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:

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

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.:

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

Here, 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:

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

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

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

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:

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

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:*

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

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:*

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

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)

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

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

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

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

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

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

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

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:

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

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

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

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

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:*

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

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.*

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

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:*

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

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:*

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

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.:

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

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

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

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

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

$$\gamma \xrightarrow{\mathsf{A}\ \textit{says}\ \mathsf{a}}\!\!\gg \mathsf{A}\ \textit{says}\ c' \mid \mathsf{B}\ \textit{says}\ d' \implies c \xrightarrow{\mathsf{a}}\!\!\gg_\sharp c'$$
$$\gamma \xrightarrow{\mathsf{B}\ \textit{says}\ \mathsf{b}}\!\!\gg \mathsf{A}\ \textit{says}\ c' \mid \mathsf{B}\ \textit{says}\ d' \implies c \xrightarrow{ctx}\!\!\gg_\sharp c'$$

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 $\overset{\mathsf{a}}{\Longrightarrow}_c^{\mathsf{A}}$ (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:*

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

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. $(\emptyset, \langle c_i \rangle \mid \overline{\mathsf{a}})$. *Similar to the previous case.*
3. $(\{c_i\}, \tau\,.\,\overline{\mathsf{a}})$. *Nothing to check, because no contracts have been stipulated yet.*
4. $(\{c_i\}, \overline{\mathsf{a}})$. *Similar to the previous case.*
5. $(c_i, \tau\,.\,\overline{\mathsf{a}})$. *We have that $\tau\,.\,\overline{\mathsf{a}}$ is ready for c_i, because for $\{\overline{\mathsf{a}}\} \in RS(c_i) = \{\{\overline{\mathsf{a}}\}, \{\overline{\mathsf{b}}\}\}$, we have $\tau\,.\,\overline{\mathsf{a}} \overset{\overline{\mathsf{a}}}{\Longrightarrow}$.*
6. $(c_i, \overline{\mathsf{a}})$. *We have that $\overline{\mathsf{a}}$ 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: $z \notin \mathrm{fnv}(P) \wedge \Gamma \vdash P \colon f \implies f(z) = f(*)$.*

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:*

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

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:*

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

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 $f(s) \stackrel{\mathsf{a}}{\Longrightarrow}^{\mathsf{A}}_{c}$, 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. □

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

$$\dfrac{\dfrac{\dfrac{\dfrac{X_M(x) \stackrel{def}{=} P_{X_M} \quad D_{X_M}}{\vdash X_M(x)\colon f_{X_M}}\ {\scriptstyle [\text{T-Def}]} \quad \dfrac{X_M(x) \stackrel{def}{=} P_{X_M} \quad D_{X_M}}{\vdash X_M(x)\colon f_{X_M}}\ {\scriptstyle [\text{T-Def}]}}{\vdash \mathsf{do}_x\,\mathsf{a}\,.\,X_m(x) + \mathsf{do}_x\,\mathsf{b}\,.\,X_m(x)\colon \lambda u\,.\,[\mathsf{do}_x\,\mathsf{a}]_u\,.\,f_{X_M}(u) + [\mathsf{do}_x\,\mathsf{b}]_u\,.\,f_{X_M}(u) = f^1_{P_M}}\ {\scriptstyle [\text{T-Sum}]}}{\vdash \mathsf{tell}_{\mathsf{K}}\downarrow_x c\,.\,(\mathsf{do}_x\,\mathsf{a}\,.\,X_M(x)\ +\ \mathsf{do}_x\,\mathsf{b}\,.\,X_M(x))\colon \lambda u\,.\,[\mathsf{tell}_{\mathsf{K}}\downarrow_x c]_u\,.\,f^1_{P_M}(u) = f^2_{P_M}}\ {\scriptstyle [\text{T-Sum}]} \qquad \dfrac{}{f^2_{P_M}(x)\ \cancel{\text{honest}}}}{\nvdash (x)\,(\mathsf{tell}_{\mathsf{K}}\downarrow_x c\,.\,(\mathsf{do}_x\,\mathsf{a}\,.\,X_M(x) + \mathsf{do}_x\,\mathsf{b}\,.\,X_M(x))) = P_M\colon f_M = f^2_{P_M}\{^{f^2_{P_M}(*)}/_x\}}\ {\scriptstyle [\text{T-Del}]}$$

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:*

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

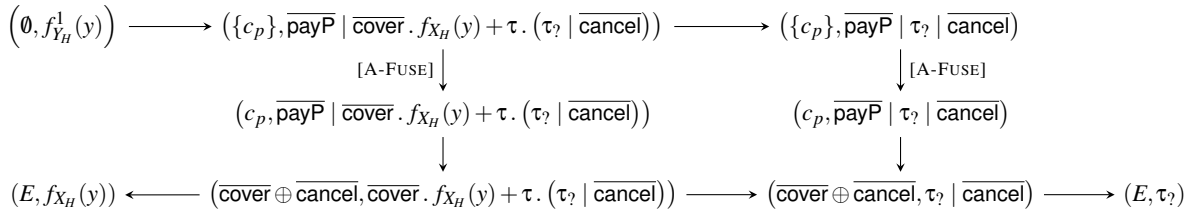

Figure 5.8: Abstract process reductions for the honest food store ($Y_H(x)$ sub-process). The graph omits the $\tau_?$ channel type transitions.

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:

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

i.e., it consists in all the free names/variables of a system S, except *those retained by a participant* A.

Lemma 57 (Channel type moves and realization)**.** *For all channel types T and contracts c, and for $\alpha \in \{\tau, \tau_?, \tau_\phi, \langle c' \rangle\}$:*

$$T \text{ realizes } c \ \wedge \ T \xrightarrow{\alpha} T' \implies T' \text{ realizes } c$$

Proof. By definition 25 on page 11, T realizes c when the abstract process (c, T) is honest, i.e. when $(c, T) \to^* (c', T'')$ implies that T'' is ready for c'. By definition 21 on page 10 (rules [A-TELL2] and [A-TAU2]), the form of α in the hypotheses reduces this proof to the analysis of the abstract process transition $(c, T) \to (c, T')$. Since (c, T) is honest, we have that $(c, T) \to (c, T') \to^* (c', T'')$ implies that T'' is ready for c'. This, by definition, implies that (c, T') is honest as well, and thus that T' realizes c. □

Lemma 58 (System typing and non-free names/variables)**.** *For all systems S:*

$$z \notin \text{fnv}(S) \ \wedge \ \vdash_{\mathsf{A}} S \colon f \implies f(z) = f(*)$$

Proof. This is the system counterpart of lemma 34 on page 13. When $z \notin \text{fnv}(S)$, we have two cases:

- z does not appear at all S. Then, for all CO_2 prefixes π, we have $[\pi]_z = [\pi]_*$ (see the prefix abstraction mapping in definition 29 on page 12). And thus, we necessarily have $f(z) = f(*)$;
- $S \equiv (z)S'$, with $z \in \text{fnv}(S')$ and $\vdash_{\mathsf{A}} P' \colon f'$. Then, considering that typing is preserved by structural equivalence for systems (lemma 44 on page 14), the typing derivation of P has the form:

$$\frac{\vdash_{\mathsf{A}} S' \colon f' \quad f'(z) \text{ honest}}{\vdash_{\mathsf{A}} S \equiv (z)S' \colon f'\{^{f'(*)}/_z\} = f} \ [\text{T-SDEL2}]$$

  Hence, $f(z) = f'\{^{f'(*)}/_z\}(z) = f'(*) = f(*)$.

□

Lemma 59 (Honesty of $f(*)$ (for processes))**.** *For all CO_2 processes P, $\Gamma \vdash P \colon f \implies f(*)$ is honest.*

Proof. We proceed by induction on the typing derivation of $\vdash P \colon f$.

- [T-SUM]. We have:

$$\frac{\Gamma \vdash P_i \colon f_i \quad \forall i \in I}{\Gamma \vdash P = \sum_{i \in I} \pi_i \,.\, P_i \colon \lambda u \,.\, \sum_{i \in I} [\pi_i]_u \,.\, f_i(u) = f} \ [\text{T-SUM}]$$

  By applying the induction hypothesis on the rule premises, let $f_i(*)$ be honest, $\forall i \in I$. Since the prefix abstraction $[\pi_i]_*$ (definition 29 on page 12) cannot possibly result in a new new contract advertisement $\langle c_i \rangle$, we have that the channel type $[\pi_i]_* \,.\, f_i(*)$ is honest. Hence, $\sum_{i \in I} [\pi_i]_* \,.\, f_i(*) = (\lambda u \,.\, \sum_{i \in I} [\pi_i]_u \,.\, f_i(u))\,(*) = f(*)$ is honest;
- [T-PAR], [T-VAR]. Trivial, by applying the induction hypothesis on the rule premises;
- [T-DEL]. We have:

$$\frac{\Gamma_{\neq u} \vdash P' \colon f \quad f(u) \text{ honest}}{\Gamma \vdash P = (u)P' \colon f\{^{f(*)}/_u\}} \ [\text{T-DEL}]$$

  By lemma 75 on page 26, the rule premise $\Gamma_{\neq u} \vdash P \colon f$ implies that the same typing can be obtained with a larger environment, i.e. $\Gamma \vdash P \colon f$. Hence, the theorem is demonstrated by applying the induction hypothesis;
- [T-DEF]. We have a typing derivation of the form:

$$\frac{X(\vec{u}) \stackrel{\text{def}}{=} P' \quad D_1 = \left\{ \dfrac{\dfrac{\vdots}{\Gamma_1 \vdash \cdots}\ [\cdots]}{\Gamma\{^f/_{X(\vec{v})}\} = \Gamma_1 \vdash P'\{^{\vec{v}}/_{\vec{u}}\} \colon f}\ [\cdots] \right.}{\Gamma \vdash P = X(\vec{v}) \colon f} \ [\text{T-DEF}]$$

  We need to show that the typing environments in the derivation will never map $X(\vec{v})$ to a process type g such that $g(*)$ contains a contract advertisement $\langle c \rangle$. We proceed by induction on the number of applications of rule [T-DEF] for $X(\vec{v})$.

– 1 application of [T-DEF] for $X(\vec{v})$ (base case). The type of $X(\vec{v})$ in the Γ_1 environment is never replaced along the derivation branch D_1, and such typing only contains instances of rules [T-SUM], [T-PAR], [T-VAR] and [T-DEL]. Considering that recursion in CO_2 processes is guarded (see section 3 on page 4), each instance of [T-VAR] can only appear above an instance of [T-SUM]. Hence, f will have the form:

$$f = \lambda u \;.\; [\pi_1]_u \,.\, \cdots \,.\, f(u) \;\mid\; [\pi_2]_u \,.\, \cdots \,.\, f(u) \;\mid\; [\pi_3]_u \,.\, \cdots$$

where each $f(u)$ is introduced by an instance of [T-VAR] (if present), each $\mid$ by [T-PAR], and each prefix abstractions $[\pi_i]_u$ by [T-SUM]. Furthermore, as shown above, we know that [T-SUM] does not allow $[\pi_i]_u$ such that the channel type $f(*)$ contains contract advertisement prefixes $\langle c_i \rangle$. Hence, $f(*)$ is trivially honest, because it cannot advertise any contract;

– $n+1$ applications of [T-DEF] for $X(\vec{v})$ (inductive step). Given a derivation with n instances of [T-DEF] for $X(\vec{v})$, a derivation with $n+1$ instances can only be obtained by replacing an instance of [T-VAR] with a further derivation D_{n+1} based on [T-DEF][1]. Such replacement has the form:

$$\dfrac{\dfrac{\Gamma_n(X(\vec{v})) = f_n}{\Gamma_n \vdash X(\vec{v}) \colon f_n}\,{\scriptstyle [\text{T-VAR}]}}{\vdots}\,{\scriptstyle [\cdots]} \quad \mapsto \quad \dfrac{\dfrac{X(\vec{u}) \stackrel{\text{def}}{=} P' \quad D_{n+1} = \left\{ \dfrac{\vdots}{\Gamma_n\{f_{n+1}/X(\vec{v})\} = \Gamma_{n+1} \vdash P\{\vec{v}/\vec{u}\} \colon f_{n+1}}\,{\scriptstyle [\cdots]} \right.}{\Gamma_n \vdash X(\vec{v}) \colon f_{n+1}}\,{\scriptstyle [\text{T-DEF}]}}{\vdots}\,{\scriptstyle [\cdots]}$$

Here, f_{n+1} replaces the occurrences of f_n downwards in the typing derivation[2], and the branch D_{n+1} does not replace the typing of $X(\vec{v})$ in Γ_{n+1} with other applications of [T-DEF] — just like the derivation D_1 in the base case above. Hence, since D_{n+1} does not introduce contract advertisements in $f_{n+1}(*)$, nor are they introduced (by inductive hypothesis) in other typing environments along the rest of the derivation, we conclude that $f(*)$ is trivially honest, because it cannot advertise any contract.

□

Lemma 60 (Honesty of $f(*)$ (for systems))**.** *For all systems CO_2 systems S, $\vdash_{\mathsf{A}} S \colon f \implies f(*)$ is honest.*

Proof. This is the system counterpart of lemma 59 on the preceding page. We proceed by induction on the typing derivation of $\vdash_{\mathsf{A}} S \colon f$.

- [T-SA]. We have:

$$\dfrac{\emptyset \vdash P \colon f}{\vdash_{\mathsf{A}} S = \mathsf{A}[P] \colon f}\,{\scriptstyle [\text{T-SA}]}$$

  Thus, $f(*)$ is honest by lemma 59 on the facing page.
- [T-SDEL2], [T-SPAR2]. Trivial, by applying the induction hypothesis on the rule premises.

□

Lemma 61 (Structural equivalence and substitutions (for processes))**.** *For all processes P, P' and for all substitutions σ,*

$$P \equiv P' \implies P\sigma \equiv P'\sigma$$

Proof. We examine all the different cases of structural equivalence between P and P'.

- $P = Q \mid \mathbf{0} \equiv Q = P'$. Trivial;
- $P = Q \mid Q' \equiv Q' \mid Q = P'$. Trivial;
- $P = (Q \mid Q') \mid Q'' \equiv Q \mid (Q' \mid Q'') = P'$. Trivial;
- $P = Q \mid (u)Q' \equiv (u)(Q \mid Q')$ if $u \notin \mathrm{fv}(Q) \cup \mathrm{fn}(Q) = P'$. Trivial;
- $P = (u)(v)Q \equiv (v)(u)Q = P'$. Trivial;
- $P = (u)Q \equiv Q$ if $u \notin \mathrm{fv}(Q) \cup \mathrm{fn}(Q) = P'$. Trivial.

□

[1] The presence of a [T-VAR] instance restricts the inductive step to recursive definitions of $X(\vec{v})$.

[2] This corresponds to the fact that the introduction of a further application of [T-DEF] amounts to an unfolding of the recursive process type.

Lemma 62 (Structural equivalence and substitutions (for systems))**.** *For all systems S, S' and for all substitutions σ,*

$$S \equiv S' \implies S\sigma \equiv S'\sigma$$

Proof. This is the system counterpart of lemma 61 on the previous page. We examine all the different cases of structural equivalence between S and S'.

- $S = (\vec{u})\mathsf{A}[(\vec{v})P] \equiv (\vec{u},\vec{v})\mathsf{A}[P] = S'$. Trivial;
- $S = \mathsf{A}[K] \mid \mathsf{A}[K'] \equiv \mathsf{A}[K \mid K'] = S'$. Trivial;
- $S = S_1 \mid \mathbf{0} \equiv S_1 = S'$. Trivial;
- $S = S_1 \mid S_2 \equiv S_2 \mid S_1 = S'$. Trivial;
- $S = (S_1 \mid S_2) \mid S_3 \equiv S_1 \mid (S_2 \mid S_3) = S'$. Trivial;
- $S = S_1 \mid (u)S_2 \equiv (u)(S_1 \mid S_2)$ if $u \notin \mathrm{fv}(S)_1 \cup \mathrm{fn}(S)_1 = S'$. Trivial;
- $S = (u)(v)S_1 \equiv (v)(u)S_1 = S'$. Trivial;
- $S = (u)S_1 \equiv S_1$ if $u \notin \mathrm{fv}(S)_1 \cup \mathrm{fn}(S)_1 = S'$. Trivial.

□

Lemma 63 (Substitution of restricted type)**.** *For all systems S, for all channel types T and for all $u \notin \mathrm{fnv}(S)$:*

$$\vdash_{\mathsf{A}} S \triangleright f \implies \vdash_{\mathsf{A}} S \triangleright f\{^{T}/_{u}\}$$

Proof. By induction on the typing derivation of $\vdash_{\mathsf{A}} S \triangleright f$, we have the following cases:

- [T-SAFREE0], [T-SAFREE1], [T-SAFREE2]: the substitution of f with $f\{^{T}/_{u}\}$ is immaterial;
- [T-SFZ1]: $x \in \mathrm{fv}(\mathsf{B}[\downarrow_x \mathsf{A}\ \textit{says}\ c])$, hence by hypothesis $u \neq x$, and so also in this case the substitution of f with $f\{^{T}/_{u}\}$ is immaterial;
- [T-SFZ2]: straightforward, by applying the induction hypothesis on the premises;
- [T-SFUSED]: $s \in \mathrm{fv}(s[\mathsf{A}\ \textit{says}\ c \mid \gamma])$, hence by hypothesis $u \neq s$, and so also in this case the substitution of f with $f\{^{T}/_{u}\}$ is immaterial;
- [T-SDEL1]: let $S = (v)S'$, with $u \notin \mathrm{fnv}((v)S')$. If we replace f with $f\{^{T}/_{u}\}$ in the conclusion of the rule, we have $\vdash_{\mathsf{A}} S \triangleright f\{^{T}/_{u}\}\{^{f\{^{T}/_{u}\}(*)}/_{u}\} = f\{^{f(*)}/_{u}\}$ in the premise. Thus, also in this case, the substitution of f with $f\{^{T}/_{u}\}$ is immaterial;
- [T-SPAR1]: let $S = S' \mid S''$. By hypothesis we have $u \notin \mathrm{fnv}(S') \cup \mathrm{fnv}(S'')$. We apply the induction hypothesis on both premises of the rule, i.e. $\vdash_{\mathsf{A}} S' \triangleright f\{^{T}/_{u}\}$ and $\vdash_{\mathsf{A}} S'' \triangleright f\{^{T}/_{u}\}$. By rule [T-SPAR1] we conclude that $\vdash_{\mathsf{A}} S' \mid S'' = S \triangleright f\{^{T}/_{u}\}$.

□

Lemma 64 (Exclusivity of typing)**.** *For all systems S, for all participants* A *and for all process types f, g,*

$$\vdash_{\mathsf{A}} S\colon f \implies \nvdash_{\mathsf{A}} S \triangleright g \tag{11}$$

$$\vdash_{\mathsf{A}} S \triangleright f \implies \nvdash_{\mathsf{A}} S\colon g \tag{12}$$

Proof. We proceed by induction on the typing derivation of S. For item 11, we have that $\vdash_{\mathsf{A}} S\colon f$ may be obtained through an instance of the following rules:

- [T-SA]. We have $S \equiv \mathsf{A}[P]$, such that $\vdash P\colon f$. Then S does not match any rule that would allow to obtain $\vdash_{\mathsf{A}} S \triangleright g$;
- [T-SDEL2]. We have $S \equiv (u)S_1$ and $\vdash_{\mathsf{A}} S_1\colon f'$, such that $f = f'\{^{f'(*)}/_{u}\}$. By applying the induction hypothesis, let $\nvdash_{\mathsf{A}} S_1 \triangleright g'$, for all g'. The only rule that could possibly allow the typing relation $\vdash_{\mathsf{A}} S \triangleright g$ is [T-SDEL1], which requires $\vdash_{\mathsf{A}} S_1 \triangleright g\{^{g(*)}/_{u}\}$ — thus contradicting the induction hypothesis;
- [T-SPAR2]. We have $S \equiv S_1 \mid S_2$, such that $\vdash_{\mathsf{A}} S_1\colon f$. By applying the induction hypothesis, let $\nvdash_{\mathsf{A}} S_1 \triangleright g$. The only rule that could possibly allow the typing relation $\vdash_{\mathsf{A}} S \triangleright g$ is [T-SDEL1], which requires $\vdash_{\mathsf{A}} S_1 \triangleright g$ — thus contradicting the induction hypothesis;

For item 12, we have that $\vdash_{\mathsf{A}} S \triangleright f$ may be obtained through an instance of the following rules:

- [T-SAFREE0], [T-SAFREE1], [T-SAFREE2], [T-SAFREE3]. When S matches one of those rules, it cannot possibly be typed by any of the rules allowing $\vdash_{\mathsf{A}} S\colon g$;
- [T-SFZ1], [T-SFZ2], [T-SFUSED]. Same as above;

- [T-SDEL1]. We have $S \equiv (u)S_1$ and $\vdash_{\mathsf{A}} S_1 \rhd f' = f\{^{f(*)}/_{u}\}$. By applying the induction hypothesis, let $\nvdash_{\mathsf{A}} S_1 : g'$, for all g'. The only rule that could possibly allow the typing relation $\vdash_{\mathsf{A}} S : g$ is [T-SDEL2], which requires $\vdash_{\mathsf{A}} S_1 : g'$ such that $g = g'\{^{g'(*)}/_{u}\}$ — thus contradicting the induction hypothesis;
- [T-SPAR1]. We have $S \equiv S_1 \mid S_2$, such that $\vdash_{\mathsf{A}} S_1 \rhd f$ and $\vdash_{\mathsf{A}} S_2 \rhd f$. By applying the induction hypothesis, let $\nvdash_{\mathsf{A}} S_2 : g$. The only rule that could possibly allow the typing relation $\vdash_{\mathsf{A}} S : g$ is [T-SPAR2], which requires $\vdash_{\mathsf{A}} S_2 : g$ — thus contradicting the induction hypothesis;

□

Lemma 65 (Participants, processes and typing (I))**.** *For all systems S,*

$$\exists f . \vdash_{\mathsf{A}} S : f \implies \exists \vec{v}, S_0, S_1 \ . \ S \equiv (\vec{v})\,(\mathsf{A}[P] \mid S_0) \mid S_1$$

(i.e., when the typing relation $\vdash_{\mathsf{A}} S : f$ holds, A *with her CO_2 process P appears in S).*

Proof. We proceed by induction on the typing derivation of S. $\vdash_{\mathsf{A}} S : f$ may be obtained through an instance of the following rules:

- [T-SA]. The rule application confirms the thesis, with $\vec{v} = \emptyset$ and $S_0 = S_1 = \mathbf{0}$;
- [T-SDEL2]. We have $S \equiv (u_0)S'$ and $\vdash_{\mathsf{A}} S' : f'$, such that $f = f'\{^{f'(*)}/_{u_0}\}$. By applying the induction hypothesis, let $S' \equiv (\vec{u})\,(\mathsf{A}[P] \mid S_0)$; then, the rule application confirms the thesis, with $\vec{v} = u_o\vec{u}$ and $S_1 = \mathbf{0}$;
- [T-SPAR2]. We have $S \equiv S' \mid S_1$, such that $\vdash_{\mathsf{A}} S' : f$ and $\vdash_{\mathsf{A}} S_1 \rhd f$. By applying the induction hypothesis, let $S' \equiv (\vec{u})\,(\mathsf{A}[P] \mid S_0)$; then, the rule application confirms the thesis.

