Towards a theory of honesty

Massimo Bartoletti[1], Alceste Scalas[1] and Roberto Zunino[2]

[1] Università degli Studi di Cagliari, Italy — {bart , alceste.scalas}@unica.it
[2] Università degli Studi di Trento, Italy — roberto.zunino@unitn.it

Abstract

In contract-oriented systems, distributed agents may advertise and stipulate contracts, and interact via contract-based sessions. However, differently to most other approaches to distributed agents, they are not assumed to always keep their promises. The *honesty* property [4] characterises those agents who always respect their contracts, in *all* possible execution contexts. This property is quite strong, because it requires that an agent is capable of fulfilling her obligations also when the context plays against her. We discuss some work in progress about weaker notions of honesty (which however still guarantee safe interactions among agents), and their relation with different cases of inter-session dependencies. Our final goal is a theory for *blame shifting*, allowing to determine when a (not-strictly-honest) agent can blame other (dishonest) agents for her contractual violations.

1 Introduction

Contract-oriented computing is a software design paradigm where the interaction between clients and services is disciplined through contracts [5, 3]. Contract-oriented services start their life-cycle by advertising contracts which specify their required and offered behaviour. When compliant contracts are found, a session is created among the respective services, which may then start interacting to fulfil their contracts. Differently from other design paradigms (e.g. those based on the session types discipline [8]), services are not assumed to be *honest*, in that they might not respect the promises made. This may happen either unintentionally (because of errors in the service specification), or because of malicious behaviour.

Dishonest behaviour is assumed to be automatically detected and sanctioned by the service infrastructure. This gives rise to a new kind of attacks, that exploit possible discrepancies between the promised and the actual behaviour. If a service does not behave as promised, an attacker can induce it to a situation where the service is sanctioned, while the attacker is reckoned honest. A crucial problem is then how to avoid that a service results definitively culpable of a contract violation, despite of the honest intentions of its developer.

Our investigation about honesty started in [4], where we formalised this property, and showed it undecidable. This led us to the problem of devising safe, computable approximations of honesty, which were proposed both in the form of a type system [2] and of a model checking technique [1].

Although honesty guarantees nice interactions among participants — e.g. a system composed by honest participants enjoys progress — and although its computable approximations can, in many relevant cases, correctly classify processes as honest or dishonest, writing honest processes is a hard task. Indeed, an honest agent is required to fulfil her obligations also in an adversarial context which plays against her.

For instance, consider a scenario with the following *dramatis personae*:

A, an apple broker
B, a buyer
M, a malevolent farmer

Suppose that B wants to buy an apple from A, who in turns needs to obtain a batch of apples from M. A possible (informal) way of specifying the behaviour of A is the following:

1. advertise the contract c = "I get €1, and then I give 1 apple";
2. when the contract c is stipulated, get €1 from B;
3. advertise the contract c' = "I pay €100, and then I get 100Kg of apples";
4. when the contract c' is stipulated, pay €100 to M, and then receive 100Kg of apples;
5. finally, give 1 apple to B.

Such transaction goes perfectly smooth when the participant M is honest. However, when M is dishonest (e.g. if he grabs the money and runs away), the apple broker may incur in a situation where he should give one apple to A, but it cannot. Therefore, we would classify A as *dishonest*.

The above scenario highlights that the honesty property is quite strong: to be honest, a participant should be able to fulfil her obligations without the help of the context. For instance, to make our apple broker honest, one could modify her contract to include a "refund" option to protect against cases like the one above. However, such escape options might not be always possible, hence it could be interesting to find weaker notions of honesty which still guarantee "safe" interactions.

In this paper we present some work in progress of our investigation about this topic. We start by giving some background about contract-oriented systems: in § 2 we present a contract model and the calculus of contracting processes CO_2, while in § 3 we formalise honesty. The new material is contained in § 4 and § 5. In § 4 we introduce the *weak honesty* property: roughly, a process is weakly honest if it behaves honestly provided that the other partners behave honestly. In § 5 we study *session dependencies*, aiming at establishing causal dependencies among contract violations. For instance, in the example above there exists a dependency between M and A, in the sense that A is culpable in her transaction with B because M is culpable in the transaction with A; or, in other words, A would have behaved honestly with B, if only M were honest with A. In this situation, it would be reasonable to say that A can *shift the blame* on M. In the conclusions (§ 6) we sketch how weak honesty and session dependencies may constitute the building blocks for a theory of *blame shifting*.

2 Contract-oriented systems

The formalisation of contract-oriented systems relies on two basic ingredients:

- a model of *contracts*, which specify the promised agent behaviour.
- a model of *processes*, which specify the actual agent behaviour. Processes may exploit the primitives defined by the contract model, e.g. to check compliance between contracts, to make a contract evolve upon an action, to query the current state of a contract, *etc.*

Ideally a general theory of honesty should abstract as much as possible from the actual choices for the two models. However, different instances of the models may give rise to different notions of honesty — in the same way as different process calculi may require different notions of observational equivalences. Continuing the parallel with process calculi, where a variety of different observational equivalences may be relevant for a single process calculus (e.g. [9] reports a multitude of different notions of bisimulation), it is reasonable as well that even by fixing the choice of the contract/process models, many relevant notions of honesty exist.

Therefore, some loss in generality is needed in order to give any theory of honesty. In this paper we focus on behavioural contracts similar to those introduced in [6], and on processes specified in the calculus CO_2 [3], which has primitives for advertising contracts, for opening

sessions between agents with compliant contracts, for doing contractual actions, and for querying contracts. We expect that the basic notions given in § 4 will at least provide useful intuitions about how to adapt the theory when considering different contract/process models.

2.1 Contracts

Contracts are modelled in a variant of CCS, inspired by [6] and refined in [4]. Here we provide only the main definitions and results, and we refer to [1] for the full technical details.

We assume a set of participants, ranged over $\mathsf{A}, \mathsf{B}, \ldots$, and a set of *atoms* $\mathsf{a}, \mathsf{b}, \ldots$, that represent the actions performed by participants. We use an involution $\bar{\mathsf{a}}$, as in CCS.

We distinguish between (*unilateral*) contracts c, which model the promised behaviour of a single participant, and *bilateral* contracts γ, which combine the contracts of two participants.

Definition 2.1 (Contracts)**.** Unilateral contracts *are defined by the following grammar:*

$$c, d \;::=\; \bigoplus_{i\in\mathcal{I}} \mathsf{a}_i;c_i \;\;\Big|\;\; \sum_{i\in\mathcal{I}} \mathsf{a}_i.c_i \;\;\Big|\;\; \mathit{ready}\ \mathsf{a}.c \;\;\Big|\;\; \mathit{rec}\ X\,.\,c \;\;\Big|\;\; X$$

where (i) *the index set* $\mathcal{I}$ *is finite;* (ii) *the ready prefix may appear at the top-level, only; and* (iii) *recursion is guarded.*

Bilateral contracts *are terms of the form* A *says* $c \mid \mathsf{B}$ *says* d*, where* $\mathsf{A} \neq \mathsf{B}$ *and at most one occurrence of ready is present.*

An internal sum $\bigoplus_{i\in\mathcal{I}} \mathsf{a}_i;c_i$ requires a participant A to choose and perform one of the actions a_i, and then to behave according to its continuation c_i. Dually, an external sum $\sum_{i\in\mathcal{I}} \mathsf{a}_i.c_i$ requires A to offer B a choice among all the branches. If B chooses a_i, then A must continue according to c_i.

The behaviour of bilateral contracts is given in terms of a labelled transition relation. Rule [IntExt] regulates the interaction between a participant A making an internal choice, and B offering an external choice:

$$\mathsf{A}\ \mathit{says}\ (\bar{\mathsf{a}};c \oplus c') \mid \mathsf{B}\ \mathit{says}\ (\mathsf{a}.d + d') \;\xrightarrow{\mathsf{A}\ \mathit{says}\ \mathsf{a}}\!\!\!\!\rightarrow\; \mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says\ ready}\ \mathsf{a}.d \qquad [\textsc{IntExt}]$$

If A chooses the branch a in her internal sum, then B is committed to the corresponding branch $\bar{\mathsf{a}}$ in his external sum. This is modelled by marking the selected branch with *ready* $\bar{\mathsf{a}}$, and by discarding the other branches. Rule [Rdy] allows B to perform the marked branch:

$$\mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says\ ready}\ \bar{\mathsf{a}}.\ d \;\xrightarrow{\mathsf{B}\ \mathit{says}\ \bar{\mathsf{a}}}\!\!\!\!\rightarrow\; \mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says}\ d \qquad [\textsc{Rdy}]$$

The previous rules do not define the behaviour of a bilateral contract in case an internal choice is not matched by any action in the external choice of the partner. To guarantee that a bilateral contract can keep evolving (until one of the participants wants to exit), we introduce the notion of compliance.

Two contracts are *compliant* if, whenever a participant A wants to choose a branch in an internal sum, then participant B always offers A the opportunity to do it. To formalise compliance, we first define a partial function rdy from bilateral contracts to sets of atoms. Intuitively, if the unilateral contracts in γ do not agree on the first step, then $\mathrm{rdy}(\gamma)$ is undefined (i.e. equal to $\bot$). Otherwise, $\mathrm{rdy}(\gamma)$ contains the atoms which could be fired in the first step.

