Honesty by Typing

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

[1] Università degli Studi di Cagliari, Italy
{bart,alceste.scalas}@unica.it
[2] University of Leicester, UK
emilio@mcs.le.ac.uk
[3] Università degli Studi di Trento and COSBI, Italy
roberto.zunino@unitn.it

Abstract. We propose a type system for a calculus of contracting processes. Processes may stipulate contracts, and then either behave honestly, by keeping the promises made, or not. Type safety guarantees that a typeable process is *honest* — that is, the process abides by the contract it has stipulated in all possible contexts, even those containing dishonest adversaries.

1 Introduction

Most approaches to the formal specification of concurrent systems typically assume that components behave *honestly*, in that they always adhere to some agreed specification (e.g., a behavioural type), under the assumption that the static behaviour safely over-approximates the dynamic one. We argue that this assumption is unrealistic in scenarios where competition prevails against cooperation. Indeed, in a competitive scenario components may act selfishly, and diverge from the agreed specification.

We envision a *contract-oriented computing* paradigm [2], for the design of distributed components which use *contracts* to discipline their interaction. In CO_2 [2], a process may advertise contracts; a session is established processes advertising *compliant* contracts, similarly to other session-centric calculi. This session dictates the actions needed to realise their contracts to processes. A distinguished feature of CO_2 is that processes are not supposed to respect their contracts, nor are they bound to them by an enforcing mechanism.

More realistically, *dishonest* processes may avoid to perform some actions they have promised in their contracts. This may happen either intentionally, e.g. a malicious process trying to swindle the system, or unintentionally, e.g. because of some implementation bug (possibly exploited by some adversary). In both cases, the infrastructure can determine which process has caused the violation, and adequately punish it. A crucial problem is how to guarantee that a process will behave honestly, in *all possible contexts*. If such guarantee can be given, the process is protected both against unintentional bugs, and against (apparently honest) adversaries which try to make it sanctioned.

A negative result in [3] is that determining if a process is honest is an undecidable problem for a relevant class of contracts (introduced in [10], and refined

D. Beyer and M. Boreale (Eds.): FMOODS/FORTE 2013, LNCS 7892, pp. 305–320, 2013.

in [11], for modelling WSDL and WSCL contracts). The problem is how to find a computable approximation of honesty, which implies the dynamic one.

Example. Let us consider an on-line store (participant A), which sells apples (a) and bottles of an expensive italian Brunello wine (b). Selling apples is quite easy: once an order is placed, A accepts it (with the feedback $\overline{\mathsf{ok}}$) and waits for a payment (pay) before shipping the goods ($\overline{\mathsf{ship\text{-}a}}$). However, if expensive bottles of Brunello are ordered, the store is entitled to either decline the order (by answering $\overline{\mathsf{no}}$), or accept it (and, as above, ship the item after the payment). Using external ($+$) and internal ($\oplus$) choice, the store contract can be modelled as

$$c \;=\; \mathsf{a}.\overline{\mathsf{ok}}.\,\mathsf{pay}.\overline{\mathsf{ship\text{-}a}} \;\;+\;\; \mathsf{b}.\,(\overline{\mathsf{no}} \;\oplus\; \overline{\mathsf{ok}}.\mathsf{pay}.\overline{\mathsf{ship\text{-}b}})$$

External choice requires the other party to decide how to drive the contract evolution; in this case, the customer chooses between apples a and bottles b. Internal choice, instead, allows the advertising party to choose; in this case, the store selects either $\overline{\mathsf{ok}}$ or $\overline{\mathsf{no}}$.

Intuitively, contract compliance hinges on the duality between internal and external choices. Hence, to sell its goods, the store needs to find an agreement with a participant advertising a contract *compliant* with its contract c, such as:

$$d \;=\; \overline{\mathsf{b}}\,.\,(\mathsf{ok}.\overline{\mathsf{pay}}.\mathsf{ship\text{-}b} \;+\; \mathsf{no})$$

A buyer B advertising d wants to buy Brunello: she promises to select $\overline{\mathsf{b}}$ (dual of b in the store contract), and then offers an external choice that lets the store choose between ok or no; in the first case, she promises to pay and wait for shipment.

In CO_2, the behaviour of each participant is a *process* capable of advertising contracts and executing the actions required to honour them. For instance, the store A can advertise its contract c by firing the prefix $\mathsf{tell}_{\mathsf{K}}{\downarrow_x}\, c$, where the index x in $\downarrow_x c$ is the name of a *channel* of A, and K is the name of an external broker, whom the contract is being advertised to. We shall not specify the behaviour of K, and just assume that when it finds a contract compliant with c, a session is established between A and another participant, say B. Technically, x is replaced with a fresh session name s. Participants A and B will then use such session to perform the actions required by their contracts.

A possible specification of the store A is e.g.:

$$\begin{aligned} P_M \;&=\; (x)\,(\mathsf{tell}_{\mathsf{K}}{\downarrow_x}\; c\,.\,(\mathsf{do}_x\,\mathsf{a}\,.\,X_M(x) \;+\; \mathsf{do}_x\,\mathsf{b}\,.\,X_M(x))) \\ X_M(x) \;&\stackrel{\text{def}}{=}\; \mathsf{do}_x\,\overline{\mathsf{ok}}\,.\,\mathsf{do}_x\,\mathsf{pay}\,.\,\mathsf{ask}_x\,\overline{\mathsf{ship\text{-}a}}?\,.\,\mathsf{do}_x\,\overline{\mathsf{ship\text{-}a}} \end{aligned}$$

Here, A creates a private channel x, and advertises the contract c. Once a session starts, P_M can accept an order for a or b on x. This is modelled by the guards $\mathsf{do}_x\,\mathsf{a}$ and $\mathsf{do}_x\,\mathsf{b}$ of the choice operator $+$ (not to be confused with $+$ of contracts). In both cases, the process $X_M(x)$ is invoked. There, A accepts the transaction with ok, and waits for payment. Then A checks whether the contract requires to ship apples: if the query $\mathsf{ask}_x\,\overline{\mathsf{ship\text{-}a}}?$ passes, the goods are shipped. Otherwise, when the customer B orders Brunello, A maliciously gets stuck, and so B has paid for nothing. This store is *dishonest*: it does not respect its own contract c.

Consider now a non-malicious implementation of the store. Before accepting orders, the store requires an insurance to cover shipment damages. With the contract $c_p = \overline{\mathsf{payP}}\,.\,(\overline{\mathsf{cover}} \oplus \overline{\mathsf{cancel}})$, A promises to pay ($\overline{\mathsf{payP}}$) and then choose between getting the coverage, or cancelling the request. The new store process is:

$$\begin{aligned} P_N &= (x,y)\big(\mathsf{tell}_{\mathsf{C}} \downarrow_y c_p \,.\, \mathsf{do}_y\, \overline{\mathsf{payP}} \,.\, \mathsf{tell}_{\mathsf{K}} \downarrow_x c \,. \\ &\qquad \big(\mathsf{do}_x\, \mathsf{a} \,.\, \mathsf{do}_x\, \overline{\mathsf{ok}} \,.\, X_N(x) + \mathsf{do}_x\, \mathsf{b} \,.\, Y_N(x,y)\big)\big) \\ X_N(x) &\stackrel{\text{def}}{=} \mathsf{do}_x\, \mathsf{pay} \,.\, \big(\mathsf{ask}_x\, \overline{\mathsf{ship\text{-}a}}? \,.\, \mathsf{do}_x\, \overline{\mathsf{ship\text{-}a}} + \mathsf{ask}_x\, \overline{\mathsf{ship\text{-}b}}? \,.\, \mathsf{do}_x\, \overline{\mathsf{ship\text{-}b}}\big) \\ Y_N(x,y) &\stackrel{\text{def}}{=} \mathsf{do}_y\, \overline{\mathsf{cover}} \,.\, \big(\mathsf{do}_x\, \overline{\mathsf{ok}} \,.\, X_N(x) \;+\; \tau \,.\, \mathsf{do}_x\, \overline{\mathsf{no}}\big) \end{aligned}$$

The store A first requests an insurance by advertising c_p to an insurance company (say C); once C agrees, A pays the premium (on channel y), and then advertises c; once an agreement with a customer B is reached, A waits for a or b orders. If apples are requested, A acknowledges ($\overline{\mathsf{ok}}$) and invokes $X_N(x)$; there, A waits for payment, checks which good has to be shipped, and actually ships it. Otherwise, if Brunello is requested, $Y_N(x,y)$ is invoked: there, A requests the insurance coverage paid in advance; then, either the order is accepted and $X_N(x)$ is invoked for payment and shipment (as above), or the transaction is declined after an internal action τ (e.g. a wake up after a timeout).

