On the Realizability of Contracts in Dishonest Systems

Massimo Bartoletti, Emilio Tuosto, Roberto Zunino

On the realizability of contracts in dishonest systems

Massimo Bartoletti[1], Emilio Tuosto[2], and Roberto Zunino[3]

[1] Dipartimento di Matematica e Informatica, Università degli Studi di Cagliari, Italy
[2] Department of Computer Science, University of Leicester, UK
[3] DISI, Università di Trento and COSBI, Italy

Abstract. We develop a theory of contracting systems, where behavioural contracts may be violated by dishonest participants after they have been agreed upon — unlike in traditional approaches based on behavioural types. We consider the contracts of [10], and we embed them in a calculus that allows distributed participants to advertise contracts, reach agreements, query the fulfilment of contracts, and realise them (or choose not to). Our contract theory makes explicit who is culpable at each step of a computation. A participant is honest in a given context S when she is not culpable in each possible interaction with S. Our main result is a sufficient criterion for classifying a participant as honest in all possible contexts.

1 Introduction

Contracts are abstract descriptions of the behaviour of services. They are used to compose services which are *compliant* according to some semantic property, e.g. the absence of deadlocks [6, 9, 10], the satisfacion of a set of constraints [8], or of some logical formula [1, 4, 15]. Most of the existing approaches tacitly assume that, once a set of compliant contracts has been found, then the services that advertised such contracts will behave accordingly. In other words, services are assumed to be *honest*, in that they always respect the promises made.

In open and dynamic systems, the assumption that all services are honest is not quite realistic. In fact, services have different individual goals, are made available by different providers, and possibly do not trust each other. What happens is that services agree upon some contracts, but may then violate them, either intentionally or not. Since this situation may repeatedly occur in practice, it should not be dealt with as the failure of the whole system. Instead, contract violations should be automatically detected and sanctioned by the service infrastructure.

The fact that violations may be sanctioned gives rise to a new kind of attacks, that exploit possible discrepancies between the promised and the runtime behaviour of services. If a service does not accurately 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 *culpable* of a contract violation, despite of the honest intentions of its developer. More formally, the problem is that of deciding if a process *realizes* a contract: when this holds, the process is guaranteed to never be culpable w.r.t. the contract in all the possible execution contexts.

In this paper we develop a formal theory of contract-oriented systems that enjoys a sound criterion for establishing if a process always realizes its contracts. Our theory

combines two basic ingredients: a calculus of contracts, and a calculus of processes that use contracts to interact. Contracts are used by distributed participants to reach agreements; once stipulated, participants can inspect them and decide what to do next.

Ideally, a honest participant is supposed to harmoniously evolve with her contracts; more realistically, our theory also encompasses computations of *dishonest* participants, which may violate at run-time some contracts they have stipulated. A remarkable result (Theorem 2) is that it is always possible to detect who is culpable of a contract violation at each state of a computation. Also, a participant can always exculpate herself by performing the needed actions (Theorems 1 and 3).

Notably, instead of defining an ad-hoc model, we have embedded the contract calculus in [10] within the process calculus CO_2 [2]. To do that, the contracts of [10] have been slightly adapted to define culpability, and CO_2 has been specialized to use these contracts. We have formalised when a participant realizes a contract in a given context, i.e. when she is never (irreparably) culpable in computations with that context, and when she is *honest*, i.e. when she realizes *all* her contracts, in *all* possible contexts. We have proved that the problem of deciding whether a participant is honest or not is undecidable (Theorem 4). Our main contribution (Theorem 6) is a sound criterion for detecting when a participant is honest. Technically this is achieved through a semantics of participants that abstracts away the behaviour of the context. Such semantics allows us to define when a participant always fulfills her contracts, even in the presence of dishonest participants.

Because of space constraints, we include the proofs of all our statements in a separated Technical Report [3].

2 A calculus of contracts

We assume a finite set of *participant names* (ranged over by $\mathsf{A},\mathsf{B},\ldots$) and a denumerable set of *atoms* (ranged over by $\mathsf{a},\mathsf{b},\ldots$). We postulate an involution $co(\mathsf{a})$, also written as $\overline{\mathsf{a}}$, extended to sets of atoms in the natural way.

Def. 1 introduces the syntax of contracts, taking inspiration from [10]. 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 1. Unilateral contracts *are defined by the following grammar:*

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

where (i) *the index set* I *is finite;* (ii) *the atoms in* $\{\mathsf{a}_i\}_{i\in I}$ *are pairwise distinct;* (iii) *the ready prefix may appear at the top-level, only;* (iv) *recursion is guarded.*

Let e *be a distinguished atom such that* $\mathsf{e}=\overline{\mathsf{e}}$*, and with continuation* $E=\mathit{rec}\,X\,.\,\mathsf{e}\,;X$*. We say that* c succeeds *iff either* $c=\mathsf{e}\,;E\oplus d$*, or* $c=\mathsf{e}\,.E+d$*, or* $c=\mathit{ready}\,\mathsf{e}.E$*. We will omit trailing occurrences of* E *in contracts.*

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.*

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

$$\mathsf{A}\ \mathit{says}\ (\mathsf{a};\, c \oplus c') \mid \mathsf{B}\ \mathit{says}\ \overline{\mathsf{a}};\, d \xrightarrow{\mathsf{A}\ \mathit{says}\ \mathsf{a}} \mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says\ ready}\ \overline{\mathsf{a}}.d \qquad [\textsc{IntInt}]$$

$$\mathsf{A}\ \mathit{says}\ (\mathsf{a}.\, c + c') \mid \mathsf{B}\ \mathit{says}\ (\overline{\mathsf{a}}.\, d + d') \xrightarrow{\mathsf{A}\ \mathit{says}\ \mathsf{a}} \mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says\ ready}\ \overline{\mathsf{a}}.d \qquad [\textsc{ExtExt}]$$

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

$$\frac{\mathsf{a} \notin co(\{\mathsf{b}_i\}_{i\in I})}{\mathsf{A}\ \mathit{says}\ \mathsf{a};\, c \oplus c' \mid \mathsf{B}\ \mathit{says}\ \sum_{i\in I} \mathsf{b}_i.\, d_i \xrightarrow{\mathsf{A}\ \mathit{says}\ \mathsf{a}} \mathsf{A}\ \mathit{says}\ E \mid \mathsf{B}\ \mathit{says}\ 0} \qquad [\textsc{IntExtFail}]$$

$$\frac{\{\mathsf{a}\} \neq co(\{\mathsf{b}_i\}_{i\in I})}{\mathsf{A}\ \mathit{says}\ \mathsf{a};\, c \oplus c' \mid \mathsf{B}\ \mathit{says}\ \bigoplus_{i\in I} \mathsf{b}_i;\, d_i \xrightarrow{\mathsf{A}\ \mathit{says}\ \mathsf{a}} \mathsf{A}\ \mathit{says}\ E \mid \mathsf{B}\ \mathit{says}\ 0} \qquad [\textsc{IntIntFail}]$$

$$\frac{(\{\mathsf{a}\} \cup \{\mathsf{a}_i\}_{i\in I}) \cap co(\{\mathsf{b}_i\}_{i\in J}) = \emptyset}{\mathsf{A}\ \mathit{says}\ (\mathsf{a}.\, c + \sum_{i\in I} \mathsf{a}_i.\, c_i) \mid \mathsf{B}\ \mathit{says}\ \sum_{i\in J} \mathsf{b}_i.\, d_i \xrightarrow{\mathsf{A}\ \mathit{says}\ \mathsf{a}} \mathsf{A}\ \mathit{says}\ E \mid \mathsf{B}\ \mathit{says}\ 0} \qquad [\textsc{ExtExtFail}]$$

Fig. 1. Semantics of contracts (symmetric rules for B actions omitted)

Intuitively, the internal sum $\bigoplus_{i\in I} \mathsf{a}_i;\, c_i$ allows to choose one of the branches $\mathsf{a}_i;\, c_i$, to perform the action a_i, and then behave according to c_i. Dually, the external sum $\sum_{i\in I} \mathsf{a}_i.\, c_i$ constrains to wait for the other participant to choose one of the branches $\mathsf{a}_i.\, c_i$, then to perform the corresponding a_i and finally behave according to c_i. Separators ; and . allow us to distinguish singleton *internal* sums (e.g., $\mathsf{a};\, c$) from singleton *external* sums (e.g., $\mathsf{a}.\, c$). The atom e (for "end") enables a participant to successfully terminate, similarly to [10]. This will be reflected in Def. 4. Hereafter, we shall always consider contracts with no free occurrences of recursion variables X. We shall use the binary operators to isolate a branch in a sum: e.g. $(\mathsf{a};\, c) \oplus c'$ where c' is an internal sum. We let ; and . have higher precedence than $\oplus$ and $+$, e.g., $\mathsf{a};\, c \oplus \mathsf{b};\, c' = (\mathsf{a};\, c) \oplus (\mathsf{b};\, c')$.

