Honesty by Typing

Technical Report

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

2013

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

1.1 The problem

It is commonplace that distributed applications are not easy to design. Besides the intrinsic issues due e.g. to physical or logical distribution, and to the fragility of communication networks and their low-level protocols, distributed applications have to be engineered within an apparent dichotomy. On the one hand, distributed components have to *cooperate* in order to achieve their goals and, on the other hand, they may have to *compete*, e.g. to acquire shared resources. This dichotomy is well represented by the service-oriented paradigm, which fosters the shift from "stand-alone" applications to dynamically composed ones.

Cooperation and competition hardly coexist harmoniously. Most approaches to the formal specification of concurrent systems typically assume that components behave *honestly*, in that they always adhere to some agreed specification. For instance, this could be some behavioural type inferred from the component, and the assumption is 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. CO_2 [2] is a core calculus for contract-oriented computing. A CO_2 process may advertise contracts to some contract broker; once the broker has found a set of compliant contracts, a session is established among the processes which advertised them. Processes may then use this session to perform the actions needed to realise their contracts, similarly to other session-centric calculi.

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 which tries 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 then how to guarantee that a process will behave honestly, in all possible contexts where it may be run. If such guarantee can be given, then 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 the problem of determining if a process is honest is undecidable for a relevant class of contracts. These are the contracts introduced in [9], and then refined in [10], for modelling WSDL and WSCL contracts. The problem is then how to find a computable approximation of honesty, which implies the dynamic one.

1.2 Example

Let us consider an on-line food store (participant A), which sells apples (a) and bottles of an expensive italian Brunello wine (b). Selling apples is quite easy: once a customer places an order, it is accepted (with the feedback $\overline{\mathsf{ok}}$) and the store waits for payment (pay) before shipping the goods ($\overline{\mathsf{ship\text{-}a}}$). However, if the customer requests an expensive bottle of Brunello, the store reserves itself the right to choose whether to accept the order (and then wait for payment and ship the item, as above), or to decline it, by answering $\overline{\mathsf{no}}$ to the customer. These intentions are modeled by the following store contract:

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

This contract features two kinds of branching operators: external choice $+$, and internal choice $\oplus$. External choice requires the other party (in this case, the customer) to choose which prefix will drive the contract evolution. The choice is between apples a and bottles b. Internal choice, instead, allows the advertising party (the store) to choose a branch, by selecting either $\overline{\mathsf{ok}}$ or $\overline{\mathsf{no}}$.

In order to sell its goods, the store needs to find an agreement with a participant advertising a *compliant* contract. Intuitively, contract compliance is based on the duality of actions and internal/external choices. For instance, the store contract c is compliant with the following customer contract:

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

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

CO_2 allows for describing the behaviour of each participant as a *process*, with the ability to advertise contracts and execute 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 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 it establishes *sessions* when compliant contracts are found. The index x in $\downarrow_x c$ is the name of a *channel* of A. When a session is established between A and (say) B, a fresh session name s is shared between A and B. Technically, x is replaced with s. Participants A and B will then use such session to perform the actions required by their contracts.

A possible specification of A is e.g.:

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

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

Consider now a non-malicious implementation of the food store. Before accepting orders, the store requires an insurance to cover shipment damages — which may be particularly useful for the expensive (and fragile) Brunello bottles. Thus, now A advertises a contract c_p to an insurance company C with an offer to pay ($\overline{\mathsf{payP}}$), followed by the possibility to choose between getting a full coverage on the value of the goods, or cancelling the request:

$$c_p = \overline{\mathsf{payP}}\,.\,\big(\overline{\mathsf{cover}} \oplus \overline{\mathsf{cancel}}\big)$$

The behaviour of the store is now modelled by the process:

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

Here, the store first requests an insurance policy, by advertising the contract c_p; once the insurance company C agrees, the store pays the premium (on channel y). Then, just like the previous case, the store advertises c, and once an agreement with a customer is reached, it waits for a or b orders. If apples are requested, the process acknowledges ($\overline{\mathsf{ok}}$) and invokes $X_N(x)$; there, the store waits for payment, checks which good is expected to be shipped according to the contract, and actually ships it. Otherwise, if Brunello is requested, $Y_N(x,y)$ is invoked: there, the store requests the insurance coverage that was 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. wake a up after a timeout).

This implementation is not as malicious as the first attempt, because at least it actually ships the goods upon payment — but it is not honest either. The problem lies in the interaction between the store and the other parties. If C does not deliver the promised $\overline{\mathsf{cover}}$, the store keeps waiting on $\mathsf{do}_y\,\overline{\mathsf{cover}}$ (which is a blocking operation), unable to honour c by providing the expected $\overline{\mathsf{ok}}/\overline{\mathsf{no}}$. Furthermore, A is dishonest w.r.t. c_p: the insurance fee is paid in advance, but A might 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 remains stuck, or if B simply chooses to buy apples. Thus, due to implementation naïveties, A may be blamed because of the unexpected (or malicious) behaviour of other participants.

An actually honest food store requires a slightly more complex implementation:

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

This time, A advertises c and waits for a or b orders. If apples are requested, the store invokes $X_H(x)$, which acknowledges ok and, just like $X_N(x)$ above, waits for payment and ships the good expected by the contract. If Brunello is requested, then $Y_H(x)$ is invoked instead. There, a new private channel y is created, the store advertises c_p and tries to pay the insurance fee on y; in parallel, the store either requests the coverage and invokes $X_H(x)$ (as above), or it performs an internal action τ (e.g. wake up after a timeout). In the latter case, the order is declined and (in parallel) the insurance request is cancelled. As a result, even if either B or C remains stuck and culpable, A is always able to honour the contract stipulated with the other party.

1.3 Contributions

The main contribution of this paper is a type discipline for statically ensuring when a CO_2 process is *honest*. The need for a static approximation is motivated by the fact that honesty is an undecidable property, as shown in [3]. Our type system associates behavioural types to each channel of a process. Checking honesty on these abstractions is decidable (Theorem 27). We establish subject reduction (Theorem 49) and progress (Theorem 51), which are then used to prove type safety: typeable processes are honest (Theorem 52).

2 A Theory of Contracts

Contracts are modelled in a variant of CCS, inspired by [10] and refined in [3]. Here we provide a brief summary, and refer to [3] for the full technical details.

We assume a set of participants, ranged over $\mathsf{A},\mathsf{B},\ldots$, and a set of *atoms* $\mathsf{a},\mathsf{b},\ldots$, that represent the actions performed by participants. We use an involution $\bar{\mathsf{a}}$, as in CCS. We assume a distinguished atom e (for "end") such that $\mathsf{e}=\bar{\mathsf{e}}$, which models a successfully terminated participant, similarly to [10].

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

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

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

$$
\mathsf{A}\ \textit{says}\ (\bar{\mathsf{a}}\,;c\oplus c')\mid\mathsf{B}\ \textit{says}\ (\mathsf{a}\,.\,d+d')\ \xrightarrow{\mathsf{A}\ \textit{says}\ \mathsf{a}}\!\!\!\!\rightarrow\ \ \mathsf{A}\ \textit{says}\ c\mid\mathsf{B}\ \textit{says ready}\ \mathsf{a}.d
$$

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

$$\mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says}\ \mathit{ready}\ \bar{\mathsf{a}}.\ d \xrightarrow{\mathsf{B}\ \mathit{says}\ \bar{\mathsf{a}}} \mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says}\ d$$

The previous rules do not define the behaviour of a bilateral contract in case an internal choice is not matched by any action in the external choice of the partner. To guarantee that a bilateral contract can keep evolving (until one of the participants wants to exit), we introduce the notion of *compliance*, by adapting that in [10]. This relies on the notion of *ready sets*.

Definition 1 (Ready sets, [3])**.** *The* ready sets *of a contract c (denoted by $RS(c)$) are defined as:*

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

Notice that, $RS(c) \neq \emptyset$ for all contracts c.

We also assume the existence of a *compliance relation* between contracts. Intuitively, we write $c \bowtie d$ when there is a correspondence between the ready sets of c and d, and such relation is maintained when the contracts evolve (i.e., $\mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says}\ d \xrightarrow{\mu} \mathsf{A}\ \mathit{says}\ c' \mid \mathsf{B}\ \mathit{says}\ d' \implies c' \bowtie d'$). The full details are are available in [3].

3 A Calculus of Contracting Processes

The contracts of Sect. 2 are embedded in the process calculus CO_2 [3]. We report in this section the main concepts and definitions. 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 $\mathcal{V} \cup \mathcal{N}$.

Definition 2 (CO_2 syntax)**.** *The syntax of CO_2 is given by:*

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

where S are systems*, K are* latent contracts*, P are* processes*, and π are* prefixes*.*

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

A *latent contract* $\downarrow_x \mathsf{A}\ \mathit{says}\ c$ 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 they are denoted by $\mathrm{fv}(\_)$ and $\mathrm{fn}(\_)$. A system/process is *closed* when it has no free variables. Each participant may have at most one process in a system, i.e. we forbid systems of the form $\mathsf{A}[P] \mid \mathsf{A}[Q]$. We say that a system is A-*free* when it does not contain the participant $\mathsf{A}[P]$, nor latent contracts of A, nor contracts of A stipulated in a session. Note that sessions cannot contain latent contracts.

