from Example 2.3 can be formalized as follows.*

$$\begin{array}{rcl} S & = & \mathsf{A}[(x,b)\,\mathsf{tell}_{\mathsf{A}}\downarrow_x ((b\ \mathit{says}\ \mathsf{pay}) \to \mathsf{ship}).\ \mathsf{fuse}_x\,(\mathsf{A}\ \mathit{says}\ \mathsf{ship}).\ \mathsf{do}_x\,\mathsf{ship}] \\ & | & \mathsf{B}[(y)\,\mathsf{tell}_{\mathsf{A}}\downarrow_y\ \mathsf{pay}.\ \mathsf{do}_y\,\mathsf{pay}] \end{array}$$

The buyer tells A *a contract which binds* B *to pay, and eventually does that. A session between the buyer and the seller is created and proceeds smoothly as expected, as shown in the trace in Figure 7.*

In the previous example, we have modelled the buyer-seller system by using a contracts-as-formulae approach. In the following example, we adopt instead the contracts-as-processes paradigm. In the meanwhile, we introduce a further participant to our system: a broker C which collects the contracts from A and B, and then uses a fuse to find when an agreement is possible.

Example 3.2. *Recall the contracts of Example 2.2. We specify the behaviour of the system as follows, where observables are expressed in LTL:*

$$\begin{array}{rcl} S & \stackrel{def}{=} & \mathsf{A}[\,(x,b)\,\mathsf{tell}_{\mathsf{C}}\,(\downarrow_x\, b\rangle\mathsf{pay}.\overline{\mathsf{ship}}\rangle b).\ \mathsf{do}_x\,b\rangle\mathsf{pay}.\ \mathsf{do}_x\,\overline{\mathsf{ship}}\rangle b\,] \\ & | & \mathsf{B}[\,(y)\,\mathsf{tell}_{\mathsf{C}}\downarrow_y\,\overline{\mathsf{pay}}\rangle\mathsf{A}.\ \mathsf{do}_y\,\overline{\mathsf{pay}}\rangle\mathsf{A}.\,\mathsf{do}_y\,\mathsf{A}\rangle\mathsf{ship}\,] \\ & | & \mathsf{C}[\,(z,a,b)\,\mathsf{fuse}_z\ \Diamond\big(\overline{\mathsf{pay}}\rangle a \wedge \Diamond\overline{\mathsf{ship}}\rangle b\big)\,] \end{array}$$

$$
\begin{array}{rcl}
S & \longrightarrow & (x,b,y)\ \big(\mathsf{A}[\downarrow_x \mathsf{A}\ \mathit{says}\ ((b\ \mathit{says}\ \mathsf{pay}) \to \mathsf{ship}) \mid \\
& & \qquad \mathsf{fuse}_x\ (\mathsf{A}\ \mathit{says}\ \mathsf{ship}).\ \mathsf{do}_x\ \mathsf{ship}] \\
& & \quad \mid \mathsf{B}[\mathsf{tell}_{\mathsf{A}} \downarrow_y \mathsf{pay}.\ \mathsf{do}_y\ \mathsf{pay}]\ \big) \\
& \longrightarrow & (x,b,y)\ \big(\mathsf{A}[\downarrow_y \mathsf{B}\ \mathit{says}\ \mathsf{pay} \mid \downarrow_x \mathsf{A}\ \mathit{says}\ ((b\ \mathit{says}\ \mathsf{pay}) \to \mathsf{ship}) \mid \\
& & \qquad \mathsf{fuse}_x\ (\mathsf{A}\ \mathit{says}\ \mathsf{ship}).\ \mathsf{do}_x\ \mathsf{ship}] \\
& & \quad \mid \mathsf{B}[\mathsf{do}_y\ \mathsf{pay}]\ \big) \\
& \longrightarrow & (s)\ \big(\mathsf{A}[\mathsf{do}_s\ \mathsf{ship}] \\
& & \quad \mid \mathsf{B}[\mathsf{do}_s\ \mathsf{pay}] \\
& & \quad \mid s[\mathsf{A}\ \mathit{says}\ ((\mathsf{B}\ \mathit{says}\ \mathsf{pay}) \to \mathsf{ship}) \mid \mathsf{B}\ \mathit{says}\ \mathsf{pay}]\ \big) \\
& \longrightarrow & (s)\ \big(\mathsf{A}[\mathbf{0}] \\
& & \quad \mid \mathsf{B}[\mathsf{do}_s\ \mathsf{pay}] \\
& & \quad \mid s[\mathsf{A}\ \mathit{says}\ ((\mathsf{B}\ \mathit{says}\ \mathsf{pay}) \to \mathsf{ship}) \mid \mathsf{B}\ \mathit{says}\ \mathsf{pay} \mid \mathsf{A}\ \mathit{says}\ !\mathsf{ship} \mid \ldots]\ \big) \\
& \longrightarrow & (s)\ \big(\mathsf{A}[\mathbf{0}] \\
& & \quad \mid \mathsf{B}[\mathbf{0}] \\
& & \quad \mid s[\mathsf{A}\ \mathit{says}\ ((\mathsf{B}\ \mathit{says}\ \mathsf{pay}) \to \mathsf{ship}) \mid \mathsf{B}\ \mathit{says}\ \mathsf{pay}\ \mid \\
& & \qquad \mathsf{A}\ \mathit{says}\ !\mathsf{ship} \mid \mathsf{B}\ \mathit{says}\ !\mathsf{pay} \mid \ldots]\ \big)
\end{array}
$$

Figure 7: A trace of the system in Example 3.1

The participants A *and* B *advertise their contracts to the broker* C*, which opens a session for their interaction. The broker checks that the buyer declares that he will eventually pay, and that the seller declares that she will eventually ship. As expected,*

$$S \longrightarrow^* (s)\ \big(\mathsf{A}[\mathbf{0}] \mid \mathsf{B}[\mathbf{0}] \mid s[\mathbf{0}]\big)$$

namely, as soon as the payment is received, the goods are shipped.

3.3 On Agreements

To give semantics to $\mathsf{fuse}_x\ \phi$ we instantiate in Definition 3.3 below the relation $\_ \rhd_x \_$, which establishes when a set of latent contracts can be stipulated. Roughly, $K \rhd \phi$ states that the latent contracts in K satisfy ϕ, according to the relation $\vdash$ of the contract model. More formally, Definition 3.3 accommodates the variables in K and in ϕ as well as it enables the creation of the session for the evolution of the stipulated contracts. Notably, in this way we allow for agreements among an arbitrary number of participants (modulo a minimality condition explained below).

Definition 3.3. *For all parallel compositions K of latent contracts, for all substitutions $\sigma : \mathcal{V} \to \mathcal{N}$, for all $x \in \mathcal{V}$, and for all observables ϕ, we write $K \rhd_x^\sigma \phi$ iff the following conditions hold:*

- $x \in \operatorname{dom}\sigma$
- $\operatorname{var}(K\sigma) = \operatorname{var}(\phi\sigma) = \emptyset$
- $\exists s \in \mathcal{N}_S : \forall y \in \operatorname{dom}\sigma \cap \mathcal{V}_S : \sigma(y) = s$
- $(\uparrow K)\sigma \vdash \phi\sigma$
- *no $\sigma' \subset \sigma$ satisfies the conditions above, i.e. σ is* minimal.

The first four items in Definition 3.3 state that an agreement is reached when the latent contracts in K entail, under a suitable substitution, an observable ϕ. Recall that ϕ is the condition used by the participant acting as broker when searching for an agreement (cf. rule (Fuse) in Figure 6). The substitution σ has to instantiate all the variables appearing in the latent contracts K as well as the session variable x; the latter, together with any other session variables in the domain of σ, is mapped to a session name s. The freshness of s is guaranteed by the rule (Fuse).

The last item in Definition 3.3, i.e. the minimality condition on σ, allows brokers to tell apart unrelated contracts. For instance, let $\downarrow_{x_1} c_1$, $\downarrow_{x_2} c_2$, and $\downarrow_{x_3} c_3$ be the latent contracts advertised to a broker; if there is an agreement on $\downarrow_{x_1} c_1$ and $\downarrow_{x_2} c_2$, the latent contract $\downarrow_{x_3} c_3$ would not be included. This is illustrated by the following informal example.

Example 3.3. *Let ϕ be the observable "*A *shall ship the goods", and consider the following latent contracts, advertised by* A*,* B*, and* C*, respectively:*

- $\downarrow_{x_1} c_1 =$ *"in session x_1, if b pays, then I shall ship the goods"*
- $\downarrow_{x_2} c_2 =$ *"in session x_2, I shall pay"*
- $\downarrow_{x_3} c_3 =$ *"in session x_3, I shall kiss a frog".*

Note that the first two latent contracts do entail ϕ when $\sigma(b) =$ B*, and $\sigma(x_1) = \sigma(x_2) = s$. Without the minimality condition, $\sigma(x_3) = s$ could have been included in the agreement of* A *and* B*, despite the offer of* C *being somewhat immaterial.*

The following example shows that our approach allows for agreements with multiple levels of compliance.

Example 3.4. *Let c_{A}, c_{B_1}, and c_{B_2} be the following contracts-as-formulae:*

$$\begin{aligned} c_{\mathsf{A}} &= \mathsf{A}\ \mathit{says}\ \big((b\ \mathit{says}\ \mathsf{pay}) \to \mathsf{ship}\ \wedge\ (b\ \mathit{says}\ \mathsf{coupon}) \to \mathsf{gift}\big) \\ c_{\mathsf{B}_1} &= \mathsf{B}_1\ \mathit{says}\ \mathsf{pay} \\ c_{\mathsf{B}_2} &= \mathsf{B}_2\ \mathit{says}\ \mathsf{pay} \wedge \mathsf{coupon} \end{aligned}$$

Now, consider the following system composed of a seller A*, which also acts as a broker, and two buyers* B_1 *and* B_2*:*

$$\begin{array}{ll} & \mathsf{A}\ [(x,b)\ \mathsf{tell}_{\mathsf{A}} \downarrow_x c_{\mathsf{A}}.\ \mathsf{fuse}_x\ (b\ \mathit{says}\ \mathsf{pay}). \\ & \qquad \big(\mathsf{ask}_x\ !\mathsf{pay}.\ \mathsf{do}_x\ \mathsf{ship} \mid \mathsf{ask}_x\ !\mathsf{coupon}.\ \mathsf{do}_x\ \mathsf{gift}\big)] \\ \mid & \mathsf{B}_1[(y)\ \mathsf{tell}_{\mathsf{A}} \downarrow_y c_{\mathsf{B}_1}.\ \mathsf{do}_y\ \mathsf{pay}] \\ \mid & \mathsf{B}_2[(z)\ \mathsf{tell}_{\mathsf{A}} \downarrow_z c_{\mathsf{B}_2}.\ \mathsf{do}_y\ \mathsf{pay}.\ \mathsf{do}_y\ \mathsf{coupon}] \end{array}$$

The seller contract c_{A} is compliant with both c_{B_1} and c_{B_2}, i.e. both buyers can be put in a session with A*. When coupled with c_{B_2}, the contract c_{A} entails both the obligations* ship *and* gift *for* A*. When coupled with c_{B_1}, we obtain a weaker agreement, as only the obligation* ship *is entailed. Although both levels of agreement are possible, in some sense the contract c_{B_2} yields a tighter Service-Level Agreement than c_{B_1}. The seller* A *detects the actual service level she has to provide through the primitive* ask *.*

3.4 On Honesty

In Definition 3.5 below we set out when a participant A is *honest* in a given system S, i.e. she always respects all the promises made. More in detail, we consider all the possible runs of S, and require that in every session the participant A eventually is not culpable. To this aim, we shall exploit the fulfilment relation $\mathsf{A} \smile c$ from the contract model.

Before defining honesty, we need to cope with a few technical issues. First, the α-conversion of session names makes it hard to track the *same* session at different points in a trace. So, we consider traces without α-conversion of session names, formalized as the LTS $\rightsquigarrow$ in Definition 3.4.

Definition 3.4. *For all systems S, we define the function $\mathit{freeze}_{\mathcal{N}}(S)$ as:*

$$\mathit{freeze}_{\mathcal{N}}(S) = \{S' \mid S \equiv (s_1, \ldots, s_j)S' \text{ and } S' \text{ delimitation-free}\}$$

where a system is delimitation-free *if it does not contain delimitations of session names. We define the LTS* $\rightsquigarrow$ *on systems as*

$$S \rightsquigarrow S'' \qquad \textit{iff} \qquad S \to S' \textit{ and } S'' \in \textit{freeze}_{\mathcal{N}}(S')$$

We say that a $\rightsquigarrow$*-trace* $(S_i)_i$ *is* maximal *iff it is either infinite, or it ends in a state* S_n *such that* $S_n \not\rightsquigarrow$.