This implementation is not malicious as the first attempt: it is keen to ship the goods, but it is not honest either due to the interaction between A and the insurance company. If C does not execute cover, A gets stuck on $\mathsf{do}_y\, \overline{\mathsf{cover}}$, unable to honour c by providing the expected $\overline{\mathsf{ok}}/\overline{\mathsf{no}}$. Furthermore, A is dishonest w.r.t. c_p: the premium is paid in advance, but A may never perform $\mathsf{do}_y\, \overline{\mathsf{cover}}$ nor $\mathsf{do}_y\, \overline{\mathsf{cancel}}$ — e.g. if no agreement on c is found, or if the customer B is stuck, or if B simply chooses to buy apples. Thus, due to implementation naïveties, A may be blamed due to the unexpected (or malicious) behaviour of other participants.

Contributions. Our main contribution is a type discipline for statically ensuring when a CO_2 process is *honest*. The need for a static approximation is due to the fact that honesty is an undecidable property [3]. Our type system associates behavioural types to process channels. Checking honesty on these abstractions is decidable (Th. 1). We establish subject reduction (Th. 2) and progress (Th. 3), which are then used to prove type safety: typeable processes are honest (Th. 4).

Due to space constraints, we publish the proofs and further examples in a separate report [1].

2 A Calculus of Contracting Processes

The calculus CO_2 is parametric on the contract model, i.e. on the language of contracts. Contract models for CO_2 have been defined based e.g. on formulae of Propositional Contract Logic [4], and on CCS processes [2]. In this paper we focus on the two-party contracts of [11], as in [3]. Due to space limits, here we only give the main definitions, and refer the reader to [1] for the full details.

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

assume a distinguished atom e, modelling a successfully terminated participant, such that $\mathsf{e} = \bar{\mathsf{e}}$.

A *unilateral contract* of [11] is a CCS-like process c which models the promised behaviour of a single participant. An internal sum $c = \bigoplus_{i\in\mathcal{I}} \mathsf{a}_i \,;\, c_i$ requires a participant (say, A) to choose one of the actions a_i, do it, and then behave according to its continuation c_i. An external sum $c = \sum_{i\in\mathcal{I}} \mathsf{a}_i \,.\, c_i$ requires A to offer a choice among all the branches; if a_i is chosen by the other participant, then A must continue according to c_i. Finally, $c = \mathit{ready}\ \mathsf{a}.c'$ models an obligation of A to do a, and then to proceed as c'. We denote (guarded) recursive contracts with $\mathit{rec}\ X\,.\,c$, and we let $E = \mathit{rec}\ X\,.\,\mathsf{e}\,;\,X$.

A *bilateral contract* $\gamma = \mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says}\ d$ combines the contracts of two participants. Its behaviour is given in [3] as a LTS. Its main rules regulate the interaction between the internal choices of A, and the external choices of B:

$$\mathsf{A}\ \mathit{says}\ (\bar{\mathsf{a}}\,;\,c \oplus c') \mid \mathsf{B}\ \mathit{says}\ (\mathsf{a}\,.\,d + d') \xrightarrow{\mathsf{A}\ \mathit{says}\ \mathsf{a}} \mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says}\ \mathit{ready}\ \mathsf{a}.d \quad (1)$$

$$\mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says}\ \mathit{ready}\ \mathsf{a}.\ d \xrightarrow{\mathsf{B}\ \mathit{says}\ \mathsf{a}} \mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says}\ d \quad (2)$$

By (1), if A chooses branch $\bar{\mathsf{a}}$ in her internal sum, then B is committed to the branch a in his external sum. We mark the selected branch with *ready* a, and discard the others. This enables (2) where B takes the marked branch. When an internal choice is not matched by an action in the external choice of the partner, the contract gets stuck. Intuitively, two contracts c, d are *compliant*, $c \bowtie d$ in symbols, when this situation cannot occur. The formal definition of compliance builds upon that of *ready sets*. For a contract c, the ready sets $RS(c)$ are:

$$\begin{array}{ll} \{\{\mathit{ready}\ \mathsf{a}\}\}, & \text{if } c = \mathit{ready}\ \mathsf{a}.c' \\ \{\{\mathsf{a}_i\} \mid i \in I\}, & \text{if } c = \bigoplus_{i\in I} \mathsf{a}_i\,;\,c_i \text{ and } I \neq \emptyset \end{array} \qquad \begin{array}{ll} RS(c'), & \text{if } c = \mathit{rec}\ X\,.\,c' \\ \{\{\mathsf{a}_i \mid i \in I\}\}, & \text{if } c = \sum_{i\in I} \mathsf{a}_i\,.\,c_i \end{array}$$

Then, the relation $\bowtie$ on contracts is the largest relation preserved by contract transitions such that, whenever $c \bowtie d$,

$$\forall \mathcal{X} \in RS(c), \mathcal{Y} \in RS(d).\ \big(\{\bar{\mathsf{a}} \mid \mathsf{a} \in \mathcal{X}\} \cap \mathcal{Y} \neq \emptyset \ \text{ or } \ \exists \mathsf{a}.\ \mathit{ready}\ \mathsf{a} \in (\mathcal{X} \cup \mathcal{Y}) \setminus (\mathcal{X} \cap \mathcal{Y})\big)$$

We now briefly review CO_2. Let $\mathcal{V}$ and $\mathcal{N}$ be disjoint sets of, respectively, *session variables* $x, y, \ldots$ and *session names* $s, t, \ldots$. Let $u, v, \ldots$ range over $\mathcal{V} \cup \mathcal{N}$. *Systems* S, *processes* P, *prefixes* π, and *latent contracts* K are defined below.

Definition 1 (CO_2 syntax). *The syntax of CO_2 is given by:*

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

Processes specify the behaviour of participants. A process can be a prefix-guarded finite sum $\sum_i \pi_i.P_i$, a parallel composition $P \mid Q$, a delimited process $(\vec{u})P$, or a constant $X(\vec{u})$. We write $\mathbf{0}$ for $\sum_\emptyset P$, and $\pi_1.Q_1 + P$ for $\sum_{i\in I\cup\{1\}} \pi_i.Q_i$ provided

that $P = \sum_{i\in I} \pi_i.Q_i$ and $1 \notin I$. We omit trailing occurrences of $\mathbf{0}$. We stipulate that each X has a unique defining equation $X(u_1,\ldots,u_j) \stackrel{\text{def}}{=} P$ s.t. $\text{fv}(P) \subseteq \{u_1,\ldots,u_j\} \subseteq \mathcal{V}$, and each constant occurring in P is prefix-guarded.

Prefixes include the silent action τ, contract advertisement $\mathsf{tell}_\mathsf{A} \downarrow_u c$, contract stipulation fuse, action execution $\mathsf{do}_u\,\mathsf{a}$, and contract query $\mathsf{ask}_u\,\phi$. In each prefix $\pi \neq \tau$, the identifier u refers to the target session involved in the execution of π. As in [3], we leave the syntax of *observables* ϕ unspecified.

A *latent contract* $\downarrow_x \mathsf{A}$ *says* c represents a contract c advertised by A but not stipulated yet. The variable x will be instantiated to a fresh session name upon stipulation. K simply stands for the parallel composition of latent contracts.

A system is composed of participants $\mathsf{A}[P]$, sessions $s[\gamma]$, sets of latent contracts advertised to A (denoted by $\mathsf{A}[K]$), and delimited systems $(\vec{u})S$. Delimitation $(\vec{u})$ binds session variables and names, both in processes and systems. Free variables and names are defined as usual, and denoted by $\text{fv}(\_)$ and $\text{fn}(\_)$. A system/process is *closed* when it has no free variables. Each participant may have at most one process, i.e. we forbid systems of the form $\mathsf{A}[P] \mid \mathsf{A}[Q]$. We say that S is A-*free* when it does not contain the participant $\mathsf{A}[P]$, nor latent contracts of A, nor sessions with A's contracts. Note that sessions cannot contain latent contracts.

The semantics of CO_2 is formalised by a reduction relation on systems (Def. 2). This relies on a standard structural congruence relation — of which we just point out that $(\vec{u})\mathsf{A}[(\vec{v})P] \equiv (\vec{u}\vec{v})\mathsf{A}[P]$ allows to move delimitations between systems and processes, while $\mathsf{A}[K] \mid \mathsf{A}[K'] \equiv \mathsf{A}[K \mid K']$ allows A to collect latent contracts.

In order to define honesty in § 3, here we decorate transitions with labels, by writing $\xrightarrow{\mathsf{A}:\pi,\sigma}$ for a reduction where participant A fires prefix π. Also, σ is a substitution which accounts for the instantiation of session variables upon a fuse.

Definition 2 (CO_2 semantics). *The relation* $\xrightarrow{\mathsf{A}:\pi,\sigma}$ *between systems (considered up-to structural congruence* $\equiv$*) is the smallest relation closed under the rules of Fig. 1. The relation* $K \rhd^\sigma \gamma$ *holds iff* (*i*) K *has the form* $\downarrow_x \mathsf{A}$ *says* $c \mid \downarrow_y \mathsf{B}$ *says* d, (*ii*) $c \bowtie d$, (*iii*) $\gamma = \mathsf{A}$ *says* $c \mid \mathsf{B}$ *says* d, *and* (*iv*) $\sigma = \{{}^{s}/_{x,y}\}$ *maps* $x, y \in \mathcal{V}$ *to* $s \in \mathcal{N}$. *The substitution* $\sigma_{\neq u}$ *in rule* [DEL2] *is defined as* $\sigma(v)$ *for all* $v \neq u$, *and it is undefined on* u.