Definition 2.2 (Compliance)**.** *Let the partial function* rdy *be defined as:*

$$\mathrm{rdy}\left(\mathsf{A}\ \mathit{says}\ \bigoplus_{i\in\mathcal{I}} \mathsf{a}_i;c_i \mid \mathsf{B}\ \mathit{says}\ \sum_{j\in\mathcal{J}} \mathsf{b}_j.c_j\right) \;=\; \{\mathsf{a}_i\}_{i\in\mathcal{I}} \quad \begin{array}{ll} \mathit{if} & \{\mathsf{a}_i\}_{i\in\mathcal{I}} \subseteq \{\bar{\mathsf{b}}_j\}_{j\in\mathcal{J}} \\ \mathit{and} & (\mathcal{I}=\emptyset \implies \mathcal{J}=\emptyset) \end{array}$$

$$\mathrm{rdy}(\mathsf{A}\ \mathit{says\ ready}\ \mathsf{a}.c \mid \mathsf{B}\ \mathit{says}\ d) \;=\; \{\mathsf{a}\}$$

Then, the compliance relation $\bowtie$ between unilateral contracts is the largest relation such that, whenever $c \bowtie d$:

(1) $\mathrm{rdy}(\mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says}\ d) \neq \bot$

(2) $\mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says}\ d \xrightarrow{\mu}\!\!\!\!\rightarrow \mathsf{A}\ \mathit{says}\ c' \mid \mathsf{B}\ \mathit{says}\ d' \implies c' \bowtie d'$

Example 2.3. *Let $\gamma = \mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says}\ d$, where $c = \mathsf{a};c_1 \oplus \mathsf{b};c_2$ and $d = \bar{\mathsf{a}}.d_1 + \bar{\mathsf{c}}.d_2$. If the participant A internally chooses to perform a, then γ will take a transition to $\mathsf{A}\ \mathit{says}\ c_1 \mid \mathsf{B}\ \mathit{says\ ready}\ \bar{\mathsf{a}}.d_1$. Suppose instead that A chooses to perform b, which is not offered by B in his external choice. In this case, $\gamma \not\xrightarrow{\mathsf{A}\ \mathit{says}\ \mathsf{b}}\!\!\!\!\rightarrow$. We have that $\mathrm{rdy}(\gamma) = \bot$, which does not respect item* (1) *of Def. 2.2. Therefore, c and d are* not *compliant.*

We now tackle the problem of determining who is expected to make the next step for the fulfilment of a bilateral contract. We call a participant A *culpable* in γ if she is expected to perform some actions so to make γ progress.

Definition 2.4 (Culpability)**.** *A participant A is culpable in γ ($\mathsf{A} \frown \gamma$ in symbols) iff $\gamma \xrightarrow{\mathsf{A}\ \mathit{says}\ \mathsf{a}}\!\!\!\!\rightarrow$ for some a. When A is* not *culpable in γ we write $\mathsf{A} \smile \gamma$.*

Theorem 2.5 below establishes that only one participant may be culpable in a bilateral contract. Also, the only contract without culpable participants is $\mathsf{A}\ \mathit{says}\ 0 \mid \mathsf{B}\ \mathit{says}\ 0$, which represents a successfully terminated interaction.

Theorem 2.5. *Let $\gamma = \mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says}\ d$, with $c \bowtie d$. If $\gamma \twoheadrightarrow^* \gamma'$, then either $\gamma' = \mathsf{A}\ \mathit{says}\ 0 \mid \mathsf{B}\ \mathit{says}\ 0$, or there exist a unique culpable in γ'.*

The following theorem states that a participant is always able to recover from culpability by performing some of her duties. This requires at most two steps.

Theorem 2.6 (Contractual exculpation)**.** *Let $\gamma = \mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says}\ d$. For all γ' such that $\gamma \twoheadrightarrow^* \gamma'$, we have that:*

(1) $\gamma' \not\twoheadrightarrow \implies \mathsf{A} \smile \gamma'$ *and* $\mathsf{B} \smile \gamma'$

(2) $\mathsf{A} \frown \gamma' \implies \forall \gamma''.\gamma' \twoheadrightarrow \gamma'' \implies \begin{cases} \mathsf{A} \smile \gamma'', \textit{ or} \\ \forall \gamma'''.\gamma'' \twoheadrightarrow \gamma''' \implies \mathsf{A} \smile \gamma''' \end{cases}$

2.2 A Calculus of Contracting Processes

We now embed the contracts of § 2.1 in the process calculus CO_2 [5, 4, 3, 2]. Let **Var** and **Nam** be disjoint sets of, respectively, *session variables* (ranged over by $x, y, \ldots$) and *session names* (ranged over by $s, t, \ldots$). Let $u, v, \ldots$ range over $\mathbf{Var} \cup \mathbf{Nam}$.

A *latent contract* $\{\downarrow_x c\}_{\mathsf{A}}$ represents a contract c signed by A but not stipulated yet. The variable x will be instantiated to a fresh session name upon stipulation. Unlike in previous versions of CO_2, here we also allow for latent *sets* of contracts $\{\downarrow_{u_1} c_1, \ldots, \downarrow_{u_k} c_k\}_{\mathsf{A}}$, to be stipulated atomically. We let $C, C', \ldots$ range over latent sets of contracts.

$$(u)\mathsf{A}[P] \equiv \mathsf{A}[(u)P] \qquad Z \mid \mathbf{0} \equiv Z \qquad Z \mid Z' \equiv Z' \mid Z \qquad (Z \mid Z') \mid Z'' \equiv Z \mid (Z' \mid Z'')$$
$$Z \mid (u)Z' \equiv (u)(Z \mid Z') \text{ if } u \notin \mathrm{fv}(Z) \cup \mathrm{fn}(Z)$$
$$(u)(v)Z \equiv (v)(u)Z \qquad (u)Z \equiv Z \text{ if } u \notin \mathrm{fv}(Z) \cup \mathrm{fn}(Z)$$

Figure 1: Structural congruence for CO_2 (Z range over processes, systems, or latent contracts)

Definition 2.7 (CO_2 syntax)**.** *The syntax of CO_2 is defined as follows:*

$$\begin{array}{lcll} \pi & ::= & \tau \mid \mathsf{tell}\ C \mid \mathsf{do}_u\, \mathsf{a} \mid \mathsf{ask}_u\, \phi & \textit{(Prefixes)} \\ P & ::= & \sum_i \pi_i.P_i \mid P \mid P \mid (\vec{u})P \mid X(\vec{u}) & \textit{(Processes)} \\ S & ::= & \mathbf{0} \mid \mathsf{A}[P] \mid C \mid s[\gamma] \mid S \mid S \mid (\vec{u})S & \textit{(Systems)} \end{array}$$

Processes specify the behaviour of participants. A process can be a prefix-guarded finite sum $\sum_i \pi_i.P_i$, a parallel composition $P \mid Q$, a delimited process $(\vec{u})P$, or a constant $X(\vec{u})$. We write $\mathbf{0}$ for $\sum_\emptyset P$ and $\pi_1.Q_1 + P$ for $\sum_{i \in I \cup \{1\}} \pi_i.Q_i$ provided that $P = \sum_{i \in I} \pi_i.Q_i$ and $1 \notin I$. We omit trailing occurrences of $\mathbf{0}$. We stipulate that each X has a unique defining equation $X(u_1, \ldots, u_j) \stackrel{\text{def}}{=} P$ such that $\mathrm{fv}(P) \subseteq \{u_1, \ldots, u_j\} \subseteq \mathbf{Var}$, and each occurrence of process identifiers in P is prefix-guarded.

Prefixes include the silent action τ, contract advertisement $\mathsf{tell}\ C$, action execution $\mathsf{do}_u\, \mathsf{a}$, and contract query $\mathsf{ask}_u\, \phi$. In each prefix $\pi \neq \tau$, the identifier u refers to the target session involved in the execution of π. As in [4], we leave the relation $\gamma \vdash \phi$ unspecified (it could be interpreted e.g. as satisfaction of LTL formulae, as done in [1]). We also assume that any $\mathsf{tell}\ C$ inside participant $\mathsf{A}[\cdots]$ is signed by A.

A system is composed of *agents* (i.e., named processes) $\mathsf{A}[P]$, *sessions* $s[\gamma]$, sets of *latent contracts* C, and delimited systems $(\vec{u})S$. Delimitation $(\vec{u})$ binds session variables and names, both in processes and systems. Free variables and names are defined as usual, and they are denoted by $\mathrm{fv}(\_)$ and $\mathrm{fn}(\_)$. A system/process is *closed* when it has no free variables. Each participant may have at most one process in a system, i.e. we forbid systems of the form $(\vec{u})(\mathsf{A}[P] \mid \mathsf{A}[Q] \mid \cdots)$. We denote with K a special participant name (which plays the role of contract broker) such that, in each system $(\vec{u})(\mathsf{A}[P] \mid \cdots)$, $\mathsf{A} \neq \mathsf{K}$. A system is A-*free* when it does not contain latent contracts of A, nor contracts of A stipulated in a session.