Another technical issue is that a participant could not get a chance to act in all the traces (see e.g. Example 3.17. To account for this fact, we will check the honesty of a participant in *fair* traces, only, i.e. those obtained by running S according to a fair scheduling algorithm. More precisely, a $\rightsquigarrow$-trace $(S_i)_i$ is *fair* if no single prefix is enabled in an infinite number of S_i.

Definition 3.5. *A participant* A *is* honest *in* S *iff for all maximal fair* $\rightsquigarrow$*-traces* $(S_i)_i$

$$\forall s.\ \exists j.\ \forall i \geq j,\ \vec{n},\ c,\ S'.\ \Big(S_i \equiv (\vec{n})(s[c]\ \mid\ S') \implies \mathsf{A} \breve{\cup} c\Big)$$

In other words, a participant A misbehaves if involved in a session s such that the contracts c of s do not settle the obligations of A.

3.5 What Can Go Wrong in Contract-Oriented Systems?

Contract-oriented systems are a subclass of distributed systems where the interaction among participants is based on the principle that "every promise must be kept". Of course, since the goals of the participants are possibly conflicting, and since systems resemble "the condition of mere nature, which is a condition of war of every man against every man" [20], this principle is not expected to be always obeyed in practice. Indeed, contract-oriented systems are characterised by a variety of possible misbehaviours and attacks, in part inherited from real-world issues in legal contracts, in part specific to (concurrent and distributed) software systems.

We outline below a coarse-grained taxonomy of possible misbehaviours in contract-oriented systems, where the participant A plays the role of the injured party. Associated to each issue in the taxonomy, we enumerate a (non-exhaustive) list of possible causes. We will do that rather informally, by appealing to the reader intuition, to be reinforced in § 3.6 with the help of a collection of relevant examples. Roughly, below we say that "A succeeds" in a session when the goals of A (either implicitly known to A, or explicitly

written in her contract) have been performed by the other participants involved in that session. We say that "A is protected" when, in each possible interaction, either A succeeds or someone else is culpable.

In Table 2 we summarize our taxonomy, and we point out some links to relevant CO_2 examples which may be found in the following section.

Agreement issues The participant A has advertised the contract c, but she is never placed in a session where c has been stipulated.

1. The contract c is over-protective for A: if stipulated by B, then A would be protected, while B would not.
2. The contracts advertised by the other participants do not guarantee the success of A.
3. The broker is over-protective (for one or more of the participants).
4. Fairness problems: even though the contract of A could be stipulated, the broker never includes A in a session.

Success issues The participant A participates in a session, but she does not succeed.

1. Some participant does not fulfil his contract, and he is considered culpable.
2. The contract of A is not strong enough to protect A: no participant is culpable.
3. The contract of A is strong enough to protect A, but the broker puts A in a session with participants which do not guarantee that the goal of A will be reached.
4. The contract of A is strong enough to protect A, but A performs unprotected actions.
5. Fairness problems: other participants cannot perform their duties because of unfair scheduling.

Culpability issues The participant A participates in a session where she tries to respect her contract c with the best intentions, but she is eventually considered culpable of a violation of c.

1. A relies on some other participant B to fulfil some of her duties, but B has "passed the buck" to A.

Agreement issues		
Cause	**Contracts-as-formulae**	**Contracts-as-processes**
Over-protective A	Ex. 3.5	Ex. 3.5
No success guarantee	Ex. 3.6	(similar)
Over-protective broker	fuse $\perp$	Ex. 3.7
Fairness problems	Ex. 3.8	Ex. 3.9

Success issues		
Cause	**Contracts-as-formulae**	**Contracts-as-processes**
Someone else culpable	Ex. 3.10	(similar)
Contract too weak	Ex. 3.11	(similar)
Deceptive broker	Ex. 3.13	Ex. 3.12
Unprotected actions	Ex. 3.14	(similar)
Fairness problems	(similar)	Ex. 3.15

Culpability issues		
Cause	**Contracts-as-formulae**	**Contracts-as-processes**
Pass the buck	(similar)	Ex. 3.16
Fairness problems	Ex. 3.17	(similar)

Table 2: Issues in contract-oriented systems.

2. Fairness problems: A cannot fulfil her contract because of unfair scheduling.

A main design principle of CO_2 is that it must allow for writing "wrong" specifications of systems which exhibit these kinds of misbehaviour and attacks. This is the reason why CO_2 does not prevent (almost) any actions in the semantics. In particular: (*i*) participants can advertise whatever contracts they wish, the only condition being that these can be expressed in the contract model; (*ii*) contract brokers can stipulate arbitrary contracts, and create sessions with an arbitrary number of participants; (*iii*) participants can perform whatever action in a session, if permitted in the current state of the contract in that session.

3.6 A Collection of Misbehaviours and Attacks

In this section we collect a set of archetypal contract-oriented scenarios, which use contracts from both paradigms introduced in § 2.1 and § 2.2. Each

of the scenarios discussed below highlights one of the causes of misbehaviour reported in the taxonomy in § 3.5.

Example 3.5. *Consider a variation of the system S in Example 3.1 where the buyer process is modified as follows:*

$$\mathsf{B}[(a,y)\ \mathsf{tell}_{\mathsf{A}}\downarrow_y ((a\ \mathit{says}\ \mathsf{ship})\to\mathsf{pay}).\ \mathsf{ask}_y\,(a\ \mathit{says}\ !\mathsf{ship}).\ \mathsf{do}_y\,\mathsf{pay}]$$

Now, the fuse *in* A *has to entail* A *says* ship *from the contract:*

$$\mathsf{A}\ \mathit{says}\ (\mathsf{B}\ \mathit{says}\ \mathsf{pay})\to\mathsf{ship}\quad\mid\quad\mathsf{B}\ \mathit{says}\ (\mathsf{A}\ \mathit{says}\ \mathsf{ship})\to\mathsf{pay}$$

Note that both participants require the other one to "make the first step". This circular dependency forbid any agreement. A similar behaviour can be modelled as follows, using contracts-as-processes:

$$\mathsf{A}\ \mathit{says}\ \mathsf{B}\rangle\mathsf{pay}.\ \overline{\mathsf{ship}}\rangle\mathsf{B}\quad\mid\quad\mathsf{B}\ \mathit{says}\ \mathsf{A}\rangle\mathsf{ship}.\ \overline{\mathsf{pay}}\rangle\mathsf{A}$$

Example 3.6. *Consider a variation of the buyer-seller system, where the buyer and seller contracts are:*

$$\begin{aligned} c_{\mathsf{A}} &= ((b\ \mathit{says}\ \mathsf{pay})\to\mathsf{snakeOil})\\ c_{\mathsf{B}} &= (a\ \mathit{says}\ \mathsf{ship})\twoheadrightarrow\mathsf{pay}\end{aligned}$$

and the processes are as follows:

$$\mathsf{A}[(x,b)\ \mathsf{tell}_{\mathsf{A}}\downarrow_x c_{\mathsf{A}}.\ \mathsf{fuse}_x\,(\mathsf{A}\ \mathit{says}\ \mathsf{snakeOil}).\ \mathsf{do}_x\,\mathsf{snakeOil}]$$

$$\mathsf{B}[(a,y)\ \mathsf{tell}_{\mathsf{A}}\downarrow_y c_{\mathsf{B}}.\ \mathsf{ask}_y\,(\mathsf{B}\ \mathit{says}\ \mathsf{pay}).\ \mathsf{do}_y\,\mathsf{pay}]$$

The interaction between the fraudulent *seller* A *and the buyer* B *gets stuck on the* fuse *in* A*, because the available latent contracts do not entail* A *says* snakeOil*. Note that we have used contractual implication $\twoheadrightarrow$, rather than standard implication $\to$. This would have allowed* B *to reach an agreement with the seller contract* $(b\ \mathit{says}\ \mathsf{pay})\to\mathsf{ship}$.

Example 3.7. *Consider the following variant of Example 3.4, using contracts-as-processes:*

$$\begin{aligned} c_{\mathsf{A}} &= \mathsf{A}\ \mathit{says}\ (\ \mathsf{B}\rangle\mathsf{pay}.\ \overline{\mathsf{ship}}\rangle\mathsf{B}\ \mid\ \mathsf{B}\rangle\mathsf{coupon}.\ \overline{\mathsf{gift}}\rangle\mathsf{B}\ )\\ c_{\mathsf{B}} &= \mathsf{B}\ \mathit{says}\ \overline{\mathsf{pay}}\rangle\mathsf{A}.\ (\ \tau.\ \mathsf{A}\rangle\mathsf{ship}+\tau.\ \overline{\mathsf{coupon}}\rangle\mathsf{A}.\ \mathsf{A}\rangle\mathsf{ship}.\ \mathsf{A}\rangle\mathsf{gift}\ )\end{aligned}$$

Now, assume that c_{A} *and* c_{B} *are advertised to a broker, which wants to establish an agreement through a* $\mathsf{fuse}\,\phi$*, with* $\phi = \Diamond\big(\overline{\mathsf{pay}}\rangle\mathsf{A} \wedge \Diamond\overline{\mathsf{coupon}}\rangle\mathsf{A}\big)$*. The broker will not be able to put* A *and* B *together in a session, because the contract* c_{B} *allows* B *to choose whether consuming the coupon or not. Instead, the broker would be able to establish the weaker agreement* $\phi' = \Diamond\overline{\mathsf{pay}}\rangle\mathsf{A}$*.*

Example 3.8. *Consider the system:*

$$S = \mathsf{A}[X_0] \mid \mathsf{B}[X_1] \mid \mathsf{C}[X_2] \mid \mathsf{K}[Y]$$

where, for $i \in 0..2$*,* $X_i \stackrel{\text{def}}{=} (x_i)\mathsf{tell}_{\mathsf{K}}\downarrow_{x_i} c_i.X_i$*,* $Y \stackrel{\text{def}}{=} (z)\mathsf{fuse}_z\,\phi.Y$*, and* ϕ *is satisfied by any* $c_i \mid c_j$ *with* $i \neq j$*. An unfair scheduler could always choose the contracts of* B *and* C*, and neglect those of* A*.*

Example 3.9. *Consider the following contracts-as-processes for the buyer-seller scenario:*

$$c_{\mathsf{A}} = X \qquad X \stackrel{def}{=} \tau.X + \mathsf{B}\rangle\mathsf{pay}.\,Y \qquad Y \stackrel{def}{=} \tau.Y + \overline{\mathsf{ship}}\rangle\mathsf{B}$$
$$c_{\mathsf{B}} = Z \mid \mathsf{A}\rangle\mathsf{ship} \qquad Z \stackrel{def}{=} \tau.Z + \overline{\mathsf{pay}}\rangle\mathsf{A}$$

Assume that the broker attempts to establish a session between A *and* B *through* $\mathsf{fuse}\,\phi$*, with* $\phi = \Diamond\big(\overline{\mathsf{pay}}\rangle\mathsf{A} \wedge \Diamond\overline{\mathsf{ship}}\rangle\mathsf{B}\big)$*. Without any fairness assumptions,* $c_{\mathsf{A}} \mid c_{\mathsf{B}} \not\models_{LTL} \phi$*, because an unfair scheduler can choose e.g. to never fire the prefix* $\overline{\mathsf{pay}}\rangle\mathsf{A}$*, even though this prefix is enabled infinitely often.*

Example 3.10. *Consider a variation of the system* S *in Example 3.1 where the seller process is modified as follows:*

$$\mathsf{A}[(x, b)\ \mathsf{tell}_{\mathsf{A}}\downarrow_x ((b\ \mathit{says}\ \mathsf{pay}) \to \mathsf{ship}).\ \mathsf{fuse}_x\,(\mathsf{A}\ \mathit{says}\ \mathsf{ship}).\ \mathbf{0}]$$