The semantics of CO_2 is formalised by a reduction relation on systems (Def. 3). This relies on a standard structural congruence, which is the smallest relation satisfying the laws in fig. 3.1 on the facing page. In particular, $(\vec{u})\mathsf{A}[(\vec{v})P] \equiv (\vec{u},\vec{v})\mathsf{A}[P]$ allows to move delimitations between CO_2 systems and processes, while $\mathsf{A}[K] \mid \mathsf{A}[K'] \equiv \mathsf{A}[K \mid K']$ allows to freely select a compliant subset from a group of latent contracts, e.g. before trying to fire rule [FUSE].

In order to define honesty in Sect. 4, 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.

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

Figure 3.1: Structural congruence for CO_2 (Z, Z', Z'' range over processes, systems, or latent contracts)

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

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

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

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

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

$$\frac{S \xrightarrow{\mathsf{A} : \pi, \{s/x\}} S'}{(x)S \xrightarrow{\mathsf{A} : \pi, \emptyset} (s)S'} \quad \text{[Del1]} \qquad \frac{S \xrightarrow{\mathsf{A} : \pi, \sigma} S' \quad u \notin \mathrm{ran}\,\sigma \quad \sigma_{\neq u} \neq \emptyset}{(u)S \xrightarrow{\mathsf{A} : \pi, \sigma_{\neq u}} (u)S'} \quad \text{[Del2]}$$

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

Figure 3.2: Reduction semantics of CO_2

Definition 3 (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. 3.2. 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. 3.2 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 a session satisfies an observable ϕ. The other rules are standard.

Example 4. *Consider the following system:*

$$S = \mathsf{A}[(x)X(x)] \mid \mathsf{B}[(y)Y(y)] \mid K[\mathsf{fuse}]$$
$$X(x) \stackrel{def}{=} \mathsf{tell}_{\mathsf{K}} \downarrow_x (\mathsf{a}; E)\,.\,\mathsf{do}_x \mathsf{a} \qquad Y(y) \stackrel{def}{=} \mathsf{tell}_{\mathsf{K}} \downarrow_y (\bar{\mathsf{a}}\,.E)\,.\,\mathsf{do}_y \bar{\mathsf{a}}$$

A possible execution of S is the following:

$$S \xrightarrow{\mathsf{B} : \mathsf{tell}_{\mathsf{C}} \downarrow_y \bar{\mathsf{a}}, \emptyset} \mathsf{A}[(x)X(x)] \mid \mathsf{C}[\mathsf{fuse}] \mid (y)\,(\mathsf{B}[\mathsf{do}_y \bar{\mathsf{a}}] \mid \mathsf{C}[\downarrow_y \mathsf{B}\ \mathit{says}\ \bar{\mathsf{a}}\,.E]) \tag{1}$$

$$\xrightarrow{\mathsf{A} : \mathsf{tell}_{\mathsf{C}} \downarrow_x \mathsf{a}, \emptyset} (x)\,(\mathsf{A}[\mathsf{do}_x \mathsf{a}] \mid (y)\,(\mathsf{B}[\mathsf{do}_y \bar{\mathsf{a}}] \mid \mathsf{C}[\mathsf{fuse}] \mid \mathsf{C}[\downarrow_x \mathsf{A}\ \mathit{says}\ \mathsf{a}; E \mid \downarrow_y \mathsf{B}\ \mathit{says}\ \bar{\mathsf{a}}\,.E])) \tag{2}$$

$$\xrightarrow{\mathsf{K} : \mathsf{fuse}, \emptyset} (s)\,(\mathsf{A}[\mathsf{do}_s \mathsf{a}] \mid (y)\,(\mathsf{B}[\mathsf{do}_s \bar{\mathsf{a}}] \mid \mathsf{C}[\mathbf{0}] \mid s[\mathsf{A}\ \mathit{says}\ \mathsf{a}; E \mid \mathsf{B}\ \mathit{says}\ \bar{\mathsf{a}}\,.E])) \tag{3}$$
$$\equiv (s)\big(\mathsf{A}[\mathsf{do}_s \mathsf{a}] \mid \mathsf{B}[\mathsf{do}_s \bar{\mathsf{a}}] \mid s[\mathsf{A}\ \mathit{says}\ \mathsf{a}; E \mid \mathsf{B}\ \mathit{says}\ \bar{\mathsf{a}}\,.E]\big)$$

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

Transitions (1) *and* (2) *above are obtained by applying rules* [TELL], [PAR], *and* [DEF]. *The derivation of transition* (3) *is obtained as follows. First, by rule* [FUSE] *we have:*

$$\mathsf{C}[\mathsf{fuse}] \mid \mathsf{C}[\downarrow_x \mathsf{A}\ \mathit{says}\ \mathsf{a}; E \mid \downarrow_y \mathsf{B}\ \mathit{says}\ \bar{\mathsf{a}}.E] \xrightarrow{\mathsf{K}\,:\,\mathsf{fuse},\{s/x,y\}} \mathsf{C}[\mathbf{0}] \mid s[\mathsf{A}\ \mathit{says}\ \mathsf{a}; E \mid \mathsf{B}\ \mathit{says}\ \bar{\mathsf{a}}.E]$$

Hence, by rules [PAR] *and* [DEL2], *we have:*

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

By applying rules [PAR] *and* [DEL1] *to the above, we obtain* (3). *Finally, transition* (4) *is obtained by rule* [DO], *since* $\gamma \xrightarrow{\mathsf{A}\ \mathit{says}\ \mathsf{a}} \gamma'$.

4 On Honesty

We now define when a participant is *honest*. Intuitively, honest participants always respect the contracts they advertise. As remarked in Sect. 1, this notion is crucial in contract-oriented systems, since honest participants will never be liable in case of misbehaviours.

More precisely, a participant A is honest when she *realizes* every contract she advertises, in every session she may be engaged in. Thus, 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.

Example 5. *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 above *readiness* property to be preserved by arbitrary transitions taken by S. This amounts to say that, in any reduct of S containing a reduct P' of P and a reduct c' of c, the process P' must still be ready for c'.

To formalise the notion of "P is ready for c", we need to inspect P and c. At the contract level, the ready sets in $RS(c)$ (Def. 1) reveal whether c is exposing an internal or an external choice. At the process level, we consider the reachable actions in P.

Example 6 (Processes and readiness)**.** *Consider the following processes:*

$$\begin{aligned} P_0 &= \mathsf{do}_s\,\mathsf{a} & P_1 &= \mathsf{do}_s\,\mathsf{a} + \mathsf{do}_s\,\mathsf{b} + \mathsf{do}_s\,\mathsf{z} \\ P_2 &= \tau.\mathsf{do}_s\,\mathsf{a} + \tau.\mathsf{do}_s\,\mathsf{b} & P_3 &= \mathsf{do}_t\,\mathsf{w}.\mathsf{do}_s\,\mathsf{a} + \mathsf{do}_t\,\mathsf{z}.\mathsf{do}_s\,\mathsf{b} \end{aligned}$$

We now study whether $P_0, \ldots, P_3$ *are ready for contracts* c_i *and* c_e *(introduced in Ex. 5) in session* s. *According to Def. 1, the ready sets of* c_i *are* $\{\mathsf{a}\}$ *and* $\{\mathsf{b}\}$, *while* c_e *has only the ready set* $\{\mathsf{a}, \mathsf{b}\}$. *We have that:*

- P_0 *is ready for* c_i, *because there exists a ready set* ($\{\mathsf{a}\}$) *in* $RS(c_i)$ *such that* $\mathsf{do}_s\,\mathsf{a}$ *is enabled in* P_0. *Instead,* P_0 *is* not *ready for* c_e, *because the ready set* $\{\mathsf{a}, \mathsf{b}\}$ *of* c_e *also contains* b, *which is not enabled in* P_0.
- P_1 *is ready for both* c_i *and* c_e. *This is because* P_1 *enables two actions,* $\mathsf{do}_s\,\mathsf{a}$ *and* $\mathsf{do}_s\,\mathsf{b}$, *which cover all the ready sets of* c_i *and* c_e. *Notice that the branch* $\mathsf{do}_s\,\mathsf{z}$ *is immaterial, because rule* [DO] *blocks any action not expected by the contract.*
- P_2 *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.*
- *The case of* P_3 *is a bit more complex than the above ones. 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 participant A in a system S.

Definition 7 (Ready do)**.** *For all S, A and u, we define the set of atoms $RD_u^{\mathsf{A}}(S)$ as:*

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

Example 8. *Consider the following system:*

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

We have that $RD_x^{\mathsf{A}}(S) = \{\bar{\mathsf{a}}\}$, and $RD_y^{\mathsf{A}}(S) = \emptyset$.

As seen in the above example for processes P_2 and P_3, readiness may also hold when the actions expected in the contract ready sets are not immediately available in the process. To check if $\mathsf{A}[P]$ is ready for session s (in a system S), we need to consider all the 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 actions performed by A. These actions are collected in the set $WRD_s^{\mathsf{A}}(S)$.