The rules in Fig. 1 are a minor variation of those presented in [3]. Their intuitive meaning is sketched in the introductory example (Sec. 1): [TELL] advertises a contract c, [FUSE] creates a new session s upon contractual compliance, [DO] performs a contractual action, [ASK] blocks until session s satisfies observable ϕ.

Example 1. A possible execution of $S = \mathsf{A}[(x)X(x)] \mid \mathsf{B}[(y)Y(y)] \mid \mathsf{C}[\mathsf{fuse}]$, where $X(x) \stackrel{\text{def}}{=} \mathsf{tell}_\mathsf{C} \downarrow_x (\mathsf{a}\,;\, E)\,.\,\mathsf{do}_x\,\mathsf{a}$ and $Y(y) \stackrel{\text{def}}{=} \mathsf{tell}_\mathsf{C} \downarrow_y (\overline{\mathsf{a}}\,.\,E)\,.\,\mathsf{do}_y\,\bar{\mathsf{a}}$, is:

$$
\begin{aligned}
S &\xrightarrow{\mathsf{B}:\mathsf{tell}_\mathsf{C}\downarrow_y \bar{\mathsf{a}},\emptyset} \mathsf{A}[(x)X(x)] \mid \mathsf{C}[\mathsf{fuse}] \mid (y)\,(\mathsf{B}[\mathsf{do}_y\,\bar{\mathsf{a}}] \mid \mathsf{C}[\downarrow_y \mathsf{B}\ \mathit{says}\ \bar{\mathsf{a}}\,.\,E]) \\
&\xrightarrow{\mathsf{A}:\mathsf{tell}_\mathsf{C}\downarrow_x \mathsf{a},\emptyset} (x)(\mathsf{A}[\mathsf{do}_x\,\mathsf{a}] \mid (y)(\mathsf{B}[\mathsf{do}_y\,\bar{\mathsf{a}}] \mid \mathsf{C}[\mathsf{fuse}] \mid \mathsf{C}[\downarrow_x \mathsf{A}\,\mathit{says}\,\mathsf{a};E \mid \downarrow_y \mathsf{B}\,\mathit{says}\,\bar{\mathsf{a}}.E])) \\
&\xrightarrow{\mathsf{C}:\mathsf{fuse},\emptyset} (s)\,(\mathsf{A}[\mathsf{do}_s\,\mathsf{a}] \mid (y)\,(\mathsf{B}[\mathsf{do}_s\,\bar{\mathsf{a}}] \mid \mathsf{C}[\mathbf{0}] \mid s[\mathsf{A}\ \mathit{says}\ \mathsf{a};E \mid \mathsf{B}\ \mathit{says}\ \bar{\mathsf{a}}.E])) \\
&\xrightarrow{\mathsf{A}:\mathsf{do}_s\,\mathsf{a},\emptyset} (s)\,(\mathsf{A}[\mathbf{0}] \mid \mathsf{B}[\mathsf{do}_s\,\bar{\mathsf{a}}] \mid s[\mathsf{A}\ \mathit{says}\ E \mid \mathsf{B}\ \mathit{says\ ready}\ \bar{\mathsf{a}}\,.\,E])
\end{aligned}
$$

$$\mathsf{A}[\tau.P + P' \mid Q] \xrightarrow{\mathsf{A} : \tau, \emptyset} \mathsf{A}[P \mid Q] \quad \text{[Tau]} \qquad \frac{S \xrightarrow{\mathsf{A} : \pi, \sigma} S' \quad \operatorname{ran}\sigma \cap \operatorname{fn}(S'') = \emptyset}{S \mid S'' \xrightarrow{\mathsf{A} : \pi, \sigma} S' \mid S''\sigma} \quad \text{[Par]}$$

$$\mathsf{A}[\mathsf{tell}_\mathsf{B} \downarrow_u c.P + P' \mid Q] \xrightarrow{\mathsf{A} : \mathsf{tell}_\mathsf{B} \downarrow_u c, \emptyset} \mathsf{A}[P \mid Q] \mid \mathsf{B}[\downarrow_u \mathsf{A}\ \mathit{says}\ c] \quad \text{[Tell]}$$

$$\frac{K \rhd^\sigma \gamma \qquad \operatorname{ran}\sigma = \{s\} \qquad s \text{ fresh}}{\mathsf{A}[\mathsf{fuse}.P + P' \mid Q] \mid \mathsf{A}[K] \xrightarrow{\mathsf{A} : \mathsf{fuse}, \sigma} \mathsf{A}[P \mid Q]\sigma \mid s[\gamma]} \quad \text{[Fuse]}$$

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

$$\frac{\gamma \vdash \phi}{\mathsf{A}[\mathsf{ask}_s\ \phi.P + P' \mid Q] \mid s[\gamma] \xrightarrow{\mathsf{A} : \mathsf{ask}_s\ \phi, \emptyset} \mathsf{A}[P \mid Q] \mid s[\gamma]} \quad \text{[Ask]} \qquad \frac{S \xrightarrow{\mathsf{A} : \pi, \{s/x\}} S'}{(x)S \xrightarrow{\mathsf{A} : \pi, \emptyset} (s)S'} \quad \text{[Del1]}$$

$$\frac{S \xrightarrow{\mathsf{A} : \pi, \sigma} S' \quad u \notin \operatorname{ran}\sigma \quad \sigma_{\neq u} \neq \emptyset}{(u)S \xrightarrow{\mathsf{A} : \pi, \sigma_{\neq u}} (u)S'} \quad \text{[Del2]} \qquad \frac{X(\vec{u}) \stackrel{\text{def}}{=} P \qquad \mathsf{A}[P\{^{\vec{v}}/_{\vec{u}}\} \mid Q] \mid S \xrightarrow{\mathsf{A} : \pi, \sigma} S'}{\mathsf{A}[X(\vec{v}) \mid Q] \mid S \xrightarrow{\mathsf{A} : \pi, \sigma} S'} \quad \text{[Def]}$$

Fig. 1. Reduction semantics of CO_2

3 On Honesty

A participant A is honest when she *realizes* every contract she advertises, in every session she may be engaged in. If a system S contains a session s with a contract c advertised by A, such as $\mathsf{A}[P] \mid s[\mathsf{A}\ \mathit{says}\ c \mid \cdots] \mid \cdots$, then A must realize c, even in a system populated by adversaries who play to cheat her. To realize c, A must be "ready" to behave according to c. For instance, if $\mathsf{A}[P]$ has advertised a contract c with an internal choice $c_i = \mathsf{a} \oplus \mathsf{b}$, then P must be ready to do *at least one* of the actions a, b. Instead, if c is an external choice $c_e = \mathsf{a} + \mathsf{b}$, then P must be ready to do *both* the actions a and b.

Realizability requires the *readiness* property to be preserved by all transitions of S. In other words, in any reduct of S containing a reduct P' of P and a reduct c' of c, the process P' must be ready for c'. To formalise when "P is ready for c" we inspect the ready sets $RS(c)$, which reveal whether c is exposing an internal or an external choice. At the process level, we consider the reachable actions in P.

Example 2. Let $c_i = \mathsf{a} \oplus \mathsf{b}$, and let $c_e = \mathsf{a} + \mathsf{b}$. Then, $RS(c_i) = \{\{\mathsf{a}\}, \{\mathsf{b}\}\}$, and $RS(c_e) = \{\{\mathsf{a}, \mathsf{b}\}\}$. Consider now the following processes:

- $P_0 = \mathsf{do}_s\ \mathsf{a}$ is ready for c_i, because $\{\mathsf{a}\} \in RS(c_i)$ and $\mathsf{do}_s\ \mathsf{a}$ is enabled in P_0. Instead, P_0 is *not* ready for c_e, since the ready set $\{\mathsf{a}, \mathsf{b}\}$ of c_e also contains b, which is not enabled in P_0.
- $P_1 = \mathsf{do}_s\ \mathsf{a} + \mathsf{do}_s\ \mathsf{b} + \mathsf{do}_s\ \mathsf{z}$ is ready for both c_i and c_e. Indeed, P_1 enables $\mathsf{do}_s\ \mathsf{a}$ and $\mathsf{do}_s\ \mathsf{b}$, thus covering the ready sets of c_i and c_e. The branch $\mathsf{do}_s\ \mathsf{z}$ is immaterial: rule [Do] blocks actions not expected by the contract.
- $P_2 = \tau.\mathsf{do}_s\ \mathsf{a} + \tau.\mathsf{do}_s\ \mathsf{b}$ is ready for c_i, because whatever branch is taken by P_2, it leads to an unguarded action which covers one of the ready sets in

c_i. Instead, P_2 is *not* ready for c_e, because after one of the two branches is chosen, one of the two actions expected by c_e is no longer available.
- $P_3 = \mathsf{do}_t\,\mathsf{w}.\mathsf{do}_s\,\mathsf{a} + \mathsf{do}_t\,\mathsf{z}.\mathsf{do}_s\,\mathsf{b}$ is a bit more complex than the above cases. Readiness w.r.t. c_i depends on the context. If the context eventually enables one of the do_t, then either $\mathsf{do}_s\,\mathsf{a}$ or $\mathsf{do}_s\,\mathsf{b}$ will be enabled, hence P_3 is ready for c_i. Otherwise, P_3 is stuck, hence it is not ready for c_i. Notice that P_3 is not ready for c_e, regardless of the context.