The semantics of CO_2 is formalised by a reduction relation on systems (Fig. 2). This relies on a structural congruence, which is the smallest relation satisfying the laws in 1. Such laws are mostly standard — we just point out that $(\vec{u})\mathsf{A}[(\vec{v})P] \equiv (\vec{u}, \vec{v})\mathsf{A}[P]$ allows to move delimitations between CO_2 systems and processes. In order to define honesty in § 3, here we decorate transitions with labels, by writing $\xrightarrow{\mathsf{A}:\, \pi}$ for a reduction where participant A fires prefix π.

Rule [Tau] just fires a τ prefix. Rule [Tell] advertises a latent set of contracts C. Rule [Fuse] inspect latent contracts, which are stipulated when compliant pairs are found through the $\cdot \rhd^{\sigma} \cdot$ relation (see Def. 2.8 below). Upon stipulation, one or more new sessions among the stipulating parties are created. Rule [Do] allows a participant A to perform an action in the session s containing γ (which, accordingly, evolves to γ'). Rule [Ask] allows A to proceed only if the contract γ at session s satisfies the property ϕ. The last three rules are mostly standard. In rule [Del] the label π fired in the premise becomes a τ in the consequence, when π contains the delimited name/variable. For instance, $(x)\,\mathsf{A}[\mathsf{tell} \downarrow_x c.P] \xrightarrow{\mathsf{A}:\, \tau} (x)\,(\mathsf{A}[P] \mid \{\downarrow_x c\}_{\mathsf{A}})$. Here, it would make little sense to have the label $\mathsf{A} : \mathsf{tell} \downarrow_x c$, as the delimited variable x may be α-converted.

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

$$\mathsf{A}[\mathsf{tell}\ C \,.\, P + P' \mid Q] \xrightarrow{\mathsf{A}:\ \mathsf{tell}\ C} \mathsf{A}[P \mid Q] \ \mid\ C \quad \text{[Tell]}$$

$$\frac{K \rhd^{\sigma} S' \quad \operatorname{ran}\sigma \cap \operatorname{fn}(S) = \emptyset}{(\operatorname{dom}\sigma)(K \mid S) \xrightarrow{\mathsf{K}:\ \tau} (\operatorname{ran}\sigma)(S' \mid S\sigma)} \quad \text{[Fuse]}$$

$$\frac{\gamma \xrightarrow{\mathsf{A}\ says\ \mathsf{a}} \gamma'}{\mathsf{A}[\mathsf{do}_s\,\mathsf{a} \,.\, P + P' \mid Q] \ \mid\ s[\gamma] \xrightarrow{\mathsf{A}:\ \mathsf{do}_s\,\mathsf{a}} \mathsf{A}[P \mid Q] \ \mid\ s[\gamma']} \quad \text{[Do]}$$

$$\frac{\gamma \vdash \phi}{\mathsf{A}[\mathsf{ask}_s\,\phi \,.\, P + P' \mid Q] \ \mid\ s[\gamma] \xrightarrow{\mathsf{A}:\ \mathsf{ask}_s\,\phi} \mathsf{A}[P \mid Q] \ \mid\ s[\gamma]} \quad \text{[Ask]}$$

$$\frac{X(\vec{u}) \stackrel{\text{def}}{=} P \qquad \mathsf{A}[P\{^{\vec{v}}/_{\vec{u}}\} \mid Q] \mid S \xrightarrow{\mathsf{A}:\ \pi} S'}{\mathsf{A}[X(\vec{v}) \mid Q] \mid S \xrightarrow{\mathsf{A}:\ \pi} S'} \quad \text{[Def]}$$

$$\frac{S \xrightarrow{\mathsf{A}:\ \pi} S'}{(u)S \xrightarrow{\mathsf{A}:\ \mathrm{del}_u(\pi)} (u)S'} \quad \text{[Del]} \qquad \text{where } \mathrm{del}_u(\pi) = \begin{cases} \tau & \text{if } u \in \mathrm{fnv}(\pi) \\ \pi & \text{otherwise} \end{cases}$$

$$\frac{S \xrightarrow{\mathsf{A}:\ \pi} S'}{S \mid S'' \xrightarrow{\mathsf{A}:\ \pi} S' \mid S''} \quad \text{[Par]}$$

Figure 2: Reduction semantics of CO_2.

Definition 2.8 (Stipulation)**.** *The relation $C_1 \mid \cdots \mid C_k \rhd^{\sigma} s_1[\gamma_1] \mid \cdots \mid s_n[\gamma_n]$ holds iff:*

1. *for all $i \in 1..k$, $C_i = \left\{\downarrow_{x_{i,1}} c_{i,1}, \ldots, \downarrow_{x_{i,m_i}} c_{i,m_i}\right\}_{\mathsf{B}_i}$, and the variables $x_{i,j}$ are pairwise distinct;*
2. *for all $i \in 1..k$, let $D_i = \{(\mathsf{B}_i, x_{i,h}, c_{i,h}) \mid h \in 1..m_i\}$. The set $\bigcup_i D_i$ is partitioned into a set of n subsets $\mathcal{M}_j = \{(\mathsf{A}_j, x_j, c_j), (\mathsf{B}_j, y_j, d_j)\}$ such that, for all $j \in 1..n$, $\mathsf{A}_j \neq \mathsf{B}_j$, $c_j \bowtie d_j$, and $\gamma_j = \mathsf{A}_j$ says $c_j \mid \mathsf{B}_j$ says d_j;*
3. *$\sigma = \{^{s_1}/_{x_1,y_1}, \cdots, {}^{s_n}/_{x_n,y_n}\}$ maps session variables to pairwise distinct session names $s_1, \ldots, s_n$.*

Example 2.9. *Consider the contracts:*

$$\begin{array}{rclcrcl} c & = & \bar{\mathsf{a}};0 & & c' & = & \bar{\mathsf{b}};0 \\ d & = & \mathsf{a}.0 & & d' & = & \mathsf{b}.0 \end{array}$$

and the CO_2 system $S = (x, y, z)\,(C_{\mathsf{A}} \mid C_{\mathsf{B}} \mid C_{\mathsf{C}} \mid S_0)$, where:

$$C_{\mathsf{A}} = \{\downarrow_x c, \downarrow_y c'\}_{\mathsf{A}} \qquad C_{\mathsf{B}} = \{\downarrow_z d\}_{\mathsf{B}} \qquad C_{\mathsf{C}} = \{\downarrow_w d'\}_{\mathsf{C}}$$

Furthermore, let $\sigma = \{^{s}/_{x,z}, {}^{t}/_{y,w}\}$, $\gamma_{\mathsf{AB}} = \mathsf{A}$ says $c \mid \mathsf{B}$ says d and $\gamma_{\mathsf{AC}} = \mathsf{A}$ says $c' \mid \mathsf{C}$ says d'. Then, according to Def. 2.8 we have that

$$C_{\mathsf{A}} \mid C_{\mathsf{B}} \mid C_{\mathsf{C}} \rhd^{\sigma} s[\gamma_{\mathsf{AB}}] \mid t[\gamma_{\mathsf{AC}}]$$

In fact:

1. *C_{A}, C_{B} and C_{C} contain pairwise distinct variables;*

2. *letting* $D_{\mathsf{A}} = \{(\mathsf{A}, x, c), (\mathsf{A}, y, d)\}$, $D_{\mathsf{B}} = \{(\mathsf{B}, z, c')\}$ *and* $D_{\mathsf{C}} = \{(\mathsf{C}, w, d')\}$, *we can partition the set* $D_{\mathsf{A}} \cup D_{\mathsf{B}} \cup D_{\mathsf{C}}$ *into the subsets* $\mathcal{M}_{\mathsf{AB}} = \{(\mathsf{A}, x, c), (\mathsf{B}, z, c')\}$ *and* $\mathcal{M}_{\mathsf{AC}} = \{(\mathsf{A}, y, c'), (\mathsf{C}, w, d')\}$, *where* $c \bowtie d$ *and* $c' \bowtie d'$.
3. σ *maps session variables* x, z, y, w *to pairwise distinct session names* s, t.