Definition 9 (Weak ready do)**.** *We write $S \xrightarrow{\neq(\mathsf{A}:\,\mathsf{do}_u)} S'$ if:*

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

We then define the set of atoms $WRD_u^{\mathsf{A}}(S)$ as:

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

Example 10. *Recall the system S from Ex. 8. We have that:*

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

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 11. *Recall the process $P_3 = \mathsf{do}_t\,\mathsf{w}\,.\,\mathsf{do}_s\,\mathsf{a} + \mathsf{do}_t\,\mathsf{z}\,.\,\mathsf{do}_s\,\mathsf{b}$ from Ex. 6. Consider the following system, where participant A is involved in two sessions s and t (respectively, with B and C):*

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

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

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

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 only reachable after B executes $\mathsf{do}_s\,\bar{\mathsf{a}}$, thus requiring a reduction trace which does not *have the form $S \xrightarrow{\neq(\mathsf{B}:\,\mathsf{do}_s)}{}^{*}$. Finally, we emphasize that, if C chooses to perform τ, then the actions in $WRD_s^{\mathsf{A}}(S)$ would not be reached. Indeed, Def. 9 only requires that each element in the set becomes reachable at the end of a suitable (weak) reduction trace — but it does not prevent S from reducing along other paths.*

A participant A is ready in a system S containing 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. Notice that A is vacuously ready in systems not containing sessions with contracts stipulated by A.

Definition 12 (Readiness). *We say that* A *is ready in* S *iff, whenever* $S \equiv (\vec{u})S'$ *for some* $\vec{u}$ *and* $S' = s[\mathsf{A}\ \mathit{says}\ c \mid \cdots] \mid S_0$,

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

A process $\mathsf{A}[P]$ is said to be honest when, for all contexts where $\mathsf{A}[P]$ may be engaged in, A is persistently ready in all the reducts of that context. Notice that $\mathsf{A}[P]$ is vacuously honest when P advertises no contracts.

Informally, we shall say that A *realizes* a contract c in a session s in S when S has the form $\mathsf{A}[P] \mid s[\mathsf{A}\ \mathit{says}\ c \mid \cdots] \mid \cdots$, and the readiness condition is satisfied in S and in all its reducts. Then, $\mathsf{A}[P]$ is honest when A realizes all the contracts she advertises.

Definition 13 (Honesty). *We say* $\mathsf{A}[P]$ honest *iff for all* A*-free* S*, and for all* S' *such that* $\mathsf{A}[P] \mid S \to^* S'$, A *is ready in* S'.

The A-freeness requirement in Def. 13 is used just to rule out those systems which already carry 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 14. *Consider the following system:*

$$\begin{array}{c} S \stackrel{def}{=} \mathsf{A}[(x,y)\ (P_A \mid \mathsf{fuse} \mid \mathsf{fuse})] \ \mid\ \mathsf{B}[P_B] \ \mid\ \mathsf{C}[P_C] \\ P_A \stackrel{def}{=} \mathsf{tell}_{\mathsf{A}}\,(\downarrow_x \mathsf{a}\,.E)\,.\,\mathsf{tell}_{\mathsf{A}}\,(\downarrow_y \mathsf{b}\,;E)\,.\,\mathsf{do}_x\mathsf{a}\,.\,\mathsf{do}_y\mathsf{b} \\ P_B \stackrel{def}{=} (z)\ (\mathsf{tell}_{\mathsf{A}}\,(\downarrow_z \bar{\mathsf{b}}\,.E)\,.\,\mathsf{do}_z\bar{\mathsf{b}}) \qquad P_C \stackrel{def}{=} (w)\ (\mathsf{tell}_{\mathsf{A}}\,(\downarrow_w \bar{\mathsf{a}}\,;E)\,.\,\mathbf{0}) \end{array}$$

Even though A *might apparently look honest, she is not. In fact, if we reduce* S *by performing all the* tell *and* fuse *actions, we obtain:*

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

Here, S' *cannot reduce further. Indeed,* C *(dishonestly) avoids to perform the internal choice required by his contract. Then,* A *is stuck, waiting for* a *from* C*. Therefore,* A *is dishonest, because she does not perform the promised action* b*. Formally, the dishonesty of* A *follows 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, which eventually allows participants to fire persistently (weakly) enabled do actions. This is illustrated by the following two examples.

Example 15. *Consider the contract* $c = \mathsf{a} \oplus \mathsf{b}$*, and let:*

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

Let $S = \mathsf{A}[P] \mid S_0$*, and assume that the* fuse *in* P *passes. Then,* S *reduces to* $S' \equiv (s)\big(\mathsf{A}[X(s)] \mid s[A\ \mathit{says}\ c \mid \cdots] \mid S_0'\big)$*. 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. Technically, the fair scheduler is rendered within Def. 9 and 13. Def. 9 considers* a *and* b *weakly enabled in* S'*, because there* exists *a way to reach each of them. Since from any reduct of* S' *either* a *or* b *are reachable, then Def. 13 considers* $\mathsf{A}[P]$ *honest.*

Example 16. *Consider the contract* $c = \mathsf{a} + \mathsf{b}$ *and let:*

$$\begin{array}{rcl} P & \stackrel{def}{=} & (x)\big(\mathsf{tell}_{\mathsf{A}} \downarrow_x c\,.\,\mathsf{fuse}\,.\,X(x)\big) \\ X(x) & \stackrel{def}{=} & \tau\,.X(x) \ +\ \tau\,.(\tau\,.X(x) + \mathsf{do}_x\mathsf{a}) \ +\ \tau\,.(\tau\,.X(x) + \mathsf{do}_x\mathsf{b}) \end{array}$$

Let $S = \mathsf{A}[P] \mid S_0$*. After the* fuse*, the system* S *reduces to* $S' \equiv (s)\big(\mathsf{A}[X(s)] \mid s[A\ \mathit{says}\ c \mid \cdots] \mid S_0'\big)$*. As in the previous example, an unfair scheduler might make* A *not respect her contract. However, in all the reducts of* S' *both* a *and* b *are reachable. Indeed, there is no branch which definitely commits to one of the two actions. Therefore, according to Def. 13,* $\mathsf{A}[P]$ *is honest.*

$$\frac{}{\alpha.T \xrightarrow{\alpha} T}\ \text{[C-Alpha]} \qquad \frac{T \xrightarrow{\alpha} T'}{T + T'' \xrightarrow{\alpha} T'}\ \text{[C-SumL]} \qquad \frac{T \xrightarrow{\alpha} T'}{T \mid T'' \xrightarrow{\alpha} T' \mid T''}\ \text{[C-ParL]}$$

$$\frac{T\{^{rec\,X.T}/_{X}\} \xrightarrow{\alpha} T'}{rec\,X.T \xrightarrow{\alpha} T'}\ \text{[C-Rec]} \qquad rec\,X.T \equiv T\{^{rec\,X.T}/_{X}\} \qquad \text{commutative monoidal laws for } \mid, +$$

Figure 5.1: Channel type semantics.

5 A Type System for CO_2

We now introduce a type system for CO_2. The main result is *type safety* (established in Th.52), which guarantees that typeable participants are honest.

The type of a process P is a function f, which maps each channel (either session name or variable) to a *channel type*. Channel types are behavioural types which essentially preserve the structure of P (branching, parallel composition, recursion), while abstracting the actual prefixes and delimitations. Mainly, the prefixes of channel types distinguish between nonblocking and possibly blocking actions.

In Sect. 5.1 we define channel types; then, in Sect. 5.2 we define process types and the type system for processes. In Sect. 5.3 we present an auxiliary set of typing rules for CO_2 systems, which are only needed to state subject reduction and progress in Sect. 5.4. Type safety is established in Sect. 5.5.

5.1 Channel types

Channel types extend Basic Parallel Processes (BPPs [15]) by allowing prefixes of the following kinds: atoms ($\mathsf{a}, \mathsf{b}, \ldots$), nonblocking silent actions (τ), possibly blocking silent actions ($\tau_?$), conditional silent actions depending on observables (τ_ϕ), and contract advertisement actions ($\langle c \rangle$).

Definition 17 (Channel types)**.** *The syntax of* channel types T *and* prefixes α *is defined as follows:*

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

We denote with $\mathbb{T}$ the set of all channel types.

The semantics of channel types is given in Def. 18, in terms of a labelled transition relation $\xrightarrow{\alpha}$.

Definition 18 (Channel type semantics)**.** *The relation $\xrightarrow{\alpha}$ is the least relation closed under the rules of Fig. 5.1.*

The rules for $\xrightarrow{\alpha}$ are the standard ones for BPPs. Hereafter, we shall identify structurally congruent channel types.