To formalise readiness, we start by defining the set $RD_u^{\mathsf{A}}(S)$ (for "Ready Do"), which collects all the atoms with an unguarded action do_u of A in a system S.

Definition 3 (Ready do). *For all S, A and u, we define the set $RD_u^{\mathsf{A}}(S)$ as:*

$$RD_u^{\mathsf{A}}(S) = \{\mathsf{a} \mid \exists \vec{v}, P, P', Q, S' \;.\; S \equiv (\vec{v})\,(\mathsf{A}[\mathsf{do}_u\,\mathsf{a}.P + P' \mid Q] \mid S') \;\wedge\; u \notin \vec{v}\}$$

As seen for P_2 and P_3, readiness may also hold when the actions expected in the contract ready sets are not immediately enabled in the process. To check if $\mathsf{A}[P]$ is ready for s (in a system S), we need to consider all actions which (1) are exposed in P after some steps, taken by P itself or by the context, and (2) are not preceded by other do_s performed by A. These actions form the set $WRD_s^{\mathsf{A}}(S)$.

Definition 4 (Weak ready do). *We define the set of atoms $WRD_u^{\mathsf{A}}(S)$ as:*

$$WRD_u^{\mathsf{A}}(S) \;=\; \{\mathsf{a} \mid \exists S' : S \xrightarrow{\neq(\mathsf{A}:\,\mathsf{do}_u)}{}^{*}\; S' \text{ and } \mathsf{a} \in RD_u^{\mathsf{A}}(S')\}$$

where $S \xrightarrow{\neq(\mathsf{A}:\,\mathsf{do}_u)} S'$ *iff* $\exists \mathsf{B}, \pi, \sigma.\;\; S \xrightarrow{\mathsf{B}:\,\pi,\sigma} S' \;\wedge\; (\mathsf{B} \neq \mathsf{A} \;\vee\; \forall \mathsf{a}.\; \pi \neq \mathsf{do}_u\,\mathsf{a})$.

Example 3. For $S = \mathsf{A}[\mathsf{do}_x\,\bar{\mathsf{a}}\,.\,\mathsf{do}_y\,\mathsf{b} + \tau\,.\,\mathsf{do}_y\,\mathsf{a}\,.\,\mathsf{do}_y\,\mathsf{c} \mid (x)\,\mathsf{do}_x\,\bar{\mathsf{b}}]$, we have:

$$WRD_x^{\mathsf{A}}(S) = \{\bar{\mathsf{a}}\} = RD_x^{\mathsf{A}}(S) \qquad WRD_y^{\mathsf{A}}(S) = \{\mathsf{a}\} \supseteq RD_y^{\mathsf{A}}(S) = \emptyset$$

On channel y, the action a is weakly reachable through its τ prefix. Action b is *not* weakly reachable, because guarded by a stuck do_x. Action c is *not* weakly reachable as well, because preceded by another do on the same channel.

Example 4. Let P_3 as in Ex. 2, and consider the following system:

$$\begin{aligned} S = \mathsf{A}[P_3] \;\mid\; \mathsf{B}[\tau\,.\,\mathsf{do}_s\,\bar{\mathsf{a}}\,.\,\mathsf{do}_s\,\bar{\mathsf{b}}] \;\mid\; \mathsf{C}[\mathsf{do}_t\,\overline{\mathsf{w}} + \mathsf{do}_t\,\bar{\mathsf{z}} + \tau] \\ \mid\; s[\mathsf{A}\ \mathit{says}\ \mathsf{a} + \mathsf{b} \mid \mathsf{B}\ \mathit{says}\ \bar{\mathsf{a}} \oplus \bar{\mathsf{b}}] \;\mid\; t[\mathsf{A}\ \mathit{says}\ \mathsf{w} + \mathsf{z} \mid \mathsf{C}\ \mathit{says}\ \overline{\mathsf{w}} \oplus \bar{\mathsf{z}}] \end{aligned}$$

In session t, A is immediately ready to do either w or z, so her ready do set coincides with her weak ready do set in t. The same for C, with the dual atoms $\overline{\mathsf{w}}$ and $\bar{\mathsf{z}}$. Thus:

$$WRD_t^{\mathsf{A}}(S) \;=\; RD_t^{\mathsf{A}}(S) \;=\; \{\mathsf{w}, \mathsf{z}\} \qquad WRD_t^{\mathsf{C}}(S) \;=\; RD_t^{\mathsf{C}}(S) \;=\; \{\overline{\mathsf{w}}, \bar{\mathsf{z}}\}$$

In session s, the ready do sets of both A and B are empty, because their actions are not immediately enabled. Before they can be reached, the whole system S must

first reduce, either with the contribution of C on session t (in the case of A), or through a τ action (in the case of B). These reductions fall within the definition of their *weak* ready do sets, which are accordingly non-empty.

$$WRD_s^{\mathsf{B}}(S) = \{\bar{\mathsf{a}}\} \supseteq RD_s^{\mathsf{B}}(S) = \emptyset \qquad WRD_s^{\mathsf{A}}(S) = \{\mathsf{a}, \mathsf{b}\} \supseteq RD_s^{\mathsf{A}}(S) = \emptyset$$

Notice that $\bar{\mathsf{b}} \notin WRD_s^{\mathsf{B}}(S)$: in fact, $\bar{\mathsf{b}}$ is guarded by $\mathsf{do}_s\, \bar{\mathsf{a}}$. Also, if C chooses to perform τ, then the actions in $WRD_s^{\mathsf{A}}(S)$ would not be reached. Indeed, Def. 4 requires that each element in the set becomes reachable at the end of a suitable reduction trace — but it does not prevent S from reducing along other paths.

A participant A is ready in a system S with a session $s[\mathsf{A}\ \mathit{says}\ c \mid \cdots]$ iff A is (weakly) ready to do all the actions in some ready set of c. Note that A is vacuously ready in systems not containing sessions with A's contracts.

Definition 5 (Honesty). *We say that* A *is ready in* S *iff, whenever* $S \equiv (\vec{u})S'$ *for some* $\vec{u}$ *and* $S' = s[\mathsf{A}\ \mathit{says}\ c \mid \cdots] \mid S_0$,

$$\exists \mathcal{X} \in RS(c)\,.\, \forall \mathsf{a} \neq \mathsf{e}\,.\, \big(\mathsf{a} \in \mathcal{X} \vee \mathit{ready}\, \mathsf{a} \in \mathcal{X} \implies \mathsf{a} \in WRD_s^{\mathsf{A}}(S')\big)$$

$\mathsf{A}[P]$ *is* honest *iff* $\forall$ A*-free* S *and* $\forall\, S'$ *such that* $\mathsf{A}[P] \mid S \to^* S'$, A *is ready in* S'.

The A-freeness requirement in Def. 5 is used just to rule out those systems already carrying stipulated or latent contracts of A outside $\mathsf{A}[P]$, e.g. $\mathsf{A}[P] \mid \mathsf{B}[\downarrow_x \mathsf{A}\ \mathit{says}\ \overline{\mathsf{pay}} \mid \cdots]$. In the absence of A-freeness, the system could trivially make $\mathsf{A}[P]$ dishonest.

Example 5. In the system below, A might look honest, although she is not.

$$\begin{array}{c} S \stackrel{\text{def}}{=} \mathsf{A}[(x,y)\ (P_A \mid \mathsf{fuse} \mid \mathsf{fuse})] \mid \mathsf{B}[P_B] \mid \mathsf{C}[P_C] \\ P_A \stackrel{\text{def}}{=} \mathsf{tell}_{\mathsf{A}}\,(\downarrow_x \mathsf{a}\,.\,E)\,.\,\mathsf{tell}_{\mathsf{A}}\,(\downarrow_y \mathsf{b}\,;\,E)\,.\,\mathsf{do}_x\,\mathsf{a}\,.\,\mathsf{do}_y\,\mathsf{b} \end{array}$$

$$P_B \stackrel{\text{def}}{=} (z)\ (\mathsf{tell}_{\mathsf{A}}\,(\downarrow_z \bar{\mathsf{b}}\,.\,E)\,.\,\mathsf{do}_z\,\bar{\mathsf{b}}) \qquad P_C \stackrel{\text{def}}{=} (w)\ (\mathsf{tell}_{\mathsf{A}}\,(\downarrow_w \bar{\mathsf{a}}\,;\,E)\,.\,\mathbf{0})$$

In fact, if we reduce S by performing all the tell and fuse actions, we obtain:

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

Here, S' cannot reduce further: A is stuck, waiting for $\bar{\mathsf{a}}$ from C, which (dishonestly) avoids to do the required internal choice. So, A is dishonest, because she does not perform the promised b. Formally, A is dishonest because $RS(\mathsf{b}\,;\,E) = \{\{\mathsf{b}\}\}$, but $\mathsf{b} \notin WRD_s^{\mathsf{A}}(S')$. Thus, A is not ready in S', hence not honest in S.