Therefore, by rule [Fuse]*:*

$$S \xrightarrow{\mathsf{K}:\, \tau} (s)\,(s[\gamma_{\mathsf{AB}}] \mid t[\gamma_{\mathsf{AC}}] \mid S_0\sigma)$$

Example 2.10. *Consider the following system:*

$$\begin{aligned} S &= \mathsf{A}[(x)X(x)] \mid \mathsf{B}[(y)Y(y)]] \\ X(x) &\stackrel{def}{=} \mathsf{tell}\ \{\downarrow_x \mathsf{a};0\}_{\mathsf{A}} \,.\, \mathsf{do}_x\, \mathsf{a} \\ Y(y) &\stackrel{def}{=} \mathsf{tell}\ \{\downarrow_y \bar{\mathsf{a}}.0\}_{\mathsf{B}} \,.\, \mathsf{do}_y\, \bar{\mathsf{a}} \end{aligned}$$

A possible execution of S *is the following:*

$$S \xrightarrow{\mathsf{B}:\, \mathsf{tell}\ \{\downarrow_y \bar{\mathsf{a}}\}_{\mathsf{B}}} \mathsf{A}[(x)X(x)] \mid (y)\left(\mathsf{B}[\mathsf{do}_y\, \bar{\mathsf{a}}] \mid \{\downarrow_y \bar{\mathsf{a}}.0\}_{\mathsf{B}}\right) \quad (1)$$

$$\xrightarrow{\mathsf{A}:\, \mathsf{tell}\ \{\downarrow_x \mathsf{a}\}_{\mathsf{A}}} (x, y)\left(\mathsf{A}[\mathsf{do}_x\, \mathsf{a}] \mid \mathsf{B}[\mathsf{do}_y\, \bar{\mathsf{a}}] \mid \{\downarrow_x \mathsf{a};0\}_{\mathsf{A}} \mid \{\downarrow_y \bar{\mathsf{a}}.0\}_{\mathsf{B}}\right) \quad (2)$$

$$\xrightarrow{\mathsf{K}:\, \tau} (s)\left(\mathsf{A}[\mathsf{do}_s\, \mathsf{a}] \mid \mathsf{B}[\mathsf{do}_s\, \bar{\mathsf{a}}] \mid s[\mathsf{A}\ \mathit{says}\ \mathsf{a};0 \mid \mathsf{B}\ \mathit{says}\ \bar{\mathsf{a}}.0]\right) \quad (3)$$

$$\xrightarrow{\mathsf{A}:\, \mathsf{do}_s\, \mathsf{a}} (s)\,(\mathsf{A}[\mathbf{0}] \mid \mathsf{B}[\mathsf{do}_s\, \bar{\mathsf{a}}] \mid s[\mathsf{A}\ \mathit{says}\ 0 \mid \mathsf{B}\ \mathit{says\ ready}\ \bar{\mathsf{a}}.0]) \quad (4)$$

Transitions (1) *and* (2) *above are obtained by applying rules* [Tell], [Par], *and* [Def]. *Transition* (3) *is obtained by rule* [Fuse]. *Finally, transition* (4) *is obtained by rule* [Do], *since* $\gamma \xrightarrow{\mathsf{A}\ \mathit{says}\ \mathsf{a}} \!\!\!\!\rightarrow\ \gamma'$.

3 Honesty

A remarkable feature of CO_2 is that it allows for writing *dishonest* agents which do not keep their promises. Intuitively, a participant is honest if she always fulfils her contractual obligations, in all possible contexts. As remarked in § 1, this notion is crucial in contract-oriented systems, since honest participants will never be liable in case of other participants misbehaviour.

Definition 3.1 (Execution context)**.** *An execution context* $\mathcal{E}$ *is a CO₂ system with at most one hole* $\bullet$*, with the form* $(\vec{u})(\bullet \mid S)$ *for some* $\vec{u}, S$. *We write* $\mathcal{E}(\!|S|\!)$ *for* $\mathcal{E}\{^{S}/_{\bullet}\}$. *We denote by* $\mathbf{E}$ *the set of all* execution contexts*; we define its subset* $\mathbf{E}_\Pi$ *indexed by a set of processes* Π *as follows:*

$$\mathbf{E}_\Pi \;=\; \{\mathcal{E} \in \mathbf{E} \mid \forall \vec{u}, \mathsf{A}, P\,.\, \mathcal{E} \equiv (\vec{u})\,(\bullet \mid \mathsf{A}[P] \mid \cdots) \implies P \in \Pi\}$$

Furthermore, we say that:

- $\mathcal{E}$ *is an* execution context for $\mathsf{A}[P]$ *iff* $\mathcal{E} \equiv (\vec{u})\,(\bullet \mid S)$ *and* $S \not\equiv (\vec{v})(\mathsf{A}[Q] \mid \cdots)$ *for all* Q.
- $\mathcal{E}$ *is* A-initial *iff* $\mathcal{E}$ *contains no latent contracts nor active sessions involving* A *— i.e., for all* $\vec{u}, s, c$: *(i)* $\mathcal{E} \equiv (\vec{u})\,(s[\mathsf{A}\ \mathit{says}\ c \mid \cdots] \mid \cdots) \implies c = 0$*; and, (ii)* $\mathcal{E} \not\equiv (\vec{u})\,(\{\cdots\}_{\mathsf{A}} \mid \cdots)$.

From now on, when writing "$\mathsf{A}[P]$ in $\mathcal{E}$", we will imply that $\mathcal{E}$ is an execution context for $\mathsf{A}[P]$. When quantifying over evaluation contexts, we will also imply the well-formedness of the CO_2 system resulting from the application of the context to a given system: for example, with "$\forall \mathcal{E}\,.\, \mathcal{E}(\!|\mathsf{A}[P] \mid s[\cdots]|\!) \ldots$" we are implicitly ruling out those $\mathcal{E}$ such that $\mathcal{E} \equiv (\vec{u})(\mathsf{A}[\cdots] \mid \cdots)$ and $\mathcal{E} \equiv (\vec{u})(\bullet \mid s[\cdots] \mid \cdots)$. We assume that the structural congruence relation on systems is extended to contexts.

Definition 3.2 (Culpability)**.** *For a participant* A *and a context* $\mathcal{E}$*, we say that:*

- A is culpable at s in $\mathcal{E}$ *iff there exist* $\mathcal{E}', \mathsf{a}$ *such that* $\mathcal{E} \equiv s[\gamma] \mid \mathcal{E}'$ *and* $\gamma \xrightarrow{\mathsf{A}\ says\ \mathsf{a}}\!\!\!\!\rightarrow$.
- A is culpable in $\mathcal{E}$ *iff there exist* $s,u,\mathcal{E}'$ *such that* $\mathcal{E} \equiv (u)\mathcal{E}'$ *and* A *is culpable at* s *in* $\mathcal{E}'$.

Definition 3.3 (Honesty)**.** *Let* $\mathcal{E}$ *be a context for* $\mathsf{A}[P]$*, with* $\mathrm{fv}(P) = \emptyset$*. We say that:*

- $\mathsf{A}[P]$ is honest at s in $\mathcal{E}$ *iff:*
  - *if* A *is culpable at* s *in* $\mathcal{E}$*, then* $\mathcal{E}(\!|\mathsf{A}[P]|\!) \rightarrow^* \xrightarrow{\mathsf{A}:\ \mathsf{do}_s\ -}$*, and*
  - $\mathcal{E}(\!|\mathsf{A}[P]|\!) \rightarrow \mathcal{E}'(\!|\mathsf{A}[P']|\!) \implies \mathsf{A}[P']$ *is honest at* s *in* $\mathcal{E}'$
- $\mathsf{A}[P]$ is honest in $\mathcal{E}$ *iff*
  - $\mathcal{E} \equiv (s')\mathcal{E}' \implies \forall s\,.\,\mathsf{A}[P]$ *is honest at* s *in* $\mathcal{E}'$*, and*
  - $\mathcal{E}(\!|\mathsf{A}[P]|\!) \rightarrow \mathcal{E}'(\!|\mathsf{A}[P']|\!) \implies \mathsf{A}[P']$ *is honest in* $\mathcal{E}'$

We define the set $\mathcal{H}$ *of* honest processes *as:*

$$\mathcal{H} \;=\; \{P \mid \forall\ \mathsf{A}\text{-}initial\ \mathcal{E}\,.\,\mathsf{A}[P]\ is\ honest\ in\ \mathcal{E}\}$$

Note that, for all processes P belonging to $\mathcal{H}$, we have $\mathrm{fv}(P) = \emptyset$ (i.e., all session variables appearing in P are delimited).

Example 3.4 ((Dis)honest processes)**.** *Consider the following process:*

$$P_{\mathsf{A}} \stackrel{def}{=} (x,y)\ \mathsf{tell}\ \{\downarrow_x \mathsf{a}.0\}_{\mathsf{A}}\,.\,\mathsf{tell}\ \{\downarrow_y \mathsf{b};0\}_{\mathsf{A}}\,.\,\mathsf{do}_x\,\mathsf{a}\,.\,\mathsf{do}_y\,\mathsf{b}$$

The process P_{A} *might apparently look honest, although it is not. In fact, consider the context:*

$$\begin{aligned} \mathcal{E} &= \bullet \mid \mathsf{B}[P_{\mathsf{B}}] \mid \mathsf{C}[P_{\mathsf{C}}] \\ P_{\mathsf{B}} &= (z)\ (\mathsf{tell}\ \{\downarrow_z \bar{\mathsf{b}}.0\}_{\mathsf{B}}\,.\,\mathsf{do}_z\,\bar{\mathsf{b}}) \\ P_{\mathsf{C}} &= (w)\ (\mathsf{tell}\ \{\downarrow_w \bar{\mathsf{a}};0\}_{\mathsf{C}}\,.\,\mathbf{0}) \end{aligned}$$

If we reduce $\mathcal{E}(\!|\mathsf{A}[P_{\mathsf{A}}]|\!)$ *by performing all the* tell *actions, we obtain:*

$$\begin{aligned} S = (s,t)\,(\ &\mathsf{A}[\mathsf{do}_t\,\mathsf{a}\,.\,\mathsf{do}_s\,\mathsf{b}] \mid \mathsf{B}[\mathsf{do}_s\,\bar{\mathsf{b}}] \mid \mathsf{C}[\mathbf{0}] \mid \\ &t[\mathsf{A}\ says\ \mathsf{a}.0 \mid \mathsf{C}\ says\ \bar{\mathsf{a}};0] \mid s[\mathsf{A}\ says\ \mathsf{b};0 \mid \mathsf{B}\ says\ \bar{\mathsf{b}}.0]\ ) \end{aligned}$$

Here, S *cannot reduce further:* A *is stuck, waiting for* $\bar{\mathsf{a}}$ *from* C*, which (dishonestly) avoids to do the required internal choice. So,* P_{A} *is dishonest, because it does not perform the promised* b.