The evolution of bilateral contracts is modelled by a labelled transition relation $\xrightarrow{\mu}\!\!\!\!\!\rightarrow$ (Def. 2), where labels $\mu = \mathsf{A}\ \mathit{says}\ \mathsf{a}$ model a participant A performing the action a.

Definition 2. *The relation $\xrightarrow{\mu}\!\!\!\!\!\rightarrow$ on bilateral contracts is the smallest relation closed under the rules in Fig. 1 and under the structural congruence relation $\equiv$, defined as the least congruence which includes α-conversion of recursion variables, and satisfies $\mathit{rec}\ X.\ c \equiv c\{^{\mathit{rec}\ X.\ c}/_X\}$ and $\bigoplus_{i\in\emptyset} \mathsf{a}_i;\, c_i \equiv \sum_{i\in\emptyset} \mathsf{a}_i.\, c_i$. Accordingly, empty sums (either internal or external) will be denoted with* 0. *We will* not *omit trailing occurrences of* 0. *Hereafter we shall consider contracts up to $\equiv$.*

In the first three rules in Fig. 1, A and B expose complementary actions $\mathsf{a}, \overline{\mathsf{a}}$. In rule [IntExt], participant A selects the branch a in an internal sum. Participant B is then forced to commit to the corresponding branch $\overline{\mathsf{a}}$ in his external sum: this is done by marking that branch with *ready* $\overline{\mathsf{a}}$ while discarding all the other branches. Participant B will then perform his action in the subsequent step, by rule [Rdy]. In rule [IntInt], both participants make an internal choice; a reaction is possible only if one of the two is

a singleton — B in the rule — namely he can only commit to his unique branch. Were B exposing multiple branches, the transition would not be allowed, to account for the fact that B could pick a conflicting internal choice w.r.t. that of A. In rule [ExtExt], both participants expose external sums with complementary actions, and each of the two can choose a branch (unlike in the case [IntExt], where the internal choice has to move first). In the [*Fail] rules, the action chosen by A is not supported by B. Then, A will reach the success state E, while B will fall into the failure state 0.

Example 1. Let $\gamma = \mathsf{A}\ \mathit{says}\ (\mathsf{a}; c_1 \oplus \mathsf{b}; c_2) \mid \mathsf{B}\ \mathit{says}\ (\overline{\mathsf{a}}.d_1 + \overline{\mathsf{c}}.d_2)$. If the participant A internally chooses to perform a, then γ will take a transition to $\mathsf{A}\ \mathit{says}\ c_1 \mid \mathsf{B}\ \mathit{says ready}\ \overline{\mathsf{a}}.d_1$. Suppose instead that A chooses for perform b, which is not offered by B in his external choice. In this case, γ will take a transition to $\mathsf{A}\ \mathit{says}\ E \mid \mathsf{B}\ \mathit{says}\ 0$, where 0 indicates that B cannot proceed with the interaction. Coherently with [10], below we will characterise this behaviour by saying that the contracts of A and B are *not* compliant.

The following lemma states that bilateral contracts are never stuck unless both participants have contract 0. Actually, if none of the first four rules in Fig. 1 can be applied, the contract can make a transition with one of the [*Fail] rules.

Lemma 1. *A bilateral contract* $\mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says}\ d$ *is stuck iff* $c = d = 0$.

Below we establish that contracts are deterministic. This is guaranteed by the requirement (ii) of Def. 1. Determinism is a very desirable property indeed, because it ensures that the duties of a participant at any given time are uniquely determined by the past actions. Note that the contracts in [10] satisfy distributivity laws like $(\mathsf{a}; c) \oplus (\mathsf{a}; d) = \mathsf{a}; (c \oplus d)$, which allow for rewriting them so that (ii) in Def. 1 holds. Therefore, (ii) is not a real restriction w.r.t. [10].

Lemma 2 (Determinism). *For all* γ, *if* $\gamma \stackrel{\mu}{\twoheadrightarrow} \gamma'$ *and* $\gamma \stackrel{\mu}{\twoheadrightarrow} \gamma''$, *then* $\gamma' = \gamma''$.

Compliance. Below we define when two contracts are *compliant*, in a similar fashion to [10]. Intuitively, two contracts are compliant if whatever sets of choices they offer, there is at least one common option that can make the contracts progress. Differently from [10], our notion of compliance is symmetric, in that we do not discriminate between the participant roles as client and server. Consequently, we do not consider compliant two contracts where only one of the parties is willing to terminate. For example, the buyer contract $\mathsf{ship}; E$ is not compliant with the seller contract $\overline{\mathsf{ship}}.\mathsf{pay}; E$, because the buyer should not be allowed to terminate if the seller still requires to be paid.

Similarly to [10], given two contracts we observe their *ready sets* (Def. 3) to detect when the enabled actions allow them to synchronise correctly.

Definition 3 (Compliance). *For all contracts c, we define the set of sets* $RS(c)$ *as:*

$$RS(0) = \{\emptyset\} \qquad RS(\mathit{ready}\ \mathsf{a}.c) = \{\{\mathit{ready}\}\} \qquad RS(\mathit{rec}\ X.\ c) = RS(c)$$

$$RS(\textstyle\bigoplus_{i\in I} \mathsf{a}_i; c_i) = \{\{\mathsf{a}_i\} \mid i \in I\}\ \mathit{if}\ I \neq \emptyset \qquad RS(\textstyle\sum_{i\in I} \mathsf{a}_i.c_i) = \{\{\mathsf{a}_i \mid i \in I\}\}\ \mathit{if}\ I \neq \emptyset$$

The relation $\bowtie$ *between contracts is the largest relation such that, whenever* $c \bowtie d$:

(1) $\forall x \in RS(c), y \in RS(d).\ co(x) \cap y \neq \emptyset$ *or* $\mathit{ready} \in (x \cup y) \setminus (x \cap y)$

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

When $c \bowtie d$*, we say that the contracts* c *and* d *are* compliant.

Example 2. Recall from Ex. 1 the contracts $c = \mathsf{a}; c_1 \oplus \mathsf{b}; c_2$ and $d = \overline{\mathsf{a}}.d_1 + \overline{\mathsf{c}}.d_2$. We have that $RS(c) = \{\{\mathsf{a}\},\{\mathsf{b}\}\}$, and $RS(d) = \{\{\overline{\mathsf{a}},\overline{\mathsf{c}}\}\}$, which do not respect item (1) of Def. 3 (take $x = \{\mathsf{b}\}$ and $y = \{\overline{\mathsf{a}},\overline{\mathsf{c}}\}$). Therefore, c and d are *not* compliant.

The following lemma provides an alternative characterization of compliance. Two contracts are compliant iff, when combined into a bilateral contract γ, no computation of γ reaches a state where one of the contracts is 0. Together with Lemma 1, we have that such γ will never get stuck. (Below, the Kleene * denotes reflexive transitive closure.)

Lemma 3. *For all bilateral contracts* $\gamma = \mathsf{A}$ *says* $c \mid \mathsf{B}$ *says* d*:*

$$c \bowtie d \iff \big(\forall c', d'.\ \gamma \twoheadrightarrow^* \mathsf{A} \text{ says } c' \mid \mathsf{B} \text{ says } d' \implies c' \neq 0 \text{ and } d' \neq 0\big)$$

The following lemma guarantees, for all c not containing 0, the existence of a contract d compliant with c. Intuitively, we can construct d from c by turning internal choices into external ones (and *viceversa*), and by turning actions into co-actions.

Lemma 4. *For all* 0*-free contracts* c*, there exists* d *such that* $c \bowtie d$*.*

Culpability. 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 action so to make γ progress. Also, we consider A culpable when she is advertising the "failure" contract 0. This agrees with our [*FAIL] rules, which set A's contract to 0 when the other participant legitimately chooses an action not supported by A. Note that we do not consider A culpable when her contract has enabled e actions.

Definition 4. *A participant* A *is culpable in* $\gamma = \mathsf{A}$ *says* $c \mid \mathsf{B}$ *says* d*, written* $\mathsf{A} \frown \gamma$*, iff:*

$$c = 0 \quad \vee \quad \big(\gamma \not\xrightarrow{\mathsf{A} \text{ says } \mathsf{e}}\!\!\!\!\rightarrow \ \wedge\ \exists \mathsf{a}.\ \gamma \xrightarrow{\mathsf{A} \text{ says } \mathsf{a}}\!\!\!\!\rightarrow\big)$$

When A *is* not *culpable in* γ *we write* $\mathsf{A} \smile \gamma$*.*

The following result states that a participant A is always able to recover from culpability by performing some of her duties. Furthermore, this requires at most two steps in an "A-solo" trace where no other participant intervenes.