Our definition of honesty subsumes a *fair* scheduler, eventually allowing participants to fire persistently (weakly) enabled do actions. For instance, let $c = \mathsf{a} \oplus \mathsf{b}$, and let:

$$P \stackrel{\text{def}}{=} (x)\big(\mathsf{tell}_{\mathsf{A}} \downarrow_x c\,.\,\mathsf{fuse}\,.\,X(x)\big) \quad \text{where } X(x) \stackrel{\text{def}}{=} \tau\,.\,X(x) + \tau\,.\,\mathsf{do}_x\,\mathsf{a} + \tau\,.\,\mathsf{do}_x\,\mathsf{b}$$

Let $S = \mathsf{A}[P] \mid S_0$, and assume that the fuse in P passes. Under an unfair scheduler, A could always take the first branch in X, while neglecting the others. Intuitively, this would make A not respect her contract, which expects a or b. However, a fair scheduler will eventually choose one of the other branches.

4 A Type System for CO_2

We now introduce a type system for CO_2. The main result is *type safety* (established in Th. 4), which guarantees that typeable participants are honest.

The type of a process P is a function f, which maps each channel to a *channel type*. Channel types are behavioural types which essentially preserve the structure of P, while abstracting the actual prefixes and delimitations. Mainly, the prefixes of channel types distinguish between nonblocking and possibly blocking actions.

4.1 Channel Types

Channel types are Basic Parallel Processes (BPPs [14]) with standard semantics. Their prefixes can be atoms ($\mathsf{a}, \mathsf{b}, \ldots$), contract advertisement actions ($\langle c \rangle$), nonblocking (τ), possibly blocking ($\tau_?$), and conditional (τ_ϕ) silent actions.

Definition 6 (Channel types). Channel types T *and* prefixes α *are:*

$$T \ ::= \ \mathbf{0} \mid \alpha \,.\, T \mid T + T \mid T \mid T \mid \mathit{rec}\ X \,.\, T \mid X \qquad \alpha \ ::= \ \mathsf{a} \mid \tau \mid \tau_? \mid \tau_\phi \mid \langle c \rangle$$

Example 6. Let $P = \mathsf{tell}_\mathsf{B} \downarrow_x c_i \mid \big(\mathsf{tell}_\mathsf{B} \downarrow_y d \,.\, \mathsf{do}_x\, \bar{\mathsf{a}}\big)$, where $c_i = \bar{\mathsf{a}} \oplus \bar{\mathsf{b}}$, and d is immaterial. We anticipate that the channel types associated by our type system to P on channels x and y are, respectively, $T_x = \langle c_i \rangle \mid \tau.\bar{\mathsf{a}}$, and $T_y = \tau \mid \langle d \rangle.\tau_?$. Note that the advertisement of $\downarrow_x c_i$ is recorded in T_x, while that of $\downarrow_y d$ is abstracted there as a τ. Instead, the $\tau_?$ in T_y represents the fact that $\mathsf{do}_x\, \bar{\mathsf{a}}$ is not visible from channel y, and may potentially block.

The execution of CO_2 systems relies both on processes and contracts. Thus, we use an abstraction of the contract/process interplay to define abstract processes.

Definition 7 (Abstract processes). *An* abstract process *is a pair* (C, T) *or* (c, T)*, where* C *is a set of contracts,* c *is a contract, and* T *is a channel type.*

An abstract process (C, T) represents a process abstracted by T on some channel x, after the contracts in C have been *advertised*. Instead, (c, T) represents a process abstracted by T on channel x, after the contract c has been *stipulated*.

The semantics of abstract processes is given in Fig. 2. The set C grows when a channel type T in (C, T) performs a transition with label $\langle c \rangle$. After a contract $c \in C$ has been stipulated, the set is reduced to c. A label a models a $\mathsf{do}\,\mathsf{a}$ action performed by T, while rule *ctx* models an (unknown) action performed by the context. Further advertisements after contract stipulation are neglected.

The relation $\twoheadrightarrow_\sharp$ abstracts the contract semantics $\twoheadrightarrow$, by considering only the contract advertised by P (instead of the whole bilateral contract). We leave the relation $\twoheadrightarrow_\sharp$ unspecified (see [3] for a possible instantiation), and we just require that $\twoheadrightarrow_\sharp$ is decidable, and for all $\gamma = \mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says}\ d$ such that $c \bowtie d$,

$$\gamma \xrightarrow{\mathsf{A}\ says\ \mathsf{a}}\!\!\!\!\twoheadrightarrow \mathsf{A}\ \mathit{says}\ c' \mid \mathsf{B}\ \mathit{says}\ d' \quad \Longrightarrow \quad c \xrightarrow{\mathsf{a}}\!\!\!\!\twoheadrightarrow_\sharp c'$$
$$\gamma \xrightarrow{\mathsf{B}\ says\ \mathsf{b}}\!\!\!\!\twoheadrightarrow \mathsf{A}\ \mathit{says}\ c' \mid \mathsf{B}\ \mathit{says}\ d' \quad \Longrightarrow \quad c \xrightarrow{ctx}\!\!\!\!\twoheadrightarrow_\sharp c'$$

$$\frac{T \xrightarrow{\langle c \rangle} T'}{(C,T) \to (C \cup \{c\}, T')} \qquad \frac{T \xrightarrow{\langle d \rangle} T'}{(c,T) \to (c,T')} \qquad \frac{c \in C}{(C,T) \to (c,T)} \qquad \frac{c \xrightarrow{ctx}\!\!\!\!\rightarrow_\sharp c'}{(c,T) \to (c',T)}$$

$$\frac{T \xrightarrow{\alpha} T' \quad \alpha \in \{\tau, \tau_?, \tau_\phi\}}{(C,T) \to (C,T')} \qquad \frac{c \xrightarrow{\mathsf{a}}\!\!\!\!\rightarrow_\sharp c' \quad T \xrightarrow{\mathsf{a}} T'}{(c,T) \to (c',T')} \qquad \frac{T \xrightarrow{\alpha} T' \quad \alpha \in \{\tau, \tau_?, \tau_\phi\}}{(c,T) \to (c,T')}$$

$$\frac{T \xrightarrow{\mathsf{a}} T'}{T \stackrel{\mathsf{a}}{\Longrightarrow} T'} \qquad \frac{T \xrightarrow{\tau} T'' \stackrel{\mathsf{a}}{\Longrightarrow} T'}{T \stackrel{\mathsf{a}}{\Longrightarrow} T'} \qquad \frac{T \xrightarrow{\langle d \rangle} T'' \stackrel{\mathsf{a}}{\Longrightarrow} T'}{T \stackrel{\mathsf{a}}{\Longrightarrow} T'} \qquad \frac{T \xrightarrow{\tau_\phi} T'' \stackrel{\mathsf{a}}{\Longrightarrow} T' \quad c \vdash^{\mathsf{A}}_\sharp \phi}{T \stackrel{\mathsf{a}}{\Longrightarrow} T'}$$

Fig. 2. Abstract processes semantics and channel type semantics

We now introduce the abstract counterpart of the dynamic notion of honesty in § 3. We shall follow the path outlined for concrete processes: first we define when a channel type T is "ready for a contract", and then when T is honest.

Weak transitions abstract the weak ready do. They are defined in Fig. 2 as a labelled relation $\stackrel{\mathsf{a}}{\Longrightarrow}^{\mathsf{A}}_c$ (simply written as $\stackrel{\mathsf{a}}{\Longrightarrow}$ when unambiguous). The first two rules are standard: they just collapse the τ actions as usual. The third rule also collapses contract advertisement actions, which are nonblocking as well. Possibly blocking actions $\tau_?$ are *not* collapsed, while τ_ϕ (which abstract $\mathsf{ask}_u\,\phi$ prefixes) are dealt with the last rule: they abstract the CO_2 prefix $\mathsf{ask}_u\,\phi$, and they are collapsed only if such ask is nonblocking. The relation $\vdash^{\mathsf{A}}_\sharp$ safely (under-) approximates this condition. We leave $\vdash^{\mathsf{A}}_\sharp$ unspecified (just like $\vdash$ in § 2), and we only require that it respects the constraint in Def. 8 below. Unlike in the concrete case, the context is immaterial in determining weak transitions.