The fraudulent seller A *promises to ship, but it does not act accordingly and simply terminates. The interaction between* A *and* B *leads to the system:*

$$(s)\big(\mathsf{A}[\mathbf{0}] \mid \mathsf{B}[\mathbf{0}] \mid s[\mathsf{A}\ \mathit{says}\ ((\mathsf{B}\ \mathit{says}\ \mathsf{pay}) \to \mathsf{ship}),\ \mathsf{B}\ \mathit{says}\ \mathsf{pay},\ \mathsf{B}\ \mathit{says}\ !\mathsf{pay}]\big)$$

where the buyer has not obtained what he has paid for. However, the insuccess of the buyer is partially mitigated by the fact that the seller is found responsible of this misbehaviour. Indeed, the seller is dishonest according to Definition 3.5, because the contracts in s *entail the promise* A *says* ship*, while the fact* A *says* !ship *is not present in the session. A judge may thus eventually punish* A *for her misconduct.*

Example 3.11. *Recall Example 3.10 and consider a fraudulent seller* A*, which promises in her contract some snake oil:*

$$c_{\mathsf{A}} \;=\; ((b \; \mathit{says} \; \mathsf{pay}) \to \mathsf{snakeOil})$$

while the contract and the behaviour of B *are as in Example 3.1. Then:*

$$\begin{array}{rcl} S & = & \mathsf{A}[(x,b)\,\mathsf{tell}_{\mathsf{A}} \downarrow_x c_{\mathsf{A}}.\ \mathsf{fuse}_x\,(\mathsf{A}\ \mathit{says}\ \mathsf{snakeOil}).\ \mathsf{do}_x\,\mathsf{snakeOil}] \mid \mathsf{B}[\cdots] \\ & \longrightarrow^* & (s)\ \big(\mathsf{A}[\mathbf{0}] \;\mid\; \mathsf{B}[\mathbf{0}] \;\mid\; s[\mathsf{A}\ \mathit{says}\ !\mathsf{snakeOil},\ \mathsf{B}\ \mathit{says}\ !\mathsf{pay},\ \ldots]\big) \end{array}$$

In this case the interaction between A *and* B *goes unhappily for* B*: he will pay for some snake oil, while* A *is not even classified as dishonest according to Definition 3.5: indeed,* A *has eventually fulfilled all her promises. The cause of this misbehaviour is that the buyer contract is too weak for being used with untrusted participants.*

Example 3.12. *Recall the buyer-seller scenario from Example 2.2. We slightly modify the seller contract as follows, while* c_{B} *stays the same:*

$$c_{\mathsf{A}} = \mathsf{A}\ \mathit{says}\ b\rangle\mathsf{pay}.(\tau + \overline{\mathsf{ship}}\rangle b)$$

The seller A *first requires the buyer to pay, then promises either to ship an item, or to do nothing. A possible evolution of the contract* $c_{\mathsf{A}} \mid c_{\mathsf{B}}$*, where the participant name* B *substitutes for the participant variable* b*, is then:*

$$\begin{array}{rl} (c_{\mathsf{A}} \mid c_{\mathsf{B}})\{\mathsf{B}/b\} & \xrightarrow{\mathsf{B}\ \mathit{does}\ \overline{\mathsf{pay}}\rangle\mathsf{A}} \!\!\!\!\rightarrow\!\!\!\!\rightarrow\ c_{\mathsf{A}} \mid \mathsf{B}\ \mathit{says}\ \mathsf{A}\rangle\mathsf{ship} \mid \mathsf{B}\rangle\mathsf{pay}\rangle\mathsf{A} \\ & \xrightarrow{\mathsf{A}\ \mathit{does}\ \mathsf{B}\rangle\mathsf{pay}} \!\!\!\!\rightarrow\!\!\!\!\rightarrow\ \mathsf{A}\ \mathit{says}\ (\tau + \overline{\mathsf{ship}}\rangle\mathsf{B}) \mid \mathsf{B}\ \mathit{says}\ \mathsf{A}\rangle\mathsf{ship} \\ & \xrightarrow{\mathsf{A}\ \mathit{does}\ \tau} \!\!\!\!\rightarrow\!\!\!\!\rightarrow\ \mathsf{A}\ \mathit{says}\ 0 \mid \mathsf{B}\ \mathit{says}\ \mathsf{A}\rangle\mathsf{ship} \end{array}$$

The contract of B *is rather naïve, because the buyer pays without requiring any guarantee to the seller. Indeed, in the final state* $c' = \mathsf{B}\ \mathit{says}\ \mathsf{A}\rangle\mathsf{ship}$*, we have that* $\mathsf{A} \breve{\smile} c'$*, i.e.* A *is not culpable. Now, consider the following system, composed of a buyer* A*, a seller* B*, and a broker* C*:*

$$\begin{array}{rcl} S & \stackrel{def}{=} & \mathsf{A}[\,(x,b)\,(\mathsf{tell}_{\mathsf{C}} \downarrow_x c_{\mathsf{A}}.\ \mathsf{do}_x\, b\rangle\mathsf{pay}.\ (\tau + \mathsf{do}_x\,\overline{\mathsf{ship}}\rangle b))\,] \\ & \mid & \mathsf{B}[\,(y)\,(\mathsf{tell}_{\mathsf{C}} \downarrow_y c_{\mathsf{B}}.\ \mathsf{do}_y\,\overline{\mathsf{pay}}\rangle\mathsf{A}.\,\mathsf{do}_y\,\mathsf{A}\rangle\mathsf{ship})\,] \\ & \mid & \mathsf{C}[\,(z,d)\,\mathsf{fuse}_z\ \Diamond(\overline{\mathsf{pay}}\rangle d)\,] \end{array}$$

The broker C*, which opens a session between* A *and* B*. Notice that* C *is rather unfair, since it only checks that the buyer will eventually pay, while it does*

not require that the seller will eventually ship. The seller can then choose to perform the internal action τ, instead of shipping the paid item, while remaining not culpable. We have the following (maximal) computation:

$$S \longrightarrow^* (s)\ \big(\mathsf{A}[\mathbf{0}] \mid \mathsf{B}[\mathsf{do}_y\, \mathsf{A}\rangle\mathsf{ship}] \mid s[\mathbf{0}]\big)$$

where B *has not succeeded, but neither* A *nor* C *are considered culpable.*

Example 3.13. *Consider the following formalization of the system in Example 3.12, using contract-as-formulae:*

$$\mathsf{A}[(x,b)\mathsf{tell}_\mathsf{C}\downarrow_x ((b\ \mathit{says}\ \mathsf{pay}) \to (\mathsf{ship} \vee \mathsf{fraud}))\mathpunct{.}\ \mathsf{do}_x\,\mathsf{fraud}]$$
$$\mathsf{B}[(y,a)\mathsf{tell}_\mathsf{C}\downarrow_y ((a\ \mathit{says}\ \mathsf{ship}) \twoheadrightarrow \mathsf{pay})\mathpunct{.}\ \mathsf{ask}_y\,\mathsf{B}\ \mathit{says}\ \mathsf{pay}\mathpunct{.}\ \mathsf{do}_y\,\mathsf{pay}]$$
$$\mathsf{C}[(z)\mathsf{fuse}_z\,\phi]$$

where the formula ϕ is chosen so to make the fuse *in* C *pass (this can be obtained e.g. by taking the conjunction of the contracts of* A *and* B*). Unlike in Example 3.12, even if a session between* A *and* B *is established by the deceptive broker, the buyer will not pay, and he will not be considered culpable for that. Indeed, the prefix* $\mathsf{ask}_y\,\mathsf{B}$ *says* pay *is stuck because the contracts in the session do not entail any obligation for* B *to pay.*

Example 3.14. *Consider the following variant of Example 3.13, where the buyer process is modified as follows:*

$$\mathsf{B}[(y,a)\mathsf{tell}_\mathsf{C}\downarrow_y ((a\ \mathit{says}\ \mathsf{ship}) \twoheadrightarrow \mathsf{pay})\mathpunct{.}\ \mathsf{do}_y\,\mathsf{pay}]$$

Although the contract of B *is strong enough to protect* B*, this protection is lost when the buyer performs the action* pay *without checking beforehand (through* $\mathsf{ask}_y\,\mathsf{B}$ *says* pay*) that he actually has to pay. This imprudence leads* B *to a situation where he does not succeed, while* A *is not considered culpable.*

Example 3.15. *Consider a variant of the buyer-seller system in Example 3.2, where the seller behaviour is modified as follows:*

$$\mathsf{A}[\,(x,b)\,(\mathsf{tell}_\mathsf{C}\,(\downarrow_x b\rangle\mathsf{pay}.\overline{\mathsf{ship}}\rangle b)\mathpunct{.}\,\mathsf{do}_x\,b\rangle\mathsf{pay}\mathpunct{.}\,X\,] \qquad X \stackrel{def}{=} \tau.X + \mathsf{do}_x\,\overline{\mathsf{ship}}\rangle b$$

Even though the prefix $\mathsf{do}_x\,\overline{\mathsf{ship}}\rangle b$ *is enabled infinitely often, an unfair process-level scheduler could prevent* A *from eventually shipping. A similar misbehaviour can be obtained through an unfair system-level scheduler. Note that in Definition 3.5, we do not consider dishonest the participants which fail to fulfil their duties because of an unfair scheduling.*

Example 3.16. *Consider the following variant of the buyer-seller scenario. The seller* A *advertises a contract where she promises to provide the buyer* b *with a proof-of-shipment (*PoS*) if* b *pays (*pay*). After the buyer has paid, the seller advertises a contract to a broker* K*, where she looks for a carrier* c *which provides* A *with a proof-of-shipment if* A *pays shipping (*payS*).*

$$\begin{aligned}
&\mathsf{A}[\,(x,y,b,c)\,\mathsf{tell}_{\mathsf{A}}\downarrow_x\,(b\rangle\mathsf{pay}.\,\overline{\mathsf{PoS}}\rangle b).\ \mathsf{fuse}_x\,b\ \textit{says}\ \overline{\mathsf{pay}}\rangle\mathsf{A}.\ \mathsf{do}_x\,b\rangle\mathsf{pay}.\,P\,]\\
&\quad P\ =\ \mathsf{tell}_{\mathsf{K}}\downarrow_y\,(\overline{\mathsf{payS}}\rangle c.\,c\rangle\mathsf{PoS}).\ \mathsf{do}_y\,\overline{\mathsf{payS}}\rangle c.\ \mathsf{do}_y\,c\rangle\mathsf{PoS}.\ \mathsf{do}_x\,\overline{\mathsf{PoS}}\rangle b\\
&\mathsf{B}[\,(y)\,\mathsf{tell}_{\mathsf{A}}\downarrow_y\,(\overline{\mathsf{pay}}\rangle\mathsf{A}.\,\mathsf{A}\rangle\mathsf{PoS}).\ \mathsf{do}_y\,\overline{\mathsf{pay}}\rangle\mathsf{A}.\,\mathsf{do}_y\,\mathsf{A}\rangle\mathsf{PoS}\,]
\end{aligned}$$

Now, assume that the contract between buyer and seller has been stipulated, and that the buyer has paid. If no carrier is found to stipulate the contract with A*, then the seller will be considered culpable of a violation, because she has not honoured her promise to provide the buyer with a proof-of-shipment.*

Example 3.17. *Consider the system* $S = \mathsf{A}[\mathsf{do}_s\,\mathsf{pay}] \mid \mathsf{B}[X] \mid S'$*, where* $X \stackrel{\text{def}}{=} \tau.X$*, and* S' *contains a session where the action of* A *is enabled. Note that* S *generates the infinite trace* $S \rightsquigarrow S \rightsquigarrow S \rightsquigarrow \cdots$ *in which* A *never pays, despite her honest intention. According to Definition 3.5, participant* A *is considered honest.*

3.7 Variants to the Basic Calculus

We mention below a few variants and extensions of CO_2.

Protection for contracts-as-processes. The contracts-as-processes model can be adapted to allow participants to defend against attacks carried over by deceptive brokers, like the one shown in Example 3.12. The variant requires the semantics of $\mathsf{fuse}\,\phi$ to be changed so that contractual obligations derive only from mutually *compliant* contracts, i.e. $c \vdash \phi$ should hold only when all the (fair) traces of c lead to 0, in addition to $c \models_{LTL} \phi$. Similarly, $\mathsf{A} \smile c$ should also hold on non-mutually-compliant c, since in this case no obligation arises. With this change, even if a participant A is somehow put in a session with fraudulent parties, A can discover (via ask) that no actual obligation is present, so avoiding to perform unprotected actions.