Definition 5. *Let* $\rightarrow$ *be an LTS with labels of the form* A_i *says* $(\cdots)$*, for* A_i *ranging over participants names. For all* A*, we say that a* $\rightarrow$*-trace* η *is* A*-solo iff* η *only contains labels of the form* A *says* $(\cdots)$*. If* $\eta = (\mu_i)_{i \in 0..n}$*, we will write* $\xrightarrow{\eta}$ *for* $\xrightarrow{\mu_0} \cdots \xrightarrow{\mu_n}$*.*

Theorem 1 (Contractual exculpation). *For all* $\gamma = \mathsf{A}$ *says* $c \mid \mathsf{B}$ *says* d *with* 0*-free* c*, there exists* γ' *and* A*-solo* η *with* $|\eta| \leq 2$ *such that* $\gamma \xrightarrow{\eta}\!\!\!\!\rightarrow \gamma'$ *and* $\mathsf{A} \smile \gamma'$*.*

commutative monoidal laws for $|$ on processes and systems

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

$$(u)Z \equiv Z \ \text{ if } u \notin \mathrm{fv}(Z) \cup \mathrm{fn}(Z) \qquad \mathsf{A}[K] \mid \mathsf{A}[P] \equiv \mathsf{A}[K \mid P] \qquad \downarrow_s c \equiv \mathbf{0} \equiv \mathsf{fuse}_s.P$$

Fig. 2. Structural equivalence for CO_2 (Z, Z' range over systems or processes)

A crucial property of culpability is to ensure that either two participants are both succeeding, or it is possible to single out who has to make the next step. An external judge is therefore always able to detect who is violating the contracts agreed upon.

Theorem 2. *For all c, d if $c \bowtie d$ and A says $c \mid \mathsf{B}$ says $d \twoheadrightarrow^* \gamma = \mathsf{A}$ says $c' \mid \mathsf{B}$ says d', then either c' and d' succeed, or $\mathsf{A} \colon\!\frown \gamma$, or $\mathsf{B} \colon\!\frown \gamma$.*

Example 3. A participant might be culpable even though her contract succeeds. For instance, let $\gamma = \mathsf{A}$ *says* $c \mid \mathsf{B}$ *says* d, where $c = \mathsf{e} + \overline{\mathsf{a}}$ and $d = \mathsf{a} + \mathsf{b}$. By Def. 1 we have that c succeeds, but A is culpable in γ because she cannot fire e, while she can fire $\overline{\mathsf{a}}$ by rule [ExtExt]. This makes quite sense, because A is saying that she is either willing to terminate or to perform $\overline{\mathsf{a}}$, but the other participant is not allowing A to terminate. Note that also B is culpable, because he can fire a.

3 A Calculus of Contracting Processes

We now embed the contracts introduced in § 2 in a specialization of the parametric process calculus CO_2 [2]. Let $\mathcal{V}$ and $\mathcal{N}$ be two disjoint countably infinite sets of *session variables* (ranged over by $x, y, \ldots$) and *session names* (ranged over by $s, t, \ldots$). Let $u, v, \ldots$ range over $\mathcal{V} \cup \mathcal{N}$.

Definition 6. *The abstract syntax of CO_2 is given by the following productions:*

Systems	$S ::=$	$\mathbf{0}$	$\mathsf{A}[P]$	$s[\gamma]$	$S \mid S$	$(u)S$	
Processes	$P ::=$	$\downarrow_u \mathsf{A}$ *says* c	$\sum_i \pi_i.P_i$	$P \mid P$	$(u)P$	$X(\vec{u})$	
Prefixes	$\pi ::=$	τ	$\mathsf{tell}_{\mathsf{A}} \downarrow_u c$	fuse_u	$\mathsf{do}_u\,\mathsf{a}$	$\mathsf{ask}_u\,\phi$	

The only binder for session variables and names is the delimitation $(\vec{u})$, both in systems and processes. Free variables/names are defined accordingly, and they are denoted by $\mathrm{fv}(\_)$ and $\mathrm{fn}(\_)$. A system or a process is *closed* when it has no free variables.

Systems are the parallel composition of *participants* $\mathsf{A}[P]$ and *sessions* $s[\gamma]$.

A *latent contract* $\downarrow_x \mathsf{A}$ *says* c represents a contract c (advertised by A) which has not been stipulated yet; upon stipulation, x will be instantiated to a fresh session name. We impose that in a system $\mathsf{A}[P] \mid \mathsf{A}[Q] \mid S$, either P or Q is a parallel composition of latent contracts. Hereafter, $K, K', \ldots$ are meta-variables for compositions of latent contracts. We allow prefix-guarded finite sums of processes, and write $\pi_1.P_1 + \pi_2.P_2$ for $\sum_{i=1,2} \pi_i.P_i$, and $\mathbf{0}$ for $\sum_\emptyset P$. Recursion is allowed only for processes; for this we

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

$$\mathsf{A}[\mathsf{tell}_{\mathsf{B}} \downarrow_x c.P + P' \mid Q] \to \mathsf{A}[P \mid Q] \mid \mathsf{B}[\downarrow_x \mathsf{A}\ \mathit{says}\ c] \qquad [\textsc{Tell}]$$

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

$$\frac{\gamma \xrightarrow{\mathsf{A}\ \mathit{says}\ \mathsf{a}} \gamma'}{s[\gamma] \mid \mathsf{A}[\mathsf{do}_s\,\mathsf{a}.P + P' \mid Q] \to s[\gamma'] \mid \mathsf{A}[P \mid Q]} \qquad [\textsc{Do}]$$

$$\frac{\gamma \vdash \phi}{\mathsf{A}[\mathsf{ask}_s\,\phi.P + P' \mid Q] \mid s[\gamma] \to \mathsf{A}[P \mid Q] \mid s[\gamma]} \qquad [\textsc{Ask}]$$

$$\frac{X(\vec{u}) \stackrel{\text{def}}{=} P \qquad P\{{}^{\vec{v}}/_{\vec{u}}\} \to P'}{X(\vec{v}) \to P'}\ [\textsc{Def}] \qquad \frac{S \to S'}{S \mid S'' \to S' \mid S''}\ [\textsc{Par}] \qquad \frac{S \to S'}{(u)S \to (u)S'}\ [\textsc{Del}]$$

Fig. 3. Reduction semantics of CO_2

stipulate that each process identifier X has a unique defining equation $X(u_1,\ldots,u_j) \stackrel{\text{def}}{=} P$ such that $\mathrm{fv}(P) \subseteq \{u_1,\ldots,u_j\} \subseteq \mathcal{V}$ and each occurrence of process identifiers in P is prefix-guarded. We shall take the liberty of omitting the arguments of $X(\vec{u})$ when they are clear from the context.

Prefixes include silent action τ, contract advertisement $\mathsf{tell}_{\mathsf{A}} \downarrow_u c$, contract stipulation fuse_u, action execution $\mathsf{do}_u\,\mathsf{a}$, and contract query $\mathsf{ask}_u\,\phi$. In each prefix $\pi \neq \tau$, u refers to the target session involved in the execution of π. We omit trailing occurrences of $\mathbf{0}$.

Note that participants can only contain latent contracts, while sessions can only contain bilateral contracts, constructed from latent contracts upon reaching agreements.

The semantics of CO_2 is formalised by a reduction relation $\to$ on systems that relies on the structural congruence defined in Fig. 2, where the last law allows for collecting garbage terms possibly arising from variable substitutions.