Example 19. *Consider the following CO_2 process:*

$$P = \mathsf{tell}_{\mathsf{B}} \downarrow_x c_i \mid \left(\mathsf{tell}_{\mathsf{B}} \downarrow_y d \,.\, \mathsf{do}_x \bar{\mathsf{a}}\right)$$

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:

$$T_x = \langle c_i \rangle \mid \tau.\bar{\mathsf{a}} \qquad\qquad T_y = \tau \mid \langle d \rangle.\tau_?$$

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 actions in its continuation. The channel type T_x can reduce in several ways, e.g.:

$$T_x \xrightarrow{\langle c_i \rangle} \tau.\bar{\mathsf{a}} \xrightarrow{\tau} \bar{\mathsf{a}} \xrightarrow{\bar{\mathsf{a}}} \mathbf{0} \tag{5}$$
$$T_x \xrightarrow{\tau} \langle c_i \rangle \mid \bar{\mathsf{a}} \xrightarrow{\bar{\mathsf{a}}} \langle c_i \rangle \xrightarrow{\langle c_i \rangle} \mathbf{0} \tag{6}$$

$$\frac{T \xrightarrow{\langle c \rangle} T'}{(C,T) \to (C \cup \{c\}, T')}\ \text{[A-Tell1]} \qquad \frac{T \xrightarrow{\langle d \rangle} T'}{(c,T) \to (c,T')}\ \text{[A-Tell2]} \qquad \frac{c \in C}{(C,T) \to (c,T)}\ \text{[A-Fuse]}$$

$$\frac{T \xrightarrow{\alpha} T' \quad \alpha \in \{\tau, \tau_?, \tau_\phi\}}{(C,T) \to (C,T')}\ \text{[A-Tau1]} \qquad \frac{c \xrightarrow{\mathsf{a}}_\sharp c' \quad T \xrightarrow{\mathsf{a}} T'}{(c,T) \to (c',T')}\ \text{[A-Do]}$$

$$\frac{T \xrightarrow{\alpha} T' \quad \alpha \in \{\tau, \tau_?, \tau_\phi\}}{(c,T) \to (c,T')}\ \text{[A-Tau2]} \qquad \frac{c \xrightarrow{ctx}_\sharp c'}{(c,T) \to (c',T)}\ \text{[A-Ctx]}$$

Figure 5.2: Abstract processes semantics.

The execution of CO_2 systems relies both on processes and on (advertised/stipulated) contracts. An abstraction of the latter is then used to define an abstract semantics of processes.

Definition 20 (Abstract processes)**.** *An* abstract process *is either a pair (C,T) or a pair (c,T), where C is a set of contracts, c is a contract, and T is a channel type.*

Definition 21 (Abstract process semantics)**.** *The semantics of abstract processes is given in terms of a transition relation $\to$, which is the least relation closed under the rules of Fig. 5.2.*

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

The set C grows when a channel type T in (C,T) performs a transition with label $\langle c \rangle$ (rule [A-Tell1]). After one of the contracts in C has been stipulated (rule [A-Fuse]), the set is reduced to c. Rule [A-Do] models a $\mathsf{do}\,\mathsf{a}$ action performed by T, while rule [A-Ctx] models an (unknown) action performed by the context. Further advertisements after contract stipulation are neglected (rule A-Tell2). Notice that in rules [A-Do] and [A-Ctx] contracts are reduced through the relation $\twoheadrightarrow_\sharp$. This relation 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 $c \bowtie d$,

$$\gamma \xrightarrow{\mathsf{A}\ \mathit{says}\ \mathsf{a}} \mathsf{A}\ \mathit{says}\ c' \mid \mathsf{B}\ \mathit{says}\ d' \implies c \xrightarrow{\mathsf{a}}_\sharp c'$$
$$\gamma \xrightarrow{\mathsf{B}\ \mathit{says}\ \mathsf{b}} \mathsf{A}\ \mathit{says}\ c' \mid \mathsf{B}\ \mathit{says}\ d' \implies c \xrightarrow{ctx}_\sharp c'$$

Example 22. *Recall the trace* (5) *in Ex. 19. That induces the following two traces for the abstract process $(\emptyset, T_x)$. Below, we annotate arrows with rule names from Fig. 5.2.*

$$(\emptyset,T_x) \xrightarrow[\text{[A-Tell1]}]{} (\{c_i\},\tau.\bar{\mathsf{a}}) \xrightarrow[\text{[A-Fuse]}]{} (c_i,\tau.\bar{\mathsf{a}}) \xrightarrow[\text{[A-Tau2]}]{} (c_i,\bar{\mathsf{a}}) \xrightarrow[\text{[A-Do]}]{} (E,\mathbf{0})$$
$$(\emptyset,T_x) \xrightarrow[\text{[A-Tell1]}]{} (\{c_i\},\tau.\bar{\mathsf{a}}) \xrightarrow[\text{[A-Tau1]}]{} (\{c_i\},\bar{\mathsf{a}}) \xrightarrow[\text{[A-Fuse]}]{} (c_i,\bar{\mathsf{a}}) \xrightarrow[\text{[A-Do]}]{} (E,\mathbf{0})$$

Instead, we are not able to follow trace (6)*, since $(\emptyset,T_x) \xrightarrow[\text{[A-Tau1]}]{} (\emptyset,\langle c_i \rangle \mid \bar{\mathsf{a}}) \not\xrightarrow[\text{[A-Do]}]{}$. Intuitively, in* (6) *the action* a *is performed before the contract c_i is advertised — but this is not possible because of rule* [A-Do].

We now introduce the abstract counterpart of the dynamic notion of honesty in Sect. 4. 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.

In the case of concrete processes, readiness requires to match the "weak ready do" set of the process against the ready sets of the contract (Def. 12). Similarly, here we shall match the "weak transitions" of a channel type with the ready sets of the contract.

Indeed, such weak transitions abstract the weak ready do set. That is, if an abstract process can take a weak transition a, then also the concrete one will do that. This under-approximation is needed to ensure the correctness of

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

Figure 5.3: Channel type semantics (weak transition, parameterised by A and c).

abstract honesty: if an abstract process is honest, then also the concrete one will be such (while the *vice versa* is not always true).

Recall that the actions a in the weak ready do set (of session s) are those to be fired in a $\mathsf{do}_s\,\mathsf{a}$ by the concrete process. Their abstract counterpart, i.e. labels of weak transitions, consider actions reachable through sequences of non-blocking (abstract) transitions, which are included in the ready do set. Unlike in the concrete case, the context is immaterial in determining weak transitions.

Weak transitions are defined in Fig. 5.3 as a labelled relation $\overset{\mathsf{a}}{\Longrightarrow}{}^{\mathsf{A}}_{c}$ (simply written as $\overset{\mathsf{a}}{\Longrightarrow}$ 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 Sect. 3), and we only require that it respects the constraint in Def. 23 below.

Definition 23 (Abstract observability)**.** *We write $c \vdash^{\mathsf{A}}_\sharp \phi$ for any decidable relation between contracts and observables satisfying:*

$$c \vdash^{\mathsf{A}}_\sharp \phi \implies \forall \mathsf{B}\,.\,\forall d\,.\,(c \bowtie d \implies \mathsf{A}\ \textit{says}\ c \mid \mathsf{B}\ \textit{says}\ d \vdash \phi)$$

The definition of abstract readiness (Def. 24) follows along the lines of Def. 12.

Definition 24 (Abstract readiness)**.** *For a channel type T and a contract c, we say that T is* abstractly ready for *c iff:*

$$\exists X \in RS(c)\,.\,\forall \mathsf{a} \neq \mathsf{e}\,.\,\big(\mathsf{a} \in X \ \vee\ \textit{ready}\ \mathsf{a} \in X \implies T \overset{\mathsf{a}}{\Longrightarrow}\big)$$

Hereafter, when referring to properties of abstract entities, we shall omit the qualifier "abstractly", e.g. we shall write that a channel type is "ready", instead of "abstractly ready".

Honesty of abstract processes is defined similarly to Def. 13. 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 reductions of the abstract process, starting from the empty set of contracts.

Definition 25 (Abstract honesty)**.** *We say that:*

- *An abstract process $(-,T)$ is honest iff*

$$\forall c, T'\,.\,(-,T) \to^* (c,T') \implies T' \textit{ is ready for } c$$

- *A channel type T is* honest *iff $(\emptyset,T)$ is honest.*

Informally, we say that T realizes *c whenever (c,T) is honest.*

Example 26. *Recall the type $T_x = \langle c_i \rangle \mid \tau\,.\,\mathsf{a}$ and the contract $c_i = \mathsf{a} \oplus \mathsf{b}$ from Ex. 22. To determine whether 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. $(\emptyset, \langle c_i \rangle \mid \bar{\mathsf{a}})$. *Similar to the previous case.*
3. $(\{c_i\}, \tau\,.\,\bar{\mathsf{a}})$. *Nothing to check, because no contracts have been stipulated yet.*
4. $(\{c_i\}, \bar{\mathsf{a}})$. *Similar to the previous case.*
5. $(c_i, \tau\,.\,\bar{\mathsf{a}})$. *We have that $\tau\,.\,\bar{\mathsf{a}}$ is ready for c_i, because for $\{\bar{\mathsf{a}}\} \in RS(c_i) = \{\{\bar{\mathsf{a}}\}, \{\bar{\mathsf{b}}\}\}$, we have $\tau\,.\,\bar{\mathsf{a}} \overset{\bar{\mathsf{a}}}{\Longrightarrow}$.*
6. $(c_i, \bar{\mathsf{a}})$. *We have that $\bar{\mathsf{a}}$ is ready for c_i, similarly to the previous case.*
7. $(E, \mathbf{0})$. *We have that $\mathbf{0}$ is vacuously ready for E.*

Summing up, we conclude that T_x is honest.

Th.27 below establishes that checking the honesty of a channel type T is decidable. Indeed, both 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 Basic Parallel Process. This product can be modelled as a Petri net. Decidability follows because reachability is decidable for Petri nets [15].

