On the decidability of honesty and of its variants

Massimo Bartoletti[1] and Roberto Zunino[2]

[1] Università degli Studi di Cagliari, Italy
[2] Università degli Studi di Trento, Italy

Abstract. We address the problem of designing distributed applications which require the interaction of loosely-coupled and mutually distrusting services. In this setting, services can use *contracts* to protect themselves from unsafe interactions with the environment: when their partner in an interaction does not respect its contract, it can be blamed (and punished) by the service infrastructure. We extend a core calculus for services, by using a semantic model of contracts which subsumes various kinds of behavioural types. In this formal framework, we study some notions of *honesty* for services, which measure their ability to respect contracts, under different assumptions about the environment. In particular, we find conditions under which these notions are (un)decidable.

1 Introduction

Service-Oriented Computing (SOC) fosters a programming paradigm where distributed applications can be constructed by discovering, integrating and using basic services [18]. These services may be provided by different organisations, possibly in competition (when not in conflict) among each other. Further, services can appear and disappear from the network, and they can dynamically discover and invoke other services in order to exploit their functionality, or to adapt to changing needs and conditions. Therefore, programmers of distributed applications have to cope with such security, dynamicity and openness issues in order to make their applications trustworthy.

A possible way to address these issues is to use *contracts*. When a service needs to use some external (possibly untrusted) service, it advertises to a SOC middleware a contract which specifies the offered/required interaction protocol. The middleware establishes sessions between services with compliant contracts, and it monitors the communication along these sessions to detect contract violations. These violations may happen either unintentionally, because of errors in the service specification, or because of malicious behaviour.

When the SOC middleware detects contract violations, it sanctions the responsible services. For instance, the middleware in [3] decreases the *reputation* of the culprit, in order to marginalise services with low reputation during the selection phase. Therefore, a new form of attacks arises: malicious users can try to make some service sanctioned by exploiting possible discrepancies between the promised and the actual behaviour of that service. A crucial problem is then how to avoid such attacks when deploying a service.

However, designing an *honest* service which always respects its contracts requires one to fulfil its obligations also in adversarial contexts which play against. We illustrate below that, even for a fairly simple application composed by only three services, this is not an easy task.

An example. Consider an online store taking orders from buyers. The store sells two items: item A, which is always available and costs €1, and item B, which costs €1 when in stock, and €3 otherwise. In the latter case, the store orders item B from an external distributor, which makes the store pay €2 per item.

The store advertises the following contract to potential buyers:

1. let the buyer choose between item A and item B;
2. if the buyer chooses item A, then receive €1, and then ship the item to him;
3. if the buyer chooses item B, offer a quotation to the buyer (€1 or €3);
4. if the quotation is €1, then receive the payment and ship;
5. if the quotation is €3, ask the buyer to pay or cancel the order;
6. if the buyer pays €3, then either ship the item to him, or refund €3.

We can formalise such contract in several process algebras. For instance, we can use the following session type [20] (without channel passing):

$$\begin{aligned} T_{\mathsf{B}} &= \mathtt{buyA}.\mathtt{pay1E}.\overline{\mathtt{shipA}} \;\&\; \\ &\quad\; \mathtt{buyB}.(\overline{\mathtt{quote1E}}.\mathtt{pay1E}.\overline{\mathtt{shipB}} \oplus \overline{\mathtt{quote3E}}.T'_{\mathsf{B}} \oplus \overline{\mathtt{abort}}) \\ T'_{\mathsf{B}} &= \mathtt{pay3E}.(\overline{\mathtt{shipB}} \oplus \overline{\mathtt{refund}}) \;\&\; \mathtt{quit} \end{aligned}$$

where e.g., buyA represents a label in a *branching* construct (i.e., receiving an order for item A from the buyer), while $\overline{\mathtt{quote1E}}$ represents a label in a *selection* construct (i.e., sending an €1 quotation to the buyer). The operator $\oplus$ separates branches in an *internal choice*, while & separates branches in an *external choice*.

The protocol between the store and the distributor is the following:

$$T_{\mathsf{D}} \;=\; \overline{\mathtt{buyB}}.(\overline{\mathtt{pay2E}}.\mathtt{shipB} \oplus \overline{\mathtt{quit}})$$

Note that the contracts above do not specify the *actual* behaviour of the store, but only the behaviour it promises towards the buyer and the distributor. A possible informal description of the actual behaviour of the store is the following:

1. advertise the contract T_{B};
2. when T_{B} is stipulated, let the buyer choose item A (buyA) or B (buyB);
3. if the buyer chooses A, get the payment (pay1E), and ship the item (shipA);
4. otherwise, if the buyer chooses B, check if the item is in stock;
5. if item B is in stock, provide the buyer the quotation of €1 (quote1E), receive the payment (pay1E), and ship the item (shipB);
6. otherwise, if item B is not in stock, advertise the contract T_{D};
7. when T_{D} is stipulated, pre-order item B from the distributor (buyB);
8. send a €3 quotation to the buyer (quote3E) and wait for the buyer's reply;
9. if the buyer pays €3 (pay3E), then pay the distributor (pay2E), receive the item from the distributor (shipB), and ship it to the buyer (shipB).

The store service terminates correctly whenever two conditions hold: the buyer is honest, and at step 7 the middleware selects an honest distributor. Such assumptions are necessary. For instance, in their absence we have that:

(a) if the buyer is dishonest, and he does not send €3 at step 9, then the store does not fulfil its obligation with the distributor, who is expecting a payment or a cancellation;
(b) if the middleware finds no distributor with a contract compliant with T_{D}, then the store is stuck at line 7, so it does not fulfil its obligation with the buyer, who is expecting a quotation or an abort;
(c) if the distributor is dishonest, and it does not ship the item at line 9, then the store does not fulfil its obligation with the buyer, who is expecting to receive the item or a refund;
(d) if the buyer chooses `quit` at line 8, the store forgets to handle it; so, it will not fulfil the contract with the distributor, who is expecting `pay2E` or `quit`.

Therefore, we would classify the store process above as *dishonest*. In practice, this implies that a concrete implementation of such store could be easily attacked. For instance, an attacker could simply order item B (when not in stock), but always cancel the transaction. The middleware will detect that the store is violating the contract with the distributor, and consequently it will sanction the store. Concretely, in the middleware of [3] the attacker will manage to never be sanctioned, and to arbitrarily decrease the store reputation, so preventing the store from being able to establish new sessions with buyers.

The example above shows that writing honest processes is an error-prone task: this is because one has to foresee all the possible points of failure of each partner. We handle all such points in Example 6, where we show a provably honest store process.

Specifying contract-oriented services. To formalise and study honesty, we first fix the formal setting, which consists of two basic ingredients:

– a model of *contracts*, which specify the *promised* behaviour of a service.
– a model of *processes*, which specify the *actual* behaviour. Such behaviour involves e.g. checking compliance between contracts, making a contract evolve upon actions, *etc.*, and so it also depends on the contract model.

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 process calculus may have several different behavioural equivalences/preorders, it is also reasonable that, even in a specific contract/process model, many relevant notions of honesty exist.

In this paper we focus on a quite general model of contracts: arbitrary LTSs. In particular, states denote contracts, and labels represent internal actions and synchronisations between two services at the endpoints of a session (Section 2).

We interpret compliance between two contracts as the absence of deadlock in their parallel execution, similarly to [1,2,13]. This model allows for a syntax-independent treatment of contracts (like e.g. session types, see Section 2.2).

To formalise processes, we build upon CO_2 [10]: this is a minimalistic calculus with primitives for advertising contracts, opening sessions, and doing contractual actions. In Section 3 we extend the calculus of [10] by modifying the synchronisation primitive to use arbitrary LTSs as contracts, and the advertisement primitive to increase its expressiveness.

Contributions. The main contribution of the paper is the study of some notions of honesty, their properties, and their decidability. In particular:

1. We show that two different notions of honesty coincide (Theorem 1). The first one (originally introduced in [8]) says that a process is honest when, in *all* possible contexts, whenever it has some contractual obligations, it can interact with the context and eventually fulfil said obligations. The second notion is a variant (introduced here), which requires a process to be able (in all possible contexts) to fulfil its obligations on its own, *without* interacting with the context. This result simplifies the design of static analyses for honesty, since it allows for abstracting the moves of the context when one has to decide whether a process is fulfilling its obligations.
2. We prove that systems of honest processes are deadlock-free (Theorem 6).
3. We introduce a weaker notion of honesty, where a process is required to behave honestly only when its partners are honest (Definition 15). For instance, weak honesty ensures the absence of attacks such as items b and d in the store example, but it does not rule out attacks such as items a and c. Unlike systems of honest processes, systems of *weakly* honest processes may get stuck, because of circular dependencies between sessions (see Example 8).
4. We show that if a process using session types as contracts is honest in all contexts *which use session types as contracts*, then it is honest in all *arbitrary* contexts (Theorem 5). This property has a practical impact: if some static analyses tailored on session types (like e.g., that in [7]) determines that a process is honest, then we can safely use such process in any context — also in those which use a different contract model.
5. We study decidability of honesty and weak honesty. First, for any given Turing Machine, we show in Theorem 7 how to craft a CO_2 process which simulates it. We then prove that this process is honest (according to *any* of the notions presented above) if and only if said Turing Machine is not halting. From this we establish the undecidability of all the above-mentioned notions of honesty, in all possible models of contracts which include session types. Overall, this generalises a result in [10], which establishes the undecidability of (strong) honesty in an instance of CO_2 using τ-less CCS contracts [13].
6. We find a syntactic restriction of CO_2 and a constraint on contracts under which honesty is decidable (Theorem 8).
7. We find a class of contracts for which *dis*honesty of (unrestricted) CO_2 processes is recursively enumerable (Theorem 9).

2 Contracts

We now provide a semantic setting for contracts. In Section 2.1 we model contracts as states of a Labelled Transition System (LTS) with two kinds of labels: *internal actions*, which represent actions performed by one participant, and *synchronisation actions*, which model interactions between participants. As an example, in Section 2.2 we show that session types can be interpreted in this setting. In Section 2.3 we provide contracts with a notion of *compliance*, which formalises correct interactions between services which respect their contracts.

2.1 A model of contracts

Assume a set of *participants* (ranged over by $\mathsf{A}, \mathsf{B}, \ldots$), a recursive set $\mathtt{L}$ (ranged over by $\mathtt{a}, \mathtt{b}, \ldots$) with an involution $\overline{\cdot}$, and a recursive set Λ^τ (ranged over by $\tau, \tau_a, \tau_i, \ldots$). We call $\Lambda^{\mathtt{a}} = \mathtt{L} \cup \overline{\mathtt{L}}$ the set of *synchronisation actions*, and Λ^τ the set of *internal actions*. We then define the set Λ of *actions* as the disjoint union of $\Lambda^{\mathtt{a}}$ and Λ^τ, and we let $\alpha, \beta, \ldots$ range over Λ.