Definition 7. *The relation $\to$ is the smallest relation closed under the rules of Fig. 3, defined over systems up to structural equivalence, as defined in Fig. 2. The relation $K \rhd_x^{\sigma} \gamma$ holds iff (i) K has the form $\downarrow_y \mathsf{A}$ says $c \mid \downarrow_z \mathsf{B}$ says d, (ii) $c \bowtie d$, (iii) $\gamma = \mathsf{A}$ says $c \mid \mathsf{B}$ says d, and (iv) $\sigma = \{{}^{s}/_{x,y,z}\}$ maps all $x,y,z \in \mathcal{V}$ to $s \in \mathcal{N}$.*

Rule [Tau] simply fires a τ prefix as expected. Rule [Tell] advertises a latent contract $\downarrow_x \mathsf{A}$ *says* c, by putting it in parallel with the existing participants and sessions (the structural congruence laws in Fig. 2 allow for latent contracts to float in a system and, by the second last law, to move across the boxes of participants as appropriate). Rule [Fuse] finds agreements among the latent contracts K of A; an agreement is reached when K contains a bilateral contract γ whose unilater contracs are compliant (cf. Def. 7). Note that, once the agreement is reached, the compliant contracts start a fresh session containing γ. Rule [Do] allows a participant A to fulfill her contract γ, by performing the needed actions in the session containing γ (which, accordingly, evolves to γ'). Rule [Ask] checks if a condition ϕ holds in a session. The actual nature of ϕ is almost immaterial in this paper: the reader may assume that ϕ is a formula in an LTL logic [13]. For

closed γ and ϕ, $\gamma \vdash \phi$ holds iff $\gamma \models_{LTL} \phi$ according to the standard LTL semantics where, for a $\twoheadrightarrow$-trace $\eta = (\gamma_i \xrightarrow{\mu_i} \twoheadrightarrow \gamma_{i+1})_i$ from $\gamma_0 = \gamma$, we define $\eta \models \mathsf{a} \iff \exists \mathsf{A}.\ \mu_0 = \mathsf{A}\ \mathit{says}\ \mathsf{a}$. The last three rules are standard.

Hereafter it will be sometimes useful to record the prefix π fired by A by implicitly decorating the corresponding reduction step, as in $\xrightarrow{\mathsf{A}:\pi}$.

The rest of this section is devoted to a few examples that highlight how bilateral contracts can be used in CO_2.

Example 4. Consider an online store A with the following contract c_A: buyers can add items to the shopping cart, and then either leave the store or pay with a credit card. Assume the store modelled as the CO_2 process $P_\mathsf{A} = (x)\,(\mathsf{tell}_\mathsf{A} \downarrow_x c_\mathsf{A}.X \mid \mathsf{fuse}_x)$, where:

$$c_\mathsf{A} = \mathit{rec}\ Z.\ \mathsf{addToCart}.Z + \mathsf{creditCard}.(\overline{\mathsf{ok}} \oplus \overline{\mathsf{no}}) + \mathsf{e}$$
$$X \stackrel{\text{def}}{=} \mathsf{do}_x\,\mathsf{addToCart}.X + \mathsf{do}_x\,\mathsf{creditCard}.(\tau.\mathsf{do}_x\,\overline{\mathsf{ok}} + \tau.\mathsf{do}_x\,\overline{\mathsf{no}})$$

Let B be a buyer with contract $c_\mathsf{B} = \overline{\mathsf{addToCart}};\ \overline{\mathsf{creditCard}};\ (\mathsf{ok} + \mathsf{no})$, and let:

$$P_\mathsf{B} = (y)\,\mathsf{tell}_\mathsf{A} \downarrow_y c_\mathsf{B}.Y \qquad\qquad Y \stackrel{\text{def}}{=} \mathsf{do}_y\,\overline{\mathsf{addToCart}}.\,\mathsf{do}_y\,\overline{\mathsf{creditCard}}.\,\mathsf{do}_y\,\mathsf{ok}$$

A possible, successful, computation of the system $S = \mathsf{A}[P_\mathsf{A}] \mid \mathsf{B}[P_\mathsf{B}]$ is the following:

$$\begin{aligned}
S &\to^*(x,y)\ \big(\mathsf{A}[\downarrow_x \mathsf{A}\ \mathit{says}\ c_\mathsf{A} \mid \downarrow_y \mathsf{B}\ \mathit{says}\ c_\mathsf{B} \mid \mathsf{fuse}_x \mid X] \mid \mathsf{B}[Y]\big)\\
&\to\ (s)\ \big(\mathsf{A}[X\{s/x\}] \mid \mathsf{B}[Y\{s/y\}] \mid s[\mathsf{A}\ \mathit{says}\ c_\mathsf{A} \mid \mathsf{B}\ \mathit{says}\ c_\mathsf{B}]\big)\\
&\to^*(s)\ \big(\mathsf{A}[X\{s/x\}] \mid \mathsf{B}[\mathsf{do}_s\,\overline{\mathsf{creditCard}}.\,\mathsf{do}_y\,\mathsf{ok}] \mid s[\mathsf{A}\ \mathit{says}\ c_\mathsf{A} \mid \mathsf{B}\ \mathit{says}\ \overline{\mathsf{creditCard}};\ (\mathsf{ok}+\mathsf{no})]\big)\\
&\to^*(s)\ \big(\mathsf{A}[\tau.\mathsf{do}_x\,\overline{\mathsf{ok}} + \tau.\mathsf{do}_x\,\overline{\mathsf{no}}] \mid \mathsf{B}[\mathsf{do}_y\,\mathsf{ok}] \mid s[\mathsf{A}\ \mathit{says}\ \overline{\mathsf{ok}} \oplus \overline{\mathsf{no}} \mid \mathsf{B}\ \mathit{says}\ \mathsf{ok}+\mathsf{no}]\big)\\
&\to\ (s)\ \big(\mathsf{A}[\mathsf{do}_x\,\overline{\mathsf{ok}}] \mid \mathsf{B}[\mathsf{do}_y\,\mathsf{ok}] \mid s[\mathsf{A}\ \mathit{says}\ \overline{\mathsf{ok}} \oplus \overline{\mathsf{no}} \mid \mathsf{B}\ \mathit{says}\ \mathsf{ok}+\mathsf{no}]\big)\\
&\to^*(s)\ \big(\mathsf{A}[\mathbf{0}] \mid \mathsf{B}[\mathbf{0}] \mid s[\mathsf{A}\ \mathit{says}\ E \mid \mathsf{B}\ \mathit{says}\ E]\big)
\end{aligned}$$

Example 5. An on-line store A offers buyers two options: clickPay or clickVoucher. If a buyer B chooses clickPay, A accepts the payment (pay) otherwise A checks the validity of the voucher with V, an electronic voucher distribution and management system. If V validates the voucher, B can use it (voucher), otherwise he will pay.

The contracts $c_\mathsf{A} = \mathsf{clickPay}.\mathsf{pay} + \mathsf{clickVoucher}.(\overline{\mathsf{reject}};\mathsf{pay} \oplus \overline{\mathsf{accept}};\mathsf{voucher})$ and $c'_\mathsf{A} = \mathsf{ok} + \mathsf{no}$ model the scenario above. A CO_2 process for A can be the following

$$\begin{aligned}
P_\mathsf{A} &= (x)(\mathsf{tell}_\mathsf{A} \downarrow_x c_\mathsf{A}.(\mathsf{do}_x\,\mathsf{clickPay}.\mathsf{do}_x\,\mathsf{pay} + \mathsf{do}_x\,\mathsf{clickVoucher}.((y)\mathsf{tell}_\mathsf{V} \downarrow_y c'_\mathsf{A}.X)))\\
X &= \mathsf{do}_y\,\mathsf{ok}.\mathsf{do}_x\,\overline{\mathsf{accept}}.\mathsf{do}_x\,\mathsf{voucher} + \mathsf{do}_y\,\mathsf{no}.\mathsf{do}_x\,\overline{\mathsf{reject}}.\mathsf{do}_x\,\mathsf{pay} + \tau.\mathsf{do}_x\,\overline{\mathsf{reject}}.\mathsf{do}_x\,\mathsf{pay}
\end{aligned}$$

Contract c_A (resp. c'_A) is stipulated when (i) B (resp. V) advertises to A (resp. V) a contract d with $c_\mathsf{A} \bowtie d$ (resp. $c'_\mathsf{A} \bowtie d$) and (ii) a fuse_z is executed in A (resp. V).

Variables x and y in P_A correspond to two separate sessions, where A respectively interacts with B and V. The semantics of CO_2 ensures that x and y will be instantiated to different session names (if at all).

The advertisement of c'_A causally depends on the stipulation of the contracts of A and B, otherwise A cannot fire $\mathsf{do}_x\,\mathsf{clickVoucher}$. Instead, A and B can interact regardless the presence of V since $\mathsf{tell}_\mathsf{V} \downarrow_y c'_\mathsf{A}$ is non blocking and the τ-branch of A in X is enabled (letting A to autonomously reject the voucher, e.g. because B is not entitled to use it).