□

Lemma 66 (Participants, processes and typing (II))**.** *For all systems S, and for all process types f,*

$$\vdash_{\mathsf{A}} S \rhd f \implies \forall \vec{v}, S_0, S_1 \ . \ S \not\equiv (\vec{v})\,(\mathsf{A}[P] \mid S_0) \mid S_1$$

(i.e., when the typing relation $\vdash_{\mathsf{A}} S \rhd f$ holds, A *with her CO_2 process P does* not *appear in S).*

Proof. We proceed by induction on the typing derivation of S. $\vdash_{\mathsf{A}} S : f$ may be obtained through an instance of the following rules:

- [T-SAFREE0], [T-SAFREE1], [T-SAFREE2], [T-SAFREE3]. None of the rules could be triggered if $S \equiv (\vec{v})\,(\mathsf{A}[P] \mid S_0) \mid S_1$;
- [T-SFZ1], [T-SFZ2], [T-SFUSED]. Same as above;
- [T-SDEL1]. We have $S \equiv (u)S_1$ and $\vdash_{\mathsf{A}} S' \rhd f' = f\{^{f(*)}/_{u}\}$. By applying the induction hypothesis let $\nexists \vec{z}, S'', S''' \ . \ S' \equiv (\vec{z})\,(\mathsf{A}[P] \mid S'') \mid S'''$. But then, if S' cannot have such structure (and thus $\mathsf{A}[P]$ cannot appear in S'), then $(u)S' \equiv S$ cannot have it either. Hence, the theorem holds;
- [T-SPAR1]. We have $S \equiv S' \mid S''$, such that $\vdash_{\mathsf{A}} S' \rhd f$ and $\vdash_{\mathsf{A}} S'' \rhd f$. By applying the induction hypothesis, let the theorem hold for the structures of both S' and S'' (and thus, we have that $\mathsf{A}[P]$ cannot appear in S' nor in S''). Then, we have $S' \mid S'' \equiv S \not\equiv (\vec{v})\,(\mathsf{A}[P] \mid S_0) \mid S_1$.

□

Lemma 67 (Participants, processes, typing and ordering)**.** *For all systems S and for all participants* $\mathsf{A}[P]$ *such that* $\vdash P : f_P$*:*

$$\vdash_{\mathsf{A}} S : f \implies \exists \vec{v}, S_0, S_1 \ . \ S \equiv (\vec{v})\,(\mathsf{A}[P] \mid S_0) \mid S_1 \ \wedge \ f \sqsubseteq f_P$$

Proof. The left element in the conjunction is implied by lemma 65 on this page. The right element is implied by lemma 37 on page 13. □

Lemma 68 (Substitutions and honesty)**.** *For all process types f, for all* $\sigma = \{^{s}/_{x}\}$*, and for all* $u \in \mathcal{N} \cup \mathcal{V}$*,*

$$f(u) \textit{ honest} \implies (f \bullet \sigma)\,(u\sigma) \textit{ honest}$$

Proof. By definition 48 on page 16, we have three cases:

- $u \notin \{x, s\}$. When u is not involved in the substitution, then clearly $(f \bullet \sigma)\,(u\sigma) = f(u)$, which is honest by hypothesis;
- $u = x$. Then, $(f \bullet \sigma)\,(x\sigma) = (f \bullet \sigma)\,(s) = f(x) = f(u)$, which is honest by hypothesis;
- $u = s$. Then, $(f \bullet \sigma)\,(s\sigma) = (f \bullet \sigma)\,(s) = f(x) = f(u)$, which is honest by hypothesis.

□

Lemma 69 (Substitution lemma)**.** *For all processes P and for all $\sigma = \{s/x\}$ with $s \notin \mathrm{fn}(P)$:*

$$\Gamma \vdash P\colon f \implies \Gamma\bullet\sigma \vdash P\sigma\colon f\bullet\sigma$$

Proof. By induction on the typing derivation of $\vdash P\colon f$, we have the following cases:

- [T-SUM], [T-PAR]. Trivial, by applying the induction hypothesis on the rule premises;
- [T-DEL]. We have:

$$\frac{\Gamma_{\neq u} \vdash P'\colon f' \quad f'(u)\text{ honest}}{\Gamma \vdash P = (u)P'\colon f'\{^{f'(*)}/_{u}\} = f}\ \text{[T-DEL]}$$

  By lemma 68 on the previous page, we have that the premise "$f(u)$ honest" implies that $(f'\bullet\sigma)(u\sigma)$ is honest. By applying the induction hypothesis on the rule premises, we have:

$$\frac{\Gamma_{\neq u}\bullet\sigma = (\Gamma\bullet\sigma)_{\neq u\sigma} \vdash P'\sigma\colon f'\bullet\sigma \quad (f'\bullet\sigma)(u\sigma)\text{ honest}}{\Gamma\bullet\sigma \vdash P\sigma = (u\sigma)P'\sigma\colon\ (f'\bullet\sigma)\{^{(f'\bullet\sigma)(*)}/_{u\sigma}\} = (f\bullet\sigma)}\ \text{[T-DEL]}$$

- [T-VAR]. Trivial;
- [T-DEF]. We have:

$$\frac{X(\vec{u}) \stackrel{\text{def}}{=} P' \quad \Gamma\{^{f}/_{X(\vec{v})}\} \vdash P'\{^{\vec{v}}/_{\vec{u}}\}\colon f}{\Gamma \vdash P = X(\vec{v})\colon f}\ \text{[T-DEF]}$$

  By applying the induction hypothesis on the rule premises, we have:

$$\frac{X(\vec{u}) \stackrel{\text{def}}{=} P' \quad \Gamma\{^{f}/_{X(\vec{v})}\}\bullet\sigma = (\Gamma\bullet\sigma)\{^{f\bullet\sigma}/_{X(\vec{v}\sigma)}\} \vdash P'\{^{\vec{v}\sigma}/_{\vec{u}}\}\colon f\bullet\sigma}{\Gamma\bullet\sigma \vdash X(\vec{v}\sigma) = X(\vec{v})\sigma = P\sigma\colon f\bullet\sigma}\ \text{[T-DEF]}$$

□

Lemma 70 (System typing and substitution)**.** *For all systems S and all substitutions $\sigma = \{s/x\}$:*

$$\vdash_{\mathsf{A}} S\colon f \implies \vdash_{\mathsf{A}} S\sigma\colon f\bullet\sigma \tag{13}$$

$$\vdash_{\mathsf{A}} S \rhd f \implies \vdash_{\mathsf{A}} S\sigma \rhd f\bullet\sigma \tag{14}$$

Proof. For (13), by induction on the typing derivation of $\vdash_{\mathsf{A}} S\colon f$, we have the following cases:

- [T-SA]: let $S = \mathsf{A}[P]$. By lemma 69 on this page, on the premise we have $\emptyset \vdash P\colon f \implies \emptyset\bullet\sigma = \emptyset \vdash P\sigma\colon f\bullet\sigma$, and thus $\vdash_{\mathsf{A}} \mathsf{A}[P\sigma] = S\sigma\colon f\bullet\sigma$;
- [T-SDEL2]: let $S = (z)S'$, and $f = f'\{^{f'(*)}/_{z}\}$. By applying the induction hypothesis on the premises of the rule, we have $\vdash_{\mathsf{A}} S'\sigma\colon f'\bullet\sigma$ and $f'\bullet\sigma(z\sigma)$*honest* (lemma 68 on the previous page). By applying rule [T-SDEL2] again, we have $\vdash_{\mathsf{A}} (z\sigma)S'\sigma = S\sigma\colon f'\bullet\sigma\{^{f'\bullet\sigma(*)}/_{z\sigma}\} = f\bullet\sigma$;
- [T-SPAR2]: let $S = S' \mid S''$. By applying the induction hypothesis on the premises of the rule, we have $\vdash_{\mathsf{A}} S'\sigma\colon f\bullet\sigma$ and $\vdash_{\mathsf{A}} S''\sigma\colon f\bullet\sigma$. Thus, by applying [T-SPAR2] again, we have $\vdash_{\mathsf{A}} S'\sigma \mid S''\sigma = (S' \mid S'')\sigma = S\sigma\colon f\bullet\sigma$.

For (14), by induction on the depth of the typing derivation of $\vdash_{\mathsf{A}} S \rhd f$, we have the following cases:

- [T-SAFREE0], [T-SAFREE1], [T-SAFREE2]: the renaming does not have any consequence on the premises, and thus the substitution of S and f respectively with $S\sigma$ and $f\bullet\sigma$ is immaterial;
- [T-SFZ1]: by substituting S and f respectively with $S\sigma$ and $f\bullet\sigma$, from the premises of the rule we need $f\bullet\sigma(x\sigma)$ to realize c — which is true both when x is not involved in the renaming (and thus $f(x)$ is not altered), and when it is (since, by definition, if σ substitutes x with s, then $f\bullet\sigma = f\{^{f(*)}/_{x}\}\{^{f(x)}/_{s}\}$, and thus $f\bullet\sigma(x\sigma) = f\bullet\sigma(s) = f(x)$);
- [T-SFUSED]: by substituting S and f respectively with $S\sigma$ and $f\bullet\sigma$, from the premises of the rule we need $f\bullet\sigma(s\sigma)$ to realize c — which is true, as seen in the proof for rule [T-SFZ1]. The other premise ("$\gamma\sigma$ A-free") is not influenced by the substitution;

- [T-SDEL1]: let $S = (z)S'$. By applying the induction hypothesis on the premise, we have:

$$\begin{aligned}\vdash_{\mathsf{A}} S'\sigma \rhd f\{f(*)/z\} \bullet \sigma &= f\{f(*)/z\}\{f\{f(*)/z(*)\}/x\}\{f\{f(*)/z\}(x)/s\} \\ &= f\{f(*)/z\}\{f(*)/x\}\{f\{f(*)/z\}(x)/s\} \\ &= f\{f(*)/x\}\{f(x)/s\}\{f\bullet\sigma(*)/z\sigma\} \\ &= f \bullet \sigma\{f\bullet\sigma(*)/z\sigma\}\end{aligned}$$

  From this we can conclude that $\vdash_{\mathsf{A}} (z)S'\sigma = S\sigma\colon\ f \bullet \sigma$.
- [T-SFZ2], [T-SPAR1]: straightforward, by applying the induction hypothesis on the premises.

□

Lemma 71 (Reductions descending from [FUSE])**.** *For all systems S, all participants* A *and all substitutions* $\sigma \neq \emptyset$*:*

$$S \xrightarrow{\mathsf{A}:\pi,\sigma} S' \ \wedge\ \sigma \neq \emptyset \implies \pi = \mathsf{fuse}$$

(i.e., the reduction derives from a [FUSE] *rule application).*

Proof. We proceed by induction on reduction semantics of CO_2 (fig. 3.2 on page 5).

- [TAU], [TELL], [DO], [ASK]: the theorem holds vacuously, since $\sigma = \emptyset$;
- [FUSE]: the rule definition asserts $\pi = \mathsf{fuse}$, while defining $\{s/x,y\} = \sigma \neq \emptyset$;
- [DEL1]: by applying the induction hypothesis on the premise, this rule generates a reduction $S = (x)S'' \xrightarrow{\mathsf{A}:\mathsf{fuse},\emptyset} S' = (s)S'''$. It makes the theorem hold vacuously, and also explains why the inverse implication does not hold (i.e., $S \xrightarrow{\mathsf{A}:\pi,\sigma} S' \wedge \sigma = \emptyset \not\implies \pi \neq \mathsf{fuse}$);
- [DEL2], [PAR], [DEF]: straightforward, by applying the induction hypothesis on the premises of the rules.

□

Lemma 72 (Channel type moves and honesty)**.** *For all channel types T, and for* $\alpha \in \{\tau, \tau_?, \tau_\phi, \langle c\rangle\}$*:*

$$T \text{ honest} \ \wedge\ T \xrightarrow{\alpha} T' \implies T' \text{ honest}$$

Proof. By definition 25 on page 11, T is honest when the abstract process $(\emptyset, T)$ is honest, i.e. when $(\emptyset, T) \to^* (c', T'')$ implies that T'' is ready for c'. By definition 21 on page 10 (rules [A-TELL1] and [A-TAU1]), the form of α in the hypotheses reduces this proof to the analysis of two possible abstract process transitions:

1. $(\emptyset, T) \to (\{c\}, T')$ (when $\alpha = \langle c\rangle$). Since $(\emptyset, T)$ is honest by hypothesis, we have that $(\emptyset, T) \to (\{c\}, T') \to^* (c', T'')$ implies that T'' is ready for c'. Then, we have that also $(\emptyset, T') \to^* (c', T'')$ implies that T'' is ready for c', i.e. T' is honest. In fact, by removing c from the set of latent contracts associated to T', the abstract process $(\emptyset, T')$ may evolve in two ways:
   (a) it may hang on the first blocking a-action expected by c. In this case, no more contracts would be advertised, and T' would be vacuously honest;
   (b) or, it may continue evolving along other branches. In those branches, more contracts could be advertised and fused. All these possibilities are already covered by the honesty of T in the hypothesis;
2. $(\emptyset, T) \to (\emptyset, T')$ (when $\alpha \in \{\tau, \tau_?, \tau_\phi\}$). Since $(\emptyset, T)$ is honest by hypothesis, we have that $(\emptyset, T) \to (\emptyset, T') \to^* (c', T'')$ implies that T'' is ready for c'. This, by definition 25 on page 11, implies that T' is honest as well.

□

Lemma 73 (Reductions and free names/variables)**.** *For all systems S, all participants* A *and all substitutions* σ*:*

$$S \xrightarrow{\mathsf{A}:\pi,\sigma} S' \ \wedge\ x \in \operatorname{dom}\sigma \implies x \notin \operatorname{fnv}(S')$$

Proof. We proceed by induction on the reduction semantics of CO_2 (fig. 3.2 on page 5).

- [TAU], [TELL], [DO], [ASK]: the theorem holds vacuously, since $\sigma = \emptyset$ (and thus $x \notin \operatorname{dom}\sigma$);
- [FUSE]: the rule replaces x in all the components of the reduct, and thus $x \notin \operatorname{fnv}(S')$;
- [DEL1], [DEL2], [PAR], [DEF]: straightforward, by applying the induction hypothesis on the premises of the rules.

□

Definition 74 (Environment concatenation)**.** *We define the* environment concatenation operator "$\cdot$" *as:*

$$(\Gamma' \cdot \Gamma)(X(\vec{v})) = \begin{cases} f & \text{if } (X(\vec{v}), f) \in \Gamma \\ f' & \text{if } (X(\vec{v}), f') \notin \Gamma \text{ and } (X(\vec{v}), f') \in \Gamma' \\ \text{undefined} & \text{otherwise} \end{cases}$$

Lemma 75 (Typing with a growing environment)**.** *For all CO_2 processes P and for all typing environments Γ, Γ':*

$$\Gamma \vdash P \colon f \implies \Gamma' \cdot \Gamma \vdash P \colon f$$

Proof. We proceed by induction on the typing derivation of $\vdash P \colon f$.

- [T-Sum], [T-Par]: trivial, by applying the induction hypothesis;
- [T-Del]: trivial, by applyig the induction hypothesis (considering that $\Gamma'_{\neq u} \cdot \Gamma_{\neq u} = (\Gamma' \cdot \Gamma)_{\neq u}$);
- [T-Def]: trivial, by applying the induction hypothesis (considering that $\Gamma'\{f/X(\vec{v})\} \cdot \Gamma\{f/X(\vec{v})\} = (\Gamma' \cdot \Gamma)\{f/X(\vec{v})\}$);
- [T-Var]: if the rule premise is satisfied by Γ, then it is necessarily satisfied also by $\Gamma' \cdot \Gamma$.

□

Definition 76 (Unfolding)**.** *The* unfolding *of a process P is defined as follows:*

$$\text{unfold}(X(\vec{v})) = P\{\vec{v}/\vec{u}\} \qquad \text{if } X(\vec{u}) \stackrel{def}{=} P \qquad \text{unfold}((\vec{u})P) = (\vec{u})\,\text{unfold}(P)$$

$$\text{unfold}(P \mid Q) = \text{unfold}(P) \mid \text{unfold}(Q) \qquad \text{unfold}(\textstyle\sum_i \pi_i.P_i) = \sum_i \pi_i.P_i$$

Lemma 77 (Unfolding and typing)**.** *For all processes P, and for all types f:*

$$\vdash P \colon f \iff \vdash \text{unfold}(P) \colon f$$

Proof. We proceed by induction on the structure of P. In all cases but $P = X(\vec{v})$, the thesis follows trivially by applying the induction hypothesis. In the base case $P = X(\vec{v})$, we have the following typing derivation for the LHS:

$$\dfrac{X(\vec{u}) \stackrel{\text{def}}{=} P' \quad \dfrac{\dfrac{\vdots}{\{f/X(\vec{v})\} \vdash \cdots}\,[R_2] \quad \{f/X(\vec{v})\} \vdash P'\{\vec{v}/\vec{u}\} \colon f}{\vdash X(\vec{v}) \colon f}\,[R_1]}{}\,[\text{T-Def}]$$

and thus, since $\text{unfold}(X(\vec{v})) = P'\{\vec{v}/\vec{u}\}$ if $X(\vec{u}) \stackrel{\text{def}}{=} P'$ (definition 76 on the current page), the typing derivation can be rewritten as:

$$D = \begin{cases} \dfrac{X(\vec{u}) \stackrel{\text{def}}{=} P' \quad \dfrac{\dfrac{\vdots}{\{f/X(\vec{v})\} \vdash \cdots}\,[R_2] \quad \{f/X(\vec{v})\} \vdash \text{unfold}(X(\vec{v})) \colon f}{\vdash X(\vec{v}) \colon f}\,[R_1]}{}\,[\text{T-Def}] \end{cases}$$

From right to left ($\Longleftarrow$), the theorem is trivial: by lemma 75 on this page, $\vdash \text{unfold}(X(\vec{v})) \colon f \implies \{f/X(\vec{v})\} \vdash \text{unfold}(X(\vec{v})) \colon f$, and we conclude by applying [T-Def].

From left to right($\Longrightarrow$), we need to show that, from any typing derivation D of the LHS, we can build a proof of $\vdash \text{unfold}(X(\vec{v})) \colon f$ — possibly using D itself as its premise. Starting with the instance of rule $[R_1]$ in D and going upwards, the derivation can be transformed using a rewriting $R^{-\{f/X(\vec{v})\}}$, defined as shown in fig. A.1 on the next page, which removes $\{f/X(\vec{v})\}$ from each environment, paying special attention in three cases:

- [T-Del]: when $w \in \vec{v}$, the $\{f/X(\vec{v})\}$ binding is removed from D_1 by [T-Del] itself (and thus, $R^{-\{f/X(\vec{v})\}}(D_1)$ has no effects); otherwise, the removal is performed by $R^{-\{f/X(\vec{v})\}}$;
- [T-Def]: when $X(\vec{v})$ is redefined along a derivation branch, we leave D_1 untouched;
- [T-Var]: when $X(\vec{v})$ is actually referenced from the environment Γ, we replace the [T-Var] application with a derivation based on D, with the final environment consisting in $\Gamma \setminus \{f/X(\vec{v})\}$. For this pourpose, we introduce another rewriting, $R^{+(\Gamma \setminus \{f/X(\vec{v})\})}$, defined in Fig .fig. A.2 on page 28.

$$\frac{D_i \quad \forall i \in I}{\Gamma \vdash \sum_{i\in I} \pi_i \,.\, P_i \colon f'} \text{ [T-Sum]} \quad \mapsto \quad \frac{R^{-\{f/X(\vec{v})\}}(D_i) \quad \forall i \in I}{(\Gamma \setminus \{f/X(\vec{v})\}) \vdash \sum_{i\in I} \pi_i \,.\, P_i \colon f'} \text{ [T-Sum]}$$

$$\frac{D_1 \quad D_2}{\Gamma \vdash P_1 \mid P_2 \colon f'} \text{ [T-Par]} \quad \mapsto \quad \frac{R^{-\{f/X(\vec{v})\}}(D_1) \quad R^{-\{f/X(\vec{v})\}}(D_2)}{(\Gamma \setminus \{f/X(\vec{v})\}) \vdash P_1 \mid P_2 \colon f'} \text{ [T-Par]}$$

$$\frac{D_1}{\Gamma \vdash (w)P'' \colon f'} \text{ [T-Del]} \quad \mapsto \quad \frac{R^{-\{f/X(\vec{v})\}}(D_1)}{(\Gamma \setminus \{f/X(\vec{v})\}) \vdash (w)P'' \colon f'} \text{ [T-Del]}$$

$$\frac{Y(\vec{w}) \stackrel{\text{def}}{=} P'' \quad D_1}{\Gamma \vdash Y(\vec{z}) \colon f'} \text{ [T-Def]} \quad \mapsto \quad \frac{Y(\vec{w}) \stackrel{\text{def}}{=} P'' \quad R^{-\{f/X(\vec{v})\}}(D_1)}{(\Gamma \setminus \{f/X(\vec{v})\}) \vdash Y(\vec{z}) \colon f'} \text{ [T-Def]} \quad \text{if } Y \neq X \text{ or } \vec{z} \neq \vec{v}$$

$$\frac{X(\vec{u}) \stackrel{\text{def}}{=} P' \quad D_1}{\Gamma \vdash X(\vec{v}) \colon f} \text{ [T-Def]} \quad \mapsto \quad \frac{X(\vec{u}) \stackrel{\text{def}}{=} P' \quad D_1}{(\Gamma \setminus \{f/X(\vec{v})\}) \vdash X(\vec{v}) \colon f} \text{ [T-Def]}$$

$$\frac{\Gamma(Y(\vec{w})) = f'}{\Gamma \vdash Y(\vec{w}) \colon f'} \text{ [T-Var]} \quad \mapsto \quad \frac{(\Gamma \setminus \{f/X(\vec{v})\})(Y(\vec{w})) = f'}{(\Gamma \setminus \{f/X(\vec{v})\}) \vdash Y(\vec{w}) \colon f'} \text{ [T-Var]} \quad \text{if } Y \neq X \text{ or } \vec{z} \neq \vec{v}$$

$$\frac{\Gamma(X(\vec{v})) = f}{\Gamma \vdash X(\vec{v}) \colon f} \text{ [T-Var]} \quad \mapsto \quad R^{+(\Gamma \setminus \{f/X(\vec{v})\})}(D) \quad \text{(see A.2)}$$

Figure A.1: Rules for the rewriting $R^{-\{f/X(\vec{v})\}}$ (see proof of lemma 77 on the facing page).