We develop our theory within the LTS $(\mathbb{U}, \Lambda, \rightarrow)$, where:

- $\mathbb{U}$ is a set (ranged over by $c, d, \ldots$), called the universe of *contracts*;
- $\rightarrow \subseteq \mathbb{U} \times \Lambda \times \mathbb{U}$ is a transition relation between contracts, with labels in Λ.

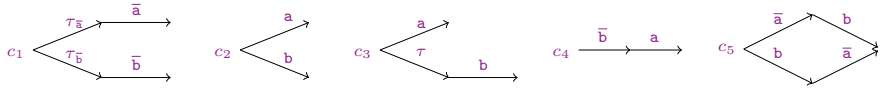
We denote with $\mathbb{U}_{fin}$ the set of *finite-state* contracts, i.e. for all $c \in \mathbb{U}_{fin}$, the contracts reachable from c with any finite sequence of transitions is finite. We denote with $\mathbf{0}$ a contract with no outgoing transitions, and we interpret it as a *success* state. We write: $\mathcal{R}^*$ for the reflexive and transitive closure of a relation $\mathcal{R}$, and $c \xrightarrow{\alpha} c'$ when $(c, \alpha, c') \in \rightarrow$. Furthermore, sometimes we express contracts through the usual CCS operators [24]: for instance, we can write the contract c_1 in Figure 1 as the term $\tau_{\overline{\mathtt{a}}}.\,\overline{\mathtt{a}} + \tau_{\overline{\mathtt{b}}}.\,\overline{\mathtt{b}}$.

While a contract describes the intended behaviour of *one* of the two participants involved in a session, the behaviour of two interacting participants A and B is modelled by the composition of two contracts, denoted by $\mathsf{A} : c \parallel \mathsf{B} : d$. We specify in Definition 1 an operational semantics of these *contract configurations*: internal actions can always be fired, while synchronisation actions require both participants to enable two complementary actions. Note that the label of a synchronisation is *not* an internal action (unlike e.g., in CCS [24]); this is because in the semantics of CO_2 we need to inspect such label in order to make two processes synchronise (see rule [DoCom] in Figure 3).

Definition 1 (Semantics of contract configurations). *We define the transition relation $\twoheadrightarrow$ between contract configurations (ranged over by $\gamma, \gamma', \ldots$) as the least relation closed under the following rules:*

$$\frac{c \xrightarrow{\tau} c'}{\mathsf{A} : c \parallel \mathsf{B} : d \xrightarrow{\{\mathsf{A}\}:\tau} \mathsf{A} : c' \parallel \mathsf{B} : d} \qquad \frac{d \xrightarrow{\tau} d'}{\mathsf{A} : c \parallel \mathsf{B} : d \xrightarrow{\{\mathsf{B}\}:\tau} \mathsf{A} : c \parallel \mathsf{B} : d'}$$

$$\frac{c \xrightarrow{\mathtt{a}} c' \qquad d \xrightarrow{\overline{\mathtt{a}}} d'}{\mathsf{A} : c \parallel \mathsf{B} : d \xrightarrow{\{\mathsf{A},\mathsf{B}\}:\mathtt{a}} \mathsf{A} : c' \parallel \mathsf{B} : d'}$$



(1) (2) (3) (4) (5)

Fig. 1: Some simple contracts.

2.2 Session types as contracts

Session types [19,20] are formal specifications of communication protocols between the participants at the endpoints of a session. We give in Definition 2 a version of session types without channel passing, similarly to [1].

Definition 2 (Session types). Session types *are terms of the grammar:*

$$T \quad ::= \quad \&_{i\in I}\mathtt{a}_i.T_i \;\mid\; \bigoplus_{i\in I}\overline{\mathtt{a}_i}.T_i \;\mid\; \mathrm{rec}_X\, T \;\mid\; X$$

where (i) *the set* I *is finite,* (ii) *all the actions in external (resp. internal) choices are pairwise distinct and in* $\mathtt{L}$ *(resp. in* $\overline{\mathtt{L}}$*), and* (iii) *recursion is prefix-guarded.*

A session type is a term of a process algebra featuring a *selection* construct (i.e., an internal choice among a set of branches, each one performing some output), and a *branching* construct (i.e., an external choice among a set of inputs offered to the environment). We write $\mathbf{0}$ for the empty (internal/external) choice, and we omit trailing occurrences of $\mathbf{0}$. We adopt the equi-recursive approach, by considering terms up-to unfolding of recursion.

We can interpret session types as contracts, by giving them a semantics in terms of the LTS defined in Section 2.1.

Definition 3. *We denote with* ST *the set of contracts of the form* T *or* $[\overline{\mathtt{a}}]\,T$*, with* T *closed, and transitions relation given by the following rules:*

$$\&_{i\in I}\,\mathtt{a}_i.T_i \xrightarrow{\mathtt{a}_k} T_k \;\; (k\in I) \qquad \bigoplus_{i\in I}\overline{\mathtt{a}_i}.T_i \xrightarrow{\tau_i} [\overline{\mathtt{a}_k}]\,T_k \;\; (k\in I) \qquad [\overline{\mathtt{a}}]\,T \xrightarrow{\overline{\mathtt{a}}} T$$

An external choice can always fire one of its prefixes. An internal choice $\bigoplus_{i\in I}\overline{\mathtt{a}_i}.T_i$ must first commit to one of the branches $\overline{\mathtt{a}_k}.T_k$, and this produces a *committed choice* $[\overline{\mathtt{a}_k}]\,T_k$, which can only fire $\overline{\mathtt{a}_k}$. As a consequence, a session type may have several outgoing transitions, but internal transitions cannot be mixed with synchronisation ones. There cannot be two internal transitions in a row, and after an internal transition, the target state will have exactly one reduct. Note that $\mathrm{ST} \subsetneq \mathbb{U}_{fin}$.

Example 1. The contract c_1 in Figure 1 represents the session type $\overline{\mathtt{a}} \oplus \overline{\mathtt{b}}$: since it is an internal choice, according to Definition 3 there is a commit on the chosen branch before actually firing the synchronisation action. The contract c_2 is in ST as well, as it represents an external choice $\mathtt{a} \,\&\, \mathtt{b}$. Instead, the last three contracts do *not* belong to ST: indeed, in c_3 an internal transition is mixed with an input one; in c_4 there is no internal transition before $\overline{\mathtt{b}}$; finally, in c_5 input and output transitions are mixed (note that c_5 represents an *asynchronous* output of $\overline{\mathtt{a}}$ followed by an input of $\mathtt{b}$, as in the asynchronous session types of [9]). □

2.3 Compliance

Among the various notions of compliance appeared in the literature [4], here we adopt *progress* (i.e. the absence of deadlock). In Definition 4 we say that c and d are compliant (in symbols, $c \bowtie d$) iff, when a reduct of $\mathsf{A} : c \parallel \mathsf{B} : d$ cannot take transitions, then both participants have reached success. A similar notion has been used in [13] (for τ-less CCS contracts) and in [1,2] (for session types).

Definition 4 (Compliance). *We write $c \bowtie d$ iff:*

$$\mathsf{A} : c \parallel \mathsf{B} : d \;\twoheadrightarrow^{*}\; \mathsf{A} : c' \parallel \mathsf{B} : d' \;\not\twoheadrightarrow \quad \textit{implies} \quad c' = \mathbf{0} \textit{ and } d' = \mathbf{0}$$

Example 2. Consider contracts in Figure 1. We have that $c_1 \bowtie c_2$ and $c_4 \bowtie c_5$, while all the other pairs of contracts are not compliant. □

3 Contract-oriented services

We now extend the process calculus CO_2 of [10], by parameterising it over an arbitrary set $\mathbb{C}$ of contracts. As a further extension, while in [10] one can advertise a single contract at a time, here we allow processes to advertise *sets* of contracts, which will be stipulated atomically (see Definition 6). This will allow us to enlarge the set of honest processes, with respect to those considered in [10].

3.1 Syntax

Let $\mathcal{V}$ and $\mathcal{N}$ 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 $\mathcal{V} \cup \mathcal{N}$, and $\boldsymbol{u}, \boldsymbol{v}, \ldots$ over $2^{\mathcal{V}\cup\mathcal{N}}$. A *latent contract* $\{\downarrow_x c\}$ represents a contract c which has not been stipulated yet; the variable x will be instantiated to a fresh session name upon stipulation. We also allow for *sets* of latent contracts $\{\downarrow_{u_1} c_1, \ldots, \downarrow_{u_k} c_k\}$, to be stipulated atomically. We let $C, C', \ldots$ range over sets of latent contracts, and we write C_{A} when the contracts are signed by A.

Definition 5 (CO_2 syntax). *The syntax of CO_2 is defined as follows:*

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

We also assume the following syntactic constraints on processes and systems:

1. *each occurrence of $\mathrm{X}(\boldsymbol{u})$ within a process is prefix-guarded;*
2. *each X has a unique defining equation $\mathrm{X}(\boldsymbol{u}) \triangleq P$, with $\mathrm{fv}(P) \subseteq \{\boldsymbol{u}\} \subseteq \mathcal{V}$;*
3. *in $(\boldsymbol{u})(\mathsf{A}[P] \mid \mathsf{B}[Q] \mid \cdots)$, it must be $\mathsf{A} \neq \mathsf{B}$;*
4. *in $(\boldsymbol{u})(s[\gamma] \mid t[\gamma'] \mid \cdots)$, it must be $s \neq t$;*

We denote with $\boldsymbol{\mathcal{P}}_{\mathbb{C}}$ the set of all processes with contracts in $\mathbb{C}$.

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

Fig. 2: Structural congruence (Z ranges over processes, systems, latent contracts)

Processes specify the actual 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 $(u)P$, or a constant $\mathsf{X}(\boldsymbol{u})$. 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 $1 \notin I$. If $\boldsymbol{u} = \{u_1, \ldots, u_k\}$, we write $(\boldsymbol{u})P$ for $(u_1) \cdots (u_k)P$. We omit trailing occurrences of $\mathbf{0}$.

Prefixes include the silent action τ, contract advertisement $\mathtt{tell}\, C$, and action execution $\mathtt{do}_u\, \alpha$, where the identifier u refers to the target session.

A system is composed of *agents* (i.e., named processes) $\mathsf{A}[P]$, *sessions* $s[\gamma]$, signed sets of latent contracts C_{A}, and delimited systems $(u)S$. Delimitation (u) binds session variables and names, both in processes and systems. Free variables and names are defined as usual, and their union is denoted by $\mathrm{fnv}(\_)$. A system/process is *closed* when it has no free variables. We denote with K a special participant name (playing the role of broker) not occurring in any system.