The process P_{A} *can be "honestified" by executing its actions on* x *and* y *in parallel. Let:*

$$P'_{\mathsf{A}} \;=\; (x,y)\ \mathsf{tell}\ \{\downarrow_x \mathsf{a}.0\}_{\mathsf{A}}\,.\,\mathsf{tell}\ \{\downarrow_y \mathsf{b};0\}_{\mathsf{A}}\,.\,(\mathsf{do}_x\,\mathsf{a} \mid \mathsf{do}_y\,\mathsf{b})$$

For instance, we have that $\mathcal{E}(\!|\mathsf{A}[P'_{\mathsf{A}}]|\!)$ *can reduce to:*

$$\begin{aligned} S' = (s,t)\,(\ &\mathsf{A}[\mathsf{do}_t\,\mathsf{a}] \mid \mathsf{B}[\mathsf{do}_s\,\bar{\mathsf{b}}] \mid \mathsf{C}[\mathbf{0}] \mid \\ &t[\mathsf{A}\ says\ \mathsf{a}.0 \mid \mathsf{C}\ says\ \bar{\mathsf{a}};0] \mid s[\mathsf{A}\ says\ 0 \mid \mathsf{B}\ says\ ready\ \bar{\mathsf{b}}.0]\ ) \end{aligned}$$

i.e., A *overcomes its duties on session* s*. Indeed, it is easy to convince oneself that* P'_{A} *is always ready to do the needed actions on both sessions, in all possible contexts where* $\mathsf{A}[P'_{\mathsf{A}}]$ *is put. Therefore,* P'_{A} *is honest.*

The following result (established in [4]), states that honesty in CO_2 is undecidable.

Theorem 3.5. *$\mathcal{H}$ is not recursive. $\overline{\mathcal{H}}$ is recursively enumerable but not recursive.*

4 On Weak Honesty

As noted in the previous section, the honesty property is quite strong, because it requires that a process P can exculpate itself even in those (dishonest) contexts where the other participants avoid to do the expected actions.

This suggests a possible variant of honesty, where a process P is only required to behave correctly provided that also the others behave correctly. More formally, we require that P is honest in the *honest contexts* only (i.e. those contexts in $\mathbf{E}_{\mathcal{H}}$). We call the processes belonging to this new class *weakly honest*.

Definition 4.1 (Weak honesty)**.** *We define the set $\mathcal{W}$ of* weakly honest processes *as:*

$$\mathcal{W} \;=\; \{P \mid \forall\ \mathsf{A}\text{-}\textit{initial}\ \mathcal{E} \in \mathbf{E}_{\mathcal{H}}\ .\ \mathsf{A}[P]\ \textit{is honest in}\ \mathcal{E}\}$$

Example 4.2. *The process P_A from Ex. 3.4 is* not *weakly honest. Consider e.g. the context:*

$$\begin{aligned}\mathcal{E} \;&=\; \bullet \mid\ \mathsf{C}[P'_{\mathsf{C}}]\\ P'_{\mathsf{C}} \;&=\; (w)\ (\mathsf{tell}\ \{\downarrow_w \bar{\mathsf{a}};0\}_{\mathsf{C}}\ .\ \mathsf{do}_w\ \bar{\mathsf{a}})\end{aligned}$$

The context $\mathcal{E}$ is clearly honest. However, by reducing $\mathcal{E}(\!|\mathsf{A}[P_{\mathsf{A}}]|\!)$ we reach the state:

$$S \;=\; (s,x)\,(\ \mathsf{A}[\mathsf{do}_x\,\mathsf{a}\ .\ \mathsf{do}_s\,\mathsf{b}]\ \mid\ \mathsf{C}[\mathbf{0}]\ \mid\ t[\mathsf{A}\ \textit{says}\ \mathsf{a}.0 \mid \mathsf{C}\ \textit{says}\ \bar{\mathsf{a}};0]\ )$$

where A is stuck, and has no chance to exculpate herself. Therefore, P_{A} is not weakly honest.

The problem here is that there is no guarantee that the contract on x is always stipulated. We can fix this by making P_{A} advertise both contracts in a single tell. This is done as follows:

$$Q_{\mathsf{A}} \;=\; (x,y)\ \mathsf{tell}\ \{\downarrow_x \mathsf{a}.0\ ,\ \downarrow_y \mathsf{b};0\}\ .\ \mathsf{do}_x\,\mathsf{a}\ .\ \mathsf{do}_y\,\mathsf{b}$$

The process Q_{A} is weakly honest. However, it is not *honest: in fact, in a context where the other participant in session x does not perform its $\bar{\mathsf{a}}$ action, $\mathsf{A}[Q_{\mathsf{A}}]$ will not be ready to fulfil its b action on session y.*

Note that if $P \in \mathcal{H}$ then $P \in \mathcal{W}$: in fact, if $\mathsf{A}[P]$ can advance in all contexts, then in particular it can also advance in the honest ones. We have then proved the following lemma.

Lemma 4.3. $\mathcal{H} \subsetneq \mathcal{W}$.

Example 4.4 (Deadlock)**.** *Consider the process Q_{A} from Ex. 4.2, and let:*

$$Q_{\mathsf{B}} \;=\; (w,z)\ \mathsf{tell}\ \left\{\downarrow_w \bar{\mathsf{b}}.0\ ,\ \downarrow_z \bar{\mathsf{a}};0\right\}\ .\ \mathsf{do}_w\,\bar{\mathsf{b}}\ .\ \mathsf{do}_z\,\bar{\mathsf{a}}$$

Both processes Q_{A} and Q_{B} are weakly honest. However, the system $S = \mathsf{A}[Q_{\mathsf{A}}] \mid \mathsf{B}[Q_{\mathsf{B}}]$ reaches a deadlock after the latent contracts are stipulated, because each participant is waiting the other one for making the first move. Therefore, a system populated by weakly honest processes does not guarantee progress.

In Def. 4.1 we have characterised weakly honest processes as those which are honest in all contexts belonging to $\mathbf{E}_{\mathcal{H}}$. By generalising this procedure, we obtain a way to construct other classes of processes: choose some class $\mathcal{C}$ of processes, and then define the class:

$$\{P \mid \forall\ \mathsf{A}\text{-initial}\ \mathcal{E} \in \mathbf{E}_{\mathcal{C}}\ .\ \mathsf{A}[P]\ \text{is honest in}\ \mathcal{E}\}$$

In particular, one may argue whether the class of processes which behave honestly in any weakly honest context is strictly between $\mathcal{H}$ and $\mathcal{W}$. Quite surprisingly, Theorem 4.5 below says that such class coincides with that of honest processes.

Theorem 4.5. $\{P \mid \forall\ \mathsf{A}\text{-}initial\ \mathcal{E} \in \mathbf{E}_{\mathcal{W}}\ .\ \mathsf{A}[P]\ is\ honest\ in\ \mathcal{E}\} = \mathcal{H}$.

The intuition behind Theorem 4.5 is that, given an agent $\mathsf{A}[P]$ and any A-initial context $\mathcal{E}$, the latter can be rewritten into a context $\tilde{\mathcal{E}}$ only composed by weakly honest participants, such that $\tilde{\mathcal{E}}$ interacts with by $\mathsf{A}[P]$ "in the same way" as $\mathcal{E}$ does (such rewriting exploits the deadlock issue mentioned in Ex. 4.4). Hence, in order to be honest in $\tilde{\mathcal{E}} \in \mathbf{E}_{\mathcal{W}}$, P also needs to be honest in $\mathcal{E}$ — and thus it follows that $P \in \mathcal{H}$.

As for honesty, the weak honesty property is not decidable.

Theorem 4.6. *$\mathcal{W}$ is not recursive. $\overline{\mathcal{W}}$ is recursively enumerable but not recursive.*

5 Session dependencies

In this section, we discuss the concept of session dependencies. Let us consider a CO_2 agent $\mathsf{A}[P]$ involved in two sessions s, t: in order to perform its contractual duties on s, A may first need to perform other actions on t; this, in turn, may require the other involved participant to cooperate (i.e., actually perform her duties in t). Our first goal is to formalise causal dependencies among sessions. This is done below in Def. 5.4. We then exploit dependencies to define finer-grained notions of culpability (Def. 5.9). This allows e.g. to single out an agent which is the "terminal" culpable in a chain of dependencies.