Theorem 27 (Decidability of abstract honesty)**.** *Abstract honesty is decidable.*

Proof. See appendix A.2 on page 27. □

5.2 Process types

Process types associate session names/variables to channel types, thus abstracting the behaviour of a process on all channels. Additionally, we consider a special "dummy" channel $* \notin \mathcal{N} \cup \mathcal{V}$, where we collect type information about unused channels.

Definition 28 (Process type)**.** *A* CO_2 process type *is a function* $f\colon \mathcal{N} \cup \mathcal{V} \cup \{*\} \to \mathbb{T}$.

Intuitively, our type system abstracts concrete prefixes of CO_2 processes as actions of channel types. Such abstraction is rendered as the mapping in Def. 29. 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:

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

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 $u = v$ and $\tau_?$ otherwise. This allows for exploiting suitable static approximations of the relation $\vdash$ (see Fig. 5.3).

Definition 29 (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:*

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

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

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

Definition 30 (Typing environment)**.** *A* typing environment Γ *is a partial function which associates process types to constants* $X(\vec{v})$.

We can now introduce the typing rules for CO_2 processes.

Definition 31 (Typing rules for processes)**.** *The typing rules for processes are shown in Fig. 5.4.*

Rule [T-SUM] abstracts the prefixes which guard the branches of a summation, according to Def. 29. 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 rule [T-DEF]. The rule for typing delimitations ([T-DEL]) is worth some extra 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 channel u, the rule also requires to check 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.

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

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

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

Figure 5.4: Typing rules for processes.

Example 32. *Recall the process $P_2 = \tau \,.\, \mathsf{do}_s\, \mathsf{a} + \tau \,.\, \mathsf{do}_s\, \mathsf{b}$ from Ex. 6. Its typing derivation is obtained by* [T-Sum] *as follows:*

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

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" from channel s, while remaining "silent" on other channels. If we slightly change the process, and consider instead $P_2' = \tau \,.\, \mathsf{do}_s\, \mathsf{a} + \tau \,.\, \mathsf{do}_t\, \mathsf{b}$, we have:

$$\frac{\dfrac{}{\vdash \mathsf{do}_s\, \mathsf{a} \colon \lambda u \,.\, [\mathsf{do}_s\, \mathsf{a}]_u} \text{ [T-Sum]} = f_1 \quad \dfrac{}{\vdash \mathsf{do}_t\, \mathsf{b} \colon \lambda u \,.\, [\mathsf{do}_t\, \mathsf{b}]_u = f_2'} \text{ [T-Sum]}}{\vdash P_2' \colon f' = \lambda u \,.\, [\tau]_u \,.\, f_1(u) + [\tau]_u \,.\, f_2'(u)} \text{ [T-Sum]}$$

and thus:

$$f'(s) = \tau \,.\, \mathsf{a} + \tau \,.\, \tau_? \qquad f'(t) = \tau \,.\, \tau_? + \tau \,.\, \mathsf{b} \qquad \forall u \notin \{s,t\} \,.\, f'(u) = f'(*) = \tau \,.\, \tau_? + \tau \,.\, \tau_?$$

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

Lemma 33 (Process typing and $*$)**.** *For all P, $\vdash P \colon f \implies f(*)$ only contains τ and $\tau_?$ actions.*

Proof. See appendix A.3 on page 27. □

Lemma 34 (Process typing and non-free names/vars)**.** *For all processes P and for all environments Γ: $z \notin \text{fnv}(P) \wedge \Gamma \vdash P \colon f \implies f(z) = f(*)$.*

Proof. See appendix A.4 on page 29. □

Types are preserved by structural equivalence of processes (lemma 35 on this page).

Lemma 35 (Structural equivalence and process typing)**.** *For all CO$_2$ processes P, P': $P \equiv P' \wedge \Gamma \vdash P \colon f \implies \Gamma \vdash P' \colon f$.*

Proof. See appendix A.5 on page 29. □

We now define a partial order on process types. Intuitively, $f \sqsubseteq f'$ holds when f and f' behave in the same way when observed on the same channels — except those in which f is silent.

Definition 36 (Process type order)**.** *We define a partial order $\sqsubseteq$ on process types as:*

$$f \sqsubseteq f' \iff \forall u \in \mathcal{N} \cup \mathcal{V} \cup \{*\} \,.\, f(u) = f'(u) \vee f(u) = f'(*)$$

Delimitation makes types smaller (i.e., "more silent") w.r.t. $\sqsubseteq$.

Lemma 37 (Delimitation and type ordering)**.** $\vdash (u)P \colon f \wedge \vdash P \colon f' \implies f \sqsubseteq f'$.

Proof. See appendix A.6 on page 30. □

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

Example 39. *Recall the process $P_1 = \mathsf{do}_s\,\mathsf{a} + \mathsf{do}_s\,\mathsf{b} + \mathsf{do}_s\,\mathsf{z}$ from Ex. 6. Its typing is $\vdash P_1 : f = \lambda u \,.\, [\mathsf{do}_s\,\mathsf{a}]_u + [\mathsf{do}_s\,\mathsf{b}]_u + [\mathsf{do}_s\,\mathsf{z}]_u$. 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 associated to some process, and $f(u)$ takes an abstract transition, then the whole f can take a transition.

Lemma 40 (Channel type and process type reductions)**.** *For all inhabited types f, and for all $u \in \mathcal{N} \cup \mathcal{V}$,*

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

Proof. See appendix A.7 on page 30. □

We extend to process types the notion of honesty of Def. 25.

Definition 41 (Process type honesty)**.** *We say that f is* honest *iff $f(u)$ is honest, for all $u \in \mathcal{N} \cup \mathcal{V} \cup \{*\}$.*

Note that, when $\vdash P : f$, checking the honesty of f amounts to checking $f(u)$ honest, for all $u \in \mathrm{fnv}(P)$. Actually, by lemma 34 on the preceding page, $f(u) = f(*)$ on the other channels, and $f(*)$ is trivially honest because it cannot advertise contracts (lemma 59 on page 20).

Lemma 42 (Process type honesty and ordering)**.** *f honest $\wedge$ $f' \sqsubseteq f \implies f'$ honest.*

Proof. See appendix A.8 on page 31. □

5.3 System typing

The type system for processes is enough to guarantee whether a participant is honest. However, in order to establish a *type safety* 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., subject reduction), we extend typing also to systems.

Type judgments for systems are of two kinds. A judgment of the form $\vdash_{\mathsf{A}} S : f$ 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's process is *not* in S, and S is guaranteed to be *compatible* with a participant A which behaves as f. Our notion of compatibility is quite liberal: intuitively, it just checks that the context S has not forged contracts of A.

Definition 43 (System typing)**.** *The relations $\vdash_{\mathsf{A}} S : f$ and $\vdash_{\mathsf{A}} S \rhd f$ are the smallest relations closed under the rules in Fig. 5.5.*

Most rules in Fig. 5.5 are straightforward: for instance, 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 such contracts.

Rule [T-SFUSED] is similar, except that it deals with stipulated contracts of A. Rule [T-SDEL2] is similar to rule [T-DEL] for typing processes. Rule [T-SDEL1] is dual, reflecting the fact that the type f in [T-SDEL2] abstracts the behaviour of A *within* S, while in [T-SDEL1] it represents the behaviour of A *outside* S.

Structural equivalence preserves system typing.

Lemma 44 (Structural equivalence and system typing)**.** *Whenever $S \equiv S'$,*

$$\vdash_{\mathsf{A}} S : f \implies \vdash_{\mathsf{A}} S' : f \tag{7}$$

$$\vdash_{\mathsf{A}} S \rhd f \implies \vdash_{\mathsf{A}} S' \rhd f \tag{8}$$

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

$$\frac{\gamma\ \mathsf{A}\text{-free}}{\vdash_{\mathsf{A}} s[\gamma] \rhd f}\ [\text{T-SAFREE3}] \qquad \frac{}{\vdash_{\mathsf{A}} \mathsf{B}[\downarrow_s \mathsf{A}\ \mathit{says}\ c] \rhd f}\ [\text{T-SFZS}] \qquad \frac{f(x)\ \text{realizes}\ c}{\vdash_{\mathsf{A}} \mathsf{B}[\downarrow_x \mathsf{A}\ \mathit{says}\ c] \rhd f}\ [\text{T-SFZ1}]$$

$$\frac{\vdash_{\mathsf{A}} \mathsf{B}[K] \rhd f \quad \vdash_{\mathsf{A}} \mathsf{B}[K'] \rhd f}{\vdash_{\mathsf{A}} \mathsf{B}[K \mid K'] \rhd f}\ [\text{T-SFZ2}] \qquad \frac{f(s)\ \text{realizes}\ c}{\vdash_{\mathsf{A}} s[\mathsf{A}\ \mathit{says}\ c \mid \cdots] \rhd f}\ [\text{T-SFUSED}]$$

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

$$\frac{\vdash_{\mathsf{A}} S \rhd f \quad \vdash_{\mathsf{A}} S' \rhd f}{\vdash_{\mathsf{A}} S \mid S' \rhd f}\ [\text{T-SPAR1}] \qquad \frac{\vdash_{\mathsf{A}} S\colon f \quad \vdash_{\mathsf{A}} S' \rhd f}{\vdash_{\mathsf{A}} S \mid S'\colon f}\ [\text{T-SPAR2}]$$

Figure 5.5: Typing rules for systems. The symmetric rules wrt to $\mid$ for [T-SFUSED] and [T-SPAR2] are omitted.