Example 6. Consider a travel agency A which queries in parallel an airline ticket broker F and a hotel reservation service H in order to complete the organization of a trip. The travel agency service $\mathsf{A}[P]$ can be defined as follows:

$$P = (x,y)(\mathsf{tell}_{\mathsf{F}}\downarrow_x \mathsf{ticket};\,(\mathsf{commitF}\oplus\mathsf{abortF}).X \mid \mathsf{tell}_{\mathsf{H}}\downarrow_y \mathsf{hotel};\,(\mathsf{commitH}\oplus\mathsf{abortH}).Y)$$

$$X \stackrel{\text{def}}{=} \mathsf{do}_x\,\mathsf{ticket}.\,((\mathsf{ask}_y\,\mathit{true}.\,\mathsf{do}_x\,\mathsf{commitF}) + \tau.\mathsf{do}_x\,\mathsf{abortF})$$

$$Y \stackrel{\text{def}}{=} \mathsf{do}_y\,\mathsf{hotel}.\,((\mathsf{ask}_x\,\mathit{true}.\,\mathsf{do}_y\,\mathsf{commitH}) + \tau.\mathsf{do}_y\,\mathsf{abortH})$$

where the τ actions model timeouts used to ensure progress. The travel agency in process X starts buying a ticket, and commits to it only when the hotel reservation session y is started. Similarly for process Y.

The next example shows a peculiar use of ask whereby a participant inspects a stipulated contract to decide its future behaviour.

Example 7. An online store A can choose whether to abort a transaction ($\overline{\mathsf{abort}}$) or to commit to the payment ($\overline{\mathsf{commit}}$). In the latter case, the buyer has two options, either he pays by credit card ($\mathsf{creditCard}$) or by bank transfer ($\mathsf{bankTransfer}$). The contract of A is modelled as $c = \overline{\mathsf{abort}} \oplus \overline{\mathsf{commit}};\,(\mathsf{creditCard} + \mathsf{bankTransfer})$. Consider the process

$$P_{\mathsf{A}} = (x)(\mathsf{tell}_{\mathsf{A}}\downarrow_x c.\,(\mathsf{ask}_x\,\phi.\,\mathsf{do}_x\,\overline{\mathsf{commit}}.\,\mathsf{do}_x\,\mathsf{creditCard} + \mathsf{do}_x\,\overline{\mathsf{abort}}))$$

where $\phi = \Box(\overline{\mathsf{commit}} \to \neg\Diamond\mathsf{bankTransfer})$. The process P_{A} first advertises c. Once a session $s[\gamma]$ is initiated with $\gamma = \mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says}\ d$, A tests γ through $\mathsf{ask}_x\,\phi$ before committing to the payment. If $\mathsf{ask}_x\,\phi$ detects that B has promised not to use the bank transfer option, then A commits to the payment, and then never offers B to perform a bank transfer. Otherwise, if d does not rule out the bank transfer, even if B might actually pay by credit card, A aborts the session. Note that in both cases A realizes her own contract, even if she is never performing the bank transfer. This notion of "realization of a contract" will be formalized in Def. 11.

4 On honesty

In this section we set out when a participant A is honest (Def. 11). Intuitively, we consider all the possible runs of all possible systems, and require that in every session A is not definitely culpable. To this aim, we first provide CO$_2$ with the counterpart of the (non)culpability relation introduced in Def. 4. Intuitively, we write $\mathsf{A} \smile_s S$ when, in the system S, if the participant A is involved in the session s, then she is not culpable w.r.t. the contract stipulated therein.

Definition 8. *We write* $\mathsf{A} \smile_s S$ *whenever* $\forall \vec{u}, \gamma, S'.\ \big(S \equiv (\vec{u})(s[\gamma] \mid S') \implies \mathsf{A} \smile \gamma\big)$. *We write* $\mathsf{A} \smile S$ *whenever* $\mathsf{A} \smile_s S$ *for all session names* s.

A technical issue is that a participant may not get a chance to act in all the traces. For instance, let $S = \mathsf{A}[\mathsf{do}_s\,\mathsf{pay}] \mid \mathsf{B}[X] \mid S'$, where S' enables A's action and $X \stackrel{\text{def}}{=} \tau.X$; note that S generates the infinite trace $S \to S \to S \to \cdots$ in which A never pays, despite her honest intention. To account for this fact, we will check the honesty of a participant in *fair* traces, only, i.e. those where persistent transitions are eventually followed.

Definition 9. *Given an LTS $\xrightarrow{\mu}$, we say that a (finite or infinite) trace $\eta = (P_i \xrightarrow{\mu_i} P_{i+1})_i$ having length $|\eta| \in \mathbb{N} \cup \{\infty\}$ is* fair *w.r.t. a set of labels $\mathcal{L}$ if and only if*

$$\forall i \in \mathbb{N}, \mu \in \mathcal{L}. \Big(i \leq |\eta| \wedge (\forall j \in \mathbb{N}.\, i \leq j \leq |\eta| \implies P_j \xrightarrow{\mu}) \implies \exists j \geq i.\, \mu_j = \mu \Big)$$

A fair trace is a trace which is fair w.r.t. all the labels in the LTS.

Note that, by Def. 9, a fair trace is also a maximal one (w.r.t. $\mathcal{L}$). Indeed, if a fair trace is finite, the condition above guarantees that its final state has no $\mathcal{L}$ transitions enabled.

Finally, when checking the fairness of a trace, we shall implicitly assume that the labels μ in our LTSs of contracts and processes always distinguish between different occurrences of the same prefix. E.g., a $\rightarrow$-fair trace of $\mathsf{A}[X \mid X]$ where $X \stackrel{\text{def}}{=} \tau.X$ is not allowed to only perform the τ's of the first X. Technically, labels μ always implicitly carry the syntactic *address* of the prefix which is being fired, in the spirit of the Enhanced Structured Operational Semantics [12].

It is often useful to reason about how a specific session s evolves in a given trace. Technically, α-conversion allows the name s to be renamed at every step, making it hard to trace the identity of names. More concretely, α-conversion is only needed to make delimitations fresh when unfolding recursive processes. Accordingly, w.l.o.g. hereafter we shall often restrict α-conversion by considering *stable* traces, only, defined below. In this way, we ensure that s represents the same session throughout the whole trace.

Definition 10. *A* stable *$\rightarrow$-trace is a trace $(\vec{u}_0)S_0 \rightarrow (\vec{u}_1)S_1 \rightarrow (\vec{u}_2)S_2 \rightarrow \cdots$ in which* (1) *all delimitations carry distinct names and variables,* (2) *delimitations have been brought to the top-level as much as possible (using $\equiv$), and* (3) *no α-conversion is performed in the trace except when unfolding recursive processes.*

Below, we define several notions of contract faithfulness for participants. We start by clarifying when a participant A *realizes* a contract (inside a session s) within a specific context. This happens when from any reachable system state S_0, participant A will eventually perform actions to exculpate herself (in s). In this phase, A is protected from interference with other participants. Then, we say A *honest in a system* if she realizes every contract in that system. When $\mathsf{A}[P]$ is honest independently of the system, we simply say that $\mathsf{A}[P]$ is *honest*. In this last case, we rule out those systems carrying stipulated or latent contracts of A outside of $\mathsf{A}[P]$; otherwise the system can trivially make A culpable: e.g., we disallow $\mathsf{A}[P] \mid \mathsf{B}[\downarrow_x \mathsf{A}\ \mathit{says}\ \overline{\mathsf{pay}} \mid \cdots]$.

Definition 11 (Honesty). *We say that:*

- A realizes c at s in S *iff whenever* $S = (\vec{u})(s[\mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says}\ d] \mid S')$, $S \rightarrow^* S_0$, *and* $(S_i)_i$ *is a* $\{\mathsf{A} : \pi\}$*-fair* A*-solo stable $\rightarrow$-trace then* $\mathsf{A} \ddot{\smile}_s S_j$ *for some* $j \geq 0$*;*
- A *is* honest in S *iff for all c and s,* A *realizes c at s in S;*
- $\mathsf{A}[P]$ *is* honest *iff for all S with no* A *says* $\cdots$ *nor* $\mathsf{A}[\cdots]$, A *is honest in* $\mathsf{A}[P] \mid S$.