Intuitively, knowing the exact dependencies will provide us with more information than weak honesty, allowing participants to shift the blame for contract violations. For instance, consider a system with just two sessions s and t, where A is culpable at t and B is culpable at s. If there is a dependency from t to s (but not *vice versa*), then A can shift the blame to B.

In order to formalize of session dependencies, we first need to introduce some technical machinery. We start with the set of processes which are honest at a given session s, where they fulfil a given contract c.

Definition 5.1 (Honest processes realizing c at s)**.** *Given a contract c and a session s, we define the set of processes:*

$$\mathcal{H}_{s,c} = \big\{P \mid \forall \mathcal{E} \equiv (\vec{u})\big(s[\mathsf{A}\ says\ c \mid \cdots] \mid \bullet \mid \cdots\big)\ .\ \mathsf{A}[P]\ is\ honest\ at\ s\ in\ \mathcal{E}\big\}$$

Note that, when considering a session name s which is not free in $\mathcal{E}$ (i.e., s does not appear in $\mathcal{E}$, or $s \in \vec{u}$ in Definition 5.1), then $\mathsf{A}[P]$ is trivially honest at s in $\mathcal{E}$ — and therefore, for all c, $P \in \mathcal{H}_{s,c}$.

The predicate $\mathrm{H}(\mathsf{A}, s, \mathcal{E})$ below means that, in the context $\mathcal{E}$, the partner of A at session s is honest at s.

Definition 5.2 (Honest context wrt. session s of A)**.** *We define the predicate $\mathrm{H}(\mathsf{A}, s, \mathcal{E})$ as follows:*

$$\mathrm{H}(\mathsf{A}, s, \mathcal{E}) \iff \exists \vec{u}, c, P, \mathsf{B}\ .\ \mathcal{E} \equiv (\vec{u})\big(s[\mathsf{A}\ says\ \cdots \mid \mathsf{B}\ says\ c] \mid \mathsf{B}[P] \mid \cdots\big) \wedge s \notin \vec{u} \wedge P \in \mathcal{H}_{s,c}$$

Definition 5.3 (Free sessions of A in $\mathcal{E}$)**.** $\mathrm{fn}(\mathsf{A}, \mathcal{E}) = \{s \mid \mathcal{E} \equiv s[A\ says\ \cdots \mid \cdots] \mid \cdots\}$.

Definition 5.4 below formalizes the dependency between a session s and a set of sessions $t_1, \ldots, t_n$. Given a system S, $s \prec_S t_1, \ldots, t_n$ intuitively means that, if S reduces along sessions $t_1, \ldots, t_n$ reaching a state S', then S' will eventually reduce through s. Therefore, if a participant A is culpable at s in S, it will be able to perform its contractual duties after $t_1, \ldots, t_n$ advance.

Definition 5.4 (Session dependencies)**.** *We write* $s \prec_S T$ *iff:*

1. A *is culpable at* s *in* S*,*
2. $S \equiv (\vec{u})\,(\mathsf{A}[P] \mid S_{\mathsf{A}} \mid \cdots)$*, where* S_{A} *is a parallel composition of sessions of* A *with* $\mathrm{fn}(P) \subseteq \mathrm{fn}(S_{\mathsf{A}})$*,*
3. $T \cap \vec{u} = \emptyset$*,*
4. $T \subseteq \mathrm{fn}(\mathsf{A}, S_{\mathsf{A}})$
5. T *is a minimal set such that, for all* $\widetilde{S} = (\vec{u})\,(\mathsf{A}[P] \mid S_{\mathsf{A}} \mid \cdots)$ *with* $\mathrm{H}\Big(\mathsf{A}, t, \widetilde{S}\Big)$ *for all* $t \in T$*:*

$$\widetilde{S} \xrightarrow{\neq \mathsf{A}\colon \mathsf{do}_s\, -}{}^{*}\ \widetilde{S}' \implies \widetilde{S}' \longrightarrow^{*} \xrightarrow{\mathsf{A}\colon \mathsf{do}_s\, -}$$

We write $s \prec_S^* t$ *when:*

$$\exists t_1, \ldots, t_m \ .\ s \prec_S \{t_1, \cdots\} \wedge t_1 \prec_S \{t_2, \cdots\} \wedge \ \ldots \ \wedge t_m \prec_S \{t, \cdots\}$$

Roughly, in order to determine that s depends on $t_1, \ldots, t_n$, Definition 5.4 considers the agent $\mathsf{A}[P]$ currently culpable at s in S, and inserts it in a "cooperating system" $\tilde{S}$ containing (some of) the sessions she is involved in (S_{A}), together with other agents that are honest at these sessions. If $\tilde{S}$ reduces, eventually allowing A to perform a do_s action, then we know that there may be a causal relation between the reduction along (some of) the sessions in $\tilde{S}$, and the avancement of A at s; the minimality requirement of item 5 allows to restrict this dependency to a more precise set $t_1, \ldots, t_n$.

If A is culpable at s in S, session dependencies will result in three main cases (which are *not* mutually exclusive):

- if $s \prec_S \emptyset$, then no other session in S needs to advance before $\mathsf{A}[P]$ can perform its contractual duties at s;
- if $s \prec_S T$ and $s \prec_S T'$, then *either* the sessions in T or T' need to advance, in order for A to advance at s;
- $\nexists T$ such that $s \prec_S T$, then $\mathsf{A}[P]$ is unable to perform its contractual duties at s, even in a cooperating context.

It is worth emphasizing that, when $s \prec_S t_1, \ldots, t_n$ with A culpable at s:

1. A may be or *not* be culpable at any of $t_1, \ldots, t_n$ in S. However, her advancement at these sessions *always* requires the cooperation of other agents, before she can also advance at s. For instance, consider $\mathsf{A}[P]$, with $P = \mathsf{do}_{t_1}\, \overline{\mathsf{a}} \,.\, \mathsf{do}_{t_1}\, \mathsf{b} \,.\, \mathsf{do}_s\, \overline{\mathsf{a}}$. A may be culpable at t_1, e.g. because her contract therein is $c = \overline{\mathsf{a}};\mathsf{b}.0$ — and therefore, she has to perform the first step. Then, it will be the other participant's turn to execute $\overline{\mathsf{b}}$, before A can execute $\mathsf{do}_s\, \overline{\mathsf{a}}$. If the other participant's cooperation was not required, then the minimality condition of Definition 5.4 (item 5) would have ruled out the dependency between s and t_1;
2. the dependency *only refers to the first* do_s*-action* of A. After such action is performed, the dependencies of s may completely change.

Both aspect will be reprised later, in Example 5.15.

Example 5.5 (Deadlocks and dependency loops)**.** *Consider the system* $S = \mathsf{A}[Q_{\mathsf{A}}] \mid \mathsf{B}[Q_{\mathsf{B}}]$ *from Example 4.4. We have* $S \to^* S' = (s,t)S'_0$*, where:*

$$S'_0 = \mathsf{A}[\mathsf{do}_s\, \mathsf{a} \,.\, \mathsf{do}_t\, \mathsf{b}] \ \mid\ \mathsf{B}[\mathsf{do}_t\, \overline{\mathsf{b}} \,.\, \mathsf{do}_s\, \overline{\mathsf{a}}] \ \mid\ s[\mathsf{A}\ \textit{says}\ \mathsf{a}.0 \mid \mathsf{B}\ \textit{says}\ \overline{\mathsf{a}};0] \ \mid\ t[\mathsf{A}\ \textit{says}\ \mathsf{b};0 \mid \mathsf{B}\ \textit{says}\ \overline{\mathsf{b}}.0]$$

and S' *is deadlocked: in fact,* A *is culpable at* t*, and* B *is culpable at* s *in* S'_0 *— but neither of them can advance, because one is waiting for the other to move first. This situation is reflected in session dependencies: in fact, we have* $s \prec_{S'_0} \{t\}$ *and* $t \prec_{S'_0} \{s\}$ *— i.e., a dependency loop.*

Example 5.6 ($\prec_S$ and minimality)**.** *Consider the following system:*

$$S = \mathsf{A}[\mathsf{do}_s\ \mathsf{a} \mid \mathsf{do}_t\ \overline{\mathsf{b}}] \mid s[\mathsf{A}\ \textit{says}\ \mathsf{a}.0 \mid \mathsf{B}\ \textit{says}\ \overline{\mathsf{a}};0] \mid t[\mathsf{A}\ \textit{says}\ \overline{\mathsf{b}};0 \mid \mathsf{B}\ \textit{says}\ \mathsf{b}.0] \mid \mathsf{B}[\mathsf{do}_t\ \mathsf{b} \mid \mathsf{do}_s\ \overline{\mathsf{a}}]$$

According to Definition 5.4, we have $s \prec_S \emptyset$ and $t \prec_S \emptyset$, with A *culpable at t and* B *culpable at s. Without the minimality requirement, we could also have $s \prec_S \{t\}$ and $t \prec_S \{s\}$ — i.e., seemingly a dependency loop, as found e.g. in the deadlock of Example 5.5. The minimality condition allows for distinguishing "real" dependency loops (and deadlocks).*

A negative result about session dependencies is that they are *not* decidable. This is obtained by simulating Turing machines, similarly to [4].