It is easy to check that the result is a proof of the form:

$$\frac{\dfrac{\vdots}{\vdash \ldots} \text{ [}R_2\text{]}}{\vdash \text{unfold}(X(\vec{v})) \colon f} \text{ [}R_1\text{]}$$

□

A.2 Proof of Theorem 27 on page 12

Proof. (Sketch) Abstract readiness and abstract dishonesty are reachability properties. Abstract processes are the product of a finite state system (sets of contracts C and and single stipulated contracts 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]. □

A.3 Proof of Lemma 33 on page 13

Proof. We proceed by induction on the typing derivation of $\vdash P \colon f$.

- [T-Sum]. We have:
$$\frac{\Gamma \vdash P_i \colon f_i \quad \forall i \in I}{\Gamma \vdash P = \sum_{i\in I} \pi_i \,.\, P_i \colon \lambda u \,.\, \sum_{i\in I} [\pi_i]_u \,.\, f_i(u) = f} \text{ [T-Sum]}$$
and we observe that the prefix abstraction $[\pi_i]_*$ (definition 29 on page 12) can only result in τ or $\tau_?$, when $f(\tau_?)$ is evaluated;
- [T-Par], [T-Var]. Trivial, by applying the induction hypothesis on the rule premises;

$$\frac{D_i \quad \forall i \in I}{\Gamma' \vdash \sum_{i\in I} \pi_i \,.\, P_i \colon f'}\ \text{[T-Sum]} \quad \mapsto \quad \frac{R^{+\Gamma}(D_i) \quad \forall i \in I}{(\Gamma\cdot\Gamma') \vdash \sum_{i\in I} \pi_i \,.\, P_i \colon f'}\ \text{[T-Sum]}$$

$$\frac{D_1 \quad D_2}{\Gamma' \vdash P_1 \mid P_2 \colon f'}\ \text{[T-Par]} \quad \mapsto \quad \frac{R^{+\Gamma}(D_1) \quad R^{+\Gamma}(D_2)}{(\Gamma\cdot\Gamma') \vdash P_1 \mid P_2 \colon f'}\ \text{[T-Par]}$$

$$\frac{D_1}{\Gamma' \vdash (w)P'' \colon f'}\ \text{[T-Del]} \quad \mapsto \quad \frac{R^{+\Gamma_{\neq w}}(D_1)}{(\Gamma\cdot\Gamma') \vdash (w)P'' \colon f'}\ \text{[T-Del]}$$

$$\frac{Y(\vec{w}) \stackrel{\text{def}}{=} P'' \quad D_1}{\Gamma' \vdash Y(\vec{z}) \colon f'}\ \text{[T-Def]} \quad \mapsto \quad \frac{Y(\vec{w}) \stackrel{\text{def}}{=} P'' \quad R^{+\Gamma}(D_1)}{(\Gamma\cdot\Gamma') \vdash Y(\vec{z}) \colon f'}\ \text{[T-Def]}$$

$$\frac{\Gamma'(Y(\vec{w})) = f'}{\Gamma' \vdash Y(\vec{w}) \colon f'}\ \text{[T-Var]} \quad \mapsto \quad \frac{(\Gamma\cdot\Gamma')(Y(\vec{w})) = f'}{(\Gamma\cdot\Gamma') \vdash Y(\vec{w}) \colon f'}\ \text{[T-Var]}$$

Figure A.2: Rules for the rewriting $R^{+\Gamma}$ (used in fig. A.1 on the previous page for the proof of lemma 77 on page 26).

- [T-Del]. We have:

$$\frac{\Gamma_{\neq u} \vdash P' \colon f \quad f(u) \text{ honest}}{\Gamma \vdash P = (u)P' \colon f\{^{f(*)}/_{u}\}}\ \text{[T-Del]}$$

  By lemma 75 on page 26, the rule premise $\Gamma_{\neq u} \vdash P \colon f$ implies that the same typing can be obtained with a larger environment, i.e. $\Gamma \vdash P \colon f$. Hence, the theorem is demonstrated by applying the induction hypothesis;
- [T-Def]. We have a typing derivation of the form:

$$\frac{X(\vec{u}) \stackrel{\text{def}}{=} P' \quad D_1 = \left\{\begin{array}{c} \dfrac{\vdots}{\Gamma_1 \vdash \cdots}\ [\cdots] \\ \dfrac{}{\Gamma\{^{f}/_{X(\vec{v})}\} = \Gamma_1 \vdash P'\{^{\vec{v}}/_{\vec{u}}\} \colon f}\ [\cdots] \end{array}\right.}{\Gamma \vdash P = X(\vec{v}) \colon f}\ \text{[T-Def]}$$

  We need to show that the typing environments in the derivation will never map $X(\vec{v})$ to a process type g such that $g(*)$ contains actions differing from τ and $\tau_?$. We proceed by induction on the number of applications of rule [T-Def] for $X(\vec{v})$.
  - 1 application of [T-Def] for $X(\vec{v})$ (base case). The type of $X(\vec{v})$ in the Γ_1 environment is never replaced along the derivation branch D_1, and such typing only contains instances of rules [T-Sum], [T-Par], [T-Var] and [T-Del]. Considering that recursion in CO_2 processes is guarded (see section 3 on page 4), each instance of [T-Var] can only appear above an instance of [T-Sum]. Hence, f will have the form:

$$f = \lambda u \ .\ [\pi_1]_u \,.\, \cdots \,.\, f(u) \ \mid\ [\pi_2]_u \,.\, \cdots \,.\, f(u) \ \mid\ [\pi_3]_u \,.\, \cdots$$

    where each $f(u)$ is introduced by an instance of [T-Var] (if present), each $\mid$ by [T-Par], and each prefix abstractions $[\pi_i]_u$ by [T-Sum]. Furthermore, as shown above, we know that [T-Sum] only allows $[\pi_i]_*$ that evaluate to τ or $\tau_?$: hence, $f(*)$ only contains actions τ and $\tau_?$;
  - $n+1$ applications of [T-Def] for $X(\vec{v})$ (inductive step). Given a derivation with n instances of [T-Def] for $X(\vec{v})$, a derivation with $n+1$ instances can only be obtained by replacing an instance of [T-Var] with a

further derivation D_{n+1} based on [T-DEF][3]. Such replacement has the form:

$$\dfrac{\dfrac{\Gamma_n(X(\vec{v})) = f_n}{\Gamma_n \vdash X(\vec{v}) \colon f_n}\ {\scriptstyle [\text{T-VAR}]}}{\vdots}\ {\scriptstyle [\cdots]} \quad \mapsto \quad \dfrac{X(\vec{u}) \stackrel{\text{def}}{=} P' \quad D_{n+1} = \left\{\dfrac{\vdots}{\Gamma_n\{f_{n+1}/X(\vec{v})\} = \Gamma_{n+1} \vdash P\{\vec{v}/\vec{u}\} \colon f_{n+1}}\ {\scriptstyle [\cdots]}\right.}{\dfrac{\Gamma_n \vdash X(\vec{v}) \colon f_{n+1}}{\vdots}\ {\scriptstyle [\cdots]}}\ {\scriptstyle [\text{T-DEF}]}$$

Here, f_{n+1} replaces the occurrences of f_n downwards in the typing derivation[4], and the branch D_{n+1} does not replace the typing of $X(\vec{v})$ in Γ_{n+1} with other application of [T-DEF] — just like the derivation D_1 in the base case above. Hence, since D_{n+1} does not introduce in $f_{n+1}(*)$ actions differing from τ and $\tau_?$, nor are they introduced (by inductive hypothesis) in other typing environments along the rest of the derivation, we conclude that $f(*)$ only contains actions τ and $\tau_?$.

□

A.4 Proof of Lemma 34 on page 13

Proof. When $z \notin \text{fnv}(P)$, we have two cases:

- z does not appear at all P. Then, for all CO_2 prefixes π, we have $[\pi]_z = [\pi]_*$ (see the prefix abstraction mapping in definition 29 on page 12). And thus, by applying the same proof technique of lemma 33 on page 13 (i.e., showing that $f(z)$ can only contain τ and $\tau_?$ actions), we conclude that $f(z) = f(*)$;
- $P \equiv (z)P'$, with $z \in \text{fnv}(P')$ and $\Gamma \vdash P' \colon f'$. Then, considering that typing is preserved by structural equivalence (lemma 35 on page 13), the typing derivation of P has the form:

$$\dfrac{\Gamma \vdash P' \colon f' \quad f'(z) \text{ honest}}{\Gamma \vdash P \equiv (z)P' \colon f'\{^{f'(*)}/_z\} = f}\ {\scriptstyle [\text{T-DEL}]}$$

Hence, $f(z) = f'\{^{f'(*)}/_z\}(z) = f'(*) = f(*)$.

□

A.5 Proof of Lemma 35 on page 13

Proof. . We proceed by induction on the typing derivation of $\Gamma \vdash P \colon f$. We consider the different cases of structural equivalence between processes, and show that when the LHS (resp. RHS) is typed with f, the same premises of its typing derivation can be used to type the RHS (resp. LHS) with f.

- $P \mid (u)P' \equiv (u)(P \mid P')$ if $u \notin \text{fv}(P) \cup \text{fn}(P)$. Let $\Gamma \vdash P \colon f$ and $\Gamma_{\neq u} \vdash P \colon f$ (since $f(u) = f(*)$ by lemma 34 on page 13) and $\Gamma_{\neq u} \vdash P' \colon f'$. Then we have the following typing derivations for the two sides of the equivalence:

$$\dfrac{\dfrac{\Gamma \vdash P \colon f \qquad \dfrac{\Gamma_{\neq u} \vdash P' \colon f' \quad f'(u) \text{ honest}}{\Gamma \vdash (u)P' \colon f'\{^{f'(*)}/_u\}}\ {\scriptstyle [\text{T-DEL}]}}{\Gamma \vdash P \mid (u)P' \colon \lambda v.\, f(v) \mid f'\{^{f'(*)}/_u\}(v)}\ {\scriptstyle [\text{T-PAR}]}$$

$$\dfrac{\dfrac{\Gamma_{\neq u} \vdash P \colon f \quad \Gamma_{\neq u} \vdash P' \colon f'}{\Gamma_{\neq u} \vdash P \mid P' \colon f'' = \lambda v.\, f(v) \mid f'(v)}\ {\scriptstyle [\text{T-PAR}]} \qquad f''(u) \text{ honest}}{\Gamma \vdash (u)(P \mid P') \colon f''\{^{f''(*)}/_u\}}\ {\scriptstyle [\text{T-DEL}]}$$

Considering that $f(u) = f(*)$ implies $f = f\{^{f(*)}/_u\}$, the last typing relation can be rewritten as:

$$\begin{aligned}\Gamma \vdash (u)(P \mid P') \colon f''\{^{f''(*)}/_u\} &= \big(\lambda v.\, f(v) \mid f'(v)\big)\,\{^{(f(*)\mid f'(*))}/_u\} \\ &= \lambda v.\, f\{^{f(*)}/_u\}(v) \mid f'\{^{f'(*)}/_u\}(v) \\ &= \lambda v.\, f(v) \mid f'\{^{f'(*)}/_u\}(v)\end{aligned}$$

[3] The presence of a [T-VAR] instance restricts the inductive step to recursive definitions of $X(\vec{v})$.

[4] This corresponds to the fact that the introduction of a further application of [T-DEF] amounts to an unfolding of the recursive process type.

- $(u)P \equiv P$ if $u \notin \mathrm{fv}(P) \cup \mathrm{fn}(P)$. Let $\Gamma \vdash P\colon f$ and $\Gamma_{\neq u} \vdash P\colon f$ (since $f(u) = f(*)$ by lemma 34 on page 13). Then we have the following typing derivation for the LHS of the equivalence:

$$\frac{\Gamma_{\neq u} \vdash P\colon f \quad f(u)\text{ honest}}{\Gamma \vdash (u)P\colon f\{^{f(*)}/_{u}\}}\ [\text{T-DEL}]$$

  But $f(u) = f(*)$ implies $f\{^{f(*)}/_{u}\} = f$, which is the type of P.
- $(u)(v)P \equiv (v)(u)P$. The typing derivations for the two sides of the equivalence are:

$$\frac{\dfrac{\Gamma_{\neq u,v} \vdash P\colon f \quad f(u)\text{ honest}}{\Gamma_{\neq v} \vdash (u)P\colon f\{^{f(*)}/_{u}\}}\ [\text{T-DEL}] \qquad f\{^{f(*)}/_{u}\}(v)\text{ honest}}{\Gamma \vdash (v)(u)P\colon f\{^{f(*)}/_{u}\}\{^{f\{^{f(*)}/_{u}\}(*)}/_{v}\} = f\{^{f(*)}/_{u}\}\{^{f(*)}/_{v}\}}\ [\text{T-DEL}]$$

$$\frac{\dfrac{\Gamma_{\neq v,u} \vdash P\colon f \quad f(v)\text{ honest}}{\Gamma_{\neq u} \vdash (v)P\colon f\{^{f(*)}/_{v}\}}\ [\text{T-DEL}] \qquad f\{^{f(*)}/_{v}\}(u)\text{ honest}}{\Gamma \vdash (u)(v)P\colon f\{^{f(*)}/_{v}\}\{^{f\{^{f(*)}/_{v}\}(*)}/_{u}\} = f\{^{f(*)}/_{v}\}\{^{f(*)}/_{u}\} = f\{^{f(*)}/_{u}\}\{^{f(*)}/_{v}\}}\ [\text{T-DEL}]$$

- commutative monoidal laws for $|$ ($(P \mid P') \equiv P' \mid P'$; $(P \mid P') \mid P'' \equiv P \mid (P' \mid P'')$; $P \mid \mathbf{0} \equiv P$). Follow directly from the commutative monoidal laws for $|$ on channel types (fig. 5.1 on page 9);
- equivalence properties ($P \equiv P$; $P \equiv P' \implies P' \equiv P$; $P \equiv P'' \wedge P'' \equiv P' \implies P \equiv P'$). Follow from the fact that, if $\Gamma \vdash P\colon f$, $\Gamma \vdash P'\colon f'$ and $P \equiv P'$, then P and P' can be typed with the same $f = f'$, using the same premises;
- congruence laws. Let $P \equiv P'$ and, by applying the induction hypothesis, let $\Gamma' \vdash P\colon f^{P} \implies \Gamma' \vdash P'\colon f^{P}$:
  - $\Gamma \vdash P \mid Q\colon f \implies \Gamma \vdash P' \mid Q\colon f$. Trivial, by rule [T-PAR];
  - $\Gamma \vdash \pi.P\colon f \implies \Gamma \vdash \pi.P'\colon f$. Trivial, by rule [T-SUM];
  - $\Gamma \vdash P + Q\colon f \implies \Gamma \vdash P' + Q\colon f$. Trivial, by rule [T-SUM];
  - $\Gamma \vdash (u)P\colon f \implies \Gamma \vdash (u)P'\colon f$. Trivial, by rule [T-DEL];
  - $\left(X(\vec{u}) \stackrel{\text{def}}{=} P \implies \Gamma \vdash X(\vec{v})\colon f\right) \implies \left(Y(\vec{u}) \stackrel{\text{def}}{=} P' \implies \Gamma \vdash Y(\vec{v})\colon f\right)$. By lemma 61 on page 21, $P\{^{\vec{v}}/_{\vec{u}}\} \equiv P'\{^{\vec{v}}/_{\vec{u}}\}$, and thus, by applying the induction hypothesis, their type is the same in any environment. By applying [T-DEF] on the LHS and RHS of the implication, we have:

$$\frac{X(\vec{u}) \stackrel{\text{def}}{=} P \quad \Gamma\{^{f}/_{X(\vec{v})}\} \vdash P\{^{\vec{v}}/_{\vec{u}}\}\colon f}{\Gamma \vdash X(\vec{v})\colon f}\ [\text{T-DEF}] \qquad \frac{Y(\vec{u}) \stackrel{\text{def}}{=} P' \quad \Gamma\{^{f}/_{Y(\vec{v})}\} \vdash P'\{^{\vec{v}}/_{\vec{u}}\}\colon f}{\Gamma \vdash Y(\vec{v})\colon f}\ [\text{T-DEF}]$$

□

A.6 Proof of Lemma 37 on page 13

Proof. $(u)P$ has a typing derivation of the form:

$$\frac{\vdash P\colon f' \quad f'(u)\text{ honest}}{\vdash (u)P\colon f'\{^{f'(*)}/_{u}\} = f}\ [\text{T-DEL}]$$

Thus, we have $f(u) = f'(*)$, and $\forall z \in (\mathcal{N} \cup \mathcal{V} \cup \{*\}) \setminus \{u\}\,.\, f(z) = f'(z)$. Hence, $f \sqsubseteq f'$ (definition 36 on page 13). □

A.7 Proof of Lemma 40 on page 14

Proof. We proceed by cases on the value of α.

- $\alpha = \tau$. By definition 29 on page 12, $[\pi]_u = \alpha = \tau$ only when $\pi = \tau$ or $\pi = \mathsf{tell}\downarrow_w c$, with $w \neq u$. Then, $f(u) \xrightarrow{\tau}$ implies that, for all $u \in \mathcal{N} \cup \mathcal{V}$, the semantics derivation of the channel type must be of the form:

$$\frac{\dfrac{\dfrac{}{[\pi]_u\,.\,T_u^2 \xrightarrow{\tau} T_u^2}\ [\text{C-ALPHA}]}{[\pi]_u\,.\,T_u^2 + T_u^3 \xrightarrow{\tau} T_u^2}\ [\text{C-SUML}]}{f(u) = [\pi]_u\,.\,T_u^2 + T_u^3 \mid T_u^4 \xrightarrow{\tau} T_u^2 \mid T_u^4 = T'}\ [\text{C-PARL}]$$

Since f is inhabited by hypothesis, its components must be other inhabited process types, fitting some derivation according to the rules in fig. 5.4 on page 13. Thus, there exist f_2, f_3, f_4 such that $f_2(u) = T_u^2$, $f_3(u) = T_u^3$, $f_4(u) = T_u^4$: then, we have $f = \lambda v.\,[\pi]_v\,.\,f_2(v) + f_3(v) \mid f_4(v)$, and $f' = \lambda v.\,f_2(v) \mid f_4(v)$. Thus, there exists f' such that $f'(u) = T'$, and $f \xrightarrow{\pi} f'$;

- $\alpha = \langle c \rangle$. By definition 29 on page 12, $[\pi]_u = \alpha = \langle c \rangle$ only when $\pi = \mathsf{tell}_\mathsf{B} \downarrow_u c$. The proof is similar to the $\alpha = \tau$ case above — except that $[\tau]_u$ occurrences must be replaced with $[\mathsf{tell}_\mathsf{B} \downarrow_w c]_u$, and $\xrightarrow{\tau}$-transitions with $\xrightarrow{\langle c \rangle}$-transitions;
- $\alpha = \mathsf{a}$. By definition 29 on page 12, $[\pi]_u = \alpha = \mathsf{a}$ only when $\pi = \mathsf{do}_u\,\mathsf{a}$. The proof is similar to the $\alpha = \tau$ case above — except that $[\tau]_u$ occurrences must be replaced with $[\mathsf{do}_x\,\mathsf{a}]_u$, and $\xrightarrow{\tau}$-transitions with $\xrightarrow{\mathsf{a}}$-transitions;
- $\alpha = \tau_\phi$. By definition 29 on page 12, $[\pi]_u = \alpha = \tau_\phi$ only when $\pi = \mathsf{ask}_u\,\mathsf{a}$. The proof is similar to the $\alpha = \tau$ case above — except that $[\tau]_u$ occurrences must be replaced with $[\mathsf{ask}_u\,\mathsf{a}]_u$,and $\xrightarrow{\tau}$-transitions with $\xrightarrow{\tau_?}$-transitions;
- $\alpha = \tau_?$. By definition 29 on page 12, $[\pi]_u = \alpha = \tau_?$ holds in several cases, depending on the value of u wrt. the form of π. In each of these cases, we have that the proof is similar, *mutatis mutandis*, to the $\alpha = \tau$ case, as seen throughout this proof.

□

A.8 Proof of Lemma 42 on page 14

Proof. By definition 41 on page 14, we have that $f(u)$ is honest, for all $u \in \mathcal{N} \cup \mathcal{V} \cup \{*\}$. Then, when $f' \sqsubseteq f$, by definition 36 on page 13 we have two cases:

- $f'(u) = f(u)$. Then, $f'(u)$ is honest as well;
- $f'(u) = f(*)$. Then, $f'(u)$ is trivially honest because it cannot advertise contracts.