Local actions. In CO_2, agents $\mathsf{A}[P]$ only carry latent contracts in P. Hence, communication between participants is limited to latent contract exchange. Allowing general data exchange in CO_2 can be done in a natural

way by following CCP. Basically, P would include CCP constraints t, ranging over a constraint system $\langle T, \tilde{\vdash} \rangle$, a further parameter to CO_2. Assume that A *says* $t \in T$ for all $t \in T$ and for all A. The following rules will augment the semantics of CO_2 (the syntax is extended accordingly):

$$\mathsf{A}[\mathsf{tell}_{\mathsf{A}}\, t.P + P' \mid Q] \to \mathsf{A}[\mathsf{A}\ \mathit{says}\ t \mid P \mid Q]$$

$$\mathsf{A}[\mathsf{tell}_{\mathsf{B}}\, t.P + P' \mid Q] \mid \mathsf{B}[R] \to \mathsf{A}[P \mid Q] \mid \mathsf{B}[\mathsf{A}\ \mathit{says}\ t \mid R]$$

$$\mathsf{A}[\mathsf{ask}_{\mathsf{A}}\, t.P + P' \mid T \mid Q] \ \to\ \mathsf{A}[P \mid T \mid Q] \quad \text{provided that} \quad T \tilde{\vdash} t$$

Note that the above semantics does not allow A to corrupt the constraint store of B augmenting it with arbitrary constraints. Indeed, exchanged data is automatically tagged on reception with the name of the sender. So, in the worst case, a malicious A can only insert garbage A *says* t into the constraint store of B.

Retracting latent contracts. A retract primitive could allow a participant A to remove a latent contract of hers after its advertisement. Therefore, A could change her mind until her latent contract is actually used to establish a session, where A is bound to her duties. Of course a contract cannot be retracted after it has been agreed upon.

Consistency check. The usual check t primitive from CCP, which checks the consistency of the constraint store with t, can also be added to CO_2. When check t is executed by a participant $\mathsf{A}[P]$, t is checked for consistency against the constraints in P. Note that checking the whole world would be unfeasible in a distributed system.

Forwarding latent contracts. A forward primitive could allow a broker A to move a latent contract from her environment to that of another broker B, without tagging it with A *says* . In this way A delegates to B the actual opening of a session.

Remote queries. More primitives to access the remote participants could be added. Note however that while it would be easy e.g. to allow $\mathsf{A}[\mathsf{ask}_{\mathsf{B}}\, t]$ to query the constraint of B, this would probably be undesirable for security reasons. Ideally, B should be allowed to express whether A can access his own constraint store. This requires some access control mechanism.

4 Related Work

Contracts have been used at different levels of abstraction, and with different purposes. A coarse classification of the approaches appeared in the literature separates the approaches that use contracts to model the possible interaction patterns of services, from those where contracts are used to model Service Level Agreements. In the former category, the typical goal is that of composing services only if their interactions enjoy progress properties (e.g. deadlock-freedom). In the latter, the goal is that of matching clients and services, so that they agree on the respective rights and obligations.

Multi-party session types [22] are integrated in [6] with decidable fragments of first-order logic (e.g., Presburger arithmetic) to transfer the design-by-contract of object-oriented programming to the design of distributed interactions. We follow a methodologically opposite direction. In fact, in [6] one starts from a *global assertion* (that is global choreography decorated with logical assertions) to arrive to a set of *local assertions*; distributed processes abiding by local assertions are guaranteed to have correct interactions. In our framework, a participant declares its contract independently of the others and then advertises it; a CO_2 primitive (fuse) tries then to harmonise contracts by searching for a suitable agreement. In other words, one could think of our approach as based on orchestration rather than choreography. The same considerations above apply to [23]; there protocols (state machines with memory) represent global choreographies and contracts are represented as parallel state machines (according to a CSP-like semantics). Basically, the contract model of [23] coincides with its choreography model.

Global assertions in [6] enable for the automatic generation of monitors. At a first glance, this looks similar to our use of contracts as driver of the computation. However, a main difference is that the monitors synthesised from global assertions have a "local" view of the computation as they basically check the incoming messages of a distributed component. Instead in CO_2 , the contracts stipulated in a session form a "global view" of the (current) evolution of the computation. This enables us to formalise a notion of *honesty* (cf. Definition 3.5) and single out culpable components during the computation. We argue that it would be hard to attain the same within the approach in [6]. For instance, a monitor obtained from a global assertion can check for an incoming message m from a partner A; however the monitor cannot distinguish if the absence of m is due to a violation the contract of A, or due to the fact that other duties (possibly from other participants) required by A are not accomplished. In other words, the monitor cannot

deem A culpable.

An advantage of the approach in [6] is that projecting a global assertion yields local assertions, that is the contracts of each component in the choreography. Using local assertions one can check if a component A abide by its contract L, namely if A *realises* L. In our framework this is more complex to be achieved since abstracting away from contracts makes it hard to give a general notion of "realisability".

In cc-pi [9], CCP is mixed with communication via name fusion so that parties establish SLA by merging the constraints representing their requirements. Constraints are values in a c-semiring advertised in a global store. It is not permitted to merge constraints making the global store inconsistent, since an agreement cannot be reached in that case. Conversely, CO_2 envisages contracts as binding promises rather than requirements. Actually, even if a participant A tells an absurdum, this will result in a contract like: "A is stating a contradiction" added to the environment. When this happens, our approach is not "contracts are inconsistent, do not open a session", but rather "A is promising the impossible, she will not be able to keep her promise, and she will be blamed for that". The cc-pi calculus is further developed in [8] to include long running transactions and compensations. There, besides the global store, the calculus features a local store for each transaction. Local inconsistencies are then used to trigger compensations. In CO_2 , compensations do not represent exceptional behaviour; they fall within "normal" behaviour and have to be spelt out inside contracts. Indeed, after a session has been established, each honest participant A either maintains her promises, or she is culpable of a violation; A cannot simply try to execute arbitrary compensations in place of the due actions. Of course, other participants may deem her promise too weak and avoid establishing a session with A.

In [15] a calculus is proposed to model SLAs which combines π-calculus communication, concurrent constraints, and sessions. There, the constraint store is global and sessions are established between two processes whenever the stated requirements are consistent. Interaction in sessions happens through communication and label branching/selection. A type system is provided to guarantee safe communication, although not ensuring progress. Essentially, the main role of constraints in this calculus is that of driving session establishment. Instead, in CO_2 the contracts of an agreement leading to a session are still relevant e.g. to detect violations.

In [14], contracts are modelled in a fragment of CCS which includes

prefixing, internal/external choice, and recursion. This model is not directly comparable with our CCS-like model in § 2.1: while our contracts do not feature external choice, they have parallel composition and synchronization. The notion of compliance between two contracts used in [14] is "boolean": either the client contract is compliant with the service contract, or it is not. In Example 3.4 we show that our approach allows for a "multi-level" notion of compliance, encompassing more than two contracts. In [14], contracts which are not compliant may become compliant by adjusting the order of asynchronous actions. When this is possible, an orchestrator can be synthesised from the client and service contracts. In some sense, the orchestrator acts as an "adapter" between the client and the service. In our approach, the orchestrator behaves as a "planner" which finds a suitable set of contracts and puts in a session all the participants involved in these contracts.

Our approach differs from those discussed above, as well as from all the other approaches we are aware of (e.g. [7, 12, 13]), w.r.t. two general principles. First, we depart from the common principle that contracts are always respected after their stipulation. We represent instead the more realistic situation where promises are not always maintained. As a consequence, in CO_2 we do not discard contracts after they have been used to couple services and put them in a session, as done e.g. in all the approaches dealing with behavioural types. In our approach, contracts are also used to drive computations after sessions have been established (cf.§ 3), e.g. to detect violations and to provide the agreed compensations.

The second general difference is that CO_2 smoothly allows for handling contracts-as-processes (cf. § 2.1). To the best of our knowledge, it seems hard to accommodate these contracts within frameworks based on constraint systems ([5, 15]), logics ([4, 6]), or c-semirings (e.g. [9, 8, 18]).

The contract-as-formulae model proposed here hinges on PCL 's contractual implication to handle the intrinsic circularity present in contracts. This is motivated by the fact that the standard intuitionistic implication would fall short in modelling (circular) require-guarantee conditions to participants in distributed systems. Some relations between PCL and other logics are reported in [3].

A recent research direction is modelling contracts as formulae in suitable deontic logics [26]. We argue that those approaches are complementary to ours. On the one hand, deontic logics are more suitable than PCL to model the dynamic execution of contracts (e.g., to assign blame to participants

and to provide reparations to contract violations). On the other hand, PCL targets more directly the problem of finding agreements, especially in presence of circular dependencies among the participant requirements.

The present paper extends and enhances prior work appeared in [2, 4, 5]. The calculi presented in [4, 5] were hard-wired to the PCL contract model, and they did not explicitly keep apart promises from facts. Instead, CO_2 distinguishes between promises (latent contracts, advertised through a tell prefix) and facts (actions performed through a do prefix), so allowing us to give a general definition of when a participant is honest in a given context (Definition 3.5). Also, the calculi in [4, 5] recorded contracts into a global constraint store, while CO_2 has local environments for participants and sessions. This makes CO_2 amenable to possible distributed implementations.

With respect to [2], the main improvements concern the contracts-as-processes model, and its relation with contracts-as-formulae. More precisely, in the current paper (*i*) contracts explicitly mention the participants from which they expect to receive an input / send an output, (*ii*) outputs are asynchronous, using an implicit buffer with unbounded capacity which preserves no ordering among outputs, and (*iii*) there is a stronger correspondence result between contracts-as-formulae and contracts-as-processes.

The feature (*i*) allows for a finer-grain design of contracts-as-processes. For instance, instead of just requiring "if someone pays, then I will ship", explicit senders/receivers allow for modelling the contract "if B pays, then I will ship to B", specified formally as $A \; \mathit{says} \; B\langle \mathsf{pay}.\overline{\mathsf{ship}}\rangle B$. Also, (*i*) helps in improving the precision of the correspondence between contracts-as-processes and $1N$-PCL contracts, which natively feature explicit senders through the *says* in the antecedent of $\rightarrow/\twoheadrightarrow$.

The feature (*ii*) allows for a definition of culpability which is fairer w.r.t. that in [2]. For instance, assume that A promises to pay (i.e. she outputs pay), while B promises to receive the payment (i.e. he inputs pay). Consider the situation were A exposes the output prefix of pay, but B does not expose the corresponding input prefix. In [2], communication is synchronous, and a participant is culpable until it reaches the state 0: therefore, both A and B are considered culpable. This is quite unfair for A, who after all has attempted to fulfil her duties. Also the converse situation, where A has not paid and B is stuck on the input, is unfair in [2]: this time B is considered culpable, even if he is just waiting that A has fulfilled her duties. In the current paper, instead, when A wants to pay she can do it asynchronously; B will be culpable until he eventually performs the input. Conversely, when

A has not paid and B is stuck on the input, B will not be culpable.

The correspondence result between contracts-as-formulae and contracts-as-processes featured in this paper is more precise than the result in [2]. The encoding in [2] guarantees that a $1N$-PCL contract c logically entails *all* the atoms contained in c iff its encoding $[c]$ can reach the state 0. The encoding presented here (Definition 2.5) guarantees a stronger result. Roughly, a $1N$-PCL contract c logically entails *some* atom a iff its encoding $[c]$ has a trace containing a, and leading to a state where all participants are not culpable (Theorem 2.7).

While taking inspiration from Concurrent Constraint Programming, CO_2 makes use of more concrete communication primitives which do not assume a global constraint store, so reducing the gap towards a possible distributed implementation. The main differences between CO_2 and CCP are: (i) in CO_2 constraints are contracts, (ii) in CO_2 there is no global store of constraints: all the prefixes act on a local environment, (iii) the prefix ask of CO_2 may instantiate variables to names, and (iv) the prefixes do (which makes contracts evolve) and fuse (which establishes a new session) have no counterpart in CCP.

5 Conclusions

We have developed a formal model for reasoning about contract-oriented distributed programs. The overall contribution of this paper is a contract calculus (CO_2) that is parametric in the choice of contract model. In CO_2, participants can advertise their own contracts, find other participants with compatible contracts, and establish a new multi-party session with those which comply with the global contract. We have set out two crucial issues: how to reach agreements, and how to detect violations. We have presented two concretisations of the abstract contract model. The first is an instance of the contracts-as-processes paradigm, while the second is an instance of the contracts-as-processes paradigm.

We envision a methodology where one uses contract-as-formulae at design-time, reasons about them through the contract logic, and then concretises them at run-time to contracts-as-processes. As a first step towards this goal, we have related contract-as-processes and contracts-as-formulae, by devising a mapping from contracts based on the logic PCL [4] into CCS-like contracts. Theorem 2.7 guarantees that contracts-as-processes can reach success in those cases where an agreement would be possible in the logic

model, hence providing a connection between the two worlds.