Proof. See appendix A.9 on page 31. □

The following is the system typing counterpart of lemma 37 on page 13.

Lemma 45 (Delimitation and type ordering for systems)**.** $\vdash_{\mathsf{A}} (u)S\colon f \wedge \vdash_{\mathsf{A}} S\colon f' \implies f \sqsubseteq f'$.

Proof. See appendix A.10 on page 35. □

If a participant $\mathsf{A}[P]$ is typeable, then it can be inserted in any A-free system, and the composed system will remain typeable.

Example 46. *Consider a participant* $\mathsf{A}[P]$ *such that* $\vdash P\colon f$*, and let* $S_0 = \mathsf{B}[Q] \mid \mathsf{C}[\downarrow_x \mathsf{B}\ \mathit{says}\ c]$*, with* $\mathsf{B} \neq \mathsf{A}$*. Notice that* S_0 *is* A*-free. The typing derivation of* $S = \mathsf{A}[P] \mid S_0$ *is:*

$$\dfrac{\dfrac{\vdash P\colon f}{\vdash_{\mathsf{A}} \mathsf{A}[P]\colon f}\ [\text{T-SA}] \qquad \dfrac{\dfrac{\mathsf{B} \neq \mathsf{A}}{\vdash_{\mathsf{A}} \mathsf{B}[Q] \rhd f}\ [\text{T-SAFREE1}] \qquad \dfrac{\mathsf{B} \neq \mathsf{A}}{\vdash_{\mathsf{A}} \mathsf{C}[\downarrow_x \mathsf{B}\ \mathit{says}\ c] \rhd f}\ [\text{T-SAFREE2}]}{\vdash_{\mathsf{A}} \mathsf{B}[Q] \mid \mathsf{C}[\downarrow_x \mathsf{B}\ \mathit{says}\ c] = S_0 \rhd f}\ [\text{T-SPAR1}]}{\vdash_{\mathsf{A}} S = \mathsf{A}[P] \mid S_0\colon f}\ [\text{T-SPAR2}]$$

Example 47. *Consider now a non-*A*-free system* S_0*, e.g. let* $S_0 = \mathsf{B}[Q] \mid \mathsf{C}[\downarrow_x \mathsf{A}\ \mathit{says}\ c]$*, with* $\mathsf{B} \neq \mathsf{A}$*. Notice that* S_0 *is* not A*-free. The typing derivation of* $S = \mathsf{A}[P] \mid S_0$ *is as follows:*

$$\dfrac{\dfrac{\vdash P\colon f}{\vdash_{\mathsf{A}} \mathsf{A}[P]\colon f}\ [\text{T-SA}] \qquad \dfrac{\dfrac{\mathsf{B} \neq \mathsf{A}}{\vdash_{\mathsf{A}} \mathsf{B}[Q] \rhd f}\ [\text{T-SAFREE1}] \qquad \dfrac{f(x)\ \mathit{realizes}\ c}{\vdash_{\mathsf{A}} \mathsf{C}[\downarrow_x \mathsf{A}\ \mathit{says}\ c] \rhd f}\ [\text{T-SFZ1}]}{\vdash_{\mathsf{A}} \mathsf{B}[Q] \mid \mathsf{C}[\downarrow_x \mathsf{A}\ \mathit{says}\ c] = S_0 \rhd f}\ [\text{T-SPAR1}]}{\vdash_{\mathsf{A}} S = \mathsf{A}[P] \mid S_0\colon f}\ [\text{T-SPAR2}]$$

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

5.4 Subject reduction and progress

To establish subject reduction, we need to cope with the fact that the evaluation of a fuse prefix substitutes session names for variables. This substitution also affects the type of the reduct process. For instance, consider the system $\mathsf{A}[P] \mid S$, where $\vdash P\colon f$ and $f(x) = T$. Assume that now the context S fires a fuse, which substitutes a fresh session name s for x. The typing of the reduct system will accommodate this by mapping s to T, while x is mapped to $f(*)$, because x is no longer free after the substitution.

Technically, this type substitution is obtained through the operator $\bullet$, introduced in the following definition.

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

$$(\Gamma \bullet \{{}^{v}/_{u_0}\})(Y(\vec{w})) = \begin{cases} \Gamma(Y(\vec{w}\{{}^{u_0}/_{v}\})) \bullet \{{}^{v}/_{u_0}\} & \text{if } u_0 \notin \vec{w} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Figure 5.6: Type substitutions.

Definition 48 (Type substitutions)**.** *For a mapping σ of the form $\{{}^{v}/_{\vec{u}}\}$ we define the substitutions $f \bullet \sigma$ on types and $\Gamma \bullet \sigma$ on type environments as in Fig. 5.6.*

When querying a typing environments on which a substitution is applied, we use the reverse substitution to retrieve the original entry, as recorded by [T-DEF]; then, we actually apply the substitution to the retrieved type. Note that we do not allow replaced variables to appear in the query.

Subject reduction guarantees that typeability is preserved by transitions. We need to distinguish between two cases, according to which participant moves: either the participant A under typing, or any other participant B. If the transition is done by A, then also its process type must take a transition, otherwise the type is preserved as is. In both cases, the substitution σ is applied to the type, to deal with possible variable fusions.

Theorem 49 (Subject reduction)**.** *If $\vdash_{\mathsf{A}} S \colon f$ with f honest, then:*

$$S \xrightarrow{\mathsf{A} : \pi, \sigma} S' \implies \exists f' \,.\, f \xrightarrow{\pi} f' \wedge \vdash_{\mathsf{A}} S' \colon f' \bullet \sigma \tag{9}$$

$$S \xrightarrow{\mathsf{B} : \pi, \sigma} S' \implies \vdash_{\mathsf{A}} S' \colon f \bullet \sigma \qquad (\textit{when } \mathsf{B} \neq \mathsf{A}) \tag{10}$$

Proof. See appendix A.11 on page 35. □

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 51 on the current page). To prove that, we first establish a progress result for systems. We write $S \vdash_s \phi$ when $S \equiv s[\gamma] \mid S''$ and $\gamma \vdash \phi$, for some S'' and γ.

Lemma 50 (System progress)**.** *For all systems S, if $\vdash_{\mathsf{A}} S \colon f$ with f honest, and $f \xrightarrow{\pi} f'$, then: (a)*

1. *if $\pi = \tau$, or $\pi = \mathsf{tell}_{\mathsf{B}} \downarrow_w c$, or $\pi = \mathsf{ask}_s \phi$ and $S \vdash_s \phi$,*

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

2. *if $\pi = \mathsf{do}_u \mathsf{a}$, then $\mathsf{a} \in RD^{\mathsf{A}}_u(S)$.*

Proof. See appendix A.12 on page 48. □

Theorem 51 (Progress)**.** *For all $S \equiv s[\mathsf{A} \textit{ says } c \mid \cdots] \mid S'$, if $\vdash_{\mathsf{A}} S \colon f$ with f honest, and $f(s) \stackrel{\mathsf{a}}{\Longrightarrow}_c^{\mathsf{A}}$, then $\mathsf{a} \in WRD^{\mathsf{A}}_s(S)$.*

Proof. See appendix A.13 on page 48. □

5.5 Type safety

The main result of this paper is the type safety of CO_2 processes (Th. 52). They ensure 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 52 (Type safety on processes)**.** *For all participants $\mathsf{A}[P]$ with P closed, if $\vdash P \colon f$ then $\mathsf{A}[P]$ is honest.*

Proof. See appendix A.14 on page 49. □

$$D_{X_M} = \left\{ \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \mathsf{do}_x\,\overline{\mathsf{ship\text{-}a}} \colon \lambda u\,.\,[\mathsf{do}_x\,\overline{\mathsf{ship\text{-}a}}]_u = f^1_{X_M}}\,[\text{T-Sum}]}{\vdash \mathsf{ask}_x\,\overline{\mathsf{ship\text{-}a}}?\,.\,\mathsf{do}_x\,\overline{\mathsf{ship\text{-}a}} \colon \lambda u\,.\,[\mathsf{ask}_x\,\overline{\mathsf{ship\text{-}a}}?]_u\,.\,f^1_{X_M}(u) = f^2_{X_M}}\,[\text{T-Sum}]}{\vdash \mathsf{do}_x\,\mathsf{pay}\,.\,\mathsf{ask}_x\,\overline{\mathsf{ship\text{-}a}}?\,.\,\mathsf{do}_x\,\overline{\mathsf{ship\text{-}a}} \colon \lambda u\,.\,[\mathsf{do}_x\,\mathsf{pay}]_u\,.\,f^2_{X_M}(u) = f^3_{X_M}}\,[\text{T-Sum}]}{\vdash P_{X_M} = \mathsf{do}_x\,\overline{\mathsf{ok}}\,.\,\mathsf{do}_x\,\mathsf{pay}\,.\,\mathsf{ask}_x\,\overline{\mathsf{ship\text{-}a}}?\,.\,\mathsf{do}_x\,\overline{\mathsf{ship\text{-}a}} \colon \lambda u\,.\,[\mathsf{do}_x\,\overline{\mathsf{ok}}]_u\,.\,f^3_{X_M}(u) = f_{X_M}}\,[\text{T-Sum}]} \right.$$