Thus, for all $u \in \mathcal{N} \cup \mathcal{V} \cup \{*\}$, $f'(u)$ is honest — and hence f' is honest. □

A.9 Proof of Lemma 44 on page 14

Proof. As in the proof of lemma 35 on page 13, we proceed by induction on the typing derivation of $\Gamma \vdash S\colon f$. We consider the different cases of structural equivalence between systems, and show that when the LHS (resp. RHS) is typed with f, the same premises of its typing derivation can be used to type the RHS (resp. LHS) with f.

- commutative monoidal laws for $|$:
  - $S = S' \mid S''' \equiv S''' \mid S' = S'$. Follows directly from the existence of a symmetric rule for [T-SPAR2];
  - $S = (S' \mid S''') \mid S'''' \equiv S' \mid (S''' \mid S'''') = S'$. In the LHS of the equivalence, $\vdash_\mathsf{A} S\colon f$ may be obtained via rule [T-SPAR2] in two cases:
    - $\vdash_\mathsf{A} S' \mid S'''\colon f$ and $\vdash_\mathsf{A} S'''' \rhd f$. Again, we have two cases:
      - $\vdash_\mathsf{A} S'\colon f$ and $\vdash_\mathsf{A} S''' \rhd f$. We have the following typing derivation for the LHS of the equivalence:

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

The same premises can be used for typing the RHS of the equivalence:

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

      - $\vdash_\mathsf{A} S'''\colon f$ and $\vdash_\mathsf{A} S' \rhd f$. We have the following typing derivation for the LHS of the equivalence:

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

The same premises can be used for typing the RHS of the equivalence:

$$\dfrac{\vdash_{\mathsf{A}} S'' \rhd f \qquad \dfrac{\vdash_{\mathsf{A}} S''' : f \quad \vdash_{\mathsf{A}} S'''' \rhd f}{\vdash_{\mathsf{A}} S''' \mid S'''' : f}\ {\scriptstyle [\text{T-SPAR2}]}}{\vdash_{\mathsf{A}} S' : f}\ {\scriptstyle [\text{T-SPAR2}]}$$

$*$ $\vdash_{\mathsf{A}} S'' \mid S''' \rhd f$ and $\vdash_{\mathsf{A}} S'''' : f$. We have the following typing derivation for the LHS of the equivalence:

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

The same premises can be used for typing the RHS of the equivalence:

$$\dfrac{\vdash_{\mathsf{A}} S'' \rhd f \qquad \dfrac{\vdash_{\mathsf{A}} S''' \rhd f \quad \vdash_{\mathsf{A}} S'''' : f}{\vdash_{\mathsf{A}} S''' \mid S'''' : f}\ {\scriptstyle [\text{T-SPAR2}]}}{\vdash_{\mathsf{A}} S' : f}\ {\scriptstyle [\text{T-SPAR2}]}$$

– $S = S' \mid \mathbf{0} \equiv S'$. By rule [T-SAFREE0], $\vdash_{\mathsf{A}} \mathbf{0} \rhd f$. By rule [T-SPAR2], in order to have $\vdash_{\mathsf{A}} S = S' \mid \mathbf{0} : f$ we necessarily have $\vdash_{\mathsf{A}} S' : f$;

- $S = S'' \mid (u)S''' \equiv (u)(S'' \mid S''') = S'$ if $u \notin \text{fnv}(S'')$. $\vdash_{\mathsf{A}} S : f$ could be obtained through [T-SPAR2] in two cases:
  – $\vdash_{\mathsf{A}} S'' : f$ and $\vdash_{\mathsf{A}} (u)S''' \rhd f$. From the first hypothesis and lemma 58 on page 20, since $u \notin \text{fnv}(S'')$, we obtain $f = f\{^{f(*)}/_{u}\}$. Furthermore, from lemma 60 on page 21, we have that $f(u) = f(*)$ is honest. Thus, we have the following typing derivations for the two sides of the equivalence, starting with the same premises:

$$\dfrac{\vdash_{\mathsf{A}} S'' : f \qquad \dfrac{\vdash_{\mathsf{A}} S''' \rhd f\{^{f(*)}/_{u}\}}{\vdash_{\mathsf{A}} (u)S''' \rhd f}\ {\scriptstyle [\text{T-SDEL1}]}}{\vdash_{\mathsf{A}} S'' \mid (u)S''' : f}\ {\scriptstyle [\text{T-SPAR2}]}$$

$$\dfrac{\dfrac{\vdash_{\mathsf{A}} S'' : f = f\{^{f(*)}/_{u}\} \quad \vdash_{\mathsf{A}} S''' \rhd f\{^{f(*)}/_{u}\} = f}{\vdash_{\mathsf{A}} S'' \mid S''' : f}\ {\scriptstyle [\text{T-SPAR2}]} \qquad f(u) \text{ honest}}{\vdash_{\mathsf{A}} (u)(S'' \mid S''') : f\{^{f(*)}/_{u}\} = f}\ {\scriptstyle [\text{T-SDEL2}]}$$

  – $\vdash_{\mathsf{A}} S'' \rhd f$ and $\vdash_{\mathsf{A}} (u)S''' : f$. The typing derivation is:

$$\dfrac{\vdash_{\mathsf{A}} S'' \rhd f = f'\{^{f'(*)}/_{u}\} \qquad \dfrac{\vdash_{\mathsf{A}} S''' : f' \quad f'(u) \text{ honest}}{\vdash_{\mathsf{A}} (u)S''' : f = f'\{^{f'(*)}/_{u}\}}\ {\scriptstyle [\text{T-SDEL2}]}}{\vdash_{\mathsf{A}} S'' \mid (u)S''' : f = f'\{^{f'(*)}/_{u}\}}\ {\scriptstyle [\text{T-SPAR2}]}$$

  We can use the same premises for typing the RHS of the equivalence, considering that by lemma 63 on page 22, since $u \notin \text{fnv}(S'')$, $\vdash_{\mathsf{A}} S'' \rhd f'\{^{f'(*)}/_{u}\} \implies \vdash_{\mathsf{A}} S'' \rhd f'\{^{f'(*)}/_{u}\}\{^{f'(u)}/_{u}\} = f'$;

$$\dfrac{\dfrac{\vdash_{\mathsf{A}} S'' \rhd f' \quad \vdash_{\mathsf{A}} S''' : f'}{\vdash_{\mathsf{A}} S'' \mid S''' : f'}\ {\scriptstyle [\text{T-SPAR2}]} \qquad f'(u) \text{ honest}}{\vdash_{\mathsf{A}} (u)(S'' \mid S''') : f'\{^{f'(*)}/_{u}\} = f}\ {\scriptstyle [\text{T-SDEL2}]}$$

  When reasoning in the other direction, the premises of the RHS could be reused for typing the LHS, considering that by lemma 63 on page 22, since $u \notin \text{fnv}(S'')$, $\vdash_{\mathsf{A}} S'' \rhd f' \implies \vdash_{\mathsf{A}} S'' \rhd f'\{^{f'(*)}/_{u}\} = f$;

- $S \equiv (u)S$ if $u \notin \text{fnv}(S)$. On the LHS of the equivalence we have $\vdash_{\mathsf{A}} S : f = f\{^{f(*)}/_{u}\}$ (58). We can use this premise for typing the RHS of the equivalence:

$$\dfrac{\vdash_{\mathsf{A}} S : f \quad f(u) = f(*) \text{ honest (60)}}{\vdash_{\mathsf{A}} (u)S : f\{^{f(*)}/_{u}\} = f}\ {\scriptstyle [\text{T-SDEL2}]}$$

- $(u)(v)S \equiv (v)(u)S$. The typing derivations for the two sides of the equivalence are:

$$\dfrac{\dfrac{\vdash_{\mathsf{A}} S\colon f \quad f(u) \text{ honest}}{\vdash_{\mathsf{A}} (u)S\colon f\{^{f(*)}/_{u}\}}\,{\scriptstyle [\text{T-SDEL2}]} \qquad f\{^{f(*)}/_{u}\}(v) \text{ honest}}{\vdash_{\mathsf{A}} (v)(u)S\colon f\{^{f(*)}/_{u}\}\{^{f\{^{f(*)}/_{u}\}(*)}/_{v}\} = f\{^{f(*)}/_{u}\}\{^{f(*)}/_{v}\}}\,{\scriptstyle [\text{T-SDEL2}]}$$

$$\dfrac{\dfrac{\vdash_{\mathsf{A}} S\colon f \quad f(v) \text{ honest}}{\vdash_{\mathsf{A}} (v)S\colon f\{^{f(*)}/_{v}\}}\,{\scriptstyle [\text{T-SDEL2}]} \qquad f\{^{f(*)}/_{v}\}(u) \text{ honest}}{\vdash_{\mathsf{A}} (u)(v)S\colon f\{^{f(*)}/_{v}\}\{^{f\{^{f(*)}/_{v}\}(*)}/_{u}\} = f\{^{f(*)}/_{v}\}\{^{f(*)}/_{u}\} = f\{^{f(*)}/_{u}\}\{^{f(*)}/_{v}\}}\,{\scriptstyle [\text{T-SDEL2}]}$$

  The derivations are based on the same premises. In particular, when considering "$f\{^{f(*)}/_{u}\}(v)$ honest" and "$f\{^{f(*)}/_{v}\}(u)$ honest", we have two cases:
  - $u = v$: then they both reduce to "$f(*)$ honest", which is always true (lemma 60 on page 21);
  - $u \neq v$: then they reduce respectively to "$f(v)$ honest" and "$f(u)$ honest", and each of them appears as a premise in the other derivation;
- $S = \mathsf{A}[K] \mid \mathsf{A}[K'] \equiv \mathsf{A}[K \mid K'] = S'$. The rules for $\vdash_{\mathsf{A}} \cdot\colon \cdot$ do not allow S nor S' to be typed with any f;
- $S = \mathsf{A}[(u)P] \equiv (u)\mathsf{A}[P] = S'$. The typing derivation for the LHS of the equivalence is:

$$\dfrac{\dfrac{\emptyset_{\neq u} \vdash P\colon f \quad f(u) \text{ honest}}{\emptyset \vdash (u)P\colon f\{^{f(*)}/_{u}\}}\,{\scriptstyle [\text{T-DEL}]}}{\vdash_{\mathsf{A}} \mathsf{A}[(u)P]\colon f\{^{f(*)}/_{u}\}}\,{\scriptstyle [\text{T-SA}]}$$

  By considering that $\emptyset_{\neq u} = \emptyset$, the same premises ($\emptyset \vdash P\colon f$ and $f(u)$ honest) can be used to obtain the same type for the RHS of the equivalence:

$$\dfrac{\dfrac{\emptyset \vdash P\colon f}{\vdash_{\mathsf{A}} \mathsf{A}[P]\colon f}\,{\scriptstyle [\text{T-SA}]} \qquad f(u) \text{ honest}}{\vdash_{\mathsf{A}} (u)\mathsf{A}[P]\colon f\{^{f(*)}/_{u}\}}\,{\scriptstyle [\text{T-SDEL2}]}$$

- equivalence properties:
  - $S \equiv S$: trivial;
  - $S \equiv S' \implies S' \equiv S$. As shown in all the previous cases, if $\vdash_{\mathsf{A}} S\colon f$, $\vdash_{\mathsf{A}} S'\colon f'$ and $S \equiv S'$, then the typing derivations of S and S' start from the same premises, and bring to the conclusion that $f = f'$;
  - $S \equiv S'' \wedge S'' \equiv S' \implies S \equiv S'$. Trivial, from the consideration above;
- congruence laws. Let $S \equiv S'$ and, by applying the induction hypothesis, let $\vdash_{\mathsf{A}} S\colon f^{S} \implies \vdash_{\mathsf{A}} S'\colon f^{S}$:
  - $\vdash_{\mathsf{A}} S \mid S''\colon f \implies \vdash_{\mathsf{A}} S' \mid S''\colon f$. Trivial, by rule [T-SPAR2];
  - $\vdash_{\mathsf{A}} (u)S\colon f \implies \vdash_{\mathsf{A}} (u)S'\colon f$. Trivial, by rule [T-SDEL2].

For (8), let us consider the different cases of structural equivalence between systems.

- commutative monoidal laws for $|$:
  - $S = S'' \mid S''' \equiv S''' \mid S'' = S'$. Follows directly from [T-SPAR1];
  - $S = (S'' \mid S''') \mid S'''' \equiv S'' \mid (S''' \mid S'''') = S'$. We have the following typing derivation for the LHS of the equivalence:

$$\dfrac{\dfrac{\vdash_{\mathsf{A}} S'' \rhd f \quad \vdash_{\mathsf{A}} S''' \rhd f}{\vdash_{\mathsf{A}} S'' \mid S''' \rhd f}\,{\scriptstyle [\text{T-SPAR1}]} \qquad \vdash_{\mathsf{A}} S'''' \rhd f}{\vdash_{\mathsf{A}} S \rhd f}\,{\scriptstyle [\text{T-SPAR1}]}$$

    The same premises can be used for typing the RHS of the equivalence:

$$\dfrac{\vdash_{\mathsf{A}} S'' \rhd f \qquad \dfrac{\vdash_{\mathsf{A}} S''' \rhd f \quad \vdash_{\mathsf{A}} S'''' \rhd f}{\vdash_{\mathsf{A}} S''' \mid S'''' \rhd f}\,{\scriptstyle [\text{T-SPAR1}]}}{\vdash_{\mathsf{A}} S' \rhd f}\,{\scriptstyle [\text{T-SPAR1}]}$$

  - $S = S' \mid \mathbf{0} \equiv S'$. By rule [T-SAFREE0], $\vdash_{\mathsf{A}} \mathbf{0} \rhd f$. By rule [T-SPAR1], in order to have $\vdash_{\mathsf{A}} S = S' \mid \mathbf{0} \rhd f$ we necessarily have $\vdash_{\mathsf{A}} S' \rhd f$;

- $S = S'' \mid (u)S''' \equiv (u)(S'' \mid S''') = S'$ if $u \notin \text{fnv}(S'')$. We have the following typing derivation for the LHS of the equivalence:

$$\dfrac{\dfrac{\vdash_{\mathsf{A}} S'' \rhd f \qquad \dfrac{\vdash_{\mathsf{A}} S''' \rhd f\{^{f(*)}/_{u}\}}{\vdash_{\mathsf{A}} (u)S''' \rhd f}\ \text{[T-SDEL1]}}{\vdash_{\mathsf{A}} S'' \mid (u)S''' \rhd f}\ \text{[T-SPAR1]}$$

  The same premises can be used for typing the RHS of the equivalence, considering that by lemma 63 on page 22, since $u \notin \text{fnv}(S'')$, $\vdash_{\mathsf{A}} S'' \rhd f \implies S'' \rhd f\{^{f(*)}/_{u}\}$:

$$\dfrac{\dfrac{\vdash_{\mathsf{A}} S'' \rhd f\{^{f(*)}/_{u}\} \quad \vdash_{\mathsf{A}} S''' \rhd f\{^{f(*)}/_{u}\}}{\vdash_{\mathsf{A}} S'' \mid S''' \rhd f\{^{f(*)}/_{u}\}}\ \text{[T-SPAR1]}}{\vdash_{\mathsf{A}} (u)(S'' \mid S''') \rhd f}\ \text{[T-SDEL1]}$$

  When reasoning in the other direction, the premises of the RHS could be reused for typing the LHS, considering that by lemma 63 on page 22, since $u \notin \text{fnv}(S'')$, $\vdash_{\mathsf{A}} S'' \rhd f\{^{f(*)}/_{u}\} \implies \vdash_{\mathsf{A}} S'' \rhd f\{^{f(*)}/_{u}\}\{^{f(u)}/_{u}\} = f$;
- $S \equiv (u)S$ if $u \notin \text{fnv}(S)$. On the LHS of the equivalence we have $\vdash_{\mathsf{A}} S \rhd f \implies \vdash_{\mathsf{A}} S \rhd f\{^{f(*)}/_{u}\}$ (lemma 63 on page 22). We can use this premise for typing the RHS of the equivalence:

$$\dfrac{\vdash_{\mathsf{A}} S \rhd f\{^{f(*)}/_{u}\}}{\vdash_{\mathsf{A}} (u)S \rhd f}\ \text{[T-SDEL1]}$$

  When reasoning in the other direction, the premises of the RHS could be reused for typing the LHS, by considering that by lemma 63 on page 22, since $u \notin \text{fnv}(S)$, $\vdash_{\mathsf{A}} S \rhd f\{^{f(*)}/_{u}\} \implies \vdash_{\mathsf{A}} S \rhd f\{^{f(*)}/_{u}\}\{^{f(u)}/_{u}\} = f$;
- $(u)(v)S \equiv (v)(u)S$. The typing derivations for the two sides of the equivalence are:

$$\dfrac{\dfrac{\vdash_{\mathsf{A}} S\colon f\{^{f(*)}/_{v}\}\{^{f\{^{f(*)}/_{v}\}(*)}/_{u}\} = f\{^{f(*)}/_{v}\}\{^{f(*)}/_{u}\}}{\vdash_{\mathsf{A}} (u)S \rhd f\{^{f(*)}/_{v}\}}\ \text{[T-SDEL1]}}{\vdash_{\mathsf{A}} (v)(u)S \rhd f}\ \text{[T-SDEL1]}$$

$$\dfrac{\dfrac{\vdash_{\mathsf{A}} S\colon f\{^{f(*)}/_{u}\}\{^{f\{^{f(*)}/_{u}\}(*)}/_{v}\} = f\{^{f(*)}/_{u}\}\{^{f(*)}/_{v}\} = f\{^{f(*)}/_{v}\}\{^{f(*)}/_{u}\}}{\vdash_{\mathsf{A}} (v)S \rhd f\{^{f(*)}/_{u}\}}\ \text{[T-SDEL1]}}{\vdash_{\mathsf{A}} (u)(v)S \rhd f}\ \text{[T-SDEL1]}$$

- $S = \mathsf{B}[K] \mid \mathsf{B}[K'] \equiv \mathsf{B}[K \mid K'] = S'$. The typing derivation for the two sides of the equivalence are:

$$\dfrac{\vdash_{\mathsf{A}} \mathsf{B}[K] \rhd f \quad \vdash_{\mathsf{A}} \mathsf{B}[K'] \rhd f}{\vdash_{\mathsf{A}} S \rhd f}\ \text{[T-SPAR1]}$$

$$\dfrac{\vdash_{\mathsf{A}} \mathsf{B}[K] \rhd f \quad \vdash_{\mathsf{A}} \mathsf{B}[K'] \rhd f}{\vdash_{\mathsf{A}} S' \rhd f}\ \text{[T-SFZ2]}$$

- $S = \mathsf{B}[(u)P] \equiv (u)\mathsf{B}[P] = S'$. We have two cases:
  - $\mathsf{A} \neq \mathsf{B}$. We have the following typing derivations for the two sides of the equivalence:

$$\dfrac{\mathsf{B} \neq \mathsf{A}}{\vdash_{\mathsf{A}} S \rhd f}\ \text{[T-SAFREE1]}$$

$$\dfrac{\dfrac{\mathsf{B} \neq \mathsf{A}}{\vdash_{\mathsf{A}} \mathsf{B}[P] \rhd f\{^{f(*)}/_{u}\}}\ \text{[T-SAFREE1]}}{\vdash_{\mathsf{A}} S' \rhd f}\ \text{[T-SDEL1]}$$

    where the conditions on f in the premises of the two [T-SAFREE1] instances are equivalent (i.e., if a f satisfies one, then it also satisfies the other).
  - $\mathsf{A} = \mathsf{B}$. In this case, none of the sides of the equivalence can be typed with $\vdash_{\mathsf{A}} \cdot \rhd \cdot$;
- equivalence properties:
  - $S \equiv S$. Trivial;

- $S \equiv S' \implies S' \equiv S$. As shown in all the previous cases, if $\vdash_{\mathsf{A}} S \rhd f$, $\vdash_{\mathsf{A}} S' \rhd f'$ and $S \equiv S'$, then the typing derivations of S and S' start from the same premises, and bring to the conclusion that $f = f'$;
- $S \equiv S'' \wedge S'' \equiv S' \implies S \equiv S'$. Trivial, from the consideration above;

- congruence laws. Let $S \equiv S'$ and, by applying the induction hypothesis, let $\vdash_{\mathsf{A}} S \rhd f^S \implies \vdash_{\mathsf{A}} S' \rhd f^S$:
  - $\vdash_{\mathsf{A}} S \mid S'' \rhd f \implies \vdash_{\mathsf{A}} S' \mid S'' \rhd f$. Trivial, by rule [T-SPAR1];
  - $\vdash_{\mathsf{A}} S \mid S'' \colon f \implies \vdash_{\mathsf{A}} S' \mid S'' \colon f$. Trivial, by rule [T-SPAR2];
  - $\vdash_{\mathsf{A}} (u)S \rhd f \implies \vdash_{\mathsf{A}} (u)S' \rhd f$. Trivial, by rule [T-SDEL1].

□

A.10 Proof of Lemma 45 on page 15

Proof. $(u)S$ has a typing derivation of the form:

$$\frac{\vdash_{\mathsf{A}} S \colon f' \quad f'(u) \text{ honest}}{\vdash_{\mathsf{A}} (u)S \colon f'\{^{f'(*)}/_{u}\} = f} \text{ [T-SDEL2]}$$

Thus, we have $f(u) = f'(*)$, and $\forall z \in (\mathcal{N} \cup \mathcal{V} \cup \{*\}) \setminus \{u\} \,.\, f(z) = f'(z)$. Hence, $f \sqsubseteq f'$ (definition 36 on page 13). □

A.11 Proof of Theorem 49 on page 16

Proof. We additionally prove that, for all S and f such that $\forall u \in \overline{\mathrm{fnv}_{\mathsf{B}}}(S) \,.\, f(u) = f(*)$ (definition 56 on page 19):

$$S \xrightarrow{\mathsf{A} : \pi, \sigma} S' \wedge \vdash_{\mathsf{B}} S \rhd f \implies \vdash_{\mathsf{B}} S' \rhd f \bullet \sigma \qquad (\mathsf{B} \neq \mathsf{A}) \tag{15}$$

We also point out that there could not be a fourth subject reduction case under the hypotheses $S \xrightarrow{\mathsf{A} : \pi, \sigma} S'$ and $\vdash_{\mathsf{A}} S \rhd f$, because they are mutually exclusive: by lemma 66 on page 23, $\vdash_{\mathsf{A}} S \rhd f$ implies that $\mathsf{A}[P]$ cannot appear in S, and thus S cannot reduce through A.