Lemma 5.7. *Session dependencies are undecidable.*

5.1 Session dependencies hypergraph

The session dependency relations in a system S naturally induce a directed hypergraph [7], whose nodes are the (free) sessions in S. For easier readability, in Definition 5.8 below we denote each node with both the session name, and the name of the participant who is culpable therein.

Definition 5.8 (Session dependency hypergraph)**.** *For all S, we define the* session dependency (directed) hypergraph $\mathcal{G}(S) = (N, E)$ *where:*

- $N = \{\langle \mathsf{A}:s\rangle \mid \mathsf{A}\ \textit{is culpable at}\ s\ \textit{in}\ S\}$*;*
- $(\{\langle \mathsf{A}:s\rangle\}, \{\langle \mathsf{B_1}:t_1\rangle, \ldots, \langle \mathsf{B_n}:t_n\rangle\}) \in E$ *iff* $\{\langle \mathsf{A}:s\rangle, \langle \mathsf{B_1}:t_1\rangle, \ldots, \langle \mathsf{B_n}:t_n\rangle\} \subseteq N$ *and* $s \prec_S \{t_1, \ldots, t_n\}$.

We write $\mathrm{conn}(\mathcal{G}(S))$ *for the set of connected pairs in $\mathcal{G}(S)$. With a slight abuse of notation, we write $v \in \mathcal{G}(S)$ when $v \in N$ and $(v, V') \in \mathcal{G}(S)$ when $(v, V') \in E$.*

Note that the hypergraph arcs may also connect a node to an empty set of nodes: for example, if in a system S we have $s \prec_S \emptyset$ with A culpable at s, then $(\langle \mathsf{A}:s\rangle, \emptyset) \in \mathcal{G}(S)$.

The session dependency hypergraph allows for a simple (and easy to visualize) definition of the dependencies arising in a CO_2 system.

Definition 5.9 (Session dependency degrees)**.** *Given a system S and a participant* A *culpable at s in S, we define the following* session dependency degrees *(in ascending order):*

1. A is innocent at s in S *iff* $\langle \mathsf{A}:s\rangle \notin \mathcal{G}(S)$*;*
2. A is buck-passer at s in S *iff* $\exists v\,.\,(\langle \mathsf{A}:s\rangle, \{v, \cdots\}) \in \mathcal{G}(S)$ *and* $(v, \langle \mathsf{A}:s\rangle) \notin \mathrm{conn}(\mathcal{G}(S))$*;*
3. A is dependently culpable at s in S *iff there exists* $(\langle \mathsf{A}:s\rangle, V) \in \mathcal{G}(S)$*, with* $V \neq \emptyset$*;*
4. A is terminally culpable at s in S *iff there is no* $(\langle \mathsf{A}:s\rangle, \cdot) \in \mathcal{G}(S)$.

Intuitively, item 2 of Definition 5.9 says that A's advancement at s depends on someone else's advancement at another session — and since there is t s.t. $s \prec_S t \wedge t \not\prec_S^* s$, A is involved in at least one dependency path that is *not* a loop. Item 3 says that A's advancement at s depends on some other participant advancing at another session — but loops are *not* ruled out. Item 4 says that A will be unable to fulfil her contractual duties at s, and her culpability does not depend on any other session.

Lemma 5.10 (Dependency degree implications)**.** *For all* A*, s and S, we have:*

- A *culpable at s in S* $\impliedby$ A *dependently culpable at s in S* $\impliedby$ A *buck-passer at s in S;*
- A *culpable at s in S* $\impliedby$ A *terminally culpable at s in S;*
- A *terminally culpable at s in S* $\implies$ A not *dependently culpable at s in S.*

5.2 Session dependency examples

In this section we illustrate session dependencies (and their hypergraph) with some examples. We also show how session dependencies evolve along the reductions of the system on which they are computed.

For the rest of the examples, let us consider the system:

$$\begin{array}{rcl} S & = & \mathsf{A}[P_{\mathsf{A}}] \mid \mathsf{B}[P_{\mathsf{B}}] \mid \mathsf{C}[P_{\mathsf{C}}] \mid s[\mathsf{A}\ \mathit{says}\ c_s \mid \mathsf{B}\ \mathit{says}\ d_s] \\ & & \mid t_1[\mathsf{A}\ \mathit{says}\ c_{t_1} \mid \mathsf{C}\ \mathit{says}\ d_{t_1}] \mid t_2[\mathsf{A}\ \mathit{says}\ c_{t_2} \mid \mathsf{C}\ \mathit{says}\ d_{t_2}] \end{array}$$

where:

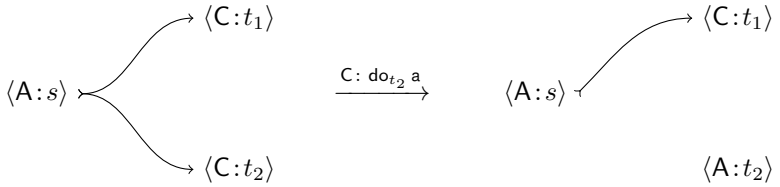
$$\begin{array}{rclcrclcrcl} c_s & = & \overline{\mathsf{b}};\overline{\mathsf{b}};E & \quad & c_{t_1} & = & \mathsf{a}.\mathsf{a}.E & \quad & c_{t_2} & = & \mathsf{a}.\mathsf{a}.E \\ d_s & = & \mathsf{b}.\mathsf{b}.E & & d_{t_1} & = & \overline{\mathsf{a}};\overline{\mathsf{a}};E & & d_{t_2} & = & \overline{\mathsf{a}};\overline{\mathsf{a}};E \end{array}$$

and A is culpable at s, and C is culpable at t_1 and t_2.

Example 5.11 (Reducing the cardinality of hyperarcs (I))**.** *Let:*

$$P_{\mathsf{A}} = \mathsf{do}_{t_1}\,\mathsf{a}\,.\,\mathsf{do}_{t_2}\,\mathsf{a}\,.\,\mathsf{do}_{s}\,\overline{\mathsf{b}}$$

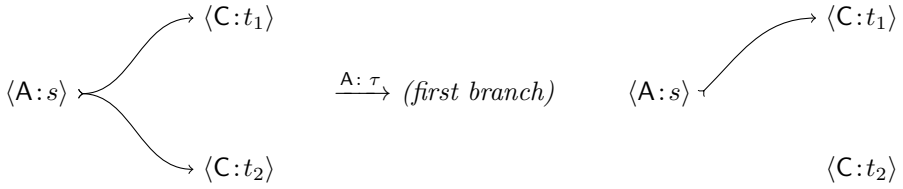
In order to eventually reduce along s, $\mathsf{A}[P_{\mathsf{A}}]$ sequentially depends on both t_1 and t_2. Therefore, $\mathcal{G}(S)$ features a hyperarc connecting s to both t_1 and t_2. When t_2 advances, such hyperarc will only point to t_1. Note that A becomes culpable at t_2.



Example 5.12 (Reducing the cardinality of hyperarcs (II))**.** *Let:*

$$P_{\mathsf{A}} = \tau\,.\,\mathsf{do}_{t_1}\,\mathsf{a}\,.\,\mathsf{do}_{s}\,\overline{\mathsf{b}} \;+\; \tau\,.\,\mathsf{do}_{t_2}\,\mathsf{a}\,.\,\mathsf{do}_{s}\,\overline{\mathsf{b}}$$

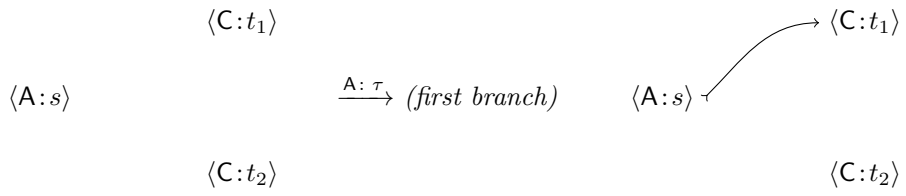
In order to eventually reduce along s, $\mathsf{A}[P_{\mathsf{A}}]$ depends on both t_1 and t_2: in fact, if e.g. t_1 is stuck, and the top-level choice commits to the corresponding branch, then no action will be performed on s. The same holds for t_2. Therefore, $\mathcal{G}(S)$ features a hyperarc connecting s to both t_1 and t_2. After one of the top-level branches is chosen, the cardinality of such hyperarc decreases.



