HONESTY BY TYPING

MASSIMO BARTOLETTI, ALCESTE SCALAS, EMILIO TUOSTO, AND ROBERTO ZUNINO

Università degli Studi di Cagliari, Italy
e-mail address: bart@unica.it

Università degli Studi di Cagliari, Italy and Imperial College London, UK
e-mail address: alceste.scalas@imperial.ac.uk

University of Leicester, UK
e-mail address: emilio@le.ac.uk

Università degli Studi di Trento, Italy
e-mail address: roberto.zunino@unitn.it

Abstract. We propose a type system for a calculus of contracting processes. Processes can establish *sessions* by stipulating *contracts*, and then can interact either by keeping the promises made, or not. Type safety guarantees that a typeable process is *honest* — that is, it abides by the contracts it has stipulated in all possible contexts, even in presence of dishonest adversaries. Type inference is decidable, and it allows to safely approximate the honesty of processes using either *synchronous* or *asynchronous* communication.

1. Introduction

It is commonplace that distributed applications are not easy to develop. Besides the intrinsic issues due e.g. to physical distribution, and to the fragility of communication networks, 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 access resources or other components with limited availability. This dichotomy is well witnessed by the *service-oriented* paradigm, which fosters *dynamic* composition of distributed applications, with the interaction of components, a.k.a. "services", deployed by different vendors.

Cooperation and competition hardly coexist harmoniously. A frequent simplification is to neglect competition, by assuming that *all* components (including third-party ones) always adhere to some declared specification. For instance, such a specification could be a *behavioural type*, abstracting the input/output behaviour of a component — and the simplifying assumption is that each component is verified against its declared specification,

1998 ACM Subject Classification: [**Theory of computation**]: Models of computation; Concurrency – Semantics and reasoning – Program reasoning – Program specifications – Program verification; Semantics and reasoning – Program semantics – Operational semantics.

Key words and phrases: contract-oriented computing, verification, session types.

Full version of an Extended Abstract presented at FORTE'13.

LOGICAL METHODS
IN COMPUTER SCIENCE
DOI:10.2168/LMCS-???

and has a corresponding runtime behaviour. We argue that this assumption is hardly realistic in scenarios where third-party components can be used but not inspected, and thus are not guaranteed to honour their declared behaviour. This is particularly relevant in competitive scenarios, which may incentivise "selfish" components that diverge from their declared specification.

Contract-oriented computing [14, 12] addresses these concerns by disciplining the interaction among components through *contracts*, which formalise promised runtime behaviours. In a contract-oriented setting, the components of a distributed application are implemented as *processes* interacting through a *contract-oriented middleware*. Processes may *advertise* contracts; if the middleware finds a set of *compliant* contracts, it establishes a new *session* involving the advertising processes. When involved in a session, a process should perform the interactions needed to realise its contract; however, a process could also *violate* its contract, e.g. by not performing some promised input/output action: this may happen either maliciously, or unintentionally, because of some implementation bug. To discourage contract violations, the middleware can monitor the sessions, and sanction the processes which do not respect their contracts. These sanctions can be e.g. pecuniary compensations, or marginalisation: if a process misbehaves, the middleware could decrease its reputation, and consequently its chances of being involved in further sessions [6, 41].

In this scenario, to avoid sanctions, a process must be able to respect *all* the contracts it advertises, in *all* possible contexts — even those populated by adversaries which try to make it sanctioned. We call this property *honesty*. A crucial problem is then how to guarantee that a process is honest.

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

The store advertises the following contract to potential buyers B:

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

We can formalise such contract in several process algebras. For instance, we can use the following session type [32] (without channel passing):

$$\begin{aligned} C_{\mathsf{B}} &= \mathtt{buyA?}\,.\,\mathtt{pay1E?}\,.\,\mathtt{shipA!} \;\&\\ &\qquad \mathtt{buyB?}\,.\,(\mathtt{quote1E!}\,.\,\mathtt{pay1E?}\,.\,\mathtt{shipB!} \oplus \mathtt{quote3E!}\,.\,C'_{\mathsf{B}} \oplus \mathtt{abort!}) \\ C'_{\mathsf{B}} &= \mathtt{pay3E?}\,.\,(\mathtt{shipB!} \oplus \mathtt{refund!}) \,\&\, \mathtt{quit?} \end{aligned}$$

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

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

$$C_{\mathsf{D}} = \mathtt{buyB!}\,.\,(\mathtt{pay2E!}\,.\,\mathtt{shipB?} \oplus \mathtt{quit!})$$

Note that the contracts above do not specify the *actual* behaviour of the store, but only the behaviour it promises towards the buyer and the distributor. A possible informal description of the actual behaviour of the store S is the following (see Example 3.6 for a formal specification):

(1) S advertises the contract C_{B};
(2) when C_{B} is stipulated, the buyer chooses apples (buyA) or bananas (buyB);
(3) if the buyer chooses apples, S gets the payment (pay1E), and ships the item (shipA);
(4) otherwise, if the buyer chooses bananas, S checks if the item is in stock;
(5) if bananas are in stock, S provides the buyer with the quotation of €1 (quote1E), receives the payment (pay1E), and ships the item (shipB);
(6) otherwise, if bananas are not in stock, S advertises the contract C_{D};
(7) when C_{D} is stipulated, S pre-orders bananas from the distributor (buyB);
(8) S sends a €3 quotation to the buyer (quote3E) and waits for the buyer's reply;
(9) if the buyer pays €3 (pay3E), then S pays the distributor (pay2E), receives the item from the distributor (shipB), and ships it to the buyer (shipB).

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

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

Therefore, we would classify the store process above as *dishonest* (we will formalise an honest variant of the store later on, in Example 7.7). In practice, this implies that a concrete implementation of such a store could be easily attacked. For instance, an attacker could simply order bananas (when not in stock), but always cancel the transaction. The middleware will detect that the store is violating the contract with the distributor, and consequently it will sanction the store. Concretely, in the middleware of [6] the attacker will manage to never be sanctioned, and to arbitrarily decrease the store reputation, so preventing the store from being able to establish new sessions with buyers.

The example above shows that writing honest processes is an error-prone task: this is because one has to foresee all the possible points of failure of each partner.

Contributions. We address the problem of honesty in CO_2 [12], a core calculus for contract-oriented computing. 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 [15].

To obtain this result, we first study a theory of compliance between session types, both with a synchronous and an asynchronous semantics. We prove that asynchronous session types are Turing-powerful (Theorem 2.5). In Theorem 2.8 we show that asynchronous compliance is undecidable, while the synchronous one is decidable.

We introduce a type system for CO_2 processes, which associates behavioural types (based on Basic Parallel Processes) to each session name and variable. For these types we define a notion of abstract honesty, which we prove decidable (Theorem 6.6). Exploiting this result, we show that our type system has a decidable type inference (Theorem 8.6).

We establish subject reduction, i.e. types simulate processes (Theorem 9.6), and a strong form of progress which ensures that processes simulate types (Theorem 9.7). We then exploit these results to prove type safety, which guarantees that typeable processes are honest (Theorem 9.9). Further, using Theorem 4.9, we can lift our type safety result to the case of asynchronous CO_2 processes.

Together with the decidability of type inference, we obtain an algorithm to safely approximate the honesty of CO_2 processes, both synchronous and asynchronous. Our algorithm is more precise than the model-checking technique in [9], which can only establish the honesty of essentially finite-state processes (i.e., without parallel nor delimitation under recursion). For instance, Example 7.8 shows a process for which the analysis technique proposed in this paper is more precise than the one in [9].

2. Session types as contracts

We use first-order binary session types [3] as contracts. These are terms of a process algebra featuring internal/external choice, and recursion. We consider two semantics: an asynchronous one, using unbounded queues (Definition 2.2), and a synchronous one (Definition 2.3). We then study a *compliance* relation between session types, which is common for both semantics: roughly, the compliance relation formalizes when two contracts do not yield communication errors. In particular, we show that compliance between session types is not decidable (Theorem 2.8) under the asynchronous semantics, while it is decidable under the synchronous one. We then show that synchronous compliance is a sound approximation of the asynchronous one (Theorem 2.9).

2.1. **Syntax.** We assume a set of *participants* (ranged over by $\mathsf{A}, \mathsf{B}, \ldots$), and a set of *actions*, partitioned into *input actions* $\mathtt{a?}, \mathtt{b?}, \ldots \in \mathtt{A}^{?}$, and *output actions* $\mathtt{a!}, \mathtt{b!}, \ldots \in \mathtt{A}^{!}$. We let $a, b, \ldots$ range over $\mathtt{A}^{?} \cup \mathtt{A}^{!}$.

Definition 2.1 (Session types)**.** Session types are terms of the following grammar:

$$C, D \quad ::= \quad \bigoplus_{i \in I} \mathtt{a}_i\mathtt{!} \,.\, C_i \quad \mid \quad \&_{i \in I} \mathtt{a}_i\mathtt{?} \,.\, C_i \quad \mid \quad \mathtt{rec}\, X \,.\, C \quad \mid \quad X$$

where (i) the index set I is finite, (ii) in a summation, $\mathtt{a}_i = \mathtt{a}_j$ implies $i = j$, and (iii) recursion variables X are prefix-guarded.

An internal sum $\bigoplus_i \mathtt{a}_i\mathtt{!} \,.\, C_i$ allows a participant to choose one of the labels $\mathtt{a}_i$, and then to behave according to the branch C_i. Dually, an external sum $\&_i \mathtt{a}_i\mathtt{?} \,.\, C_i$ allows to wait for the other participant to choose one of the labels $\mathtt{a}_i$, and then behave according to the branch C_i. Empty internal/external sums are identified, and they are denoted with $\mathtt{1}$, which represents a *success state* wherein the interaction has terminated correctly.

We use the binary operators to isolate a branch in a sum: e.g., $C = (\mathtt{a!} \,.\, C') \oplus C''$ means that C has the form $\bigoplus_{i \in I} \mathtt{a}_i\mathtt{!} \,.\, C_i$ and there exist some $i \in I$ such that $\mathtt{a!} \,.\, C' = \mathtt{a}_i\mathtt{!} \,.\, C_i$. Hereafter, we will omit the trailing occurrences of $\mathtt{1}$, and we will only consider session types without free occurrences of recursion variables X.

$$\mathsf{A} : (\mathtt{a!} \,.\, C \oplus C') \; [\beta_{\mathsf{A}}] \mid \mathsf{B} : D \; [\beta_{\mathsf{B}}] \xrightarrow{\mathsf{A}:\mathtt{a!}}\!\!\!\!\rightarrow_{\infty} \mathsf{A} : C \; [\beta_{\mathsf{A}}] \mid \mathsf{B} : D \; [\beta_{\mathsf{B}}\; \mathtt{a!}] \quad [\textsc{Out}]$$

$$\mathsf{A} : C \; [\beta_{\mathsf{A}}] \mid \mathsf{B} : (\mathtt{a?} \,.\, D \,\&\, D') \; [\mathtt{a!}\; \beta_{\mathsf{B}}] \xrightarrow{\mathsf{B}:\mathtt{a?}}\!\!\!\!\rightarrow_{\infty} \mathsf{A} : C \; [\beta_{\mathsf{A}}] \mid \mathsf{B} : D \; [\beta_{\mathsf{B}}] \quad [\textsc{In}]$$

FIGURE 1. Asynchronous semantics of session types (symmetric rules omitted).

2.2. **Semantics.** While a session type models the intended behaviour of *one* of the two participants involved in a session, the interaction of *two* participants A and B (say, with session types C and D, respectively) is modelled by *configurations* $\gamma, \gamma', \ldots$ of the form $\mathsf{A} : C \; [\beta_{\mathsf{A}}] \mid \mathsf{B} : D \; [\beta_{\mathsf{B}}]$, where β_{A} (resp. β_{B}) is an unbounded queue that stores the messages sent by B (resp. by A) and not read yet.

Definition 2.2 (Asynchronous semantics of session types)**.** A *contract configuration* is a term of the form $\mathsf{A} : C \; [\beta_{\mathsf{A}}] \mid \mathsf{B} : D \; [\beta_{\mathsf{B}}]$, where $\mathsf{A} \neq \mathsf{B}$ and $\beta_{\mathsf{A}}, \beta_{\mathsf{B}} \in (\mathtt{A}^!)^*$. We define the relation $\equiv$ between session types as the least equivalence including α-conversion of recursion variables and unfolding of recursion (that is, $\mathtt{rec}\, X \,.\, C \equiv C\{^{\mathtt{rec}\, X.C}/_X\}$, as in the usual equi-recursive approach). The labelled transition relation $\twoheadrightarrow_{\infty}$ is the smallest relation between contract configurations induced by the rules in Figure 1 up to $\equiv$.

In rule [Out], participant A chooses the branch $\mathtt{a!}$ in an internal sum, and sends $\mathtt{a!}$ to the other participants' queue. The other participant B can read the message from the queue through rule [In], and then proceed as in the corresponding branch in its external choice.

Definition 2.3 (Synchronous semantics of session types)**.** We define the relation $\twoheadrightarrow_1$ as the subset of $\twoheadrightarrow_{\infty}$ where: (i) each queue in γ contains at most one message; (ii) there exists at most one non-empty queue in γ. Hereafter, we use the symbol $\circ \in \{1, \infty\}$ to parameterise various notions over the synchronous/asynchronous semantics.

Example 2.4. Let $C = \mathtt{a!.b?}$, $D = \mathtt{b!.a?}$ and $\gamma = \mathsf{A} : C[] \mid \mathsf{B} : D[]$. Under the $\twoheadrightarrow_1$ semantics, we have the following traces, where no participant reaches a success state[1]:

$$\gamma \xrightarrow{\mathsf{A}:\mathtt{a!}}\!\!\!\!\rightarrow_1 \mathsf{A} : \mathtt{b?}\; [] \mid \mathsf{B} : D \; [\mathtt{a!}] \not\twoheadrightarrow_1 \qquad \text{and} \qquad \gamma \xrightarrow{\mathsf{B}:\mathtt{b!}}\!\!\!\!\rightarrow_1 \mathsf{A} : C \; [\mathtt{b!}] \mid \mathsf{B} : \mathtt{a?}\; [] \not\twoheadrightarrow_1$$

Under the $\twoheadrightarrow_{\infty}$ semantics, instead, we have the following traces, all leading to the success state for both participants:

$$\gamma \xrightarrow{\mathsf{A}:\mathtt{a!}}\!\!\!\!\rightarrow_{\infty} \mathsf{A}{:}\mathtt{b?}[]\mid\mathsf{B}{:}D[\mathtt{a!}] \xrightarrow{\mathsf{B}:\mathtt{b!}}\!\!\!\!\rightarrow_{\infty} \mathsf{A}{:}\mathtt{b?}[\mathtt{b!}]\mid\mathsf{B}{:}\mathtt{a?}[\mathtt{a!}] \xrightarrow{\mathsf{A}:\mathtt{b?}}\!\!\!\!\rightarrow_{\infty} \mathsf{A}{:}\mathtt{1}[]\mid\mathsf{B}{:}\mathtt{a?}[\mathtt{a!}] \xrightarrow{\mathsf{B}:\mathtt{a?}}\!\!\!\!\rightarrow_{\infty} \mathsf{A}{:}\mathtt{1}[]\mid\mathsf{B}{:}\mathtt{1}[]$$
$$\gamma \xrightarrow{\mathsf{A}:\mathtt{a!}}\!\!\!\!\rightarrow_{\infty} \mathsf{A}{:}\mathtt{b?}[]\mid\mathsf{B}{:}D[\mathtt{a!}] \xrightarrow{\mathsf{B}:\mathtt{b!}}\!\!\!\!\rightarrow_{\infty} \mathsf{A}{:}\mathtt{b?}[\mathtt{b!}]\mid\mathsf{B}{:}\mathtt{a?}[\mathtt{a!}] \xrightarrow{\mathsf{B}:\mathtt{a?}}\!\!\!\!\rightarrow_{\infty} \mathsf{A}{:}\mathtt{b?}[\mathtt{b!}]\mid\mathsf{B}{:}\mathtt{1}[] \xrightarrow{\mathsf{A}:\mathtt{b?}}\!\!\!\!\rightarrow_{\infty} \mathsf{A}{:}\mathtt{1}[]\mid\mathsf{B}{:}\mathtt{1}[]$$
$$\gamma \xrightarrow{\mathsf{B}:\mathtt{b!}}\!\!\!\!\rightarrow_{\infty} \mathsf{A}{:}C[\mathtt{b!}]\mid\mathsf{B}{:}\mathtt{a?}[] \xrightarrow{\mathsf{A}:\mathtt{a!}}\!\!\!\!\rightarrow_{\infty} \mathsf{A}{:}\mathtt{b?}[\mathtt{b!}]\mid\mathsf{B}{:}\mathtt{a?}[\mathtt{a!}] \xrightarrow{\mathsf{A}:\mathtt{b?}}\!\!\!\!\rightarrow_{\infty} \mathsf{A}{:}\mathtt{1}[]\mid\mathsf{B}{:}\mathtt{a?}[\mathtt{a!}] \xrightarrow{\mathsf{B}:\mathtt{a?}}\!\!\!\!\rightarrow_{\infty} \mathsf{A}{:}\mathtt{1}[]\mid\mathsf{B}{:}\mathtt{1}[]$$
$$\gamma \xrightarrow{\mathsf{B}:\mathtt{b!}}\!\!\!\!\rightarrow_{\infty} \mathsf{A}{:}C[\mathtt{b!}]\mid\mathsf{B}{:}\mathtt{a?}[] \xrightarrow{\mathsf{A}:\mathtt{a!}}\!\!\!\!\rightarrow_{\infty} \mathsf{A}{:}\mathtt{b?}[\mathtt{b!}]\mid\mathsf{B}{:}\mathtt{a?}[\mathtt{a!}] \xrightarrow{\mathsf{B}:\mathtt{a?}}\!\!\!\!\rightarrow_{\infty} \mathsf{A}{:}\mathtt{b?}[\mathtt{b!}]\mid\mathsf{B}{:}\mathtt{1}[] \xrightarrow{\mathsf{A}:\mathtt{b?}}\!\!\!\!\rightarrow_{\infty} \mathsf{A}{:}\mathtt{1}[]\mid\mathsf{B}{:}\mathtt{1}[]$$

The following theorem states that session types with the asynchronous semantics simulate Turing machines. Our result is closely related to analogous expressiveness results for Communicating Finite State Machines (CFSMs) [19, 30]. The construction in [19] is used to show that a system of two CFSMs with *mixed choices* can simulate Turing machines; the one in [30] is technically different, as it uses two copies of the same *non-deterministic* CFSM with no mixed choices. Compared to [19, 30], our result is slightly stronger: indeed, session types are deterministic (in the sense of constraint (ii) in Definition 2.1) and without mixed

[1]We will see that both final configurations are deadlocks under $\twoheadrightarrow_1$, by Definition 2.6.

choices, so our encoding has to work in a restricted model compared to CFSMs. Essentially, our proof constructs a system of two deterministic CFSMs without mixed choice.

Theorem 2.5. *Session types (with the $\twoheadrightarrow_\infty$ semantics) can simulate Turing machines.*

Proof. (Sketch). Assume as given a deterministic Turing machine M with states $Q = \{q_1, \ldots, q_n\}$, initial/halting states $q_0, q_h \in Q$, tape alphabet Σ (with blank symbol $\#$), and transition function δ. Let $s_0 \ldots s_n \in \Sigma^*$ be the initial tape. Each configuration of M can be represented as a string $s_0 s_1 \ldots s_{i-1} q_k s_i \ldots s_m \, end$, which is obtained from the tape $s_0 s_1 \ldots s_m$ by inserting the state/symbol q_k at the position of the head (i).

We now define two session types C, D such that $\mathsf{A} : C \; [] \mid \mathsf{B} : D \; []$ simulates M stepwise. Intuitively, C acts as a transducer that inputs the symbols of a configuration of M, and outputs the symbols of the next configuration. Instead, D simply outputs the initial configuration of M and then echoes every symbol back, until a *stop* message is received. To perform its task, C works in a streaming fashion, emitting the output configuration while reading the input one. Indeed, to determine the next symbol to emit, we need only a bounded lookahead. In this way, the procedure requires a finite amount of memory, and is amenable to be implemented in a session type. After reaching the *end* marker, C restarts, so to handle the new configuration received from D. If C eventually finds a configuration in the halting state, it signals *stop* to D.

More precisely, we define the session type D as follows:

$$D \;=\; \mathtt{q_0!}\,.\,\mathtt{s_0!}\,.\,\ldots\,.\,\mathtt{s_n!}\,.\,\mathtt{end!}\,.\,(\mathtt{rec}\,X\,.\,\mathtt{stop?}\;\&\;\&_{\mathtt{x}\in\Sigma\cup Q\cup\{end\}}\,\mathtt{x?}\,.\,\mathtt{x!}\,.\,X)$$

In order to define the session type C, it is convenient to first give a set of (recursive) defining equations, and then apply Bekić theorem [16] (see Theorem 10.1 in [46]) to obtain the (recursive) term C. To this purpose, we introduce some auxiliary notation. The shortcut $write\langle x\rangle.T$ stands for $\mathtt{x!}.T$, or for T if x is the special value $\bot$. Dually, $read(x).T$ stands for $\&_{x\in\Sigma\cup Q\cup\{end\}}\,\mathtt{x?}\,.\,T$; here the index x may appear in T, hence it is considered a binder. The defining equations are as follows:

$$T_0(p) = read(s).T_1(p,s)$$

$$T_1(p,s) = \begin{cases} write\langle p\rangle.T_0(s) & \text{if } s \in \Sigma \\ read(s_2).T_2(p,s,s_2) & \text{if } s \in Q \\ write\langle p\rangle.write\langle\#\rangle.write\langle end\rangle.T_0(\bot) & \text{if } s = end \end{cases}$$

$$T_2(p,q,s_2) = \begin{cases} write\langle p\rangle.write\langle q_h\rangle.write\langle s_2\rangle.write\langle stop\rangle & \text{if } q = q_h \\ write\langle q'\rangle.write\langle p\rangle.T_0(s_3) & \text{if } dir = L \\ write\langle p\rangle.write\langle q'\rangle.T_0(s_3) & \text{if } dir = - \\ write\langle p\rangle.write\langle s_3\rangle.write\langle q'\rangle.read(s_4).T_0(s_4) & \text{if } dir = R \end{cases}$$

$$\text{where } (q', s_3, dir) = \delta(q, s_2)$$

and we obtain C by applying Bekić theorem on $T_0(\bot)$.

The contract configuration $\mathsf{A} : C \; [] \mid \mathsf{B} : D \; []$ simulates M on the initial tape. This is because C receives its own outputs back, and so repeatedly performs the steps of M. □

2.3. **Compliance.** We define compliance between session types by taking inspiration from the notion of safety on Communicating Finite State Machines [23, 38]. We start with some auxiliary definitions.

Definition 2.6 (Deadlock & message-obliviousness)**.** We say that a configuration γ is:

- a *deadlock under the* $\twoheadrightarrow_\circ$ *semantics* iff: $\gamma \not\twoheadrightarrow_\circ$ and $\gamma \neq \mathsf{A} : \mathsf{1}\ [] \mid \mathsf{B} : \mathsf{1}\ []$
- *message-oblivious under the* $\twoheadrightarrow_\circ$ *semantics* iff $\gamma = \mathsf{A} : C\ [\mathtt{a}!\beta_\mathsf{A}] \mid \mathsf{B} : D\ [\beta_\mathsf{B}]$ and

$$\gamma \xrightarrow{\mathsf{A}:\mathtt{a}_1!}\!\!\!\!\twoheadrightarrow_\circ \cdots \xrightarrow{\mathsf{A}:\mathtt{a}_k!}\!\!\!\!\twoheadrightarrow_\circ \gamma' \quad \text{implies} \quad \gamma' \not\xrightarrow{\mathsf{A}:\mathtt{a}?}\!\!\!\!\twoheadrightarrow_\circ$$

  or the symmetric condition holds when the roles of A and B are swapped.

The notion of deadlock is standard; *message-obliviousness* characterizes the configurations where an enqueued message is left unread forever.

Our compliance relation requires that no reachable configuration is a deadlock or is message-oblivious.

Definition 2.7 (Compliance)**.** We say that C and D are *compliant under the* $\twoheadrightarrow_\circ$ *semantics* (in symbols, $C \bowtie_\circ D$) whenever: $\mathsf{A} : C\ [] \mid \mathsf{B} : D\ [] \twoheadrightarrow_\circ^* \gamma$ implies that γ is neither deadlock nor a message-oblivious configuration under the $\twoheadrightarrow_\circ$ semantics.

Note that when applying this notion to the synchronous semantics, deadlock-freedom implies absence of message-oblivious configurations, and so our compliance $\bowtie_1$ coincides exactly with the usual (symmetric) progress-based notion in [3, 11, 7].

Theorem 2.8. *$\bowtie_1$ is decidable; $\bowtie_\infty$ is undecidable.*

Proof. Decidability of $\bowtie_1$ follows because the state space of $\mathsf{A} : C\ [] \mid \mathsf{B} : D\ []$ under $\twoheadrightarrow_1$ is finite. Undecidability of $\bowtie_\infty$ follows by Theorem 2.5. □

The following theorem, which is a corollary of results in [11, 43], shows that synchronous compliance implies asynchronous compliance. We can then use a decision procedure for synchronous compliance (which is decidable) to safely approximate asynchronous compliance (which is undecidable).

Theorem 2.9. $\bowtie_1 \subsetneq \bowtie_\infty$.

Proof. (Sketch). The inclusion $\subseteq$ is proved in two parts: first we show that deadlock-freedom holds in the asynchronous semantics; second, we show that absence of message-oblivious configurations holds as well.

The first part follows from Proposition 3 in [11], showing that, for session types, client-biased progress (denoted by $\dashv$ in [11]), can be lifted from synchronous to asynchronous semantics. Our deadlock-freedom is equivalent to the intersection between client-biased progress $\dashv$ and the symmetric server-biased progress $\vdash$. Applying Proposition 3 of [11] twice, we get that the relation $\dashv\vdash = \dashv \cap \vdash$ (i.e., our deadlock freedom) can be lifted to the asynchronous semantics as well.

For the second part we exploit Theorems 4.9, 4.13 and 5.22 in [43]. These results establish relations between progress, the *I/O compliance* relation of [11] (a notion of compliance which is stricter than progress on arbitrary LTSs, and coincides with it on the LTSs of session types), and the notion *orphan message configuration* of [43] (i.e., configurations containing messages which are sent but never received). More precisely, assume that $C \bowtie_1 D$, i.e. that $C \dashv\vdash D$ using the notation of [43]. Then:

(1) by Theorem 4.9 in [43], $C \dashv\vdash D$ iff C and D are I/O compliant;

$\mathsf{A}, \mathsf{B}, \ldots$	Participant names	$u, v, \ldots$	Channels, comprising:
$a, b, \ldots$	Actions	$\quad s, t, \ldots \in \mathcal{N}$	Session names
$C, D, \ldots$	Contracts	$\quad x, y, \ldots \in \mathcal{V}$	Variables
$\gamma, \gamma', \ldots$	Contract configurations	$P, Q, \ldots$	Processes
$\gamma \twoheadrightarrow \gamma'$	Transition of contract configurations	$S, S', \ldots$	Systems
		$S \to S'$	Transition of systems

TABLE 1. Summary of notation.

(2) by Theorem 4.13 in [43], I/O compliance between session types is preserved when passing from the synchronous to the asynchronous semantics;
(3) by Theorem 5.22 in [43], if two session types are I/O compliant in the asynchronous semantics, their asynchronous execution cannot have orphan message configurations;
(4) by Definition 5.5 in [43] and by Definition 2.6, orphan message and message-oblivious configurations coincide for session types.

To obtain the thesis, assume that $C \bowtie_1 D$ and $\mathsf{A} : C \; [] \mid \mathsf{B} : D \; [] \twoheadrightarrow_\infty^* \gamma$. By items (1)–(4) above, γ is *not* a message-oblivious configuration.

The strict inclusion it witnessed by $C = \mathtt{a!.b?}$ and $D = \mathtt{b!.a?}$, since $C \not\bowtie_1 D$ and $C \bowtie_\infty D$ (see the traces in Example 2.4). □

3. The CO_2 calculus

We now present an instance the process calculus CO_2 [13] with the contracts of Section 2.

3.1. **Syntax.** Let $\mathcal{V}$ and $\mathcal{N}$ be disjoint sets of, respectively, *session variables* (ranged over by $x, y, \ldots$) and *session names* (ranged over by $s, t, \ldots$), and let $u, v, \ldots$ range over *channels* $\mathcal{V} \cup \mathcal{N}$. Finite sequences are in bold, e.g. $\boldsymbol{u}, \boldsymbol{v}, \ldots$ denote finite sequences of channels.

Definition 3.1 (CO_2 syntax)**.** The syntax of CO_2 is defined as follows:

$$P \;::=\; \textstyle\sum_i \pi_i.P_i \;\mid\; P \mid P \;\mid\; (u)P \;\mid\; (\mathtt{rec}\ \mathrm{X}(\boldsymbol{y}).\ P)(\boldsymbol{u}) \;\mid\; \mathrm{X}(\boldsymbol{u}) \qquad \textit{(Processes)}$$

$$\pi \;::=\; \tau \;\mid\; \mathtt{tell}{\downarrow_u}\, C \;\mid\; \mathtt{do}_u\, a \qquad \textit{(Prefixes)}$$

$$S \;::=\; \mathbf{0} \;\mid\; \mathsf{A}[P] \;\mid\; s[\gamma] \;\mid\; \{{\downarrow_u}\, C\}_{\mathsf{A}} \;\mid\; (u)S \;\mid\; S \mid S \qquad \textit{(Systems)}$$

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

(1) recursion is prefix-guarded;
(2) in $(\boldsymbol{u})(\mathsf{A}[P] \mid \mathsf{B}[Q] \mid \cdots)$, it must be $\mathsf{A} \neq \mathsf{B}$;
(3) in $(\boldsymbol{u})(s[\gamma] \mid t[\gamma'] \mid \cdots)$, it must be $s \neq t$;
(4) we denote with $\mathtt{fuse}$ and $\tau_?$ two special prefixes which cannot occur in processes, and with K a special participant name which cannot occur in systems.

Processes specify the behaviour of participants; they can be prefix-guarded finite sums $\sum_i \pi_i.P_i$, parallel compositions $P \mid Q$, delimited processes $(u)P$, recursive calls $\mathrm{X}(\boldsymbol{u})$, or recursive processes $(\mathtt{rec}\ \mathrm{X}(\boldsymbol{y}).\ P)(\boldsymbol{u})$. Prefixes include the silent action τ, *contract advertisement* $\mathtt{tell}{\downarrow_u}\, C$, and *action execution* $\mathtt{do}_u\, a$. In a prefix $\pi \in \{\mathtt{tell}{\downarrow_u}\, C, \mathtt{do}_u\, a\}$, the identifier u refers to the target session involved in the execution of π. The special prefixes

$$(\boldsymbol{u})\mathsf{A}[(\boldsymbol{v})P] \equiv (\boldsymbol{u},\boldsymbol{v})\mathsf{A}[P] \qquad Z \mid \mathbf{0} \equiv Z \qquad Z \mid Z' \equiv Z' \mid Z \qquad (Z \mid Z') \mid Z'' \equiv Z \mid (Z' \mid Z'')$$
$$\alpha\text{-conversion of delimited channels} \qquad Z \mid (u)Z' \equiv (u)(Z \mid Z') \text{ if } u \notin \mathrm{fnv}(Z)$$
$$(u)(v)Z \equiv (v)(u)Z \qquad (u)Z \equiv Z \text{ if } u \notin \mathrm{fnv}(Z) \qquad \{\downarrow_s C\}_{\mathsf{A}} \equiv \mathbf{0}$$

FIGURE 2. Structural congruence for CO_2 (Z ranges over processes/systems).

`fuse` and $\tau_?$ are technical, and are only used in labels of system transitions (see Figure 3). Intuitively, $\tau_?$ labels $\mathtt{do}_x$ actions under a delimitation (x), while `fuse` labels session creations. Systems are parallel compositions of *agents* $\mathsf{A}[P]$, *sessions* $s[\gamma]$, *latent contracts* $\{\downarrow_u C\}_{\mathsf{A}}$, and delimited systems $(u)S$. A *latent contract* $\{\downarrow_x C\}_{\mathsf{A}}$ represents a contract C signed by participant A but not stipulated yet; the variable x will be instantiated to a fresh session name upon stipulation.

Delimitation (u) binds channels, both in processes and systems. In the recursive process $(\mathtt{rec}\ \mathrm{X}(\boldsymbol{y}).\ P)(\boldsymbol{u})$, the variables in $\boldsymbol{y}$ bind their occurrences in P. Variables and names which are not bound by such binders are *free*; their sets are denoted by $\mathrm{fv}(\_)$ and $\mathrm{fn}(\_)$, respectively. For a system/process Z, we denote with $\mathrm{fnv}(Z)$ its free channels, i.e. $\mathrm{fnv}(Z) = \mathrm{fn}(Z) \cup \mathrm{fv}(Z)$, and we say that Z is *closed* when $\mathrm{fnv}(Z) = \emptyset$.

Notation 3.2. *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* $1 \notin I$*. As usual, we omit trailing occurrences of* $\mathbf{0}$*. When the sequence of arguments* $\boldsymbol{u}$ *is empty, we will write* $\mathrm{X}()$ *instead of* $\mathrm{X}(\boldsymbol{u})$*.*

3.2. **Semantics.** We formalise the semantics of CO_2 as a reduction relation on systems (Definition 3.3). This uses a structural congruence $\equiv$, which is the smallest congruence relation satisfying the equations in Figure 2. The axioms in Figure 2 are mostly standard; note that $(\boldsymbol{u})\mathsf{A}[(\boldsymbol{v})P] \equiv (\boldsymbol{u},\boldsymbol{v})\mathsf{A}[P]$ allows to move delimitations between systems and processes; we use the last axiom $\{\downarrow_s C\}_{\mathsf{A}} = \mathbf{0}$ to collect garbage terms possibly arising from variable substitutions. In order to define honesty in Section 4, we decorate transitions with labels, by writing $\xrightarrow{\mathsf{A}:\pi}$ for a reduction where participant A fires prefix π.