We proceed by induction on the reduction semantics of CO_2 (fig. 3.2 on page 5), analysing the possible typing derivations of the redex and the reduct.

- [TAU]. We have $S = \mathsf{A}[\tau.P + P' \mid Q]$ and $S \xrightarrow{\mathsf{A} : \tau, \emptyset} S' = \mathsf{A}[P \mid Q]$.
  - for item (9), the redex can only be typed via rule [T-SA], and thus $\vdash_{\mathsf{A}} S \colon f = \lambda u \,.\, [\tau]_u \,.\, f^P(u) + f^{P'}(u) \mid f^Q(u)$ (where f^P, $f^{P'}$ and f^Q are respectively the types of P, P' and Q). Let us consider the type $f' = \lambda u \,.\, f^P(u) \mid f^Q(u)$, and for all $v \in \mathcal{N} \cup \mathcal{V}$, let $T_v^P = f^P(v)$, $T_v^{P'} = f^P(v)$ and $T_v^Q = f^Q(v)$. We have:

$$f(v) = \tau \,.\, T_v^P + T_v^{P'} \mid T_v^Q$$
$$f'(v) = T_v^P \mid T_v^Q$$

    Hence, we have the following derivation on the channel type semantics:

$$\frac{\dfrac{\dfrac{}{\tau \,.\, T_v^P \xrightarrow{\tau} T_v^P} \text{ [C-ALPHA]}}{\tau \,.\, T_v^P + T_v^{P'} \xrightarrow{\tau} T_v^P} \text{ [C-SUML]}}{f(v) = \tau \,.\, T_v^P + T_v^{P'} \mid T_v^Q \xrightarrow{\tau} T_v^P \mid T_v^Q = f'(v)} \text{ [C-PARL]}$$

    and thus, by definition 38 on page 14, $f \xrightarrow{\tau} f'$. We can now use f' for concluding the proof: in fact, after the reduction $S \xrightarrow{\mathsf{A} : \tau, \emptyset} S'$, we have:

$$\frac{\dfrac{\emptyset \vdash P \colon f^P \quad \emptyset \vdash Q \colon f^Q}{\emptyset \vdash P \mid Q \colon \lambda u \,.\, f^P(u) \mid f^Q(u) = f'} \text{ [T-PAR]}}{\vdash_{\mathsf{A}} S' \colon f' = f' \bullet \emptyset} \text{ [T-SA]}$$

  - for item (10), the theorem holds vacuously: the rule only allows S to reduce via A, but for all $\mathsf{B} \neq \mathsf{A}$, $\not\vdash_{\mathsf{B}} S \colon f$ (lemma 65 on page 23);

– for item (15), according to rule [T-SAFREE1], we have $\vdash_{\mathsf{B}} S \rhd f$ and $\vdash_{\mathsf{B}} S' \rhd f$. Thus, $\vdash_{\mathsf{B}} S \rhd f \wedge S \xrightarrow{\mathsf{A}:\tau,\emptyset} S' \Longrightarrow \vdash_{\mathsf{B}} S' \rhd f = f \bullet \emptyset$.

- [TELL]. We have $S = \mathsf{A}[\mathsf{tell}_{\mathsf{C}} \downarrow_w c.P + P' \mid Q]$ and $S \xrightarrow{\mathsf{A}:\mathsf{tell}_{\mathsf{C}}\downarrow_w c,\emptyset} S' = \mathsf{A}[P \mid Q] \mid \mathsf{C}[\downarrow_w \mathsf{A}\ says\ c]$.
  – For item (9), the redex can only be typed via rule [T-SA], and thus $\vdash_{\mathsf{A}} S\colon f = \lambda u\,.\,[\mathsf{tell}_{\mathsf{C}} \downarrow_w c]_u\,.\,f^P(u) + f^{P'}(u) \mid f^Q(u)$ (where f^P, $f^{P'}$ and f^Q are respectively the types of P, P' and Q). Let us consider the type $f' = \lambda u\,.\,f^P(u) \mid f^Q(u)$, and for all $v \in \mathcal{N} \cup \mathcal{V}$, let $T_v^P = f^P(v)$, $T_v^{P'} = f^P(v)$ and $T_v^Q = f^Q(v)$. There are two possible ways in which f can move:
    * $v \neq w$. Then:
    $$f(v) = \tau\,.\,T_v^P + T_v^{P'} \mid T_v^Q$$
    $$f'(v) = T_v^P \mid T_v^Q$$
    Hence, we have the following derivation on the channel type semantics:
    $$\dfrac{\dfrac{\dfrac{}{\tau\,.\,T_v^P \xrightarrow{\tau} T_v^P}\,[\text{C-ALPHA}]}{\tau\,.\,T_v^P + T_v^{P'} \xrightarrow{\tau} T_v^P}\,[\text{C-SUML}]}{f(v) = \tau\,.\,T_v^P + T_v^{P'} \mid T_v^Q \xrightarrow{\tau} T_v^P \mid T_v^Q = f'(v)}\,[\text{C-PARL}]$$
    * $v = w$. Then we have:
    $$f(w) = \langle c \rangle\,.\,T_w^P + T_w^{P'} \mid T_w^Q$$
    $$f'(w) = T_w^P \mid T_w^Q$$
    from which we deduce $f(w) \xrightarrow{\langle c \rangle} f'(w)$.

  Thus, by definition 38 on page 14, $f \xrightarrow{\mathsf{tell}_{\mathsf{C}}\downarrow_w c} f'$. Furthermore, since f is honest by hypothesis, we have that $f(w)$ is honest (Def definition 25 on page 11) — and in particular, since $(\emptyset, f(w)) \to (\{c\}, f'(w))$ (fig. 5.2 on page 10, rule [A-TELL1]), we have that $f'(w)$ realizes c (definition 25 on page 11). We can now use f' for concluding the proof: in fact, after the reduction $S \xrightarrow{\mathsf{A}:\mathsf{tell}_{\mathsf{C}}\downarrow_w c,\emptyset} S'$, we have:
  $$\dfrac{\dfrac{\dfrac{\emptyset \vdash P\colon f^P \quad \emptyset \vdash Q\colon f^Q}{\emptyset \vdash P \mid Q\colon \lambda u\,.\,f^P(u) \mid f^Q(u) = f'}\,[\text{T-PAR}]}{\vdash_{\mathsf{A}} \mathsf{A}[P \mid Q]\colon f'}\,[\text{T-SA}] \quad \dfrac{f'(w)\ \text{realizes}\ c}{\vdash_{\mathsf{A}} \mathsf{C}[\downarrow_w \mathsf{A}\ says\ c] \rhd f'}\,[\text{T-SFZ1}]}{\vdash_{\mathsf{A}} S'\colon f' = f' \bullet \emptyset}\,[\text{T-SPAR2}]$$

  – for item (10), the theorem holds vacuously: the rule only allows S to reduce via A, but for all $\mathsf{B} \neq \mathsf{A}$, $\not\vdash_{\mathsf{B}} S\colon f$ (lemma 65 on page 23);
  – for item (15), according to rule [T-SAFREE1], we have $\vdash_{\mathsf{B}} S \rhd f$ and $\vdash_{\mathsf{B}} S' \rhd f$. Thus, $\vdash_{\mathsf{B}} S \rhd f \wedge S \xrightarrow{\mathsf{A}:\mathsf{tell}_{\mathsf{C}}\downarrow_w c,\emptyset} S' \Longrightarrow \vdash_{\mathsf{B}} S' \rhd f = f \bullet \emptyset$.

- [FUSE]. We have:
  $$\dfrac{K \rhd^{\sigma} \gamma \quad \operatorname{ran}\sigma = \{s\} \quad s\ \text{fresh}}{S = \mathsf{A}[\mathsf{fuse}\,.\,P + P' \mid Q] \mid \mathsf{A}[K] \xrightarrow{\mathsf{A}:\mathsf{fuse},\sigma} \mathsf{A}[P \mid Q]\sigma \mid s[\gamma\sigma] = S'}\,[\text{FUSE}]$$
  where, according to definition 3 on page 5, $K = \downarrow_x \mathsf{A}\ says\ c \mid \downarrow_y \mathsf{C}\ says\ d$, $\sigma = \{s/x,y\}$ and $\gamma = \gamma\sigma = \mathsf{A}\ says\ c \mid \mathsf{C}\ says\ d$. Without loss of generality, we assume that variable x is retained by A, while y is retained by C.
  – For item (9), the redex can only be typed as shown in fig. A.3a on the next page, where the premise "$f(x)$ realizes c" must hold because otherwise S would not be typeable, thus contradicting the theorem hypothesis. Let us consider the type $f' = \lambda u\,.\,f^P(u) \mid f^Q(u)$, and for all $v \in \mathcal{N} \cup \mathcal{V}$, let $T_v^P = f^P(v)$, $T_v^{P'} = f^P(v)$ and $T_v^Q = f^Q(v)$. We have:
  $$f(v) = \tau_? \,.\, T_v^P + T_v^{P'} \mid T_v^Q$$
  $$f'(v) = T_v^P \mid T_v^Q$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\emptyset \vdash P\colon f^P \quad \emptyset \vdash P'\colon f^{P'}}{\emptyset \vdash \mathsf{fuse}.P + P'\colon \lambda u\,.\,[\mathsf{fuse}]_u\,.\,f^P(u) + f^{P'}(u)}\ \text{[T-Sum]} \qquad \emptyset \vdash Q\colon f^Q}{\emptyset \vdash \mathsf{fuse}.P + P' \mid Q\colon \lambda u\,.\,[\mathsf{fuse}]_u.f^P(u) + f^{P'}(u) \mid f^Q(u) = f}\ \text{[T-Par]}}{\vdash_{\mathsf{A}} \mathsf{A}[\mathsf{fuse}.P + P' \mid Q]\colon f}\ \text{[T-SA]} \qquad \dfrac{\dfrac{f(x) \text{ realizes } c}{\vdash_{\mathsf{A}} \mathsf{A}[\downarrow_x \mathsf{A}\ \mathit{says}\ c] \rhd f}\ \text{[T-SFz1]} \qquad \dfrac{\mathsf{C} \neq \mathsf{A}}{\vdash_{\mathsf{A}} \mathsf{A}[\downarrow_y \mathsf{C}\ \mathit{says}\ d] \rhd f}\ \text{[T-SAFree2]}}{\vdash_{\mathsf{A}} K \rhd f}\ \text{[T-SFz2]}}{\vdash_{\mathsf{A}} S\colon f}\ \text{[T-SPar2]}$$

(a)

$$\dfrac{\dfrac{\dfrac{\emptyset \vdash P\sigma\colon f^P \bullet \sigma \quad \emptyset \vdash Q\sigma\colon f^Q \bullet \sigma}{\emptyset \vdash P\sigma \mid Q\sigma\colon \lambda u\,.\,f^P \bullet \sigma(u) \mid f^Q \bullet \sigma(u) = f' \bullet \sigma}\ \text{[T-Par]}}{\vdash_{\mathsf{A}} \mathsf{A}[P\sigma \mid Q\sigma] = \mathsf{A}[P \mid Q]\sigma\colon f' \bullet \sigma}\ \text{[T-SA]} \qquad \dfrac{f' \bullet \sigma(s) \text{ realizes } c\sigma \quad (\mathsf{C}\ \mathit{says}\ d)\,\sigma \text{ A-free}}{\vdash_{\mathsf{A}} s[(\mathsf{A}\ \mathit{says}\ c \mid \mathsf{C}\ \mathit{says}\ d)\sigma] \rhd f' \bullet \sigma}\ \text{[T-SFused]}}{\vdash_{\mathsf{A}} S'\colon f' \bullet \sigma}\ \text{[T-SPar2]}$$

(b)

Figure A.3: Typing derivations for subject reduction proof ([Fuse] rule).

Hence, we have the following derivation on the channel type semantics:

$$\dfrac{\dfrac{\dfrac{}{\tau_? \,.\, T_v^P \xrightarrow{\tau_?} T_v^P}\,[\text{C-Alpha}]}{\tau_? \,.\, T_v^P + T_v^{P'} \xrightarrow{\tau} T_v^P}\,[\text{C-SumL}]}{f(v) = \tau \,.\, T_v^P + T_v^{P'} \mid T_v^Q \xrightarrow{\tau_?} T_v^P \mid T_v^Q = f'(v)}\,[\text{C-ParL}]$$

and thus, by 38, $f \xrightarrow{\tau_?} f'$. By lemma 57 on page 20, we have that, when $f(x)$ realizes c (as required for typing S), then also $f'(x)$ realizes c. This, in turn, implies that $f'(x) = f'\{^{f'(x)}/_s\}(s) = f' \bullet \{^s/_{x,y}\}(s) = f' \bullet \sigma(s)$ realizes c. Furthermore, by lemma 70 on page 24, we have:

$$\begin{aligned} \emptyset \vdash P\colon f^P &\implies \emptyset \bullet \sigma = \emptyset \vdash P\sigma\colon f^P \bullet \sigma \\ \emptyset \vdash Q\colon f^Q &\implies \emptyset \bullet \sigma = \emptyset \vdash Q\sigma\colon f^Q \bullet \sigma \end{aligned}$$

We can now use $f' \bullet \sigma$ for typing the reduct S': in fact, considering that $\mathsf{A}[P \mid Q]\sigma = \mathsf{A}[P\sigma \mid Q\sigma]$, we have the typing derivation shown in fig. A.3b on the previous page;

– for item (10), the theorem holds vacuously: the rule only allows S to reduce via A, but for all $\mathsf{B} \neq \mathsf{A}$, $\not\vdash_{\mathsf{B}} S\colon f$ (lemma 65 on page 23);

– for item (15), we observe that the condition on f guarantees that $f \bullet \sigma$ is always defined: σ can map at most two variables to s (e.g. , x and y), and by variable retention, at most one of them (e.g. , y) can appear outside S (and thus, $f(x) = f(*)$, as required by definition 48 on page 16). The redex S can be typed in two ways:

  ∗ when considering participant $\mathsf{B} \neq \mathsf{A}$ with a latent contract in K (i.e., $\mathsf{C} = \mathsf{B}$):

$$\dfrac{\dfrac{\mathsf{A} \neq \mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[\mathsf{fuse}.P + P' \mid Q] \rhd f}\,[\text{T-SAFree2}] \qquad \dfrac{\dfrac{\mathsf{A} \neq \mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[\downarrow_x \mathsf{A}\ \mathit{says}\ c] \rhd f}\,[\text{T-SAFree2}] \qquad \dfrac{f(y) \text{ realizes } d}{\vdash_{\mathsf{B}} \mathsf{A}[\downarrow_y \mathsf{B}\ \mathit{says}\ d] \rhd f}\,[\text{T-SFz1}]}{\vdash_{\mathsf{B}} K \rhd f}\,[\text{T-SFz2}]}{\vdash_{\mathsf{B}} S \rhd f}\,[\text{T-SPar2}]$$

  where the premise "$f(y)$ realizes d" must hold because otherwise S would not be typeable, thus contradicting the theorem hypothesis. This, in turn, implies that $f(y) = f\{^{f(y)}/_s\}(s) = f \bullet \{^s/_{x,y}\}(s) = f \bullet \sigma(s)$ realizes d as well. Thus, we have that the reduct S' can be typed as:

$$\dfrac{\dfrac{\mathsf{A} \neq \mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[P \mid Q]\sigma \rhd f \bullet \sigma}\,[\text{T-SAFree2}] \qquad \dfrac{(\mathsf{A}\ \mathit{says}\ c)\,\sigma \text{ B-free} \quad f \bullet \sigma(s) \text{ realizes } d\sigma}{\vdash_{\mathsf{B}} s[(\mathsf{A}\ \mathit{says}\ c \mid \mathsf{C}\ \mathit{says}\ d)\,\sigma] \rhd f \bullet \sigma}\,[\text{T-SFused}]}{\vdash_{\mathsf{B}} S' \rhd f \bullet \sigma}\,[\text{T-SPar1}]$$

  ∗ when considering participant $\mathsf{B} \neq \mathsf{C}$ with no latent contracts in K:

$$\dfrac{\dfrac{\mathsf{A} \neq \mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[\mathsf{fuse}.P + P' \mid Q] \rhd f}\,[\text{T-SAFree2}] \qquad \dfrac{\dfrac{\mathsf{A} \neq \mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[\downarrow_x \mathsf{A}\ \mathit{says}\ c] \rhd f}\,[\text{T-SAFree2}] \qquad \dfrac{\mathsf{A} \neq \mathsf{C}}{\vdash_{\mathsf{B}} \mathsf{A}[\downarrow_y \mathsf{C}\ \mathit{says}\ d] \rhd f}\,[\text{T-SAFree2}]}{\vdash_{\mathsf{B}} K \rhd f}\,[\text{T-SFz2}]}{\vdash_{\mathsf{B}} S \rhd f}\,[\text{T-SPar2}]$$

  while the reduct S' can be typed as:

$$\dfrac{\dfrac{\mathsf{A} \neq \mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[P \mid Q]\sigma \rhd f \bullet \sigma}\,[\text{T-SAFree2}] \qquad \dfrac{(\mathsf{A}\ \mathit{says}\ c \mid \mathsf{C}\ \mathit{says}\ d)\,\sigma \text{ B-free}}{\vdash_{\mathsf{B}} s[(\mathsf{A}\ \mathit{says}\ c \mid \mathsf{C}\ \mathit{says}\ d)\,\sigma] \rhd f \bullet \sigma}\,[\text{T-SAFree3}]}{\vdash_{\mathsf{B}} S' \rhd f \bullet \sigma}\,[\text{T-SPar2}]$$

• [Do]. We have:

$$\dfrac{(\mathsf{A}\ \mathit{says}\ c \mid \gamma) \xrightarrow{\mathsf{A}\ \mathit{says}\ \mathsf{a}} (\mathsf{A}\ \mathit{says}\ c' \mid \gamma)}{S = \mathsf{A}[\mathsf{do}_s\,\mathsf{a}.P + P' \mid Q] \;\mid\; s[\mathsf{A}\ \mathit{says}\ c \mid \gamma] \xrightarrow{\mathsf{A} : \mathsf{do}_s\,\mathsf{a}, \emptyset} \mathsf{A}[P \mid Q] \;\mid\; s[\mathsf{A}\ \mathit{says}\ c' \mid \gamma] = S'}\,[\text{Do}]$$

– For item (9), the redex can only be typed with $f = \lambda u\,.\,[\mathsf{do}_s\,\mathsf{a}]_u\,.\,f^P(u) + f^{P'}(u) \mid f^Q(u)$, according to the derivation:

$$\dfrac{\dfrac{\dfrac{\vdots}{\emptyset \vdash \mathsf{do}_s\,\mathsf{a}.P + P' \mid Q\colon f}\,\text{[T-Par]}}{\vdash_{\mathsf{A}} \mathsf{A}[\mathsf{do}_s\,\mathsf{a}.P + P' \mid Q]\colon f}\,\text{[T-SA]} \quad \dfrac{f(s)\text{ realizes } c \quad \gamma\,\mathsf{A}\text{-free}}{\vdash_{\mathsf{A}} s[\mathsf{A}\ \mathit{says}\ c \mid \gamma] \rhd f}\,\text{[T-SFused]}}{\vdash_{\mathsf{A}} S\colon f}\,\text{[T-SPar2]}$$

where the premise "$f(s)$ realizes c" must hold (because otherwise S would not be typeable, thus contradicting the theorem hypothesis), and f^P, $f^{P'}$ and f^Q are respectively the types of P, P' and Q. Let us consider the type $f' = \lambda u\,.\,f^P(u) \mid f^Q(u)$, and for all $v \in \mathcal{N} \cup \mathcal{V}$, let $T_v^P = f^P(v)$, $T_v^{P'} = f^P(v)$ and $T_v^Q = f^Q(v)$. There are two possible ways in which f can move:

∗ $v = s$. Then:

$$f(s) = \mathsf{a}.T_s^P + T_s^{P'} \mid T_s^Q$$
$$f'(s) = T_s^P \mid T_s^Q$$

Hence, we have the following derivation on the channel type semantics:

$$\dfrac{\dfrac{\dfrac{}{\mathsf{a}\,.\,T_s^P \xrightarrow{\mathsf{a}} T_s^P}\,\text{[C-Alpha]}}{\mathsf{a}\,.\,T_s^P + T_s^{P'} \xrightarrow{\tau} T_s^P}\,\text{[C-SumL]}}{f(s) = \mathsf{a}\,.\,T_s^P + T_s^{P'} \mid T_s^Q \xrightarrow{\mathsf{a}} T_s^P \mid T_s^Q = f'(s)}\,\text{[C-ParL]}$$

∗ $v \neq s$. Then we have:

$$f(v) = \tau_?\,.\,T_v^P + T_v^{P'} \mid T_v^Q$$
$$f'(v) = T_v^P \mid T_v^Q$$

from which we deduce $f(v) \xrightarrow{\tau_?} f'(v)$.

Thus, by definition 38 on page 14, $f \xrightarrow{\mathsf{do}_s\,\mathsf{a}} f'$. We can now use f' for concluding the proof: in fact, after the reduction $S \xrightarrow{\mathsf{A}\,:\,\mathsf{do}_s\,\mathsf{a},\emptyset} S'$, we have:

$$\dfrac{\dfrac{\dfrac{\emptyset \vdash P\colon f^P \quad \emptyset \vdash Q\colon f^Q}{\emptyset \vdash P \mid Q\colon \lambda u\,.\,f^P(u) \mid f^Q(u) = f'}\,\text{[T-Par]}}{\vdash_{\mathsf{A}} \mathsf{A}[P \mid Q]\colon f'}\,\text{[T-SA]} \quad \dfrac{f'(s)\text{ realizes } c' \quad \gamma\,\mathsf{A}\text{-free}}{\vdash_{\mathsf{A}} s[\mathsf{A}\ \mathit{says}\ c' \mid \gamma] \rhd f}\,\text{[T-SFused]}}{\vdash_{\mathsf{A}} S'\colon f' = f' \bullet \emptyset}\,\text{[T-SPar2]}$$

where the premise "$f'(s)$ realizes c'" holds because:

1. $f \xrightarrow{\mathsf{do}_s\,\mathsf{a}} f'$ implies $f(s) \xrightarrow{\mathsf{a}} f'(s)$;
2. [Do] rule premises make c reduce to c' through a transition of type $\xrightarrow{\mathsf{A}\ \mathit{says}\ \mathsf{a}}\!\!\!\!\!\rightarrow$;
3. thus, $c \xrightarrow{\mathsf{a}}\!\!\!\!\!\rightarrow_\sharp c'$, which enables the abstract process reduction $(f(s), c) \to (f'(s), c')$ (by rule [A-Do] in fig. 5.2 on page 10);
4. finally, since $f(s)$ realizes c (as shown above), by definition 25 on page 11, we have that $f'(s)$ realizes c'.