Example 5.13 (Generating new hyperarcs)**.** *Let:*

$$P_{\mathsf{A}} = \tau\,.\,\mathsf{do}_{t_1}\,\mathsf{a}\,.\,\mathsf{do}_{s}\,\overline{\mathsf{b}} \;+\; \tau\,.\,\mathbf{0}$$

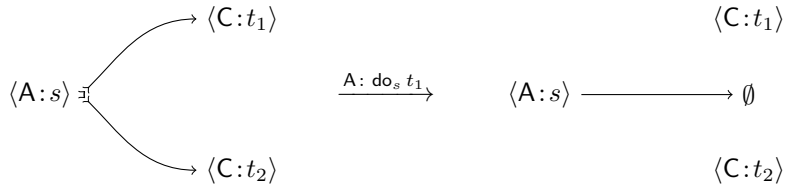
P_{A} is dishonest: if the top-level choice commits to the second branch, then A will be persistently culpable at s. Therefore, s has no dependencies, and $\mathcal{G}(S)$ has no hyperarcs originating from s — but if $S \xrightarrow{\mathsf{A}:\,\tau} S'$ by committing to the first branch, then $\mathcal{G}(S')$ will feature an arc connecting s to t_1.



Example 5.14 (Multiple hyperarcs and empty dependencies)**.** *Let:*

$$P_{\mathsf{A}} = \mathsf{do}_{t_1}\,\mathsf{a}\,.\,\mathsf{do}_{s}\,\overline{\mathsf{b}} \;+\; \mathsf{do}_{t_2}\,\mathsf{a}\,.\,\mathsf{do}_{s}\,\overline{\mathsf{b}}$$

The advancement of A *on* s *depends on the advancement of either* t_1 *or* t_2*. Therefore,* $\mathcal{G}(S)$ *features two distinct hyperarcs. After* S *riduces along either* t_1 *or* t_2*, the cardinality of the corresponding hyperarc decreases, and we obtain an hyperarc pointing from* s *to an empty set of nodes:* A *can now advance at* s*, without depending on other sessions.*

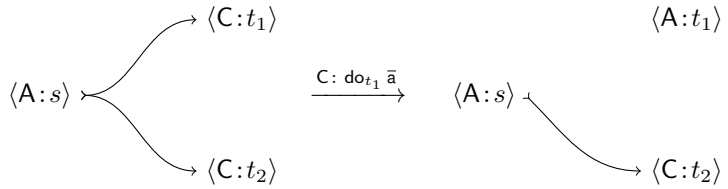


Example 5.15 (System and dependency graph evolution)**.** *Let:*

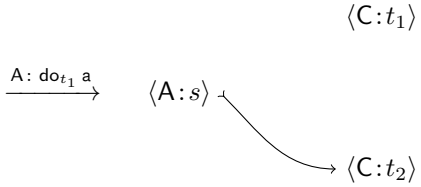
$$P_{\mathsf{A}} = \mathsf{do}_{t_1}\,\mathsf{a}\,.\,\mathsf{do}_{t_2}\,\mathsf{a}\,.\,\mathsf{do}_{s}\,\overline{\mathsf{b}}\,.\,\mathsf{do}_{t_1}\,\mathsf{a}\,.\,\mathsf{do}_{t_2}\,\mathsf{a}\,.\,\mathsf{do}_{s}\,\overline{\mathsf{b}}$$

We show the session dependencies hypergraph corresponding to the evolution of S*.*

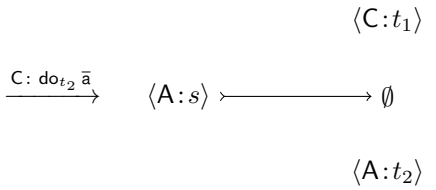
Initially, s *depends on* t_1, t_2 *— and therefore we have an hyperarc from the former to the latter. After* t_1 *advances, its cardinality is reduced, since now* s *only depends on* t_2*. Note that* A *becomes culpable at* t_1*.*



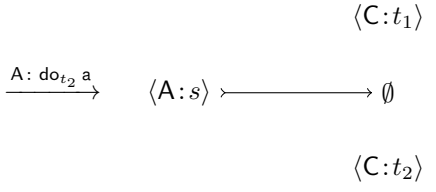
When A *performs its contractual duties at* t_1*,* C *becomes culpable therein — but the overall shape of the hypergraph does not change.*



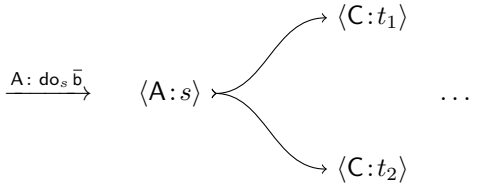
It is now C*'s turn to advance at* t_2*. This event makes the hyperarc cardinality reduce further, thus obtaining an empty hyperarc originating from* s*:* A *can now advance at* s*, without depending on other sessions. Note, however, that* A *becomes culpable at* t_2*.*



When A *performs its contractual duties at* t_2, C *becomes culpable therein — but the overall shape of the hypergraph does not change.*



When A *advances at* s*, the shape of the graph completely changes. According to its contract,* A *is still culpable at* s*, and the dependencies for executing the next step are computed on the remaining part of* P_{A} *(i.e.,* $\mathsf{do}_{t_1}\,\mathsf{a}\,.\,\mathsf{do}_{t_2}\,\mathsf{a}\,.\,\mathsf{do}_{s}\,\overline{\mathsf{b}}$*).*



6 Conclusions: towards blame shifting

We have discussed some work in progress about a theory of honesty and its variants in contract-oriented systems. The theory builds upon two basic notions, i.e. honesty and weak honesty. The classes $\mathcal{H}$ (Definition 3.3) and $\mathcal{W}$ (Definition 4.1) represent two extremes in a (hypothetical) taxonomy of classes of honesty. At the first extreme, there is a rather strict class $\mathcal{H}$ of processes, which always manage to respect their contracts without the help of the context. Systems of honest participants guarantee some nice properties, e.g. no agent is permanently culpable, and the overall system enjoys progress. However, this comes at a cost. In practice, honest processes either realize their contracts by operating independently on the respective sessions, or they exploit "escape options" in contracts to overcome the dependance from the context. At the other extreme, we have a larger class $\mathcal{W}$ of processes, which are more realistically implementable, and still enjoy some decent properties. However (unlike honesty), weak honesty does not guarantee progress, e.g. a system of weakly honest participants might get stuck in a deadlock.

We conjecture that some relevant relations exist between (weak) honesty and dependency hypergraphs, and thus with the culpability degrees of Definition 5.9. For honesty, it seems that a process P is in $\mathcal{H}$ iff, in each reduct of an execution context in which $\mathsf{A}[P]$ is put, if A is culpable at s then it can always exculpate herself — and therefore, the dependency hypergraph has an edge connecting s to the empty set. Instead, for weak honesty it seems that a process P is in $\mathcal{W}$ iff, in each reduct of an execution context in which $\mathsf{A}[P]$ is put, if A is culpable at s then she can either exculpate "by herself" (i.e., the dependency hypergraph has an edge from s to the empty set) or she is dependently culpable (i.e., there is an edge from s to some other session).

The dependent culpability of weakly honest processes may be a useful property in some scenarios (in the spirit of *"company in distress makes sorrow less"*). However, dependent

culpability is not "strong enough" to guarantee that a weakly honest agent will always *shift the blame* in a meaningful way. Suppose, for example, that $\mathsf{A}[P]$ and $\mathsf{B}[Q]$ are weakly honest, and A becomes culpable at some s because B is culpable at some t; a limitation of dependent culpability is that it does not consider dependency loops among violations: therefore, due to the possibility of deadlocks in systems of weakly honest processes, B may be culpable at t because A is culpable at s. A reasonable requirement for "proper" blame shifting is that it must happen in a non-circular way: ideally, A should find some other agent(s) to blame, who in turn will not blame her.

Such non-circularity could be guaranteed for a hypothetical class of processes, say "buck-passer processes" that, when culpable at s, will either exculpate themselves or be buck-passers at s (Definition 5.9). However, we conjecture that such hypothetical class *cannot* be expressed with the contract model we used in this paper. The main issue is that the absence of blame loops can only be obtained by restricting not just the "locality" of an agent in the dependency hypergraph (i.e., the sessions she creates along her reductions, and their dependencies towards other sessions), but also the *global* topology of the hypergraph itself (i.e., the way *other* agents create ad interleave their sessions). Finding the less invasive way to address this problem is part of our future work.

References

[1] Massimo Bartoletti, Maurizio Murgia, Alceste Scalas, and Roberto Zunino. Modelling and verifying contract-oriented systems in Maude. `http://tcs.unica.it/software/co2-maude`.

[2] Massimo Bartoletti, Alceste Scalas, Emilio Tuosto, and Roberto Zunino. Honesty by typing. In *Proc. FORTE*, 2013.

[3] Massimo Bartoletti, Emilio Tuosto, and Roberto Zunino. Contract-oriented computing in CO_2. *Scientific Annals in Computer Science*, 22(1):5–60, 2012.

[4] Massimo Bartoletti, Emilio Tuosto, and Roberto Zunino. On the realizability of contracts in dishonest systems. In *Proc. COORDINATION*, 2012.

[5] Massimo Bartoletti and Roberto Zunino. A calculus of contracting processes. In *LICS*, 2010.

[6] Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A theory of contracts for web services. *ACM TOPLAS*, 31(5), 2009.

[7] Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2–3):177 – 201, 1993.

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

[9] Davide Sangiorgi. *An introduction to bisimulation and coinduction.* Cambridge University Press, Cambridge, UK New York, 2012.