Definition 3.3 (CO_2 semantics)**.** The relation $\xrightarrow{\mathsf{A}:\pi}$ between systems (considered up-to structural congruence $\equiv$, and parametric w.r.t. synchronous/asynchronous semantics of session types) is the smallest relation closed under the rules of Figure 3.

We now comment on the rules in Figure 3. Rule [TAU] is standard. Rule [TELL] advertises the latent contract $\{\downarrow_x C\}_{\mathsf{A}}$. Rule [FUSE] stipulates contract: if there are two compliant contracts, a fresh session s is created; the latent contracts are consumed, and the substitution σ is applied to the system, to instantiate the variables x, y to the session name s. The participant K performing this move models a contract broker, similar to the one in [6]. Rule [DO] allows a participant A to perform some action a in a session s, whose state γ evolves accordingly to γ'. Rule [DEL] allows a system to evolve under a delimitation. Note that the label π fired in the premise becomes τ or $\tau_?$ in the consequence, when π contains the delimited channel. This transformation is defined by the function $\mathrm{del}_u(\pi)$: for instance, $(x)\,\mathsf{A}[\mathtt{tell}\downarrow_x C.P] \xrightarrow{\mathsf{A}:\tau} (x)\,(\mathsf{A}[P] \mid \{\downarrow_x C\}_{\mathsf{A}})$. Here, it would make little sense to have the

$$\mathsf{A}[\tau.P + P' \mid Q] \xrightarrow{\mathsf{A}:\tau}_\circ \mathsf{A}[P \mid Q] \quad [\textsc{Tau}]$$

$$\mathsf{A}[\mathtt{tell}\downarrow_u C.P + P' \mid Q] \xrightarrow{\mathsf{A}:\mathtt{tell}\downarrow_u C}_\circ \mathsf{A}[P \mid Q] \mid \{\downarrow_u C\}_\mathsf{A} \quad [\textsc{Tell}]$$

$$\frac{\gamma \xrightarrow{\mathsf{A}:a}\!\!\!\!\rightarrow_\circ \gamma'}{\mathsf{A}[\mathtt{do}_s\, a.P + P' \mid Q] \mid s[\gamma] \xrightarrow{\mathsf{A}:\mathtt{do}_s\, a}_\circ \mathsf{A}[P \mid Q] \mid s[\gamma']} \quad [\textsc{Do}]$$

$$\frac{\mathsf{A}[P\{\mathtt{rec}\ \mathrm{X}(\boldsymbol{y}).\ P/\mathrm{X}\}\{\boldsymbol{u}/\boldsymbol{y}\} \mid Q] \mid S \xrightarrow{\mathsf{A}:\pi}_\circ S'}{\mathsf{A}[(\mathtt{rec}\ \mathrm{X}(\boldsymbol{y}).\ P)(\boldsymbol{u}) \mid Q] \mid S \xrightarrow{\mathsf{A}:\pi}_\circ S'} \quad [\textsc{Rec}]$$

$$\frac{C \bowtie_\circ D \quad \gamma = \mathsf{A} : C\ [] \mid \mathsf{B} : D\ [] \quad \sigma = \{^s/_{x,y}\} \quad s \notin \mathrm{fnv}(S)}{(x,y)(S \mid \{\downarrow_x C\}_\mathsf{A} \mid \{\downarrow_y D\}_\mathsf{B}) \xrightarrow{\mathsf{K}:\mathtt{fuse}}_\circ (s)(S\sigma \mid s[\gamma])} \quad [\textsc{Fuse}]$$

$$\frac{S \xrightarrow{\mathsf{A}:\pi}_\circ S'}{S \mid S'' \xrightarrow{\mathsf{A}:\pi}_\circ S' \mid S''} \quad [\textsc{Par}]$$

$$\frac{S \xrightarrow{\mathsf{A}:\pi}_\circ S'}{(u)S \xrightarrow{\mathsf{A}:\mathrm{del}_u(\pi)}_\circ (u)S'} \quad [\textsc{Del}] \qquad \text{where } \mathrm{del}_u(\pi) = \begin{cases} \tau & \text{if } \pi = \mathtt{tell}\downarrow_u C \\ \tau_? & \text{if } \pi = \mathtt{do}_u\, a \\ \pi & \text{otherwise} \end{cases}$$

FIGURE 3. Reduction semantics of CO_2.

label $\mathsf{A} : \mathtt{tell}\downarrow_x C$, as x (being delimited) may be α-converted. Rule [Par] is standard. Rule [Rec] is used to unfold recursions.

3.3. **Examples.**

Example 3.4 (Honest/dishonest choice)**.** Consider the following processes:

$$P = (x)\ \mathtt{tell}\downarrow_x (\mathtt{a!} \oplus \mathtt{b!}).\mathtt{do}_x\, \mathtt{a!}$$
$$Q = (y)\ \mathtt{tell}\downarrow_y (\mathtt{a?} \,\&\, \mathtt{b?}).\mathtt{do}_y\, \mathtt{a?}$$

A possible computation of the system $S_0 = \mathsf{A}[P] \mid \mathsf{B}[Q]$, both under the synchronous and the asynchronous semantics, is the following:

$$S_0 \xrightarrow{\mathsf{B}:\tau}_\circ \quad \mathsf{A}[P] \mid (y)\ (\mathsf{B}[\mathtt{do}_y\, \mathtt{a?}] \mid \{\downarrow_y \mathtt{a?} \,\&\, \mathtt{b?}\}_\mathsf{B}) \tag{1}$$

$$\xrightarrow{\mathsf{A}:\tau}_\circ \quad (x,y)\ (\mathsf{A}[\mathtt{do}_x\, \mathtt{a!}] \mid \mathsf{B}[\mathtt{do}_y\, \mathtt{a!}] \mid \{\downarrow_x \mathtt{a!} \oplus \mathtt{b!}\}_\mathsf{A} \mid \{\downarrow_y \mathtt{a?} \,\&\, \mathtt{b?}\}_\mathsf{B}) \tag{2}$$

$$\xrightarrow{\mathsf{K}:\mathtt{fuse}}_\circ \quad (s)\ (\mathsf{A}[\mathtt{do}_s\, \mathtt{a!}] \mid \mathsf{B}[\mathtt{do}_s\, \mathtt{a?}] \mid s[\mathsf{A} : \mathtt{a!} \oplus \mathtt{b!}\ [] \mid \mathsf{B} : \mathtt{a?} \,\&\, \mathtt{b?}\ []]) \tag{3}$$

$$\xrightarrow{\mathsf{A}:\tau_?}_\circ \quad (s)\ (\mathsf{A}[\mathbf{0}] \mid \mathsf{B}[\mathtt{do}_s\, \mathtt{a?}] \mid s[\mathsf{A} : \mathtt{1}\ [] \mid \mathsf{B} : \mathtt{a?} \,\&\, \mathtt{b?}\ [\mathtt{a!}]]) \tag{4}$$

$$\xrightarrow{\mathsf{B}:\tau_?}_\circ \quad (s)\ (\mathsf{A}[\mathbf{0}] \mid \mathsf{B}[\mathbf{0}] \mid s[\mathsf{A} : \mathtt{1}\ [] \mid \mathsf{B} : \mathtt{1}\ []]) \tag{5}$$

Transitions (1) and (2) are obtained by applying rules [Tell], [Par], and [Del], and by using structural congruence to move delimitations. Transition (3) is obtained by rule [Fuse], since

$\mathtt{a!} \oplus \mathtt{b!}$ and $\mathtt{a?} \,\&\, \mathtt{b?}$ are compliant (both under $\bowtie_1$ and $\bowtie_\infty$). Finally, transitions (4) and (5) are obtained by rule [Do].

We anticipate that, under both semantics of CO_2, P is honest while Q is dishonest. Intuitively, P is honest because, in all possible contexts, it always fulfils its contract $\mathtt{a!} \oplus \mathtt{b!}$ by performing $\mathtt{do}_x\,\mathtt{a!}$. The process Q is not honest because e.g. the system $\mathsf{A}[(x)\,\mathtt{tell}\downarrow_x \mathtt{b!}.\mathtt{do}_x\,\mathtt{b!}] \mid \mathsf{B}[Q]$ admits a computation leading to:

$$(s)\ (\mathsf{A}[\mathbf{0}] \ \mid\ \mathsf{B}[\mathtt{do}_s\,\mathtt{a?}] \ \mid\ s[\mathsf{A} : \mathtt{1}\ [] \mid \mathsf{B} : \mathtt{a?} \,\&\, \mathtt{b?}\ [\mathtt{b!}]])$$

where B is not fulfilling its contract at session s. Note in fact that B declares in his contract to be able to read $\mathtt{a}$ or $\mathtt{b}$ (chosen *externally*, by the other endpoint of the session), while the only action in the process of B is to read $\mathtt{a}$. We will formally establish the honesty of P in Example 7.3, and the dishonesty of Q in Example 4.6. □

Example 3.5 (Dishonest interleaving)**.** Let:

$$P \ = \ (x,y)\,\mathtt{tell}\downarrow_x \mathtt{a?}\,.\,\mathtt{tell}\downarrow_y \mathtt{b!}\,.\,\mathtt{do}_x\,\mathtt{a?}\,.\,\mathtt{do}_y\,\mathtt{b!}$$

We anticipate that P is *not* honest (neither under the synchronous nor the asynchronous semantics of CO_2). Indeed, in both semantics we can reduce the system

$$\mathsf{A}[P] \ \mid\ \mathsf{B}[(z)\,\mathtt{tell}\downarrow_z \mathtt{b?}\,.\,\mathtt{do}_z\,\mathtt{b?}] \ \mid\ \mathsf{C}[(w)\,\mathtt{tell}\downarrow_w \mathtt{a!}\,.\,\mathbf{0}]$$

into the system:

$$S \ = \ (s,t)\,(\mathsf{A}[\mathtt{do}_t\,\mathtt{a?}.\mathtt{do}_s\,\mathtt{b!}] \mid \mathsf{B}[\mathtt{do}_z\,\mathtt{b?}] \mid \mathsf{C}[\mathbf{0}] \mid t[\mathsf{A} : \mathtt{a?}\ [] \mid \mathsf{C} : \mathtt{a!}\ []] \mid s[\mathsf{A} : \mathtt{b!}\ [] \mid \mathsf{B} : \mathtt{b?}\ []])$$

The system S cannot reduce further. Indeed, C (dishonestly) avoids to perform the internal choice $\mathtt{a!}$ required by his contract, and so A is stuck, waiting for $\mathtt{a?}$ from C. Intuitively, P is dishonest because A does not perform the obligation $\mathtt{b!}$ at session s. This intuitive argument will be made formal in Example 4.7.

We anticipate that an honest variant of the process P would be the following:

$$Q \ = \ (x,y)\,\mathtt{tell}\downarrow_x \mathtt{a?}\,.\,\mathtt{tell}\downarrow_y \mathtt{b!}\,.\,(\mathtt{do}_x\,\mathtt{a?} \mid \mathtt{do}_y\,\mathtt{b!})$$

Note that in Q the causal dependency between $\mathtt{do}_x\,\mathtt{a?}$ and $\mathtt{do}_y\,\mathtt{b?}$ is lost. Another honest variant of P, preserving such causal dependency (but slightly changing the contract at y) will be presented in Example 7.6. □

Example 3.6 (Online store)**.** We formalise the online store S from Section 1 as the CO_2 process P below, where we assume that $R_i = \mathbf{0}$, for $i \in 1..4$.

$$\begin{aligned}
P \ &= \ (x)\,\mathtt{tell}\downarrow_x C_{\mathsf{B}}\,.\big(\mathtt{do}_x\,\mathtt{buyA?}.\mathtt{do}_x\,\mathtt{pay1E?}.\mathtt{do}_x\,\mathtt{shipA!} \ + \ \mathtt{do}_x\,\mathtt{buyB?}.P'(x)\big)\\
P'(x) \ &= \ \mathtt{do}_x\,\mathtt{quote1E!}.\mathtt{do}_x\,\mathtt{pay1E?}.\mathtt{do}_x\,\mathtt{shipB!} \ + \ (y)\,\mathtt{tell}\downarrow_y C_{\mathsf{D}}.P''(x,y)\\
P''(x,y) \ &= \ \mathtt{do}_y\,\mathtt{buyB!}\ .\ \mathtt{do}_x\,\mathtt{quote3E!}\ .\ (\\
&\qquad \mathtt{do}_x\,\mathtt{pay3E?}\ .\ (\\
&\qquad\quad \mathtt{do}_y\,\mathtt{pay2E!}\ .\ (\\
&\qquad\qquad \mathtt{do}_y\,\mathtt{shipB?}\ .\ \mathtt{do}_x\,\mathtt{shipB!}\\
&\qquad\qquad \ + \ R_1\\
&\qquad\quad)\ + \ R_2\\
&\qquad)\ + \ R_3\\
&\quad)\ + \ R_4
\end{aligned}$$

As anticipated in Section 1, this process is not honest; we will show later on in Example 7.7 an honest version of the online store (advertising the same contracts). This version will be obtained by suitably instantiating the processes R_i within P''. □

4. Honesty

We now define when participants are *honest*, i.e. when they fulfil their contracts, in *all* execution contexts. We start by introducing some auxiliary notions, which are parameterised over the synchronous/asynchronous semantics of session types (i.e., over $\circ \in \{1, \infty\}$). The definition of honesty given in this section extends to the asynchronous case the one in [15].

The set of *obligations* $\mathrm{O}_\circ^{\mathsf{A}@s}(S)$ yields the actions (at a session s in S) participant A must choose from in order to respect her contract.

Definition 4.1 (Obligations)**.** We define the set of actions $\mathrm{O}_\circ^{\mathsf{A}@s}(S)$ as follows:

$$\mathrm{O}_\circ^{\mathsf{A}@s}(S) \;=\; \{a \mid \exists \gamma, S' \,:\, S \equiv s[\gamma] \mid S' \text{ and } \gamma \xrightarrow{\mathsf{A}:a}\!\!\!\!\!\rightarrow\!\!\!\!\!\!\rightarrow_\circ\}$$

The set $S \downarrow_u^{\mathsf{A}}$ (called *ready-do* set) collects the actions a that the process of A would perform, if enabled in session u. The set $S \Downarrow_\circ^{\mathsf{A}@u}$ is a weak variant of $S \downarrow_u^{\mathsf{A}}$ (parameterised over the $\rightarrow_\circ$ semantics), which contains the next reachable ready actions of A.

Definition 4.2 (Ready-do)**.** We define the sets of actions $S \downarrow_u^{\mathsf{A}}$ and $S \Downarrow_\circ^{\mathsf{A}@u}$ as:

$$\begin{array}{rcl} S \downarrow_u^{\mathsf{A}} & = & \{a \mid \exists \boldsymbol{v}, P, P', Q, S' \,.\; S \equiv (\boldsymbol{v})\,(\mathsf{A}[\mathtt{do}_u\, a.P + P' \mid Q] \mid S') \;\wedge\; u \notin \boldsymbol{v}\} \\ S \Downarrow_\circ^{\mathsf{A}@u} & = & \left\{a \mid \exists S' : S \xrightarrow{\neq(\mathsf{A}:\, \mathtt{do}_u)}_\circ{}^{*}\; S' \text{ and } a \in S' \downarrow_u^{\mathsf{A}}\right\} \end{array}$$

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

A participant A is *ready* in session s when either A has no obligations at s, or A is weakly ready to perform *some* output action in her obligations, or A is weakly ready to perform *all* the input actions in her obligations. This reflects the fact that to respect an internal choice it is enough to perform one of its outputs, while to respect an external choice one has to be able to perform all of its inputs.

Definition 4.3 (Readiness)**.** $\mathrm{Rdy}_\circ^{\mathsf{A}@s}$ is the set of systems S such that:

$$\mathrm{O}_\circ^{\mathsf{A}@s}(S) = \emptyset \quad\vee\quad \mathrm{O}_\circ^{\mathsf{A}@s}(S) \cap \mathtt{A}^{!} \cap S \Downarrow_\circ^{\mathsf{A}@s} \neq \emptyset \quad\vee\quad \emptyset \neq (\mathrm{O}_\circ^{\mathsf{A}@s}(S) \cap \mathtt{A}^{?}) \subseteq S \Downarrow_\circ^{\mathsf{A}@s}$$

Then, we say that A is $\circ$-*ready in* S iff for all $s, S', \boldsymbol{u}$, $S \equiv (\boldsymbol{u})S'$ implies $S' \in \mathrm{Rdy}_\circ^{\mathsf{A}@s}$.

Remark 4.4. *Definition 4.3 could be simplified as* $\mathrm{O}_\circ^{\mathsf{A}@s}(S) = \emptyset \vee \mathrm{O}_\circ^{\mathsf{A}@s}(S) \cap S \Downarrow_\circ^{\mathsf{A}@s} \neq \emptyset$*, because* $(\mathrm{O}_\circ^{\mathsf{A}@s}(S) \cap \mathtt{A}^{?})$ *contains at most one element. However, we prefer to use the same definition of* [15] *to inherit its undecidability results.*

A process P is honest when, for all contexts where $\mathsf{A}[P]$ may be engaged in, A is persistently ready in all the reducts of that context.

Definition 4.5 (Honesty)**.** We say that:

(1) S *is* A*-free* iff it has no latent/stipulated contracts of A, nor processes of A
(2) P is $\circ$-*honest in* S iff $\forall \mathsf{A} : (S \text{ is } \mathsf{A}\text{-free} \wedge \mathsf{A}[P] \mid S \rightarrow_\circ^* S') \implies \mathsf{A}$ is $\circ$-ready in S'
(3) P is $\circ$-*honest* iff $\forall S : P$ is $\circ$-honest in S.

The A-freeness requirement in Definition 4.5 is used just to rule out those systems which already carry stipulated or latent contracts of A outside $\mathsf{A}[P]$, e.g., $\{\downarrow_x \mathtt{pay!}\}_{\mathsf{A}}$. In the absence of A-freeness, the context could trivially make a process dishonest. Note that $\mathsf{A}[P]$ is vacuously honest when P advertises no contracts.

We prove below the dishonesty of the process Q from Example 3.4 and of the process P from Example 3.5. Note that processes P from Example 3.4 and Q from Example 3.5 are honest. However, at this point of the paper we do not have a convenient proof technique to cope with the universal quantification over contexts required in Definition 4.5. We will establish the honesty of these processes using the type system in Section 7.

Example 4.6 (Dishonest choice)**.** Recall from Example 3.4 the process:

$$Q \;=\; (y)\,\mathtt{tell}\downarrow_y (\mathtt{a?} \,\&\, \mathtt{b?}).\mathtt{do}_y\,\mathtt{a?}$$

We prove that Q is *not* ∞-honest (by Theorem 4.9, Q is not even 1-honest). Recall from Example 3.4 that the system $\mathsf{A}[(x)\,\mathtt{tell}\downarrow_x \mathtt{b!}.\mathtt{do}_x\,\mathtt{b!}] \mid \mathsf{B}[Q]$ may evolve (under the asynchronous semantics) to:

$$S \;=\; (s)\,S' \qquad \text{where} \quad S' \;=\; \mathsf{A}[\mathbf{0}] \;\mid\; \mathsf{B}[\mathtt{do}_s\,\mathtt{a?}] \;\mid\; s[\mathsf{A} : \mathtt{1}\ [] \mid \mathsf{B} : \mathtt{a?} \,\&\, \mathtt{b?}\ [\mathtt{b!}]]$$

We have that $S' \notin \mathrm{Rdy}_{\infty}^{\mathsf{B}@s}$, because none of the three disjunctive clauses in Definition 4.3 are safisfied: indeed, the first two clauses are trivially false, while the third one is false because $\mathrm{O}_{\infty}^{\mathsf{B}@s}(S') = \{\mathtt{b?}\} \not\subseteq \{\mathtt{a?}\} = S'\Downarrow_{\infty}^{\mathsf{B}@s}$. Therefore, A is *not* ∞-ready in S. □

Example 4.7 (Dishonest interleaving)**.** Recall from Example 3.5 the process

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

Recall from Example 3.5 that $\mathsf{A}[P]$ can be put in a system which evolves (under both the synchronous and asynchronous semantics) to $S \equiv (s)\,S'$, where:

$$S' \;=\; (t)\,(\mathsf{A}[\mathtt{do}_t\,\mathtt{a?}.\mathtt{do}_s\,\mathtt{b!}] \mid \mathsf{B}[\mathtt{do}_z\,\mathtt{b?}] \mid \mathsf{C}[\mathbf{0}] \mid t[\mathsf{A} : \mathtt{a?}\ [] \mid \mathsf{C} : \mathtt{a!}\ []] \mid s[\mathsf{A} : \mathtt{b!}\ [] \mid \mathsf{B} : \mathtt{b?}\ []])$$

We have that $S' \notin \mathrm{Rdy}_{\infty}^{\mathsf{A}@s}$, because none of the clauses in Definition 4.3 are safisfied: in particular, $\mathrm{O}_{\infty}^{\mathsf{A}@s}(S') = \{\mathtt{b!}\}$ and $S'\Downarrow_{\infty}^{\mathsf{A}@s} = \emptyset$, so their intersection is empty. Therefore, A is *not* ready in S', and so P is *not* ∞-honest (by Theorem 4.9, P is not even 1-honest). □

The following theorem states the undecidability of honesty (both under the synchronous and the asynchronous semantics).

Theorem 4.8. *The problem of deciding whether P is $\circ$-honest is not recursive.*

Proof. For $\circ = 1$, we can reduce the halting problem to checking dishonesty, similarly to [15]. The construction in [15] can be easily adapted also to the case $\circ = \infty$. □

Honesty under the synchronous semantics implies honesty under the asynchronous one (Theorem 4.9). As a consequence, any static analysis over-approximating synchronous honesty (like e.g., the type system in Section 7) also over-approximates asynchronous honesty.

Theorem 4.9. *If P is 1-honest, then P is ∞-honest.*

Proof. (Sketch). We prove the contrapositive: if P is ∞-dishonest then it is also 1-dishonest. Let S_∞ be such that P is ∞-dishonest in S_∞. By Definition 4.5, there exists a trace $\mathsf{A}[P] \mid S_\infty \to_\infty^* S'_\infty$ such that A is *not* ready in S'_∞. We can then craft a new context S_1 for which P is 1-dishonest. Dishonesty is achieved by allowing P to perform the same transitions as in the previous trace, so reaching a state where A is not ready. To make this

possible, we need to ensure that the prefixes fired in the asynchronous semantics do not become stuck in the synchronous one. In particular, a $\mathtt{do}_x\, a$ prefix can become stuck for two reasons: (1) when a is an output and the message queue is full, or (2) the session x cannot be established because the contracts fused in the original trace are compliant under $\bowtie_\infty$ but not under $\bowtie_1$. To address these issues, we craft S_1 so that: (1) it always keeps its (1-bounded) queues empty; (2) it advertises all the *syntactic duals* [3] of the contracts in P, instead of the contracts used in the asynchronous trace. Consequently, the context S_1 allows A to: (i) fire all the prefixes as in the asynchronous trace, and (ii) reach a state S_1' where A has the same process as in S_∞', and the contracts of A in all sessions are the same as in S_∞'. To make A not ready in S_1' (under the synchronous semantics), the context simply avoids to do anything (but keeping its queues empty). Since A is not ready in the asynchronous trace, it has some unfulfilled obligation in a session of S_∞'; hence, the same obligation is unfulfilled in S_1', because S_1' allows A to fire no more prefixes than S_∞'. □

Example 4.10. We now provide an example of the context construction sketched in the proof of Theorem 4.9. Consider the process:

$$P \;=\; (x)\,\mathtt{tell}\downarrow_x (\mathtt{a!}.\mathtt{b!}.\mathtt{c!}.\mathtt{d?}).\mathtt{do}_x\,\mathtt{a!}.\mathtt{do}_x\,\mathtt{b!}.\mathtt{do}_x\,\mathtt{c!}$$

We have that P is ∞-dishonest. To show that, consider e.g. the context:

$$Q \;=\; (y)\,\mathtt{tell}\downarrow_y (\mathtt{d!}.\mathtt{a?}.\mathtt{b?}.\mathtt{c?}).\mathtt{do}_y\,\mathtt{d!}$$

Under the asynchronous semantics, the system $\mathsf{A}[P] \mid \mathsf{B}[Q]$ may evolve to a system wherein A is not ready. In this computation, first a session between A and B is created, since their contracts are compliant under $\bowtie_\infty$. Then, A enqueues all the outputs $\mathtt{a!}, \mathtt{b!}, \mathtt{c!}$. Finally, B enqueues $\mathtt{d!}$. This enables the input action $\mathtt{d?}$ of A, but since A does not perform such obligation, we conclude that A is not ready.

Under the synchronous semantics, however, $\mathsf{A}[P] \mid \mathsf{B}[Q]$ does *not* evolve to a system where A is not ready. Indeed, in that case no session is created since the two contracts are no longer compliant: intuitively, making two contracts interact requires queues to be longer than one message. However, following the proof of Theorem 4.9, we can craft a context where A is not 1-honest, changing the process of B as follows:

$$Q' \;=\; (y)\,\mathtt{tell}\downarrow_y (\mathtt{a?}.\mathtt{b?}.\mathtt{c?}.\mathtt{d!}).\mathtt{do}_y\,\mathtt{a?}.\mathtt{do}_y\,\mathtt{b?}.\mathtt{do}_y\,\mathtt{c?}.\mathtt{do}_y\,\mathtt{d!}$$

Now the contract of A and the one of B are compliant under $\bowtie_1$, and the system $\mathsf{A}[P] \mid \mathsf{B}[Q']$ may evolve to a system wherein A is not ready. In this computation, after the session is created A sends all the outputs to B, which are immediately received by B (who always empties his queue). When B enqueues $\mathtt{d!}$, A is not ready to receive. □

Example 4.11. The converse of Theorem 4.9 does not hold, in general. E.g., the process:

$$(x, y)\,\mathtt{tell}\downarrow_x (\mathtt{a!}\,.\,\mathtt{a!}).\mathtt{do}_x\,\mathtt{a!}.\mathtt{tell}\downarrow_y \mathtt{b?}.\mathtt{do}_x\,\mathtt{a!}.\mathtt{do}_y\,\mathtt{b?}$$

is ∞-honest but not 1-honest. Indeed, under the asynchronous semantics all the outputs can always be fired, while this is not the case in the synchronous case. In particular, under the synchronous semantics, $\mathtt{do}_y\,\mathtt{b?}$ is reachable only if the process at the other endpoint of session x has read the first $\mathtt{a!}$, allowing the second output to be performed. □

$$\mathtt{a!}\,.\,C \oplus C' \xrightarrow{\mathtt{a!}}\!\!\!\!\!\rightarrow_\sharp \mathtt{ctx}\ \mathtt{a?}.\,C \qquad \mathtt{rdy}\ \mathtt{a?}.\,C \xrightarrow{\mathtt{a?}}\!\!\!\!\!\rightarrow_\sharp C$$

$$\mathtt{a?}\,.\,C \,\&\, C' \xrightarrow{\mathtt{ctx:a!}}\!\!\!\!\!\rightarrow_\sharp \mathtt{rdy}\ \mathtt{a?}.\,C \qquad \mathtt{ctx}\ \mathtt{a?}.\,C \xrightarrow{\mathtt{ctx:a?}}\!\!\!\!\!\rightarrow_\sharp C$$

FIGURE 4. Semantics of abstract contract configurations.

5. TYPES

In this section we introduce types for CO_2 processes. We will exploit them later on in Section 7 to devise a type system for honesty. Note that, by Theorem 4.9, we can focus on the synchronous semantics of contracts and systems, only: therefore, hereafter we will implicitly assume that the synchronicity parameter $\circ$ is always 1. If a process is typeable, then we will guarantee its honesty both under the synchronous and the asynchronous semantics.

Assume we want to verify the honesty of a participant A in a system S. We start by fixing an arbitrary channel u, and transforming S so to gather all the information involving u, while abstracting away the rest of the system. The result is a *pointed abstract system* (Section 5.2), i.e., a pair $(\Gamma, \mathcal{P})$, whose first component describes the possible sessions which might be established on u, while the component $\mathcal{P}$ (a *pointed abstract process*, see Definition 5.3) abstracts the behavior of A on channel u. In particular, the component Γ is either a set of contracts (when no session has been established yet), or otherwise an *abstract contract configuration* $\mathcal{C}$ (Section 5.1). Our type system will abstract processes and systems on *all* channels simultaneously, inferring a *type* f, which is a function mapping each channel u to a pointed abstract process $f(u) = \mathcal{P}$.

5.1. **Abstract contract configurations.** Let A be a participant, and let γ be a contract configuration (as in Definition 2.2): in the *abstract contract configuration* $\alpha_\mathsf{A}(\gamma)$ we maintain only the contract of A, while recording the message sent by A (if any), and abstracting the context. An abstract contract configuration can be either a contract C, a term $\mathtt{ctx}\ \mathtt{a?}.C$ representing that A has sent a message and the context has not read it yet, or a term $\mathtt{rdy}\ \mathtt{a?}.C$ representing that A has to read a message sent by the context.

Definition 5.1 (Abstract contract configurations)**.** The syntax of *abstract contract configurations* $\mathcal{C}, \mathcal{D}, \ldots$ is defined as follows, where C is a (concrete) contract from Definition 2.1:

$$\mathcal{C}, \mathcal{D} \ ::= \ C \ \mid \ \mathtt{ctx}\ \mathtt{a?}.C \ \mid \ \mathtt{rdy}\ \mathtt{a?}.C$$

For all participants A and contract configurations γ involving A, we define the *abstraction of* γ w.r.t. A, in symbols $\alpha_\mathsf{A}(\gamma)$, as follows (symmetric cases omitted):

$$\alpha_\mathsf{A}(\mathsf{A} : C\ [] \mid \mathsf{B} : D\ []) \ = \ C$$
$$\alpha_\mathsf{A}(\mathsf{A} : C\ [] \mid \mathsf{B} : D\ [\mathtt{a!}]) \ = \ \mathtt{ctx}\ \mathtt{a?}.C$$
$$\alpha_\mathsf{A}(\mathsf{A} : \mathtt{a?}.C \,\&\, C'\ [\mathtt{a!}] \mid \mathsf{B} : D\ []) \ = \ \mathtt{rdy}\ \mathtt{a?}.C$$

The LTS $\twoheadrightarrow_\sharp$ on abstract contract configurations is defined by the rules in Figure 4.

In an internal sum, A chooses a branch; in an external sum, the choice is made by the context (the $\mathtt{ctx}$ in the label indicates that the action is performed by the participant at the other endpoint of the session); in a $\mathtt{rdy}\ \mathtt{a?}.C$ the atom $\mathtt{a?}$ is fired.

The following lemma states that each transition of a contract configuration γ can be simulated by its abstraction $\alpha_{\mathsf{A}}(\gamma)$. This result was already established as Theorem 4.5 in [9], so we omit its proof here.

Lemma 5.2. *For all contract configurations γ, γ' such that $\alpha_{\mathsf{A}}(\gamma)$ is defined:*

$$\gamma \xrightarrow{\mathsf{A}:a}\!\!\!\!\rightarrow\, \gamma' \implies \alpha_{\mathsf{A}}(\gamma) \xrightarrow{a}\!\!\!\!\rightarrow_\sharp \alpha_{\mathsf{A}}(\gamma') \tag{5.2a}$$

$$\gamma \xrightarrow{\mathsf{B}:a}\!\!\!\!\rightarrow\, \gamma' \implies \alpha_{\mathsf{A}}(\gamma) \xrightarrow{\mathtt{ctx}:a}\!\!\!\!\rightarrow_\sharp \alpha_{\mathsf{A}}(\gamma') \qquad (\mathsf{B} \neq \mathsf{A}) \tag{5.2b}$$

5.2. **Pointed abstract systems.** Pointed abstract processes are Basic Parallel Processes (BPPs) [40] where prefixes are of the following kinds: atoms $\mathtt{a!}, \mathtt{b?}, \ldots$, nonblocking silent actions τ, possibly blocking silent actions $\tau_?$, and contract advertisement actions $\langle C \rangle$.

Definition 5.3 (Pointed abstract processes and systems)**.** The syntax of *pointed abstract processes* $\mathcal{P}$ and *prefixes* α is defined as follows:

$$\mathcal{P} \;::=\; \mathbf{0} \mid \alpha.\mathcal{P} \mid \mathcal{P} + \mathcal{P} \mid \mathcal{P} \mid \mathcal{P} \mid \mathit{rec}\, X.\mathcal{P} \mid X$$
$$\alpha \;::=\; \mathtt{a!} \mid \mathtt{a?} \mid \tau \mid \tau_? \mid \langle C \rangle$$

where sums and recursions are prefix-guarded. We denote with $\mathbb{P}$ the set of all pointed abstract processes. A *pointed abstract system* is a pair $(\Gamma, \mathcal{P})$, where Γ is either a set of (concrete) contracts (Definition 2.1) or an abstract contract configuration (Definition 5.1). The semantics of pointed abstract processes and systems is given in Figure 5.

The set Γ grows when the process $\mathcal{P}$ in $(\Gamma, \mathcal{P})$ advertises a contract D (rule [A-Tell1]). After one of the contracts in Γ has been stipulated, the set is reduced to a single contract C_i (rule [A-Fuse]), and further advertisements are neglected (rule [A-Tell2]). Rule [A-Do] models a $\mathtt{do}\ a$ action performed by $\mathcal{P}$, while rule [A-Ctx] models an action performed by the context (i.e., the participant at the other endpoint of the session). Some pointed abstract system transitions will be shown in Example 6.3.