Example 8. A computation of the store-buyer system $S = \mathsf{A}[P_\mathsf{A}] \mid \mathsf{B}[P_\mathsf{B}]$ from Ex. 4 is:

$$\begin{aligned} S &\rightarrow^* (s)\ \big(\mathsf{A}[\tau.\mathsf{do}_x\,\overline{\mathsf{ok}} + \tau.\mathsf{do}_x\,\overline{\mathsf{no}}] \mid \mathsf{B}[\mathsf{do}_y\,\mathsf{ok}] \mid s[\mathsf{A}\ \mathit{says}\ \overline{\mathsf{ok}} \oplus \overline{\mathsf{no}} \mid \mathsf{B}\ \mathit{says}\ \mathsf{ok} + \mathsf{no}]\big) \\ &\rightarrow\ (s)\ \big(\mathsf{A}[\mathsf{do}_x\,\overline{\mathsf{no}}] \mid \mathsf{B}[\mathsf{do}_y\,\mathsf{ok}] \mid s[\mathsf{A}\ \mathit{says}\ \overline{\mathsf{ok}} \oplus \overline{\mathsf{no}} \mid \mathsf{B}\ \mathit{says}\ \mathsf{ok} + \mathsf{no}]\big) \\ &\rightarrow\ (s)\ \big(\mathsf{A}[\mathbf{0}] \mid \mathsf{B}[\mathsf{do}_y\,\mathsf{ok}] \mid s[\gamma]\big) \end{aligned}$$

where $\gamma = \mathsf{A}$ *says* $E \mid \mathsf{B}$ *says ready* no. The system is then stuck, because γ is not allowing the [DO] step. By Def. 4 we have $\mathsf{A} \smile \gamma$, $\mathsf{B} \frown \gamma$, so A is honest in S while B is not. Actually, B has violated the contract agreed upon, because he is waiting for a positive answer from the store, while in c_{B} he also promised to accept a $\overline{\mathsf{no}}$. By Def. 11, B is not honest, while we will show in § 5 that A is honest (see Ex. 10).

Example 9. Consider the system $\mathsf{A}[(x,y)\ (P_A \mid \mathsf{fuse}_x \mid \mathsf{fuse}_y)] \mid \mathsf{B}[P_B] \mid \mathsf{C}[P_C]$, where:

$$P_A \stackrel{\text{def}}{=} \mathsf{tell}_{\mathsf{A}}\,(\downarrow_x \mathsf{a}\,.E)\,.\,\mathsf{tell}_{\mathsf{A}}\,(\downarrow_y \mathsf{b}\,;E)\,.\,\mathsf{do}_x\,\mathsf{a}\,.\,\mathsf{do}_y\,\mathsf{b}$$
$$P_B \stackrel{\text{def}}{=} (z)\ (\mathsf{tell}_{\mathsf{A}}\,(\downarrow_z \overline{\mathsf{b}}\,.E)\,.\,\mathsf{do}_z\,\mathsf{b})$$
$$P_C \stackrel{\text{def}}{=} (w)\ (\mathsf{tell}_{\mathsf{A}}\,(\downarrow_w \overline{\mathsf{a}}\,;E)\,.\,\mathbf{0})$$

Even though A might apparently look honest, she is not. Indeed, A cannot fulfill her contract with B, because the $\mathsf{do}_x\,\mathsf{a}$ is blocked due to the fact that C (dishonestly) does not perform his internal choice. Note that, if we considered honest a participant whose culpability only depends on the culpability of someone else, then a participant could cunningly have one of her contracts violated, so to avoid fulfilling another contract (e.g., to avoid paying one million euros to B, A stipulates a dummy contract "I ship one candy if you pay 1 cent", which is then violated by a colluding participant C).

We now define when a process enables a contract transition, independently from the context. To do that, first we define the set $RD_s(P)$ (after "ready do"), which collects all the atoms with an unguarded action do_s in P.

Definition 12. *For all P and all s, we define the set of atoms $RD_s(P)$ as:*

$$RD_s(P) \;=\; \{\mathsf{a} \mid \exists \vec{u}, P', Q, R\ .\ P \equiv (\vec{u})\ (\mathsf{do}_s\,\mathsf{a}.P' + Q \mid R) \text{ and } s \notin \vec{u}\}$$

Next, we check when a contract "unblocks" a set of atoms $\mathcal{X}$: e.g., if $\mathcal{X}$ accounts for at least one branch of an internal choice, or for all the branches of an external choice.

Definition 13. *For all sets of atoms $\mathcal{X}$ and for all $c \neq 0$, we say that c unblocks $\mathcal{X}$ iff:*

$$\exists Y \in RS(c).\, Y \subseteq \mathcal{X} \cup \{\mathsf{e}\} \qquad \text{or} \qquad c = \mathit{ready}\ \mathsf{a}.c' \ \wedge\ \mathsf{a} \in \mathcal{X} \cup \{\mathsf{e}\}$$

Lemma 5. *For all P and for all $\gamma = \mathsf{A}$ says $c \mid \mathsf{B}$ says d, if c unblocks $RD_s(P)$ and $S = (\vec{u})(\mathsf{A}[P] \mid s[\gamma] \mid S')$, then either $\mathsf{A} \smile \gamma$ or $S \xrightarrow{\mathsf{A}\,:\,\mathsf{do}_s\,\mathsf{a}}$.*

The following theorem is the CO_2 counterpart of Theorem 1. It states that, when a session s is established between two participants A and B, A can always exculpate herself by performing (at most) two actions $\mathsf{A} : \mathsf{do} -$. Note that when the contracts used to establish s are compliant, then we deduce the stronger thesis $\mathsf{A} \smile_s S_j$.

Theorem 3 (Factual exculpation). *Let $(S_i)_i$ be the following* A*-solo stable $\rightarrow$-trace, with $S_i = (\vec{u}_i)\ \big(\mathsf{A}[Q_i] \mid s[\mathsf{A}$ says $c_i \mid \mathsf{B}$ says $d_i] \mid S_i'\big)$, and:*

$$S_0 \xrightarrow{\mu_0} \cdots \xrightarrow{\mu_{i-2}} S_{i-1} \xrightarrow{\mathsf{A}\,:\,\mathsf{do}_s\,\mathsf{a}} S_i \xrightarrow{\mu_i} \cdots \xrightarrow{\mu_{j-2}} S_{j-1} \xrightarrow{\mathsf{A}\,:\,\mathsf{do}_s\,\mathsf{b}} S_j \xrightarrow{\mu_j} \cdots$$

where $\mu_h \neq \mathsf{A} : \mathsf{do}_s -$ for all $h \in [i, j-2]$. Then, either $c_j = 0$ or $\mathsf{A} \smile_s S_j$.

$$\mathsf{a};\, c \oplus c' \xrightarrow{\mathsf{a}}_\sharp c \qquad \mathsf{a}.\, c + c' \xrightarrow{\mathsf{a}}_\sharp c \qquad \mathit{ready}\ \mathsf{a}.\, c \xrightarrow{\mathsf{a}}_\sharp c \qquad \mathsf{a};\, c \oplus c' \xrightarrow{\mathsf{a}}_\sharp E \qquad \mathsf{a}.\, c + c' \xrightarrow{\mathsf{a}}_\sharp E$$

$$\bigoplus \mathsf{a}_i;\, c_i \xrightarrow{0}_\sharp 0 \qquad \sum \mathsf{a}_i.\, c_i \xrightarrow{0}_\sharp 0 \qquad \sum \mathsf{a}_i.\, c_i \xrightarrow{ctx}_\sharp \mathit{ready}\ \mathsf{a}_n.\, c_n \qquad \mathsf{a};\, c \xrightarrow{ctx}_\sharp \mathit{ready}\ \mathsf{a}.\, c \qquad c \xrightarrow{ctx}_\sharp c$$

$$\pi.P + Q \mid R \xrightarrow{\pi}_\sharp \begin{cases} \mathit{open}(\downarrow_x \mathsf{A}\ \mathit{says}\ c \mid P \mid R) & \text{if } \pi = \mathsf{tell}_\mathsf{A} \downarrow_x c \qquad P \xrightarrow{ctx}_\sharp \downarrow_x \mathsf{B}\ \mathit{says}\ c \mid P \ \text{ if } \mathsf{B} \neq \mathsf{A} \\ \mathit{open}(P \mid R)\sigma & \text{otherwise} \qquad\qquad\qquad\quad P \xrightarrow{ctx}_\sharp P\sigma \end{cases}$$

$\mathit{open}(P) = P'$ where $P \equiv (\vec{u}_i)P'$ and no delimitation of P' can be brought to the top level

Fig. 4. Abstract LTSs for contracts and processes ($\sigma : \mathcal{V} \to \mathcal{N}$, name A in $\to_\sharp^\mathsf{A}$ is omitted).

The following theorem states the undecidability of honesty. Our proof reduces the halting problem to checking dishonesty.

Theorem 4. *The problem of deciding whether a participant* $\mathsf{A}[P]$ *is dishonest is recursively enumerable, but not recursive.*

5 A criterion for honesty

In this section we devise a sufficient criterion for honesty. Actually, checking honesty is a challenging task: indeed, by Th. 4, it is not even decidable. We will then provide a semantics of contracts and processes, that focusses on the actions performed by a single participant A, while abstracting from those made by the context. Note that our abstract semantics assumes processes without top-level delimitations, in accordance with Def. 10 which lifts such delimitations outside participants. Further, we sometimes perform this lifting explicitly through the $\mathit{open}(-)$ operator.