– for item (10), the theorem holds vacuously: the rule only allows S to reduce via A, but for all $\mathsf{B} \neq \mathsf{A}$, $\nvdash_{\mathsf{B}} S\colon f$ (lemma 65 on page 23);

– for item (15), according to rule [T-SAFree1], we have $\vdash_{\mathsf{B}} S \rhd f$ and $\vdash_{\mathsf{B}} S' \rhd f$. Thus, $\vdash_{\mathsf{B}} S \rhd f \wedge S \xrightarrow{\mathsf{A}\,:\,\mathsf{do}_s\,\mathsf{a},\emptyset} S' \implies \vdash_{\mathsf{B}} S' \rhd f = f \bullet \emptyset$.

• [Ask]. We have:

$$\dfrac{(\mathsf{A}\ \mathit{says}\ c \mid \gamma) \vdash \phi}{S = \mathsf{A}[\mathsf{ask}_s\,\phi.P + P' \mid Q] \ \mid\ s[\mathsf{A}\ \mathit{says}\ c \mid \gamma] \xrightarrow{\mathsf{A}\,:\,\mathsf{ask}_s\,\phi,\emptyset} \mathsf{A}[P \mid Q] \ \mid\ s[\mathsf{A}\ \mathit{says}\ c \mid \gamma] = S'}\,\text{[Ask]}$$

– For item (9), the redex can only be typed with $f = \lambda u\,.\,[\mathsf{ask}_s\,\phi]_u\,.\,f^P(u) + f^{P'}(u) \mid f^Q(u)$, according to the derivation:

$$\dfrac{\dfrac{\dfrac{\vdots}{\emptyset \vdash \mathsf{ask}_s\,\phi.P + P' \mid Q\colon f}\,\text{[T-Par]}}{\vdash_{\mathsf{A}} \mathsf{A}[\mathsf{ask}_s\,\phi.P + P' \mid Q]\colon f}\,\text{[T-SA]} \quad \dfrac{f(s)\text{ realizes }c \quad \gamma\,\mathsf{A}\text{-free}}{\vdash_{\mathsf{A}} s[\mathsf{A}\ \mathit{says}\ c \mid \gamma] \rhd f}\,\text{[T-SFused]}}{\vdash_{\mathsf{A}} S\colon f}\,\text{[T-SPar2]}$$

where the premise "$f(s)$ realizes c" must hold (because otherwise S would not be typeable, thus contradicting the theorem hypothesis), and f^P, $f^{P'}$ and f^Q are respectively the types of P, P' and Q. Let us consider the type $f' = \lambda u\,.\,f^P(u) \mid f^Q(u)$, and for all $v \in \mathcal{N} \cup \mathcal{V}$, let $T_v^P = f^P(v)$, $T_v^{P'} = f^{P'}(v)$ and $T_v^Q = f^Q(v)$. There are two possible ways in which f can move:

∗ $v = s$. Then:

$$f(s) = \tau_\phi.T_s^P + T_s^{P'} \mid T_s^Q$$
$$f'(s) = T_s^P \mid T_s^Q$$

Hence, we have the following derivation on the channel type semantics:

$$\dfrac{\dfrac{\dfrac{}{\tau_\phi\,.\,T_s^P \xrightarrow{\tau_\phi} T_s^P}\,\text{[C-Alpha]}}{\tau_\phi\,.\,T_s^P + T_s^{P'} \xrightarrow{\tau_\phi} T_s^P}\,\text{[C-SumL]}}{f(s) = \tau_\phi\,.\,T_s^P + T_s^{P'} \mid T_s^Q \xrightarrow{\tau_\phi} T_s^P \mid T_s^Q = f'(s)}\,\text{[C-ParL]}$$

∗ $v \neq s$. Then we have:

$$f(v) = \tau_?\,.\,T_v^P + T_v^{P'} \mid T_v^Q$$
$$f'(v) = T_v^P \mid T_v^Q$$

from which we deduce $f(v) \xrightarrow{\tau_?} f'(v)$.

Thus, by definition 38 on page 14, $f \xrightarrow{\mathsf{ask}_s\,\phi} f'$. We can now use f' for concluding the proof: in fact, after the reduction $S \xrightarrow{\mathsf{A}\,:\,\mathsf{ask}_s\,\phi,\emptyset} S'$, we have:

$$\dfrac{\dfrac{\dfrac{\emptyset \vdash P\colon f^P \quad \emptyset \vdash Q\colon f^Q}{\emptyset \vdash P \mid Q\colon \lambda u\,.\,f^P(u) \mid f^Q(u) = f'}\,\text{[T-Par]}}{\vdash_{\mathsf{A}} \mathsf{A}[P \mid Q]\colon f'}\,\text{[T-SA]} \quad \dfrac{f'(s)\text{ realizes }c \quad \gamma\,\mathsf{A}\text{-free}}{\vdash_{\mathsf{A}} s[\mathsf{A}\ \mathit{says}\ c \mid \gamma] \rhd f}\,\text{[T-SFused]}}{\vdash_{\mathsf{A}} S'\colon f' = f' \bullet \emptyset}\,\text{[T-SPar2]}$$

where the premise "$f'(s)$ realizes c" holds because $f(s)$ realizes c (as seen above), $f(s) \xrightarrow{\tau_\phi} f'(s)$ and lemma 57 on page 20;

– for item (10), the theorem holds vacuously: the rule only allows S to reduce via A, but for all $\mathsf{B} \neq \mathsf{A}$, $\not\vdash_{\mathsf{B}} S\colon f$ (lemma 65 on page 23);

– for item (15), according to rule [T-SAFree1], we have $\vdash_{\mathsf{B}} S \rhd f$ and $\vdash_{\mathsf{B}} S' \rhd f$. Thus, $\vdash_{\mathsf{B}} S \rhd f \wedge S \xrightarrow{\mathsf{A}\,:\,\mathsf{ask}_s\,\phi,\emptyset} S' \implies \vdash_{\mathsf{B}} S' \rhd f = f \bullet \emptyset$.

• [Del1]. We have:

$$\dfrac{S'' \xrightarrow{\mathsf{A}\,:\,\pi,\{s/x\}} S'''}{S = (x)S'' \xrightarrow{\mathsf{A}\,:\,\pi,\emptyset} (s)S''' = S'}\,\text{[Del1]}$$

We observe that the rule premise contains a non-empty substitution, which can only descend from a [Fuse] (lemma 71 on page 25). Therefore, we have that $\pi = \mathsf{fuse}$, s is fresh, and thus $s \neq x$, $s \notin \mathrm{fnv}(S'')$, and finally $s \notin \mathrm{fnv}(S)$.

– for item (9), let $\vdash_{\mathsf{A}} S'' \colon f''$, and applying the induction hypothesis, let us consider f''' such that $f'' \xrightarrow{\pi} f'''$ and $\vdash_{\mathsf{A}} S''' \colon f''' \bullet \{s/x\}$. The redex can only be typed as:

$$\frac{\vdash_{\mathsf{A}} S'' \colon f'' \quad f''(x) \text{ honest}}{\vdash_{\mathsf{A}} S = (x)S'' \colon f''\{f''(*)/x\} = f} \text{ [T-SDEL2]}$$

where the premise "$f''(x)$ honest" must hold, because otherwise S would not be typeable, thus contradicting the theorem hypothesis.
Let us consider the type $f' = f'''\{f'''(*)/x\} \bullet \{s/x\}$. We have that:

* for all $u \neq x$ and $u \neq s$, then clearly $f(u) = f''(u) \xrightarrow{[\pi]_u} f'''(u) = f'(u)$;
* $f(x) = f''(*) \xrightarrow{[\pi]_*} f'''(*) = (f'''\{f'''(*)/x\} \bullet \{s/x\})(x) = f'(x)$ (due to the way the substitution works in $f' = f'''\{f'''(*)/x\} \bullet \{s/x\}$ — see definition 48 on page 16);
* $f(s) = f''(*) \xrightarrow{[\pi]_*} f'''(*) = (f'''\{f'''(*)/x\} \bullet \{s/x\})(s) = f'(s)$, due to lemma 58 on page 20 (considering that $s \notin \mathrm{fnv}(S)$ and $s \notin \mathrm{fnv}(S'')$).

and thus for all $u \in \mathcal{V} \cup \mathcal{N}$, we have $f(u) \xrightarrow{[\pi]_u} f'(u)$, i.e. $f \xrightarrow{\pi} f'$ (definition 38 on page 14).
We can now use $f' \bullet \emptyset$ for typing the reduct S':

$$\frac{\vdash_{\mathsf{A}} S''' \colon f''' \bullet \{s/x\} \quad (f''' \bullet \{s/x\})(s) \text{ honest}}{\vdash_{\mathsf{A}} S' = (s)S''' \colon (f''' \bullet \{s/x\})\{(f''' \bullet \{s/x\})(*)/s\} = f'''\{f'''(*)/x\} \bullet \{s/x\} = f' = f' \bullet \emptyset} \text{ [T-SDEL2]}$$

where the premise "$(f''' \bullet \{s/x\})(s)$ honest" is implied by the redex premise "$f''(x)$ honest", since $f'' \xrightarrow{\pi} f'''$ implies $f''(x) \xrightarrow{[\pi]_x} f'''(x)$ (definition 38 on page 14), "$f''(x)$ honest" and $[\pi]_x = \tau_?$ imply "$f'''(x)$ honest" (lemma 72 on page 25) and $f'''(x) = (f''' \bullet \{s/x\})(s)$;
– for item (10), let $\vdash_{\mathsf{B}} S'' \colon f''$, and applying the induction hypothesis, let $\vdash_{\mathsf{A}} S''' \colon f'' \bullet \{s/x\}$. The redex can only be typed as:

$$\frac{\vdash_{\mathsf{B}} S'' \colon f'' \quad f''(x) \text{ honest}}{\vdash_{\mathsf{B}} S = (x)S'' \colon f''\{f''(*)/x\} = f} \text{ [T-SDEL2]}$$

where the premise "$f''(x)$ honest" must hold, because otherwise S would not be typeable, thus contradicting the theorem hypothesis.
We observe that $f \bullet \{s/x\} = f = f \bullet \emptyset$, since:

* for all $u \neq x$ and $u \neq s$, then clearly $f(u) = (f \bullet \{s/x\})(u)$;
* $f(x) = f''(*) = (f \bullet \{s/x\})(x)$ (due to the way the substitution works — see definition 48 on page 16);
* $f(s) = f(*) = f''(*) = f(x) = (f \bullet \{s/x\})(s)$, due to lemma 58 on page 20 (considering that $s \notin \mathrm{fnv}(S)$ and $s \notin \mathrm{fnv}(S'')$).

We can now use $f \bullet \emptyset$ for typing the reduct S':

$$\frac{\vdash_{\mathsf{B}} S''' \colon f'' \bullet \{s/x\} \quad (f'' \bullet \{s/x\})(s) \text{ honest}}{\vdash_{\mathsf{B}} S' = (s)S''' \colon (f'' \bullet \{s/x\})\{(f'' \bullet \{s/x\})(*)/s\} = f''\{f''(*)/x\} \bullet \{s/x\} = f \bullet \{s/x\} = f \bullet \emptyset} \text{ [T-SDEL2]}$$

where the premise "$(f'' \bullet \{s/x\})(s)$ honest" is implied by the redex premise "$f''(x)$ honest" (since $f''(x) = (f'' \bullet \{s/x\})(s)$);
– for item (15), let the redex be typed as:

$$\frac{\vdash_{\mathsf{B}} S'' \rhd f\{f(*)/x\}}{\vdash_{\mathsf{B}} S = (x)S'' \rhd f} \text{ [T-SDEL1]}$$

By applying the induction hypothesis, the reduct of S'' in the [DEL1] rule premise can be typed as $\vdash_{\mathsf{B}} S''' \rhd f\{f(*)/x\} \bullet \{s/x\}$. From this, we can build the following typing derivation:

$$\frac{\dfrac{\vdash_{\mathsf{B}} S''' \rhd f\{f(*)/x\} \bullet \{s/x\} = (f\{f(*)/x\})\{(f\{f(*)/x\})(*)/s\}}{\vdash_{\mathsf{B}} (s)S''' \rhd f\{f(*)/x\}} \text{ [T-SDEL1]}}{\vdash_{\mathsf{B}} (x)(s)S''' \rhd f = f \bullet \emptyset} \text{ [T-SDEL1]}$$

We observe that $(x)(s)S''' \equiv (s)S'''$, since the reduction $S'' \xrightarrow{\mathsf{A}:\pi,\{s/x\}} S'''$ removes all the occurrences of x (lemma 73 on page 25), and thus $x \notin \mathrm{fnv}((s)S''')$. Finally, we exploit the persistence of typing wrt. structural equivalence (lemma 44 on page 14) to conclude:

$$\vdash_{\mathsf{B}} (x)(s)S''' \equiv (s)S''' = S' \rhd f \bullet \emptyset$$

- [DEL2]. We have:

$$\frac{S'' \xrightarrow{\mathsf{A}:\pi,\sigma} S''' \quad u \notin \mathrm{ran}\,\sigma}{S = (u)S'' \xrightarrow{\mathsf{A}:\pi,\sigma_{\neq u}} (u)S''' = S'} \text{[DEL2]}$$

We observe that, when σ is not empty, it must have been generated by a previous [FUSE] application (lemma 71 on page 25): therefore, we have $\pi = \mathsf{fuse}$, and σ maps some variables to a fresh s — i.e., $s \neq u$, $s \notin \mathrm{fnv}(S'')$, and finally $s \notin \mathrm{fnv}(S)$.

  - for item (9), let $\vdash_{\mathsf{A}} S'' : f''$, and applying the induction hypothesis, let us consider f''' such that $f'' \xrightarrow{\pi} f'''$ and $\vdash_{\mathsf{A}} S''' : f''' \bullet \sigma$. The redex can only be typed as:

$$\frac{\vdash_{\mathsf{A}} S'' : f'' \quad f''(u) \text{ honest}}{\vdash_{\mathsf{A}} S = (u)S'' : f''\{^{f''(*)}/_{u}\} = f} \text{[T-SDEL2]}$$

    where the premise "$f''(u)$ honest" must hold, because otherwise S would not be typeable, thus contradicting the theorem hypothesis.
    Let us consider the type $f' = f'''\{^{f'''(*)}/_{u}\}$. We have that:
    * for all $v \neq u$, then clearly $f(v) = f''(v) \xrightarrow{[\pi]_v} f'''(v) = f'(v)$;
    * $f(u) = f''(*) \xrightarrow{[\pi]_*} f'''(*) = (f'''\{^{f'''(*)}/_{u}\})(u) = f'(u)$.

    and thus $\forall v \in \mathcal{V} \cup \mathcal{N} \,.\, f(v) \xrightarrow{[\pi]_v} f'(v)$, i.e. $f \xrightarrow{\pi} f'$ (definition 38 on page 14).
    We also observe that the equality $f' \bullet \sigma = f' \bullet \sigma_{\neq u}$ holds:
    * if $u \notin \mathrm{dom}\,\sigma$, it holds trivially;
    * otherwise, σ maps u to a fresh s, and then we have three cases:
      - for all $v \neq u$ and $v \neq s$, then clearly $(f' \bullet \sigma)(v) = (f' \bullet \sigma_{\neq u})(v)$;
      - $(f' \bullet \sigma)(u) = f'(*) = f'''(*) = (f'''\{^{f'''(*)}/_{u}\})(u) = f'(u) = (f' \bullet \sigma_{\neq u})(u)$ (considering the way the substitution works — see definition 48 on page 16);
      - considering that $f''(s) = f''(*)$ (lemma 58 on page 20, we have that $f'' \xrightarrow{\pi} f'''$ (by induction hypothesis) implies $f''(s) = f''(*) \xrightarrow{[\pi]_*} f'''(*) = f'''(s)$; thus, $(f' \bullet \sigma)(s) = f'(u) = (f'''\{^{f'''(*)}/_{u}\})(u) = f'''(*) = f'''(s) = (f' \bullet \sigma_{\neq u})(s)$.

    We can now use $f' \bullet \sigma_{\neq u}$ for typing the reduct S':

$$\frac{\vdash_{\mathsf{A}} S''' : f''' \bullet \sigma \quad (f''' \bullet \sigma)(u) \text{ honest}}{\vdash_{\mathsf{A}} S' = (u)S''' : (f''' \bullet \sigma)\{^{(f''' \bullet \sigma)(*)}/_{u}\} = f'''\{^{f'''(*)}/_{u}\} \bullet \sigma = f' \bullet \sigma = f' \bullet \sigma_{\neq u}} \text{[T-SDEL2]}$$

    where the premise "$(f''' \bullet \sigma)(u)$ honest" holds, either because:
    * if $u \in \mathrm{dom}\,\sigma$, then $(f''' \bullet \sigma)(u) = f'''(*)$ (definition 48 on page 16), and $f'''(*)$ is honest by lemma 60 on page 21;
    * otherwise, if $u \notin \mathrm{dom}\,\sigma$, then it is implied by the redex premise "$f''(u)$ honest", since $f'' \xrightarrow{\pi} f'''$ implies $f''(u) \xrightarrow{[\pi]_u} f'''(u)$ (definition 38 on page 14), "$f''(u)$ honest" and $[\pi]_x = \tau_?$ imply "$f'''(u)$ honest" (lemma 72 on page 25) and $f'''(u) = (f''' \bullet \sigma)(u)$;
  - for item (10), let $\vdash_{\mathsf{B}} S'' : f''$, and applying the induction hypothesis, let $\vdash_{\mathsf{A}} S''' : f'' \bullet \sigma$. The redex can only be typed as:

$$\frac{\vdash_{\mathsf{B}} S'' : f'' \quad f''(u) \text{ honest}}{\vdash_{\mathsf{B}} S = (u)S'' : f''\{^{f''(*)}/_{u}\} = f} \text{[T-SDEL2]}$$

    where the premise "$f''(u)$ honest" must hold, because otherwise S would not be typeable, thus contradicting the theorem hypothesis.
    We observe that $f \bullet \sigma = f \bullet \sigma_{\neq u}$, since:

* if $u \notin \mathrm{dom}\,\sigma$, it holds trivially;
* otherwise, σ maps u to a fresh s, and then we have three cases:
  - for all $v \neq u$ and $v \neq s$, then clearly $(f \bullet \sigma)(v) = (f \bullet \sigma_{\neq u})(v)$;
  - $(f \bullet \sigma)(u) = f(*) = f''(*) = (f''\{^{f''(*)}/_{u}\})(u) = f(u) = (f \bullet \sigma_{\neq u})(u)$ (due to the way the substitution works — see definition 48 on page 16);
  - considering that $f''(s) = f''(*)$ (lemma 58 on page 20, we have $(f \bullet \sigma)(s) = f(u) = (f''\{^{f''(*)}/_{u}\})(u) = f''(*) = f''(s) = (f \bullet \sigma_{\neq u})(s)$.