5.3. **Types.** We will abstract the behavior of processes and systems as a *type* f, which describes the abstract behaviour on *all* channels as a pointed abstract process. Intuitively, we abstract the behaviour of P on a *free* channel u as $f(u)$, a pointed abstract process in $\mathbb{P}$. On all other channels (i.e., those not free in P), the process P has the same behaviour: therefore, we cumulatively abstract the behaviour of P on these channels as $f(*)$, where $*$ is a "dummy" element not in $\mathcal{N} \cup \mathcal{V}$. This makes it possible to limit f to a finite domain, namely the free channels of P and $*$ (as we will establish in Lemma 8.2).

Definition 5.4 (Type)**.** A *type* is a partial function $f : \mathcal{N} \cup \mathcal{V} \cup \{*\} \to \mathbb{P}$ from names/variables to pointed abstract processes, such that $\operatorname{dom} f$ is finite and comprises $*$.

Recall that we consider pointed abstract processes and systems up-to structural equivalence: consequently, also types are up-to structural equivalence, i.e. $f = f'$ whenever $\operatorname{dom} f = \operatorname{dom} f'$ and $f(u) \equiv f'(u)$ for all $u \in \operatorname{dom} f$.

We define below an operator which expands the domain of types. This will be exploited later on in Section 7.1 in our type system. For instance, to obtain the type of $P \mid Q$ we need to expand the domains of the types of P and Q to all the free channels of $P \mid Q$.

$$\frac{}{\alpha.\mathcal{P} \xrightarrow{\alpha}_\sharp \mathcal{P}}\ [\text{C-Pref}] \qquad \frac{\mathcal{P} \xrightarrow{\alpha}_\sharp \mathcal{P}'}{\mathcal{P} + \mathcal{P}'' \xrightarrow{\alpha}_\sharp \mathcal{P}'}\ [\text{C-SumL}] \qquad \frac{\mathcal{P} \xrightarrow{\alpha}_\sharp \mathcal{P}'}{\mathcal{P} \mid \mathcal{P}'' \xrightarrow{\alpha}_\sharp \mathcal{P}' \mid \mathcal{P}''}\ [\text{C-ParL}]$$

$$\frac{\mathcal{P}\{rec\ \mathcal{X}.\mathcal{P}/\mathcal{X}\} \xrightarrow{\alpha}_\sharp \mathcal{P}'}{rec\ \mathcal{X}.\mathcal{P} \xrightarrow{\alpha}_\sharp \mathcal{P}'}\ [\text{C-Rec}] \qquad \text{commutative monoidal laws for } \mid \text{ and } +$$

$$\frac{\mathcal{P} \xrightarrow{\langle D \rangle}_\sharp \mathcal{P}'}{(\{C_1, \ldots, C_n\}, \mathcal{P}) \xrightarrow{\tau}_\sharp (\{C_1, \ldots, C_n, D\}, \mathcal{P}')}\ [\text{A-Tell1}] \qquad \frac{\mathcal{P} \xrightarrow{\langle D \rangle}_\sharp \mathcal{P}'}{(\mathcal{C}, \mathcal{P}) \xrightarrow{\tau}_\sharp (\mathcal{C}, \mathcal{P}')}\ [\text{A-Tell2}]$$

$$\frac{i \in \{1, \ldots, n\}}{(\{C_1, \ldots, C_n\}, \mathcal{P}) \xrightarrow{\tau_?}_\sharp (C_i, \mathcal{P})}\ [\text{A-Fuse}] \qquad \frac{\mathcal{P} \xrightarrow{\alpha}_\sharp \mathcal{P}' \quad \alpha \in \{\tau, \tau_?\}}{(\Gamma, \mathcal{P}) \xrightarrow{\alpha}_\sharp (\Gamma, \mathcal{P}')}\ [\text{A-Tau}]$$

$$\frac{\mathcal{C} \xrightarrow{a}\!\!\!\!\rightarrow_\sharp \mathcal{C}' \quad \mathcal{P} \xrightarrow{a}_\sharp \mathcal{P}'}{(\mathcal{C}, \mathcal{P}) \xrightarrow{a}_\sharp (\mathcal{C}', \mathcal{P}')}\ [\text{A-Do}] \qquad \frac{\mathcal{C} \xrightarrow{\texttt{ctx}:a}\!\!\!\!\rightarrow_\sharp \mathcal{C}'}{(\mathcal{C}, \mathcal{P}) \xrightarrow{\tau_?}_\sharp (\mathcal{C}', \mathcal{P})}\ [\text{A-Ctx}]$$

FIGURE 5. Semantics of pointed abstract processes and systems.

Definition 5.5 (Domain expansion)**.** For all types f and for all $A \subseteq \mathcal{N} \cup \mathcal{V}$, we define the type $f{\uparrow_A}$ as: $f{\uparrow_A} \;=\; f\{A \setminus \operatorname{dom} f \mapsto f(*)\}$.

6. Abstract honesty

We now introduce a notion of *abstract* honesty for pointed abstract processes and systems. Unlike honesty for concrete processes, the abstract notion does not require a universal quantification over all contexts, which is key to prove its decidability (Theorem 6.6). We will exploit abstract honesty in our type system, when typing delimited processes. Intuitively, the typing of P constructs a pointed abstract process for each name/variable in P. The typing also checks the abstract honesty of these pointed abstract processes: the proof of type safety exploits these checks to guarantee that typeability implies honesty.

As done for concrete processes, we build abstract honesty over readiness. Intuitively, a pointed abstract system $(\mathcal{C}, \mathcal{P})$ is ready if it can weakly perform some action whenever $\mathcal{C}$ has enabled actions of A. When checking these weak transitions, we only consider those representing non-blocking steps, i.e. τ actions. By contrast, $\tau_?$ transitions represent potentially blocking actions, and so they are not followed, since there is no guarantee that they are enabled in the concrete system.

In order to be honest, a pointed abstract process must keep itself ready upon transitions, including the potentially blocking ones. 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 6.1 below formalises the abstract honesty for types (Item 4). This involves an auxiliary definition, i.e. the abstract honesty of pointed abstract processes (Item 3), which in turn involves defining the readiness and abstract honesty (respectively, Items 1 and 2) for pointed abstract systems. Note that, unlike the corresponding condition in Definition 4.5, Item 3 in Definition 6.1 does not universally quantify over all contexts.

Definition 6.1 (Abstract honesty)**.** We say that:

(1) $(\Gamma, \mathcal{P})$ is *ready* iff: $\Gamma \xrightarrow{a}\!\!\!\!\rightarrow_\sharp \implies \exists b\ .\ (\Gamma, \mathcal{P}) \xrightarrow{\tau}_\sharp{}^* \xrightarrow{b}_\sharp$
(2) $(\Gamma, \mathcal{P})$ is *honest* iff: $(\Gamma, \mathcal{P}) \rightarrow_\sharp^* (\mathcal{C}, \mathcal{P}') \implies (\mathcal{C}, \mathcal{P}')$ is ready
(3) $\mathcal{P}$ is *honest* iff: $(\emptyset, \mathcal{P})$ is honest
(4) f is *honest* iff: $f(u)$ is honest, for all $u \in \operatorname{dom} f$.

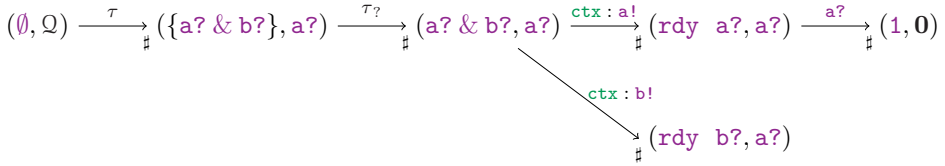
Example 6.2 (Honest/dishonest choice)**.** Let $\mathcal{P} = \langle \mathtt{a!} \oplus \mathtt{b!} \rangle . \mathtt{a!}$. The LTS of the pointed abstract system $(\emptyset, \mathcal{P})$ is:

$$(\emptyset, \mathcal{P}) \xrightarrow[\sharp]{\tau} (\{\mathtt{a!} \oplus \mathtt{b!}\}, \mathtt{a!}) \xrightarrow[\sharp]{\tau_?} (\mathtt{a!} \oplus \mathtt{b!}, \mathtt{a!}) \xrightarrow[\sharp]{\mathtt{a!}} (\mathtt{ctx\ a?}, \mathbf{0}) \xrightarrow[\sharp]{\tau_?} (\mathtt{1}, \mathbf{0})$$

To prove that $\mathcal{P}$ is honest, we check for readiness all the reducts in the LTS:

(1) $(\emptyset, \mathcal{P})$: nothing to check (no contracts advertised yet).
(2) $(\{\mathtt{a!} \oplus \mathtt{b!}\}, \mathtt{a!})$ nothing to check (no contracts stipulated yet).
(3) $(\mathtt{a!} \oplus \mathtt{b!}, \mathtt{a!})$ is ready, because $(\mathtt{a!} \oplus \mathtt{b!}, \mathtt{a!}) \xrightarrow{\mathtt{a!}}_\sharp$.
(4) $(\mathtt{ctx\ a?}.\mathtt{1}, \mathbf{0})$ and $(\mathtt{1}, \mathbf{0})$ are vacuously ready, since $\mathtt{1}$ and $\mathtt{ctx\ a?}.\mathtt{1}$ cannot take $\xrightarrow{a}\!\!\!\!\rightarrow_\sharp$-transitions (for any a).

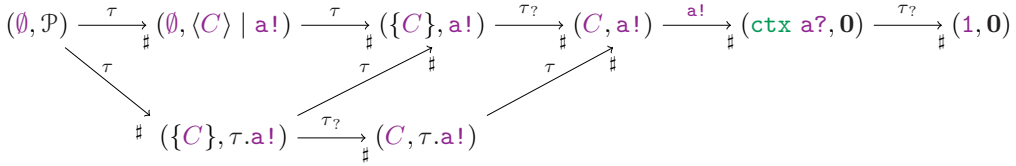
Now, let $\mathcal{Q} = \langle \mathtt{a?} \,\&\, \mathtt{b?} \rangle . \mathtt{a?}$. The LTS of $(\emptyset, \mathcal{Q})$ is the following:



In this case we have that the reduct $(\mathtt{rdy\ b?}, \mathtt{a?})$ is *not* ready: indeed, $\mathtt{rdy\ b?} \xrightarrow{\mathtt{b?}}\!\!\!\!\rightarrow_\sharp$, while $(\mathtt{rdy\ b?}, \mathtt{a?})$ cannot take $\xrightarrow{a}_\sharp$-transitions (for any a). Therefore, $\mathcal{Q}$ is *not* abstractly honest.

We anticipate that $\mathcal{P}$ and $\mathcal{Q}$ are the pointed abstract processes inferred by our type system, under the delimitations of the processes P and Q discussed in Example 3.4. Using the abstract honesty of $\mathcal{P}$ we will show in Example 7.3 that P is typeable, hence honest. Instead, the abstract dishonesty of $\mathcal{Q}$ will prevent us from typing Q — and rightly so, because we know from Example 4.6 that Q is dishonest. □

Example 6.3. Let $C = \mathtt{a!} \oplus \mathtt{b!}$, and let $\mathcal{P} = \langle C \rangle \mid \tau.\mathtt{a!}$. To determine whether $\mathcal{P}$ is honest, we check for readiness all the reducts of the pointed abstract system $(\emptyset, \mathcal{P})$:



We have that:

(1) $(\emptyset, \mathcal{P})$ and $(\emptyset, \langle C \rangle \mid \mathtt{a!})$: nothing to check (no contracts advertised yet).
(2) $(\{C\}, \tau.\mathtt{a!})$ and $(\{C\}, \mathtt{a!})$: nothing to check (no contracts stipulated yet).
(3) $(C, \tau.\mathtt{a!})$ is ready, because $(C, \tau.\mathtt{a!}) \xrightarrow{\tau}_\sharp \xrightarrow{\mathtt{a!}}_\sharp$.
(4) $(C, \mathtt{a!})$ is ready, because $(C, \mathtt{a!}) \xrightarrow{\mathtt{a!}}_\sharp$.
(5) $(\mathtt{1}, \mathbf{0})$ and $(\mathtt{ctx\ a?}, \mathbf{0})$ are vacuously ready, because $\mathtt{1}$ and $\mathtt{ctx\ a?}.\mathtt{1}$ cannot take $\xrightarrow{a}\!\!\!\!\rightarrow_\sharp$-transitions (for all a).

Summing up, we conclude that $\mathcal{P}$ is honest. □

Silent moves and contract advertisements of pointed abstract processes preserve honesty, while input/output moves may break honesty: for instance, in $\mathcal{P} = \mathtt{a!}.\langle \mathtt{b!} \rangle \xrightarrow{\mathtt{a!}}_\sharp \langle \mathtt{b!} \rangle = \mathcal{P}'$ we have that $\mathcal{P}$ is honest (because $(\emptyset, \mathcal{P})$ is stuck), while $\mathcal{P}'$ is dishonest.

Lemma 6.4. *For all* $\alpha \in \{\tau, \tau_?, \langle C' \rangle\}$*:* $\quad \mathcal{P} \xrightarrow{\alpha}_\sharp \mathcal{P}' \wedge \mathcal{P}$ *honest* $\implies \mathcal{P}'$ *honest*

Proof. See section A on page 37. □

The following lemma gives a compositional criterion to check the abstract readiness of a pointed abstract system $(\Gamma, \mathcal{P})$: indeed, it is enough to check the parallel components of $\mathcal{P}$ independently.

Lemma 6.5 (Abstract readiness and parallel composition)**.** *For all* Γ*,* $\mathcal{P}$*, and* $\mathcal{Q}$*:*

$$(\Gamma, \mathcal{P} \mid \mathcal{Q}) \text{ ready} \iff (\Gamma, \mathcal{P}) \text{ ready} \vee (\Gamma, \mathcal{Q}) \text{ ready}$$

Proof. See section A on page 37. □

Note that, unlike readiness, honesty is *not* compositional. E.g., the direction $\Rightarrow$ of Lemma 6.5 would be false since $(\mathtt{a!.b!},\ \mathtt{a!} \mid \mathtt{b!})$ is honest, while neither $(\mathtt{a!.b!},\ \mathtt{a!})$ nor $(\mathtt{a!.b!},\ \mathtt{b!})$ are such. The direction $\Leftarrow$ would be false since $(\mathtt{a!.b!},\ \mathtt{a!.b!})$ is honest, while $(\mathtt{a!.b!},\ \mathtt{a!} \mid \mathtt{a!.b!})$ is not.

Theorem 6.6 below establishes one of our main results: checking the honesty of a type f is decidable. Since abstract honesty will be used as a side condition in our typing rules for CO_2 processes, this result is crucial to obtain decidability for both type checking and inference (Theorem 8.6). Our proof reduces abstract honesty to submarking reachability in Petri nets, which is decidable [31, 39]. To define the reduction, we first map a pointed abstract system $(\emptyset, \mathcal{P})$ into a Petri Net which preserves its semantics. Roughly, all the reducts of $\mathcal{P}$ are parallel compositions of processes taken, possibly more than once, from a finite set of subterms of $\mathcal{P}$. Hence, we can associate a place to each such subterm, and use the tokens to count their multiplicity. Further, there are only finitely many states for the Γ component, so we can associate a place to each of them, and use a single token to represent the current Γ. The correctness of our reduction relies on Lemma 6.5, which implies that readiness can be established by inspecting at most one token for each place.

Theorem 6.6 (Decidability of abstract honesty)**.** *Abstract honesty of pointed abstract processes is decidable.*

Proof. To decide whether $\mathcal{P}$ is (abstractly) honest, by Item 3 of Definition 6.1 we need to decide whether the pointed abstract system $(\emptyset, \mathcal{P})$ is honest. We define the sets:

- $\{\mathcal{P}_j\}_j$, comprising the closed nonempty sums which are subterms of some unfolding of $\mathcal{P}$. We consider each $\mathcal{P}_j$ up-to $\equiv$ and unfolding of recursion.
- $\Gamma(\mathcal{P})$, comprising elements of two kinds: (i) sets $\{C_1, \ldots, C_n\}$ of contracts occurring in $\mathcal{P}$, and (ii) the reducts of each C_j occurring in $\mathcal{P}$, according to the semantics in Figure 4.

Every pointed abstract system reachable from $(\emptyset, \mathcal{P})$ has the form $(\Gamma_i, \mathcal{Q}_i)$, where $\Gamma_i \in \Gamma(\mathcal{P})$, and $\mathcal{Q}_i$ can be uniquely written (up to $\equiv$ and unfolding) as the parallel composition of some terms in $\{\mathcal{P}_j\}_j$, possibly taken multiple times.

We now define a Petri net $N = (P, T)$. The places P comprise all the elements in $\Gamma(\mathcal{P})$ (called Γ*-places*), and all the terms in $\{\mathcal{P}_j\}_j$ (called $\mathcal{P}$*-places*). Intuitively, we want the

reachable markings of N to have exactly one token in a Γ-place (while all the other Γ-places have none); instead, $\mathcal{P}$-places can contain any number of tokens. The idea is that the single token in the Γ-places corresponds to the first component in $(\Gamma_i, \mathcal{Q}_i)$, while the tokens in $\mathcal{P}$-places determine the component $\mathcal{Q}_i$. More precisely, the number of tokens in place $\mathcal{P}_j$ is the number of terms $\mathcal{P}_j$ in the parallel decomposition of $\mathcal{Q}_i$ (see [1]). The initial marking of N is the one corresponding to $(\emptyset, \mathcal{P})$. The transitions of N reflect the semantics of pointed abstract systems in Figure 5. For instance, we encode rule [A-Do] in the Petri net by moving the single token in the Γ-places from the place $\mathcal{C}$ to $\mathcal{C}'$, and simulating the firing of action a as follows. First, we find the parallel component $\sum_k a_k.\mathcal{R}_k$ of $\mathcal{P}$ in the rule with the prefix $a_k = a$ to be fired. To rewrite one copy of that sum with $\mathcal{R}_k$, we consume one token in the $\mathcal{P}$-place associated to the sum, and we produce tokens for $\mathcal{R}_k$ in all the places corresponding to its parallel decomposition. This construction extends the one in [29], which shows an isomorphism between the transition system of a BPP and the reachability graph of its Petri net. Summing up, the firing sequences of N correspond to the computations of $(\emptyset, \mathcal{P})$.

Now, note that the set $\Gamma(\mathcal{P})$ is finite, because its elements of the form $\{C_1, \ldots, C_n\}$ contain only contracts syntactically occurring in $\mathcal{P}$, and its elements of the form $\mathcal{C}'$ are reducts of some C_j, so they are finite because the semantics in Figure 4 is finite-state. Further, also the set $\{\mathcal{P}_j\}_j$ is finite, because its elements are considered up-to. Therefore, N is a finite Petri net.

We reduce the problem of checking dishonesty of $(\emptyset, \mathcal{P})$ to the *submarking reachability problem* in N, which is decidable for finite Petri nets [31, 39]. A submarking is a mapping from a *subset* of the places P' to $\mathbb{N}$, which partially specifies a marking of the whole net. The submarking reachability problem asks, given a submarking $m' : P' \to \mathbb{N}$, to establish whether or not some marking m is reachable such that $m(p) = m'(p)$ for all $p \in P'$.

Say that a marking is *ready* if it corresponds to a ready pointed abstract system. For every multiset X of places, we denote with χ_X the marking which associates each place to the corresponding number of occurrences in X. Then, for all Γ we define the submarking M_Γ as follows:

$$M_\Gamma(p) = \begin{cases} 1 & \text{if } p = \Gamma \\ 0 & \text{if } p \text{ is a } \mathcal{P}\text{-place and } \chi_{\{\Gamma,p\}} \text{ is ready} \\ \text{undefined} & \text{otherwise} \end{cases} \tag{6.6a}$$

We now prove that, for any reachable marking m:

$$m \text{ non ready} \iff \exists a, \Gamma : \Gamma \xrightarrow{a}\!\!\!\!\!\rightarrow_\sharp \ \wedge\ M_\Gamma \text{ submarking of } m \tag{6.6b}$$

For the $\Rightarrow$ direction, let Γ_i be the single Γ-place with one token. Since m is not ready, we must have $\Gamma_i \xrightarrow{a}\!\!\!\!\!\rightarrow_\sharp$ for some a. To obtain the thesis, choose $\Gamma = \Gamma_i$, and assume by contradiction that M_Γ is *not* a submarking of m, i.e. $M_\Gamma(p) \neq m(p)$ for some p in the domain of M_Γ. We cannot have that $p = \Gamma$, since in that case by Equation (6.6a) it must be $M_\Gamma(p) = m(p) = 1$. Hence, p must be a $\mathcal{P}$-place for which $\chi_{\{\Gamma,p\}}$ is ready. Further, since $M_\Gamma(p) = 0 \neq m(p)$, we have that $m(p)$ has at least as many $\mathcal{P}$-tokens as the ready marking $\chi_{\{\Gamma,p\}}$. By Lemma 6.5 (adapted to Petri nets in the natural way), adding more $\mathcal{P}$-tokens to a ready marking cannot make it non ready, hence m is ready — contradiction.

For the $\Leftarrow$ direction of Equation (6.6b), assume that $\Gamma \xrightarrow{a}\!\!\!\!\!\rightarrow_\sharp$ and M_Γ is a submarking of m. By Equation (6.6a), $m(\Gamma) = 1$. By contradiction, assume that m is ready. Then, by (the adaptation of) Lemma 6.5, one can remove all the $\mathcal{P}$-tokens from m but one, while preserving its readiness; so, assume that such token is in place p (this implies that $m(p) > 0$).

The marking obtained in this way is $\chi_{\{\Gamma,p\}}$, and since it is ready, Equation (6.6a) gives $M_\Gamma(p) = 0$. Since M_Γ is a submarking of m, this implies that $m(p) = 0$ — contradiction.

To conclude, we now exploit the decidability of the submarking reachability problem on Petri nets to decide abstract dishonesty. First, we equivalently rephrase dishonesty in terms of submarking reachability:

$$\begin{aligned}
(\emptyset,\mathcal{P}) \text{ dishonest} &\iff \exists (\mathcal{C}',\mathcal{P}') \text{ non ready} : (\emptyset,\mathcal{P}) \to_\sharp^* (\mathcal{C}',\mathcal{P}') && \text{(by Definition 6.1)}\\
&\iff \exists m \text{ reachable in } N \text{ and non ready} && \text{(by construction of } N)\\
&\iff \exists m \text{ reachable in } N \text{ and} \\
&\qquad \exists a, \Gamma : \Gamma \overset{a}{\twoheadrightarrow}_\sharp \wedge M_\Gamma \text{ submarking of } m && \text{(by Equation (6.6b))}\\
&\iff \exists a, \Gamma : \Gamma \overset{a}{\twoheadrightarrow}_\sharp \wedge M_\Gamma \text{ reachable submarking} && \text{(6.6c)}
\end{aligned}$$

To conclude, we show how to verify the last formulation of dishonesty (Equation (6.6c)). To this purpose, note that we can effectively and finitely enumerate all the possible Γ and a. In each case we can easily check whether $\Gamma \overset{a}{\twoheadrightarrow}_\sharp$. Further, we can effectively construct the submarking M_Γ following Equation (6.6a). The only non-trivial task is checking whether $\chi_{\{\Gamma,p\}}$ is ready. According to Definition 6.1 (adapted to Petri nets), this just requires to check whether such marking has some weakly ready action, i.e., if it can fire some action b after a finite sequence of τ actions. This can be reduced once again to a submarking reachability problem: more precisely, we start by labelling the transitions of N as for the moves of pointed abstract systems; then we remove from N all the $\tau_?$ transitions, while making all the non-τ transitions (i.e., the ready actions b) fill a special place. At this point, it suffices to check whether the special place can eventually become nonempty, which is a submarking reachability problem. □

7. A type system for honesty

We now introduce a type system for CO_2. Type inference is decidable (Theorem 8.6), and type safety guarantees that typeable processes are honest (Theorem 9.9).

7.1. **Process typing.** Our type system associates types to CO_2 processes. Basically, in Definition 7.1 we abstract the CO_2 prefixes as actions of pointed abstract processes (Definition 5.3). To give some intuition, assume we want to abstract the behaviour of a process P over a channel u, and P has a prefix acting on some channel v. We have two cases.

- If $v \neq u$, we abstract the prefix as a τ when it is statically known to be unblocking, otherwise we abstract it as $\tau_?$. For instance, if P has a prefix $\mathtt{tell}{\downarrow_v}\, C$, which requires no synchronisation with the context, then we abstract it with a τ action (unblocking). Instead, if P has a prefix $\mathtt{do}_v\, a$, which can only be fired with a suitable configuration in session v, we abstract it as $\tau_?$ (potentially blocking).
- If $v = u$, the abstraction is more precise, recording the effect of the prefix. For instance, we abstract the prefix $\mathtt{tell}{\downarrow_v}\, C$ as $\langle C \rangle$, while we abstract $\mathtt{do}_v\, a$ as a.

$$\frac{}{\vdash \mathbf{0}\colon \lambda u\,.\,\mathit{if}\ u = *\ \mathit{then}\ \mathbf{0}\ \mathit{else}\ \bot}\ [\text{T-Nil}] \qquad \frac{\vdash P\colon f \quad \vdash Q\colon g \quad A = \operatorname{dom} f \cup \operatorname{dom} g}{\vdash P \mid Q\colon \lambda u\,.\, f{\uparrow_A}(u) \mid g{\uparrow_A}(u)}\ [\text{T-Par}]$$

$$\frac{\forall i \in I \neq \emptyset.\, \vdash P_i\colon f_i \quad A = \bigcup_{i\in I}(\operatorname{dom} f_i \cup \operatorname{fnv}(\pi_i))}{\vdash \sum_{i\in I} \pi_i.P_i\colon \lambda u\,.\,\sum_{i\in I}[\pi_i]_u.f_i{\uparrow_A}(u)}\ [\text{T-Sum}] \qquad \frac{\vdash P\colon f \quad f{\uparrow_{\{u\}}}(u)\ \text{honest}}{\vdash (u)P\colon f\{u \mapsto \bot\}}\ [\text{T-Del}]$$

$$\frac{\vdash P\colon f}{\vdash (\mathtt{rec}\ \mathrm{X}().\ P)()\colon \lambda u.\, \mathit{rec}\ @\mathrm{X}.f(u)}\ [\text{T-Rec}] \qquad \frac{}{\vdash \mathrm{X}()\colon \lambda u\,.\,\mathit{if}\ u = *\ \mathit{then}\ @\mathrm{X}\ \mathit{else}\ \bot}\ [\text{T-Var}]$$

FIGURE 6. Typing rules for processes.

Definition 7.1 (Prefix abstraction)**.** For all $u \in \mathcal{N} \cup \mathcal{V} \cup \{*\}$, we define the mapping $[\cdot]_u$ from CO_2 prefixes to prefixes of pointed abstract processes as follows:

$$[\tau]_u = \tau \qquad [\tau_?]_u = \tau_? \qquad [\mathtt{fuse}]_u = \tau_?$$

$$[\mathtt{tell}{\downarrow_v}\, C]_u = \begin{cases} \langle C \rangle & \text{if } v = u \\ \tau & \text{otherwise} \end{cases} \qquad [\mathtt{do}_v\, a]_u = \begin{cases} a & \text{if } v = u \\ \tau_? & \text{otherwise} \end{cases}$$

Our type system extends prefix abstraction to the whole process, on *all* channels. To properly deal with delimited channels, we additionally check honesty of the pointed abstract processes associated to them. Typing judgments for processes have the form $\vdash P\colon f$.

Definition 7.2 (Process typing)**.** Typing rules for processes are given in Figure 6. We assume an injective function @ which associates a recursion variable $\mathcal{X}$ to each constant $\mathrm{X}()$. We say that f is *inhabited* whenever $\vdash P\colon f$, for some P.

We now comment on the rules in Figure 6. In all the rules we take care of making the types defined only on the channels that can be observed, and on the dummy channel $*$, which represents the other channels (see Lemma 8.2): technically, in each judgement $\vdash P\colon f$ we ensure that $\operatorname{dom} f = \operatorname{fnv}(P) \cup \{*\}$. To this purpose we often suitably extend, in the conclusions of the rules, the domain of the types mentioned in the premises; this is done through the operator $\cdot{\uparrow_{\cdot}}$ introduced in Definition 5.5. Rule [T-Nil] types the empty process with a map assigning the type $\mathbf{0}$ to the dummy channel $*$ (and undefined on the other channels, since $\operatorname{fnv}(\mathbf{0}) = \emptyset$). The type of a parallel composition is the pointwise parallel composition of the component types (rule [T-Par]). Rule [T-Sum] types non-empty summations as (abstract) summations, by abstracting the prefixes according to Definition 7.1.

Rule [T-Del] types delimited processes $(u)P$: since the channel u is bound in $(u)P$ in the conclusion we remove it from the domain of the type. In the rule premise, the pointed abstract process $f(u)$ (with the domain expanded to include u, if needed), abstracts the *whole* behaviour of the participant under observation at session u. At this point, checking the (abstract) honesty of $f(u)$ guarantees that P respects its obligations at session u. Note that omitting or delaying the honesty check of $f(u)$ at this point would allow a dishonest behaviour to be typeable: for instance, we would incorrectly type the dishonest process $(u)\mathtt{tell}{\downarrow_u}\ \mathtt{a!}.\mathbf{0}$; if we delay the the honesty check after the delimitation we would be able to type the process, since the resulting type f' is only defined on $*$, and $f'(*) = \tau$, which is abstractly honest. Note that verifying $f(u)$ honest is decidable by Theorem 6.6: we exploit this fact to prove that type inference is decidable as well.

Finally, rules [T-Rec] and [T-Var] deal with recursive processes and process variables. Note that only recursive calls without parameters are typeable.

Example 7.3 (Honest choice)**.** Recall from Example 3.4 the process:

$$P \;=\; (x)P' \qquad \text{where} \quad P' \;=\; \mathtt{tell}\downarrow_x (\mathtt{a!} \oplus \mathtt{b!}).\, \mathtt{do}_x\, \mathtt{a!}$$

We can type P as follows, where $f_{\mathbf{0}} = \lambda u\,.\, \mathit{if}\ u = *\ \mathit{then}\ \mathbf{0}\ \mathit{else}\ \bot = \{* \mapsto \mathbf{0}\}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \mathbf{0} \colon f_{\mathbf{0}}}\,[\text{T-Nil}]}{\vdash \mathtt{do}_x\, \mathtt{a!}.\mathbf{0} \colon \lambda u.\, [\mathtt{do}_x\, \mathtt{a!}]_u.\, f_{\mathbf{0}}{\uparrow_{\{x,*\}}}(u) \;=\; f''}\,[\text{T-Sum}]}{\vdash P' \colon \lambda u.\, [\mathtt{tell}\downarrow_x (\mathtt{a!} \oplus \mathtt{b!})]_u.\, f''(u) \;=\; f'}\,[\text{T-Sum}] \qquad f'(x)\ \text{honest}}{\vdash P \colon f'\{x \mapsto \bot\} \;=\; f}\,[\text{T-Del}]$$

where $f' = \{x \mapsto \langle \mathtt{a!} \oplus \mathtt{b!}\rangle.\mathtt{a!}.\mathbf{0},\ * \mapsto \tau.\tau_?.\mathbf{0}\}$. Note that the premise of rule [T-Del] holds, because $\langle \mathtt{a!} \oplus \mathtt{b!}\rangle.\mathtt{a!}.\mathbf{0}$ is abstractly honest, as shown in Example 6.2. Since P has no free variables, its type f has domain $\{*\}$, and we have $f(*) = f'(*) = \tau.\tau_?.\mathbf{0}$. From the typeability of P, type safety (Theorem 9.9) will allow us to deduce that P is honest. □

Example 7.4 (Dishonest choice)**.** Recall from Example 3.4 the process:

$$Q \;=\; (y)\, Q' \qquad \text{where} \quad Q' \;=\; \mathtt{tell}\downarrow_y (\mathtt{a?} \,\&\, \mathtt{b?}).\mathtt{do}_y\, \mathtt{a?}$$

We show that Q is *not* typeable. The only possible typing derivation for Q would have the following form, where $g' = \{y \mapsto \langle \mathtt{a?} \,\&\, \mathtt{b?}\rangle.\mathtt{a?}.\mathbf{0},\ * \mapsto \tau.\tau_?.\mathbf{0}\}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \mathbf{0} \colon f_{\mathbf{0}}}\,[\text{T-Nil}]}{\vdash \mathtt{do}_y\, \mathtt{a?}.\mathbf{0} \colon \lambda u.\, [\mathtt{do}_y\, \mathtt{a?}]_u.\, f_{\mathbf{0}}{\uparrow_{\{y,*\}}}(u) \;=\; g''}\,[\text{T-Sum}]}{\vdash Q' \colon \lambda u.\, [\mathtt{tell}\downarrow_y (\mathtt{a?} \,\&\, \mathtt{b?})]_u.\, g''(u) \;=\; g'}\,[\text{T-Sum}] \qquad g'(y)\ \text{honest}}{\vdash Q \colon g'\{y \mapsto \bot\} \;=\; g}\,[\text{T-Del}]$$

The rightmost premise in rule [T-Del] is false, because $g'(y) = \langle \mathtt{a?} \,\&\, \mathtt{b?}\rangle.\mathtt{a?}.\mathbf{0}$ is *not* abstractly honest, as shown in Example 6.2. Therefore, the (dishonest) process Q is not typeable. □

Example 7.5 (Dishonest interleaving)**.** Recall from Example 3.5 the process:

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

The only possible typing derivation for P would require the use of rule [T-Del] to close the delimitation on y. The premise of such rule should verify the abstract honesty of $\tau.\langle \mathtt{b!}\rangle.\tau_?.\mathtt{b!}$ — which is *not* abstractly honest, because the required $\mathtt{b!}$ action is potentially blocked by the prefix $\tau_?$. Therefore, P is not typeable. □

The following example shows an honest process which interleaves its actions in two sessions, while enjoying typeability. As such, it respects its obligations in both sessions.