3.2 Semantics

We define the semantics of CO_2 as a reduction relation on systems (Figure 3). This uses a structural congruence, which is the smallest relation satisfying the equations in Figure 2. Such equations are mostly standard — we just note that $(u)\mathsf{A}[(v)P] \equiv (u)(v)\mathsf{A}[P]$ allows to move delimitations between CO_2 systems and processes. In order to define honesty in Section 4, we decorate transitions with labels, by writing $\xrightarrow{\mathcal{A} : \pi}$ for a reduction where participants $\mathcal{A}$ fire π.

Rule [Tau] fires a τ prefix. Rule [Tell] advertises a set of latent contracts C. Rule [Fuse] inspects latent contracts, which are stipulated when compliant pairs are found through the $\triangleright$ relation (see Definition 6 below). Upon stipulation, one or more new sessions among the stipulating parties are created. Rule [DoTau] allows a participant A to perform an internal action in the session s with contract configuration γ (which, accordingly, evolves to γ'). Rule [DoCom] allows two participants to synchronise in a session s. The last three rules are standard.

Definition 6. *The relation* $C^1{}_{\mathsf{A}_1} \mid \cdots \mid C^k{}_{\mathsf{A}_k} \triangleright^\sigma s_1[\gamma_1] \mid \cdots \mid s_n[\gamma_n]$ *holds iff:*

1. *for all* $i \in 1..k$, $C^i = \{\downarrow_{x_{i,1}} c_{i,1}, \ldots, \downarrow_{x_{i,m_i}} c_{i,m_i}\}$, *and the variables* $x_{i,j}$ *are pairwise distinct;*

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

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

$$\frac{{C^1}_{\mathsf{A}_1} \mid \cdots \mid {C^k}_{\mathsf{A}_k} \rhd^{\sigma} S' \qquad \operatorname{ran}\sigma \cap \operatorname{fn}(S) = \emptyset}{(\operatorname{dom}\sigma)({C^1}_{\mathsf{A}_1} \mid \cdots \mid {C^k}_{\mathsf{A}_k} \mid S) \xrightarrow{\{\mathsf{K}\} : \tau} (\operatorname{ran}\sigma)(S' \mid S\sigma)} \quad \text{[Fuse]}$$

$$\frac{\gamma \xrightarrow{\{\mathsf{A}\}:\tau_a\!\!\!\!\rightarrow\!\!\!\!\rightarrow} \gamma'}{\mathsf{A}[\mathtt{do}_s\, \tau_a.\, P + P' \mid Q] \mid s[\gamma] \xrightarrow{\{\mathsf{A}\} : \mathtt{do}_s\, \tau_a} \mathsf{A}[P \mid Q] \mid s[\gamma']} \quad \text{[DoTau]}$$

$$\frac{\gamma \xrightarrow{\{\mathsf{A},\mathsf{B}\}:\mathsf{a}\!\!\!\!\rightarrow\!\!\!\!\rightarrow} \gamma'}{\begin{array}{l}\mathsf{A}[\mathtt{do}_s\, \mathsf{a}.\, P + P' \mid P''] \mid \mathsf{B}[\mathtt{do}_s\, \overline{\mathsf{a}}.\, Q + Q' \mid Q''] \mid s[\gamma] \\ \mathsf{A}[P \mid P''] \qquad\qquad\quad \mid \mathsf{B}[Q \mid Q''] \qquad\quad \mid s[\gamma']\end{array} \xrightarrow{\{\mathsf{A},\mathsf{B}\} : \mathtt{do}_s\, \mathsf{a}} \quad \text{[DoCom]}$$

$$\frac{\mathrm{X}(\boldsymbol{u}) \triangleq P \qquad \mathsf{A}[P\{^{v}/_{u}\} \mid Q] \mid S \xrightarrow{\mathcal{A} : \pi} S'}{\mathsf{A}[\mathrm{X}(\boldsymbol{v}) \mid Q] \mid S \xrightarrow{\mathcal{A} : \pi} S'} \quad \text{[Def]}$$

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

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

Fig. 3: Reduction semantics of CO_2.

2. *for all* $i \in 1..k$, *let* $D_i = \{(\mathsf{A}_i, x_{i,h}, c_{i,h}) \mid h \in 1..m_i\}$. *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* $\gamma_j = \mathsf{A}_j : c_j \parallel \mathsf{B}_j : d_j$;
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 3. Let $S = (x, y, z, w)\,(C_{\mathsf{A}} \mid C'_{\mathsf{B}} \mid C''_{\mathsf{C}} \mid S_0)$, with S_0 immaterial, and:

$$C = \{\downarrow_x \overline{\mathsf{a}}, \downarrow_y \overline{\mathsf{b}}\} \qquad C' = \{\downarrow_z \mathsf{a}\} \qquad C'' = \{\downarrow_w \mathsf{b}\}$$

Further, let $\sigma = \{{}^{s}/_{x,z}, {}^{t}/_{y,w}\}$, $\gamma_{\mathsf{AB}} = \mathsf{A} : \overline{\mathsf{a}} \mid \mathsf{B} : \mathsf{a}$ and $\gamma_{\mathsf{AC}} = \mathsf{A} : \overline{\mathsf{b}} \mid \mathsf{C} : \mathsf{b}$. According to Definition 6 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, C' and C'' contain pairwise distinct variables;
2. letting $D_{\mathsf{A}} = \{(\mathsf{A}, x, \overline{\mathsf{a}}), (\mathsf{A}, y, \overline{\mathsf{b}})\}$, $D_{\mathsf{B}} = \{(\mathsf{B}, z, \mathsf{a})\}$ and $D_{\mathsf{C}} = \{(\mathsf{C}, w, \mathsf{b})\}$, we can partition $D_{\mathsf{A}} \cup D_{\mathsf{B}} \cup D_{\mathsf{C}}$ into the subsets $\mathcal{M}_{\mathsf{AB}} = \{(\mathsf{A}, x, \overline{\mathsf{a}}), (\mathsf{B}, z, \mathsf{a})\}$ and $\mathcal{M}_{\mathsf{AC}} = \{(\mathsf{A}, y, \overline{\mathsf{b}}), (\mathsf{C}, w, \mathsf{b})\}$, where $\overline{\mathsf{a}} \bowtie \mathsf{a}$ and $\overline{\mathsf{b}} \bowtie \mathsf{b}$.
3. σ maps session variables x, z, y, w to pairwise distinct session names s, t.

Therefore, by rule [Fuse], we have: $S \xrightarrow{\{\mathsf{K}\} : \tau} (s, t)\,(s[\gamma_{\mathsf{AB}}] \mid t[\gamma_{\mathsf{AC}}] \mid S_0\sigma)$. □

Example 4. Let $S = \mathsf{A}[(x)\,\mathrm{X}(x)] \mid \mathsf{B}[(y)\,\mathrm{Y}(y)]$, where:

$$\mathrm{X}(x) \triangleq \mathtt{tell}\,\{\downarrow_x \overline{\mathsf{a}}\}.\,\mathtt{do}_x\,\overline{\mathsf{a}} \qquad \mathrm{Y}(y) \triangleq \mathtt{tell}\,\{\downarrow_y \mathsf{a}\}.\,\mathtt{do}_y\,\mathsf{a}$$

A maximal computation of S is the following:

$$\begin{array}{lll}
S \xrightarrow{\{\mathsf{B}\}:\tau} & \mathsf{A}[(x)\,\mathrm{X}(x)] \mid (y)\,(\mathsf{B}[\mathtt{do}_y\,\mathsf{a}] \mid \{\downarrow_y \mathsf{a}\}_{\mathsf{B}}) & [\textsc{Tell}] \\
\xrightarrow{\{\mathsf{A}\}:\tau} & (x,y)\,(\mathsf{A}[\mathtt{do}_x\,\overline{\mathsf{a}}] \mid \mathsf{B}[\mathtt{do}_y\,\mathsf{a}] \mid \{\downarrow_x \overline{\mathsf{a}}\}_{\mathsf{A}} \mid \{\downarrow_y \mathsf{a}\}_{\mathsf{B}}) & [\textsc{Tell}] \\
\xrightarrow{\{\mathsf{K}\}:\tau} & (s)\big(\mathsf{A}[\mathtt{do}_s\,\overline{\mathsf{a}}] \mid \mathsf{B}[\mathtt{do}_s\,\mathsf{a}] \mid s[\mathsf{A} : \overline{\mathsf{a}} \parallel \mathsf{B} : \mathsf{a}]\big) & [\textsc{Fuse}] \\
\xrightarrow{\{\mathsf{A},\mathsf{B}\}:\mathtt{do}_s\,\mathsf{a}} & (s)\,(\mathsf{A}[\mathbf{0}] \mid \mathsf{B}[\mathbf{0}] \mid s[\mathsf{A} : \mathbf{0} \parallel \mathsf{B} : \mathbf{0}]) & [\textsc{DoCom}]
\end{array}$$

4 Honesty: properties and variants

CO_2 allows for writing *dishonest* processes which do not fulfil their contracts, in some contexts. Below we formalise some notions of honesty, which vary according to the assumptions on the context. We start by introducing some auxiliary notions. The *obligations* $\mathrm{O}^{\mathsf{A}}_s(S)$ of a participant A at a session s in S are those actions of A enabled in the contract configuration within s in S.

Definition 7 (Obligations). *We define the set of actions* $\mathrm{O}^{\mathsf{A}}_s(S)$ *as:*

$$\mathrm{O}^{\mathsf{A}}_s(S) = \begin{cases} \mathrm{O}^{\mathsf{A}}(\gamma) & \text{if } \exists S' \,.\, S \equiv s[\gamma] \mid S' \\ \emptyset & \text{otherwise} \end{cases} \quad \text{where } \mathrm{O}^{\mathsf{A}}(\gamma) = \{\alpha \mid \exists \mathcal{A} . \gamma \xrightarrow{\{\mathsf{A}\}\cup\mathcal{A}:\alpha} \!\!\!\!\rightarrow\!\!\!\!\rightarrow\}$$

The set $S\downarrow^{\mathsf{A}}_u$ (called ready-do set) collects all the actions α such that the process of A in S has some unguarded prefixes $\mathtt{do}_u\,\alpha$.

Definition 8 (Ready-do). *We define the set of actions* $S\downarrow^{\mathsf{A}}_u$ *as:*

$$S\downarrow^{\mathsf{A}}_u = \left\{\alpha \mid \exists \boldsymbol{v}, P, P', Q, S' \,.\, S \equiv (\boldsymbol{v})\,(\mathsf{A}[\mathtt{do}_u\,\alpha.\,P + P' \mid Q] \mid S') \,\wedge\, u \notin \boldsymbol{v}\right\}$$

4.1 Honesty

A participant is *ready* in a system if she can fulfil some of her obligations there (Definition 10). To check if A is ready in S, we consider all the sessions s in S involving A. For each of them, we check that some obligations of A at s are exposed after some steps (of A or of the context), *not* preceded by other $\mathtt{do}_s$ of A. These actions are collected in the set $S\Downarrow^{\mathsf{A}}_s$.