The correspondence result established by Theorem 2.7 may be refined in two directions. First, while the goal of the current encoding is to preserve the entailment, one could also focus on precisely preserving the culpability of participants. Note e.g. that the encoding of B *says* (A *says* a) $\twoheadrightarrow \mathsf{b}$ in Definition 2.5 makes it possible to perform b "on credit". However, this results in B being culpable because of the $DEBT(\mathsf{B}, \mathsf{b})$ process. In order to precisely preserve culpability, the encoding should instead make A culpable, since she has to "pay" the debt. Still, the current encoding preserves a weak form of culpability, i.e. the fact that some participant is culpable. This is enough to establish the correspondence in Theorem 2.7.

A second direction for refining the encoding is to preserve some correspondence property in *all* the traces of the contract-as-process. This seems to suggest to exploit reversibility techniques, i.e. by assuming that all the actions can be undone, and then by undoing those traces which do not lead to a successful state. Note in passing that Theorem 2.7 guarantees that at least one property-preserving trace exists. Yet, this is enough to preserve agreements. For instance, assume that $\mathsf{fuse}\,(\mathsf{A}$ *says* $\mathsf{a})$ finds an agreement in a PCL contract c. Then, it is possible to reach an agreement also with $[c]$. This is obtained through a $\mathsf{fuse}\,\phi$, where ϕ requires that there exists some trace of $[c]$ where (i) A *does* $\overline{\mathsf{a}}\rangle\mathsf{B}$ eventually holds (for some B), and after that (ii) no participant is culpable. Such ϕ can be expressed e.g. in the branching temporal logic EF.

Acknowledgements

We thank the anonymous reviewers for their valuable suggestions and comments which helped us in improving the paper.

This work has been partially supported by the Aut. Region of Sardinia under grants L.R.7/2007 CRP2-120 (Project TESLA) and CRP-17285 (Project TRICS), and by the Leverhulme Trust Programme Award "Tracing Networks".

References

[1] Alexander Artikis, Marek J. Sergot, and Jeremy V. Pitt. Specifying norm-governed computational societies. *ACM Trans. Comput. Log.*, 10(1), 2009.

[2] Massimo Bartoletti, Emilio Tuosto, and Roberto Zunino. Contracts in distributed systems. In *ICE*, volume 59 of *EPTCS*, pages 130–147, 2011.

[3] Massimo Bartoletti and Roberto Zunino. A logic for contracts. Technical Report DISI-09-034, DISI - Università di Trento, 2009.

[4] Massimo Bartoletti and Roberto Zunino. A calculus of contracting processes. In *LICS*, pages 332–341. IEEE Computer Society, 2010.

[5] Massimo Bartoletti and Roberto Zunino. Primitives for contract-based synchronization. In *ICE*, volume 38 of *EPTCS*, pages 67–82, 2010.

[6] Laura Bocchi, Kohei Honda, Emilio Tuosto, and Nobuko Yoshida. A theory of design-by-contract for distributed multiparty interactions. In *CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 2010.

[7] Mario Bravetti and Gianluigi Zavattaro. Towards a unifying theory for choreography conformance and contract compliance. In *Software Composition*, volume 4829 of *Lecture Notes in Computer Science*, pages 34–50. Springer, 2007.

[8] Maria Grazia Buscemi and Hernán C. Melgratti. Transactional service level agreement. In *TGC*, volume 4912 of *Lecture Notes in Computer Science*, pages 124–139. Springer, 2007.

[9] Maria Grazia Buscemi and Ugo Montanari. CC-Pi: A constraint-based language for specifying service level agreements. In *ESOP*, volume 4421 of *Lecture Notes in Computer Science*, pages 18–32. Springer, 2007.

[10] Felice Cardone. The geometry and algebra of commitment. In *Ludics, dialogue and interaction*, pages 147–160. Springer, 2011.

[11] Samuele Carpineti, Giuseppe Castagna, Cosimo Laneve, and Luca Padovani. A formal account of contracts for web services. In *WS-FM*, volume 4184 of *Lecture Notes in Computer Science*, pages 148–162. Springer, 2006.

[12] Samuele Carpineti and Cosimo Laneve. A basic contract language for web services. In *ESOP*, volume 3924 of *Lecture Notes in Computer Science*, pages 197–213. Springer, 2006.

[13] Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A theory of contracts for web services. *ACM Transactions on Programming Languages and Systems*, 31(5), 2009.

[14] Giuseppe Castagna and Luca Padovani. Contracts for mobile processes. In *Proc. CONCUR*, volume 5710 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2009.

[15] Mario Coppo and Mariangiola Dezani-Ciancaglini. Structured communications with concurrent constraints. In *TGC*, volume 5474 of *Lecture Notes in Computer Science*, pages 104–125. Springer, 2008.

[16] Mads Dam. On the Decidability of Process Equivalences for the π-calculus. *Theoretical Computer Science*, 183(2):215–228, 1997.

[17] E. Allen Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 995–1072. North-Holland Pub. Co./MIT Press, 1990.

[18] Gian Luigi Ferrari and Alberto Lluch-Lafuente. A logic for graphs with QoS. *Electr. Notes Theor. Comput. Sci.*, 142:143–160, 2006.

[19] Deepak Garg and Martín Abadi. A modal deconstruction of access control logics. In *FoSSaCS*, volume 4962 of *Lecture Notes in Computer Science*, pages 216–230. Springer, 2008.

[20] Thomas Hobbes. *Leviathan or The Matter, Forme and Power of a Common Wealth Ecclesiasticall and Civil*. 1651. Chapter XIV — Of the first and second natural laws, and of contracts.

[21] Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP*, volume 1381, 1998.

[22] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *POPL*, pages 273–284. ACM, 2008.

[23] Ashley T. McNeile. Protocol contracts with application to choreographed multiparty collaborations. *Service Oriented Computing and Applications*, 4(2):109–136, 2010.

[24] Robin Milner. *Communication and concurrency*. Prentice-Hall, Inc., 1989.

[25] Cristian Prisacariu and Gerardo Schneider. A formal language for electronic contracts. In *FMOODS*, volume 4468 of *Lecture Notes in Computer Science*, pages 174–189. Springer, 2007.

[26] Cristian Prisacariu and Gerardo Schneider. A dynamic deontic logic for complex contracts. *The Journal of Logic and Algebraic Programming (JLAP)*, 81(4):458–490, 2012.

[27] Vijay Saraswat, Prakash Panangaden, and Martin Rinard. Semantic foundations of concurrent constraint programming. In *POPL*, pages 333–352, 1991.

[28] Anne Troelstra and Dirk van Dalen. *Constructivism in Mathematics, vol. 1.* North-Holland, 1988.

A Proofs

Notation. Hereafter, for c, c' contracts-as-formulae, we write c, c' for $c \wedge c'$.

Definition A.1. *The Gentzen-style sequent calculus of* PCL *is defined by the rules in Figure 8.*

As proved in [4], the sequent calculus of PCL enjoys cut elimination. A cut on a formula p is replaced by cuts on strict subformulae of p, and cuts on p having a shorter proof tree. This makes PCL decidable.

Theorem A.2 (Cut Elimination [4])**.** *If p is provable in* PCL*, then there exists a proof of p not using the* (Cut) *rule.*

Definition A.3. *Let u be the homomorphic function from* PCL *formulae to* PCL *formulae such that $u(A \text{ says } p) = u(p)$. We say that a formula c is says-free $1N$-PCL when there exists some c' in $1N$-PCL such that $c = u(c')$.*

Lemma A.4. *For all says-free $1N$-PCL formulae c, and for $\circ \in \{\to, \twoheadrightarrow\}$:*

$$c, \mathsf{a} \circ \mathsf{b} \vdash \mathsf{c} \implies c \vdash \mathsf{c} \ \vee \ c, \mathsf{a} \circ \mathsf{b} \vdash \mathsf{a}, \mathsf{b}, \mathsf{c}$$

Proof. By Theorem A.2, consider a proof tree Δ of $c, \mathsf{a} \circ \mathsf{b} \vdash \mathsf{c}$ without occurrences of the (Cut) rule. The RHS of each sequent in Δ has the form $\bigwedge_{i \in I} \mathsf{a}_i$, and so Δ only contains occurrences of the rules (Id), ($\wedge$L1), ($\wedge$L2), ($\wedge$R), ($\to$L), (Fix). There are two cases, according to the choice for $\circ$.

- $\circ = \to$. If the rule ($\to$L) has never been used in Δ, consider the proof tree Δ' obtained by replacing each $\Gamma, \mathsf{a} \to \mathsf{b} \vdash p$ in Δ with $\Gamma \vdash p$. Since there is no other rule in those mentioned above which can use $\to$ in the LHS, then Δ' proves $c \vdash \mathsf{c}$.

  Otherwise, if Δ contains an occurrence of the rule ($\to$L), then one of its premises must be $c, \mathsf{a} \to \mathsf{b} \vdash \mathsf{a}$. By the rule ($\to$L), we have:

  $$\frac{c, \mathsf{a} \to \mathsf{b} \vdash \mathsf{a} \qquad c, \mathsf{a} \to \mathsf{b}, \mathsf{b} \vdash \mathsf{b}}{c, \mathsf{a} \to \mathsf{b} \vdash \mathsf{b}}$$

- $\circ = \twoheadrightarrow$. If the rule (Fix) has never been used in Δ, consider the proof tree Δ' obtained by replacing each $\Gamma, \mathsf{a} \twoheadrightarrow \mathsf{b} \vdash p$ in Δ with $\Gamma \vdash p$. Since

$$\frac{}{\Gamma, p \vdash p}\ (\textsc{Id}) \qquad \frac{\Gamma \vdash p \quad \Gamma,\ p \vdash q}{\Gamma \vdash q}\ (\textsc{Cut})$$

$$\frac{\Gamma,\ p \wedge q,\ p \vdash r}{\Gamma,\ p \wedge q \vdash r}\ (\wedge\text{L1}) \qquad \frac{\Gamma,\ p \wedge q,\ q \vdash r}{\Gamma,\ p \wedge q \vdash r}\ (\wedge\text{L2}) \qquad \frac{\Gamma \vdash p \quad \Gamma \vdash q}{\Gamma \vdash p \wedge q}\ (\wedge\text{R})$$

$$\frac{\Gamma, p \vee q, p \vdash r \quad \Gamma, p \vee q, q \vdash r}{\Gamma, p \vee q \vdash r}\ (\vee\text{L})$$

$$\frac{\Gamma \vdash p}{\Gamma \vdash p \vee q}\ (\vee\text{R1}) \qquad \frac{\Gamma \vdash q}{\Gamma \vdash p \vee q}\ (\vee\text{R2})$$

$$\frac{\Gamma,\ p \to q \vdash p \quad \Gamma,\ p \to q,\ q \vdash r}{\Gamma,\ p \to q \vdash r}\ (\to\text{L}) \qquad \frac{\Gamma,\ p \vdash q}{\Gamma \vdash p \to q}\ (\to\text{R})$$

$$\frac{\Gamma, \neg p \vdash p}{\Gamma, \neg p \vdash r}\ (\neg\text{L}) \qquad \frac{\Gamma, p \vdash \bot}{\Gamma \vdash \neg p}\ (\neg\text{R})$$

$$\frac{}{\Gamma, \bot \vdash p}\ (\bot\text{L}) \qquad \frac{}{\Gamma \vdash \top}\ (\top\text{R}) \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash p}\ (\textsc{WeakR})$$

$$\frac{\Gamma \vdash q}{\Gamma \vdash p \twoheadrightarrow q}\ (\textsc{Zero})$$

$$\frac{\Gamma,\ p \twoheadrightarrow q,\ a \vdash p \quad \Gamma,\ p \twoheadrightarrow q,\ q \vdash b}{\Gamma,\ p \twoheadrightarrow q \vdash a \twoheadrightarrow b}\ (\textsc{PrePost})$$

$$\frac{\Gamma,\ p \twoheadrightarrow q,\ r \vdash p \quad \Gamma,\ p \twoheadrightarrow q,\ q \vdash r}{\Gamma,\ p \twoheadrightarrow q \vdash r}\ (\textsc{Fix})$$

$$\frac{\Gamma \vdash p}{\Gamma \vdash A\ \mathit{says}\ p}\ (\textsc{SaysR}) \qquad \frac{\Gamma,\ A\ \mathit{says}\ p,\ p \vdash A\ \mathit{says}\ q}{\Gamma, A\ \mathit{says}\ p \vdash A\ \mathit{says}\ q}\ (\textsc{SaysL})$$

Figure 8: Genzten-style proof system for PCL.

there is no other rule in those mentioned above which can use $\twoheadrightarrow$ in the LHS, then Δ' proves $c \vdash \mathsf{c}$.