Example 7.6 (Honest interleaving)**.** Consider the process:

$$R' \;=\; \mathtt{tell}\downarrow_x \mathtt{a?}.\,\mathtt{tell}\downarrow_y (\mathtt{b!} \oplus \mathtt{c!}).\,\big(\mathtt{do}_x\, \mathtt{a?}.\mathtt{do}_y\, \mathtt{b!} + \tau.(\mathtt{do}_y\, \mathtt{c!} \mid \mathtt{do}_x\, \mathtt{a?})\big)$$

The process $R = (x, y)\, R'$ can be seen as an honest variant of the process P in Example 3.5. The key difference w.r.t. P is that, in the internal choice at session y, R adds the option $\mathtt{c!}$, playing the role of an "abort" message. After advertising the two contracts, the implementation of R proceeds as follows: (i) if R receives $\mathtt{a}$ in session x, it will send $\mathtt{b}$ in session y; (ii) otherwise, if the τ prefix is fired (modelling e.g., a timeout), then R will send the abort message $\mathtt{c}$ in y, while staying ready to receive $\mathtt{a}$ in x.

We can type the process R' with the following type f (where we omit $f(*)$):

$$\{\, x \mapsto \langle \mathtt{a?} \rangle.\tau.\big(\mathtt{a?}.\tau_? + \tau.(\tau_? \mid \mathtt{a?})\big),\;\; y \mapsto \tau.\langle \mathtt{b!} \oplus \mathtt{c!} \rangle.\big(\tau_?.\mathtt{b!} + \tau.(\mathtt{c!} \mid \tau_?)\big) \,\}$$

Since both $f(x)$ and $f(y)$ are abstractly honest, then we can apply twice rule [T-Del], obtaining that R is typeable, hence honest by type safety (Theorem 9.9). □

Example 7.7 (Online store)**.** Recall from Example 3.6 the dishonest specification P of the online store in Section 1. By defining the processes R_i within P as follows:

$$\begin{aligned}
R_1 &= \tau\,.\,(\mathtt{do}_x\,\mathtt{refund!} \mid \mathtt{do}_y\,\mathtt{shipB?}) \\
R_2 &= \tau\,.\,(\mathtt{do}_x\,\mathtt{refund!} \mid \mathtt{do}_y\,\mathtt{quit!}) \\
R_3 &= \mathtt{do}_x\,\mathtt{quit?}.\mathtt{do}_y\,\mathtt{quit!} \;+\; \tau.((\mathtt{do}_x\,\mathtt{pay3E?}.\mathtt{do}_x\,\mathtt{refund!} + \mathtt{do}_x\,\mathtt{quit?}) \mid \mathtt{do}_y\,\mathtt{quit!}) \\
R_4 &= \tau\,.\,(\mathtt{do}_x\,\mathtt{abort!} \mid \mathtt{do}_y\,\mathtt{buyB!}.\mathtt{do}_y\,\mathtt{quit!})
\end{aligned}$$

we obtain an honest (and typeable) variant of the online store. Intuitively, P in Example 3.6 is dishonest in the contexts where the counterpart in one of the two sessions stops to cooperate. That makes P stuck waiting for a message from that session, and no longer interacting in the other session, hence becoming not ready there. The processes R_i above deal with these situations, by performing the needed compensations in order to make the store ready. For instance, R_4 deals with the case where the session y with the distributor is not established (or delayed): in such case, the action $\mathtt{buyB!}$ at y cannot be fired, but still the store must carry on the interaction with the buyer at x. To this purpose, R_4 starts with a τ prefix, modelling a timeout, and then performs $\mathtt{abort!}$ on x. The compensation actions at y are needed in case the session with the distributor is established after the timeout. □

In the following example we type a process which recursively advertises contracts and respects its obligations in all sessions.

Example 7.8. Let $P' = \mathtt{tell}{\downarrow_x}\,\mathtt{a!}\,.\,\mathrm{Y}() \mid \mathtt{do}_x\,\mathtt{a!}$. We can type the process P' as follows, where $f_{\mathrm{Y}} = \lambda u\,.\,\mathit{if}\;\; u = *\;\; \mathit{then}\;\; @\mathrm{Y}\;\; \mathit{else}\;\; \bot$, and $f_{\mathbf{0}}$, f'' are as in the previous example:

$$\dfrac{\dfrac{\dfrac{}{\vdash \mathrm{Y}\colon f_{\mathrm{Y}}}\,[\text{T-Var}]}{\vdash \mathtt{tell}{\downarrow_x}\,\mathtt{a!}\,.\,\mathrm{Y}()\colon \lambda u\,.\,[\mathtt{tell}{\downarrow_x}\,\mathtt{a!}]_u.f_{\mathrm{Y}}{\uparrow_{\{x,*\}}}(u)}\,[\text{T-Sum}] \qquad \dfrac{\dfrac{}{\vdash \mathbf{0}\colon f_{\mathbf{0}}}\,[\text{T-Nil}]}{\vdash \mathtt{do}_x\,\mathtt{a!}\colon f''}\,[\text{T-Sum}]}{\vdash P'\colon \lambda u\,.\,\mathit{if}\;\; u \in \{x,*\}\;\; \mathit{then}\;\; ([\mathtt{tell}{\downarrow_x}\,\mathtt{a!}]_u.@\mathrm{Y} \mid [\mathtt{do}_x\,\mathtt{a!}]_u)\;\; \mathit{else}\;\; \bot \;=\; f'}\,[\text{T-Par}]$$

Note that $f'(x) = \langle \mathtt{a!} \rangle.@\mathrm{Y} \mid \mathtt{a!}$ is abstractly honest. Therefore, we can type the recursive process $P = (\mathtt{rec}\;\mathrm{Y}().\,(x)P')()$ as follows:

$$\dfrac{\dfrac{\vdash P'\colon f' \qquad f'(x)\ \text{honest}}{\vdash (x)P'\colon f'\{x \mapsto \bot\} \;=\; f}\,[\text{T-Del}]}{\vdash (\mathtt{rec}\;\mathrm{Y}().\;P)()\colon \lambda u.\,\mathit{rec}\;@\mathrm{Y}.f(u)}\,[\text{T-Rec}]$$

We note that the process P is infinite-state, because of the delimitation and the parallel under recursion. □

$$\frac{\vdash P : f}{\vdash_{\mathsf{A}} \mathsf{A}[P] : f}\ [\text{T-SA}] \qquad \frac{\vdash_{\mathsf{A}} S : f \quad f{\uparrow_{\{u\}}}(u)\ \text{honest}}{\vdash_{\mathsf{A}} (u)S : f\{u \mapsto \bot\}}\ [\text{T-SDel2}] \qquad \frac{\vdash_{\mathsf{A}} S : f \quad \vdash_{\mathsf{A}} S' \rhd f}{\vdash_{\mathsf{A}} S \mid S' : f}\ [\text{T-SPar2}]$$

$$\frac{}{\vdash_{\mathsf{A}} \mathbf{0} \rhd f}\ [\text{T-SAFree0}] \qquad \frac{\mathsf{B} \neq \mathsf{A} \quad \mathrm{fv}(P) \cap \mathrm{dom}\, f = \emptyset}{\vdash_{\mathsf{A}} \mathsf{B}[P] \rhd f}\ [\text{T-SAFree1}] \qquad \frac{\mathsf{B} \neq \mathsf{A} \quad f(x) = \bot}{\vdash_{\mathsf{A}} \{{\downarrow_x}\ C\}_{\mathsf{B}} \rhd f}\ [\text{T-SAFree2}]$$

$$\frac{s[\gamma]\ \mathsf{A}\text{-free} \quad f(s) = \bot}{\vdash_{\mathsf{A}} s[\gamma] \rhd f}\ [\text{T-SAFree3}] \qquad \frac{(C, f{\uparrow_{\{x\}}}(x))\ \text{honest}}{\vdash_{\mathsf{A}} \{{\downarrow_x}\ C\}_{\mathsf{A}} \rhd f}\ [\text{T-SFz1}] \qquad \frac{(\alpha_{\mathsf{A}}(\gamma), f{\uparrow_{\{s\}}}(s))\ \text{honest}}{\vdash_{\mathsf{A}} s[\gamma] \rhd f}\ [\text{T-SFuse}]$$

$$\frac{}{\vdash_{\mathsf{A}} \{{\downarrow_s}\ C\}_{\mathsf{B}} \rhd f}\ [\text{T-SFzS}] \qquad \frac{\vdash_{\mathsf{A}} S \rhd f\{u \mapsto \bot\}}{\vdash_{\mathsf{A}} (u)S \rhd f}\ [\text{T-SDel1}] \qquad \frac{\vdash_{\mathsf{A}} S \rhd f \quad \vdash_{\mathsf{A}} S' \rhd f}{\vdash_{\mathsf{A}} S \mid S' \rhd f}\ [\text{T-SPar1}]$$

FIGURE 7. Typing rules for systems. Symmetric rules w.r.t. $\mid$ for [T-SFuse] and [T-SPar2] are omitted.

Example 7.8 shows a case where the analysis technique proposed in this paper is more precise than the one in [9]. Indeed, since P is typeable (and type inference is decidable, Theorem 8.6), then by Theorem 9.9 our analysis technique effectively proves that P is honest. Instead, the model checking algorithm in [9] would diverge on P, because it can only handle finite-state processes. The technique in [9] could be extended by exploiting standard model-checking algorithms for Petri nets (such as [28]), so to be capable of verifying the honesty of some infinite-state processes. However, such an extension would still fail to handle the process P of Example 7.8, because the delimitation under the recursion makes P not expressible as a Petri net.

7.2. **System typing.** Observe that the type system for processes is enough to guarantee whether a participant is honest. However, in order to establish subject reduction we have to consider system transitions (because the semantics of a process depends on the system wherein it is run), and so we need to extend our type system to CO_2 systems.

Type judgments for systems are of two kinds, $\vdash_{\mathsf{A}} :$ and $\vdash_{\mathsf{A}} \rhd$. 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 every contract of A in the context S has indeed been advertised by A.

Definition 7.9 (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 Figure 7.

The first three rules in Figure 7 deal with the typing judgements $\vdash_{\mathsf{A}} :$ for systems. Rule [T-SA] extends to $\mathsf{A}[P]$ a typing of P. Rule [T-SDel2] is similar to the rule [T-Del] for processes. Rule [T-SPar2] types as f the parallel composition of two systems, one of which must contain A and be typeable with f (under the $\vdash_{\mathsf{A}} :$ typing), while the rest of the system must be compatible with f (using the $\vdash_{\mathsf{A}} \rhd$ typing).

All the other rules define the compatibility judgements $\vdash_{\mathsf{A}} \rhd$. For instance, rules [T-SAFree*] tell that A-free systems are compatible with all types f undefined on the channels

in the conclusion of the rules. For instance, in rule [T-SAFree1] we forbid B to use the free variables of A (i.e., those in $\operatorname{dom} f$), to avoid potentially harmful name instantiations. Similar preconditions are required by rules [T-SAFree2] and [T-SAFree3]. Rule [T-SFz1] states that a latent contract $\{\downarrow_x C\}_{\mathsf{A}}$ is typeable with $\vdash_{\mathsf{A}} \triangleright$ only when $f(x)$ "realizes" such contract. Rule [T-SFused] is similar, except that a contract of A occurs inside a session; also in this case, A must realize her contract. Rule [T-SFzS] deals with garbage latent contracts $\{\downarrow_s C\}_{\mathsf{A}}$, which cannot be fused in any sessions (because s is already a session name, so it cannot be instantiated). Rule [T-SDel1] is symmetrical to [T-SDel2]: for system $(u)S$ to be compatible with f, the use of u in S has to be decoupled from the abstraction $f(u)$. This is necessary to prevent confusion between the channel u occurring bound in $(u)S$ and *another* channel named u occurring in the process of A, hence found in $\operatorname{dom} f$. Although these two channels have the same name, their scope is different, so they must be treated as distinct. For this reason, we need to check S to be compatible to a type obtained by ignoring in f the presence of u (see Example 7.10 below). The last rule [T-SPar1] is straightforward.

Example 7.10 (Honest choice)**.** Recall the process $P' = \mathtt{tell}\downarrow_x (\mathtt{a!} \oplus \mathtt{b!}).\, \mathtt{do}_x\, \mathtt{a!}$ and its type f' from Example 7.3. We can type the system $S = \mathsf{A}[P'] \mid (x)\, \mathsf{B}[\mathtt{tell}\downarrow_x \mathtt{a?}]$ as follows:

$$\dfrac{\dfrac{\vdash P' \colon f'}{\vdash_{\mathsf{A}} \mathsf{A}[P'] \colon f'}\ [\text{T-SA}] \qquad \dfrac{\dfrac{\mathsf{B} \neq \mathsf{A} \qquad \{x\} \cap \operatorname{dom} f'\{x \mapsto \bot\} = \emptyset}{\vdash_{\mathsf{A}} \mathsf{B}[\mathtt{tell}\downarrow_x \mathtt{a?}] \triangleright f'\{x \mapsto \bot\}}\ [\text{T-SAFree1}]}{\vdash_{\mathsf{A}} (x)\, \mathsf{B}[\mathtt{tell}\downarrow_x \mathtt{a?}] \triangleright f'}\ [\text{T-SDel2}]}{\vdash_{\mathsf{A}} S \colon f'}\ [\text{T-SPar2}]$$

Note that the typing is possible because the delimited variable x in the process of B does not interfere with the free variable x in P'. This would be consistent with α-converting x.

8. Basic properties of the type system

In this section we present some basic properties of our type system for CO_2; we defer to the next section for subject reduction, progress, and type safety.

The type system assigns to $*$ a pointed abstract process $f(*)$ which may only contain τ and $\tau_?$ actions, hence $f(*)$ is always honest.

Lemma 8.1 (Honesty of $f(*)$)**.**

$$\vdash P \colon f \implies f(*) \textit{ only contains } \tau \textit{ and } \tau_? \textit{ actions} \tag{8.1a}$$

$$\vdash P \colon f \implies f(*) \textit{ honest} \tag{8.1b}$$

$$\vdash_{\mathsf{A}} S \colon f \implies f(*) \textit{ honest} \tag{8.1c}$$

Proof. See section B on page 38. □

The following lemma relates the free channels of processes and systems with the domain of their type. While these sets are the same for processes, in the case of systems we only have inclusion. For instance, for a system $S = \mathsf{A}[\mathtt{tell}\downarrow_x C] \mid \mathsf{B}[\mathtt{tell}\downarrow_y D]$ with type f, we have that x and y are free in S, but y does not belong to $\operatorname{dom} f$.

Lemma 8.2 (Free channels of typed processes/systems)**.**

$$\vdash P \colon f \implies \operatorname{dom} f = \operatorname{fnv}(P) \cup \{*\} \tag{8.2a}$$

$$\vdash S \colon f \implies \operatorname{dom} f \subseteq \operatorname{fnv}(S) \cup \{*\} \tag{8.2b}$$

Proof. Straightforward induction on the typing derivations of $\vdash P \colon f$ and $\vdash S \colon f$. □

Item (8.3a) of the following lemma states that, when the typing relation $\vdash_{\mathsf{A}} S : f$ holds, then the process of A occurs in S. Conversely, item (8.3b) states that when the typing relation $\vdash_{\mathsf{A}} S \rhd f$ holds, then the process of A does *not* occur in S.

Lemma 8.3 (Participants and system typing)**.** *For all systems S and process types f:*

$$\vdash_{\mathsf{A}} S : f \implies \exists \boldsymbol{v}, S_0, P \;.\; S \equiv (\boldsymbol{v})\,(\mathsf{A}[P] \mid S_0) \tag{8.3a}$$

$$\vdash_{\mathsf{A}} S \rhd f \implies \forall \boldsymbol{v}, S_0, P \;.\; S \not\equiv (\boldsymbol{v})\,(\mathsf{A}[P] \mid S_0) \tag{8.3b}$$

Proof. Easy induction on the typing derivation and inspection of the typing rules. □

Types are preserved by structural equivalence of processes and systems.

Lemma 8.4 (Type congruence)**.**

$$\vdash P : f \;\wedge\; P \equiv P' \implies \vdash P' : f \tag{8.4a}$$

$$\vdash_{\mathsf{A}} S : f \;\wedge\; S \equiv S' \implies \vdash_{\mathsf{A}} S' : f \tag{8.4b}$$

$$\vdash_{\mathsf{A}} S \rhd f \;\wedge\; S \equiv S' \implies \vdash_{\mathsf{A}} S' \rhd f \tag{8.4c}$$

Proof. See section B on page 40. □

The following lemma states that the type of a process is unique, up-to structural equivalence of pointed abstract processes and systems. The same holds for $\vdash_{\mathsf{A}}:$ typing of systems. On the contrary, the type obtained by the judgements $\vdash_{\mathsf{A}} \rhd$ is *not* unique: for instance, we have that $\vdash_{\mathsf{A}} \mathsf{B}[\mathbf{0}] \rhd f$, for all types f.

Lemma 8.5 (Uniqueness of typing)**.**

$$\vdash P : f \;\wedge\; \vdash P : f' \implies f = f' \tag{8.5a}$$

$$\vdash_{\mathsf{A}} S : f \;\wedge\; \vdash_{\mathsf{A}} S : f' \implies f = f' \tag{8.5b}$$

Proof. Item (8.5a) follows by easy induction on the derivation of $\vdash P : f$, by noting that each process can be typed with exactly one rule, and all the rules are deterministic. Likewise for item (8.5b), where in the case of rule [T-SA] we exploit Equation (8.5a) □

The following theorem is a cornerstone of our analysis technique, since it establishes the decidability of type inference. This gives us a terminating algorithm to statically analyse the honesty of a process P. If we succeed in inferring the type of P, then we know that P is honest (by type safety, Theorem 9.9); otherwise, we cannot establish whether P is honest or not. Hence, our analysis safely over-approximates honesty.

Theorem 8.6 (Decidability of type inference)**.** *Type inference for processes is decidable.*

Proof. All the typing rules in Figure 6 follow the syntactic structure of processes. We can infer the type of P by structural recursion, inferring the types of the sub-processes, and composing them according to the typing rules. The only non-trivial rule is [T-DEL], which requires to check abstract honesty; however, this is decidable by Theorem 6.6. □

9. Subject reduction and type safety

In this section we establish the main result of this paper, i.e. type safety for CO_2 processes (Theorem 9.9), ensuring that typeable processes are honest. As usual, the proof of type safety builds upon subject reduction and progress, hence we start by proving these results.

Subject reduction states that each step of the process of A within a system is matched by a step of its type. Formalising this requires to define a transition system over types: roughly, a type f takes a transition on a prefix π when all its points $f(u)$ agree to take a transition on the abstract prefix $[\pi]_u$.

Definition 9.1 (Type transitions)**.** We write $f \xrightarrow{\pi}_\sharp f'$ when $\operatorname{dom} f' \subseteq \operatorname{dom} f$, and:

$$\forall u \in \operatorname{dom} f' \quad : \quad f(u) \xrightarrow{[\pi]_u}_\sharp f'(u)$$

$$\forall u \in \operatorname{dom} f \setminus \operatorname{dom} f' \quad : \quad f(u) \xrightarrow{[\pi]_u}_\sharp f'(*)$$

The second clause of Definition 9.1 accounts for the transitions of a system that discharge free channels: the transition system of types must also allow to restrict the domain of types accordingly, as per Lemma 8.2a. When a free channel u is lost by a system transition, it is also lost from the domain of its type; further, when u is no longer free in the system, at the type level it is represented by the dummy $*$.

Example 9.2. Consider the following transition, where $\pi = \mathtt{do}_s\,\mathtt{a!}$:

$$\begin{array}{rcl} S & = & \mathsf{A}[\mathtt{do}_s\,\mathtt{a!}] \mid \mathsf{B}[\cdots] \mid s[\mathsf{A} : \mathtt{a!}\ [] \mid \mathsf{B} : \mathtt{a?}\ []] \xrightarrow{\mathsf{A} : \pi} \\ & & \mathsf{A}[\mathbf{0}] \quad\ \mid \mathsf{B}[\cdots] \mid s[\mathsf{A} : \mathtt{1}\ [] \ \mid \mathsf{B} : \mathtt{a?}\ [\mathtt{a!}]] \ = \ S' \end{array}$$

We have the following typings for S and its reduct S':

$$\vdash_{\mathsf{A}} S : \{s \mapsto \mathtt{a!},\ * \mapsto \tau_?\} \ = \ f \qquad\qquad \vdash_{\mathsf{A}} S' : \{* \mapsto \mathbf{0}\} \ = \ f'$$

We have $f(*) \xrightarrow{\tau_?}_\sharp f'(*)$, which satisfies the first clause of Definition 9.1, and $f(s) \xrightarrow{\mathtt{a!}}_\sharp f'(*)$, which satisfies also the second clause. Therefore, $f \xrightarrow{\pi}_\sharp f'$.

To prove subject reduction, we need to cope with the fact that rule [Fuse] substitutes session names for variables. These substitutions affect its typing derivation, as shown by the following example.

Example 9.3. Let $S = \mathsf{A}[\mathtt{do}_x\,\mathtt{a!}] | \{\downarrow_x \mathtt{a!}\}_{\mathsf{A}} | \{\downarrow_y \mathtt{a?}\}_{\mathsf{B}}$, and $S' = \mathsf{A}[\mathtt{do}_s\,\mathtt{a!}] | s[\mathsf{A} : \mathtt{a!}\ [] \mid \mathsf{B} : \mathtt{a?}\ []]$. Then, by rule [Fuse] we have the transition $(x, y)\, S \xrightarrow{\mathsf{K} : \mathtt{fuse}} (s)\, S'$. The typings of the (open) systems S, S' are:

$$\vdash_{\mathsf{A}} S : f = \{x \mapsto \mathtt{a!},\ * \mapsto \tau_?\} \qquad\qquad \vdash_{\mathsf{A}} S' : f' = \{s \mapsto \mathtt{a!},\ * \mapsto \tau_?\}$$

Note that variable x in the domain of f has been "substituted" with s in f'. Technically, such substitutions are obtained through the operator $\bullet$ formalised below.

Definition 9.4 (Substitutions on types)**.** We define substitutions on types as follows:

$$f \bullet \{^s/_{\boldsymbol{x}}\} = \begin{cases} f & \text{if } \boldsymbol{x} \cap \operatorname{dom} f = \emptyset \\ f\{y \mapsto \bot\}\{s \mapsto f(y)\} & \text{if } \boldsymbol{x} \cap \operatorname{dom} f = \{y\} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Substituting a (free) variable with a fresh session name does not affect the typeability of a system or process, but requires adjusting the type with operator $\bullet$.

Lemma 9.5 (Typing and substitution)**.** *For all processes P, systems S, types f, and for all substitutions $\sigma = \{^{s}/_{x}\}$ such that $s \notin \mathrm{fnv}(P) \cup \mathrm{fnv}(S)$ and $f \bullet \sigma$ is defined:*

$$\vdash P : f \implies \vdash P\sigma : f \bullet \sigma \tag{9.5a}$$

$$\vdash_{\mathsf{A}} S : f \implies \vdash_{\mathsf{A}} S\sigma : f \bullet \sigma \tag{9.5b}$$

$$\vdash_{\mathsf{A}} S \rhd f \implies \vdash_{\mathsf{A}} S\sigma \rhd f \bullet \sigma \tag{9.5c}$$

Proof. See section B on page 41. □

We can now state subject reduction: typeability is preserved by system transitions. We need to consider a few cases, depending on which participant moves (either A under typing, or any other participant B), and on which typing relation is used ($:$ or $\rhd$). Note that, by Lemma 8.3b, when A moves the typing relation $\rhd$ cannot hold, so we have only three cases. When a system S takes a transition due to A, the $:$-type of the reduct cannot be the same as the type of S, because the action consumed in S has to be consumed also in the type. Rather, such move of A can be "simulated" by a corresponding move of the type (Definition 9.1). System transitions caused by $\mathsf{B} \neq \mathsf{A}$ preserve the type (both for $:$ and $\rhd$).

Note that the hypothesis that f is honest always holds for *closed* systems, as it is implied by Lemma 8.1c and Lemma 8.2b.

Theorem 9.6 (Subject reduction)**.** *If $S \xrightarrow{\mathsf{B} : \pi} S'$ and f is honest:*

$$\vdash_{\mathsf{A}} S : f \ \wedge\ \mathsf{B} = \mathsf{A} \implies \exists f' \,.\, f \xrightarrow{\pi}_{\sharp} f' \wedge \vdash_{\mathsf{A}} S' : f' \tag{9.6a}$$

$$\vdash_{\mathsf{A}} S : f \ \wedge\ \mathsf{B} \neq \mathsf{A} \implies \vdash_{\mathsf{A}} S' : f \tag{9.6b}$$

$$\vdash_{\mathsf{A}} S \rhd f \ \wedge\ \mathsf{B} \neq \mathsf{A} \implies \vdash_{\mathsf{A}} S' \rhd f \tag{9.6c}$$

Proof. See Section C on page 46. □

Progress is somehow dual to subject reduction: roughly, it guarantees that type transitions are "simulated" by system transitions. However, this does not necessarily holds for `do` transitions, since they may be enabled in the type but forbidden in the system. For `do` transitions, progress only guarantees that they are *ready* in the system. For instance, in

$$\vdash_{\mathsf{A}} \ \mathsf{A}[\mathtt{do}_s\,\mathtt{a!} \mid \mathtt{do}_s\,\mathtt{b!}] \mid s[\mathsf{A} : \mathtt{b!}\ [] \mid \mathsf{B} : \cdots] \ : \ \{s \mapsto \mathtt{a!} \mid \mathtt{b!},\ * \mapsto \cdots\}$$

the type can perform both `a!` and `b!` at session s, while the system can only perform `b!` (the action `a!` is ready but not fireable).

Theorem 9.7 (Progress)**.** *If $\vdash_{\mathsf{A}} S : f$ with f honest, $f \xrightarrow{\pi}_{\sharp} f'$, and $u \in \mathrm{dom}\, f$, then:*

$$\pi \in \{\tau,\ \mathtt{tell}\downarrow_u C\} \implies \exists S' \,.\, S \xrightarrow{\mathsf{A} : \pi} S' \ \wedge\ \vdash_{\mathsf{A}} S' : f' \tag{9.7a}$$

$$\pi = \mathtt{do}_u\, a \implies a \in S\downarrow_u^{\mathsf{A}} \tag{9.7b}$$

Proof. See section D on page 58. □

In order to prove type safety, we first show that if f is the type associated to some process, and $f(u)$ takes a transition, then the whole f can take a transition.

Lemma 9.8 (Self-concordance)**.** *If f is inhabited, then for all $u \in \mathrm{dom}\, f$:*

$$f(u) \xrightarrow{\alpha}_{\sharp} \mathcal{P}' \implies \exists \pi, f' \,.\, [\pi]_u = \alpha \ \wedge\ f \xrightarrow{\pi}_{\sharp} f' \ \wedge\ f'{\uparrow_{\{u\}}}(u) = \mathcal{P}'$$

Proof. See section E on page 59. □

Type safety guarantees that typeable closed processes are honest.

Theorem 9.9 (Type safety)**.** *For all closed P, if $\vdash P : f$ then P is honest.*

Proof. By Definition 4.5 on page 12, we need to prove that for all A-free S:

$$\mathsf{A}[P] \mid S \to^* S' \qquad \Longrightarrow \qquad \mathsf{A} \text{ ready in } S'$$

Since P is closed, then by Lemma 8.2a it follows that $\operatorname{dom} f = \{*\}$; together with the fact that S is A-free, then by a simple structural induction on S we can apply the rules in Figure 7 to obtain the typing $\vdash_{\mathsf{A}} S \rhd f$. Hence, we can reconstruct the following typing derivation:

$$\dfrac{\dfrac{\vdash P : f}{\vdash_{\mathsf{A}} \mathsf{A}[P] : f}\ [\text{T-SA}] \qquad \vdash_{\mathsf{A}} S \rhd f}{\vdash_{\mathsf{A}} \mathsf{A}[P] \mid S : f}\ [\text{T-SPAR2}]$$

Since $\operatorname{dom} f = \{*\}$, then by Lemma 8.1b it follows that f is honest. Since honesty is preserved by transitions of process types (Lemma 6.4), then by iterating Theorem 9.6 (Subject reduction) we have that:

$$\vdash_{\mathsf{A}} S' : f' \qquad\qquad \text{for some } f' \text{ honest}$$

By Definition 4.3, we must prove that, whenever $S' \equiv (\boldsymbol{\nu})S'_0$ for some $\boldsymbol{\nu}$ and S'_0, then, for all s, $S'_0 \in \mathrm{Rdy}^{\mathsf{A}@s}$. This is equivalent to:

$$\mathrm{O}^{\mathsf{A}@s}(S'_0) \neq \emptyset \implies \mathrm{O}^{\mathsf{A}@s}(S'_0) \cap S'_0{\Downarrow}^{\mathsf{A}@s} \neq \emptyset$$

The above equivalence holds because $\mathrm{O}^{\mathsf{A}@s}(S'_0) \cap \mathtt{A}^?$ contains at most one element[2]. So, assume $b \in \mathrm{O}^{\mathsf{A}@s}(S'_0) \neq \emptyset$. Then, S'_0 must be structurally equivalent to:

$$(\boldsymbol{u})\,\big(\mathsf{A}[P'] \mid s[\gamma] \mid S''_0\big) \qquad\qquad \text{with } \gamma = \mathsf{A} : C\ [\beta_{\mathsf{A}}] \mid \mathsf{B} : D\ []$$

for some $\boldsymbol{u}$, B, C, D, P', S''_0 and β_{A} such that $s \notin \boldsymbol{u}$, and β_{A} is either empty or a singleton. Since $S' \equiv (\boldsymbol{\nu u})\,(\mathsf{A}[P'] \mid s[\gamma] \mid S''_0)$ and $\vdash_{\mathsf{A}} S' : f'$, by Lemma 8.4 we have that:

$$\vdash_{\mathsf{A}} (\boldsymbol{\nu u})\,\big(\mathsf{A}[P'] \mid s[\gamma] \mid S''_0\big) : f'$$

By inverting the typing derivation, we obtain some honest g such that:

$$\vdash_{\mathsf{A}} \mathsf{A}[P'] : g \qquad\qquad \vdash_{\mathsf{A}} s[\gamma] \rhd g \qquad\qquad \vdash_{\mathsf{A}} S''_0 \rhd g$$

Since $\vdash_{\mathsf{A}} s[\gamma] \rhd g$ can only by typed via rule [T-SFUSE], then $g{\uparrow_{\{s\}}}(s)$ must realize $\mathcal{C} = \alpha_{\mathsf{A}}(\gamma)$. Let $\mathcal{P} = g{\uparrow_{\{s\}}}(s)$. Since $\mathcal{P}$ realizes $\mathcal{C}$, then by Definition 6.1 we know that $(\mathcal{C}, \mathcal{P})$ is honest, i.e. for all $\mathcal{D}, \mathcal{Q}$:

$$(\mathcal{C}, \mathcal{P}) \to_\sharp^* (\mathcal{D}, \mathcal{Q}) \implies \mathcal{Q} \text{ is abstractly ready for } \mathcal{D}$$

In particular, $\mathcal{P}$ is abstractly ready for $\mathcal{C}$, i.e. by Definition 6.1:

$$\mathcal{C} \xrightarrow{b}\!\!\!\!\rightarrow_\sharp \qquad \Longrightarrow \qquad \exists a\ .\ (\mathcal{C}, \mathcal{P}) \xrightarrow{\tau}_\sharp{}^* \xrightarrow{a}_\sharp$$

Since $\gamma \xrightarrow{\mathsf{A}:b}\!\!\!\!\rightarrow$, then by Lemma 5.2a we have that $\mathcal{C} \xrightarrow{b}\!\!\!\!\rightarrow_\sharp$, and so from the above implication:

$$(\mathcal{C}, \mathcal{P}) \xrightarrow{\tau}_\sharp (\mathcal{C}, \mathcal{P}_1) \xrightarrow{\tau}_\sharp \cdots \xrightarrow{\tau}_\sharp (\mathcal{C}, \mathcal{P}_n) \xrightarrow{a}_\sharp (\mathcal{C}', \mathcal{P}')$$

By the rules in Figure 5, we obtain a corresponding trace of $\mathcal{P}$:

$$\mathcal{P} \xrightarrow{\tau}_\sharp \mathcal{P}_1 \xrightarrow{\tau}_\sharp \cdots \xrightarrow{\tau}_\sharp \mathcal{P}_n \xrightarrow{a}_\sharp \mathcal{P}'$$

[2] This holds for both synchronous and asynchronous semantics of session types: see Remark 4.4.