We can now use $f \bullet \sigma_{\neq u}$ for typing the reduct S':

$$\frac{\vdash_{\mathsf{B}} S''' \colon f'' \bullet \sigma \quad (f'' \bullet \sigma)(u) \text{ honest}}{\vdash_{\mathsf{B}} S' = (u)S''' \colon (f'' \bullet \sigma)\{^{(f'' \bullet \sigma)(*)}/_{u}\} = f''\{^{f''(*)}/_{u}\} \bullet \sigma = f \bullet \sigma = f \bullet \sigma_{\neq u}} \text{ [T-SDEL2]}$$

where the premise "$(f'' \bullet \sigma)(u)$ honest" holds, either because:

* if $u \in \mathrm{dom}\,\sigma$, then $(f'' \bullet \sigma)(u) = f''(*)$ (definition 48 on page 16), and $f''(*)$ is honest by lemma 60 on page 21;
* otherwise, if $u \notin \mathrm{dom}\,\sigma$, then it is implied by the redex premise "$f''(u)$ honest";

– for item (15), let the redex be typed as:

$$\frac{\vdash_{\mathsf{B}} S'' \rhd f\{^{f(*)}/_{u}\}}{\vdash_{\mathsf{B}} S = (u)S'' \rhd f} \text{ [T-SDEL1]}$$

By applying the induction hypothesis, the reduct of S'' in the [DEL2] rule premise can be typed as $\vdash_{\mathsf{B}} S''' \rhd f\{^{f(*)}/_{x}\} \bullet \sigma$. In order to proceed, we need prove that the following equality holds:

$$f\{^{f(*)}/_{u}\} \bullet \sigma \;=\; (f \bullet \sigma_{\neq u})\{^{(f \bullet \sigma_{\neq u})(*)}/_{u}\} \tag{16}$$

We have two cases:

* $u \notin \mathrm{dom}\,\sigma$. Then $\sigma_{\neq u} = \sigma$, and thus:

$$f\{^{f(*)}/_{u}\} \bullet \sigma = f\{^{f(*)}/_{u}\} \bullet \sigma_{\neq u} = (f \bullet \sigma_{\neq u})\{^{(f \bullet \sigma_{\neq u})(*)}/_{u}\}$$

* $u \in \mathrm{dom}\,\sigma$. Then, σ maps u to a fresh s. We will demonstrate eq. (16) on the current page by structural induction on the reduction $S'' \xrightarrow{\mathsf{A} : \pi, \sigma} S'''$, considering the cases in which σ has the form $\{^{s}/_{x,y}\}$ (as required by [DEL2] premises).
  - [FUSE]. We have two possible cases for the redex S''. In the simpler one, B is not involved in any way in the reduction:

$$\frac{\vdash_{\mathsf{B}} S'' = \mathsf{A}[\mathsf{fuse}.P + P' \mid Q] \mid \mathsf{A}[\downarrow_x \mathsf{A} \; says \; c \mid \downarrow_y \mathsf{C} \; says \; c'] \rhd f\{^{f(*)}/_{u}\}}{\vdash_{\mathsf{B}} S = (u)S'' \rhd f} \text{ [T-SDEL1]}$$

    and, as shown in the typing derivations A.4c and fig. A.4d on page 45, S can be typed with any f. By applying the induction hypothesis on the reduction $S'' \xrightarrow{\mathsf{A} : \mathsf{fuse}, \sigma} S'''$, we have only one possible typing for S''', shown in fig. A.4e on page 45 — where u is either x or y, depending on the delimitation in the redex. This gives us two possibilities:

    1. $u = x$. Then we have simply:

$$f\{^{f(*)}/_{x}\} \bullet \sigma = f\{^{f(*)}/_{x}\} \bullet \sigma_{\neq x} = (f \bullet \sigma_{\neq x})\{^{(f \bullet \sigma_{\neq x})(*)}/_{x}\}$$

    2. $u = y$. We can observe that the conditions on f imposed by the subject reduction hypothesis guarantee $f(x) = f(*)$, and then:

$$f\{^{f(*)}/_{y}\} \bullet \sigma = f\{^{f(*)}/_{y}\} \bullet \sigma_{\neq y} = (f \bullet \sigma_{\neq y})\{^{(f \bullet \sigma_{\neq y})(*)}/_{y}\}$$

    Hence, in this simpler case, we have verified eq. (16) on this page as:

$$f\{^{f(*)}/_{u}\} \bullet \sigma = f\{^{f(*)}/_{u}\} \bullet \sigma_{\neq u} = (f \bullet \sigma_{\neq u})\{^{(f \bullet \sigma_{\neq u})(*)}/_{u}\}$$

In the complex case, a latent contract of B is involved in the reduction:

$$\frac{\vdash_{\mathsf{B}} S'' = \mathsf{A}[\mathsf{fuse}.P + P' \mid Q] \mid \mathsf{A}[\downarrow_x \mathsf{A}\ \mathit{says}\ c \mid \downarrow_y \mathsf{B}\ \mathit{says}\ c'] \rhd f\{^{f(*)}/_u\}}{\vdash_{\mathsf{B}} S = (u)S'' \rhd f} \text{ [T-SDEL1]}$$

And thus, by applying the induction hypothesis, the reduct of S'' in the premise of [DEL2] can be typed as $\vdash_{\mathsf{B}} S''' \rhd f\{^{f(*)}/_u\} \bullet \sigma$, where $\sigma = \{^s/_{x,y}\}$. We now need to consider two cases for u:

1. $u = x$. Then, S is typed as shown in fig. A.4a on the facing page. Let $\sigma_{\neq u} = \sigma_{\neq x} = \{^s/_y\}$. We have $f\{^{f(*)}/_x\} \bullet \sigma = (f \bullet \{^s/_y\})\{^{(f \bullet \{^s/_y\})(*)}/_x\}$, since:
   (a) for all $v \notin \{x, y, s\}$, we have $(f\{^{f(*)}/_x\} \bullet \sigma)(v) = f(v) = ((f \bullet \{^s/_y\})\{^{(f \bullet \{^s/_y\})(*)}/_x\})(v)$;
   (b) $(f\{^{f(*)}/_x\} \bullet \sigma)(x) = f(*) = ((f \bullet \{^s/_y\})\{^{(f \bullet \{^s/_y\})(*)}/_x\})(x)$;
   (c) $(f\{^{f(*)}/_x\} \bullet \sigma)(y) = f(*) = ((f \bullet \{^s/_y\})\{^{(f \bullet \{^s/_y\})(*)}/_x\})(y)$ (due to the way the substitution works — see definition 48 on page 16);
   (d) $(f\{^{f(*)}/_x\} \bullet \sigma)(s) = f(y) = ((f \bullet \{^s/_y\})\{^{(f \bullet \{^s/_y\})(*)}/_x\})(s)$.
2. $u = y$. Then, S is typed as shown in fig. A.4b on the facing page. The premise "$f(*)$ realizes c'" is only true when c' is the empty contract $c' = \mathsf{e}$ — which implies that for all $z \in \mathcal{V} \cup \mathcal{N}$, $f(z) = f(*) = (f\{^{f(*)}/_u\} \bullet \sigma)(z) = \left((f \bullet \sigma_{\neq u})\{^{(f \bullet \sigma_{\neq u})(*)}/_u\}\right)(z)$ [5]. Hence, $f\{^{f(*)}/_u\} \bullet \sigma = (f \bullet \sigma_{\neq u})\{^{(f \bullet \sigma_{\neq u})(*)}/_u\}$.

· [DEL2]. Since rule [FUSE] only generates substitutions of the form $\{^s/_{x,y}\}$, there could not be two consecutive applications of rule [DEL2] — otherwise, the premise $\sigma_{\neq u} \neq \emptyset$ would not hold for the second one;

· [PAR]. We have:

$$\frac{S_1 \xrightarrow{\mathsf{A}:\pi,\sigma} S_2 \quad \operatorname{ran}\sigma \cap \operatorname{fn}(S_3) = \emptyset}{S'' = S_1 \mid S_3 \xrightarrow{\mathsf{A}:\pi,\sigma} S_2 \mid S_3\sigma = S'''} \text{ [PAR]}$$

We have the following typing derivation for S'':

$$\frac{S_1 \rhd f\{^{f(*)}/_u\} \quad S_3 \rhd f\{^{f(*)}/_u\}}{\vdash_{\mathsf{B}} S'' = S_1 \mid S_3 \rhd f\{^{f(*)}/_u\}} \text{ [T-SPAR1]}$$

By applying the induction hypothesis for the subject reduction, from $S_1 \xrightarrow{\mathsf{A}:\pi,\sigma} S_2$ we have $\vdash_{\mathsf{B}} S_2 \rhd f\{^{f(*)}/_u\} \bullet \sigma$, and thus $\vdash_{\mathsf{B}} S_2 \mid S_3\sigma = S''' \rhd f\{^{f(*)}/_u\} \bullet \sigma$. Also, by applying the induction hypothes for eq. (16) on the previous page, we have that the equality $f\{^{f(*)}/_u\} \bullet \sigma = (f \bullet \sigma_{\neq u})\{^{(f \bullet \sigma_{\neq u})(*)}/_u\}$ holds for $S_1 \xrightarrow{\mathsf{A}:\pi,\sigma} S_2$; and thus, since the types are the same, it must also hold for $S'' \xrightarrow{\mathsf{A}:\pi,\sigma} S'''$;

· [DEF]. We have:

$$\frac{X(\vec{w}) \stackrel{\text{def}}{=} P \quad \mathsf{A}[P\{^{\vec{v}}/_{\vec{w}}\} \mid Q] \mid S_0 = S_1 \xrightarrow{\mathsf{A}:\pi,\sigma} S'''}{S'' = \mathsf{A}[X(\vec{v}) \mid Q] \mid S_0 \xrightarrow{\mathsf{A}:\pi,\sigma} S'''} \text{ [DEF]}$$

We have the following typing derivation for S'':

$$\frac{\dfrac{\mathsf{A} \neq \mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[X(\vec{v}) \mid Q] \rhd f\{^{f(*)}/_u\}}\ \text{[T-SAFREE1]} \qquad \vdash_{\mathsf{B}} S_0 \rhd f}{\vdash_{\mathsf{B}} S'' \rhd f\{^{f(*)}/_u\}} \text{ [T-SPAR2]}$$

and we have the same typing for S_1:

$$\frac{\dfrac{\mathsf{A} \neq \mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[P\{^{\vec{v}}/_{\vec{w}}\} \mid Q] \rhd f\{^{f(*)}/_u\}}\ \text{[T-SAFREE1]} \qquad \vdash_{\mathsf{B}} S_0 \rhd f}{\vdash_{\mathsf{B}} S_1 \rhd f\{^{f(*)}/_u\}} \text{ [T-SPAR2]}$$

By applying the induction hypothesis for subject reduction, from $S_1 \xrightarrow{\mathsf{A}:\pi,\sigma} S'''$ we have $\vdash_{\mathsf{B}} S''' \rhd f\{^{f(*)}/_u\} \bullet \sigma$. Also, by applying the induction hypothesis for eq. (16) on the preceding page, we have

[5] This peculiar case says that, in general, a system is untypeable if a session variable delimitation includes a latent contract, but not the process which should realize it. There is only one exception, i.e. when the contract is empty (and thus always realized).

$$\dfrac{\dfrac{\dfrac{\mathsf{A}\neq\mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[\mathsf{fuse}.P+P' \mid Q]\rhd f\{^{f(*)}/_{x}\}}\ {\scriptstyle[\text{T-SAFREE1}]} \quad \dfrac{\dfrac{\mathsf{A}\neq\mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[\downarrow_x \mathsf{A}\ says\ c]\rhd f\{^{f(*)}/_{x}\}}\ {\scriptstyle[\text{T-SAFREE2}]} \quad \dfrac{(f\{^{f(*)}/_{x}\})(y)=f(y)\text{ realizes }c'}{\vdash_{\mathsf{B}} \mathsf{A}[\downarrow_y \mathsf{B}\ says\ c']\rhd f\{^{f(*)}/_{x}\}}\ {\scriptstyle[\text{T-SFZ1}]}}{\mathsf{A}[\downarrow_x \mathsf{A}\ says\ c \mid \downarrow_y \mathsf{B}\ says\ c']\rhd f\{^{f(*)}/_{x}\}}\ {\scriptstyle[\text{T-SFZ2}]}}{\vdash_{\mathsf{B}} S''=\mathsf{A}[\mathsf{fuse}.P+P' \mid Q] \mid \mathsf{A}[\downarrow_x \mathsf{A}\ says\ c \mid \downarrow_y \mathsf{B}\ says\ c']\rhd f\{^{f(*)}/_{x}\}}\ {\scriptstyle[\text{T-SPAR1}]}}{\vdash_{\mathsf{B}} (x)S''=S\rhd f}\ {\scriptstyle[\text{T-SDEL1}]}$$

(a)

$$\dfrac{\dfrac{\dfrac{\mathsf{A}\neq\mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[\mathsf{fuse}.P+P' \mid Q]\rhd f\{^{f(*)}/_{y}\}}\ {\scriptstyle[\text{T-SAFREE1}]} \quad \dfrac{\dfrac{\mathsf{A}\neq\mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[\downarrow_x \mathsf{A}\ says\ c]\rhd f\{^{f(*)}/_{y}\}}\ {\scriptstyle[\text{T-SAFREE2}]} \quad \dfrac{(f\{^{f(*)}/_{y}\})(y)=f(*)\text{ realizes }c'}{\vdash_{\mathsf{B}} \mathsf{A}[\downarrow_y \mathsf{B}\ says\ c']\rhd f\{^{f(*)}/_{y}\}}\ {\scriptstyle[\text{T-SFZ1}]}}{\mathsf{A}[\downarrow_x \mathsf{A}\ says\ c \mid \downarrow_y \mathsf{B}\ says\ c']\rhd f\{^{f(*)}/_{y}\}}\ {\scriptstyle[\text{T-SFZ2}]}}{\vdash_{\mathsf{B}} S''=\mathsf{A}[\mathsf{fuse}.P+P' \mid Q] \mid \mathsf{A}[\downarrow_x \mathsf{A}\ says\ c \mid \downarrow_y \mathsf{B}\ says\ c']\rhd f\{^{f(*)}/_{y}\}}\ {\scriptstyle[\text{T-SPAR1}]}}{\vdash_{\mathsf{B}} (y)S''=S\rhd f}\ {\scriptstyle[\text{T-SDEL1}]}$$

(b)

$$\dfrac{\dfrac{\dfrac{\mathsf{A}\neq\mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[\mathsf{fuse}.P+P' \mid Q]\rhd f\{^{f(*)}/_{x}\}}\ {\scriptstyle[\text{T-SAFREE1}]} \quad \dfrac{\dfrac{\mathsf{A}\neq\mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[\downarrow_x \mathsf{A}\ says\ c]\rhd f\{^{f(*)}/_{x}\}}\ {\scriptstyle[\text{T-SAFREE2}]} \quad \dfrac{\mathsf{C}\neq\mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[\downarrow_y \mathsf{C}\ says\ c']\rhd f\{^{f(*)}/_{x}\}}\ {\scriptstyle[\text{T-SAFREE2}]}}{\mathsf{A}[\downarrow_x \mathsf{A}\ says\ c \mid \downarrow_y \mathsf{C}\ says\ c']\rhd f\{^{f(*)}/_{x}\}}\ {\scriptstyle[\text{T-SFZ2}]}}{\vdash_{\mathsf{B}} S''=\mathsf{A}[\mathsf{fuse}.P+P' \mid Q] \mid \mathsf{A}[\downarrow_x \mathsf{A}\ says\ c \mid \downarrow_y \mathsf{C}\ says\ c']\rhd f\{^{f(*)}/_{x}\}}\ {\scriptstyle[\text{T-SPAR1}]}}{\vdash_{\mathsf{B}} (x)S''=S\rhd f}\ {\scriptstyle[\text{T-SDEL1}]}$$

(c)

$$\dfrac{\dfrac{\dfrac{\mathsf{A}\neq\mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[\mathsf{fuse}.P+P' \mid Q]\rhd f\{^{f(*)}/_{y}\}}\ {\scriptstyle[\text{T-SAFREE1}]} \quad \dfrac{\dfrac{\mathsf{A}\neq\mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[\downarrow_x \mathsf{A}\ says\ c]\rhd f\{^{f(*)}/_{y}\}}\ {\scriptstyle[\text{T-SAFREE2}]} \quad \dfrac{\mathsf{C}\neq\mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[\downarrow_y \mathsf{C}\ says\ c']\rhd f\{^{f(*)}/_{y}\}}\ {\scriptstyle[\text{T-SAFREE2}]}}{\mathsf{A}[\downarrow_x \mathsf{A}\ says\ c \mid \downarrow_y \mathsf{C}\ says\ c']\rhd f\{^{f(*)}/_{y}\}}\ {\scriptstyle[\text{T-SFZ2}]}}{\vdash_{\mathsf{B}} S''=\mathsf{A}[\mathsf{fuse}.P+P' \mid Q] \mid \mathsf{A}[\downarrow_x \mathsf{A}\ says\ c \mid \downarrow_y \mathsf{C}\ says\ c']\rhd f\{^{f(*)}/_{y}\}}\ {\scriptstyle[\text{T-SPAR1}]}}{\vdash_{\mathsf{B}} (y)S''=S\rhd f}\ {\scriptstyle[\text{T-SDEL1}]}$$

(d)

$$\dfrac{\dfrac{\mathsf{A}\neq\mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[P]\sigma\rhd f\{^{f(*)}/_{u}\}\bullet\sigma}\ {\scriptstyle[\text{T-SAFREE1}]} \quad \dfrac{(\mathsf{A}\ says\ c \mid \mathsf{C}\ says\ c')\sigma\ \mathsf{B}\text{-free}}{s[(\mathsf{A}\ says\ c \mid \mathsf{C}\ says\ c')\sigma]\rhd f\{^{f(*)}/_{u}\}\bullet\sigma}\ {\scriptstyle[\text{T-SAFREE3}]}}{\vdash_{\mathsf{B}} S'''=\mathsf{A}[P]\sigma \mid s[(\mathsf{A}\ says\ c \mid \mathsf{C}\ says\ c')\sigma]\rhd f\{^{f(*)}/_{u}\}\bullet\sigma}\ {\scriptstyle[\text{T-SPAR1}]}$$

(e)

Figure A.4: Typing derivations for subject reduction proof ([DEL2] rule, [FUSE] case).

that the equality $f\{^{f(*)}/_{u}\}\bullet\sigma=(f\bullet\sigma_{\neq u})\{^{(f\bullet\sigma_{\neq u})(*)}/_{u}\}$ hold for $S_1 \xrightarrow{\mathsf{A}:\pi,\sigma} S'''$; and thus, since the types are the same, it must also hold for $S'' \xrightarrow{\mathsf{A}:\pi,\sigma} S'''$;

Having demonstrated that eq. (16) on page 43 always holds, we can now type the reduct S' of [DEL2] on page 42 as:

$$\dfrac{\vdash_{\mathsf{B}} S'''\rhd f\{^{f(*)}/_{u}\}\bullet\sigma=(f\bullet\sigma_{\neq u})\{^{(f\bullet\sigma_{\neq u})(*)}/_{u}\}}{\vdash_{\mathsf{B}} S'=(u)S'''\rhd f\bullet\sigma_{\neq u}}\ {\scriptstyle[\text{T-SDEL1}]}$$

- [PAR]. We have:

$$\dfrac{S'' \xrightarrow{\mathsf{A}:\pi,\sigma} S''' \quad \operatorname{ran}\sigma\cap\operatorname{fn}(S'''')=\emptyset}{S=S'' \mid S'''' \xrightarrow{\mathsf{A}:\pi,\sigma} S''' \mid S''''\sigma=S'}\ {\scriptstyle[\text{PAR}]}$$

  - For item (9), we know that participant A appears with its process in S'' (since it reduces through A) and

cannot appear at the same time in S''''. The redex can only be typed as:

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

By applying the induction hypothesis on the premise, let $\vdash_{\mathsf{A}} S''' : f' \bullet \sigma$, with $f \xrightarrow{\pi} f'$. We know that, for some P', $\mathsf{A}[P']$ appears in S''', with $\vdash P' : f' \bullet \sigma$ (lemma 65 on page 23). This implies that $\mathsf{A}[P']$ cannot appear in $S''''\sigma$, and thus we have the following typing derivation for the reduct:

$$\frac{\vdash_{\mathsf{A}} S''' : f \bullet \sigma \quad \vdash_{\mathsf{A}} S''''\sigma \rhd f \bullet \sigma}{\vdash_{\mathsf{A}} S''' \mid S''''\sigma = S' : f' \bullet \sigma} \text{ [T-SPAR2]}$$

– for item (10), the redex can be typed via rule [T-SPAR2] in two ways:
  * $\vdash_{\mathsf{B}} S'' : f$ and $\vdash_{\mathsf{B}} S'''' \rhd f$. By applying the induction hypothesis on the premise, let $\vdash_{\mathsf{B}} S''' : f \bullet \sigma$. Considering that $\vdash_{\mathsf{B}} S'''' \rhd f \implies \vdash_{\mathsf{B}} S''''\sigma \rhd f \bullet \sigma$ (lemma 70 on page 24), the reduct can be typed as:

$$\frac{\vdash_{\mathsf{B}} S''' : f \bullet \sigma \quad \vdash_{\mathsf{B}} S''''\sigma \rhd f \bullet \sigma}{\vdash_{\mathsf{B}} S''' \mid S''''\sigma = S' : f \bullet \sigma} \text{ [T-SPAR2]}$$

  * $\vdash_{\mathsf{B}} S'' \rhd f$ and $\vdash_{\mathsf{B}} S'''' : f$. By applying the induction hypothesis for item (15) on the premise, let $\vdash_{\mathsf{B}} S''' \rhd f \bullet \sigma$. Considering that $\vdash_{\mathsf{B}} S'''' : f \implies \vdash_{\mathsf{B}} S''''\sigma : f \bullet \sigma$ (lemma 70 on page 24), the reduct can be typed as:

$$\frac{\vdash_{\mathsf{B}} S''' \rhd f \bullet \sigma \quad \vdash_{\mathsf{B}} S''''\sigma : f \bullet \sigma}{\vdash_{\mathsf{B}} S''' \mid S''''\sigma = S' : f \bullet \sigma} \text{ [T-SPAR2]}$$

– for item (15), we have the following typing derivation for the redex:

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

by applying the induction hypothesis on the premise, let $\vdash_{\mathsf{B}} S''' \rhd f \bullet \sigma$. Considering that $\vdash_{\mathsf{B}} S'''' \rhd f \implies \vdash_{\mathsf{B}} S''''\sigma \rhd f \bullet \sigma$ (lemma 70 on page 24), the reduct can be typed as:

$$\frac{\vdash_{\mathsf{B}} S''' \rhd f \bullet \sigma \quad \vdash_{\mathsf{B}} S''''\sigma \rhd f \bullet \sigma}{\vdash_{\mathsf{B}} S''' \mid S''''\sigma = S' \rhd f \bullet \sigma} \text{ [T-SPAR1]}$$

• [DEF]. We have:

$$\frac{X(\vec{u}) \stackrel{\text{def}}{=} P \quad \mathsf{A}[P\{\vec{v}/\vec{u}\} \mid Q] \mid S_0 = S'' \xrightarrow{\mathsf{A} : \pi, \sigma} S'}{S = \mathsf{A}[X(\vec{v}) \mid Q] \mid S_0 \xrightarrow{\mathsf{A} : \pi, \sigma} S'} \text{ [DEF]}$$

– For item (9), the redex can only be typed via rule [T-SPAR2]. Let us consider the typing derivation of S:

$$\dfrac{\dfrac{\dfrac{\dfrac{\emptyset \vdash X(\vec{v}) : f^X \quad \emptyset \vdash Q : f^Q}{\emptyset \vdash X(\vec{v}) \mid Q : \lambda u . f^X(u) \mid f^Q(u) = f} \text{ [T-PAR]}}{\vdash_{\mathsf{A}} \mathsf{A}[X(\vec{v}) \mid Q] : f} \text{ [T-SA]} \qquad \vdash_{\mathsf{A}} S_0 \rhd f}{\vdash_{\mathsf{A}} S : f} \text{ [T-SPAR2]}$$

Note that the typing relations in the premises of rule [T-SPAR2] cannot be swapped, considering lemma 65 on page 23 and the fact that participant A cannot appear with her process in S_0, too. By lemma 77 on page 26, since $\emptyset \vdash X(\vec{v}) : f^X$ then $\emptyset \vdash P\{\vec{v}/\vec{u}\} : f^X$. We then have the following typing derivation for S'':

$$\dfrac{\dfrac{\dfrac{\dfrac{\emptyset \vdash P\{\vec{v}/\vec{u}\} : f^X \quad \emptyset \vdash Q : f^Q}{\emptyset \vdash P\{\vec{v}/\vec{u}\} \mid Q : \lambda w . f^X(w) \mid f^Q(w) = f} \text{ [T-PAR]}}{\vdash_{\mathsf{A}} \mathsf{A}[P\{\vec{v}/\vec{u}\} \mid Q] : f} \text{ [T-SA]} \qquad \vdash_{\mathsf{A}} S_0 \rhd f}{\vdash_{\mathsf{A}} S'' : f} \text{ [T-SPAR2]}$$

We can now apply the induction hypothesis on the premise $S'' \xrightarrow{\mathsf{A} : \pi, \sigma} S'$, obtaining an f' such that $f \xrightarrow{\pi} f'$ and $\vdash_{\mathsf{A}} S' : f' \bullet \sigma$.