Definition 8 (Abstract observability). *We write $c \vdash^{\mathsf{A}}_\sharp \phi$ for any decidable relation satisfying: $c \vdash^{\mathsf{A}}_\sharp \phi \implies \forall \mathsf{B}\,.\,\forall d\,.\,(c \bowtie d \implies \mathsf{A}$ says $c \mid \mathsf{B}$ says $d \vdash \phi)$.*

Below we relate the weak transitions of a channel type with the ready sets of a contract, similarly to what we did in Def. 5 for CO_2 processes. Weak transitions under-approximate the weak ready do set. Thus, if an abstract process is honest then also the concrete one will be such (while the *vice versa* is not always true).

Honesty of abstract processes is defined similarly to Def. 5. In order to be honest, a process must keep itself (abstractly) ready upon transitions. Readiness must be checked against all the contracts that may be stipulated along the abstract process reductions.

Definition 9 (Abstract honesty). *We say that channel type T is* abstractly ready for contract c *iff* $\exists \mathcal{X} \in RS(c).\forall \mathsf{a} \neq \mathsf{e}.\big((\mathsf{a} \in \mathcal{X} \vee$ *ready* $\mathsf{a} \in \mathcal{X}) \implies T \stackrel{\mathsf{a}}{\Longrightarrow}\big)$. *An abstract process $(-,T)$ is honest iff, whenever $(-,T) \to^* (c,T')$, then T' is ready for c. A channel type T is* honest *iff $(\emptyset,T)$ is honest.*

Example 7. Let $T_x = \langle c_i \rangle \mid \tau\,.\,\mathsf{a}$, and recall the contract $c_i = \mathsf{a} \oplus \mathsf{b}$ from Ex. 2. To prove T_x is honest, we examine all the reducts of the abstract process $(\emptyset, T_x)$ to check for readiness. We have the following cases:

1. $(\emptyset, T_x)$. Nothing to check, because no contracts have been advertised yet.
2. $(\emptyset, \langle c_i \rangle \mid \mathsf{a})$. Similar to the previous case.
3. $(\{c_i\}, \tau \,.\, \mathsf{a})$. Nothing to check, since no contracts have been stipulated yet.
4. $(\{c_i\}, \mathsf{a})$. Similar to the previous case.
5. $(c_i, \tau \,.\, \mathsf{a})$. Here $\tau \,.\, \mathsf{a}$ is ready for c_i, as $\{\mathsf{a}\} \in RS(c_i) = \{\{\mathsf{a}\}, \{\mathsf{b}\}\}$ and $\tau \,.\, \mathsf{a} \stackrel{\mathsf{a}}{\Longrightarrow}$.
6. (c_i, a). We have that a is ready for c_i, similarly to the previous case.
7. $(E, \mathbf{0})$. We have that $\mathbf{0}$ is vacuously ready for E.

Th. 1 below establishes that checking the honesty of a channel type T is decidable. Indeed, abstract readiness and abstract dishonesty are reachability properties. Abstract processes are the product of a finite state system (C and c only admit finitely many states), and a BPP T. This product can be modelled as a Petri net. Decidability follows since reachability is decidable for Petri nets [17].

Theorem 1. *Abstract honesty is decidable.*

4.2 Process Types

A *CO_2 process type* associates session names/variables to channel types, thus abstracting the behaviour of a process on all channels. Here, we introduce a "dummy" channel $* \notin \mathcal{N} \cup \mathcal{V}$, for collecting type information about unused channels. Formally, a process type is a function f from $\mathcal{N} \cup \mathcal{V} \cup \{*\}$ to channel types.

Our type system abstracts concrete prefixes of CO_2 processes as actions of channel types. We observe the behaviour of a process P on each channel, say u. When P performs an action on one of its channels, say v, we have two cases:

$v \neq u$: we will only observe a silent action, either nonblocking (τ) or blocking ($\tau_?$), depending on the concrete prefix fired.

$v = u$: we may observe more information, depending on the concrete prefix.

For instance, if P advertises a contract c with a $\mathsf{tell} \downarrow_v c$, then the action $\langle c \rangle$ will be visible if $v = u$, while we shall just observe a τ if $v \neq u$ (because tell is nonblocking). Similarly, if P performs $\mathsf{do}_v \, \mathsf{a}$ we shall observe the action a if $v = u$ and $\tau_?$ if $v \neq u$ (because do is blocking). Finally, if P executes a query $\mathsf{ask}_u \, \phi$ we shall observe the conditional silent action τ_ϕ if $v = u$ and $\tau_?$ otherwise.

Definition 10 (Prefix abstraction). *For all $u \in \mathcal{N} \cup \mathcal{V} \cup \{*\}$, we define the mapping $[\cdot]_u$ from CO_2 prefixes to channel type prefixes as follows:*

$$[\tau]_u \;=\; \tau \qquad [\mathsf{fuse}]_u \;=\; \tau_? \qquad [\mathsf{tell}_{\mathsf{A}} \downarrow_v c]_u \;=\; \mathtt{if}\ v = u\ \mathtt{then}\ \langle c \rangle\ \mathtt{else}\ \tau$$

$$[\mathsf{do}_v \, \mathsf{a}]_u \;=\; \mathtt{if}\ v = u\ \mathtt{then}\ \mathsf{a}\ \mathtt{else}\ \tau_? \qquad [\mathsf{ask}_v \, \phi]_u \;=\; \mathtt{if}\ v = u\ \mathtt{then}\ \tau_\phi\ \mathtt{else}\ \tau_?$$

The typing judgments for processes have the form $\Gamma \vdash P \colon f$, where Γ is a typing environment, giving types to processes of the form $X(\vec{v})$.

Definition 11 (Typing environment). *A* typing environment *Γ is a partial function associating process types to constants $X(\vec{v})$.*

$$\frac{\Gamma \vdash P_i : f_i \quad \forall i \in I}{\Gamma \vdash \sum_{i\in I} \pi_i \,.\, P_i : \lambda u \,.\, \sum_{i\in I} [\pi_i]_u \,.\, f_i(u)}\ \text{[T-Sum]} \qquad \frac{\Gamma \vdash P : f \quad \Gamma \vdash Q : g}{\Gamma \vdash P \mid Q : \lambda u \,.\, f(u) \mid g(u)}\ \text{[T-Par]}$$

$$\frac{X(\vec{u}) \stackrel{\text{def}}{=} P \quad \Gamma\{^{f}/_{X(\vec{v})}\} \vdash P\{^{\vec{v}}/_{\vec{u}}\} : f}{\Gamma \vdash X(\vec{v}) : f}\ \text{[T-Def]} \qquad \frac{\Gamma(X(\vec{v})) = f}{\Gamma \vdash X(\vec{v}) : f}\ \text{[T-Var]}$$

$$\frac{\Gamma_{\neq u} \vdash P : f \quad f(u) \text{ honest}}{\Gamma \vdash (u)P : f\{^{f(*)}/_{u}\}}\ \text{[T-Del]} \quad \text{where } \Gamma_{\neq \vec{v}}(Y(\vec{w})) = \begin{cases} \Gamma(Y(\vec{w})) & \text{if } \vec{w} \cap \vec{v} = \emptyset \\ \text{undefined} & \text{otherwise} \end{cases}$$

Fig. 3. Typing rules for processes

In Fig. 3 we introduce the typing rules for CO_2 processes. Rule [T-Sum] abstracts the prefixes which guard the branches of a summation, according to Def. 10. The resulting process type is expressed through the usual λ-notation. The type of a parallel composition is the pointwise parallel composition of the component types (rule [T-Par]). Rules [T-Def] and [T-Var] are mostly standard. Rule [T-Var] retrieves the type of a process variable from the typing environment, which is populated by [T-Def]. The rule for typing delimitations ([T-Del]) is worth some comments. Assume that P is typed with f. Since u in not free $(u)P$, the actions on channel u must not be observable in the typing of $(u)P$. To do that, in the typing of $(u)P$ we discard the information on u, by replacing it with the typing information on the "dummy" channel $*$. However, since this might hide a dishonest behaviour on u, the rule also checks that $f(u)$ is honest. Moreover, if the environment Γ has typing information on channel u, this cannot be used while typing P. The typing environment $\Gamma_{\neq u}$, which discards the information on u, is used to this purpose.

Example 8. Recall process $P_2 = \tau.\mathsf{do}_s\ \mathsf{a} + \tau.\mathsf{do}_s\ \mathsf{b}$ from Ex. 2. Its typing derivation is:

$$\frac{\dfrac{}{\vdash \mathsf{do}_s\ \mathsf{a} : \lambda u \,.\, [\mathsf{do}_s\ \mathsf{a}]_u = f_1}\ \text{[T-Sum]} \quad \dfrac{}{\vdash \mathsf{do}_s\ \mathsf{b} : \lambda u \,.\, [\mathsf{do}_s\ \mathsf{b}]_u = f_2}\ \text{[T-Sum]}}{\vdash P_2 : f = \lambda u \,.\, [\tau]_u \,.\, f_1(u) + [\tau]_u \,.\, f_2(u)}\ \text{[T-Sum]}$$

We have $f(s) = \tau \,.\, \mathsf{a} + \tau \,.\, \mathsf{b}$, and for all $u \neq s$, $f(u) = f(*) = \tau \,.\, \tau_? + \tau \,.\, \tau_?$. In other words, the process type f performs some visible actions when observed at channel s, while remaining "silent" on other channels.