We now exploit Lemma 9.8 to prove that there exist $\pi_1, \ldots, \pi_n, \pi'$, and $g_1, \ldots, g_n, g'$ such that $[\pi_i] = \tau$ for all $i \in 1..n$, $[\pi'] = a$, and:

$$g \xrightarrow{\pi_1}_\sharp g_1 \xrightarrow{\pi_2}_\sharp \cdots \xrightarrow{\pi_n}_\sharp g_n \xrightarrow{\pi'}_\sharp g'$$

To justify the first step, we observe that, by Definition 5.5, it must either be $\mathcal{P} = g(s)$, or $\mathcal{P} = g(*)$. Since g is inhabited (by $S_0 = \mathsf{A}[P'] \mid s[\gamma] \mid S_0''$), by applying Lemma 9.8 on $\mathcal{P} \xrightarrow{\tau}_\sharp \mathcal{P}_1$ (with $u = s$ or $u = *$, accordingly) we obtain that $g \xrightarrow{\pi_1}_\sharp g_1$, for some π_1 such that $[\pi_1] = \tau$ (and so, we have either $\pi_1 = \tau$ or $\pi_1 = \mathtt{tell}$). Then, we have either $g_1(s) = \mathcal{P}_1$ or $g_1(*) = \mathcal{P}_1$. Moreover, since g is honest, then by Theorem 9.7a it follows that g_1 is inhabited by some S_1 such that $S_0 \xrightarrow{\mathsf{A} : \pi_1} S_1$. Hence, by Lemma 6.4 also g_1 is honest. By iterating the same argument to $g_1, \ldots, g_{n-1}$, we obtain that g_n is honest, inhabited, and either $g_n(s) = \mathcal{P}_n$ or $g_n(*) = \mathcal{P}_n$. Therefore, we can apply once again Lemma 9.8, from which we obtain some π' and g' such that $[\pi'] = a$, and $g_n \xrightarrow{\pi'}_\sharp g'$. Note that, in the meanwhile, we have constructed a trace:

$$S_0 \xrightarrow{\mathsf{A} : \pi_1} S_1 \xrightarrow{\mathsf{A} : \pi_2} \cdots \xrightarrow{\mathsf{A} : \pi_n} S_n$$

Since $[\pi'] = a$, the abstraction cannot be done on $*$, and so $\pi' = \mathtt{do}_s\, a$. Then, by Theorem 9.7b it must be $a \in S_n{\downarrow_s^{\mathsf{A}}}$. Then, by Definition 4.2, we also have $a \in S_0 \Downarrow^{\mathsf{A}@s}$. Therefore, since $s \notin \boldsymbol{u}$, then $a \in S_0' \Downarrow^{\mathsf{A}@s}$. To conclude, just note that, since $(\mathcal{C}, \mathcal{P}_n) \xrightarrow{a}_\sharp (\mathcal{C}', \mathcal{P}')$, then $a \in \mathrm{O}^{\mathsf{A}@s}(S_0')$. We have then proved $\mathrm{O}^{\mathsf{A}@s}(S_0') \cap S_0' \Downarrow^{\mathsf{A}@s} \neq \emptyset$, which concludes the proof. □

The following example shows that our type system is incomplete, i.e. there exists an honest process which is not typeable.

Example 9.10. We can type the process $P' = \mathtt{tell}{\downarrow_x}\, \mathtt{a!}\,.\,(\mathtt{do}_x\, \mathtt{a!} + \mathtt{do}_y\, \mathtt{b!})$ with:

$$f = \{\, x \mapsto \langle \mathtt{a!} \rangle.(\mathtt{a!} + \tau_?),\ \ y \mapsto \tau.(\tau_? + \mathtt{b!}),\ \ * \mapsto \tau.(\tau_? + \tau_?) \,\}$$

Since $f(x)$ is *not* abstractly honest, then $P = (x)(y)P'$ is *not* typeable. However, P is honest: indeed, the branch $\mathtt{do}_y\, \mathtt{b!}$ is immaterial, since the session y cannot be established.

10. Related work and conclusions

The concept of *contract-oriented computing* (as surveyed in Section 1) has been introduced in [14], and CO_2 has been later proposed as a *contract-agnostic* calculus for contract-oriented computing in [12]. CO_2 has been instantiated with several contract models — both binary [13, 10, 9] and multiparty [12, 36, 8]. Here, similarly to [9], we consider bilateral contracts, formalised as binary session types (Section 2). A minor difference w.r.t. [13, 10, 36] is that in the present work we do not have `fuse` as a language primitive: the creation of fresh sessions is performed non-deterministically by the context (rule [Fuse] in Figure 3). This is equivalent to assume a contract broker which collects all contracts, and may establish sessions when compliant ones are found. The notion of honesty used here is slightly different from the one in [13]. There, A is considered *culpable* in a session s when she has enabled moves in s; by performing such moves, A can exculpate herself. Honesty in [13] requires A to be always able to exculpate herself, in all contexts and in all sessions. This is a mild variation of the notion of honesty considered here: we believe that these two notions are equivalent, under a fair semantics. A survey of other variants of honesty, and of their properties, is in [15].

A type system to safely over-approximate honesty in CO_2 has been first proposed in [10]. The present work improves those results in two main directions. First, we have redesigned the type system so that now it has a decidable type inference (Theorem 8.6). This result relies on a (sound and complete) algorithm for deciding abstract honesty, based on submarking reachability in Petri nets (Theorem 6.6). Second, we can safely over-approximate the honesty of processes which interact through *asynchronous* session types (Theorem 4.9).

The programming model envisioned by CO_2 has been implemented as a *contract-oriented middleware* [6] featuring *timed* session types [5] as contracts. This middleware collects the contracts advertised by services, and creates a session between two services when their contracts are compliant. The middleware monitors all the intra-session communications (similarly to [42]), also checking that services respect the time constraints specified in their contracts. When a participant is culpable of a contract violation its reputation is decreased, consequently reducing its chances of being involved in further sessions.

The contract model in the present work (Section 2) is based on an interpretation of session types as *behavioural contracts* equipped with an LTS semantics, which are also studied in [3, 11, 17, 4, 43]. Similarly to [11, 43], here we combine contracts with buffers and define semantics and notions of compliance accounting for asynchronous interactions. A novel contribution in this work is Theorem 2.5, which proves the undecidability of compliance between session types interacting via unbounded buffers.

The problem of ensuring safe interactions in session-based systems has been addressed to a wide extent in the literature, e.g. in [27, 18, 22, 32, 35, 18, 33, 45, 24, 25]. In many of these approaches (surveyed in [34]), deadlock-freedom in the presence of interleaved sessions is not directly implied by typeability. For instance, the two (dishonest) processes:

$$\begin{aligned} P &= (x,y)\,\mathtt{tell}\downarrow_x \mathtt{a?}.\mathtt{tell}\downarrow_y \mathtt{b?}.\mathtt{do}_x\,\mathtt{a?}.\mathtt{do}_y\,\mathtt{b?} \\ Q &= (x,y)\,\mathtt{tell}\downarrow_x \mathtt{a!}.\mathtt{tell}\downarrow_y \mathtt{b!}.\mathtt{do}_y\,\mathtt{b!}.\mathtt{do}_x\,\mathtt{a!} \end{aligned}$$

would typically be well-typed. However, the composition $\mathsf{A}[P] \mid \mathsf{B}[Q]$ reaches a deadlock after fusing the sessions: in fact, A remains waiting on x (while not being ready at y), and B remains waiting on y (while not being ready at x).

Multiple interleaved sessions has been tackled e.g. in [27, 18, 22, 24, 25]. To guarantee deadlock freedom, these approaches usually require that all the interactions on a session must end before another session can be used. For instance, the system $\mathsf{A}[P] \mid \mathsf{B}[Q]$ above would *not* be typeable in [22], coherently with the fact that it is not deadlock-free. The resulting notions seem however quite different from honesty, because we do not necessarily classify as dishonest processes with interleaved sessions. For instance, the processes:

$$(x,y)\,\mathtt{tell}\downarrow_x \mathtt{a?}.\mathtt{tell}\downarrow_y \mathtt{b!}.\big(\mathtt{do}_x\,\mathtt{a?}.\mathtt{do}_y\,\mathtt{b!} + \mathtt{do}_y\,\mathtt{b!}.\mathtt{do}_x\,\mathtt{a?}\big)$$

$$(x,y)\,\mathtt{tell}\downarrow_x \mathtt{a?}.\,\mathtt{tell}\downarrow_y (\mathtt{b!} \oplus \mathtt{c!}).\,\big(\mathtt{do}_x\,\mathtt{a?}.\mathtt{do}_y\,\mathtt{b!} + \tau.(\mathtt{do}_y\,\mathtt{c!} \mid \mathtt{do}_x\,\mathtt{a?})\big)$$

would not be typeable according to [22], but they are honest in our theory (see Example 7.6). A further difference between these approaches and ours is that we do *not* assume the knowledge of the whole system, but instead we focus on typing standalone participants. Once a participant A is typed using the rules in Figure 6, then A will always be ready to make her sessions progress. Our type discipline does not make assumptions on contexts other than their A-freeness (which can be easily obtained via digital signatures). Deadlocks can only occur when the context starts behaving dishonestly.

The problem of checking if the abstract behaviour of a service conforms to a role of a given choreography has been investigated in [20]. Under suitable well-formed conditions, conformance is attained exploiting the *should testing* pre-order. Similar techniques have been used in [21] to define contract-based composition of services. A main difference between these approaches and ours is that we also consider the contexts where some participants can be dishonest, i.e. we aim at establishing whether a process abides by its own contract regardless of its execution context.

In the top-down approach to design a distributed application, one specifies its overall communication behaviour through a *choreography*, which validates some global properties of the application (e.g. safety, deadlock-freedom, *etc.*). To ensure that the application enjoys such properties, all the components forming the application have to be verified; this can be done e.g. by projecting the choreography to end-point views, against which these components are verified [44, 33]. This approach assumes that designers control the whole application, e.g., they develop all the needed components. However, in many real-world scenarios several components are developed independently, without knowing at design time which other components they will be integrated with. In these scenarios, the compositional verification pursued by the top-down approach is not immediately applicable, because the choreography is usually unknown, and even if it were known, only a subset of the needed components is available for verification. The ideas pursued in this paper depart from the top-down approach, because designers can advertise contracts to discover the needed components (and so ours can be considered a *bottom-up* approach). Coherently, the main property we are interested in is *honesty*, which is a property of components, and not of global applications. Some works mixing top-down and bottom-up composition have been proposed in the past few years [26, 37, 36, 8].

Future works. An interesting direction for future research would be extending our type discipline to the version of CO_2 studied in [9], which features value-passing processes and conditionals. The behaviour of conditional processes $\texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q$ is delicate to abstract in the presence of recursion. A naïve attempt could simply abstract it as $\tau.\mathcal{P} + \tau.\mathcal{Q}$, where $\mathcal{P}$ and $\mathcal{Q}$ are the abstractions of P and Q, respectively. However, this abstraction would not be safe. For instance, consider the process:

$$R \;=\; \texttt{tell}\downarrow_x \texttt{a!}.(\texttt{rec}\ \mathrm{X}().\ \texttt{if false then do}_x\ \texttt{a! else}\ \tau.\mathrm{X}())()$$

We have that $(x)R$ is *not* honest, since the promised $\texttt{a!}$ action will never be performed. However, the naïve abstraction of P' on channel x would be $\langle \texttt{a!} \rangle.(\mathit{rec}\ @\mathrm{X}.\tau.\texttt{a!} + \tau.\tau.@\mathrm{X})$ which is abstractly honest, since after $\langle \texttt{a!} \rangle$ the action $\texttt{a!}$ is persistently weakly enabled.

The issue is that, under recursion, $\tau.\mathcal{P} + \tau.\mathcal{Q}$ does not precisely represent the internal non-determinism caused by conditionals. We could tackle this issue in two different ways: either refining the notion of readiness as in [9], or — more directly — ruling out conditional processes wherein some recursive calls are not guarded by $\texttt{do}$-actions (as in $\tau.\mathrm{X}()$ above).

Another possible extension of our type system is a more precise handling of recursion, so to type processes with non-empty lists of parameters in recursive calls $\mathrm{X}(\boldsymbol{u})$. However this would be a non-trivial task, due to the presence of processes which can possibly extrude names through recursive invocations. This is why the typing rule [T-Rec] in Figure 6 only allows recursion with no parameters. This forbids, e.g., to type the following honest process:

$$(x)\,\texttt{tell}\downarrow_x \texttt{a!}.(\texttt{rec}\ \mathrm{X}(y).\ \texttt{do}_y\ \texttt{a!}.(z)\,\texttt{tell}\downarrow_z \texttt{a!}.\mathrm{X}(z))(x)$$

Note however that the current type system is precise enough to correctly establish the honesty of complex processes, like e.g. the travel agency case study in [9].

In our work, we focused mainly on the synchronous setting. Asynchrony exposes several technical challenges that are not present in the synchronous case. For instance, the operational semantics of CO_2 becomes undecidable, because compliance between asynchronous session types is undecidable (Theorem 2.9). As for honesty, the set of ∞-honest processes is larger than the set of 1-honest one, and it is still undecidable (Theorem 4.8). For instance, Example 4.11 provides a ∞-honest process which is not 1-honest (and so, not typeable). While Theorem 4.9 allows us to lift 1-honesty to ∞-honesty, making our type system sound even in the asynchronous setting, it would be desirable to extend our analysis so to cover a larger class of processes that includes some ∞-honest but not 1-honest processes. A possible refinement of our type system would be to modify the abstraction of prefixes to take into account the fact that, in the asynchronous setting, enabled outputs on a session u are never blocking once u has been established. Technically, this would require to improve the abstraction of prefixes (Definition 7.1) so that the enabled output actions (but for the first action in u) are abstracted on $v \neq u$ as τ, instead of $\tau_?$. For instance, the process in Example 4.11 would be typeable in this modified analysis.

Another research direction is the integration of contract-oriented primitives within mainstream programming languages. This can be done e.g. as in [6], where Java APIs are provided to interact with a middleware which handles contracts and sessions. The problem of honesty of Java programs is analogous to that of CO_2, with the additional issue that contract violations are explicitly sanctioned by the middleware in terms of reputation loss. The suite of tools Diogenes [2] supports programmers in writing honest Java code. The tool translates honest CO_2 specifications into skeletal Java programs, and checks that their honesty is preserved upon refinement. To this purpose, the tool first infers a CO_2 process which approximates the behaviour of a Java program; the honesty of this process is then verified through the model checker in [9].

Acknowledgement

This work has been partially supported by Aut. Reg. of Sardinia P.I.A. 2013 "NOMAD", and by EU COST Action IC1201 "Behavioural Types for Reliable Large-Scale Software Systems" (BETTY). We thank the anonymous reviewers and Nicola Atzei for their insightful comments on a preliminary version of this paper.

References

[1] L. Aceto, W. Fokkink, A. Ingólfsdóttir, and B. Luttik. A finite equational base for CCS with left merge and communication merge. *ACM Trans. Comput. Log.*, 10(1), 2009.

[2] N. Atzei and M. Bartoletti. Developing honest Java programs with Diogenes. In *Proc. FORTE*, volume 9688 of *LNCS*, pages 52–61. Springer, 2016.

[3] F. Barbanera and U. de'Liguoro. Two notions of sub-behaviour for session-based client/server systems. In *Proc. PPDP*, pages 155–164, 2010.

[4] F. Barbanera and U. de'Liguoro. Sub-behaviour relations for session-based client/server systems. *Mathematical Structures in Computer Science*, 25:1339–1381, 9 2015.

[5] M. Bartoletti, T. Cimoli, M. Murgia, A. S. Podda, and L. Pompianu. Compliance and subtyping in timed session types. In *Proc. FORTE*, LNCS, pages 161–177. Springer, 2015.

[6] M. Bartoletti, T. Cimoli, M. Murgia, A. S. Podda, and L. Pompianu. A contract-oriented middleware. In *Proc. FACS*, 2015. Extended version available at `http://co2.unica.it`.

[7] M. Bartoletti, T. Cimoli, and R. Zunino. Compliance in behavioural contracts: a brief survey. In *Programming Languages with Applications to Biology and Security*, volume 9465 of *LNCS*, pages 103–121. Springer, 2015.
[8] M. Bartoletti, J. Lange, A. Scalas, and R. Zunino. Choreographies in the wild. *Science of Computer Programming*, 109:36 – 60, 2015.
[9] M. Bartoletti, M. Murgia, A. Scalas, and R. Zunino. Verifiable abstractions for contract-oriented systems. *Journal of Logical and Algebraic Methods in Programming*, 2015. To appear.
[10] M. Bartoletti, A. Scalas, E. Tuosto, and R. Zunino. Honesty by typing. In *Proc. FMOODS/FORTE*, volume 7892 of *LNCS*, pages 305–320. Springer, 2013.
[11] M. Bartoletti, A. Scalas, and R. Zunino. A semantic deconstruction of session types. In *Proc. CONCUR*, volume 8704 of *LNCS*, pages 402–418. Springer, 2014.
[12] M. Bartoletti, E. Tuosto, and R. Zunino. Contract-oriented computing in CO_2. *Sci. Ann. Comp. Sci.*, 22(1):5–60, 2012.
[13] M. Bartoletti, E. Tuosto, and R. Zunino. On the realizability of contracts in dishonest systems. In *Proc. COORDINATION*, volume 7274 of *LNCS*, pages 245–260. Springer, 2012.
[14] M. Bartoletti and R. Zunino. A calculus of contracting processes. In *Proc. LICS*, 2010.
[15] M. Bartoletti and R. Zunino. On the decidability of honesty and of its variants. In *Web Services, Formal Methods, and Behavioral Types*, volume 9421 of *LNCS*. Springer, 2015.
[16] H. Bekić. *Programming Languages and Their Definition: H. Bekić (1936–1982)*, chapter Definable operations in general algebras, and the theory of automata and flowcharts, pages 30–55. Springer, 1984.
[17] G. Bernardi and M. Hennessy. Using higher-order contracts to model session types. In *Proc. CONCUR*, LNCS, pages 387–401. Springer, 2014.
[18] L. Bettini, M. Coppo, L. D'Antoni, M. D. Luca, M. Dezani-Ciancaglini, and N. Yoshida. Global progress in dynamically interleaved multiparty sessions. In *Proc. CONCUR*, LNCS, pages 418–433. Springer, 2008.
[19] D. Brand and P. Zafiropulo. On communicating finite-state machines. Technical report, IBM Zurich Research Laboratory, 1981.
[20] M. Bravetti and G. Zavattaro. Towards a unifying theory for choreography conformance and contract compliance. In *Software Composition*, 2007.
[21] M. Bravetti and G. Zavattaro. Contract-based discovery and composition of web services. In *Formal Methods for Web Services*, volume 5569 of *LNCS*, pages 261–295. Springer, 2009.
[22] G. Castagna, M. Dezani-Ciancaglini, E. Giachino, and L. Padovani. Foundations of session types. In *Proc. PPDP*, 2009.
[23] G. Cécé and A. Finkel. Verification of programs with half-duplex communication. *Inf. Comput.*, 202(2):166–190, 2005.
[24] M. Coppo, M. Dezani-Ciancaglini, L. Padovani, and N. Yoshida. Inference of global progress properties for dynamically interleaved multiparty sessions. In *Proc. COORDINATION*, pages 45–59, 2013.
[25] M. Coppo, M. Dezani-Ciancaglini, N. Yoshida, and L. Padovani. Global progress for dynamically interleaved multiparty sessions. *MSCS*, 760:1–65, 2015.
[26] P.-M. Deniélou and N. Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *Proc. ICALP*, pages 174–186, 2013.
[27] M. Dezani-Ciancaglini, U. de'Liguoro, and N. Yoshida. On progress for structured communications. In *Proc. TGC*, pages 257–275, 2007.
[28] J. Esparza. On the decidability of model checking for several μ-calculi and Petri nets. In *Proc. CAAP*, 1994.
[29] J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fundamenta Informaticae*, 31(1):13–25, 1997.
[30] A. Finkel and P. McKenzie. Verifying identical communicating processes is undecidable. *Theor. Comput. Sci.*, 174(1-2):217–230, 1997.
[31] M. Hack. Decidability questions for Petri nets. Technical report, M.I.T., 1976. Ph. D. Thesis.
[32] K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type disciplines for structured communication-based programming. In *Proc. ESOP*, volume 1381 of *LNCS*, 1998.
[33] K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. *J. ACM*, 63(1):9:1–9:67, 2016. Extended version of a paper presented at POPL'08.

[34] H. Hüttel, I. Lanese, V. T. Vasconcelos, L. Caires, M. Carbone, P.-M. Deniélou, D. Mostrous, L. Padovani, A. Ravara, E. Tuosto, H. T. Vieira, and G. Zavattaro. Foundations of session types and behavioural contracts. *ACM Comput. Surv.*, 49(1):3:1–3:36, 2016.
[35] N. Kobayashi. A new type system for deadlock-free processes. In *Proc. CONCUR*, pages 233–247, 2006.
[36] J. Lange and A. Scalas. Choreography synthesis as contract agreement. In *Proc. ICE*, 2013.
[37] J. Lange and E. Tuosto. Synthesising choreographies from local session types. In *Proc. CONCUR*, 2012.
[38] J. Lange, E. Tuosto, and N. Yoshida. From communicating machines to graphical choreographies. In *Proc. POPL*, pages 221–232, 2015.
[39] E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984.
[40] R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems*. PhD thesis, Technische Universität München, 1998.
[41] A. Mukhija, A. Dingwall-Smith, and D. Rosenblum. QoS-aware service composition in Dino. In *Proc. ECOWS*, pages 3–12, 2007.
[42] R. Neykova, L. Bocchi, and N. Yoshida. Timed runtime monitoring for multiparty conversations. In *Proc. BEAT*, pages 19–26, 2014.
[43] A. Scalas. *A semantic deconstruction of session types*. PhD thesis, University of Cagliari, 2015.
[44] W. M. P. van der Aalst, N. Lohmann, P. Massuthe, C. Stahl, and K. Wolf. Multiparty contracts: Agreeing and implementing interorganizational processes. *Comput. J.*, 53(1):90–106, 2010.
[45] V. T. Vasconcelos. Fundamentals of Session Types. *Information and Computation*, 217:52–70, 2012.
[46] G. Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, Cambridge, MA, USA, 1993.

A. Proofs for Section 6

Notation A.1 (Realizes)**.** *We say that:*

- $\mathcal{P}$ *is* ready for $\mathcal{C}$ *whenever* $(\mathcal{C}, \mathcal{P})$ *is ready;*
- $\mathcal{P}$ realizes $\mathcal{C}$ *whenever* $(\mathcal{C}, \mathcal{P})$ *is honest.*

Lemma A.2. *For all* $\alpha \in \{\tau, \tau_?, \langle C' \rangle\}$*:* $\quad \mathcal{P} \xrightarrow{\alpha}_\sharp \mathcal{P}' \ \wedge \ (\Gamma, \mathcal{P})$ *honest* $\implies (\Gamma, \mathcal{P}')$ *honest*

Proof. Assume $(\Gamma, \mathcal{P})$ honest, $\alpha \in \{\tau, \tau_?, \langle C' \rangle\}$, and $\mathcal{P} \xrightarrow{\alpha}_\sharp \mathcal{P}'$. To show $(\Gamma, \mathcal{P}')$ honest, assume that $(\Gamma, \mathcal{P}') \to_\sharp^* (\mathcal{C}, \mathcal{P}'')$. We have the following two cases:

- $\alpha \in \{\tau, \tau_?\}$. Since $\mathcal{P} \xrightarrow{\alpha}_\sharp \mathcal{P}'$, then by rule [A-Tau] we have $(\Gamma, \mathcal{P}) \xrightarrow{\alpha}_\sharp (\Gamma, \mathcal{P}')$. Hence, $(\Gamma, \mathcal{P}) \xrightarrow{\alpha}_\sharp (\Gamma, \mathcal{P}') \to_\sharp^* (\mathcal{C}, \mathcal{P}'')$. Since $(\Gamma, \mathcal{P})$ is honest, then $(\mathcal{C}, \mathcal{P}'')$ is ready.
- $\alpha = \langle C' \rangle$. We can modify the trace $(\Gamma, \mathcal{P}') \to_\sharp^* (\mathcal{C}, \mathcal{P}'')$ by adding C' to all the sets of contracts in the trace. The result is still a valid trace, because enlarging the set of contracts does not reduce the applicability of the rules in Figure 5. In this way, we obtain the trace $(\Gamma', \mathcal{P}') \to_\sharp^* (\mathcal{C}, \mathcal{P}'')$, where Γ' is either $\Gamma \cup \{C'\}$ if Γ is a set of contracts, or $\Gamma' = \Gamma$ otherwise. Therefore, we have $(\Gamma', \mathcal{P}) \xrightarrow{\tau}_\sharp (\Gamma', \mathcal{P}') \to_\sharp^* (\mathcal{C}, \mathcal{P}'')$. Since $(\Gamma, \mathcal{P})$ is honest, then $(\mathcal{C}, \mathcal{P}'')$ is ready. □

Proof of Lemma 6.4. Immediate consequence of Lemma A.2.

Proof of Lemma 6.5 (Abstract readiness and parallel composition). First, we note that if $\Gamma \not\twoheadrightarrow_\sharp$, then both sides are trivially true by Definition 6.1, hence they are equivalent. Therefore, we can assume that $\Gamma \xrightarrow{a}\!\!\!\!\twoheadrightarrow_\sharp$ for some a, and so $\Gamma = \mathcal{C}$ for some $\mathcal{C}$.

For the $\Leftarrow$ direction, assume w.l.o.g. that $(\mathcal{C}, \mathcal{P})$ is ready. By Definition 6.1, this implies that $(\mathcal{C}, \mathcal{P}) \xrightarrow{\tau}_\sharp{}^* \xrightarrow{b}_\sharp$ for some b. Inverting the rules of the semantics of pointed abstract systems, we find a corresponding trace for the pointed abstract process $\mathcal{P}$, namely $\mathcal{P} \xrightarrow{\tau}_\sharp{}^* \xrightarrow{b}_\sharp$. Consequently, by rule [C-ParL], $\mathcal{P} \mid \mathcal{Q} \xrightarrow{\tau}_\sharp{}^* \xrightarrow{b}_\sharp$, hence $(\mathcal{C}, \mathcal{P} \mid \mathcal{Q}) \xrightarrow{\tau}_\sharp{}^* \xrightarrow{b}_\sharp$.

For the $\Rightarrow$ direction, assume that $(\mathcal{C}, \mathcal{P} \mid \mathcal{Q})$ is ready, hence $(\mathcal{C}, \mathcal{P} \mid \mathcal{Q}) \xrightarrow{\tau}_\sharp{}^* \xrightarrow{b}_\sharp$. As above, inverting the rules we obtain $\mathcal{P} \mid \mathcal{Q} \xrightarrow{\tau}_\sharp{}^* \xrightarrow{b}_\sharp$. Therefore, we have that either of $\mathcal{P} \xrightarrow{\tau}_\sharp{}^* \xrightarrow{b}_\sharp$ and $\mathcal{Q} \xrightarrow{\tau}_\sharp{}^* \xrightarrow{b}_\sharp$ because parallel pointed abstract processes cannot interact with each other (as their semantics does not allow synchronization or communication among parallel processes). In both cases, we can then lift the trace on pointed abstract processes to a trace on pointed abstract systems, showing that $(\mathcal{C}, \mathcal{P})$ or $(\mathcal{C}, \mathcal{Q})$ is ready. □

Lemma A.3. *If* $\mathcal{Q}$ *only contains* τ *and* $\tau_?$ *actions, then:*

$$\mathcal{P} \text{ honest} \iff \mathcal{P} \mid \mathcal{Q} \text{ honest} \tag{A.3a}$$

$$\mathcal{P} \text{ honest} \implies \mathcal{P}\{\mathcal{Q}/\mathcal{X}\} \text{ honest} \tag{A.3b}$$

Proof. For the $\Leftarrow$ direction of item (A.3a), assume that $(\emptyset, \mathcal{P}) \to_\sharp^* (\mathcal{C}, \mathcal{P}')$. By the semantics of pointed abstract systems, this implies that $(\emptyset, \mathcal{P} \mid \mathcal{Q}) \to_\sharp^* (\mathcal{C}, \mathcal{P}' \mid \mathcal{Q})$, which is ready because $\mathcal{P} \mid \mathcal{Q}$ is honest. By Lemma 6.5, either $(\mathcal{C}, \mathcal{P}')$ is ready or $(\mathcal{C}, \mathcal{Q})$ is ready. If $(\mathcal{C}, \mathcal{P}')$ is ready, we have the thesis. Otherwise, $\mathcal{C}$ has obligations, and $(\mathcal{C}, \mathcal{Q})$ is ready — contradicting $\mathcal{Q}$ performing only τ and $\tau_?$ moves (indeed, by Definition 6.1, a pointed abstract systems with only τ e $\tau_?$ cannot fulfil obligations).

For the $\Rightarrow$ direction of item (A.3a), assume that

$$(\emptyset, \mathcal{P} \mid \mathcal{Q}) \;\to_\sharp^*\; (\mathcal{C}, \mathcal{R}) \tag{1}$$

Assuming that $\mathcal{C}$ has some obligations, we need to prove that $(\mathcal{C}, \mathcal{R})$ $\xrightarrow{\tau}_\sharp^* \xrightarrow{a}_\sharp$ for some a. We must have that $\mathcal{R} = \mathcal{P}' \mid \mathcal{Q}'$, where $\mathcal{P} \longrightarrow_\sharp^* \mathcal{P}'$ and $\mathcal{Q} \longrightarrow_\sharp^* \mathcal{Q}'$. This is because two parallel components can only interact by performing actions on the contract. Exploiting (1), we can construct a trace $(\emptyset, \mathcal{P}) \to_\sharp^* (\mathcal{C}, \mathcal{P}')$. Indeed, in the transitions $\mathcal{Q} \longrightarrow_\sharp^* \mathcal{Q}'$ there are only τ and $\tau_?$ actions, so the evolution of the contract in (1) only depends on the transitions of $\mathcal{P}$. Since $\mathcal{P}$ is honest and $\mathcal{C}$ has some obligations, we must have $(\mathcal{C}, \mathcal{P}')$ $\xrightarrow{\tau}_\sharp^* \xrightarrow{a}_\sharp$, hence by rule [C-ParL] we conclude that $(\mathcal{C}, \mathcal{P}' \mid \mathcal{Q}')$ $\xrightarrow{\tau}_\sharp^* \xrightarrow{a}_\sharp$.

For item (A.3b), note that the process $\mathcal{P}\{\mathcal{Q}/\mathcal{X}\}$ differs from $\mathcal{P}$ in that the occurrences of the free variables $\mathcal{X}$ have been replaced by $\mathcal{Q}$ (with the usual assumption that the substitution is capture-avoiding). According to the transition semantics, $\mathcal{X}$ is a stuck process, while by hypothesis $\mathcal{Q}$ only performs τ and $\tau_?$ actions. Intuitively, the substitution $\{\mathcal{Q}/\mathcal{X}\}$ is irrelevant for the transitions of $\mathcal{P}\{\mathcal{Q}/\mathcal{X}\}$, except for those cases where a trace of $\mathcal{P}$ would expose an $\mathcal{X}$ at the top level. Hence, a residual of $\mathcal{P}\{\mathcal{Q}/\mathcal{X}\}$ is formed by a parallel component where the substitution had no effect in the trace, and the residuals of all the substituted top-level $\mathcal{X}$'s — which are residuals of $\mathcal{Q}$. More precisely, we have that in every trace $(\emptyset, \mathcal{P}\{\mathcal{Q}/\mathcal{X}\}) \to_\sharp^* (\mathcal{C}, \mathcal{R})$:

$$\mathcal{R} \;=\; \mathcal{P}'\{\mathcal{Q}/\mathcal{X}\} \mid \mathcal{Q}_1 \mid \cdots \mid \mathcal{Q}_k \qquad \text{where } \mathcal{Q}_i \text{ are residuals of } \mathcal{Q} \text{ and } \mathcal{P} \longrightarrow_\sharp^* \mathcal{P}' \mid \underbrace{\mathcal{X} \mid \cdots \mid \mathcal{X}}_{k \text{ times}}$$

This fact relies on two properties of pointed abstract systems. First, the residuals of $\mathcal{Q}$ never interact with $\mathcal{P}'\{\mathcal{Q}/\mathcal{X}\}$, since $\mathcal{Q}$ can only perform τ and $\tau_?$ actions. Second, sums in pointed abstract processes are prefix-guarded (otherwise, substituting the honest $(\mathtt{a!}, \mathcal{X} + \mathtt{a!})$ we would obtain $(\mathtt{a!}, \tau + \mathtt{a!})$, which is no longer honest). Roughly, prefix-guardedness ensures that the moves of $\mathcal{Q}$ in $\mathcal{P}\{\mathcal{Q}/\mathcal{X}\}$ do not conflict with the moves of $\mathcal{P}$.

Hence, from the trace $(\emptyset, \mathcal{P}\{\mathcal{Q}/\mathcal{X}\}) \to_\sharp^* (\mathcal{C}, \mathcal{R})$ we can construct a trace $(\emptyset, \mathcal{P}) \to_\sharp^* (\mathcal{C}, \mathcal{P}' \mid \mathcal{X} \mid \cdots \mid \mathcal{X})$. Since $\mathcal{P}$ is honest, then $(\mathcal{C}, \mathcal{P}' \mid \mathcal{X} \mid \cdots \mid \mathcal{X})$ is ready, and since the components $\mathcal{X}$ are stuck, then also $(\mathcal{C}, \mathcal{P}')$ must be ready. Then, $(\mathcal{C}, \mathcal{P}'\{\mathcal{Q}/\mathcal{X}\} \mid \mathcal{Q}_1 \mid \cdots \mid \mathcal{Q}_k)$ is ready. □

B. Proofs for Section 7

Proof of Lemma 8.1 (Honesty of $f(*)$). Item (8.1a) follows by easy induction on the typing derivation of $\vdash P : f$ (Figure 6). Item (8.1b) is a direct consequence of Equation (8.1a). For item (8.1c), we proceed by induction on the typing derivation of $\vdash_{\mathsf{A}} S : f$.

- [T-SA]. We have:
$$\frac{\vdash P : f}{\vdash_{\mathsf{A}} S = \mathsf{A}[P] : f}\ \text{[T-SA]}$$
Thus, $f(*)$ is honest by item (8.1b).
- [T-SDel2], [T-SPar2]. Straightforward, by applying the induction hypothesis on the rule premises.