– for item (10), we have the following typing derivation for S:

$$\dfrac{\dfrac{\mathsf{A} \neq \mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[X(\vec{v}) \mid Q] \rhd f}\ {\scriptstyle [\text{T-SAFREE1}]} \qquad \vdash_{\mathsf{B}} S_0 \colon f}{\vdash_{\mathsf{B}} S \colon f}\ {\scriptstyle [\text{T-SPAR2}]}$$

We can then reconstruct the following typing derivation for S'':

$$\dfrac{\dfrac{\mathsf{A} \neq \mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[P\{\vec{v}/\vec{u}\} \mid Q] \rhd f}\ {\scriptstyle [\text{T-SAFREE1}]} \qquad \vdash_{\mathsf{B}} S_0 \colon f}{\vdash_{\mathsf{B}} S'' \colon f}\ {\scriptstyle [\text{T-SPAR2}]}$$

Since $S'' \xrightarrow{\mathsf{A}\,:\,\pi,\sigma} S'$ and $\vdash_{\mathsf{B}} S'' \colon f$, by applying the induction hypothesis of item (10), we obtain the thesis $\vdash_{\mathsf{B}} S' \colon f \bullet \sigma$.

– for item (15), we have the following typing derivation for S:

$$\dfrac{\dfrac{\mathsf{A} \neq \mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[X(\vec{v}) \mid Q] \rhd f}\ {\scriptstyle [\text{T-SAFREE1}]} \qquad \vdash_{\mathsf{B}} S_0 \rhd f}{\vdash_{\mathsf{B}} S \rhd f}\ {\scriptstyle [\text{T-SPAR1}]}$$

We can then reconstruct the following typing derivation for S'':

$$\dfrac{\dfrac{\mathsf{A} \neq \mathsf{B}}{\vdash_{\mathsf{B}} \mathsf{A}[P\{\vec{v}/\vec{u}\} \mid Q] \rhd f}\ {\scriptstyle [\text{T-SAFREE1}]} \qquad \vdash_{\mathsf{B}} S_0 \rhd f}{\vdash_{\mathsf{B}} S'' \rhd f}\ {\scriptstyle [\text{T-SPAR1}]}$$

By applying the induction hypothesis of item (15), we obtain the thesis $\vdash_{\mathsf{B}} S' \rhd f \bullet \sigma$.

□

Lemma 78 (Process progress)**.** *For all processes P in a system S, if $\vdash P \colon f$ and $f \xrightarrow{\pi} f'$, then:*

1. *if $\pi = \tau$ or $\pi = \mathsf{tell}_{\mathsf{B}} \downarrow_w c$, or $\pi = \mathsf{ask}_s\, \phi$ and $S \vdash_s \phi$, then $\exists P', S_0, S_1 \,.\, S = \mathsf{A}[P] \mid S_0 \xrightarrow{\mathsf{A}\,:\,\pi,\emptyset} \mathsf{A}[P'] \mid S_1 \ \wedge\ \vdash P' \colon f'$*
2. *if $\pi = \mathsf{do}_u\, \mathsf{a}$, then $\mathsf{a} \in RD_u^{\mathsf{A}}(\mathsf{A}[P])$.*

Proof. For item item 1 on the current page, we have two cases:

- $\pi = \tau$. Then, $f \xrightarrow{\tau} f'$ implies that, for all $u \in \mathcal{N} \cup \mathcal{V}$, the semantics derivation of the channel type must be of the form:

$$\dfrac{\dfrac{\dfrac{}{[\tau]_u \,.\, T_u^2 \xrightarrow{[\tau_u]} T_u^2}\ {\scriptstyle [\text{E-ALPHA}]}}{[\tau]_u \,.\, T_u^2 + T_u^3 \xrightarrow{[\tau]_u} T_u^2}\ {\scriptstyle [\text{E-SUML}]}}{f(u) = [\tau]_u \,.\, T_u^2 + T_u^3 \mid T_u^4 \xrightarrow{[\tau]_u} T_u^2 \mid T_u^4 = f'(u)}\ {\scriptstyle [\text{E-PARL}]}$$

Since f is inhabited by hypothesis, its components must be other inhabited process types, fitting some derivation according to the rules in fig. 5.4 on page 13. Thus, let us take f_2, f_3, f_4 such that $f_2(u) = T_u^2$, $f_3(u) = T_u^3$ and $f_4(u) = T_u^4$, and thus $f(u) = [\tau]_u \,.\, f_2(u) + f_3(u) \mid f_4(u)$ and $f'(u) = f_2(u) \mid f_4(u)$. This, in turn, implies that the typing derivation of P must be of the form:

$$\dfrac{\dfrac{\vdash P_2 \colon f_2 \quad \vdash P_3 \colon f_3}{\vdash \tau \,.\, P_2 + P_3 \colon \lambda v \,.\, [\tau]_v \,.\, f_2(v) + f_3(v)}\ {\scriptstyle [\text{T-SUM}]} \qquad \vdash P_4 \colon f_4}{\vdash \tau \,.\, P_2 + P_3 \mid P_4 \colon \lambda v \,.\, [\tau]_v \,.\, f_2(v) + f_3(v) \mid f_4(v)}\ {\scriptstyle [\text{T-PAR}]}$$

which implies that P must be of the form $P = \tau \,.\, P_2 + P_3 \mid P_4$. We also have the following typing derivation for the reduct P':

$$\dfrac{\vdash P_2 \colon f_2 \quad \vdash P_4 \colon f_4}{\vdash P_2 \mid P_4 \colon \lambda v \,.\, f_2(v) \mid f_4(v)}\ {\scriptstyle [\text{T-PAR}]}$$

where the premises are the same of the typing derivation of the redex P. This derivation implies that P' must be of the form $P' = P_2 \mid P_4$.
We conclude by letting $\sigma = \emptyset$, $P' = P_2 \mid P_4$ and $S_0 = S_1$, thus obtaining (by rule [TAU] of CO_2 semantics):

$$\mathsf{A}[\tau \,.\, P_2 + P_3 \mid P_4] \;\mid\; S_0 \xrightarrow{\mathsf{A}:\tau,\emptyset} \mathsf{A}[P_2 \mid P_4] \;\mid\; S_0$$

- $\pi = \mathsf{tell}_\mathsf{B} \downarrow_w c$. The proof is similar, *mutatis mutandis*, to the $\pi = \tau$ case above. We conclude by letting $\sigma = \emptyset$, $P' = P_2 \mid P_4$ and $S_1 = S_0 \mid \mathsf{B}[\downarrow_w \mathsf{A}\ \mathit{says}\ c]$, thus obtaining (by rule [TELL] of CO_2 semantics):

$$\mathsf{A}[\mathsf{tell}_\mathsf{B} \downarrow_w c \,.\, P_2 + P_3 \mid P_4] \;\mid\; S_0 \xrightarrow{\mathsf{A}:\mathsf{tell}_\mathsf{B} \downarrow_w c,\emptyset} \mathsf{A}[P_2 \mid P_4] \;\mid\; S_0 \mid \mathsf{B}[\downarrow_w \mathsf{A}\ \mathit{says}\ c]$$

- $\pi = \mathsf{ask}_s\, \phi$ and $S \vdash_s \phi$. The proof is similar, *mutatis mutandis*, to the $\pi = \tau$ case above. Let $\sigma = \emptyset$, $P' = P_2 \mid P_4$ and $S_1 = S_0 = s[\gamma] \mid S_{01}$ such that $\gamma \vdash \phi$ (which satisfies the hypothesis $S \vdash_s \phi$). We obtain, (by rule [ASK] of CO_2 semantics):

$$\frac{\gamma \vdash \phi}{\mathsf{A}[\mathsf{ask}_s\, \phi \,.\, P_2 + P_3 \mid P_4] \;\mid\; s[\gamma] \mid S_{01} \xrightarrow{\mathsf{A}:\mathsf{ask}_s\, \phi,\emptyset} \mathsf{A}[P_2 \mid P_4] \;\mid\; s[\gamma] \mid S_{01}}\ [\text{ASK}]$$

For item item 2 on the preceding page, the proof is similar, *mutatis mutandis*, to the cases above. Thus, we have that P must be of the form $P = \mathsf{do}_u\, \mathsf{a} \,.\, P_2 + P_3 \mid P_4$. This implies that $\mathsf{a} \in RD_u^\mathsf{A}(\mathsf{A}[P])$ (see definition 7 on page 7). □

A.12 Proof of Lemma 50 on page 16

Proof. By lemma 67 on page 23, $\vdash_\mathsf{A} S\colon f$ implies that S must be of the form $S = (\vec{v})\,(\mathsf{A}[P] \mid S_0)$ such that $\vdash P\colon f^P \sqsupseteq f$. Focusing on the inner part of the delimitation in S (i.e., $\mathsf{A}[P] \mid S_0$), have that:

1. if $\pi = \tau$, or $\pi = \mathsf{tell}_\mathsf{B} \downarrow_w c$, or $\pi = \mathsf{ask}_s\, \phi$ and $S \vdash_s \phi$, we observe that the hypothesis $f \xrightarrow{\pi} f'$ implies $f^P \xrightarrow{\pi} f^{P'}$, with $f' \sqsubseteq f^{P'}$; furthermore, by lemma 78 on the preceding page, $f^P \xrightarrow{\pi} f^{P'}$ implies that there exist P', S_0, S_1 such that $\mathsf{A}[P] \mid S_0 \xrightarrow{A:\pi,\emptyset} \mathsf{A}[P'] \mid S_1$ and $\vdash P'\colon f^{P'}$. Thus, we can reuse the cases in the proof of 78 in order to infer the mandatory forms of P and P' (and their types) depending on π, and then the value for S' that verify the theorem thesis;
2. if $\pi = \mathsf{do}_u\, \mathsf{a}$, we observe that $f(u)$ moves as $f(u) \xrightarrow{\mathsf{a}}$: since its prefix is not $\tau_?$, $f(u) \neq f(*)$, and thus, by lemma 58 on page 20, $u \in \mathrm{fnv}(S)$; furthermore, by applying lemma 78 on the preceding page to f^P and $\mathsf{A}[P] \mid S_0$, we have that $\mathsf{a} \in RD_u^\mathsf{A}(\mathsf{A}[P]) = RD_u^\mathsf{A}(\mathsf{A}[P] \mid S_0)$, where P has the form $P = \mathsf{do}_u\, \mathsf{a} \,.\, P_2 + P_3 \mid P_4$. Since $u \in \mathrm{fnv}(S)$ implies $u \notin \vec{v}$, we also have $\mathsf{a} \in RD_u^\mathsf{A}(\mathsf{A}[P] \mid S_0) = RD_u^\mathsf{A}((\vec{v})\,(\mathsf{A}[P] \mid S_0)) = RD_u^\mathsf{A}(S)$.

□

A.13 Proof of Theorem 51 on page 16

Proof. $f(s) \stackrel{\mathsf{a}}{\Longrightarrow}_c^\mathsf{A}$ means that, according to the rules in fig. 5.1 on page 9, $f(s)$ can move as $f(s) \xrightarrow{\tau}$ for 0 or more times; then, there exists an effect trace $\mathcal{T}$ and a channel type T_1 such that $\mathcal{T} = f(s) \xrightarrow{\tau} T' \xrightarrow{\tau} \ldots \xrightarrow{\tau} T_1 \xrightarrow{\mathsf{a}} T_1'$.

By lemma 40 on page 14, $f(u) \xrightarrow{\tau} T'$ implies that there exists f' such that $f'(u) = T'$ and $f \xrightarrow{\tau} f'$. According to lemma 50 on page 16 (system progress) $\vdash_\mathsf{A} S\colon f$ and $f \xrightarrow{\tau} f'$ imply that S is able to reduce through $S \xrightarrow{\mathsf{A}:\tau,\emptyset} S'$, with $\vdash_\mathsf{A} S'\colon f'$. By applying the same reasoning to S' and f' and their reducts, we can make both the system and its type reduce together through all the τ-moves (if any) required by $\mathcal{T}$, until a state S_1 is reached, such that $\vdash_\mathsf{A} S_1\colon f_1$, and $f_1(u) = T_1 \xrightarrow{\mathsf{a}} T_1'$.

When this happens, by lemma 40 on page 14 we have that there exists f_1' such that $f_1'(u) = T_1'$ and $f_1 \xrightarrow{\mathsf{do}_u\,\mathsf{a}} f_1'$; and then, by lemma 50 on page 16, we have that $\mathsf{a} \in RD_u^\mathsf{A}(S_1)$.

Thus, we have shown that, from the theorem hypothesis, there exists a (possibly empty) sequence of reductions $S \xrightarrow{\neq(\mathsf{A}:\,\mathsf{do}_u)}{}^* S_1$, where each move is a τ-move, and where $\mathsf{a} \in RD_u^\mathsf{A}(S_1)$ — which is precisely the definition of $\mathsf{a} \in WRD_u^\mathsf{A}(S)$ (definition 9 on page 7). □

A.14 Proof of Theorem 52 on page 16

Proof. By definition 13 on page 8, we need to prove that for all A-free S, and for all S' such that $\mathsf{A}[P] \mid S \to^* S'$, A is ready in S' (definition 12 on page 8). The fact that P is closed implies that $P \equiv (u_1,\ldots,u_n)P_0$, where $\{u_1,\ldots,u_n\} = \mathrm{fnv}(P_0)$. We observe that $\vdash_{\mathsf{A}} \mathsf{A}[P] \colon f$, and $\mathsf{A}[P] \equiv \mathsf{A}[(u_1,\ldots,u_n)P_0] \equiv (u_1,\ldots,u_n)\mathsf{A}[P_0]$. We also have that, for any f_0 such that $\vdash P_0 \colon f_0$, $f = f_0\{{}^{f_0(*)}/_{u_1,\ldots,u_n}\}$ (by typing rule [T-DEL]), $\vdash_{\mathsf{A}} \mathsf{A}[P_0] \colon f_0$, and $f \sqsubseteq f_0$ (lemma 37 on page 13); furthermore, in the typing derivation of P (or $\mathsf{A}[P]$) each $u \in \mathrm{fnv}(P_0)$ is involved by an instance of [T-DEL] (or [T-SDEL2] in fig. 5.5 on page 15) — which mandates $f_0(u)$ to be honest, and thus f_0 to be honest . This allows to apply subject reduction (theorem 49 on page 16) to reduce $\mathsf{A}[P_0]$. We will demonstrate that $\mathsf{A}[P_0]$ is honest, which directly implies that $\mathsf{A}[P]$ is honest as well (by lemma 42 on page 14).

Let us take any A-free system S^+, and consider $S_0 \equiv \mathsf{A}[P_0] \mid S^+$. We have that $\vdash_{\mathsf{A}} S_0 \colon f_0$. We observe that, if S_0 reduces without advertising any contract, A is always vacuously ready, and thus vacuously honest. However, S_0 may eventually reduce into a state $S_0' \equiv \mathsf{A}[P_0'] \mid s[\mathsf{A}\ \mathit{says}\ c \mid \gamma] \mid S_0^+$ such that S_0^+ is A-free and $\vdash_{\mathsf{A}} S_0' \colon f_0'$. In such state, an advertised contract c is fused into a session s through a substitution σ. And since f_0 is honest and (by subject reduction) $f_0 \to^* f_0^{\frown}$ such that $f_0' = f_0^{\frown} \bullet \sigma$, by definition 25 on page 11 we have that $f_0'(s)$ must be abstractly ready for c — i.e., by definition 24 on page 11, each possible reduction of $f_0'(s)$ generates a trace $\mathcal{T}$ of weak transitions which satisfy the ready sets exposed along the evolutions of c. Thus, for some atomic actions $\mathsf{a}, \mathsf{a}', \mathsf{a}'', \ldots$, the evolution of c, the shape of $\mathcal{T}$ and the corresponding abstract process trace $\mathcal{T}_c$ will match the following diagram:

$$\begin{array}{rcccccccc} & & c & \overset{\mathsf{a}}{\twoheadrightarrow} & c' & \overset{\mathsf{a}'}{\twoheadrightarrow} & c'' & \overset{\mathsf{a}''}{\twoheadrightarrow} & \ldots \\ \mathcal{T} & = & f_0'(s) & \overset{\mathsf{a}}{\Longrightarrow} & T_1 & \overset{\mathsf{a}'}{\Longrightarrow} & T_2 & \overset{\mathsf{a}''}{\Longrightarrow} & \ldots \\ \mathcal{T}_c & = & (c, f_0'(s)) & \to^* & (c', T_1) & \to^* & (c'', T_2) & \to^* & \ldots \end{array}$$

Let us consider the first weak transition (i.e., $f_0'(s) \overset{\mathsf{a}}{\Longrightarrow} T_1$): by theorem 51 on page 16 (progress), we have $\mathsf{a} \in \mathit{WRD}_s^{\mathsf{A}}(S_0')$; then, we can use system progress (lemma 50 on page 16) together with lemma 40 on page 14 to make S_0' perform the initial τ and nonblocking τ_ϕ actions (if any), reaching a state S_0'' such that $\vdash_{\mathsf{A}} S_0'' \colon f_0''$ and $S_0'' \xrightarrow{\mathsf{A} : \mathsf{do}_s\, \mathsf{a}, \emptyset} S_1$; at this point, subject reduction implies that exists f_1 such that $\vdash_{\mathsf{A}} S_1 \colon f_1$ and $f_0'' \xrightarrow{\mathsf{do}_s\, \mathsf{a}} f_1$ (and thus, $f_0''(s) \xrightarrow{\mathsf{a}} f_1(s) = T_1$, where T_1 realizes c'). This reasoning can be iterated to follow the rest of the weak traces and the evolutions of S_1. In conclusion, we have shown that for all A-free S^+ such that $\mathsf{A}[P_0] \mid S^+ \to^* S_0'$, A is ready in S_0', and thus $\mathsf{A}[P_0]$ is honest — i.e., $\mathsf{A}[P] \equiv \mathsf{A}[(u_1,\ldots,u_n)P_0]$ is honest. □

Theorem 79 (Type safety on systems)**.** *For all closed systems S, if $\vdash_{\mathsf{A}} S \colon f$, then* A *is honest in S.*

Proof. S closed and $\vdash_{\mathsf{A}} S \colon f$ imply $S \equiv (u_1,\ldots,u_n)S_0$, where $S_0 \equiv \mathsf{A}[P_0] \mid S^+$, $\{u_1,\ldots,u_n\} = \mathrm{fnv}(S_0)$. Let $\vdash_{\mathsf{A}} S_0 \colon f_0$: the only possible typing derivation of S_0 (via rule [T-SDEL2]) mandates $\vdash_{\mathsf{A}} \mathsf{A}[P_0] \colon f_0$ (and thus, $\vdash P_0 \colon f_0$) and $\vdash_{\mathsf{A}} S^+ \rhd f_0$, with $f \sqsubseteq f_0$ (by 45). We will demonstrate that $\mathsf{A}[P_0]$ is honest for any S^+, which directly implies that A is honest in S (by 42). The proof is the same we have used for assessing the type safety on processes (52). We only need to add one observation: if $S_0 \equiv \mathsf{A}[P_0] \mid s[\mathsf{A}\ \mathit{says}\ c \mid \gamma] \mid S_0^+$, then $\vdash_{\mathsf{A}} S_0 \colon f_0$ must have been obtained through a typing derivation including an instance of rule [T-SFUSED] (fig. 5.5 on page 15), which mandates $f_0(s)$ to realize c. □

References

[1] M. Bartoletti, E. Tuosto, and R. Zunino. Contracts in distributed systems. In *ICE*, 2011.

[2] M. Bartoletti, E. Tuosto, and R. Zunino. Contract-oriented computing in CO_2. *Scientific Annals in Computer Science*, 22(1):5–60, 2012.

[3] M. Bartoletti, E. Tuosto, and R. Zunino. On the realizability of contracts in dishonest systems. In *Proc. COORDINATION*, 2012.

[4] L. Bocchi, K. Honda, E. Tuosto, and N. Yoshida. A theory of design-by-contract for distributed multiparty interactions. In *CONCUR*, 2010.

[5] M. Bravetti and G. Zavattaro. Towards a unifying theory for choreography conformance and contract compliance. In *Software Composition*, 2007.

[6] M. Bravetti and G. Zavattaro. Contract-based discovery and composition of web services. In *Formal Methods for Web Services*. 2009.

[7] M. Buscemi, M. Coppo, M. Dezani-Ciancaglini, and U. Montanari. Constraints for service contracts. In *Proc. TGC*, 2011.

[8] M. G. Buscemi and U. Montanari. CC-Pi: A constraint-based language for specifying service level agreements. In *ESOP*, 2007.

[9] S. Carpineti, G. Castagna, C. Laneve, and L. Padovani. A formal account of contracts for web services. In *WS-FM*, 2006.

[10] G. Castagna, N. Gesbert, and L. Padovani. A theory of contracts for web services. *ACM TOPLAS*, 31(5), 2009.

[11] T.-C. Chen, L. Bocchi, P.-M. Denilou, K. Honda, and N. Yoshida. Asynchronous distributed monitoring for multiparty session enforcement. In *Proc. TGC*. 2012.

[12] T.-C. Chen and K. Honda. Specifying Stateful Asynchronous Properties for Distributed Programs. In *Proc. CONCUR*, 2012.

[13] M. Coppo, M. Dezani-Ciancaglini, and N. Yoshida. Global Progress for Dynamically Interleaved Multiparty Sessions. *MSCS*, 2012.

[14] K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL*, 2008.

[15] R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems*. PhD thesis, Technische Universität München, 1998.

List of Figures

List of Definitions, Theorems, Examples