Definition 9 (Weak ready-do). *We define the set of actions* $S\Downarrow^{\mathsf{A}}_u$ *as:*

$$S\Downarrow^{\mathsf{A}}_u = \left\{\alpha \mid \exists S' : S \xrightarrow{\neq(\mathsf{A}\,:\,\mathtt{do}_u)}{}^{*} S' \text{ and } \alpha \in S'\downarrow^{\mathsf{A}}_u\right\}$$

where $S \xrightarrow{\neq(\mathsf{A}\,:\,\mathtt{do}_u)} S'$ *iff* $\exists \mathcal{A}, \pi \,.\; S \xrightarrow{\mathcal{A}:\,\pi} S' \,\wedge\, (\mathsf{A} \notin \mathcal{A} \,\vee\, \forall \alpha \,.\, \pi \neq \mathtt{do}_u\,\alpha)$.

The set $\mathrm{Rdy}_s^{\mathsf{A}}$ collects all the systems where A is ready at session s. This happens in three cases: either A has no obligations, or A may perform *some* internal action which is also an obligation, or A may perform *all* the synchronisation actions which are obligations.

Definition 10 (Readiness). $\mathrm{Rdy}_s^{\mathsf{A}}$ *is the set of systems* S *such that:*

$$\mathrm{O}_s^{\mathsf{A}}(S) = \emptyset \quad \vee \quad \mathrm{O}_s^{\mathsf{A}}(S) \cap \Lambda^{\tau} \cap S\Downarrow_s^{\mathsf{A}} \neq \emptyset \quad \vee \quad \emptyset \neq (\mathrm{O}_s^{\mathsf{A}}(S) \cap \Lambda^{\mathtt{a}}) \subseteq S\Downarrow_s^{\mathsf{A}}$$

We say that A *is* ready in S *iff* $\forall S', \boldsymbol{u}, s \,.\, S \equiv (\boldsymbol{u})S'$ *implies* $S' \in \mathrm{Rdy}_s^{\mathsf{A}}$.

We can now formalise when a participant is *honest*. Roughly, $\mathsf{A}[P]$ is honest in a *fixed* system S when A is ready in all reducts of $\mathsf{A}[P] \mid S$. Then, we say that $\mathsf{A}[P]$ is honest when she is honest in *all* systems S.

Definition 11 (Honesty). *Given a set of contracts* $\mathbb{C} \subseteq \mathbb{U}$ *and a set of processes* $\mathcal{P} \subseteq \boldsymbol{\mathcal{P}}_{\mathbb{C}}$*, we say that:*

1. S is A-free *iff it has no latent/stipulated contracts of* A*, nor processes of* A
2. P *is* honest in $\mathcal{P}$ *iff, for all* S *made of agents with processes in* $\mathcal{P}$*:*
$$\forall \mathsf{A} : \left(S \text{ is } \mathsf{A}\text{-free} \wedge \mathsf{A}[P] \mid S \to^* S'\right) \implies \mathsf{A} \text{ is ready in } S'$$
3. P *is* honest *iff* $P \in \boldsymbol{\mathcal{H}}_{\mathbb{C}}$*, where:*
$$\boldsymbol{\mathcal{H}}_{\mathbb{C}} \;=\; \{P \in \boldsymbol{\mathcal{P}}_{\mathbb{U}} \mid P \text{ is honest in } \boldsymbol{\mathcal{P}}_{\mathbb{C}}\}$$

Note that in item 2 we quantify over all A: this is needed to associate P to a participant name, with the only constraint that such name must not be present in the context S used to test P. In the absence of the A-freeness constraint, honesty would be impractically strict: indeed, were S already carrying stipulated or latent contracts of A, e.g. with $S = s[\mathsf{A} : \overline{\mathtt{pay100Keu}} \parallel \mathsf{B} : \mathtt{pay100Keu}]$, it would be unreasonable to ask participant A to fulfil them. Note however that S can contain latent contracts and sessions involving *any* other participant different from A: in a sense, the honesty of $\mathsf{A}[P]$ ensures a good behaviour even in the (quite realistic) case where $\mathsf{A}[P]$ is inserted in a system which has already started.

Example 5. Consider the following processes:

1. $P_1 \;=\; (x)\,\mathtt{tell}\,\{\downarrow_x \mathtt{a} + \tau.\mathtt{b}\}.\,\mathtt{do}_x\,\tau.\,\mathtt{do}_x\,\mathtt{b}$
2. $P_2 \;=\; (x)\,\mathtt{tell}\,\{\downarrow_x \mathtt{a}\}.\,(\tau.\mathtt{do}_x\,\mathtt{a} + \tau.\mathtt{do}_x\,\mathtt{b})$
3. $P_3 \;=\; (x)\,\mathtt{tell}\,\{\downarrow_x \mathtt{a} + \mathtt{b}\}.\,\mathtt{do}_x\,\mathtt{a}$
4. $P_4 \;=\; (x)\,\mathtt{tell}\,\{\downarrow_x \mathtt{a}\}.\,\mathrm{X}(x) \qquad \mathrm{X}(x) \triangleq \tau.\,\mathtt{do}_x\,\mathtt{a} + \tau.\,\mathrm{X}(x)$
5. $P_5 \;=\; (x\,y)\,\mathtt{tell}\,\{\downarrow_x \mathtt{a}\}.\,\mathtt{tell}\,\{\downarrow_y \mathtt{b}\}.\,\mathtt{do}_x\,\mathtt{a}.\,\mathtt{do}_y\,\mathtt{b}$

Processes P_1 and P_4 are honest, while the others are not. In P_2, if the rightmost τ is fired, then the process cannot do the promised $\mathtt{a}$. In P_3, if the contract of other participant at x is $\overline{\mathtt{b}}$, then P_3 cannot do the corresponding $\mathtt{b}$. There are two different reasons for which P_5 is not honest. First, in contexts where y is fused and x is not, the $\mathtt{do}_y\,\mathtt{b}$ can not be reached (and so the contract at y is not respected). Second, also in those contexts where both sessions are fused, if the other participant at x never does $\overline{\mathtt{a}}$, then $\mathtt{do}_y\,\mathtt{b}$ cannot be reached. □

Example 6. We now model in CO_2 the store process outlined in Section 1. Rather than giving a faithful formalisation of the pseudo-code in Section 1, which we observed to be dishonest, we present an alternative version. The process P below is honest, and it can be proved such by the honesty model checker in [7]. Within this example, we use $\mathtt{do}_x^\tau\,\overline{\mathtt{a}}$ as an abbreviation for $\mathtt{do}_x\,\tau_{\mathtt{a}}.\,\mathtt{do}_x\,\overline{\mathtt{a}}$.

$$
\begin{aligned}
P &= (x)\ \mathtt{tell}\,\{\downarrow_x T_{\mathsf{B}}\}_{\mathsf{A}}.\,(\mathtt{do}_x\,\mathtt{buyA}.\,P_A(x) + \mathtt{do}_x\,\mathtt{buyB}.\,P_B(x))\\
P_A(x) &\triangleq \mathtt{do}_x\,\mathtt{pay1E}.\,\mathtt{do}_x^\tau\,\overline{\mathtt{shipA}}\\
P_B(x) &\triangleq (y)\ \big(\tau.\,\mathtt{do}_x^\tau\,\overline{\mathtt{quote1E}}.\mathtt{do}_x\,\mathtt{pay1E}.\,\mathtt{do}_x^\tau\,\overline{\mathtt{shipB}} +\\
&\qquad \tau.\,\mathtt{tell}\,\{\downarrow_x T_{\mathsf{D}}\}_{\mathsf{A}}.\mathtt{do}_y^\tau\,\overline{\mathtt{buyB}}.\mathtt{do}_x^\tau\,\overline{\mathtt{quote3E}}.\,P_{B2}(x,y) +\\
&\qquad \tau.\,P_{abort}(x,y)\big)\\
P_{abort}(x,y) &\triangleq \mathtt{do}_x^\tau\,\overline{\mathtt{abort}} \mid \mathtt{do}_y^\tau\,\overline{\mathtt{buyB}} \mid \mathtt{do}_y^\tau\,\overline{\mathtt{quit}}\\
P_{B2}(x,y) &\triangleq \mathtt{do}_x\,\mathtt{pay3E}.\,P_{B3}(x,y) + \mathtt{do}_x\,\mathtt{quit}.\,\mathtt{do}_y^\tau\,\overline{\mathtt{quit}} + \tau.\,P_{abort2}(x,y)\\
P_{abort2}(x,y) &\triangleq (\mathtt{do}_x\,\mathtt{pay3E}.\,\mathtt{do}_x^\tau\,\overline{\mathtt{refund}} + \mathtt{do}_x\,\mathtt{quit}) \mid \mathtt{do}_y^\tau\,\overline{\mathtt{quit}}\\
P_{B3}(x,y) &\triangleq \mathtt{do}_y^\tau\,\overline{\mathtt{pay2E}}.\,P_{B4}(x,y) + \tau.\,P_{abort3}(x,y)\\
P_{abort3}(x,y) &\triangleq \mathtt{do}_x^\tau\,\overline{\mathtt{refund}} \mid \mathtt{do}_y^\tau\,\overline{\mathtt{quit}}\\
P_{B4}(x,y) &\triangleq \mathtt{do}_y\,\mathtt{shipB}.\,\mathtt{do}_x^\tau\,\overline{\mathtt{shipB}} + \tau.\,P_{abort4}(x,y)\\
P_{abort4}(x,y) &\triangleq \mathtt{do}_x^\tau\,\overline{\mathtt{refund}} \mid \mathtt{do}_y\,\mathtt{shipB}
\end{aligned}
$$

4.2 Solo-honesty

The notion of honesty studied so far requires that, in all contexts, whenever A has some obligations, *the system* must be able to evolve to a state in which A exposes some $\mathtt{do}$ (the *ready-do*) to fulfil her obligations. In other words, A is allowed to interact with the context, from which she can receive some help.

A natural variant of honesty would require A to be able to fulfil her obligations without any help from the context. To define this (intuitively stricter) variant of honesty, we modify the definition of *weak ready-do* to forbid the rest of the system to move. The actions reachable in such way are then named *solo weak ready-do*, and form a smaller set than the previous notion. The definitions of *solo-ready* and *solo-honest* consequently follow — *mutatis mutandis*.

Definition 12 (Solo weak ready-do). $S \Downarrow_u^{\mathsf{A}\text{-solo}}$ *is the sets of actions:*

$$S \Downarrow_u^{\mathsf{A}\text{-solo}} = \left\{ \alpha \mid \exists S' \,.\, S \xrightarrow{(\mathsf{A}\,:\,\neq \mathtt{do}_u)}{}^{*}\ S' \text{ and } \alpha \in S' \downarrow_u^{\mathsf{A}} \right\}$$

where $S \xrightarrow{(\mathsf{A}\,:\,\neq \mathtt{do}_u)} S'$ *iff* $\exists \pi\ .\ \ S \xrightarrow{\{\mathsf{A}\}\,:\,\pi} S' \ \wedge\ (\forall \alpha.\ \pi \neq \mathtt{do}_u\,\alpha)$.