The type system assigns the same type (up-to structural congruence) to all non-free session names/variables, and to $*$; such type may only contain τ and $\tau_?$.

Lemma 1. *For all* P, $\vdash P : f \implies f(*)$ *only contains* τ *and* $\tau_?$ *actions.*

Lemma 2. $z \notin \text{fnv}(P) \ \wedge\ \Gamma \vdash P : f \implies f(z) = f(*)$

A process type f takes a transition on a CO_2 prefix π when all its points $f(u)$ agree to take a transition on the abstract prefix $[\pi]_u$.

Definition 12. *We write* $f \xrightarrow{\pi} f'$ *whenever* $\forall u \in \mathcal{N} \cup \mathcal{V} \cup \{*\} \,.\, f(u) \xrightarrow{[\pi]_u} f'(u)$.

Example 9. Recall $P_1 = \mathsf{do}_s\,\mathsf{a} + \mathsf{do}_s\,\mathsf{b} + \mathsf{do}_s\,\mathsf{z}$ from Ex. 2. Its typing is $\vdash P_1 \colon f = \lambda u\,.\,[\mathsf{do}_s\,\mathsf{a}]_u + [\mathsf{do}_s\,\mathsf{b}]_u + [\mathsf{do}_s\,\mathsf{z}]_u$. Let $f' = \lambda u\,.\,\mathbf{0}$. We have that $f \xrightarrow{\mathsf{do}_s\,\mathsf{a}} f'$, since:

- $[\mathsf{do}_s\,\mathsf{a}]_s = \mathsf{a}$ and $f(s) = \mathsf{a} + \mathsf{b} + \mathsf{z} \xrightarrow{\mathsf{a}} \mathbf{0} = f'(s)$;
- $\forall v \neq s\,.\,[\mathsf{do}_s\,\mathsf{a}]_v = \tau_?$ and $f(v) = \tau_? + \tau_? + \tau_? \xrightarrow{\tau_?} \mathbf{0} = f'(v)$.

Note that, in this case, we also have $f \xrightarrow{\mathsf{do}_s\,\mathsf{b}} f'$ and $f \xrightarrow{\mathsf{do}_s\,\mathsf{z}} f'$.

If f is the type of some CO_2 process (i.e., f is *inhabited*), and a single point $f(u)$ takes an abstract transition, then the whole f can take a transition.

Lemma 3. $f(u) \xrightarrow{\alpha} T' \wedge f\,\mathit{inhabited} \implies \exists \pi, f'\,.\,[\pi]_u = \alpha \wedge f'(u) = T' \wedge f \xrightarrow{\pi} f'$.

Definition 13 (Type honesty). *f is* honest *iff $f(u)$ is honest, $\forall u \in \mathcal{N} \cup \mathcal{V} \cup \{*\}$.*

Note that, when $\vdash P \colon f$, checking the honesty of f amounts to checking $f(u)$ honest *only* for each $u \in \mathrm{fnv}(P)$. In fact, $f(v) = f(*)$ when $v \notin \mathrm{fnv}(P)$, and $f(*)$ is trivially honest because it cannot advertise contracts.

4.3 System Typing

In this section we establish *type safety* for CO_2 processes: for any typeable closed P, $\mathsf{A}[P]$ is honest. To prove this result, we have to consider the transitions of a process within a system. Hence, in order to construct an invariant of the system transitions (i.e., proving subject reduction), we extend typing to systems.

Type judgments for systems are of two kinds. A judgment of the form $\vdash_{\mathsf{A}} S \colon f$ guarantees that a participant A in S behaves according to f. Instead, a judgment of the form $\vdash_{\mathsf{A}} S \rhd f$ means that A is *not* in S, and S is *compatible* with any participant A behaving as f. Our notion of compatibility is liberal: intuitively, it just checks that the context S does not contain forged contracts of A.

Definition 14 (System typing). *The relations $\vdash_{\mathsf{A}} S \colon f$ and $\vdash_{\mathsf{A}} S \rhd f$ are the smallest relations closed under the rules in Fig. 4.*

Most rules in Fig. 4 are straightforward. Rules [T-SAFree*] tell that A-free systems are compatible with all f. Rules [T-SFz*] state that an f-compatible context (where f is the behaviour of A) may contain latent contracts of A if f realizes them. Rule [T-SFused] is similar, but it deals with stipulated contracts of A. Rule [T-SDel2] is similar to [T-Del] for typing processes. Rule [T-SDel1] is dual: while in [T-SDel2] the type f abstracts the behaviour of A *within* S, in [T-SDel1] it represents the behaviour of A *outside* S. Note that if $\mathsf{A}[P]$ is typeable, then it can be inserted in any A-free system, and the composed system will be typeable.

Example 10. Let $S_0 = \mathsf{B}[Q] \mid \mathsf{C}[\downarrow_x \mathsf{A}\ \mathit{says}\ c]$, with $\mathsf{B} \neq \mathsf{A}$ (note that S_0 is not A-free). Assume that $\vdash P \colon f$. Then, a typing derivation of $S = \mathsf{A}[P] \mid S_0$ is:

$$\dfrac{\dfrac{\vdash P \colon f}{\vdash_{\mathsf{A}} \mathsf{A}[P] \colon f}\ [\text{T-SA}] \qquad \dfrac{\dfrac{\mathsf{B} \neq \mathsf{A}}{\vdash_{\mathsf{A}} \mathsf{B}[Q] \rhd f}\ [\text{T-SAFree1}] \qquad \dfrac{f(x)\ \text{realizes}\ c}{\vdash_{\mathsf{A}} \mathsf{C}[\downarrow_x \mathsf{A}\ \mathit{says}\ c] \rhd f}\ [\text{T-SFz1}]}{\vdash_{\mathsf{A}} \mathsf{B}[Q] \mid \mathsf{C}[\downarrow_x \mathsf{A}\ \mathit{says}\ c] = S_0 \rhd f}\ [\text{T-SPar1}]}{\vdash_{\mathsf{A}} \mathsf{A}[P] \mid S_0 = S \colon f}\ [\text{T-SPar2}]$$

Notice that S is typeable with f only if $f(x)$ realizes A's contract c.

$$\frac{}{\vdash_{\mathsf{A}} \mathbf{0} \rhd f}\ [\text{T-SAFREE0}] \qquad \frac{\mathsf{B} \neq \mathsf{A}}{\vdash_{\mathsf{A}} \mathsf{B}[P] \rhd f}\ [\text{T-SAFREE1}] \qquad \frac{\mathsf{B} \neq \mathsf{A}}{\vdash_{\mathsf{A}} \mathsf{C}[\downarrow_x \mathsf{B}\ \mathit{says}\ c] \rhd f}\ [\text{T-SAFREE2}] \qquad \frac{\gamma\ \ \mathsf{A}\text{-free}}{\vdash_{\mathsf{A}} s[\gamma] \rhd f}\ [\text{T-SAFREE3}]$$

$$\frac{}{\vdash_{\mathsf{A}} \mathsf{B}[\downarrow_s \mathsf{A}\ \mathit{says}\ c] \rhd f}\ [\text{T-SFZS}] \qquad \frac{f(x)\ \text{realizes}\ c}{\vdash_{\mathsf{A}} \mathsf{B}[\downarrow_x \mathsf{A}\ \mathit{says}\ c] \rhd f}\ [\text{T-SFZ1}] \qquad \frac{\vdash_{\mathsf{A}} \mathsf{B}[K] \rhd f \quad \vdash_{\mathsf{A}} \mathsf{B}[K'] \rhd f}{\vdash_{\mathsf{A}} \mathsf{B}[K \mid K'] \rhd f}\ [\text{T-SFZ2}]$$

$$\frac{\emptyset \vdash P\colon f}{\vdash_{\mathsf{A}} \mathsf{A}[P]\colon f}\ [\text{T-SA}] \qquad \frac{\vdash_{\mathsf{A}} S \rhd f\{^{f(*)}/_{u}\}}{\vdash_{\mathsf{A}} (u)S \rhd f}\ [\text{T-SDEL1}] \qquad \frac{\vdash_{\mathsf{A}} S\colon f \quad f(u)\ \text{honest}}{\vdash_{\mathsf{A}} (u)S\colon f\{^{f(*)}/_{u}\}}\ [\text{T-SDEL2}]$$

$$\frac{f(s)\ \text{realizes}\ c}{\vdash_{\mathsf{A}} s[\mathsf{A}\ \mathit{says}\ c \mid \cdots] \rhd f}\ [\text{T-SFUSED}] \qquad \frac{\vdash_{\mathsf{A}} S \rhd f \quad \vdash_{\mathsf{A}} S' \rhd f}{\vdash_{\mathsf{A}} S \mid S' \rhd f}\ [\text{T-SPAR1}] \qquad \frac{\vdash_{\mathsf{A}} S\colon f \quad \vdash_{\mathsf{A}} S' \rhd f}{\vdash_{\mathsf{A}} S \mid S'\colon f}\ [\text{T-SPAR2}]$$

Fig. 4. Typing rules for systems. Symmetric rules wrt | for [T-SFUSED], [T-SPAR2] omitted.