$$\dfrac{\dfrac{\dfrac{\dfrac{X_M(x) \stackrel{def}{=} P_{X_M} \quad D_{X_M}}{\vdash X_M(x) \colon f_{X_M}}\,[\text{T-Def}] \quad \dfrac{X_M(x) \stackrel{def}{=} P_{X_M} \quad D_{X_M}}{\vdash X_M(x) \colon f_{X_M}}\,[\text{T-Def}]}{\vdash \mathsf{do}_x\,\mathsf{a}\,.\,X_m(x) + \mathsf{do}_x\,\mathsf{b}\,.\,X_m(x) \colon \lambda u\,.\,[\mathsf{do}_x\,\mathsf{a}]_u\,.\,f_{X_M}(u) + [\mathsf{do}_x\,\mathsf{b}]_u\,.\,f_{X_M}(u) = f^1_{P_M}}\,[\text{T-Sum}]}{\vdash \mathsf{tell}_\mathsf{K}\downarrow_x c\,.\,(\mathsf{do}_x\,\mathsf{a}\,.\,X_M(x) \;+\; \mathsf{do}_x\,\mathsf{b}\,.\,X_M(x)) : \lambda u\,.\,[\mathsf{tell}_\mathsf{K}\downarrow_x c]_u\,.\,f^1_{P_M}(u) = f^2_{P_M}}\,[\text{T-Sum}] \qquad \dfrac{}{f^2_{P_M}(x)\ \cancel{\text{honest}}}}{\not\vdash (x)\,(\mathsf{tell}_\mathsf{K}\downarrow_x c\,.\,(\mathsf{do}_x\,\mathsf{a}\,.\,X_M(x) + \mathsf{do}_x\,\mathsf{b}\,.\,X_M(x))) = P_M \colon f_M = f^2_{P_M}\{f^2_{P_M}(*)/x\}}\,[\text{T-Del}]$$

Figure 5.7: Tentative typing derivation for the malicious food store. P_{X_M} is the body of $X_M(x)$ in Sect. 1.2, and its typing derivation D_{X_M} is used in the (tentative) typing derivation of P_M.

We conclude by checking the type safety of the food store example in Sect. 1.2: we analyse the malicious implementation (Ex. 53), the non-malicious one (Ex. 54), and finally the honest one (Ex. 55).

Example 53. *In Fig. 5.7 we give the (tentative) typing of the malicious food store process P_M with*

$$f^2_{P_M}(x) = \langle c \rangle\,.\,(\mathsf{a}\,.\,f_{X_M}(x) \;+\; \mathsf{b}\,.\,f_{X_M}(x))$$

where $f_{X_M}(x) = \overline{\mathsf{ok}}\,.\,\mathsf{pay}\,.\,\tau_{\overline{\mathsf{ship\text{-}a}}?}\,.\,\overline{\mathsf{ship\text{-}a}}$

The typing of P_M fails because [T-Del] *requires $f^2_{P_M}(x)$ to be honest, which is not the case. In fact, if the customer selects* b*, $f^2_{P_M}(x)$ takes the following transitions:*

$$\begin{aligned} f^2_{P_M}(x) \quad &\xrightarrow{\langle c \rangle} \mathsf{a}\,.\,f_{X_M}(x) + \mathsf{b}\,.\,f_{X_M}(x) \xrightarrow{\mathsf{b}} f_{X_M}(x) \\ &\xrightarrow{\overline{\mathsf{ok}}} \mathsf{pay}\,.\,\tau_{\overline{\mathsf{ship\text{-}a}}?}\,.\,\overline{\mathsf{ship\text{-}a}} \xrightarrow{\mathsf{pay}} \tau_{\overline{\mathsf{ship\text{-}a}}?}\,.\,\overline{\mathsf{ship\text{-}a}} \xrightarrow{\tau_{\overline{\mathsf{ship\text{-}a}}?}} \overline{\mathsf{ship\text{-}a}} \xrightarrow{\overline{\mathsf{ship\text{-}a}}} \mathbf{0} \end{aligned}$$

Correspondingly, the abstract process $(\emptyset, f^2_{P_M}(x))$ can evolve as:

$$\begin{aligned} \big(\emptyset, f^2_{P_M}(x)\big) \quad &\to (\{c\}, \mathsf{a}\,.\,f_{X_M}(x) + \mathsf{b}\,.\,f_{X_M}(x)) \xrightarrow[\text{[A-Fuse]}]{} (c, \mathsf{a}\,.\,f_{X_M}(x) + \mathsf{b}\,.\,f_{X_M}(x)) \\ &\xrightarrow[\text{[A-Ctx]}]{} \big(\mathit{ready}\ \mathsf{b}\,.\,\big(\overline{\mathsf{ok}}\,;\,\mathsf{pay}\,.\,\overline{\mathsf{ship\text{-}b}} \oplus \overline{\mathsf{no}}\big), \mathsf{a}\,.\,f_{X_M}(x) + \mathsf{b}\,.\,f_{X_M}(x)\big) \\ &\to \big(\overline{\mathsf{ok}}\,;\,\mathsf{pay}\,.\,\overline{\mathsf{ship\text{-}b}} \oplus \overline{\mathsf{no}}, f_{X_M}(x)\big) \\ &= \Big(\overline{\mathsf{ok}}\,;\,\mathsf{pay}\,.\,\overline{\mathsf{ship\text{-}b}} \oplus \overline{\mathsf{no}}, \overline{\mathsf{ok}}.\mathsf{pay}.\tau_{\overline{\mathsf{ship\text{-}a}}?}.\overline{\mathsf{ship\text{-}a}}\Big) \\ &\to \Big(\mathsf{pay}\,.\,\overline{\mathsf{ship\text{-}b}}, \mathsf{pay}\,.\,\tau_{\overline{\mathsf{ship\text{-}a}}?}\,.\,\overline{\mathsf{ship\text{-}a}}\Big) \\ &\xrightarrow[\text{[A-Ctx]}]{} \Big(\mathit{ready}\ \mathsf{pay}\,.\,\overline{\mathsf{ship\text{-}b}}, \mathsf{pay}\,.\,\tau_{\overline{\mathsf{ship\text{-}a}}?}\,.\,\overline{\mathsf{ship\text{-}a}}\Big) \\ &\to \Big(\overline{\mathsf{ship\text{-}b}}, \tau_{\overline{\mathsf{ship\text{-}a}}?}\,.\,\overline{\mathsf{ship\text{-}a}}\Big) \;\to \big(\overline{\mathsf{ship\text{-}b}}, \overline{\mathsf{ship\text{-}a}}\big) \end{aligned}$$

Notice that, in the last step, we have that $\overline{\mathsf{ship\text{-}a}}$ is not ready for $\overline{\mathsf{ship\text{-}b}}$, hence $\mathsf{A}[P_M]$ is not honest.

We consider now the non-malicious food store process.

Example 54. *If we try to type P_N, we incur in problems similar to the previous example. In fact, the top-level delimitation of x requires applying rule* [T-Del]*, which mandates the related channel type to be honest. Such type is:*

$$\begin{aligned} f^1_{P_N}(x) &= \tau\,.\,\tau_?\,.\,\langle c \rangle\,.\,\big(\mathsf{a}\,.\,\overline{\mathsf{ok}}\,.\,f_{X_N}(x) \;+\; \mathsf{b}\,.\,f_{Y_N}(x)\big) \\ \textit{where}\ \ f_{X_N}(x) &= \mathsf{pay}\,.\,\Big(\tau_{\overline{\mathsf{ship\text{-}a}}?}\,.\,\overline{\mathsf{ship\text{-}a}} + \tau_{\overline{\mathsf{ship\text{-}b}}?}\,.\,\overline{\mathsf{ship\text{-}b}}\Big) \\ f_{Y_N}(x) &= \tau_?\,.\,\big(\overline{\mathsf{ok}}\,.\,f_{X_N}(x) + \tau\,.\,\overline{\mathsf{no}}\big) \end{aligned}$$

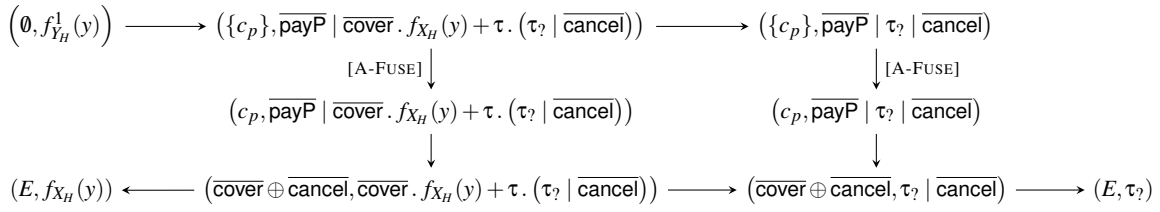

Figure 5.8: Abstract process reductions for the honest food store ($Y_H(x)$ sub-process). The graph omits the $\tau_?$ channel type transitions.