Definition 13 (Solo readiness). $\mathrm{Rdy}_s^{\mathsf{A}\text{-solo}}$ *is the set of systems* S *such that:*

$$\mathrm{O}_s^{\mathsf{A}}(S) = \emptyset \quad \vee \quad \mathrm{O}_s^{\mathsf{A}}(S) \cap \Lambda^\tau \cap S \Downarrow_s^{\mathsf{A}\text{-solo}} \neq \emptyset \quad \vee \quad \emptyset \neq (\mathrm{O}_s^{\mathsf{A}}(S) \cap \Lambda^{\mathtt{a}}) \subseteq S \Downarrow_s^{\mathsf{A}\text{-solo}}$$

We say that A *is* solo-ready *in* S *iff* $\forall S', \boldsymbol{u}, s\,.\,S \equiv (\boldsymbol{u})S'$ *implies* $S' \in \mathrm{Rdy}_s^{\mathsf{A}\text{-solo}}$.

Definition 14 (Solo honesty). *We say that* P *is* solo-honest *in* S *iff*

$$\forall \mathsf{A} : \big(S \text{ is } \mathsf{A}\text{-free} \wedge \mathsf{A}[P] \mid S \to^* S'\big) \implies \mathsf{A} \text{ is solo-ready in } S'$$

We now relate solo honesty with the notion of honesty in Definition 11. As expected, when considering a fixed context S, solo honesty implies honesty, and is in general a stricter notion. However, being honest in *all* contexts is equivalent to being solo-honest in *all* contexts, as established by the following theorem.

Theorem 1. *For all processes* P *and systems* S*:*

1. *if* P *is solo-honest in* S*, then* P *is honest in* S*;*
2. *the converse of item 1 does not hold, in general;*
3. P *is solo-honest iff* P *is honest.*

Proof. Item 1 follows from definition of solo-readiness and $S \Downarrow_s^{\mathsf{A}\text{-solo}} \subseteq S \Downarrow_s^{\mathsf{A}}$.

For item 2, let:

$$\begin{aligned} P &= \mathsf{A}[(x,y)\ \mathtt{tell}\ \{\downarrow_x \overline{\mathtt{a}}\}.\ \mathtt{tell}\ \{\downarrow_y \overline{\mathtt{b}}\}.\ \mathtt{do}_x\ \overline{\mathtt{a}}.\ \mathtt{do}_y\ \overline{\mathtt{b}}] \\ S &= \mathsf{B}[(z)\ \mathtt{tell}\ \{\downarrow_z \mathtt{a}\}.\ \mathtt{do}_z\ \mathtt{a}] \mid \mathsf{C}[(w)\ \mathtt{tell}\ \{\downarrow_w \mathtt{b}\}.\ \mathtt{do}_w\ \mathtt{b}] \end{aligned}$$

We have that P is honest in S, but not solo-honest in S. Indeed, after both contracts of A get stipulated, A needs to perform $\overline{\mathtt{b}}$ in session y, but she can only do that if B cooperates, allowing A to first perform $\overline{\mathtt{a}}$ in session x.

For item 3, the "only if" direction immediately follows from item 1. For the "if" direction, assume by contradiction that P is honest but not solo-honest, i.e.:

$$\mathsf{A}[P] \mid S \to^* (\boldsymbol{\nu})\ (\mathsf{A}[P'] \mid S')$$

where $\mathsf{A}[P']$ has some obligations to perform for which she can not reach any related *ready do* on her own, but needs to interact with the context S' to do that. In such case, it is possible to craft another A-free initial system S'', which behaves exactly as S in the computation shown above, yet stops interacting at the end of such computation. Basically, given the computation above, we can construct S'' as the parallel composition of agents of the form $\mathsf{B}[(\boldsymbol{x})\pi_1.\ldots.\pi_n]$. Each prefix π_i performs a `tell` or a `do` in the same order as in the computation above. This makes it possible to obtain an analogous computation

$$\mathsf{A}[P] \mid S'' \to^* (\boldsymbol{\nu})\ (\mathsf{A}[P'] \mid S''')$$

where S''' does no longer interact with A. However, since A is honest, she must be able to fulfil her obligations with the help of her context in $\mathsf{A}[P'] \mid S'''$. Since the context does not cooperate, she must actually be able to do that with solo transitions — contradiction. □

4.3 Weak honesty

The honesty property requires a process to be ready even in those (dishonest) contexts where the other participants avoid to do the required actions. A weaker variant of honesty may require a process P to behave correctly provided that also the others behave correctly, i.e. that P is ready in *honest contexts*, only.

Definition 15 (Weak honesty). *Given a set of contracts $\mathbb{C}$, we define the set of weakly honest processes as:*

$$\boldsymbol{\mathcal{W}}_{\mathbb{C}} \;=\; \{P \in \boldsymbol{\mathcal{P}}_{\mathbb{U}} \mid P \textit{ is honest in } \boldsymbol{\mathcal{H}}_{\mathbb{C}}\}$$

Example 7. The process P_5 from Example 5 is *not* weakly honest. Let, e.g.:

$$Q_5 \;=\; (w)\,(\mathtt{tell}\,\{\downarrow_w \overline{\mathtt{b}}\}.\,\mathtt{do}_w\,\overline{\mathtt{b}})$$

which is clearly honest. However, by reducing $\mathsf{A}[P_5] \mid \mathsf{C}[Q_5]$ we reach the state:

$$S \;=\; (s,x)\,\big(\,\mathsf{A}[\mathtt{do}_x\,\mathtt{a}.\,\mathtt{do}_s\,\mathtt{b}] \;\mid\; \mathsf{C}[\mathtt{do}_s\,\overline{\mathtt{b}}] \;\mid\; s[\mathsf{A} : \mathtt{b} \parallel \mathsf{C} : \overline{\mathtt{b}}]\,\big)$$

where A is not ready. The problem here is that there is no guarantee that the contract on x is always stipulated. We can fix this by making A advertise both contracts atomically. This is done as follows:

$$P_5{}' \;=\; (x,y)\,\mathtt{tell}\,\{\downarrow_x \mathtt{a}\,,\downarrow_y \mathtt{b}\}.\,\mathtt{do}_x\,\mathtt{a}.\,\mathtt{do}_y\,\mathtt{b}$$

The process $P_5{}'$ is weakly honest, but it is *not* honest: in fact, in a context where the other participant in session x does not fire $\overline{\mathtt{a}}$, A is not ready at y. □

The following theorem states that the set of weakly honest processes is larger (for certain classes of contracts, *strictly*) than the set of honest ones.

Theorem 2. *For all $\mathbb{C}$, $\boldsymbol{\mathcal{H}}_{\mathbb{C}} \subseteq \boldsymbol{\mathcal{W}}_{\mathbb{C}}$. Furthermore, $\boldsymbol{\mathcal{H}}_{\mathrm{ST}} \neq \boldsymbol{\mathcal{W}}_{\mathrm{ST}}$.*

Proof. The inclusion follows from Definition 15; the inequality from the process $P_5{}'$ in Example 7, which belongs to $\boldsymbol{\mathcal{W}}_{\mathrm{ST}}$ but not to $\boldsymbol{\mathcal{H}}_{\mathrm{ST}}$. □

The definition of $\boldsymbol{\mathcal{H}}_{\mathbb{C}}$ requires honesty in *all* contexts, i.e. in all systems composed of processes in $\boldsymbol{\mathcal{P}}_{\mathbb{C}}$. Instead, $\boldsymbol{\mathcal{W}}_{\mathbb{C}}$ requires honesty in all $\boldsymbol{\mathcal{H}}_{\mathbb{C}}$ contexts. This step can be iterated further: what if we require honesty in all $\boldsymbol{\mathcal{W}}_{\mathbb{C}}$ contexts? As we establish below, we get back to $\boldsymbol{\mathcal{H}}_{\mathbb{C}}$.

Theorem 3. *For all $\mathbb{C}$:* $\boldsymbol{\mathcal{H}}_{\mathbb{C}} \;=\; \{P \mid P \textit{ is honest in } \boldsymbol{\mathcal{W}}_{\mathbb{C}}\}$.

Proof (Sketch). The $\subseteq$ inclusion trivially holds. For the $\supseteq$ inclusion, it is possible to craft a context of weakly honest processes which open sessions with P, possibly interact with P in such sessions for a while, and then stop to perform any action. This can be achieved as follows:

$$\mathsf{B}[(x,y,z)\;\mathtt{tell}\,\{\downarrow_z c\}.\,\mathtt{tell}\,\{\downarrow_x \mathtt{a}\,,\downarrow_y \mathtt{b}\}.\,\mathtt{do}_x\,\mathtt{a}\,.\,\mathtt{do}_y\,\mathtt{b}.\,Q] \mid$$
$$\mathsf{C}[(v,w)\;\mathtt{tell}\,\{\downarrow_v \overline{\mathtt{a}}\,,\downarrow_w \overline{\mathtt{b}}\}.\,\mathtt{do}_w\,\overline{\mathtt{b}}.\,\mathtt{do}_v\,\overline{\mathtt{a}}]$$

where c is a contract compliant with some of the contracts P advertises, and Q is a honest implementation of c. Note that B above can also start two sessions with contracts $\{\downarrow_x \mathtt{a}, \downarrow_y \mathtt{b}\}$ with C, which however will deadlock because B and C perform the actions in a different order. This will cause Q to never be reached. Yet, both B and C are weakly honest: each of them would work fine in a honest context, since no deadlock would be possible there. The context above can also be adapted to postpone the deadlock so to effectively stop in the middle of executing Q, i.e. in the middle of session z. Because P must be honest in this weakly honest context, P must, at any time, be able to perform its obligations without relying on the context. Hence, $P \in \boldsymbol{\mathcal{H}}_{\mathbb{C}}$. □

4.4 Some properties

The function $\lambda X.\, \boldsymbol{\mathcal{H}}_X$ is anti-monotonic, as formalised by the following theorem (which follows directly from Definition 11).

Theorem 4. *If* $\mathbb{C} \subseteq \mathbb{D}$, *then* $\boldsymbol{\mathcal{H}}_{\mathbb{C}} \supseteq \boldsymbol{\mathcal{H}}_{\mathbb{D}}$.

The following theorem states a peculiar property of processes which use session types as contracts. If some of such processes is honest in all contexts where contracts are session types, then it is honest in *all* possible contexts.