Otherwise, if there exists an occurrence of the rule (Fix), then its premise contains the entailment $c, \mathsf{a} \twoheadrightarrow \mathsf{b}, r \vdash \mathsf{a}$, for some r. By case analysis on the rules (Id), ($\wedge$L1), ($\wedge$L2), ($\wedge$R), ($\rightarrow$L), (Fix), it follows that, for all Γ, Γ', p, q, if $\Gamma \vdash p$ and $\Gamma' \vdash q$ belong to Δ, then Γ and Γ' are logically equivalent. Therefore, $c, \mathsf{a} \twoheadrightarrow \mathsf{b}, r$ is logically equivalent to $c, \mathsf{a} \twoheadrightarrow \mathsf{b}$, that is to say $c, \mathsf{a} \twoheadrightarrow \mathsf{b} \vdash \mathsf{a}$. To conclude, by the rule (Fix):

$$\frac{c, \mathsf{a} \twoheadrightarrow \mathsf{b}, \mathsf{b} \vdash \mathsf{a} \qquad c, \mathsf{a} \twoheadrightarrow \mathsf{b}, \mathsf{b} \vdash \mathsf{b}}{c, \mathsf{a} \twoheadrightarrow \mathsf{b} \vdash \mathsf{b}}$$

□

Definition A.5. *Let $\pi : \mathcal{A} \to \mathcal{N}_P \cup \mathcal{V}_P$. We define the partial function $s_\pi()$ from unsigned says-free $1N$-PCL formulae to $1N$-PCL formulae as follows:*

$$s_\pi(\bigwedge_{i\in\mathcal{I}} \gamma_i) = \bigwedge_{i\in\mathcal{I}} P_\pi(\gamma_i) \textit{ says } \gamma_i$$

$$s_\pi(\gamma) = \begin{cases} \bigwedge_{j\in\mathcal{J}} \pi(\mathsf{a}_j) \textit{ says } \mathsf{a}_j & \textit{if } \gamma = \bigwedge_{j\in\mathcal{J}} \mathsf{a}_j \\ P_\pi(\gamma) \textit{ says } \left(s_\pi(\bigwedge_{i\in\mathcal{I}} \mathsf{b}_i) \circ \bigwedge_{j\in\mathcal{J}} \mathsf{a}_j\right) & \textit{if } \gamma = (\bigwedge_{i\in\mathcal{I}} \mathsf{b}_i) \circ \bigwedge_{j\in\mathcal{J}} \mathsf{a}_j \end{cases}$$

where $\circ \in \{\rightarrow, \twoheadrightarrow\}$, and $P_\pi(\gamma)$ is defined as follows:

$$P_\pi(\bigwedge_{j\in\mathcal{J}} \mathsf{a}_j) \;=\; \begin{cases} A & \textit{if } \forall j \in \mathcal{J}.\ \pi(\mathsf{a}_j) = A \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

$$P_\pi((\bigwedge_{i\in\mathcal{I}} \mathsf{b}_i) \circ \bigwedge_{j\in\mathcal{J}} \mathsf{a}_j) = P_\pi(\bigwedge_{j\in\mathcal{J}} \mathsf{a}_j)$$

Lemma A.6. *For all* PCL *formulae p, q, if $p \vdash q$ then $u(p) \vdash u(q)$.*

Proof. Straightforward by induction on the derivation of $p \vdash q$. □

Lemma A.7. *Let $\pi : \mathcal{A} \to \mathcal{N}_P \cup \mathcal{V}_P$. For all says-free $1N$-PCL formulae c and $a = \bigwedge_{i\in I} \mathsf{a}_i$ such that $s_\pi(c)$ is defined:*

$$c \vdash a \iff s_\pi(c) \vdash s_\pi(a)$$

Proof. To prove the ($\Rightarrow$) direction, assume that $c \vdash a$. By Theorem A.2, consider a proof tree Δ of $c \vdash a$ without occurrences of the (Cut) rule. The RHS of each sequent in Δ has the form $\bigwedge_{i\in J} \mathsf{a}_i$, with $J \subseteq I$, and so Δ only contains occurrences of the rules (Id), ($\wedge$L1), ($\wedge$L2), ($\wedge$R), ($\rightarrow$L), (Fix). We have the following cases, according to the last rule used in Δ:

- (Id). We have that $c = c', a$, and rule (Id) has been instantiated as:

$$\frac{}{c', a \ \vdash \ a}$$

  The thesis follows trivially by $s_\pi(a) \vdash s_\pi(a)$.

- ($\wedge$L1). We have that $c = p \wedge q$, and the instance of rule ($\wedge$L1) is:

$$\frac{c', p \wedge q, p \ \vdash \ a}{c', p \wedge q \ \vdash \ a}$$

  By the induction hypothesis,

$$s_\pi(c'), s_\pi(p \wedge q), s_\pi(p) \ \vdash \ s_\pi(a)$$

  Since $s_\pi(p \wedge q) \leftrightarrow s_\pi(p) \wedge s_\pi(q)$, by ($\wedge$L1) we conclude:

$$\frac{s_\pi(c'), s_\pi(p) \wedge s_\pi(q), s_\pi(p) \ \vdash \ s_\pi(a)}{s_\pi(c'), s_\pi(p) \wedge s_\pi(q) \ \vdash \ s_\pi(a)}$$

  from which the thesis follows by using the same observation.

- ($\wedge$L2). Similar to the previous case.

- ($\wedge$R). We have that $a = p \wedge q$, and the instance of rule ($\wedge$R) is:

$$\frac{c \ \vdash \ p \quad c \ \vdash \ q}{c \ \vdash \ p \wedge q}$$

  By the induction hypothesis (twice), $s_\pi(c) \vdash s_\pi(p)$ and $s_\pi(c) \vdash s_\pi(q)$. Therefore, by ($\wedge$R):

$$\frac{s_\pi(c) \vdash s_\pi(p) \quad s_\pi(c) \vdash s_\pi(q)}{s_\pi(c) \vdash s_\pi(p) \wedge s_\pi(q)}$$

  and the thesis follows because $s_\pi(p) \wedge s_\pi(q) \leftrightarrow s_\pi(p \wedge q) = s_\pi(a)$.

- ($\to$L). We have that $c = c', p \to q$, with $p = \bigwedge_i \mathsf{b}_i$ and $q = \bigwedge_j \mathsf{a}_j$. The instance of rule ($\to$L) is:

$$\frac{c', p \to q \ \vdash \ p \quad c', p \to q, q \ \vdash \ a}{c', p \to q \ \vdash \ a}$$

  By the induction hypothesis, applied twice:

$$s_\pi(c'), s_\pi(p \to q) \ \vdash \ s_\pi(p) \tag{5}$$
$$s_\pi(c'), s_\pi(p \to q), s_\pi(q) \ \vdash \ s_\pi(a) \tag{6}$$

  By Definition A.5,

$$s_\pi(p \to q) = P_\pi(p \to q) \ says \ \left(s_\pi(p) \to \bigwedge_j \mathsf{a}_j\right)$$

  Since $P_\pi(q) = P_\pi(p \to q)$ is defined by hypothesis, and since

$$\begin{array}{rcl} A \ says \ (p' \to q) & \to & (p' \to A \ says \ q) \\ A \ says \ \bigwedge_j \mathsf{a}_j & \leftrightarrow & \bigwedge_j A \ says \ \mathsf{a}_j \end{array}$$

  are theorems in PCL , we have that:

$$s_\pi(c'), s_\pi(p \to q) \ \vdash \ s_\pi(p) \to s_\pi(q) \tag{7}$$

  By (7) and (5), it follows that:

$$s_\pi(c'), s_\pi(p) \to s_\pi(q) \ \vdash \ s_\pi(q)$$

  Together with (6), we conclude that:

$$s_\pi(c'), s_\pi(p \to q) \ \vdash \ s_\pi(a)$$

- (Fix). We have that $c = c', p \twoheadrightarrow q$, with $p = \bigwedge_i \mathsf{b}_i$ and $q = \bigwedge_j \mathsf{a}_j$. The instance of rule (Fix) is:

$$\frac{c', p \twoheadrightarrow q, a \ \vdash \ p \quad c', p \twoheadrightarrow q, q \ \vdash \ a}{c', p \twoheadrightarrow q \ \vdash \ a}$$

By the induction hypothesis, applied twice:

$$s_\pi(c'), s_\pi(p \twoheadrightarrow q), s_\pi(a) \ \vdash \ s_\pi(p) \tag{8}$$
$$s_\pi(c'), s_\pi(p \twoheadrightarrow q), s_\pi(q) \ \vdash \ s_\pi(a) \tag{9}$$

By Definition A.5,

$$s_\pi(p \twoheadrightarrow q) = P_\pi(p \twoheadrightarrow q) \ \mathit{says} \ \big(s_\pi(p) \twoheadrightarrow \bigwedge_j \mathsf{a}_j\big)$$

Since $P_\pi(q) = P_\pi(p \twoheadrightarrow q)$ is defined by hypothesis, and since

$$A \ \mathit{says} \bigwedge_j \mathsf{a}_j \quad \leftrightarrow \quad \bigwedge_j A \ \mathit{says} \ \mathsf{a}_j \tag{10}$$

is a theorem in PCL, we have that $q \vdash s_\pi(q)$, and so by (9):

$$s_\pi(c'), s_\pi(p \twoheadrightarrow q), q \ \vdash \ s_\pi(a) \tag{11}$$

By (9) and (11), it follows that:

$$s_\pi(c'), s_\pi(p \twoheadrightarrow q), q \ \vdash \ s_\pi(p) \tag{12}$$

By Definition A.5, $s_\pi(p \twoheadrightarrow q) = P_\pi(q) \ \mathit{says} \ (s_\pi(p) \twoheadrightarrow q)$. Now, since $p \to A \ \mathit{says} \ p$ is a theorem in PCL, by (12) it follows that:

$$s_\pi(c'), s_\pi(p) \twoheadrightarrow q, q \ \vdash \ s_\pi(p) \tag{13}$$

Therefore, by rule (SaysL):

$$\dfrac{\dfrac{\dfrac{(13) \qquad \dfrac{}{\cdots, s_\pi(p) \twoheadrightarrow q, q \ \vdash \ q}(\textsc{Id})}{\cdots, s_\pi(p) \twoheadrightarrow q \ \vdash \ q}(\textsc{Fix})}{\cdots, P_\pi(q) \ \mathit{says} \ (s_\pi(p) \twoheadrightarrow q), s_\pi(p) \twoheadrightarrow q \ \vdash \ P_\pi(q) \ \mathit{says} \ q}(\textsc{SaysR})}{s_\pi(c'), P_\pi(q) \ \mathit{says} \ (s_\pi(p) \twoheadrightarrow q) \ \vdash \ P_\pi(q) \ \mathit{says} \ q} \tag{14}$$

Now, since $P_\pi(q) = P_\pi(p \to q)$ is defined and because of (10), we have that $P_\pi(q) \ \mathit{says} \ q \leftrightarrow s_\pi(q)$. Also, by Definition A.5,

$$P_\pi(q) \ \mathit{says} \ (s_\pi(p) \twoheadrightarrow q) \ = \ s_\pi(p \twoheadrightarrow q)$$

We can then rewrite (14) as:

$$s_\pi(c'), s_\pi(p \twoheadrightarrow q) \;\vdash\; s_\pi(q)$$

Together with (9), this gives the thesis:

$$s_\pi(c'), s_\pi(p \twoheadrightarrow q) \;\vdash\; s_\pi(a)$$

To prove the ($\Leftarrow$) direction, assume that $s_\pi(c) \vdash s_\pi(\mathsf{a})$. By Lemma A.6, we have that $u(s_\pi(c)) \vdash u(s_\pi(\mathsf{a}))$. The thesis follows by noting that $u(s_\pi(c)) = c$, and $u(s_\pi(\mathsf{a})) = \mathsf{a}$. □

Definition A.8. *For all traces η of a contract-as-process c, we define the set of* $1N$-PCL *actions $act(\eta)$ as follows:*

$$act(\eta) = \begin{cases} \emptyset & \text{if } \eta \text{ is empty} \\ act(\eta') & \text{if } \eta = (A \text{ says } B\rangle\mathsf{a})\,\eta' \\ \{A \text{ says } \mathsf{a}\} \cup act(\eta') & \text{if } \eta = (A \text{ says } \overline{\mathsf{a}}\rangle B)\,\eta' \end{cases}$$

Notation. Hereafter, to indicate any contract c such that $\mathsf{A} \smile c$ for all A, we sometimes just write $\smile$. We write $A\rangle\mathsf{a}\langle$ for a pending message where the receiver is immaterial.