In case of b *orders,* $f^1_{P_N}(x)$ *takes the following transitions:*

$$f^1_{P_N}(x) \xrightarrow{\tau}\xrightarrow{\tau_?}\xrightarrow{\langle c\rangle} \mathsf{a}\,.\,\overline{\mathsf{ok}}\,.\,f_{X_N}(x)+\mathsf{b}\,.\,f_{Y_N}(x) \xrightarrow{\mathsf{b}} f_{Y_N}(x) \xrightarrow{\tau_?} \overline{\mathsf{ok}}\,.\,f_{X_N}(x)+\tau\,.\,\overline{\mathsf{no}} \to \ldots$$

The corresponding transitions of the abstract process are:

$$\begin{aligned}\left(\emptyset, f^1_{P_N}(x)\right) &\to^* \left(c, \mathsf{a}\,.\,\overline{\mathsf{ok}}\,.\,f_{X_N}(x)+\mathsf{b}\,.\,f_{Y_N}(x)\right) \to^* \left(\overline{\mathsf{ok}}\,;\,\mathsf{pay}\,.\,\overline{\mathsf{ship\text{-}b}} \oplus \overline{\mathsf{no}}, f_{Y_N}(x)\right)\\ &= \left(\overline{\mathsf{ok}}\,;\,\mathsf{pay}\,.\,\overline{\mathsf{ship\text{-}b}} \oplus \overline{\mathsf{no}}, \tau_?.\left(\overline{\mathsf{ok}}\,.\,f_{X_N}(x)+\tau.\overline{\mathsf{no}}\right)\right)\end{aligned}$$

In the last step, we have that $\tau_?.\left(\overline{\mathsf{ok}}\,.\,f_{X_N}(x)+\tau.\overline{\mathsf{no}}\right)$ *is not ready for* $\overline{\mathsf{ok}}\,;\,\mathsf{pay}\,.\,\overline{\mathsf{ship\text{-}b}} \oplus \overline{\mathsf{no}}$*. Indeed, the prefix* $\tau_?$ *is not collapsed by* $\Rightarrow$*. Therefore,* $f^1_{P_N}(x)$ *is not honest.*

We also have a similar negative result for the channel type:

$$f^1_{P_N}(y) = \langle c_p\rangle\,.\,\overline{\mathsf{payP}}\,.\,\tau\,.\,(\tau_?\,.\,\tau_?\,.\,f_{X_N}(y) \;+\; \tau_?\,.\,f_{Y_N}(y))$$

Here, the unavoidable $\tau_?$ *actions make the* $f^1_{P_N}(y)$ *reduct non-ready for the reduct of* c_p *after* $\overline{\mathsf{payP}}$*. As a result,* P_N *is untypeable.*

Example 55. *Finally, let us consider the last food store implementation,* P_H*. Let* P_{Y_H} *be the process under delimitation of* (y) *in* $Y_H(x)$*. Processes* P_{Y_H} *and* $X_H(x)$ *have the following channel types:*

$$\begin{aligned} f_{Y_H}(y) &= \langle c_p\rangle\,.\,\overline{\mathsf{payP}} \mid \left(\overline{\mathsf{cover}}\,.\,f_{X_H}(y)+\tau\,.\,\left(\tau_? \mid \overline{\mathsf{cancel}}\right)\right)\\ f_{Y_H}(x) &= \tau\,.\,\tau_? \;\mid\; (\tau_?\,.\,f_{X_H}(x) \;+\; \tau\,.\,(\overline{\mathsf{no}} \mid \tau_?))\\ f_{X_H}(x) &= \overline{\mathsf{ok}}\,.\,\mathsf{pay}\,.\,\left(\tau_{\overline{\mathsf{ship\text{-}a}?}}\,.\,\overline{\mathsf{ship\text{-}a}} + \tau_{\overline{\mathsf{ship\text{-}b}?}}\,.\,\overline{\mathsf{ship\text{-}b}}\right)\\ f_{X_H}(y) &= f_{X_H}(*) = \tau_?\,.\,\tau_?\,.\,(\tau_?\,.\,\tau_? + \tau_?\,.\,\tau_?)\end{aligned}$$

The relevant transitions of the abstract processes above are shown in Fig. 5.8. By observing the abstract transitions we detect that $f_{Y_H}(y)$ *is honest, hence we can apply rule* [T-DEL] *to derive from the typing* $\vdash P_{Y_H} : f_{Y_H}$ *a type for* $Y_H(x)$*.*

The process under delimitation in P_H *is typeable as well, and it has the following channel type:*

$$f^1_{P_H}(x) = \langle c\rangle\,.\,(\mathsf{a}\,.\,f_{X_H}(x) \;+\; \mathsf{b}\,.\,f_{Y_H}(x))$$

By examining all the states of the transitions of the abstract process we obtain that $f^1_{P_H}(x)$ *is honest. To do that, it is crucial to ensure that the relation* $\vdash_\#$ *allows* $\Rightarrow$ *to collapse the abstract prefixes* $\tau_{\overline{\mathsf{ship\text{-}a}?}}$ *and* $\tau_{\overline{\mathsf{ship\text{-}b}?}}$*. Since* P_H *is typeable, type safety guarantees that the food store is honest.*

6 Concluding Remarks and Related Work

Building on CO_2 we gave a type system that allows for the static checking of *honesty* of systems. The channels onto which a CO_2 process interacts are typed with a behavioural type. Such type abstracts the actual prefixes of the process while mimicking the non-deterministic and parallel branching of the process as well as its recursive behaviour. Our

typing enjoys the subject reduction (Th. 49) and progress properties (Th. 51). More importantly, type safety establishes honesty of typeable processes, that is typeable processes honour their contracts in all contexts.

The process calculus CO_2 has been introduced in [1], and in [3] it has been instantiated to a theory of bilateral contracts inspired by [10]. We refer the reader to [3] for a comparison between our contract theory and the one in [10]. In [3] a process A is honest when, for each session she is engaged in, A is not definitely *culpable*. That is, A eventually performs the actions her contract prescribes. The definition of honesty we adopt here is based on readiness rather than culpability and we conjecture that it is equivalent to the notion of honesty in [3]. The main advantage of this novel 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 [4] (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 [14] the progress property is checked only when participants engage at most in one session at a time. The type system for honesty we give here allows participants to interleave many sessions as done in [13]. A crucial difference with respect to [13] 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 typed 'in isolation' and to establish the honesty of a participant A our typing discipline only imposes that the surrunding context is A-free.

Other approaches deal with safety properties, by generating monitors that check at runtime the interactions of processes against their local contract (e.g.,[12, 11]).

The problem of checking if a contract c representing the behaviour of a service conforms to a role r of a given choreography H has been investigated in [5]. Under suitable well-formed conditions, 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 [6] to define contract-based composition of services. A main difference with respect to our approach is that [5, 6] do not consider conformance in the presence of dishonest participants. Actually, these papers focus on using the testing pre-order to determine if the abstract behaviour of a service (i.e., its contract), comply with a role of the choreography. Instead, we are interested in establishing weather a process abides by its own contract regardless its execution context.

Contracts for service-level agreement have been modelled in [8] as constraint-semirings. Such model is used in [7] for compiling clients and services so to guaranteed that, whener compatible, they progress harmoniously. This is orthogonal to our approach since our aim is not to rule out "inconsistent" executions, rather to blame participants that misbehave.

A Proofs

A.1 Additional Definitions and Lemmata

This section contains some definitions and auxiliary lemmata which are not part of the main treatment of this technical report, but are used in the main proofs in the rest of the appendix.

Definition 56 (Free names/variables of a system wrt. a participant)**.** *For all participants* A*, and for all* $\mathsf{B}, \mathsf{B}' \neq \mathsf{A}$*, the* free names/variables of a system wrt. a participant A *are defined as follows:*

$$\begin{array}{rlrl}
\mathrm{fnv}_\mathsf{A}(S \mid S') & = \mathrm{fnv}_\mathsf{A}(S) \cup \mathrm{fnv}_\mathsf{A}(S') & \mathrm{fnv}_\mathsf{A}((u)S) & = \mathrm{fnv}_\mathsf{A}(S) \setminus \{u\} \\
\mathrm{fnv}_\mathsf{A}(\mathsf{A}[P]) & = \mathrm{fnv}(P) & \mathrm{fnv}_\mathsf{A}(\mathsf{B}[P]) & = \emptyset \\
\mathrm{fnv}_\mathsf{A}(\mathsf{C}[\downarrow_x \mathsf{A}\ \mathit{says}\ c]) & = \{x\} & \mathrm{fnv}_\mathsf{A}(\mathsf{B}[\downarrow_x \mathsf{C}\ \mathit{says}\ c]) & = \emptyset \\
\mathrm{fnv}_\mathsf{A}(s[\mathsf{A}\ \mathit{says}\ c \mid \mathsf{B}\ \mathit{says}\ d]) & = \{s\} & \mathrm{fnv}_\mathsf{A}(s[\mathsf{B}\ \mathit{says}\ c \mid \mathsf{B}'\ \mathit{says}\ d]) & = \emptyset
\end{array}$$

The dual of the function above is defined as:

$$\overline{\mathrm{fnv}_\mathsf{A}}(S) = \bigcup_{\mathsf{B} \neq \mathsf{A}} \mathrm{fnv}_\mathsf{B}(S)$$