Lemma B.1 (Structural equivalence and substitutions)**.** *For all processes P, P', systems S, S', and for all substitutions σ:*

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

Proof. Case analysis on all the different cases of structural equivalence. □

Lemma B.2 (Substitution of delimited channels)**.** *For all S, f, u, and $\mathcal{P}$:*

$$\vdash S \rhd f \ \wedge\ u \notin \mathrm{fnv}(S) \implies \vdash S \rhd f\{u \mapsto \mathcal{P}\} \ \text{ and } \ \vdash S \rhd f\{u \mapsto \bot\}$$

Furthermore, these typing derivations have the same depth as the original one.

Proof. Straightforward induction, by inspection of the rules in Figure 7. □

Lemma B.3 (Substitution of recursion variables)**.** *For all P, Q, f, and g:*

$$\vdash P : f \ \wedge\ \vdash Q : g \implies \vdash P\{Q/\mathrm{X}\} : \lambda u\,.\, f{\uparrow_A}(u)\{g{\uparrow_A}(u)/@\mathrm{X}\} \quad \text{where } A = \mathrm{fnv}(P\{Q/\mathrm{X}\}).$$

Proof. Tedious induction on the typing derivation of $\vdash P : f$. □

Lemma B.4. *If f is honest, then $f \bullet \sigma$ and $f{\uparrow_A}$ are honest, for all σ and for all A.*

Proof. Assume that f is honest, and that $u \in \mathrm{dom}\,(f \bullet \sigma)$. We have that $(f \bullet \sigma)(u) = f(v)$, for some $v \in \mathrm{dom}\, f$. Similarly, for all $u \in \mathrm{dom}\,(f{\uparrow_A})$, we have that $f{\uparrow_A}(u) = f(v)$, for some $v \in \mathrm{dom}\, f$. The thesis follows by the assumption that f is honest. □

Lemma B.5. *For all S, f, f', and π such that $\vdash_{\mathsf{A}} S \rhd f$ and $f \xrightarrow{\pi}_\sharp f'$:*

$$\pi = \mathtt{do}_s\, a \ \wedge\ s[\cdots] \notin S \implies \vdash_{\mathsf{A}} S \rhd f' \tag{1}$$
$$(\forall s, a\,.\, \pi \neq \mathtt{do}_s\, a) \implies \vdash_{\mathsf{A}} S \rhd f' \tag{2}$$

Proof. By induction on the typing derivation of $\vdash_{\mathsf{A}} S \rhd f$. All cases are straightforward, but the following ones:

- [T-SFz1]. We have:
$$\frac{f{\uparrow_{\{x\}}}(x) \text{ realizes } C}{\vdash_{\mathsf{A}} S = \{\downarrow_x C\}_{\mathsf{A}} \rhd f}\ \text{[T-SFz1]}$$
We first note that, since $f \xrightarrow{\pi}_\sharp f'$, then $f{\uparrow_{\{x\}}}(x) \xrightarrow{[\pi]_x}_\sharp f'{\uparrow_{\{x\}}}(x)$. By Definition 7.1 we have that for item (1) $[\pi]_x = \tau_?$, while for item (2) $[\pi]_x \neq a$, for all a. Then, in both cases by Lemma A.2 we have that $f'{\uparrow_{\{x\}}}(x)$ realizes C. Therefore, the thesis follows by using rule [T-SFz1].
- [T-SFuse]. We have:
$$\frac{f{\uparrow_{\{t\}}}(t) \text{ realizes } \alpha_{\mathsf{A}}(\gamma)}{\vdash_{\mathsf{A}} S = t[\gamma] \rhd f}\ \text{[T-SFuse]}$$
We first note that, since $f \xrightarrow{\pi}_\sharp f'$, then $f{\uparrow_{\{t\}}}(t) \xrightarrow{[\pi]_t}_\sharp f'{\uparrow_{\{t\}}}(t)$. By Definition 7.1 we have that for item (1) $[\pi]_t = \tau_?$ (because the assumption $s[\cdots] \notin S$ implies that $t \neq s$), while for item (2) $[\pi]_t \neq a$, for all a. Then, in both cases by Lemma A.2 we have that $f'{\uparrow_{\{t\}}}(t)$ realizes $\alpha_{\mathsf{A}}(\gamma)$. The thesis follows by using rule [T-SFuse].

- [T-SDel1]. We have:
$$\frac{\vdash_{\mathsf{A}} S_0 \rhd f\{u \mapsto \bot\}}{\vdash_{\mathsf{A}} S = (u)S_0 \rhd f} \text{ [T-SDel1]}$$
Since $f \xrightarrow{\pi}_\sharp f'$, then $f\{u \mapsto \bot\} \xrightarrow{\pi}_\sharp f'\{u \mapsto \bot\}$. For item (2) we can apply the induction hypothesis, which gives $\vdash_{\mathsf{A}} S_0 \rhd f'\{u \mapsto \bot\}$; the thesis then follows by rule [T-SDel1]. For item (1) there are two cases, according to whether $s = u$ or not. If $s \neq u$, then we can apply the induction hypothesis, and proceed as above. Otherwise, if $s = u$, then it might be the case that $s[\cdots] \in S_0$, i.e. $S_0 = s[\gamma] \mid \cdots$. In this case, by Definition 9.1 we also have that:
$$f\{u \mapsto \bot\} \xrightarrow{\pi'}_\sharp f'\{u \mapsto \bot\}$$
where $\pi' = \mathtt{do}_t\, a$, with $t[\cdots] \notin S_0$. This holds because $[\pi]_v = \tau_? = [\pi']_v$, for all $v \in \operatorname{dom} f$. Therefore, we can apply the induction hypothesis, which gives $\vdash_{\mathsf{A}} S_0 \rhd f'\{u \mapsto \bot\}$; we then conclude by rule [T-SDel1].
- [T-SPar1]. Straightforward, by applying the induction hypothesis on both premises.
- [T-SAFree2], [T-SAFree3]. The thesis follows because $\operatorname{dom} f' \subseteq \operatorname{dom} f$. □

Lemma B.6.
$$f{\uparrow_A}\{u \mapsto \mathcal{P}\} \;=\; (f\{u \mapsto \mathcal{P}\}){\uparrow_A} \tag{B.6a}$$
$$f{\uparrow_A}\{u \mapsto \bot\} \;=\; (f\{u \mapsto \bot\}){\uparrow_{A\setminus\{u\}}} \tag{B.6b}$$

Proof. Straightforward case analysis on $u \in A$ in Definition 5.5. □

Notation B.7. *For all α, and for $\circ \in \{|, +\}$, we will use the compact notation $\alpha.f$ for $\lambda u.\ \alpha.f(u)$, and $f \circ g$ for $\lambda u.\ f(u) \circ g(u)$.*

Proof of Lemma 8.4 (Structural equivalence and typing). By induction on the typing derivation. Most cases are straightforward; we only show the case of scope extrusion.

For Equation (8.4a), consider the case where:
$$P \;=\; (u)(P_0 \mid P_1) \;\equiv\; P_0 \mid (u)P_1 \;=\; P' \qquad \text{with } u \notin \operatorname{fnv}(P_0)$$
We have the following typing derivation for P, where $A = \operatorname{dom} g_0 \cup \operatorname{dom} g_1$:
$$\dfrac{\dfrac{\vdash P_0 \colon g_0 \qquad \vdash P_1 \colon g_1}{\vdash P_0 \mid P_1 \colon g_0{\uparrow_A} \mid g_1{\uparrow_A} \;=\; g}\text{ [T-Par]} \qquad g{\uparrow_{\{u\}}}(u) \text{ honest}}{\vdash P \colon g\{u \mapsto \bot\} = f}\text{ [T-Del]}$$
Since $\vdash P_0 \colon g_0$ and $u \notin \operatorname{fnv}(P_0)$, then by Lemma 8.2a it must be $g_0{\uparrow_{\{u\}}}(u) = g_0(*)$. Hence, by Lemma 8.1a, $g_0{\uparrow_{\{u\}}}(u)$ only contains τ and $\tau_?$ actions. Together with the fact that $g{\uparrow_{\{u\}}}(u) = g_0{\uparrow_{\{u\}}}(u) \mid g_1{\uparrow_{\{u\}}}(u)$ is honest, by Lemma A.3a we deduce that $g_1{\uparrow_{\{u\}}}(u)$ is honest. We can then construct the following typing derivation for P', where $A' = \operatorname{dom} g_0 \cup (\operatorname{dom} g_1 \setminus \{u\})$:
$$\dfrac{\vdash P_0 \colon g_0 \qquad \dfrac{\vdash P_1 \colon g_1 \qquad g_1{\uparrow_{\{u\}}}(u) \text{ honest}}{\vdash (u)P_1 \colon g_1\{u \mapsto \bot\}}\text{ [T-Del]}}{\vdash P' \colon g_0{\uparrow_{A'}} \mid g_1\{u \mapsto \bot\}{\uparrow_{A'}} = f'}\text{ [T-Par]}$$

To conclude the proof, we need to prove that $f' = f$. Since $u \notin \mathrm{fnv}(P_0) \cup \{*\} = \mathrm{dom}\, g_0$, then $A' = (\mathrm{dom}\, g_0 \cup \mathrm{dom}\, g_1) \setminus \{u\} = A \setminus \{u\}$, and so $g_0 = g_0\{u \mapsto \bot\}$.

We then have:

$$\begin{aligned}
f &= g\{u \mapsto \bot\} && \text{by def. } f \\
&= (g_0{\uparrow_A} \mid g_1{\uparrow_A})\{u \mapsto \bot\} && \text{by def. } g \\
&= g_0{\uparrow_A}\{u \mapsto \bot\} \mid g_1{\uparrow_A}\{u \mapsto \bot\} \\
&= g_0\{u \mapsto \bot\}{\uparrow_{A'}} \mid g_1\{u \mapsto \bot\}{\uparrow_{A'}} && \text{by Lemma B.6b} \\
&= g_0{\uparrow_{A'}} \mid g_1\{u \mapsto \bot\}{\uparrow_{A'}} && \text{as shown before} \\
&= f' && \text{by def. } f'
\end{aligned}$$

The analogous case for systems (for 8.4b) is similar. □

Proof of Lemma 9.5 (Typing and substitution). We start by proving item (9.5a). We proceed by induction on the typing derivation of $\vdash P : f$. We have the following cases, according to the last rule used in the derivation:

- [T-Nil], [T-Var]. Trivial, since the substitution is vacuous (both on P and on f).
- [T-Par]. We have:

$$\frac{\vdash P_0 : f_0 \quad \vdash P_1 : f_1 \quad A = \mathrm{dom}\, f_0 \cup \mathrm{dom}\, f_1}{\vdash P = P_0 \mid P_1 : \lambda u \,.\, f_0{\uparrow_A}(u) \mid f_1{\uparrow_A}(u)}\ \text{[T-Par]}$$

  Since $\mathrm{dom}\, f_0 \subseteq \mathrm{dom}\, f \supseteq \mathrm{dom}\, f_1$ and $f \bullet \sigma$ is defined, then also $f_0 \bullet \sigma$ and $f_1 \bullet \sigma$ are defined. Then, by applying the induction hypothesis on both premises:

$$\vdash P_0\sigma : f_0 \bullet \sigma \qquad \vdash P_1\sigma : f_1 \bullet \sigma$$

  Then, by rule [T-Par]:

$$\frac{\vdash P_0\sigma : f_0 \bullet \sigma \quad \vdash P_1\sigma : f_1 \bullet \sigma \quad B = \mathrm{dom}\, f_0 \bullet \sigma \cup \mathrm{dom}\, f_1 \bullet \sigma}{\vdash P\sigma = P_0\sigma \mid P_1\sigma : \lambda u \,.\, (f_0 \bullet \sigma){\uparrow_B}(u) \mid (f_1 \bullet \sigma){\uparrow_B}(u) = g}\ \text{[T-Par]}$$

  To conclude, we need to prove that $g = f \bullet \sigma$, i.e.:

$$(f_i{\uparrow_A}) \bullet \sigma \;=\; (f_i \bullet \sigma){\uparrow_B} \qquad (\text{for } i \in \{0,1\})$$

  We proceed by cases on $(\mathrm{dom}\, \sigma \cap \mathrm{dom}\, f_0, \mathrm{dom}\, \sigma \cap \mathrm{dom}\, f_1)$. Since $f_0 \bullet \sigma$ and $f_1 \bullet \sigma$ are defined, we have the following cases:
  - $(\emptyset, \emptyset)$. We have that $f_i \bullet \sigma = f_i$ for $i \in \{0,1\}$, hence $B = A$. We have that:

$$\begin{aligned}
(f_i{\uparrow_A}) \bullet \sigma &= f_i{\uparrow_A} && (\mathrm{dom}\, \sigma \cap \mathrm{dom}\, f_i{\uparrow_A} = \mathrm{dom}\, \sigma \cap A = \emptyset) \\
&= (f_i \bullet \sigma){\uparrow_A} \\
&= (f_i \bullet \sigma){\uparrow_B}
\end{aligned}$$

– $(\{x\},\emptyset)$. By Definition 9.4 we have $B = (A \setminus \{x\}) \cup \{s\}$. For $i = 0$, we have:

$$\begin{aligned}
(f_0{\uparrow_A}) \bullet \sigma &= f_0{\uparrow_A}\{x \mapsto \bot\}\{s \mapsto f_0{\uparrow_A}(x)\} && (\operatorname{dom}\sigma \cap \operatorname{dom} f_0{\uparrow_A} = \{x\})\\
&= f_0{\uparrow_A}\{x \mapsto \bot\}\{s \mapsto f_0(x)\} && (x \in \operatorname{dom} f_0)\\
&= (f_0\{s \mapsto f_0(x)\}){\uparrow_A}\{x \mapsto \bot\} && (\text{(B.6a)})\\
&= (f_0\{s \mapsto f_0(x)\}\{x \mapsto \bot\}){\uparrow_{A\setminus\{x\}}} && (\text{(B.6b)})\\
&= (f_0\{s \mapsto f_0(x)\}\{x \mapsto \bot\}){\uparrow_{(A\setminus\{x\})\cup\{s\}}} && (s \in \operatorname{dom} f_0\{s \mapsto f_0(x)\}\cdots)\\
&= (f_0 \bullet \sigma){\uparrow_B} && (B = (A \setminus \{x\}) \cup \{s\})
\end{aligned}$$

For $i = 1$, we have:

$$\begin{aligned}
(f_1{\uparrow_A}) \bullet \sigma &= f_1{\uparrow_A}\{x \mapsto \bot\}\{s \mapsto f_1{\uparrow_A}(x)\} && (\operatorname{dom}\sigma \cap \operatorname{dom} f_1{\uparrow_A} = \{x\})\\
&= f_1{\uparrow_A}\{x \mapsto \bot\}\{s \mapsto f_1(*)\} && (x \notin \operatorname{dom} f_1)\\
&= f_1{\uparrow_{A\setminus\{x\}}}\{s \mapsto f_1(*)\} && (x \notin \operatorname{dom} f_1)\\
&= f_1{\uparrow_{(A\setminus\{x\})\cup\{s\}}} && (s \notin \operatorname{dom} f_1)\\
&= (f_1 \bullet \sigma){\uparrow_B} && (B = (A \setminus \{x\}) \cup \{s\})
\end{aligned}$$

– $(\emptyset,\{x\})$. Symmetrical to the previous case.
– $(\{x\},\{x\})$. For $i = 0$, the proof is analogous to the corresponding subcase $i = 0$ in the case $(\{x\},\emptyset)$. For $i = 1$, the proof is symmetric.
– $(\{x\},\{y\})$ with $x \neq y$. This case does not apply, because otherwise $f \bullet \sigma$ would be undefined.

- [T-Sum]. Similar to the previous case.
- [T-Del]. We have:

$$\frac{\vdash P' \colon f' \quad f'(u) \text{ honest}}{\vdash P = (u)P' \colon f'\{u \mapsto \bot\} = f}\ \text{[T-Del]}$$

Since α-conversion does not change typing, we can assume that $s \neq u$. Now, let $\boldsymbol{z} = \operatorname{dom}\sigma \cap \operatorname{dom} f$. We proceed by cases on the possible values of $\boldsymbol{z}$. Since $f \bullet \sigma$ is defined, we only have the following three cases:

– $\boldsymbol{z} = \emptyset$. By the induction hypothesis (with substitution $\sigma_{\neq u}$), we have:

$$\vdash P'\sigma_{\neq u} \colon f' \bullet \sigma_{\neq u} = f'$$

Hence, the thesis follows by rule [T-Del].
– $\boldsymbol{z} = \{u\}$. This case does not apply, because $u \notin \operatorname{dom} f$.
– $\boldsymbol{z} = \{x\}$, with $x \neq u$. By the induction hypothesis (with substitution $\sigma_{\neq u}$):

$$\vdash P'\sigma_{\neq u} \colon f' \bullet \sigma_{\neq u} = f'\{x \mapsto \bot\}\{s \mapsto f'(x)\}$$

Let $g = f'\{x \mapsto \bot\}\{s \mapsto f'(x)\}$. Since $f'(u)$ is honest, then $g(u) = f'(u)$ is honest as well. Then, by rule [T-Del]:

$$\frac{\vdash P'\sigma_{\neq u} \colon g \quad g(u) \text{ honest}}{\vdash P\sigma = (u)P'\sigma_{\neq u} \colon g\{u \mapsto \bot\}}\ \text{[T-Del]}$$

The thesis follows because:

$$\begin{aligned} g\{u \mapsto \bot\} &= f'\{x \mapsto \bot\}\{s \mapsto f'(x)\}\{u \mapsto \bot\} && \text{(by def. of } g\text{)} \\ &= f'\{u \mapsto \bot\}\{x \mapsto \bot\}\{s \mapsto (f'\{u \mapsto \bot\})(x)\} \\ &= f'\{u \mapsto \bot\} \bullet \sigma && \text{(by Definition 9.4)} \\ &= f \bullet \sigma && \text{(by def. of } f\text{)} \end{aligned}$$

- [T-Rec]. We have:

$$\frac{\vdash P' \colon f}{\vdash P = (\texttt{rec X}().\ P')() \colon \lambda u.\ \mathit{rec}\ @\mathrm{X}.f(u)}\ \text{[T-Rec]}$$

  By the induction hypothesis we have $\vdash P'\sigma \colon f \bullet \sigma$. Then, by rule [T-Rec]:

$$\frac{\vdash P'\sigma \colon f \bullet \sigma}{\vdash (\texttt{rec X}().\ P'\sigma)() \colon \lambda u.\ \mathit{rec}\ @\mathrm{X}.(f \bullet \sigma)(u)}\ \text{[T-Rec]}$$

  The thesis is obtained from $(\texttt{rec X}().\ P'\sigma)() = (\texttt{rec X}().\ P')()\sigma$ and

$$\lambda u.\ \mathit{rec}\ @\mathrm{X}.(f \bullet \sigma)(u) = (\lambda u.\ \mathit{rec}\ @\mathrm{X}.f(u)) \bullet \sigma$$

  which follows by case analysis on the argument: x, s or something else.

We now prove item (9.5c) (which is needed in order to prove item (9.5b)). By induction on the typing derivation of $\vdash_{\mathsf{A}} S \rhd f$, we have the following cases, according to the last rule used in the typing derivation:

- [T-SAFree0], [T-SAFree1], [T-SFzS]. Trivial: the premise is not affected by σ.
- [T-SAFree2]. We have the following two cases:
  - $x \in \operatorname{dom}\sigma$. Then, $S\sigma = \{\downarrow_s C\}_{\mathsf{B}}$, and so the thesis follows by rule [T-SFsZ].
  - $x \notin \operatorname{dom}\sigma$. Then, $S\sigma = S$, and since $f(x) = \bot$, then $(f \bullet \sigma)(x) = \bot$. Hence, the thesis follows by rule [T-SAFree2].
- [T-SAFree3]. We have that:

$$\frac{s'[\gamma]\ \mathsf{A}\text{-free} \quad f(s') = \bot}{\vdash_{\mathsf{A}} S = s'[\gamma] \rhd f}\ \text{[T-SAFree3]}$$

  Since $s \notin \mathrm{fnv}(S)$ by hypothesis, then it must be $s' \neq s$. The thesis follows because $(f \bullet \sigma)(s') = f(s')$.
- [T-SFz1]. We have:

$$\frac{f{\uparrow_{\{x\}}}(x) \text{ realizes } C}{\vdash_{\mathsf{A}} S = \{\downarrow_x C\}_{\mathsf{A}} \rhd f}\ \text{[T-SFz1]}$$

  We have the following two cases:
  - $x \in \operatorname{dom}\sigma$. Then, $S\sigma = \{\downarrow_s C\}_{\mathsf{A}}$, and so the thesis follows by rule [T-SFsZ].
  - $x \notin \operatorname{dom}\sigma$. Then, $(f \bullet \sigma){\uparrow_{\{x\}}}(x) = f{\uparrow_{\{x\}}}(x)$, which realizes C by the premise. Hence, the thesis follows by rule [T-SFz1].
- [T-SFuse]:

$$\frac{f{\uparrow_{\{s'\}}}(s') \text{ realizes } \alpha_{\mathsf{A}}(\gamma)}{\vdash_{\mathsf{A}} S = s'[\gamma] \rhd f}\ \text{[T-SFuse]}$$

  Since $s \notin \mathrm{fnv}(S)$ by hypothesis, then it must be $s' \neq s$. The thesis follows because $(f \bullet \sigma)(s') = f(s')$, which preserves the truth of the premise.

- [T-SDel1]. We have:
$$\frac{\vdash_{\mathsf{A}} S_0 \rhd f\{u \mapsto \bot\}}{\vdash_{\mathsf{A}} (u)S_0 \rhd f} \text{ [T-SDel1]}$$
W.l.o.g. we can assume $u \neq s$: otherwise we can α-convert the typing derivation and obtain the same typing. Now, let $\boldsymbol{z} = \operatorname{dom}\sigma \cap \operatorname{dom} f$. We proceed by cases on the possible values of $\boldsymbol{z}$. Since $f \bullet \sigma$ is defined, we only have the following three cases:
  - $\boldsymbol{z} = \emptyset$. By the induction hypothesis (with substitution σ), we have:
$$\vdash_{\mathsf{A}} S_0\sigma \rhd f\{u \mapsto \bot\} \bullet \sigma$$
By Definition 9.4 we have that:
$$f\{u \mapsto \bot\} \bullet \sigma = f\{u \mapsto \bot\} = (f \bullet \sigma)\{u \mapsto \bot\}$$
Hence, the thesis follows by rule [T-SDel1].
  - $\boldsymbol{z} = \{u\}$. Since $s \notin \operatorname{fnv}(S_0)$, by applying Lemma B.2 we obtain:
$$\vdash_{\mathsf{A}} S_0 \rhd f\{u \mapsto \bot\}\{s \mapsto f(u)\}$$
with a typing derivation of the same depth as the original one. By the induction hypothesis (with substitution $\sigma_{\neq u}$), we have:
$$\vdash_{\mathsf{A}} S_0\sigma_{\neq u} \rhd (f\{u \mapsto \bot\}\{s \mapsto f(u)\}) \bullet \sigma_{\neq u}$$
By Definition 9.4 we have that:
$$\begin{aligned}(f\{u \mapsto \bot\}\{s \mapsto f(u)\}) \bullet \sigma_{\neq u} &= f\{u \mapsto \bot\}\{s \mapsto f(u)\} \\ &= f\{s \mapsto f(u)\}\{u \mapsto \bot\}\end{aligned}$$
Then, by rule [T-SDel1]:
$$\frac{\vdash_{\mathsf{A}} S_0\sigma_{\neq u} \rhd f\{s \mapsto f(u)\}\{u \mapsto \bot\}}{\vdash_{\mathsf{A}} S\sigma = (u)S_0\sigma_{\neq u} \rhd f\{s \mapsto f(u)\}} \text{ [T-SDel1]}$$
Since $u \notin \operatorname{fnv}(S\sigma)$, by applying Lemma B.2 again we conclude that:
$$\vdash_{\mathsf{A}} S\sigma \rhd f\{s \mapsto f(u)\}\{u \mapsto \bot\} \;=\; f \bullet \sigma$$
  - $\boldsymbol{z} = \{x\}$, with $x \neq u$. By applying the induction hypothesis on the premise, we have: $\vdash_{\mathsf{A}} S_0\sigma \rhd f\{u \mapsto \bot\} \bullet \sigma$. We have that:
$$\begin{aligned}f\{u \mapsto \bot\} \bullet \sigma &= f\{u \mapsto \bot\}\{x \mapsto \bot\}\{s \mapsto f\{u \mapsto \bot\}(x)\} \\ &= f\{x \mapsto \bot\}\{s \mapsto f(x)\}\{u \mapsto \bot\} \\ &= (f \bullet \sigma)\{u \mapsto \bot\}\end{aligned}$$
Then, by rule [T-SDel1] we conclude:
$$\frac{\vdash_{\mathsf{A}} S_0\sigma \rhd (f \bullet \sigma)\{u \mapsto \bot\}}{\vdash_{\mathsf{A}} S\sigma = (u)S_0\sigma \rhd f \bullet \sigma} \text{ [T-SDel1]}$$
- [T-SPar1]. Straightforward, by applying the induction hypothesis on both premises.

To prove item (9.5b), we proceed by induction on the typing derivation of $\vdash_{\mathsf{A}} S : f$. We have the following cases, according to the last rule used in the typing derivation:

- [T-SA]. We have:

$$\frac{\vdash P : f}{\vdash_{\mathsf{A}} S = \mathsf{A}[P] : f}\ [\text{T-SA}]$$

By applying Lemma 9.5 on the rule premise, we obtain $\vdash P\sigma : f \bullet \sigma$. The thesis follows by rule [T-SA].

- [T-SDel2]. We have:

$$\frac{\vdash_{\mathsf{A}} S_0 : f_0 \quad f_0{\uparrow_{\{u\}}}(u) \text{ honest}}{\vdash_{\mathsf{A}} S = (u)S_0 : f_0\{u \mapsto \bot\} = f}\ [\text{T-SDel2}]$$

W.l.o.g. we can assume $u \neq s$: otherwise we can α-convert the typing derivation and obtain the same typing. Since $u \neq s$, then $s \notin \mathrm{fnv}(S_0)$, hence by Lemma 8.2 $s \notin \mathrm{dom}\, f_0$. Since $f \bullet \sigma$ is defined and $\mathrm{dom}\, f_0 \cap \mathrm{dom}\, \sigma_{\neq u} \subseteq \mathrm{dom}\, f \cap \mathrm{dom}\, \sigma$, then $f_0 \bullet \sigma_{\neq u}$ is defined. Then, by the induction hypothesis we obtain $\vdash S_0\, \sigma_{\neq u} : f_0 \bullet \sigma_{\neq u}$. To prove that $f_0{\uparrow_{\{u\}}}(u)$ honest implies $(f_0 \bullet \sigma_{\neq u}){\uparrow_{\{u\}}}(u)$ honest, we consider the following two cases:

  - $u \in \mathrm{dom}\, f_0$. Then:

$$(f_0 \bullet \sigma_{\neq u}){\uparrow_{\{u\}}}(u) = f_0{\uparrow_{\{u\}}}(u) = f_0(u) \quad \text{honest}$$

  - $u \notin \mathrm{dom}\, f_0$. Then:

$$(f_0 \bullet \sigma_{\neq u}){\uparrow_{\{u\}}}(u) = f_0{\uparrow_{\{u\}}}(u) = f_0(*) \quad \text{honest}$$

Then we can construct the following typing derivation:

$$\frac{\vdash_{\mathsf{A}} S_0\, \sigma_{\neq u} : f_0 \bullet \sigma_{\neq u} \quad (f_0 \bullet \sigma_{\neq u}){\uparrow_{\{u\}}}(u) \text{ honest}}{\vdash_{\mathsf{A}} S\, \sigma_{\neq u} = (u)(S_0\, \sigma_{\neq u}) : (f_0 \bullet \sigma_{\neq u})\{u \mapsto \bot\}}\ [\text{T-SDel2}]$$

Now, let $\boldsymbol{z} = \mathrm{dom}\, \sigma \cap \mathrm{dom}\, f$. We proceed by cases on the possible values of $\boldsymbol{z}$. Since $f \bullet \sigma$ is defined, we only have the following three cases:

  - $\boldsymbol{z} = \emptyset$. We have that:

$$(f_0 \bullet \sigma_{\neq u})\{u \mapsto \bot\} = f_0\{u \mapsto \bot\} = f = f \bullet \sigma$$

  - $\boldsymbol{z} = \{u\}$. This case does not apply, because $u \notin \mathrm{dom}\, f$.
  - $\boldsymbol{z} = \{x\}$, with $x \neq u$. We have that:

$$\begin{aligned}
(f_0 \bullet \sigma_{\neq u})\{u \mapsto \bot\} &= f_0\{x \mapsto \bot\}\{s \mapsto f_0(x)\}\{u \mapsto \bot\} && (\mathrm{dom}\, \sigma_{\neq u} \cap \mathrm{dom}\, f_0 = \{x\}) \\
&= f_0\{x \mapsto \bot\}\{s \mapsto f(x)\}\{u \mapsto \bot\} && (f_0(x) = f(x)) \\
&= f\{x \mapsto \bot\}\{s \mapsto f(x)\} && (f = f_0\{u \mapsto \bot\}) \\
&= f \bullet \sigma && (\mathrm{dom}\, \sigma \cap \mathrm{dom}\, f = \{x\})
\end{aligned}$$

- [T-SPar2]. We have:

$$\frac{\vdash_{\mathsf{A}} S_0 : f \quad \vdash_{\mathsf{A}} S_1 \rhd f}{\vdash_{\mathsf{A}} S = S_0 \mid S_1 : f}\ [\text{T-SPar2}]$$

By the induction hypothesis of item (9.5b), we have $\vdash_{\mathsf{A}} S_0\sigma : f \bullet \sigma$. By item (9.5c), we have $\vdash_{\mathsf{A}} S_1\sigma \rhd f \bullet \sigma$. Then, by applying rule [T-SPar2]:

$$\frac{\vdash_{\mathsf{A}} S_0\sigma : f \bullet \sigma \quad \vdash_{\mathsf{A}} S_1\sigma \rhd f \bullet \sigma}{\vdash_{\mathsf{A}} S\sigma = S_0\sigma \mid S_1\sigma : f \bullet \sigma}\ [\text{T-SPar2}]$$

C. Proof of Theorem 9.6 (Subject Reduction)

Assume that $S \xrightarrow{\mathsf{B}:\pi} S'$, for some B. We proceed by induction on the proof of the reduction from S to S'. We have the following cases, according to the last rule used in the reduction.

Rule [Tau]. We have:

$$S \;=\; \mathsf{B}[\tau.P + P' \mid Q] \xrightarrow{\mathsf{B}:\tau} \mathsf{B}[P \mid Q] \;=\; S'$$

- Item (9.6a). Let $P_0 = \tau.P + P' \mid Q$. By hypothesis we have that $\mathsf{B} = \mathsf{A}$, and $\vdash_{\mathsf{A}} S : f$. The only typing rule applicable for S is [T-SA], which requires $\vdash_{\mathsf{A}} P_0 : f$. Let $A = \operatorname{dom} f = \operatorname{fnv}(P_0) \cup \{*\}$, as implied by Lemma 8.2. We have:

  $$\vdash_{\mathsf{A}} P_0 : f \;=\; \lambda u\,.\,[\tau]_u\,.\,f^P{\uparrow_A}(u) + f^{P'}{\uparrow_A}(u) \mid f^Q{\uparrow_A}(u)$$

  where f^P, $f^{P'}$ and f^Q are, respectively, the types of P, P' and Q. Let $A' = \operatorname{fnv}(P) \cup \operatorname{fnv}(Q)$, and let:

  $$f' \;=\; \lambda u\,.\,f^P{\uparrow_{A'}}(u) \mid f^Q{\uparrow_{A'}}(u)$$

  We now prove that $f \xrightarrow{\tau}_\sharp f'$, according to Definition 9.1 on page 28. There are the following two cases:
  - $v \in \operatorname{dom} f'$. We must show that $f(v) \xrightarrow{\tau}_\sharp f'(v)$. We have that:

    $$\dfrac{\dfrac{\dfrac{}{\tau.f^P{\uparrow_A}(v) \xrightarrow{\tau}_\sharp f^P{\uparrow_A}(v)}\,[\text{C-Pref}]}{\tau.f^P{\uparrow_A}(v) + f^{P'}{\uparrow_A}(v) \xrightarrow{\tau}_\sharp f^P{\uparrow_A}(v)}\,[\text{C-SumL}]}{\tau.f^P{\uparrow_A}(v) + f^{P'}{\uparrow_A}(v) \mid f^Q{\uparrow_A}(v) \xrightarrow{\tau}_\sharp f^P{\uparrow_A}(v) \mid f^Q{\uparrow_A}(v)}\,[\text{C-ParL}]$$

    and since $\operatorname{dom} f' \subseteq A$, then $f'(v) = f^P{\uparrow_A}(v) \mid f^Q{\uparrow_A}(v)$. Hence, we conclude that $f(v) \xrightarrow{\tau}_\sharp f'(v)$.
  - $v \in \operatorname{dom} f \setminus \operatorname{dom} f'$. We must show that $f(v) \xrightarrow{\tau}_\sharp f'(*)$. We have that $v \notin \operatorname{fnv}(P) \cup \operatorname{fnv}(Q)$, thus $f^P{\uparrow_A}(v) \mid f^Q{\uparrow_A}(v) = f^P(*) \mid f^Q(*) = f'(*)$. The thesis follows by using the same derivation of the previous case.