Lemma A.9. *For all* $1N$-PCL *contracts c, if*

$$[c] \;\mid\; \big|\big|_{i\in I} DEBT(B_i, \mathsf{b}_i) \;\mid\; \big|\big|_{j\in J} A_j\rangle\mathsf{a}_j\langle \quad \xrightarrow{\eta}\!\!\!\!\twoheadrightarrow \quad \smile$$

then

$$c,\ \bigwedge_{j\in J} A_j \text{ says } \mathsf{a}_j \;\vdash\; act(\eta),\ \bigwedge_{i\in I} B_i \text{ says } \mathsf{b}_i$$

Proof. We proceed by induction on the length of η.

To keep the proof succinct, we make some mild simplifying assumptions on the form of the contract c. More precisely, we assume that in conjunctions $\bigwedge_{j\in\mathcal{J}} \mathsf{a}_j$, the set $\mathcal{J}$ is always a singleton (we do not make this assumption for the antecedent of $\rightarrow$ and $\twoheadrightarrow$). The proof for the general case can be easily obtained from the current one, at the cost of some extra details in the management of conjunctions.

In the base case η is empty, and so $act(\eta) = \emptyset$. Since, by hypothesis, $A \smile \big|\big|_{i\in I} DEBT(B_i, \mathsf{b}_i)$ for all A, then it must be the case that $I = \emptyset$: by

contradiction, if this were not the case and $i \in I$, we would have that the participant who says $DEBT(B_i, \mathsf{b}_i) = \tau.DEBT(B_i, \mathsf{b}_i) + \cdots$ is culpable. Therefore, the base case requires to prove that $c, \cdots \vdash \emptyset$, which holds trivially.

For the inductive case, let

$$c_0 \;=\; [c] \mid \big|\big|_{i\in I} DEBT(B_i, \mathsf{b}_i) \mid \big|\big|_{j\in J} A_j\rangle\mathsf{a}_j\rangle$$

and assume a trace:

$$c_0 \xrightarrow{A \textit{ says } \alpha}\!\!\!\!\rightarrow\; c_1 \xrightarrow{\eta'}\!\!\!\!\rightarrow\; \circlearrowleft$$

where $\eta = (A \textit{ says } \alpha)\eta'$. We have the following exhaustive cases, obtained by analysing the possible choices for the first transition.

1. $c = c', A \textit{ says } \mathsf{a}$, transition by the rule (Tau) on $[\mathsf{a}] = OUT(\mathsf{a})$. Then:

   $$c_0 \xrightarrow{A \textit{ says } \tau}\!\!\!\!\rightarrow\; [c'] \mid \big|\big|_{i\in I} DEBT(B_i, \mathsf{b}_i) \mid \big|\big|_{j\in J} A_j\rangle\mathsf{a}_j\rangle$$

   The thesis follows directly from the induction hypothesis.

2. $c = c', A \textit{ says } \mathsf{a}$, transition by the rule (Out) on $[\mathsf{a}] = OUT(\mathsf{a})$. Then, $\alpha = \overline{\mathsf{a}}\rangle B$ for some B, and:

   $$\begin{aligned} c_0 \xrightarrow{A \textit{ says } \overline{\mathsf{a}}\rangle B}\!\!\!\!\rightarrow\; & [c'] \mid OUT(\mathsf{a}) \mid A\rangle\overline{\mathsf{a}}\rangle B \mid \big|\big|_{i\in I} DEBT(B_i, \mathsf{b}_i) \mid \big|\big|_{j\in J} A_j\rangle\mathsf{a}_j\rangle \\ =\; & [c', \mathsf{a}] \mid A\rangle\overline{\mathsf{a}}\rangle B \mid \big|\big|_{i\in I} DEBT(B_i, \mathsf{b}_i) \mid \big|\big|_{j\in J} A_j\rangle\mathsf{a}_j\rangle \end{aligned}$$

   By the induction hypothesis, it follows that:

   $$c',\; A \textit{ says } \mathsf{a},\; \bigwedge_{j\in J} A_j \textit{ says } \mathsf{a}_j \;\vdash\; act(\eta'),\; \bigwedge_{i\in I} B_i \textit{ says } \mathsf{b}_i \qquad (15)$$

   By the rule (Id), $c \vdash A \textit{ says } \mathsf{a}$. Then:

   $$c,\; \bigwedge_{j\in J} A_j \textit{ says } \mathsf{a}_j \;\vdash\; A \textit{ says } \mathsf{a}, act(\eta'),\; \bigwedge_{i\in I} B_i \textit{ says } \mathsf{b}_i$$

   and to conclude it suffices to note that $act(\eta) = \{A \textit{ says } \mathsf{a}\} \cup act(\eta')$.

3. $c = c', (\bigwedge_{h\in 1..n} D_h \textit{ says } \mathsf{d}_h) \to \mathsf{b}$, transition by the rule (In) on the leftmost input of $[(\bigwedge_{h\in 1..n} D_h \textit{ says } \mathsf{d}_h) \to \mathsf{b}]$. This requires that a suitable pending message is available in c_0, i.e. there exists $j_1 \in J$ such that $A_{j_1}\rangle\mathsf{a}_{j_1}\rangle = D_1\rangle\mathsf{d}_1\rangle A$. There are two subcases.

- if $n = 1$, then:

$$\begin{aligned} c_0 \xrightarrow{A\ says\ D_1\langle \mathsf{d}_1} & [c'] \mid OUT(\mathsf{b}) \mid \big|\big|_{i\in I} DEBT(B_i, \mathsf{b}_i) \mid \big|\big|_{j\in J\setminus\{j_1\}} A_j\langle \mathsf{a}_j\rangle \\ = & [c', \mathsf{b}] \mid \big|\big|_{i\in I} DEBT(B_i, \mathsf{b}_i) \mid \big|\big|_{j\in J\setminus\{j_1\}} A_j\langle \mathsf{a}_j\rangle \end{aligned}$$

  By the induction hypothesis, we have that:

$$c',\ \mathsf{b},\ \bigwedge_{j\in J\setminus\{j_1\}} A_j\ says\ \mathsf{a}_j \ \vdash\ act(\eta'),\ \bigwedge_{i\in I} B_i\ says\ \mathsf{b}_i$$

  The above entailment is weakened to:

$$c',\ (D_1\ says\ \mathsf{d}_1) \to \mathsf{b},\ D_1\ says\ \mathsf{d}_1,\ \bigwedge_{j\in J\setminus\{j_1\}} A_j\ says\ \mathsf{a}_j \ \vdash\ \cdots$$

  Now, since $A_{j_1} = D_1$ and $\mathsf{a}_{j_1} = \mathsf{d}_1$, the above is equivalent to:

$$c',\ (D_1\ says\ \mathsf{d}_1) \to \mathsf{b},\ \bigwedge_{j\in J} A_j\ says\ \mathsf{a}_j \ \vdash\ \cdots$$

  from which the thesis follows, because $act(\eta) = act(\eta')$.

- if $n > 1$, then:

$$\begin{aligned} c_0 \xrightarrow{A\ says\ D_1\langle \mathsf{d}_1} & [c'] \mid \big|\big|_{i\in I} DEBT(B_i, \mathsf{b}_i) \mid \\ & D_2\langle \mathsf{d}_2.\cdots.D_n\langle \mathsf{d}_n. \mid \big|\big|_{j\in J\setminus\{j_1\}} A_j\langle \mathsf{a}_j\rangle \\ = & [c', (\textstyle\bigwedge_{h\in 2..n} D_h\ says\ \mathsf{d}_h) \to \mathsf{b}] \mid \\ & \big|\big|_{i\in I} DEBT(B_i, \mathsf{b}_i) \mid \big|\big|_{j\in J\setminus\{j_1\}} A_j\langle \mathsf{a}_j\rangle \end{aligned}$$

  By the induction hypothesis, we have that:

$$c',\ (\bigwedge_{h\in 2..n} D_h\ says\ \mathsf{d}_h) \to \mathsf{b},\ \bigwedge_{j\in J\setminus\{j_1\}} A_j\ says\ \mathsf{a}_j \ \vdash\ act(\eta'),\ \bigwedge_{i\in I} B_i\ says\ \mathsf{b}_i$$

  The above entailment can be weakened as follows:

$$c',\ (\bigwedge_{h\in 2..n} D_h\ says\ \mathsf{d}_h) \to \mathsf{b},\ D_1\ says\ \mathsf{d}_1,\ \bigwedge_{j\in J\setminus\{j_1\}} A_j\ says\ \mathsf{a}_j \ \vdash\ \cdots$$

  which is equivalent to:

$$c',\ (\bigwedge_{h\in 1..n} D_h\ says\ \mathsf{d}_h) \to \mathsf{b},\ D_1\ says\ \mathsf{d}_1,\ \bigwedge_{j\in J\setminus\{j_1\}} A_j\ says\ \mathsf{a}_j \ \vdash\ \cdots$$

Now, since $A_{j_1} = D_1$ and $\mathsf{a}_{j_1} = \mathsf{d}_1$, the above is equivalent to:

$$c',\ (\bigwedge_{h\in 1..n} D_h\ says\ \mathsf{d}_h) \to \mathsf{b},\ \bigwedge_{j\in J} A_j\ says\ \mathsf{a}_j\ \vdash\ \cdots$$

from which the thesis follows, because $act(\eta) = act(\eta')$.

4. $c = c', (\bigwedge_{h\in H} D_h\ says\ \mathsf{d}_h) \twoheadrightarrow \mathsf{b}$, transition by the rule (TAU) on the leftmost τ of $[(\bigwedge_{h\in H} D_h\ says\ \mathsf{d}_h) \twoheadrightarrow \mathsf{b}]$. Then:

$$c_0 \xrightarrow{A\ says\ \tau} [c'] \mid \big|\big|_{i\in I} DEBT(B_i, \mathsf{b}_i) \mid \big|\big|_{j\in J} A_j\langle \mathsf{a}_j\rangle$$

By the induction hypothesis, we have that:

$$c',\ \bigwedge_{j\in J} A_j\ says\ \mathsf{a}_j\ \vdash\ act(\eta'),\ \bigwedge_{i\in I} B_i\ says\ \mathsf{b}_i$$

Since $act(\eta) = act(\eta')$, to conclude it suffices to weaken the entailment:

$$c,\ \bigwedge_{j\in J} A_j\ says\ \mathsf{a}_j\ \vdash\ act(\eta),\ \bigwedge_{i\in I} B_i\ says\ \mathsf{b}_i$$

5. $c = c', (\bigwedge_{h\in H} D_h\ says\ \mathsf{d}_h) \twoheadrightarrow \mathsf{b}$, transition by the rule (TAU) on the rightmost τ of $[(\bigwedge_{h\in H} D_h\ says\ \mathsf{d}_h) \twoheadrightarrow \mathsf{b}]$. Then:

$$\begin{aligned} c_0 \xrightarrow{A\ says\ \tau} [c'] \mid\ & \big|\big|_{h\in H} DEBT(D_h, \mathsf{d}_h) \mid OUT(\mathsf{b}) \mid \\ & \big|\big|_{i\in I} DEBT(B_i, \mathsf{b}_i) \mid \big|\big|_{j\in J} A_j\langle \mathsf{a}_j\rangle \\ = [c', \mathsf{b}] \mid & \big|\big|_{h\in H} DEBT(D_h, \mathsf{d}_h) \mid \\ & \big|\big|_{i\in I} DEBT(B_i, \mathsf{b}_i) \mid \big|\big|_{j\in J} A_j\langle \mathsf{a}_j\rangle \end{aligned}$$

By the induction hypothesis, we have that:

$$c',\ \mathsf{b},\ \bigwedge_{j\in J} A_j\ says\ \mathsf{a}_j\ \vdash\ act(\eta'),\ \bigwedge_{h\in H} D_h\ says\ \mathsf{d}_h,\ \bigwedge_{i\in I} B_i\ says\ \mathsf{b}_i \quad (16)$$