Theorem 5. $\boldsymbol{\mathcal{P}}_{\mathrm{ST}} \cap \boldsymbol{\mathcal{H}}_{\mathrm{ST}} = \boldsymbol{\mathcal{P}}_{\mathrm{ST}} \cap \boldsymbol{\mathcal{H}}_{\mathbb{U}}$.

Proof. The inclusion $\supseteq$ follows by Theorem 4. For the inclusion $\subseteq$, assume by contradiction that $P \in \boldsymbol{\mathcal{H}}_{\mathrm{ST}} \setminus \boldsymbol{\mathcal{H}}_{\mathbb{U}}$, i.e. P is honest in $\boldsymbol{\mathcal{P}}_{\mathrm{ST}}$, but *not* honest in $\boldsymbol{\mathcal{P}}_{\mathbb{U}}$. Then, there exists some S made of agents with processes in $\boldsymbol{\mathcal{P}}_{\mathbb{U}}$ such that:

$$\mathsf{A}[P] \mid S \to^* (\boldsymbol{\nu})\, (\mathsf{A}[P'] \mid S' \mid s[\mathsf{A} : c \parallel \mathsf{B} : d]) \tag{1}$$

where $\mathsf{A}[P']$ has some obligations at s, such that either:

1. c is an internal choice, and no internal transition of A is included in the weak ready-do set of A at s, or
2. c is an external (or committed) choice, and the weak ready-do set does not include all the labels enabled by $\mathsf{A} : c \parallel \mathsf{B} : d$.

We can craft an A-free system S'' (with processes in $\boldsymbol{\mathcal{P}}_{\mathrm{ST}}$) which interacts with A as S in (1), after which it does nothing (except possibly firing $\mathtt{do}_s\, \tau$). We can construct S'' as the parallel composition of agents of the form $\mathsf{B}[(\boldsymbol{x})\pi_1. \ldots . \pi_n]$. Each prefix π_i performs a $\mathtt{tell}$ or a $\mathtt{do}$ in the same order as in (1), after removing from it the steps not involving A: e.g., a $\mathtt{tell}$ of a contract which is not stipulated with A is omitted. Instead, a $\mathtt{tell}$ of a contract $d_i \notin \mathrm{ST}$ which will be fused with some c_i of A is replaced by $\mathtt{tell}\, \overline{c_i}$, where $\overline{c_i}$ is the *syntactic dual* of c_i (which always exists and belongs to ST). We then obtain a computation:

$$\mathsf{A}[P] \mid S'' \to^* (\boldsymbol{\nu})\, (\mathsf{A}[P'] \mid S''' \mid s[\mathsf{A} : c \parallel \mathsf{B} : \overline{c}])$$

where S''' does no longer interact with A, except possibly firing $\mathtt{do}_s\, \tau$, if enabled. In the resulting system, A is *not* ready: therefore, P is not honest in S'. □

The following theorem establishes a crucial property of honest processes, i.e. that deadlock-freedom at the level of contracts is preserved when passing to the level of (honest) processes. This means that all open sessions can be carried forward until their successful termination.

Theorem 6 (Deadlock freedom). *Let S be a system of honest agents. If $S \to^* (\boldsymbol{u})\,(S' \mid s[\gamma])$ with $\mathrm{O}^{\mathsf{A}}(\gamma) \neq \emptyset$, then there exist S'', $\mathcal{A}$, and $\alpha \in \mathrm{O}^{\mathsf{A}}(\gamma)$ such that $S' \mid s[\gamma] \to^* S'' \xrightarrow{\{\mathsf{A}\}\cup\mathcal{A} : \mathtt{do}_s\ \alpha}$.*

Proof. Assume first that $\mathrm{O}^{\mathsf{A}}(\gamma)$ only contains synchronisation actions, and let:

$$\gamma \xrightarrow{\{\mathsf{A},\mathsf{B}\}:\mathsf{a}} \qquad S = \mathsf{A}[P] \mid \mathsf{B}[Q] \mid \cdots \qquad S_0 = S' \mid s[\gamma]$$

with P and Q honest by hypothesis. By item 3 of Theorem 1, P and Q are also solo-honest. By Definition 7 it must be $\mathtt{a} \in \mathrm{O}^{\mathsf{A}}_s(S_0)$ and $\overline{\mathtt{a}} \in \mathrm{O}^{\mathsf{B}}_s(S_0)$, and so by Definition 14 it must be $\mathtt{a} \in S_0 \Downarrow^{\mathsf{A}\text{-solo}}_s$ and $\overline{\mathtt{a}} \in S_0 \Downarrow^{\mathsf{B}\text{-solo}}_s$. Since P is solo-honest, by Definition 13 we have that $\exists S'_0 . S_0 \xrightarrow{(\mathsf{A} :\, \neq \mathtt{do}_s)}{}^* S'_0$ and $\mathtt{a} \in S'_0 \downarrow^{\mathsf{A}}_s$. Since B has taken no transitions in this computation, and the contract configuration at s is still γ, it must be $\overline{\mathtt{a}} \in \mathrm{O}^{\mathsf{B}}_s(S'_0)$, and $\overline{\mathtt{a}} \in S'_0 \Downarrow^{\mathsf{B}\text{-solo}}_s$. Since Q is solo-honest, by Definition 13 we have that $\exists S'' . S'_0 \xrightarrow{(\mathsf{B} :\, \neq \mathtt{do}_s)}{}^* S''$ and $\overline{\mathtt{a}} \in S'' \downarrow^{\mathsf{B}}_s$. Since A has taken no transitions in this computation, and the contract configuration at s is still γ, at this point we have $\mathtt{a} \in S'' \downarrow^{\mathsf{A}}_s$ and $\overline{\mathtt{a}} \in S'' \downarrow^{\mathsf{B}}_s$. Then, by rule [DoCom], we obtain the thesis $S_0 = S' \mid s[\gamma] \to^* S'' \xrightarrow{\{\mathsf{A},\mathsf{B}\} : \mathtt{do}_s\ \alpha}$. The case where $\mathrm{O}^{\mathsf{A}}(\gamma)$ may contain internal actions is similar. □

Example 8. Note that Theorem 6 would not hold if we required weak honesty instead of honesty. For instance, consider the process $P_5{}'$ in Example 7, and let:

$$Q_5{}' \;=\; (x, y)\ \mathtt{tell}\,\{\downarrow_x \overline{\mathtt{a}}\,, \downarrow_y \overline{\mathtt{b}}\}.\ \mathtt{do}_y\,\overline{\mathtt{b}}.\ \mathtt{do}_x\,\overline{\mathtt{a}}$$

Both $P_5{}'$ and $Q_5{}'$ are weakly honest, but their composition $\mathsf{A}[P_5{}'] \mid \mathsf{B}[Q_5{}']$ gets stuck on the first $\mathtt{do}$, since neither $\mathtt{do}_x\,\mathtt{a}$ nor $\mathtt{do}_y\,\overline{\mathtt{b}}$ can be fired. □

5 Decidability results

In this section we prove that both honesty and weak honesty are undecidable.

5.1 Honesty is undecidable

The following theorem states that honesty is undecidable, when using contracts which are at least as expressive as session types. To prove it, we show that the complement problem, i.e. deciding if a participant is *dishonest*, is not recursive.

Theorem 7. *$\mathcal{H}_{\mathbb{C}}$ is not recursive if $\mathbb{C} \supseteq \mathrm{ST}$.*

Proof. We reduce the halting problem on Turing machines to the problem of checking *dis*honesty of $P_0 \in \boldsymbol{\mathcal{P}}_{\mathbb{C}}$. This immediately gives the thesis. Given an arbitrary Turing machine M, we represent its configurations as finite sequences $(\lambda_0, \star)\ (\lambda_1, \star)\ \cdots\ (\lambda_n, q)\ \cdots\ (\lambda_k, \star)$, where:

1. λ_i represents the symbol written at the i-th cell of the tape,
2. $\star$ is not a state of M (just used to represent the absence of the head);
3. the single occurrence of the pair (λ_n, q) denotes that the head of M is over the n-th cell, and M is in state q,
4. the tape implicitly contains "blank" symbols at cells after position k,
5. λ_i and q range over finite sets.

Without loss of generality, assume that M halts only when its head is over λ_0 and M is in the halting state q_{stop}.

We now devise an effective procedure to construct a process P_0 which is dishonest if and only if M halts on the empty tape. This P_0 has the form:

$$(x)\ \mathtt{tell}\ \{\downarrow_x c\}.\ \mathtt{do}_x\, \tau_{\overline{\mathtt{a}}}.\ \mathtt{do}_x\, \overline{\mathtt{a}}.\ P \tag{2}$$

where $c = \mathtt{rec}\, X.\overline{\mathtt{a}}.X$, and P will be defined below. Intuitively, P_0 will interact with the context in order to simulate M; concretely, this will require P_0 to create new sessions. Note that some contexts may hinder P_0 in this simulation, e.g. by not advertising contracts or by refusing to interact properly in these sessions. Roughly, we will have that:

- in all contexts, P_0 will behave honestly in all sessions, except possibly in x;
- if the context does not cooperate, then P_0 will stop simulating M, but will still behave honestly in all sessions (including x);
- if the context cooperates, then P_0 will simulate M while being honest; only when M halts, P_0 will become dishonest, by stopping to do the required actions in session x.

The above intuition suffices for our purposes. Formally, we guarantee that:

1. if M does *not* halt, then P_0 is honest in *all* contexts (and therefore honest);
2. if M halts, then P_0 is *not* honest in *at least one* (cooperating) context (and therefore dishonest).

We represent each cell of the tape as a contract $d_{\lambda,\rho}$ in which λ is a symbol of the alphabet of M, and ρ is either a state of M or $\star$. More precisely, we specify $d_{\lambda,\rho}$ by mutual recursion as:

$$d_{\lambda,\rho} \;=\; \overline{\mathtt{read}_{\lambda,\rho}}.d_{\lambda,\rho} \;\oplus\; \bigoplus\nolimits_{\lambda'} \overline{\mathtt{write}_{\lambda'}}.d_{\lambda',\rho} \;\oplus\; \bigoplus\nolimits_{\rho'} \overline{\mathtt{write}_{\rho'}}.d_{\lambda,\rho'}$$

where $\overline{\mathtt{read}_{\lambda,\rho}}$, $\overline{\mathtt{write}_{\lambda}}$, $\overline{\mathtt{write}_{\rho}}$ are output actions. Note in passing that mutual recursion can be reduced to single recursion via the *rec* construct (up to some unfolding, as by Bekić's Theorem): therefore, $d_{\lambda,\rho} \in \mathrm{ST}$.