  We conclude the proof by showing that $\vdash_{\mathsf{A}} S' : f'$:

  $$\dfrac{\dfrac{\vdash P : f^P \quad \vdash Q : f^Q}{\vdash P \mid Q : \lambda u\,.\,f^P{\uparrow_{A'}}(u) \mid f^Q{\uparrow_{A'}}(u) = f'}\,[\text{T-Par}]}{\vdash_{\mathsf{A}} S' : f'}\,[\text{T-SA}]$$

- Item (9.6b). The thesis holds vacuously, since, for any f, it is not possible to have a typing $\vdash_{\mathsf{A}} \mathsf{B}[\cdots] : f$ when $\mathsf{B} \neq \mathsf{A}$ (see Lemma 8.3 on page 27).
- Item (9.6c). The typing $\vdash_{\mathsf{A}} S \rhd f$ must have been obtained by rule [T-SAFree1], whose premise requires that $\operatorname{fv}(\tau.P + P' \mid Q)$. With the same rule we can derive $\vdash_{\mathsf{A}} S' \rhd f$, since $\operatorname{fv}(P \mid Q) \subseteq \operatorname{fv}(\tau.P + P' \mid Q)$.

Rule [Tell]**.** We have:

$$S \;=\; \mathsf{B}[\mathtt{tell}\downarrow_w C.P + P' \mid Q] \xrightarrow{\mathsf{B}\,:\,\mathtt{tell}\downarrow_w C} \mathsf{B}[P \mid Q] \mid \{\downarrow_w C\}_{\mathsf{B}} \;=\; S'$$

- Item (9.6a). Let $P_0 = \mathtt{tell}\downarrow_w C.P + P' \mid Q$. By hypothesis we have that $\mathsf{B} = \mathsf{A}$, and $\vdash_{\mathsf{A}} S : f$. The only typing rule applicable for S is [T-SA], which requires $\vdash_{\mathsf{A}} P_0 : f$. Let $A = \operatorname{dom} f = \operatorname{fnv}(P_0) \cup \{*\}$, as implied by Lemma 8.2. We have:

$$\vdash_{\mathsf{A}} P_0 : f \;=\; \lambda u \,.\, [\mathtt{tell}\downarrow_w C]_u \,.\, f^P{\uparrow_A}(u) + f^{P'}{\uparrow_A}(u) \mid f^Q{\uparrow_A}(u)$$

where f^P, $f^{P'}$ and f^Q are, respectively, the types of P, P' and Q. Also, by Definition 7.1 we have that $[\mathtt{tell}\downarrow_w C]_u = \mathtt{if}\ w = u\ \mathtt{then}\ \langle C\rangle\ \mathtt{else}\ \tau$. Let $A' = \operatorname{fnv}(P) \cup \operatorname{fnv}(Q)$, and let:

$$f' \;=\; \lambda u \,.\, f^P{\uparrow_{A'}}(u) \mid f^Q{\uparrow_{A'}}(u)$$

We now prove that $f \xrightarrow{\mathtt{tell}\downarrow_w C}_\sharp f'$, according to Definition 9.1 on page 28. There are the following two cases:

- $v \in \operatorname{dom} f'$. We must show that $f(v) \xrightarrow{[\mathtt{tell}\downarrow_w C]_v}_\sharp f'(v)$. We have that:

$$\dfrac{\dfrac{\dfrac{}{[\mathtt{tell}\downarrow_w C]_v.f^P{\uparrow_A}(v) \xrightarrow{[\mathtt{tell}\downarrow_w C]_v}_\sharp f^P{\uparrow_A}(v)}\,[\text{C-Pref}]}{[\mathtt{tell}\downarrow_w C]_v.f^P{\uparrow_A}(v) + f^{P'}{\uparrow_A}(v) \xrightarrow{[\mathtt{tell}\downarrow_w C]_v}_\sharp f^P{\uparrow_A}(v)}\,[\text{C-SumL}]}{[\mathtt{tell}\downarrow_w C]_v.f^P{\uparrow_A}(v) + f^{P'}{\uparrow_A}(v) \mid f^Q{\uparrow_A}(v) \xrightarrow{[\mathtt{tell}\downarrow_w C]_v}_\sharp f^P{\uparrow_A}(v) \mid f^Q{\uparrow_A}(v)}\,[\text{C-ParL}]$$

  and since $\operatorname{dom} f' \subseteq A$, then $f'(v) = f^P{\uparrow_A}(v) \mid f^Q{\uparrow_A}(v)$. Hence, we conclude that $f(v) \xrightarrow{[\mathtt{tell}\downarrow_w C]_v}_\sharp f'(v)$.
- $v \in \operatorname{dom} f \setminus \operatorname{dom} f'$. We must show that $f(v) \xrightarrow{[\mathtt{tell}\downarrow_w C]_v}_\sharp f'(*)$. We have that $v \notin \operatorname{fnv}(P) \cup \operatorname{fnv}(Q)$, thus $f^P{\uparrow_A}(v) \mid f^Q{\uparrow_A}(v) = f^P(*) \mid f^Q(*) = f'(*)$. The thesis follows by using the same derivation of the previous case.

We conclude the proof by showing that $\vdash_{\mathsf{A}} S' : f'$. We start by proving that $f'{\uparrow_{\{w\}}}(w)$ realizes C. To do that, we use the assumption of Theorem 9.6 which says that f is honest. By Definition 6.1 and Definition 5.4, since $w \in \operatorname{fnv}(P_0) = \operatorname{dom} f$, this implies that $(\emptyset, f(w))$ is honest. Let $\mathcal{P} = f^P{\uparrow_A}(w) \mid f^Q{\uparrow_A}(w)$. We have that:

$$(\emptyset, f(w)) \;\to_\sharp\; (\{C\}, \mathcal{P}) \;\to_\sharp\; (C, \mathcal{P})$$

By Definition 6.1 we know that $\mathcal{P}$ realizes C. We are left with proving that $f'{\uparrow_{\{w\}}}(w) = \mathcal{P}$. For this we consider two cases, depending on whether $w \in A'$.

If $w \in A' = \operatorname{dom} f'$, then we have $f'{\uparrow_{\{w\}}}(w) = f'(w) = f^P{\uparrow_{A'}}(w) \mid f^Q{\uparrow_{A'}}(w)$. From this, and $A' \subseteq A$, we obtain $f^P{\uparrow_{A'}}(w) \mid f^Q{\uparrow_{A'}}(w) = f^P{\uparrow_A}(w) \mid f^Q{\uparrow_A}(w) = \mathcal{P}$, hence $f'{\uparrow_{\{w\}}}(w)$ realizes C.

Otherwise, if $w \notin A' = \operatorname{dom} f' = \operatorname{fnv}(P \mid Q)$. Here we have

$$\begin{aligned} f'{\uparrow_{\{w\}}}(w) &= f'(*) = f^P{\uparrow_{A'}}(*) \mid f^Q{\uparrow_{A'}}(*) \\ &= f^P{\uparrow_A}(*) \mid f^Q{\uparrow_A}(*) = f^P{\uparrow_A}(w) \mid f^Q{\uparrow_A}(w) = \mathcal{P} \end{aligned}$$

Therefore, $f'{\uparrow_{\{w\}}}(w)$ realizes C, and we have the following typing derivation:

$$\dfrac{\dfrac{\dfrac{\vdash P : f^P \quad \vdash Q : f^Q}{\vdash P \mid Q : \lambda u\,.\, f^P{\uparrow_{A'}}(u) \mid f^Q{\uparrow_{A'}}(u) = f'}\ [\text{T-Par}]}{\vdash_{\mathsf{A}} \mathsf{A}[P \mid Q] : f'}\ [\text{T-SA}] \quad \dfrac{f'{\uparrow_{\{w\}}}(w) \text{ realizes } C}{\vdash_{\mathsf{A}} \{\downarrow_w C\}_{\mathsf{A}} \rhd f'}\ [\text{T-SFz1}]}{\vdash_{\mathsf{A}} \mathsf{A}[P \mid Q] \mid \{\downarrow_w C\}_{\mathsf{A}} : f'}\ [\text{T-SPar2}]$$

- Item (9.6b). The thesis holds vacuously, since, for any f, it is not possible to have a typing $\vdash_{\mathsf{A}} \mathsf{B}[\cdots] : f$ when $\mathsf{B} \neq \mathsf{A}$ (see Lemma 8.3 on page 27).
- Item (9.6c). Let $P_0 = \mathtt{tell}{\downarrow_w} C.P + P' \mid Q$. The typing $\vdash_{\mathsf{A}} S \rhd f$ must have been obtained by rule [T-SAFree1] as follows:

$$\dfrac{\mathsf{B} \neq \mathsf{A} \quad \mathrm{fv}(P_0) \cap \mathrm{dom}\, f = \emptyset}{\vdash_{\mathsf{A}} S = \mathsf{B}[P_0] \rhd f}\ [\text{T-SAFree1}]$$

Since $\mathrm{fv}(P \mid Q) \subseteq \mathrm{fv}(P_0)$, then $\mathrm{fv}(P \mid Q) \cap \mathrm{dom}\, f = \emptyset$. Further, since $w \in \mathrm{fv}(P_0)$ and $\mathrm{fv}(P_0) \cap \mathrm{dom}\, f = \emptyset$, then $f(w) = \bot$. Then, we can construct the following typing derivation for S':

$$\dfrac{\dfrac{\mathsf{B} \neq \mathsf{A} \quad \mathrm{fv}(P \mid Q) \cap \mathrm{dom}\, f = \emptyset}{\vdash_{\mathsf{A}} \mathsf{B}[P \mid Q] \rhd f}\ [\text{T-SAFree1}] \quad \dfrac{\mathsf{B} \neq \mathsf{A} \quad f(w) = \bot}{\vdash_{\mathsf{A}} \{\downarrow_w C\}_{\mathsf{B}} \rhd f}\ [\text{T-SAFree2}]}{\vdash_{\mathsf{A}} S' \rhd f}\ [\text{T-SPar1}]$$

Rule [Fuse]. We have $\mathsf{B} = \mathsf{K} \neq \mathsf{A}$, and:

$$\dfrac{C \bowtie D \quad \gamma = \mathsf{C} : C\ [] \mid \mathsf{D} : D\ [] \quad \sigma = \{s/_{x,y}\} \quad s \notin \mathrm{fnv}(S_0)}{S \;=\; (x,y)(S_0 \mid \{\downarrow_x C\}_{\mathsf{C}} \mid \{\downarrow_y D\}_{\mathsf{D}}) \xrightarrow{\mathsf{K} : \mathtt{fuse}} (s)(S_0\sigma \mid s[\gamma]) \;=\; S'}\ [\text{Fuse}]$$

where $\mathsf{C} \neq \mathsf{D}$. We prove the following items:

- Item (9.6a) is trivial, because $\mathsf{B} = \mathsf{K} \neq \mathsf{A}$.
- Item (9.6b). The redex S can only be typed as follows (up-to associativity):

$$\dfrac{\dfrac{\dfrac{\vdash_{\mathsf{A}} S_0 : f_0 \quad \dfrac{\boxed{D_1} \quad \boxed{D_2}}{\vdash_{\mathsf{A}} \{\downarrow_x C\}_{\mathsf{C}} \mid \{\downarrow_y D\}_{\mathsf{D}} \rhd f_0}\ [\text{T-SPar1}]}{\vdash_{\mathsf{A}} S_0 \mid \{\downarrow_x C\}_{\mathsf{C}} \mid \{\downarrow_y D\}_{\mathsf{D}} : f_0}\ [\text{T-SPar2}] \quad \underset{\text{honest}}{f_0{\uparrow_{\{y\}}}(y)}}{\vdash_{\mathsf{A}} (y)(S_0 \mid \{\downarrow_x C\}_{\mathsf{C}} \mid \{\downarrow_y D\}_{\mathsf{D}}) : f_0\{y \mapsto \bot\}}\ [\text{T-SDel2}] \quad \underset{\text{honest}}{f_0\{y \mapsto \bot\}{\uparrow_{\{x\}}}(x)}}{\vdash_{\mathsf{A}} S : f_0\{y \mapsto \bot\}\{x \mapsto \bot\} = f}\ [\text{T-SDel2}]$$

where the typing derivations D_1 and D_2 for the latent contracts depend on whether $\mathsf{A} \in \{\mathsf{C}, \mathsf{D}\}$ or not. By Lemma 8.2 we have $\mathrm{dom}\, f_0 \subseteq \mathrm{fnv}(S_0) \cup \{*\}$, so from the premise $s \notin \mathrm{fnv}(S_0)$ it follows that $s \notin \mathrm{dom}\, f_0$. Since $\vdash_{\mathsf{A}} S_0 : f_0$, by Lemma 9.5 we have that $\vdash_{\mathsf{A}} S_0\sigma : f_0 \bullet \sigma$. There are the following two cases:

  - $\mathsf{A} \notin \{\mathsf{C}, \mathsf{D}\}$. Then:

$$\boxed{D_1} \;=\; \dfrac{f_0(x) = \bot}{\vdash_{\mathsf{A}} \{\downarrow_x C\}_{\mathsf{C}} \rhd f_0}\ [\text{T-SAFree2}]$$

$$\boxed{D_2} \;=\; \dfrac{f_0(y) = \bot}{\vdash_{\mathsf{A}} \{\downarrow_y D\}_{\mathsf{D}} \rhd f_0}\ [\text{T-SAFree2}]$$

Since $x, y \notin \operatorname{dom} f_0$, by Definition 9.4 we have that $f_0 \bullet \sigma = f_0$. Since $s \notin \operatorname{dom} f_0$, we have that $(f_0 \bullet \sigma)(s) = f_0(s) = \bot$. Then,

$$(f_0 \bullet \sigma)\!\uparrow_{\{s\}}(s) \;=\; f_0(*) \;=\; f(*)$$

which is honest by the assumption that f is honest. Then, we can construct the following typing derivation:

$$\dfrac{\dfrac{\vdash_{\mathsf{A}} S_0\sigma : f_0 \bullet \sigma \qquad \dfrac{s[\gamma]\ \mathsf{A}\text{-free} \quad (f_0 \bullet \sigma)(s) = \bot}{\vdash_{\mathsf{A}} s[\gamma] \rhd f_0 \bullet \sigma}\ [\text{T-SAFREE3}]}{\vdash_{\mathsf{A}} S_0\sigma \mid s[\gamma] : f_0 \bullet \sigma}\ [\text{T-SPAR2}] \qquad (f_0 \bullet \sigma)\!\uparrow_{\{s\}}(s)\ \text{honest}}{\vdash_{\mathsf{A}} S' : (f_0 \bullet \sigma)\{s \mapsto \bot\}}\ [\text{T-SDEL2}]$$

Since $x, y, s \notin \operatorname{dom} f_0$, we conclude that:

$$(f_0 \bullet \sigma)\{s \mapsto \bot\} \;=\; f_0\{s \mapsto \bot\} \;=\; f_0 \;=\; f_0\{y \mapsto \bot\}\{x \mapsto \bot\} \;=\; f$$

– $\mathsf{A} = \mathsf{D}$. Then, $\mathsf{A} \neq \mathsf{C}$, and:

$$\boxed{D_1} \;=\; \dfrac{f_0(x) = \bot}{\vdash_{\mathsf{A}} \{\downarrow_x C\}_{\mathsf{C}} \rhd f_0}\ [\text{T-SAFREE2}]$$

$$\boxed{D_2} \;=\; \dfrac{f_0\!\uparrow_{\{y\}}(y)\ \text{realizes}\ D}{\vdash_{\mathsf{A}} \{\downarrow_y D\}_{\mathsf{D}} \rhd f_0}\ [\text{T-SFZ1}]$$

Therefore, $x \notin \operatorname{dom} f_0$, and we have two cases, according to whether $y \in \operatorname{dom} f_0$ or not.

If $y \notin \operatorname{dom} f_0$, then $f_0 \bullet \sigma = f_0$. Since $f_0\!\uparrow_{\{y\}}(y) = f_0(*)$ realizes D, and since $D = \alpha_{\mathsf{A}}(\gamma)$ by Definition 5.1, we have that $f_0\!\uparrow_{\{s\}}(s) = f_0(*)$ realizes $\alpha_{\mathsf{A}}(\gamma)$, and it is honest since f is honest. Then, we can construct the following typing derivation:

$$\dfrac{\dfrac{\vdash_{\mathsf{A}} S_0\sigma : f_0 \qquad \dfrac{f_0\!\uparrow_{\{s\}}(s)\ \text{realizes}\ \alpha_{\mathsf{A}}(\gamma)}{\vdash_{\mathsf{A}} s[\gamma] \rhd f_0}\ [\text{T-SFUSE}]}{\vdash_{\mathsf{A}} S_0\sigma \mid s[\gamma] : f_0}\ [\text{T-SPAR2}] \qquad f_0\!\uparrow_{\{s\}}(s)\ \text{honest}}{\vdash_{\mathsf{A}} S' : f_0\{s \mapsto \bot\}}\ [\text{T-SDEL2}]$$

Since $x, y, s \notin \operatorname{dom} f_0$, we conclude that:

$$f_0\{s \mapsto \bot\} \;=\; f_0\{y \mapsto \bot\}\{x \mapsto \bot\} \;=\; f$$

Otherwise, if $y \in \operatorname{dom} f_0$, the premise of [T-SFZ1] implies that $f_0(y)$ realizes D. By Definition 9.4 we have that

$$f_0 \bullet \sigma \;=\; f_0\{y \mapsto \bot\}\{s \mapsto f_0(y)\}$$

from which we can compute:

$$(f_0 \bullet \sigma)\!\uparrow_{\{s\}}(s) \;=\; (f_0 \bullet \sigma)(s) \;=\; f_0(y)$$

Since $f_0(y)$ realizes D, and since $D = \alpha_{\mathsf{A}}(\gamma)$ by Definition 5.1, we have that $(f_0 \bullet \sigma)\!\uparrow_{\{s\}}(s)$ realizes $\alpha_{\mathsf{A}}(\gamma)$. Furthermore, since $y \in \operatorname{dom} f_0$, then:

$$(f_0 \bullet \sigma)\!\uparrow_{\{s\}}(s) \;=\; f_0(y) \;=\; f_0\!\uparrow_{\{y\}}(y)$$

which is honest by the premise of [T-SDel2] in the typing derivation of S. Then, we can construct the following typing derivation:

$$\dfrac{\dfrac{\vdash_{\mathsf{A}} S_0\sigma\colon f_0 \bullet \sigma \qquad \dfrac{(f_0 \bullet \sigma)\!\uparrow_{\{s\}}(s)\ \text{realizes}\ \alpha_{\mathsf{A}}(\gamma)}{\vdash_{\mathsf{A}} s[\gamma] \rhd f_0 \bullet \sigma}\ [\text{T-SFuse}]}{\vdash_{\mathsf{A}} S_0\sigma \mid s[\gamma]\colon f_0 \bullet \sigma}\ [\text{T-SPar2}] \qquad (f_0 \bullet \sigma)\!\uparrow_{\{s\}}(s)\ \text{honest}}{\vdash_{\mathsf{A}} S'\colon (f_0 \bullet \sigma)\{s \mapsto \bot\}}\ [\text{T-SDel2}]$$

Since $x, s \notin \operatorname{dom} f_0$, we conclude that:

$$(f_0 \bullet \sigma)\{s \mapsto \bot\} \;=\; f_0\{y \mapsto \bot\}\{s \mapsto \bot\} \;=\; f_0\{y \mapsto \bot\}\{x \mapsto \bot\} \;=\; f$$

- Item (9.6c) We have the following typing derivation for S:

$$\dfrac{\dfrac{\vdash_{\mathsf{A}} S_0 \rhd f_0 \qquad \dfrac{\boxed{D_1} \quad \boxed{D_2}}{\vdash_{\mathsf{A}} \{\downarrow_x C\}_{\mathsf{C}} \mid \{\downarrow_y D\}_{\mathsf{D}} \rhd f_0}\ [\text{T-SPar1}]}{\vdash_{\mathsf{A}} S_0 \mid \{\downarrow_x C\}_{\mathsf{C}} \mid \{\downarrow_y D\}_{\mathsf{D}} \rhd f\{y \mapsto \bot\}\{x \mapsto \bot\} = f_0}\ [\text{T-SPar1}]}{\vdash_{\mathsf{A}} (y)(S_0 \mid \{\downarrow_x C\}_{\mathsf{C}} \mid \{\downarrow_y D\}_{\mathsf{D}}) \rhd f\{x \mapsto \bot\}}\ [\text{T-SDel1}]}{\vdash_{\mathsf{A}} S \rhd f}\ [\text{T-SDel1}]$$

where the typing derivations D_1 and D_2 for the latent contracts depend on whether $\mathsf{A} \in \{\mathsf{C}, \mathsf{D}\}$ or not. There are the following two cases:

  - $\mathsf{A} \notin \{\mathsf{C}, \mathsf{D}\}$. Then:

$$\boxed{D_1} \;=\; \dfrac{f_0(x) = \bot}{\vdash_{\mathsf{A}} \{\downarrow_x C\}_{\mathsf{C}} \rhd f_0}\ [\text{T-SAFree2}]$$

$$\boxed{D_2} \;=\; \dfrac{f_0(y) = \bot}{\vdash_{\mathsf{A}} \{\downarrow_y D\}_{\mathsf{D}} \rhd f_0}\ [\text{T-SAFree2}]$$

Since $x, y \notin \operatorname{dom} f_0$, by Definition 9.4 we have that: $f_0 \bullet \sigma = f_0$. Since $\vdash_{\mathsf{A}} S_0 \rhd f_0$ and $s \notin \operatorname{fnv}(S_0)$, then by Lemma B.2 it follows that:

$$\vdash_{\mathsf{A}} S_0 \rhd f_0\{s \mapsto \bot\}$$

and so by Lemma 9.5, and since $x, y \notin \operatorname{dom} f_0$ we have that:

$$\vdash_{\mathsf{A}} S_0\sigma \rhd (f_0\{s \mapsto \bot\}) \bullet \sigma \;=\; f_0\{s \mapsto \bot\}$$

Therefore, we can construct the following typing derivation for S':

$$\dfrac{\dfrac{\vdash_{\mathsf{A}} S_0\sigma \rhd f_0\{s \mapsto \bot\} \qquad \dfrac{s[\gamma]\ \mathsf{A}\text{-free} \quad f_0\{s \mapsto \bot\}(s) = \bot}{\vdash_{\mathsf{A}} s[\gamma] \rhd f_0\{s \mapsto \bot\}}\ [\text{T-SAFree3}]}{\vdash_{\mathsf{A}} S_0\sigma \mid s[\gamma] \rhd f_0\{s \mapsto \bot\}}\ [\text{T-SPar1}]}{\vdash_{\mathsf{A}} S' \rhd f_0}\ [\text{T-SDel1}]$$

Since $S' = (s)(S_0\sigma \mid s[\gamma])$ and $\operatorname{dom}\sigma = \{x, y\}$, then $x, y \notin \operatorname{fnv}(S')$. Then, from the typing $\vdash_{\mathsf{A}} S' \rhd f_0$ by Lemma B.2 we obtain the typing:

$$\vdash_{\mathsf{A}} S' \rhd f_0\{y \mapsto f(y)\}\{x \mapsto f(x)\} \;=\; f$$

– $\mathsf{A} = \mathsf{D}$. Then, $\mathsf{A} \neq \mathsf{C}$, and:

$$\boxed{D_1} \;=\; \frac{f_0(x) = \bot}{\vdash_{\mathsf{A}} \{\downarrow_x C\}_{\mathsf{C}} \rhd f_0}\ [\text{T-SAFREE2}]$$

$$\boxed{D_2} \;=\; \frac{f_0{\uparrow_{\{y\}}}(y) \text{ realizes } D}{\vdash_{\mathsf{A}} \{\downarrow_y D\}_{\mathsf{D}} \rhd f_0}\ [\text{T-SFZ1}]$$

By the typing derivation of S, we have that $x, y \notin \operatorname{dom} f_0$, hence by Definition 9.4 we have that $f_0 \bullet \sigma = f_0$. Since $\vdash_{\mathsf{A}} S_0 \rhd f_0$ and $s \notin \operatorname{fnv}(S_0)$, then by Lemma B.2 it follows that:

$$\vdash_{\mathsf{A}} S_0 \rhd f_0\{s \mapsto \bot\}$$

and so by Lemma 9.5, using the fact that $x, y \notin \operatorname{dom} f_0$, we have that:

$$\vdash_{\mathsf{A}} S_0\sigma \rhd (f_0\{s \mapsto \bot\}) \bullet \sigma \;=\; f_0\{s \mapsto \bot\}$$

Since $y \notin \operatorname{dom} f_0$, the premise of [T-SFZ1] implies that $f_0(*)$ realizes D. Then, also $(f_0\{s \mapsto \bot\}){\uparrow_{\{s\}}}(s) = f_0(*)$ realizes D. By Definition 5.1 we have $D = \alpha_{\mathsf{A}}(\gamma)$, so $(f_0\{s \mapsto \bot\}){\uparrow_{\{s\}}}(s)$ realizes $\alpha_{\mathsf{A}}(\gamma)$. Therefore, we can construct the following typing derivation for S':

$$\dfrac{\dfrac{\vdash_{\mathsf{A}} S_0\sigma \rhd f_0\{s \mapsto \bot\} \qquad \dfrac{(f_0\{s \mapsto \bot\}){\uparrow_{\{s\}}}(s) \text{ realizes } \alpha_{\mathsf{A}}(\gamma)}{\vdash_{\mathsf{A}} s[\gamma] \rhd f_0\{s \mapsto \bot\}}\ [\text{T-SFUSE}]}{\vdash_{\mathsf{A}} S_0\sigma \mid s[\gamma] \rhd f_0\{s \mapsto \bot\}}\ [\text{T-SPAR1}]}{\vdash_{\mathsf{A}} S' \rhd f_0}\ [\text{T-SDEL1}]$$

Since $S' = (s)(S_0\sigma \mid s[\gamma])$ and $\operatorname{dom}\sigma = \{x, y\}$, then $x, y \notin \operatorname{fnv}(S')$. Then, from the typing $\vdash_{\mathsf{A}} S' \rhd f_0$ by Lemma B.2 we obtain the typing:

$$\vdash_{\mathsf{A}} S' \rhd f_0\{y \mapsto f(y)\}\{x \mapsto f(x)\} \;=\; f$$

Rule [Do]. We have:

$$\frac{\gamma \xrightarrow{\mathsf{B}:a} \!\!\!\!\rightarrow\, \gamma'}{S \;=\; \mathsf{B}[\mathtt{do}_s\, a.P + P' \mid Q] \mid s[\gamma] \;\xrightarrow{\mathsf{B}\,:\,\mathtt{do}_s\, a}\; \mathsf{B}[P \mid Q] \mid s[\gamma'] \;=\; S'}\ [\text{DO}]$$

- Item (9.6a). We have $\mathsf{B} = \mathsf{A}$, and the typing derivation for S must be of the following form, where $\gamma = \mathsf{A} : C\ [\beta_{\mathsf{A}}] \mid \mathsf{C} : D\ [\beta_{\mathsf{C}}]$:

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash \mathtt{do}_s\, a.P + P' \mid Q : f}\ [\text{T-PAR}]}{\vdash_{\mathsf{A}} \mathsf{A}[\mathtt{do}_s\, a.P + P' \mid Q] : f}\ [\text{T-SA}] \qquad \dfrac{f{\uparrow_{\{s\}}}(s) \text{ realizes } \alpha_{\mathsf{A}}(\gamma)}{\vdash_{\mathsf{A}} s[\gamma] \rhd f}\ [\text{T-SFUSE}]}{\vdash_{\mathsf{A}} S : f}\ [\text{T-SPAR2}]$$

Let $A = \operatorname{fnv}(S) \cup \{*\} = \operatorname{dom} f$, as implied by Lemma 8.2. Hence:

$$f \;=\; \lambda u\,.\,[\mathtt{do}_s\, a]_u\,.\,f^P{\uparrow_A}(u) + f^{P'}{\uparrow_A}(u) \mid f^Q{\uparrow_A}(u)$$

where f^P, $f^{P'}$ and f^Q are, respectively, the types of P, P' and Q. Let $A' = \operatorname{fnv}(P) \cup \operatorname{fnv}(Q)$. Hence, if we choose:

$$f' \;=\; \lambda u\,.\,f^P{\uparrow_{A'}}(u) \mid f^Q{\uparrow_{A'}}(u)$$

we have a typing judgement $\vdash P \mid Q : f'$. Since $A = \operatorname{dom} f$ and $s \in \mathrm{fnv}(S)$, then $s \in \operatorname{dom} f$. Hence, $f(s)$ realizes $\alpha_{\mathsf{A}}(\gamma)$. By using the similar arguments to those used in case [Tau], we can deduce that $f \xrightarrow{\mathtt{do}_s\, a}_\sharp f'$. Then,

$$f{\uparrow_{\{s\}}}(s) \;=\; f(s) \xrightarrow{[\mathtt{do}_s\, a]_s}_\sharp f'{\uparrow_{\{s\}}}(s)$$

Note that $[\mathtt{do}_s\, a]_s = a$. Since $\gamma \xrightarrow{\mathsf{A}:a}\!\!\!\!\rightarrow \gamma'$ then by Lemma 5.2 it follows that $\alpha_{\mathsf{A}}(\gamma) \xrightarrow{a}\!\!\!\!\rightarrow_\sharp \alpha_{\mathsf{A}}(\gamma')$. Then, by rule [A-Do] in Figure 5 and by Definition 6.1 we deduce that $f'{\uparrow_{\{s\}}}(s)$ realizes $\alpha_{\mathsf{A}}(\gamma')$. Thus, we have the following typing derivation:

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash P \mid Q : f'}\,[\text{T-Par}]}{\vdash_{\mathsf{A}} \mathsf{A}[P \mid Q] : f'}\,[\text{T-SA}] \qquad \dfrac{f'{\uparrow_{\{s\}}}(s) \text{ realizes } \alpha_{\mathsf{A}}(\gamma')}{\vdash_{\mathsf{A}} s[\gamma'] \rhd f'}\,[\text{T-SFuse}]}{\vdash_{\mathsf{A}} S' : f'}\,[\text{T-SPar2}]$$

- Item (9.6b) is vacuous, because S is not typeable with ": ".
- Item (9.6c). Let $P_0 = \mathtt{do}_s\, a.P + P' \mid Q$. There are two possible typing derivations for $\vdash_{\mathsf{A}} S \rhd f$, according to whether A occurs in γ or not.

  If A does *not* occur in γ, then we have the following typing derivation:

$$\dfrac{\dfrac{\mathsf{B} \neq \mathsf{A} \quad \mathrm{fv}(P_0) \cap \operatorname{dom} f = \emptyset}{\vdash_{\mathsf{A}} \mathsf{B}[P_0] \rhd f}\,[\text{T-SAFree1}] \qquad \dfrac{s[\gamma]\ \mathsf{A}\text{-free} \quad f(s) = \bot}{\vdash_{\mathsf{A}} s[\gamma] \rhd f}\,[\text{T-SAFree3}]}{\vdash_{\mathsf{A}} \mathsf{B}[P_0] \mid s[\gamma] \rhd f}\,[\text{T-SPar1}]$$

  The thesis follows by a similar typing derivation, where rule [T-SAFree1] can be used because γ' is still A-free and $\mathrm{fv}(P \mid Q) \subseteq \mathrm{fv}(P_0)$.

  Otherwise, if A occurs in γ, then we have the following typing derivation:

$$\dfrac{\dfrac{\mathsf{B} \neq \mathsf{A} \quad \mathrm{fv}(P_0) \cap \operatorname{dom} f = \emptyset}{\vdash_{\mathsf{A}} \mathsf{B}[P_0] \rhd f}\,[\text{T-SAFree1}] \qquad \dfrac{f{\uparrow_{\{s\}}}(s) \text{ realizes } \alpha_{\mathsf{A}}(\gamma)}{\vdash_{\mathsf{A}} s[\gamma] \rhd f}\,[\text{T-SFuse}]}{\vdash_{\mathsf{A}} \mathsf{B}[P_0] \mid s[\gamma] \rhd f}\,[\text{T-SPar1}]$$

  Since $\gamma \xrightarrow{\mathsf{B}:a}\!\!\!\!\rightarrow \gamma'$ then by Lemma 5.2 it follows that $\alpha_{\mathsf{A}}(\gamma) \xrightarrow{\mathtt{ctx}:a}\!\!\!\!\rightarrow_\sharp \alpha_{\mathsf{A}}(\gamma')$. Then, by rule [A-Ctx] in Figure 5 and by Definition 6.1 we deduce that $f{\uparrow_{\{s\}}}(s)$ realizes $\alpha_{\mathsf{A}}(\gamma')$. Thus, we have the following typing derivation:

$$\dfrac{\dfrac{\mathsf{B} \neq \mathsf{A} \quad \mathrm{fv}(P \mid Q) \cap \operatorname{dom} f = \emptyset}{\vdash_{\mathsf{A}} \mathsf{B}[P \mid Q] \rhd f}\,[\text{T-SAFree1}] \qquad \dfrac{f{\uparrow_{\{s\}}}(s) \text{ realizes } \alpha_{\mathsf{A}}(\gamma')}{\vdash_{\mathsf{A}} s[\gamma'] \rhd f}\,[\text{T-SFuse}]}{\vdash_{\mathsf{A}} S' \rhd f}\,[\text{T-SPar1}]$$

Rule [Del]. We have $\pi = \mathrm{del}_u(\pi')$, and:

$$\dfrac{S_0 \xrightarrow{\mathsf{B}\,:\,\pi'} S_0'}{S \;=\; (u)S_0 \xrightarrow{\mathsf{B}\,:\,\mathrm{del}_u(\pi')} (u)S_0' \;=\; S'}\,[\text{Del}]$$

- Item (9.6a). The only possible typing derivation for S is the following:

$$\dfrac{\vdash_{\mathsf{A}} S_0 : f_0 \quad f_0{\uparrow_{\{u\}}}(u) \text{ honest}}{\vdash_{\mathsf{A}} S = (u)S_0 : f_0\{u \mapsto \bot\} = f}\,[\text{T-SDel2}]$$

We have that f_0 is honest. Indeed, for all $v \in \operatorname{dom} f_0$ we have that: if $v \neq u$, then $f_0(v) = f(v)$, which is honest by the hypothesis that f is honest; instead, if $v = u$ then $f_0{\uparrow_{\{u\}}}(u) = f_0(u)$, which is honest by the rightmost premise of [T-SDEL2]. Then, by the induction hypothesis we have that there exists some f'_0 such that $f_0 \xrightarrow{\pi'}_\sharp f'_0$ and $\vdash_{\mathsf{A}} S'_0 \colon f'_0$. Since f_0 is honest, then by Lemma 6.4 also its reduct f'_0 is honest. By Lemma B.4, also $f'_0{\uparrow_{\{u\}}}$ is honest. Then, we have the following typing derivation:

$$\frac{\vdash_{\mathsf{A}} S'_0 \colon f'_0 \quad f'_0{\uparrow_{\{u\}}}(u) \text{ honest}}{\vdash_{\mathsf{A}} S' = (u)S'_0 \colon f'_0\{u \mapsto \bot\}} \text{ [T-SDEL2]}$$

Let $f' = f'_0\{u \mapsto \bot\}$. Since $f_0 \xrightarrow{\pi'}_\sharp f'_0$, then by Definition 9.1 we have that $\operatorname{dom} f'_0 \subseteq \operatorname{dom} f_0$, and:

$$\forall v \in \operatorname{dom} f'_0 \quad : \quad f_0(v) \xrightarrow{[\pi']_v}_\sharp f'_0(v) \tag{1}$$

$$\forall v \in \operatorname{dom} f_0 \setminus \operatorname{dom} f'_0 \quad : \quad f_0(v) \xrightarrow{[\pi']_v}_\sharp f'_0(*) \tag{2}$$

We have that:

$$\operatorname{dom} f' \;=\; \operatorname{dom} f'_0 \setminus \{u\} \;\subseteq\; \operatorname{dom} f_0 \setminus \{u\} \;=\; \operatorname{dom} f$$

To prove that $f \xrightarrow{\pi}_\sharp f'$, we need to consider the following two cases.