The above entailment can be weakened as follows:

$$c',\ \bigwedge_{j\in J} A_j\ says\ \mathsf{a}_j,\ (\bigwedge_{h\in H} D_h\ says\ \mathsf{d}_h) \twoheadrightarrow \mathsf{b},\ \mathsf{b}\ \vdash \bigwedge_{h\in H} D_h\ says\ \mathsf{d}_h \quad (17)$$

Also, by the rule (Id) we have that:

$$c',\ \bigwedge_{j\in J} A_j\ says\ \mathsf{a}_j,\ (\bigwedge_{h\in H} D_h\ says\ \mathsf{d}_h) \twoheadrightarrow \mathsf{b},\ \mathsf{b}\ \vdash\ \mathsf{b} \tag{18}$$

Therefore, by the rule (Fix) it follows that:

$$\frac{(17)\qquad (18)}{c',\ \bigwedge_{j\in J} A_j\ says\ \mathsf{a}_j,\ (\bigwedge_{h\in H} D_h\ says\ \mathsf{d}_h) \twoheadrightarrow \mathsf{b} \vdash \mathsf{b}}\ (\textsc{Fix}) \tag{19}$$

Since $act(\eta) = act(\eta')$, (16) and (19) allow to obtain the thesis:

$$c',\ (\bigwedge_{h\in H} D_h\ says\ \mathsf{d}_h) \twoheadrightarrow \mathsf{b},\ \bigwedge_{j\in J} A_j\ says\ \mathsf{a}_j\ \vdash\ act(\eta),\ \bigwedge_{i\in I} B_i\ says\ \mathsf{b}_i$$

6. Transition by the rule (Tau) on $DEBT(B_k, \mathsf{b}_k)$, for some $k \in I$. Then:

$$c_0\ \xrightarrow{A\ says\ \tau}\!\!\!\!\!\rightarrow\ c_0$$

and the induction hypothesis gives the thesis.

7. Transition by the rule (Out) on $DEBT(B_k, \mathsf{b}_k)$, for some $k \in I$. This requires that a suitable pending message is available in c_0, i.e. there exists $h \in J$ such that $A_h\langle \mathsf{a}_h\rangle = B_k\langle \mathsf{b}_k\rangle A$. Then:

$$c_0\ \xrightarrow{A\ says\ B_k\langle \mathsf{b}_k}\!\!\!\!\!\rightarrow\ [c'] \mid \big|_{i\in I\setminus\{k\}} DEBT(B_i, \mathsf{b}_i) \mid \big|_{j\in J\setminus\{h\}} A_j\langle \mathsf{a}_j\rangle$$

By the induction hypothesis, we have that:

$$c',\ \bigwedge_{j\in J\setminus\{h\}} A_j\ says\ \mathsf{a}_j\ \vdash\ act(\eta'),\ \bigwedge_{i\in I\setminus\{k\}} B_i\ says\ \mathsf{b}_i$$

By weakening the entailment, the above is equivalent to:

$$c',\ \bigwedge_{j\in J} A_j\ says\ \mathsf{a}_j\ \vdash\ act(\eta'),\ \bigwedge_{i\in I\setminus\{k\}} B_i\ says\ \mathsf{b}_i$$

Now, since $A_h\ says\ \mathsf{a}_h = B_k\ says\ \mathsf{b}_k$, by the rule (Id) we obtain:

$$c',\ \bigwedge_{j\in J} A_j\ says\ \mathsf{a}_j\ \vdash\ act(\eta'),\ \bigwedge_{i\in I} B_i\ says\ \mathsf{b}_i$$

and the thesis follows because $act(\eta) = act(\eta')$. □

Lemma A.10. *For all* $1N$-PCL *contracts* c *such that* $\big[c\big]_{\mathcal{P}} \xrightarrow{\eta}\!\!\!\!\rightarrow\; d$ *and* A *says* $\overline{\mathsf{a}}\rangle B \in \eta$, *we have that* $B \in \mathcal{P}$, *and either* d *has the form* $d' \mid A$ *says* $OUT_{\mathcal{P}}(\mathsf{a})$, *or there exists some* η' *with* A *says* $\overline{\mathsf{a}}\rangle B \in \eta'$ *such that* $\big[c\big]_{\mathcal{P}} \xrightarrow{\eta'}\!\!\!\!\rightarrow\; d \mid A$ *says* $OUT_{\mathcal{P}}(\mathsf{a})$.

Proof. We proceed by induction on the length of η.

In the base case, η necessarily consists of A *says* $\overline{\mathsf{a}}\rangle B$. Hence, the thesis trivially follows by construction since $\big[c\big]_{\mathcal{P}}$ must be of the form $c' \mid A$ *says* $OUT_{\mathcal{P}}(\mathsf{a})$ with $B \in \mathcal{P}$ so $d' = c' \mid A\rangle\mathsf{a}\rangle B \mid A$ *says* $OUT_{\mathcal{P}}(\mathsf{a})$ (by rules (REC), (OUT), and (PAR) in Figure 1).

For the inductive case, let $\eta = \eta_1\ A$ *says* $\overline{\mathsf{a}}\rangle B\ \eta_2$ with A *says* $\overline{\mathsf{a}}\rangle B \notin \eta_2$. Namely, assume that $\big[c\big]_{\mathcal{P}} \xrightarrow{\eta_1}\!\!\!\!\rightarrow\; d_1 \xrightarrow{A \text{ says } \overline{\mathsf{a}}\rangle B}\!\!\!\!\rightarrow\; d_2 \xrightarrow{\eta_2}\!\!\!\!\rightarrow\; d$. Note that d_2 has the form $d_2' \mid A$ *says* $OUT_{\mathcal{P}}(\mathsf{a})$ and recall that A *says* $OUT_{\mathcal{P}}(\mathsf{a}) = \tau.\mathbf{0} + \sum_{B\in\mathcal{P}} \overline{\mathsf{a}}\rangle B.A$ *says* $OUT_{\mathcal{P}}(\mathsf{a})$.

If d is not of the form $d' \mid A$ *says* $OUT_{\mathcal{P}}(\mathsf{a})$ then there must be a state d'' reached through η_2 where the τ transition of A *says* $OUT_{\mathcal{P}}(\mathsf{a})$ is taken. Removing such transition from η_2 would give another trace η_2' from d_2 where all states are the same as those in η in parallel with A *says* $OUT_{\mathcal{P}}(\mathsf{a})$, which gives the thesis. □

Definition A.11. *We say that a participant* A *is* in charge of α *in state* c_0 *if* $c_0 \xrightarrow{A \text{ says } \alpha}\!\!\!\!\rightarrow$.

Lemma A.12. *Let* c *be a non-ambiguous* $1N$-PCL *contract, and let the set* $\mathcal{P}$ *include the participant names occurring in* c. *If* $c \vdash A$ *says* a, *then:*

$$\exists \eta, d.\ \Big(\ \big[c\big]_{\mathcal{P}} \xrightarrow{\eta} d\ \wedge\ A \text{ says } \mathsf{a} \in act(\eta)\ \wedge\ \forall B \in \mathcal{P}.\ B \smile d\ \Big)$$

Proof. Since c is non-ambiguous, then by Definition 2.6 there exists a function $\pi = \pi(c)$ which maps each atom in c to a single participant. Since $c \vdash A$ *says* a by hypothesis, then $\pi(\mathsf{a}) = A$. Let c_1 be a *says*-free $1N$-PCL formula such that $c = s_\pi(c_1)$. Since $s_\pi(c_1) \vdash s_\pi(\mathsf{a})$, then by Lemma A.7 it follows that $c_1 \vdash \mathsf{a}$. By repeated applications of Lemma A.4, we can write c_1 as c_u, c', for some c_u and c' such that $c_u \vdash \mathsf{b}$ for all atoms b occurring in c_u, and $c_u \vdash \mathsf{a}$. W.l.o.g. we can assume such c_u to be minimal, so that removing a formula from it would prevent the entailment of some atom.

By observing that:

$$\big[c\big]_{\mathcal{P}} \;=\; \big[s_\pi(c_1)\big]_{\mathcal{P}} \;=\; \big[s_\pi(c_u)\big]_{\mathcal{P}} \;\mid\; \big[s_\pi(c')\big]_{\mathcal{P}}$$

we show the thesis by exhibiting a trace where all the steps are due to the subprocess $\left[\!\left[s_\pi(c_u)\right]\!\right]_{\mathcal{P}}$, while $\left[\!\left[s_\pi(c')\right]\!\right]_{\mathcal{P}}$ stays idle. The trace of $\left[\!\left[s_\pi(c_u)\right]\!\right]_{\mathcal{P}}$ is then constructed as follows:

- First, we consider the contractual implications of c_u. Let $c_u = c'_u, s_\pi((\bigwedge_{i=1}^n \mathsf{b}_i) \twoheadrightarrow \mathsf{b}), \ldots$ where c'_u is free from $\twoheadrightarrow$. In the encoding of each $\twoheadrightarrow$, we fire the second τ: we therefore have

  $$\left[\!\left[c_u\right]\!\right]_{\mathcal{P}} \xrightarrow{-\ says\ \tau\ \twoheadrightarrow}{}^{*} \left[\!\left[c'_u\right]\!\right]_{\mathcal{P}} \mid \pi(\mathsf{b})\ says\ \Big(OUT_{\mathcal{P}}(\mathsf{b}) \mid \big|\big|_{i=1}^{n}\ DEBT(\pi(\mathsf{b}_i), \mathsf{b}_i)\Big) \mid \ldots$$

  This basically makes available in the contract all the outputs b which occur as a consequence of a contractual implication, at the cost of introducing the associated debts.

- Then, using all the $OUT_{\mathcal{P}}(\mathsf{b})$ in the context, including those we have spawned above, we fire all the inputs in the encoding of the (standard) implications of c'_u, hence spawning outputs for their consequences, as well. Unlike for the contractual implications, this requires to follow a precise ordering, namely the ordering in which implications would be eliminated in a proof of their consequence. For instance, for $\mathsf{a}_1, \mathsf{a}_1 \to \mathsf{a}_2, \mathsf{a}_2 \to \mathsf{a}_3$ we would use a_1 to fire the inputs of $\mathsf{a}_1 \to \mathsf{a}_2$, and only then use a_2 to fire the inputs of $\mathsf{a}_2 \to \mathsf{a}_3$. In the general case, a correct order of such implications always exists, since otherwise we would have a circular dependency between some implications, some of which would then be redundant to entail some atom, and hence it would not belong to c_u which is minimal.

  This shows that

  $$\left[\!\left[c_u\right]\!\right]_{\mathcal{P}} \twoheadrightarrow^{*} \left[\!\left[c''_u\right]\!\right]_{\mathcal{P}} \mid \\ \pi(\mathsf{b})\ says\ \Big(OUT_{\mathcal{P}}(\mathsf{b}) \mid \big|\big|_{i=1}^{n}\ DEBT(\pi(\mathsf{b}_i), \mathsf{b}_i) \mid OUT_{\mathcal{P}}(\mathsf{c}) \mid \cdots\Big) \mid \ldots$$

  where the c's denote the consequences of the implications.

- Above we made available all the $OUT_{\mathcal{P}}(\mathsf{b})$ for any b in c_u. These allow us to fire all the inputs of all the $DEBT(\pi(\mathsf{b}_i), \mathsf{b}_i)$, so making them move to 0.

- Having removed the encoding of implications, contractual implications, and debts, now only the $OUT_{\mathcal{P}}(\mathsf{b})$ remain, for any b in c_u. We can then make them output all their atoms, so that they occur in η, and then make them move to $d = 0$, so that we can have $\mathsf{B} \breve{\cup} d$ for any B.

The trace generated above indeed includes all the entailed atoms. □

Theorem 2.7. Let c be a non-ambiguous $1N$-PCL contract, and let the set $\mathcal{P}$ include the participant names occurring in c. Then, $c \vdash A \; says \; \mathsf{a}$ iff:

$$\exists \eta, d, B.\ \Big([c]_{\mathcal{P}} \xrightarrow{\eta} d \ \wedge \ (A \; does \; \overline{\mathsf{a}}\rangle B) \in \eta \ \wedge \ \forall B' \in \mathcal{P}.\ B' \between d \Big)$$

Proof. The "only if" direction follows from Lemma A.9, by substituting the empty set for I and J. The "if" direction follows from Lemma A.12. □