We now sketch the construction of process P in (2). Intuitively, P uses the above contracts in separate sessions (one for each tape cell), and it evolves into processes of the form:

$$\mathrm{Begin}(s_0, s_1) \mid \mathrm{X}(s_0, s_1, s_2) \mid \mathrm{X}(s_1, s_2, s_3) \mid \cdots \mid \mathrm{End}(s_{n-1}, s_n)$$

where $s_0, \ldots, s_n$ are distinct session names, and the contract of P at session s_i is d_{λ_i,ρ_i}. The intuition underlying processes Begin, X, and End is the following:

- a process $\mathrm{X}(\_, s_i, \_)$ is responsible for handling the i-th cell. It starts by reading the cell, which is obtained by performing:

  $$\textstyle\sum_{\lambda,\rho} \mathtt{do}_{s_i}\, \tau_{\mathtt{read}_{\lambda,\rho}} . \mathtt{do}_{s_i}\, \overline{\mathtt{read}_{\lambda,\rho}} .\, \mathrm{Handle}_{\lambda,\rho}$$

  Note that only one branch of the above summation is enabled, i.e. the one carrying the same λ, ρ as in the contract at session s_i. We now have the following two cases:
  - if the head of M is *not* on the i-th cell (i.e., $\rho = \star$), we have that $\mathrm{Handle}_{\lambda,\rho}$ recursively calls X. This makes the process repeatedly act on s_i, so making P behave honestly at that session.
  - if the head is on the i-th cell, $\mathrm{Handle}_{\lambda,\rho}$ updates the cell according to the transition rules of M, and then it moves the head as needed. Assume that q' is the new state of M, λ' is the symbol written at the i-th cell, and that $j \in \{i-1, i+1\}$ is the new head position. In the process, the cell update is obtained by performing $\overline{\mathtt{write}_{\lambda'}}$ in s_i, and the head update is obtained by performing $\overline{\mathtt{write}_{\star}}$ in s_i and $\overline{\mathtt{write}_{q'}}$ in s_j.
- the process $\mathrm{Begin}(s_0, s_1)$ handles the leftmost cell of the tape. Intuitively, it behaves as $\mathrm{X}(\_, s_0, s_1)$, but it also keeps on performing $\mathtt{do}_x\, \tau_{\overline{\mathtt{a}}}$ and $\mathtt{do}_x\, \overline{\mathtt{a}}$. In this way, $\mathrm{Begin}(s_0, s_1)$ respects the contract c in (2). When $\mathrm{Begin}(s_0, s_1)$ reads from s_0 that $\rho = q_{\mathrm{stop}}$, it stops performing the required actions at session x. This happens when M halts (which, by the assumptions above, can only happen when the head of M is on the leftmost cell). In this way, P_0 behaves dishonestly at session x.
- the process $\mathrm{End}(s_{n-1}, s_n)$ handles the rightmost cell of the tape. Intuitively, it behaves as $\mathrm{X}(s_{n-1}, s_n, \_)$, but it also waits to read $\rho \neq \star$, meaning that the head has reached the (rightmost) n-th cell. When this happens, the process $\mathrm{End}(s_{n-1}, s_n)$ creates a new session s_{n+1}, by advertising a contract $d_{\#,\star}$, where $\#$ is the blank tape symbol. Until the new session s_{n+1} is established, it keeps on acting on s_n, in order to behave honestly on that session. Once s_{n+1} is established, it spawns a new process $\mathrm{X}(s_{n-1}, s_n, s_{n+1})$, and then recurse as $\mathrm{End}(s_n, s_{n+1})$.

A crucial property is that it is possible to craft the above processes so that in no circumstances (including hostile contexts) they make P_0 dishonest at s_i. For example, $\mathrm{X}(\_, s_i, \_)$ is built so that it never stops performing reads at s_i. This property is achieved by encoding each potentially blocking operation $\mathtt{do}_{s_k}\, \alpha .\, P'$ as $Q = \mathtt{do}_{s_k}\, \alpha .\, P' + \sum_{\lambda,\rho} \mathtt{do}_{s_i}\, \mathtt{read}_{\lambda,\rho} .\, Q$. Indeed, in this way, reads on s_i are

continuously ready, preserving honesty. A similar technique is used to handle those τ_α which need to be performed without blocking the other activities.

To conclude, given a Turing Machine M we have constructed a process P_0 such that (i) if M does not halt, then P_0 is honest, while (ii) if M halts, then P_0 is not honest in some (cooperating) context. Note that a context which cooperates with P_0 always exists: since all the advertised contracts are session types, a context can simply advertise the duals of all the contracts possibly advertised by A (a finite number), and then (recursively) perform all the promised actions. □

5.2 Decidability of honesty in fragments of CO_2

While honesty of general CO_2 processes is undecidable, we can recover decidability in fragments of CO_2. In particular, by using the model-checking technique of [7], we can verify the honesty of processes which are essentially finite state, i.e. they have no delimitation/parallel under process definitions. This technique uses an abstract semantics of CO_2 which preserves the transitions of an agent $\mathsf{A}[P]$, while abstracting from the context wherein $\mathsf{A}[P]$ is run. This is established by the following theorem.

Theorem 8. *$P \in \boldsymbol{\mathcal{H}}_{\mathbb{C}}$ is decidable if (i) P has no delimitation/parallel under process definitions, and (ii) $\mathbb{C} \subseteq \mathbb{U}_{fin}$.*

Proof (Sketch). Building upon this abstract semantics of [7], we obtain an abstract notion of honesty which simulates the moves of unknown contexts, and it is sound and complete w.r.t. honesty. (i.e., P is abstractly honest iff it is honest, see [7] for further details). Since the abstract semantics is finite-state whenever P is such, then we can decide honesty of P by model-checking its state space under the abstract semantics. □

5.3 Dishonesty is recursively enumerable

We show in Theorem 9 that dishonesty is recursively enumerable, under certain assumptions on the set of contracts. Together with Theorem 7, it follows that honesty is neither recursive nor recursively enumerable.

Theorem 9. *$\overline{\boldsymbol{\mathcal{H}}_{\mathbb{C}}}$ is recursively enumerable if (i) for all $c \in \mathbb{C}$, $\{\alpha \mid c \xrightarrow{\alpha}\}$ is a finite set, and it is computable from c, and (ii) $\mathbb{C} \subseteq \mathbb{U}_{fin}$.*

Proof. We prove that "$\mathsf{A}[P]$ dishonest" is a r.e. property. By item 3 of Theorem 1, it suffices to prove that "$\mathsf{A}[P]$ solo-dishonest" is a r.e. property. By Definition 14, $\mathsf{A}[P]$ is *not* solo-honest iff there exists some A-free context S such that A is not solo-honest in $\mathsf{A}[P] \mid S$. This holds when A is not solo-ready in some residual of $\mathsf{A}[P] \mid S$, i.e. when the following conditions hold for some $S, S', s, \boldsymbol{u}$: (1) S is A-free; (2) $\mathsf{A}[P] \mid S \rightarrow^* (\boldsymbol{u})\ S'$; (3) $S' \notin \mathrm{Rdy}_s^{\mathsf{A}\text{-solo}}$.

Recall that $p(x, y)$ r.e. implies that $q(y) = \exists x.p(x, y)$ is r.e., provided that x ranges over an effectively enumerable set (e.g., systems S, or sessions s). Thus, to prove the above existentially-quantified property r.e. it suffices to prove that

1), 2), 3) are r.e.. Property 1 is trivially recursive. Property 2 is r.e. since one can enumerate all the possible finite traces. Property 3 is shown below to be recursive, by reducing the problem to a submarking reachability problem in Petri Nets, which is decidable [17]. We recall the definition of $S' \in \mathrm{Rdy}_s^{\mathsf{A}\text{-solo}}$:

$$\mathrm{O}_s^{\mathsf{A}}(S') = \emptyset \ \vee \ \mathrm{O}_s^{\mathsf{A}}(S') \cap \Lambda^{\tau} \cap S' \Downarrow_s^{\mathsf{A}\text{-solo}} \neq \emptyset \ \vee \ \emptyset \neq (\mathrm{O}_s^{\mathsf{A}}(S') \cap \Lambda^{\mathtt{a}}) \subseteq S' \Downarrow_s^{\mathsf{A}\text{-solo}}$$

To prove the above property recursive, we start by noting that, by hypothesis, $\mathrm{O}_s^{\mathsf{A}}(S')$ is a finite set, and it can be effectively enumerated from A, s, S'. We shall shortly prove that $\alpha \in S' \Downarrow_s^{\mathsf{A}\text{-solo}}$ is a recursive property. Exploiting this, the above formula can be simply decided by enumerating all the elements of $\mathrm{O}_s^{\mathsf{A}}(S')$, and testing whether they belong to $S' \Downarrow_s^{\mathsf{A}\text{-solo}}$.

We now show how to decide $\alpha \in S' \Downarrow_s^{\mathsf{A}\text{-solo}}$. This is a reachability problem in CO_2, once restricted to solo transitions. This restriction allows us to neglect all the other participants but A in S'. Further, in the solo computations of S', A can open only as much fresh sessions as the number of latent contracts already in S', which is trivial to compute given S'. More in general, starting from S', A can only interact with a bounded number of sessions: those already open, and those which will be created later.

We now focus on the process P in $S' = (\boldsymbol{u})(\mathsf{A}[P] \mid \cdots)$. W.l.o.g., we can assume P is a (delimited) parallel composition of $\mathrm{X}_{\mathrm{i}}(\boldsymbol{u})$, where each X_{i} is defined as $\sum_j \pi_j . P_j$, where (again) P_j is a delimited parallel composition of $\mathrm{X}_{\mathrm{i}}(\boldsymbol{u})$. Note that we only need a finite number of such X_{i}. Further, in the computations of S', the process of A can only be a parallel composition of (copies of) $\mathrm{X}_{\mathrm{i}}(\boldsymbol{u})$, where the components of $\boldsymbol{u}$ range over the finitely many session names discussed earlier, and (delimited) variables. Since only a finite number of variables can actually be instantiated with a session name, we focus on these and neglect the others in a non-deterministic way (roughly, we can follow the technique used in [5] to non-deterministically choose which variables to neglect).

Overall, the process of A is a multiset of finitely many copies of $\mathrm{X}_{\mathrm{i}}(\boldsymbol{u})$: hence, it can be represented by a Petri Net whose places correspond to each $\mathrm{X}_{\mathrm{i}}(\boldsymbol{u})$, and tokens account for their multiplicity. Further, when considering solo computations, the context of $\mathsf{A}[P]$ in S' is finite-state: it has finitely many sessions, each of with finitely many states, by hypothesis. Hence, the whole system can be represented by a Petri Net, whose transitions simulate the CO_2 semantics.