4.4 Type Safety

Subject reduction guarantees that typeability is preserved by transitions. We need to distinguish two cases, according to which participant moves: either the participant A under typing, or any other B. If the transition is done by A, then also its process type must take a transition, otherwise the type is preserved.

The substitution induced when executing a fuse also affects the type of the reduct system. For instance, consider $\mathsf{A}[P] \mid S$, where $\vdash P\colon f$ and $f(x) = T$. Assume that S fires a fuse, with a substitution σ mapping x to a fresh session name s. In the type, this is reflected by updating f according to σ, so to map s to T, and x, which is no longer free, to $f(*)$. Technically, this is done as follows.

Definition 15. *For a type f and a mapping $\{^{v}/_{\vec{u}}\}$, the partial operator $\bullet$ is:*

$$f \bullet \{^{v}/_{\vec{u}}\} = \begin{cases} f & \text{if } \forall u_0 \in \vec{u}\,.\, f(u_0) = f(*) \\ f\{^{f(*)}/_{u_0}\}\{^{f(u_0)}/_{v}\} & \text{if } \exists! u_0 \in \vec{u}\,.\, f(u_0) \neq f(*) \end{cases}$$

Theorem 2 (Subject reduction). *If $\vdash_{\mathsf{A}} S\colon f$ with f honest, then:*

$$S \xrightarrow{\mathsf{A}\,:\,\pi,\sigma} S' \implies \exists f'\,.\, f \xrightarrow{\pi} f' \wedge \vdash_{\mathsf{A}} S'\colon f' \bullet \sigma$$

$$S \xrightarrow{\mathsf{B}\,:\,\pi,\sigma} S' \implies \vdash_{\mathsf{A}} S'\colon f \bullet \sigma \qquad (\textit{when } \mathsf{B} \neq \mathsf{A})$$

Progress guarantees that if a typeable process has a "non-blocking" type, then it can take a transition. More precisely, if the type of P on channel u can take a weak transition with label a, then P will have a in its weak ready do set.

Theorem 3 (Progress). *For all $S \equiv s[\mathsf{A}\ \mathit{says}\ c \mid \cdots] \mid S'$, if $\vdash_{\mathsf{A}} S\colon f$ with f honest, and $f(s) \stackrel{\mathsf{a}}{\Longrightarrow}_{c}^{\mathsf{A}}$, then $\mathsf{a} \in WRD_{s}^{\mathsf{A}}(S)$.*

The main result of this paper is the type safety of CO_2 processes. It ensures that a participant A with a well-typed process P will always respect her contracts — both those already advertised, and those that she will publish along her reductions. Therefore, A will never be considered culpable in any context.

Theorem 4 (Type safety). *$\forall\, \mathsf{A}[P]$ with P closed, $\vdash P\colon f \implies \mathsf{A}[P]$ is honest.*

5 Concluding Remarks and Related Work

We have given a type system for a calculus of contracting processes. Type safety establishes *honesty*: typeable processes honour their contracts in all contexts.

In [3], A is considered "honest" when not definitely *culpable* (in any session), i.e., A eventually performs the actions her contract prescribes. In Def. 5, honesty is based on readiness, rather than culpability; we conjecture that the two notions are equivalent. The main advantage of this new approach compared to [3] is that it simplifies the proof of the correctness of the static analysis of honesty, by more directly relating abstract transitions with concrete ones. Also, the new definition helps in proving decidability of abstract honesty, which was left open in [3].

In [5] (multiparty) asserted global types are used to adapt design-by-contract to distributed interactions. In our framework, a participant declares its contract independently of the others; a CO_2 primitive (fuse) tries then to combine advertised contracts within a suitable agreement. In other words, one could think of our approach as based on orchestration rather than choreography.

In [16] the progress property is checked only when participants engage at most in one session at a time. Honesty is a sort of progress property, and our type system allows participants to interleave many sessions as done in [15]. A crucial difference with respect to [15] is that the typing discipline there requires the *consistency* of the local types of any two participants interacting in a session. Namely, if in a session s, A and B are typed as T_{A} and T_{B} respectively and they interact then the projection of T_{A} with respect to B must be dual of the projection of T_{B} with respect to A. In our type system instead, participants are typechecked 'in isolation' and to establish the honesty of a participant A our typing discipline only imposes that the surrounding context is A-free.

Other approaches deal with safety, by generating monitors that check at runtime the interactions of processes against their local contract (e.g.,[13,12]).

The problem of checking if a contract c, representing the behaviour of a service, conforms to a role r of a choreography H has been investigated in [6]. Conformance of c is attained by establishing a *should testing* pre-order between c and the projection of H with respect to role r. Similar techniques have been used in [7] to define contract-based service composition. A main difference w.r.t. our approach, is that [6,7] do not consider conformance in the presence of dishonest participants. Actually, they focus on matching a contract as a role within a choreography, while we establish if a process abides by its own contracts, regardless of the context.

In [8,9], constraints are used to rule out "inconsistent" executions. This is orthogonal to our approach, since our aim is to blame participants that misbehave.

Acknowledgments. Work partially supported by Aut. Region of Sardinia under grants L.R.7/2007 CRP-17285 (TRICS), P.I.A. 2010 Project "Social Glue", and by MIUR PRIN 2010-11 project "Security Horizons", and by the Leverhulme Trust Programme Award "Tracing Networks", and by COST Action IC1201: Behavioural Types for Reliable Large-Scale Software Systems (BETTY), and by European Union Seventh Framework Programme under grant agreement no. 295261 (MEALS).

References

1. Bartoletti, M., Scalas, A., Tuosto, E., Zunino, R.: Honesty by typing. Technical report (2013), http://tcs.unica.it/publications
2. Bartoletti, M., Tuosto, E., Zunino, R.: Contract-oriented computing in CO_2. Scientific Annals in Computer Science 22(1), 5–60 (2012)
3. Bartoletti, M., Tuosto, E., Zunino, R.: On the realizability of contracts in dishonest systems. In: Sirjani, M. (ed.) COORDINATION 2012. LNCS, vol. 7274, pp. 245–260. Springer, Heidelberg (2012)
4. Bartoletti, M., Zunino, R.: A calculus of contracting processes. In: LICS (2010)
5. Bocchi, L., Honda, K., Tuosto, E., Yoshida, N.: A theory of design-by-contract for distributed multiparty interactions. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 162–176. Springer, Heidelberg (2010)
6. Bravetti, M., Zavattaro, G.: Towards a unifying theory for choreography conformance and contract compliance. In: Lumpe, M., Vanderperren, W. (eds.) SC 2007. LNCS, vol. 4829, pp. 34–50. Springer, Heidelberg (2007)
7. Bravetti, M., Zavattaro, G.: Contract-based discovery and composition of web services. In: Bernardo, M., Padovani, L., Zavattaro, G. (eds.) SFM 2009. LNCS, vol. 5569, pp. 261–295. Springer, Heidelberg (2009)
8. Buscemi, M.G., Coppo, M., Dezani-Ciancaglini, M., Montanari, U.: Constraints for service contracts. In: Bruni, R., Sassone, V. (eds.) TGC 2011. LNCS, vol. 7173, pp. 104–120. Springer, Heidelberg (2012)
9. Buscemi, M.G., Montanari, U.: CC-pi: A constraint-based language for specifying service level agreements. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 18–32. Springer, Heidelberg (2007)
10. Carpineti, S., Castagna, G., Laneve, C., Padovani, L.: A formal account of contracts for web services. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 148–162. Springer, Heidelberg (2006)
11. Castagna, G., Gesbert, N., Padovani, L.: A theory of contracts for web services. ACM TOPLAS 31(5) (2009)
12. Chen, T.-C., Bocchi, L., Deniélou, P.-M., Honda, K., Yoshida, N.: Asynchronous distributed monitoring for multiparty session enforcement. In: Bruni, R., Sassone, V. (eds.) TGC 2011. LNCS, vol. 7173, pp. 25–45. Springer, Heidelberg (2012)
13. Chen, T.-C., Honda, K.: Specifying Stateful Asynchronous Properties for Distributed Programs. In: Koutny, M., Ulidowski, I. (eds.) CONCUR 2012. LNCS, vol. 7454, pp. 209–224. Springer, Heidelberg (2012)
14. Christensen, S.: Decidability and Decomposition in Process Algebras. PhD thesis, Edinburgh University (1993)
15. Coppo, M., Dezani-Ciancaglini, M., Yoshida, N., Padovani, L.: Global Progress for Dynamically Interleaved Multiparty Sessions (2013), http://www.di.unito.it/~padovani/Papers/CoppoDezaniYoshidaPadovani13.pdf
16. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: POPL (2008)
17. Mayr, R.: Decidability and Complexity of Model Checking Problems for Infinite-State Systems. PhD thesis, Technische Universität München (1998)