Definition 14. *For all participant names* A*, the abstract LTSs* $\twoheadrightarrow_\sharp$ *and* $\to_\sharp^\mathsf{A}$ *on contracts and on processes, respectively, are defined by the rules in Fig. 4, where* $\sigma : \mathcal{V} \to \mathcal{N}$*.*

The intuition behind the abstract rules is provided by Lemma 6 and Lemma 7 below, which establish the soundness of the abstractions.

Lemma 6. *For all bilateral contracts* $\gamma = \mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says}\ d$*:*

1. $\gamma \xrightarrow{\mathsf{A}\ \mathit{says}\ \mathsf{a}} \mathsf{A}\ \mathit{says}\ c' \mid \mathsf{B}\ \mathit{says}\ d' \implies c \xrightarrow{\mathsf{a}}_\sharp c' \wedge (d \xrightarrow{ctx}_\sharp d' \vee d \xrightarrow{0}_\sharp d')$
2. $\gamma \xrightarrow{\mathsf{A}\ \mathit{says}\ \mathsf{a}} \mathsf{A}\ \mathit{says}\ c' \mid \mathsf{B}\ \mathit{says}\ d' \wedge c \bowtie d \implies c \xrightarrow{\mathsf{a}}_\sharp c' \wedge d \xrightarrow{ctx}_\sharp d'$

Intuitively, a move of γ is caused by an action performed by one of its components c and d. If c moves, the $\xrightarrow{\mathsf{a}}_\sharp$ rules account for its continuation. This might make d commit to one of the branches of a sum, as shown in the $\xrightarrow{ctx}_\sharp$ rules. Further, c can perform an action not supported by d, by using a [*FAIL] rule: accordingly, $\xrightarrow{0}_\sharp$ transforms d into 0. The compliance between c and d ensures the absence of such failure moves.

Lemma 7. *For each (finite or infinite) stable $\to$-trace $(S_i)_i$, with $S_i = (\vec{u}_i)(\mathsf{A}[Q_i] \mid S'_i)$, there exists a $\to_\sharp$-trace $Q_0 \xrightarrow{\mu_0}_\sharp Q_1 \xrightarrow{\mu_1}_\sharp Q_2 \xrightarrow{\mu_2}_\sharp \cdots$ where $\mu_i = \pi$ if $S_i \xrightarrow{\mathsf{A}:\pi} S_{i+1}$, and $\mu_i = ctx$ otherwise. Moreover, if $(S_i)_i$ is fair, then $(Q_i)_i$ is $\{\tau, \mathsf{tell}\}$-fair.*

In the above lemma, each step of the whole system might be due to either the process Q_i or its context. If Q_i fires a prefix π, then it changes according to the $\xrightarrow{\pi}_\sharp$ rule in Fig. 4. In particular, that accounts for $\mathsf{tell}_\mathsf{A}\, -$ adding further latent contracts to Q_i, as well as fuse possibly instantiating variables. Newly exposed delimitations are removed using $open(-)$: indeed, they already appear in $\vec{u}_i$, since the trace is stable.

We now define when a process P "$\sharp$-realizes" a contract c in a session s (written $P \models_s c$), without making any assumptions about its context. Intuitively, $P \models_s c$ holds when (1) P eventually enables the do_s actions mandated by c, and (2) in the abstract LTS $\to_\sharp$, the continuation of P after firing some do_s must realize the continuation of c (under $\twoheadrightarrow_\sharp$). Note that P is not required to actually perform the relevant do_s, because the context might prevent P from doing so. For instance, in the system $\mathsf{A}[P] \mid s[\mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says}\ \mathsf{ready}\ \mathsf{a}.d]$ the process P can not fire any do_s.

Definition 15. *Given a session s and a participant A, we define the relation $\models_s^\mathsf{A}$ ("$\sharp$-realizes") between processes and contracts as the largest relation such that, whenever $P_0 \models_s^\mathsf{A} c$, then for each $\{\tau, \mathsf{tell}\}$-fair $\to_\sharp^\mathsf{A}$-trace $(P_i)_i$ without labels $\mathsf{do}_s\, -$, we have:*

1. $\exists k.\ \forall i \geq k.\ c$ *unblocks* $RD_s(P_i)$
2. $\forall i, \mathsf{a}, P', c'.\ \left(P_i \xrightarrow{\mathsf{do}_s\,\mathsf{a}}_\sharp P' \wedge c \xrightarrow{\mathsf{a}}\!\!\!\!\twoheadrightarrow_\sharp c' \implies P' \models_s^\mathsf{A} c'\right)$

Example 10. Recall the online store A from Ex. 4. We show that $X\{s/x\} \models_s c_\mathsf{A}$. First note that transitions in $\{\tau, \mathsf{tell}\}$-fair $\to_\sharp$-traces without do_s from $X\{s/x\}$ can only be labelled with *ctx*. Thus, each P_i on such traces has the form $X\{s/x\} \mid K_i$, for some K_i. We have $RD_s(P_i) = RD_s(X\{s/x\}) = \{\mathsf{addToCart}, \mathsf{creditCard}\}$. Also, c_A *unblocks* $RD_s(X\{s/x\})$ hence condition (1) of Def. 15 holds. For condition (2), if $c_\mathsf{A} \xrightarrow{\mathsf{creditCard}}\!\!\!\!\twoheadrightarrow_\sharp c' = \overline{\mathsf{accept}} \oplus \overline{\mathsf{reject}}$ and $P_i \xrightarrow{\mathsf{do}_s\,\mathsf{creditCard}}_\sharp P' = \tau.\mathsf{do}_s\,\overline{\mathsf{accept}} + \tau.\mathsf{do}_s\,\overline{\mathsf{reject}} \mid K_i$ then $P' \models_s c'$. Actually, all processes on a $\{\tau, \mathsf{tell}\}$-fair $\to_\sharp$-traces without do_s from P' have either the form $\mathsf{do}_s\,\overline{\mathsf{accept}} \mid K$ or the form $\mathsf{do}_s\,\overline{\mathsf{reject}} \mid K$. For the recursive case, $c_\mathsf{A} \xrightarrow{\mathsf{addToCart}}\!\!\!\!\twoheadrightarrow_\sharp c_\mathsf{A}$ and $P_i \xrightarrow{\mathsf{do}_s\,\mathsf{addToCart}}_\sharp X\{s/x\}$, hence $X\{s/x\} \models_s c_\mathsf{A}$ by coinduction. Note that the case $c_\mathsf{A} \xrightarrow{\mathsf{e}}\!\!\!\!\twoheadrightarrow_\sharp$ did not apply, because P_i cannot take $\to_\sharp$-transitions labelled $\mathsf{do}_s\,\mathsf{e}$.

Theorem 5 below establishes an invariant of system transitions. If a participant $\mathsf{A}[Q_0]$ $\sharp$-realizes a stipulated contract c_0, then in each evolution of the system the descendant of $\mathsf{A}[Q_0]$ still $\sharp$-realizes the related descendant of c_0. The theorem only assumes that c_0 is in a session with a compliant contract, as it is the case after firing a fuse.

Theorem 5. *Let $(S_i)_i$ be a stable $\to$-trace with $S_i = (\vec{u}_i)(\mathsf{A}[Q_i] \mid s[\mathsf{A}\ \mathit{says}\ c_i \mid \mathsf{B}\ \mathit{says}\ d_i] \mid S'_i)$ for all i. If $c_0 \bowtie d_0$ and $Q_0 \models_s^\mathsf{A} c_0$, then $Q_i \models_s^\mathsf{A} c_i$ for all i.*

We now define when a participant is $\sharp$-*honest*. Intuitively, we classify as such a participant $\mathsf{A}[P]$ when, for all prefixes $\mathsf{tell}\downarrow_x c$ contained in P, the continuation Q of the prefix $\sharp$-realizes c. We also require that the session variable x cannot be used by any process in parallel with Q, because such processes could potentially compromise the ability of Q to realise c (see Ex. 11).

Definition 16 ($\sharp$-honest participant). *A participant* $\mathsf{A}[P]$ *is* $\sharp$-honest *iff P does not contain* $\downarrow_y \mathsf{A}$ *says c, and for all linear contexts* $\mathcal{C}(\bullet)$*, x, c, Q, R, and s fresh in P*

$$P = \mathcal{C}(\mathsf{tell}\downarrow_x c.Q + R) \implies open(Q\{s/x\}) \models^{\mathsf{A}}_s c \ \wedge \ \mathcal{C} \text{ is } x\text{-safe}$$

where $\mathcal{C}(\bullet)$ *is x-safe iff* $\exists \mathcal{C}'.\ \mathcal{C}(\bullet) = \mathcal{C}'((x)\bullet)$ *or* $\mathcal{C}$ *is free from* $\mathsf{do}_x -$.