Concluding, to decide $\alpha \in S' \Downarrow_s^{\mathsf{A}\text{-solo}}$ it suffices to build the above Petri Net, and check whether a marking is reachable with at least one token in at least one of the places corresponding to $\mathrm{X}_{\mathrm{i}}(\ldots) = \mathtt{do}_s\, \alpha . P' + Q$. This is a submarking reachability problem, which is decidable [17]. □

5.4 Weak honesty is undecidable

Theorem 10. *$\mathcal{W}_{\mathbb{C}}$ is not recursive if $\mathbb{C} \supseteq \mathrm{ST}$.*

Proof. Easy adaptation of the proof of Theorem 7. Indeed, the process P_0 defined in that proof is honest when the Turing Machine does not halt (hence it is also

weakly honest by Theorem 2), and it is dishonest when it halts. The dishonesty is caused by P_0 stopping to interact in session x, which instead requires infinitely many actions to be performed. Even in honest contexts, P_0 would still violate its contract, hence it is not weakly honest. □

6 Related work and conclusions

We have presented a theory of honesty in session-based systems. This theory builds upon two basic notions, i.e. the classes $\boldsymbol{\mathcal{H}}$ (Definition 11) and $\boldsymbol{\mathcal{W}}$ (Definition 15) which represent two extremes in a hypothetical taxonomy of "good service behaviour". At the first extreme, there is the class $\boldsymbol{\mathcal{H}}$ of honest processes, which always manage to respect their contracts, in any possible context. Systems of honest agents guarantee some nice properties, e.g. deadlock-freedom (Theorem 6). However, this comes at a cost, as honest processes must either realize their contracts by operating independently on the respective sessions, or by exploiting "escape options" in contracts to overcome the dependence from the context. At the other extreme, we have a larger class $\boldsymbol{\mathcal{W}}$ of *weakly* honest processes, which make stronger assumptions about the context, but they do not enjoy deadlock-freedom, e.g. a system of weakly honest agents might get stuck.

Our investigation about honesty started in [10], where we first formalised this property, but in a less general setting than the one used in this paper. In particular, the contracts used in [10] are prefix-guarded τ-less CCS terms [13], provided with a semantics which forces the participants at the endpoints of a session to interact in turns. This is needed because the notion of honesty introduced in [10] is based on *culpability*: roughly, a participant is culpable in γ whenever she has enabled actions there. To be honest, one must be able to exculpate himself in each reachable state. The turn-based semantics of τ-less CCS contracts ensures that at each execution step only one participant is culpable, and that one can exculpate himself by doing the required actions. The turn-based semantics of contracts has a consequence on the process level: actions must be performed *asynchronously*. This means that a participant can fire $\mathtt{do}_s\,\alpha$ whenever α is enabled by the contract configuration at s. However, the requirement of having turn-based semantics of contracts has a downside: since many semantics of session types and other formalisms for contracts are synchronous, one has to establish the equivalence between the synchronous and the turn-based semantics. We did this in [7] for untimed session types, and in [2] for timed session types. The version of CO_2 defined in this paper overcomes these issues, by allowing for synchronous actions in contracts and in processes. This extension of CO_2 also makes it possible to use arbitrary LTSs as contracts. The other extension of CO_2 we have introduced in this paper is to allow processes to atomically advertise a set of contracts, so to have a session established only when *all* of them are matched with a compliant one. This enlarges the class of honest processes, making the calculus more expressive (see e.g. process $P_5{}'$ in Example 7).

The undecidability result presented in this paper (Theorem 7) subsumes the one in [10], where honesty was proved undecidable for processes using τ-less CCS

contracts. The new result is more general, because it applies to any instance of CO_2 with a contract model as least as expressive as session types.

Safe computable approximations of honesty (with session types as contracts) were proposed in [8,7], either in the form of type systems or model checking algorithms. Since the new version of CO_2 can deal with a more general model of contracts, it would be interesting to investigate computable approximation of honesty in this extended setting. We believe that most of the techniques introduced in [7] can be reused to this purpose: indeed, their correctness only relies on the fact that contracts admit a transition relation which abstracts from the context while preserving the concrete executions (as in Theorem 4.5 in [7]).

In the top-down approach to design a distributed application, one specifies its overall communication behaviour through a *choreography*, which validates some global properties of the application (e.g. safety, deadlock-freedom, *etc.*). To ensure that the application enjoys such properties, all the components forming the application have to be verified; this can be done e.g. by projecting the choreography to end-point views, against which these components are verified [26,21]. This approach assumes that designers control the whole application, e.g., they develop all the needed components. However, in many real-world scenarios several components are developed independently, without knowing at design time which other components they will be integrated with. In these scenarios, the compositional verification pursued by the top-down approach is not immediately applicable, because the choreography is usually unknown, and even if it were known, only a subset of the needed components is available for verification. The ideas pursued in this paper depart from the top-down approach, because designers can advertise contracts to discover the needed components (and so ours can be considered a *bottom-up* approach). Coherently, the main property we are interested in is *honesty*, which is a property of components, and not of global applications. Some works mixing top-down and bottom-up composition have been proposed [15,25,23,6] in the past few years.

The problem of ensuring safe interactions in session-based systems has been addressed to a wide extent in the literature [20,21,22]. In many of these approaches, deadlock-freedom in the presence of interleaved sessions is not directly implied by typeability. For instance, the two (dishonest) processes $P_5{}'$ and $Q_5{}'$ in examples 7 and 8. would typically be well-typed. However, the composition $\mathsf{A}[P_5{}'] \mid \mathsf{B}[Q_5{}']$ reaches a deadlock after fusing the sessions: in fact, A remains waiting on x (while not being ready at y), and B remains waiting on y (while not being ready at x). Multiple interleaved sessions has been tackled e.g. in [16,11,12,14]. To guarantee deadlock freedom, these approaches usually require that all the interactions on a session must end before another session can be used. For instance, the system $\mathsf{A}[P_5{}'] \mid \mathsf{B}[Q_5{}']$ would *not* be typeable in [12], coherently with the fact that it is not deadlock-free. The resulting notions seem however quite different from honesty, because we do not necessarily classify as dishonest processes with interleaved sessions. For instance, the process:

$$(x, y)\ \mathtt{tell}\ \{\downarrow_x \mathtt{a}\}.\ \mathtt{tell}\ \{\downarrow_y \overline{\mathtt{b}}\}.\ \left(\mathtt{do}_x\,\mathtt{a}.\,\mathtt{do}_y\,\overline{\mathtt{b}} + \mathtt{do}_y\,\overline{\mathtt{b}}.\,\mathtt{do}_x\,\mathtt{a}\right)$$

would not be typeable according to [12], but it is honest in our theory.

Acknowledgments. This work has been partially supported by Aut. Reg. of Sardinia grants L.R.7/2007 CRP-17285 (TRICS) and P.I.A. 2010 ("Social Glue"), by MIUR PRIN 2010-11 project "Security Horizons", and by EU COST Action IC1201 "Behavioural Types for Reliable Large-Scale Software Systems" (BETTY).

References

1. F. Barbanera and U. de'Liguoro. Two notions of sub-behaviour for session-based client/server systems. In *Proc. PPDP*, pages 155–164, 2010.
2. M. Bartoletti, T. Cimoli, M. Murgia, A. S. Podda, and L. Pompianu. Compliance and subtyping in timed session types. In *Proc. FORTE*, pages 161–177, 2015.
3. M. Bartoletti, T. Cimoli, M. Murgia, A. S. Podda, and L. Pompianu. A contract-oriented middleware, 2015. Submitted. Available at `http://co2.unica.it`.
4. M. Bartoletti, T. Cimoli, and R. Zunino. Compliance in behavioural contracts: a brief survey. In *Programming languages with applications to biology and security — Colloquium in honour of Pierpaolo Degano for his 65th birthday*, 2015.
5. M. Bartoletti, P. Degano, G. L. Ferrari, and R. Zunino. Model checking usage policies. *Mathematical Structures in Computer Science*, 25(3):710–763, 2015.
6. M. Bartoletti, J. Lange, A. Scalas, and R. Zunino. Choreographies in the wild. *Science of Computer Programming*, 2015.
7. M. Bartoletti, M. Murgia, A. Scalas, and R. Zunino. Verifiable abstractions for contract-oriented systems. Extended version of: Modelling and verifying contract-oriented systems in Maude, in Proc. WRLA 2014. Available at `http://tcs.unica.it/software/co2-maude/co2-verifiable-abstractions.pdf`.
8. M. Bartoletti, A. Scalas, E. Tuosto, and R. Zunino. Honesty by typing. In *Proc. FORTE*, pages 305–320, 2013.
9. M. Bartoletti, A. Scalas, and R. Zunino. A semantic deconstruction of session types. In *Proc. CONCUR*, pages 402–418, 2014.
10. M. Bartoletti, E. Tuosto, and R. Zunino. On the realizability of contracts in dishonest systems. In *Proc. COORDINATION*, pages 245–260, 2012.
11. L. Bettini, M. Coppo, L. D'Antoni, M. D. Luca, M. Dezani-Ciancaglini, and N. Yoshida. Global progress in dynamically interleaved multiparty sessions. In *Proc. CONCUR*, pages 418–433, 2008.
12. G. Castagna, M. Dezani-Ciancaglini, E. Giachino, and L. Padovani. Foundations of session types. In *Proc. PPDP*, 2009.
13. G. Castagna, N. Gesbert, and L. Padovani. A theory of contracts for Web services. *ACM TOPLAS*, 31(5):19:1–19:61, 2009.
14. M. Coppo, M. Dezani-Ciancaglini, L. Padovani, and N. Yoshida. Inference of global progress properties for dynamically interleaved multiparty sessions. In *Proc. COORDINATION*, pages 45–59, 2013.
15. P.-M. Deniélou and N. Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *Proc. ICALP*, pages 174–186, 2013.
16. M. Dezani-Ciancaglini, U. de'Liguoro, and N. Yoshida. On progress for structured communications. In *Proc. TGC*, pages 257–275, 2007.
17. J. Esparza. On the decidability of model checking for several μ-calculi and Petri nets. In *Proc. CAAP*, 1994.
18. D. Georgakopoulos and M. P. Papazoglou. *Service-oriented computing.* The MIT Press, 2008.

19. K. Honda. Types for dyadic interaction. In *Proc. CONCUR*, pages 509–523, 1993.
20. K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type disciplines for structured communication-based programming. In *Proc. ESOP*, pages 122–138, 1998.
21. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *Proc. POPL*, pages 273–284, 2008.
22. N. Kobayashi. A new type system for deadlock-free processes. In *Proc, CONCUR*, pages 233–247, 2006.
23. J. Lange and E. Tuosto. Synthesising choreographies from local session types. In *Proc. CONCUR*, pages 225–239, 2012.
24. R. Milner. *Communication and concurrency*. Prentice-Hall, Inc., 1989.
25. F. Montesi and N. Yoshida. Compositional choreographies. In *Proc. CONCUR*, pages 425–439, 2013.
26. W. M. P. van der Aalst, N. Lohmann, P. Massuthe, C. Stahl, and K. Wolf. Multiparty contracts: Agreeing and implementing interorganizational processes. *Comput. J.*, 53(1):90–106, 2010.