Example 11. Substitute $Q = \mathsf{fuse}_x.\mathsf{do}_x\,\mathsf{creditCard}$ for fuse_x in the process P_A from Ex. 4. Then $\mathsf{A}[P_A]$ is not honest, because A cannot complete her contract if the do_x within Q is performed. However, the modified $\mathsf{A}[P_A]$ violates x-safety, hence it is not $\sharp$-honest.

The following lemma relates $\sharp$-honesty with the abstract semantics of processes. If a $\sharp$-honest process P abstractly fires a $\mathsf{tell}\downarrow_x c$, then the continuation of P realises c (item 1). Also, $\sharp$-honesty is preserved under abstract transitions (item 2).

Lemma 8. *For all* $\sharp$*-honest participants* $\mathsf{A}[P]$*, such that* $P = open(P)$*:*

1. *if* $P \xrightarrow{\mathsf{tell}_\mathsf{B}\downarrow_x c}_\sharp P'$*, then* $P'\{s/x\} \models^{\mathsf{A}}_s c$*, for all s fresh in P.*
2. *if* $P \rightarrow_\sharp P'$*, then* $\mathsf{A}[P']$ *is* $\sharp$*-honest.*

Our main result states that $\sharp$-honesty suffices to ensure honesty. Note that while honesty, by Def. 11, considers all the (infinite) possible contexts, $\sharp$-honesty does not. Hence, while verifying honesty can be unfeasible in the general case, it can also be ensured by establishing $\sharp$-honesty, which is more amenable to verification. For instance, for finite control processes [11] it is possible to decide $\sharp$-honesty e.g. through model-checking. In fact, in these processes parallel composition cannot appear under recursion, hence their behaviour can be represented with finitely many states.

Theorem 6. *All* $\sharp$*-honest participants are honest.*

Noteworthily, by Theorem 6 we can establish that all the participants named A in Examples 4, 5, and 6 are honest. This is obtained by reasoning as in Example 10. Instead, participant A in Example 7 is honest but not $\sharp$-honest.

6 Related Work and Conclusions

We have developed a formal model for reasoning about contract-oriented systems. Our approach departs from the common principle that contracts are always respected after they are agreed upon. We represent instead the more realistic situation where promises are not always kept. The process calculus CO_2 [2] allows participants to advertise contracts, to establish sessions with other participants with compliant contracts, and to fulfill them (or choose not to). Remarkably, instead of defining an ad-hoc contract model, we have embedded the contract theory of [10] within CO_2. To do that, we have slightly adapted the contracts of [10] in order to define culpability, and we have specialized CO_2 accordingly at the system-level. The main technical contribution of this paper is a criterion for deciding when a participant is honest, i.e., always respects the advertised contracts in all possible contexts. This is not a trivial task, especially when multiple

sessions are needed for realizing a contract (see e.g. Ex. 5 and 6) or when participants want to inspect the state of a contract to decide how to proceed next (see e.g. Ex. 7).

At the best of our knowledge, this is the first paper that addresses the problem of establishing when a participant is honest in a contract-based system populated by dishonest participants. Several papers investigated the use of contracts in concurrent systems; however, they typically focus on coupling processes which statically guarantee conformance to their contracts. This is achieved e.g. by typing [5, 9, 10], by contract-based process synthesis [7], or by approaches based on behavioural preorders [6]. As future work, it may also be interesting to study weaker notions of honesty, e.g., by requiring participants to respect contracts in *honest* contexts, only.

The process calculus CO_2 has been introduced in [2] as a generic framework for relating different contract models; the variant in this paper has been obtained by instantiating it with the contracts of [10]. Some primitives, e.g. multiparty fuse, have been consequently simplified. In [2], a participant A is honest when A becomes not culpable from a certain execution step; here, we only require that, whenever A is culpable, then she can exculpate herself by performing some actions. This change reflects the fact that bilateral contracts à la [10] can describe endless interactions.

The notion of compliance in [10] is asymmetric. Namely, if c is the client contract and d is the server contract, then c and d are compliant if c always reaches a success state or engages d in an endless interaction. In our model instead compliance is symmetric: the server contract, too, has to agree on when a state is successful. The LTS semantics of unilateral contracts in [10] yields identical synchronization trees for internal and external choice; to differentiate them, one has to consider their ready sets. We instead give semantics to *bilateral* contracts, and distinguish between choices at the LTS level. Note that we do not allow for unguarded sums, unlike [10]. Were these be allowed, we would have to deal e.g. with a participant A with a contract of the form $\mathsf{a}; c_0 \oplus (\mathsf{b}.c_1 + \mathsf{c}.c_2)$. According to our intuition A should be culpable, because of the internal choice. If A legitimately chooses not to perform a, to exculpate herself she would have to wait for the other participant to choose (internally) between b and c. Therefore, A can exculpate herself only if the other participant permits her to. By contrast, by restricting to guarded sums our theory enjoys the nice feature that a culpable participant can always exculpate herself by performing some actions, which pass the buck to the other participant (Theorems 1 and 3).

Design-by-contract is transferred in [5] to distributed interactions modelled as (multiparty) asserted global types [14]. The projection of asserted global types on local ones allows for the automatic generation of monitors whereby incoming messages are checked against the local contract. Such monitors have a "local" view of the computation, i.e. they can detect a violation but cannot, in general, single out the culpable component. In fact, a monitor cannot know if an expected message is not delivered because the partner is violating his contract, or because he is blocked on interactions with other participants. Conversely, in our approach we compose participants in a "bottom-up" fashion: a participant declares its contract independently of the others and then advertises it; the fuse primitive tries then to harmonise contracts by searching for a suitable agreement. Our notion of *honesty* singles out culpable components during the computation. An interesting problem would be to investigate how our notion of culpability could

be attained within the approach in [5]. In fact, this seems to be a non trivial problem, even if forbidding communication channels shared among more than two participants.

Contracts are rendered in [8, 7] as soft constraints (values in a c-semiring) that allow for different levels of agreement between contracts. When matching a client with a service, the constraints are composed. This restricts the possible interactions to those acceptable (if any) to both parties. A technique is proposed in [7] for compiling clients and services so that, after matching, both actually behave according to the mutually acceptable interactions, and reach success without getting stuck. Our framework is focused instead on blaming participants, and on checking when a participant is honest, i.e. always able to avoid blame in all possible contexts. The use of soft constraints in a context where participants can be dishonest seems viable, e.g. by instantiating the abstract contract model of CO_2 with the contracts in [7]. A challenging task would be that of defining culpability in such setting.

Acknowledgments. 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. A. Artikis, M. J. Sergot, and J. V. Pitt. Specifying norm-governed computational societies. *ACM Trans. Comput. Log.*, 10(1), 2009.
2. M. Bartoletti, E. Tuosto, and R. Zunino. Contracts in distributed systems. In *ICE*, 2011.
3. M. Bartoletti, E. Tuosto, and R. Zunino. On the realizability of contracts in dishonest systems. *CoRR*, abs/1201.6188, 2012.
4. M. Bartoletti and R. Zunino. A calculus of contracting processes. In *LICS*, 2010.
5. L. Bocchi, K. Honda, E. Tuosto, and N. Yoshida. A theory of design-by-contract for distributed multiparty interactions. In *CONCUR*, 2010.
6. M. Bravetti and G. Zavattaro. Towards a unifying theory for choreography conformance and contract compliance. In *Software Composition*, 2007.
7. M. Buscemi, M. Coppo, M. Dezani-Ciancaglini, and U. Montanari. Constraints for service contracts. In *TGC*, 2011. To appear.
8. M. G. Buscemi and U. Montanari. CC-Pi: A constraint-based language for specifying service level agreements. In *ESOP*, 2007.
9. S. Carpineti and C. Laneve. A basic contract language for web services. In *ESOP*, 2006.
10. G. Castagna, N. Gesbert, and L. Padovani. A theory of contracts for web services. *ACM Transactions on Programming Languages and Systems*, 31(5), 2009.
11. M. Dam. On the Decidability of Process Equivalences for the π-calculus. *Theoretical Computer Science*, 183(2):215–228, 1997.
12. P. Degano and C. Priami. Enhanced operational semantics. *ACM Comput. Surv.*, 33(2):135–176, 2001.
13. E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*. North-Holland Pub. Co./MIT Press, 1990.
14. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL*, 2008.
15. C. Prisacariu and G. Schneider. A formal language for electronic contracts. In *FMOODS*, 2007.