- $v \in \operatorname{dom} f'$. Then, $v \in \operatorname{dom} f'_0$, so from (1) we have $f_0(v) \xrightarrow{[\pi']_v}_\sharp f'_0(v)$. Since $u \neq v$, then $f(v) \xrightarrow{[\pi']_v}_\sharp f'(v)$. There are three further subcases:
  - $\pi' = \mathtt{do}_u\, a$. In this case we have $[\pi']_v = \tau_? = \operatorname{del}_u(\pi') = \pi = [\pi]_v$.
  - $\pi' = \mathtt{tell}{\downarrow_u}\, C$. In this case we have $[\pi']_v = \tau = \operatorname{del}_u(\pi') = \pi = [\pi]_v$.
  - otherwise, $\pi' = \operatorname{del}_u(\pi') = \pi$, hence $[\pi']_v = [\pi]_v$.
- $v \in \operatorname{dom} f \setminus \operatorname{dom} f'$, then $v \in \operatorname{dom} f_0 \setminus \operatorname{dom} f'_0$, so from (2) we have $f_0(v) \xrightarrow{[\pi']_v}_\sharp f'_0(*)$. Since $u \neq v$, we conclude that $f(v) \xrightarrow{[\pi']_v}_\sharp f'(*)$. The thesis follows because $[\pi']_v = [\pi]_v$, as in the previous case.

- Item (9.6b). The only possible typing derivation for S is the following:

$$\frac{\vdash_{\mathsf{A}} S_0 \colon f_0 \quad f_0{\uparrow_{\{u\}}}(u) \text{ honest}}{\vdash_{\mathsf{A}} S = (u)S_0 \colon f_0\{u \mapsto \bot\} = f} \text{ [T-SDEL2]}$$

We have that f_0 is honest. Indeed, for all $v \in \operatorname{dom} f_0$ we have that: if $v \neq u$, then $f_0(v) = f(v)$, which is honest by the hypothesis that f is honest; instead, if $v = u$ then $f_0(v) = f_0(u) = f_0{\uparrow_{\{u\}}}(u)$, which is honest by the rightmost premise of [T-SDEL2]. Hence, by the induction hypothesis we have that $\vdash_{\mathsf{A}} S'_0 \colon f_0$. Since f_0 is honest, then by Lemma B.4 it follows that $f_0{\uparrow_{\{u\}}}$ is honest. Then, we have the following typing derivation:

$$\frac{\vdash_{\mathsf{A}} S'_0 \colon f_0 \quad f_0{\uparrow_{\{u\}}}(u) \text{ honest}}{\vdash_{\mathsf{A}} S' = (u)S'_0 \colon f_0\{u \mapsto \bot\}} \text{ [T-SDEL2]}$$

The thesis follows by choosing $f'_0\{u \mapsto \bot\} = f$.

- Item (9.6c). The only possible typing derivation for S is the following:

$$\frac{\vdash_{\mathsf{A}} S_0 \rhd f\{u \mapsto \bot\}}{\vdash_{\mathsf{A}} S = (u)S_0 \rhd f} \text{ [T-SDEL1]}$$

Since $f\{u \mapsto \bot\}$ is honest and $\vdash_{\mathsf{A}} S_0 \rhd f\{u \mapsto \bot\}$, then by the induction hypothesis it follows that $\vdash_{\mathsf{A}} S_0' \rhd f\{u \mapsto \bot\}$. Then, we can reconstruct the following derivation:

$$\frac{\vdash_{\mathsf{A}} S_0' \rhd f\{u \mapsto \bot\}}{\vdash_{\mathsf{A}} S' = (u)S_0' \rhd f} \text{ [T-SDel1]}$$

Rule [Par]. We have:

$$\frac{S_0 \xrightarrow{\mathsf{B}:\pi} S_0'}{S \;=\; S_0 \mid S_1 \xrightarrow{\mathsf{B}:\pi} S_0' \mid S_1 \;=\; S'} \text{ [Par]}$$

- Item (9.6a). We have $\mathsf{B} = \mathsf{A}$, and since S_0 reduces through A, then a process of A cannot appear in S_1. Hence, the only typing derivation for S is the following:

$$\frac{\vdash_{\mathsf{A}} S_0 \colon f \quad \vdash_{\mathsf{A}} S_1 \rhd f}{\vdash_{\mathsf{A}} S \colon f} \text{ [T-SPar2]}$$

  By applying the induction hypothesis on the leftmost premise of [T-SPar2], we have some f' such that:

$$f \xrightarrow{\pi}_\sharp f' \qquad \text{and} \qquad \vdash_{\mathsf{A}} S_0' \colon f'$$

  Note that if $\pi = \mathtt{do}_s\, a$, then by rule [Do] it must be $s[\gamma] \in S_0$ (and so, $s[\gamma] \notin S_1$). Hence, by Lemma B.5 we infer that $\vdash_{\mathsf{A}} S_1 \rhd f'$. Then, we can construct the following typing derivation for S':

$$\frac{\vdash_{\mathsf{A}} S_0' \colon f' \quad \vdash_{\mathsf{A}} S_1 \rhd f'}{\vdash_{\mathsf{A}} S' \colon f'} \text{ [T-SPar2]}$$

- Item (9.6b). We have $\mathsf{B} \neq \mathsf{A}$. There are two possible typing derivations for S:

$$\frac{\vdash_{\mathsf{A}} S_0 \colon f \quad \vdash_{\mathsf{A}} S_1 \rhd f}{\vdash_{\mathsf{A}} S \colon f} \text{ [T-SPar2]} \qquad \frac{\vdash_{\mathsf{A}} S_0 \rhd f \quad \vdash_{\mathsf{A}} S_1 \colon f}{\vdash_{\mathsf{A}} S \colon f} \text{ [T-SPar2]}$$

  If the leftmost typing derivation has been used, by applying the induction hypothesis on its leftmost premise we obtain $\vdash_{\mathsf{A}} S_0' \colon f$. So, we can construct the following typing derivation for S':

$$\frac{\vdash_{\mathsf{A}} S_0' \colon f \quad \vdash_{\mathsf{A}} S_1 \rhd f}{\vdash_{\mathsf{A}} S' \colon f} \text{ [T-SPar2]}$$

  If the rightmost typing derivation has been used, by applying the induction hypothesis of item (9.6c) on its leftmost premise we obtain $\vdash_{\mathsf{A}} S_0' \rhd f$. So, we can construct the following typing derivation for S':

$$\frac{\vdash_{\mathsf{A}} S_0' \rhd f \quad \vdash_{\mathsf{A}} S_1 \colon f}{\vdash_{\mathsf{A}} S' \colon f} \text{ [T-SPar2]}$$

- Item (9.6c). We have the following typing derivation for S:

$$\frac{\vdash_{\mathsf{A}} S_0 \rhd f \quad \vdash_{\mathsf{A}} S_1 \rhd f}{\vdash_{\mathsf{A}} S \rhd f} \text{ [T-SPar1]}$$

By applying the induction hypothesis on the leftmost premise, we have that $\vdash_{\mathsf{A}} S'_0 \rhd f$. So, we can construct the following typing derivation for S':

$$\frac{\vdash_{\mathsf{A}} S'_0 \rhd f \quad \vdash_{\mathsf{A}} S_1 \rhd f}{\vdash_{\mathsf{A}} S' \rhd f}\ [\text{T-SPAR1}]$$

Rule [Rec]. We have:

$$\frac{\mathsf{B}[P\{\mathtt{rec}\ \mathrm{X}(\boldsymbol{y}).\ P/\mathrm{X}\}\{\boldsymbol{u}/\boldsymbol{y}\} \mid Q] \mid S_0 \xrightarrow{\mathsf{B}:\pi} S'}{S\ =\ \mathsf{B}[(\mathtt{rec}\ \mathrm{X}(\boldsymbol{y}).\ P)(\boldsymbol{u}) \mid Q] \mid S_0 \xrightarrow{\mathsf{B}:\pi} S'}\ [\text{REC}]$$

Let $P_0 = (\mathtt{rec}\ \mathrm{X}(\boldsymbol{y}).\ P)(\boldsymbol{u}) \mid Q$.

- Item (9.6a). We have $\mathsf{B} = \mathsf{A}$, $\boldsymbol{y} = \boldsymbol{u} = \emptyset$. The only typing derivation for S is the following, where $f = \lambda u.\, f^{\mathrm{X}}{\uparrow_A}(u) \mid f^{Q}{\uparrow_A}(u)$, $f^{\mathrm{X}} = \lambda u.\, rec\ @\mathrm{X}.f^{P}(u)$, and $A = \operatorname{dom} f^{\mathrm{X}} \cup \operatorname{dom} f^{Q} = \operatorname{dom} f^{P} \cup \operatorname{dom} f^{Q}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash P\colon f^{P}}{\vdash (\mathtt{rec}\ \mathrm{X}().\ P)()\colon f^{\mathrm{X}}}\ [\text{T-REC}] \qquad \vdash Q\colon f^{Q}}{\vdash (\mathtt{rec}\ \mathrm{X}().\ P)() \mid Q\colon f}\ [\text{T-PAR}]}{\vdash_{\mathsf{A}} \mathsf{A}[(\mathtt{rec}\ \mathrm{X}().\ P)() \mid Q]\colon f}\ [\text{T-SA}] \qquad \vdash_{\mathsf{A}} S_0 \rhd f}{\vdash_{\mathsf{A}} S\colon f}\ [\text{T-SPAR2}]$$

  Since $\vdash P\colon f^{P}$ and $\vdash (\mathtt{rec}\ \mathrm{X}().\ P)()\colon f^{\mathrm{X}}$, then by Lemma B.3 we have:

$$\vdash P\{^{(\mathtt{rec}\ \mathrm{X}().\ P)()}/_{\mathrm{X}}\}\colon \lambda u\,.\, f^{P}{\uparrow_B}(u)\{^{f^{\mathrm{X}}{\uparrow_B}(u)}/_{@\mathrm{X}}\}\ =\ g$$

  where $B = \operatorname{dom} f^{P} \cup \operatorname{dom} f^{\mathrm{X}} = \operatorname{dom} f^{P}$. Let:

$$g'\ =\ \lambda u.\, g{\uparrow_C}(u) \mid f^{Q}{\uparrow_C}(u)$$

  where $C = \operatorname{dom} g \cup \operatorname{dom} f^{Q} = \operatorname{dom} f^{P} \cup \operatorname{dom} f^{Q} = A$. Note that g' can be obtained from f, by unfolding the recursion therein: however, the unfolding does not affect the typing $\vdash_{\mathsf{A}} S_0 \rhd f$, which can then be re-typed as $\vdash_{\mathsf{A}} S_0 \rhd g'$. Therefore, we have the following typing for the premise of rule [REC]:

$$\dfrac{\dfrac{\dfrac{\vdash P\{\mathtt{rec}\ \mathrm{X}().\ P/\mathrm{X}\}\colon g \quad \vdash Q\colon f^{Q}}{\vdash P\{\mathtt{rec}\ \mathrm{X}().\ P/\mathrm{X}\} \mid Q\colon g'}\ [\text{T-PAR}]}{\vdash_{\mathsf{A}} \mathsf{A}[P\{\mathtt{rec}\ \mathrm{X}().\ P/\mathrm{X}\} \mid Q]\colon g'}\ [\text{T-SA}] \qquad \vdash_{\mathsf{A}} S_0 \rhd g'}{\vdash_{\mathsf{A}} \mathsf{A}[P\{\mathtt{rec}\ \mathrm{X}().\ P/\mathrm{X}\} \mid Q] \mid S_0\colon g'}\ [\text{T-SPAR2}]$$

  Now, since f is honest, then also its unfolding g' is honest as well. Therefore, by applying the induction hypothesis on the premise of rule [REC], we obtain some f' such that:

$$g' \xrightarrow{\pi}_{\sharp} f' \qquad\qquad \vdash_{\mathsf{A}} S'\colon f'$$

  To conclude, note that $f \xrightarrow{\pi}_{\sharp} f'$ follows by rule [C-REC] in Figure 5.
- Item (9.6b). We have $\mathsf{B} \neq \mathsf{A}$, so the only typing derivation for S is the following:

$$\dfrac{\dfrac{\mathsf{A} \neq \mathsf{B} \quad \operatorname{fv}(P_0) \cap \operatorname{dom} f = \emptyset}{\vdash_{\mathsf{A}} \mathsf{B}[P_0] \rhd f}\ [\text{T-SAFREE1}] \qquad \vdash_{\mathsf{A}} S_0\colon f}{\vdash_{\mathsf{A}} S\colon f}\ [\text{T-SPAR2}]$$

Note that the typing with rule [T-SAFREE1] is not affected by the unfolding of P_0, because the unfolding can only decrease the set of free variables. Hence, we can use the same rule to obtain $\vdash_{\mathsf{A}} \mathsf{B}[P'] \rhd f$, where P' is the process within $\mathsf{B}[\cdots]$ in the premise of rule [REC]. By applying [T-SPAR2], we then type as f the whole premise of rule [REC]. Then, by the induction hypothesis of item (9.6b), we obtain the thesis.

- Item (9.6c). The only typing derivation for S is the following:

$$\dfrac{\dfrac{\mathsf{A} \neq \mathsf{B} \quad \mathrm{fv}(P_0) \cap \mathrm{dom}\, f = \emptyset}{\vdash_{\mathsf{A}} \mathsf{B}[P_0] \rhd f}\ [\text{T-SAFREE1}] \qquad \vdash_{\mathsf{A}} S_0 \rhd f}{\vdash_{\mathsf{A}} S \rhd f}\ [\text{T-SPAR1}]$$

Note that the typing with rule [T-SAFREE1] is not affected by the unfolding of P_0, because the unfolding can only decrease the set of free variables. Hence, we can use the same rule to obtain $\vdash_{\mathsf{A}} \mathsf{B}[P'] \rhd f$, where P' is the process within $\mathsf{B}[\cdots]$ in the premise of rule [REC]. By applying [T-SPAR1], we then type as f the premise of rule [REC]. By the induction hypothesis of item (9.6c), we conclude. □

D. Proof of Theorem 9.7 (Progress)

Lemma D.1 (Process progress)**.** *If* $\vdash P \colon f \,|\, g$ *with* $f \,|\, g$ *honest,* $f \xrightarrow{\pi}_\sharp f'$*, and* $u \in \mathrm{dom}\, f = \mathrm{dom}\, g$*, then:*

$$\pi \in \{\tau, \mathtt{tell}\downarrow_u C\} \implies \exists \boldsymbol{x}, P', S' \,.\ \mathsf{A}[P] \xrightarrow{\mathsf{A}\,:\,\pi} (\boldsymbol{x})(\mathsf{A}[P'] \mid S') \ \wedge \vdash_{\mathsf{A}} (\boldsymbol{x})(\mathsf{A}[P'] \mid S') \colon f' \,|\, g \tag{D.1a}$$

$$\pi = \mathtt{do}_u\, a \implies a \in \mathsf{A}[P]{\downarrow_u^{\mathsf{A}}} \tag{D.1b}$$

Proof. We prove item (D.1a) by induction on the proof of $f \xrightarrow{\pi}_\sharp f'$, by using the syntactic notation for process types introduced in Notation B.7, and by universally quantifying P and g in the inductive statement. To reduce the technical overhead, we will omit the domain expansions $f{\uparrow_A}$ throughout the proof. We have the following cases:

- [C-PREF]. Then, $f = \alpha.f'$, and $\alpha = [\pi]$. Since $\vdash P \colon f \mid g$, then P has the form $(\boldsymbol{x})(\pi'.P_0 \,|\, Q)$, and $\mathrm{del}_{\boldsymbol{x}}(\pi') = \pi$. We proceed by cases on the form of π'. Since (D.1a) assumes that π is either a τ or a $\mathtt{tell}$, we only have the following two cases:
  - $\pi' = \tau$. Let $S' = \mathbf{0}$. Then:

$$\mathsf{A}[(\boldsymbol{x})(\pi'.P_0 \mid Q)] \xrightarrow{\mathsf{A}\,:\,\tau} (\boldsymbol{x})(\mathsf{A}[P_0 \mid Q] \mid S') \;=\; \mathsf{A}[(\boldsymbol{x})(P_0 \mid Q)]$$

    Let $P' = P_0 \mid Q$. To prove that $\vdash_{\mathsf{A}} (\boldsymbol{x})(\mathsf{A}[P'] \mid S') \colon f' \,|\, g$, we first deconstruct the typing $\vdash P \colon f \mid g$ as follows:

$$\dfrac{\dfrac{\dfrac{\vdash P_0 \colon f'_0}{\vdash \pi'.P_0 \colon f_0 = \lambda u.[\tau]_u.f'_0(u) = \tau.f'_0}\ [\text{T-SUM}] \qquad \vdash Q \colon g_0}{\vdash \pi'.P_0 \mid Q \colon f_0 \mid g_0}\ [\text{T-PAR}] \qquad (f_0 \mid g_0)(\boldsymbol{x})\ \text{honest}}{\vdash (\boldsymbol{x})(\pi'.P_0 \mid Q) \colon f \mid g = (f_0 \mid g_0)\{\boldsymbol{x} \mapsto \bot\}}\ [\text{T-DEL}^*]$$

By Lemma 6.4, we have that $(f'_0 \mid g_0)(\boldsymbol{x})$ is honest. We can then reconstruct the typing for $(\boldsymbol{x})(\mathsf{A}[P'] \mid S')$:

$$\dfrac{\dfrac{\vdash P' \colon f'_0 \mid g_0 \quad (f'_0 \mid g_0)(\boldsymbol{x}) \text{ honest}}{\vdash (\boldsymbol{x})P' \colon (f'_0 \mid g_0)\{\boldsymbol{x} \mapsto \bot\}}\,[\text{T-Del}^*]}{\vdash_{\mathsf{A}} \mathsf{A}[(\boldsymbol{x})P'] \colon (f'_0 \mid g_0)\{\boldsymbol{x} \mapsto \bot\}}\,[\text{T-SA}]$$

Since $f_0 \xrightarrow{\tau}_\sharp f'_0$ and $f \mid g = f_0\{\boldsymbol{x} \mapsto \bot\} \mid g_0\{\boldsymbol{x} \mapsto \bot\}$, then we have the thesis:

$$f' \mid g \;=\; f'_0\{\boldsymbol{x} \mapsto \bot\} \mid g_0\{\boldsymbol{x} \mapsto \bot\}$$

– $\pi' = \mathtt{tell}\downarrow_w C$. Let $S' = \{\downarrow_w C\}_{\mathsf{A}}$. Then:

$$\mathsf{A}[(\boldsymbol{x})(\pi'.P_0 \mid Q)] \;\xrightarrow{\mathsf{A} \,:\, \pi}\; (\boldsymbol{x})(\mathsf{A}[P_0 \mid Q] \mid S')$$

Let $P' = P_0 \mid Q$. To prove that $\vdash_{\mathsf{A}} (\boldsymbol{x})(\mathsf{A}[P'] \mid S') \colon f' \mid g$, we first deconstruct the typing $\vdash P \colon f \mid g$ as in the previous case:

$$\dfrac{\dfrac{\dfrac{\vdash P_0 \colon f'_0}{\vdash \pi'.P_0 \colon f_0 = \lambda u.[\pi']_u.f'_0(u) = [\pi'].f'_0}\,[\text{T-Sum}] \quad \vdash Q \colon g_0}{\vdash \pi'.P_0 \mid Q \colon f_0 \mid g_0}\,[\text{T-Par}] \quad (f_0 \mid g_0)(\boldsymbol{x}) \text{ honest}}{\vdash (\boldsymbol{x})(\pi'.P_0 \mid Q) \colon f \mid g = (f_0 \mid g_0)\{\boldsymbol{x} \mapsto \bot\}}\,[\text{T-Del}^*]$$

By Lemma 6.4, we have that $(f'_0 \mid g_0)(\boldsymbol{x})$ is honest. We now prove that $(f'_0 \mid g_0)\!\uparrow_w (w)$ realizes C. There are two cases, according to whether $w \in \boldsymbol{x}$ or not. If $w \in \boldsymbol{x}$, then since $(f_0 \mid g_0)(\boldsymbol{x})$ honest, so in particular $(f_0 \mid g_0)(w)$ is honest. Hence, by $(f_0 \mid g_0)(w) \xrightarrow{\langle C \rangle}_\sharp (f'_0 \mid g_0)(w)$ we infer that $(f'_0 \mid g_0)(w)$ realizes C. If $w \notin \boldsymbol{x}$, since $f \mid g$ is honest and $(f \mid g)(w) = (f_0 \mid g_0)(w)$, then $(f_0 \mid g_0)(w)$ is honest; we infer that $(f'_0 \mid g_0)(w)$ realizes C as in the other case. We can then reconstruct the typing for $(\boldsymbol{x})(\mathsf{A}[P'] \mid S')$ as follows:

$$\dfrac{\dfrac{\dfrac{\vdash P' \colon f'_0 \mid g_0}{\vdash_{\mathsf{A}} \mathsf{A}[P'] \colon f'_0 \mid g_0}\,[\text{T-SA}] \quad \dfrac{(f'_0 \mid g_0)\!\uparrow_{\{w\}} (w) \text{ realizes } C}{\vdash_{\mathsf{A}} \{\downarrow_w C\}_{\mathsf{A}} \rhd f'_0 \mid g_0}\,[\text{T-SFz1}]}{\vdash_{\mathsf{A}} \mathsf{A}[P'] \mid S' \colon f'_0 \mid g_0}\,[\text{T-SPar2}] \quad (f'_0 \mid g_0)\!\uparrow_{\boldsymbol{x}} (\boldsymbol{x}) \text{ honest}}{\vdash_{\mathsf{A}} (\boldsymbol{x})(\mathsf{A}[P'] \mid S') \colon (f'_0 \mid g_0)\{\boldsymbol{x} \mapsto \bot\}}\,[\text{T-SDel2}^*]$$

Since $f_0 \xrightarrow{\pi'}_\sharp f'_0$, then $f_0\{\boldsymbol{x} \mapsto \bot\} \xrightarrow{\pi}_\sharp f'_0\{\boldsymbol{x} \mapsto \bot\}$. Therefore:

$$f \mid g \;=\; f_0\{\boldsymbol{x} \mapsto \bot\} \mid g_0\{\boldsymbol{x} \mapsto \bot\} \;\xrightarrow{\pi}_\sharp\; f' \mid g \;=\; f'_0\{\boldsymbol{x} \mapsto \bot\} \mid g_0\{\boldsymbol{x} \mapsto \bot\}$$

For item (D.1b) we have $\pi = \mathtt{do}_u\, a$. Since $u \in \operatorname{dom} f$, then by Lemma 8.2 we have $u \in \operatorname{fv}(P)$, and so $u \notin \boldsymbol{x}$ and $\pi' = \mathtt{do}_u\, a$. The thesis follows by Definition 4.2.

- [C-SumL]. Easy generalisation of case [C-Pref], since all sums are guarded.
- [C-ParL]. Then, $f = f_0 \mid f_1$ and $f_0 \xrightarrow{\pi}_\sharp f'_0$ for some f_0, f_1 and f'_0. Therefore, $f' = f'_0 \mid f_1$, and $u \in \operatorname{dom} f_0 = \operatorname{dom} f_1 \mid g$. Both items (D.1a) and (D.1b) follow by the induction hypothesis, instantiating the parallel component g of the inductive statement as $f_1 \mid g$.
- [C-Rec]. We have that $f = rec\ \mathcal{X}.f_0$, and:

$$\dfrac{f_0\{^{rec\ \mathcal{X}.f_0}/_{\mathcal{X}}\} \xrightarrow{\pi}_\sharp f'}{rec\ \mathcal{X}.f_0 \xrightarrow{\pi}_\sharp f'}$$

Therefore, $P = (\boldsymbol{x})(\texttt{rec X. } P_0)$, and $\vdash (\boldsymbol{x})(P_0\{^P/_{\mathrm{X}}\})\colon f_0\{^f/_{\mathrm{X}}\}$ by Lemma B.3. Since $f \mid g$ is honest, then $f_0\{^{rec\ \mathrm{X}.f_0}/_{\mathrm{X}}\} \mid g$ is honest as well, since unfolding preserves the semantics. Further, $u \in \operatorname{dom} f_0\{^{rec\ \mathrm{X}.f_0}/_{\mathrm{X}}\} = \operatorname{dom} g$. Both items (D.1a) and (D.1b) follow by the induction hypothesis, instantiating the process P of the inductive statement as $(\boldsymbol{x})(P_0\{^P/_{\mathrm{X}}\})$. □

Proof of Theorem 9.7. By Lemma 8.3, there exist $\boldsymbol{v}, S_0, P$ such that:

$$S \equiv (\boldsymbol{v})\,(\mathsf{A}[P] \mid S_0)$$

By inverting the typing derivation of $\vdash_{\mathsf{A}} S\colon f$, we must have $f = f_0\{\boldsymbol{v} \mapsto \bot\}$, for some f_0 such that $\vdash P\colon f_0$ and $f_0{\uparrow_{\boldsymbol{v}}}(\boldsymbol{v})$ is honest; Together with the fact that f is honest, we obtain that f_0 is honest. Further, $\vdash_{\mathsf{A}} S_0 \rhd f_0$. We have the following cases, according to the form of π:

- $\pi = \tau$. By Definition 7.1, the hypothesis $f \xrightarrow{\pi}_\sharp f'$ implies $f_0 \xrightarrow{\pi'}_\sharp f'_0$, where $\pi' \in \{\tau, \texttt{tell}{\downarrow_w}\, C\}$ (with $w \in \boldsymbol{v}$), and $f' = f'_0\{\boldsymbol{v} \mapsto \bot\}$. Note that if $\pi' = \texttt{tell}{\downarrow_w}\, C$ but $w \notin \operatorname{dom} f_0$, then we also have $f_0 \xrightarrow{\tau}_\sharp f'_0$, so in this case we could also choose $\pi' = \tau$. Consequently, w.l.o.g. we can assume $w \in \operatorname{dom} f_0$. By Lemma D.1 (by choosing $g = \mathbf{0}$) there exist $\boldsymbol{x}, P', S_0$ such that:

$$\mathsf{A}[P] \xrightarrow{\mathsf{A}\,:\,\pi'} (\boldsymbol{x})(\mathsf{A}[P'] \mid S_0) \;=\; S'_0 \qquad\qquad \vdash_{\mathsf{A}} S'_0\colon f'_0$$

Hence we have the following derivation:

$$\dfrac{\dfrac{\mathsf{A}[P] \xrightarrow{\mathsf{A}\,:\,\pi'} S'_0}{\mathsf{A}[P] \mid S_0 \xrightarrow{\mathsf{A}\,:\,\pi'} S'_0 \mid S_0}\ [\textsc{Par}]}{S \xrightarrow{\mathsf{A}\,:\,\mathrm{del}_{\boldsymbol{v}}(\pi')} (\boldsymbol{v})(S'_0 \mid S_0) = S'}\ [\textsc{Del}^*]$$

By definition of π' we have that $\mathrm{del}_{\boldsymbol{v}}(\pi') = \pi$. Since $\vdash_{\mathsf{A}} S_0 \rhd f_0$ and $f_0 \xrightarrow{\pi'}_\sharp f'_0$ with $\pi' \neq \texttt{do}\cdots$, then by Lemma B.5 we also have $\vdash_{\mathsf{A}} S_0 \rhd f'_0$. Let $S' = (\boldsymbol{v})(S'_0 \mid S_0)$. Then, we have the following typing derivation for S':

$$\dfrac{\dfrac{\vdash_{\mathsf{A}} S'_0\colon f'_0 \quad \vdash_{\mathsf{A}} S_0 \rhd f'_0}{\vdash_{\mathsf{A}} S'_0 \mid S_0\colon f'_0}\ [\text{T-SPar2}] \qquad f'_0{\uparrow_{\boldsymbol{v}}}(\boldsymbol{v})\ \text{honest}}{\vdash_{\mathsf{A}} S'\colon f'_0\{\boldsymbol{v} \mapsto \bot\} = f'}\ [\text{T-SDel2}^*]$$

- $\pi = \texttt{tell}{\downarrow_u}\, C$. By Definition 7.1, the hypothesis $f \xrightarrow{\pi}_\sharp f'$ implies $f_0 \xrightarrow{\pi}_\sharp f'_0$, using the assumption $u \in \operatorname{dom} f \subseteq \operatorname{dom} f_0$. The proof then proceeds similarly to the previous case, by exploiting Lemma D.1.
- $\pi = \texttt{do}_u\, a$. By Definition 7.1, the hypothesis $f \xrightarrow{\pi}_\sharp f'$ implies $f_0 \xrightarrow{\pi}_\sharp f'_0$, using the assumption $u \in \operatorname{dom} f \subseteq \operatorname{dom} f_0$. By Lemma D.1, $a \in \mathsf{A}[P]{\downarrow_u^{\mathsf{A}}}$. Since $u \notin \boldsymbol{v}$, then we have the thesis $a \in S{\downarrow_u^{\mathsf{A}}}$. □

E. Other proofs for Section 9 (Type safety)

Proof of Lemma 9.8. We prove the following stronger statement. If $\Gamma \vdash P \colon f$, for some process P and some self-concordant Γ, then f is self-concordant. We proceed by induction on this typing derivation. We have the following cases, according to the last rule applied in the derivation:

- [T-Sum]. The rule specifies a set of prefixes π_i; then, $\alpha = [\pi_i]_u$ for some of these i. Taking $\pi = \pi_i$ and $f' = f_i$ gives the thesis.
- [T-Par]. The thesis follows directly by the induction hypothesis.
- [T-Def]. We have $P = \mathrm{X}(\boldsymbol{v})$, and:

$$\frac{\mathrm{X}(\boldsymbol{u}) \triangleq P' \quad \mathrm{X}(\boldsymbol{v}) \notin \operatorname{dom}\Gamma \quad \Gamma\{\mathrm{X}(\boldsymbol{v}) \mapsto \lambda v.\, \mathcal{X}\} \vdash P'\{\boldsymbol{u} \mapsto \boldsymbol{v}\} \colon g}{\Gamma \vdash \mathrm{X}(\boldsymbol{v}) \colon \lambda v.\, \mathit{rec}\ \mathcal{X}.g(v)}\ \text{[T-Def]}$$

  with $f = \lambda v.\, \mathit{rec}\ \mathcal{X}.g(v)$. Since $\mathit{rec}\ \mathcal{X}.g(u) \xrightarrow{\alpha}_\sharp \mathcal{P}'$, then by inverting rule [C-Rec] it must be $g(u)\{\mathcal{X} \mapsto \mathit{rec}\ \mathcal{X}.g(v)\} \xrightarrow{\alpha}_\sharp \mathcal{P}'$. From this, we can prove that $g(u) \xrightarrow{\alpha}_\sharp \mathcal{P}''$, for some $\mathcal{P}''$ such that $\mathcal{P}' = \mathcal{P}''\{\mathcal{X} \mapsto \mathit{rec}\ \mathcal{X}.g(u)\}$. Note that $\Gamma\{\mathrm{X}(\boldsymbol{v}) \mapsto \lambda v.\, \mathcal{X}\}$ is self-concordant. By the induction hypothesis, there exist π and g' such that $[\pi]_u = \alpha$, $g'(u) = \mathcal{P}''$, and $g \xrightarrow{\pi}_\sharp g'$. The thesis follows by taking $f' = \lambda v.g'(v)\{\mathcal{X} \mapsto \mathit{rec}\ \mathcal{X}.g(v)\}$.
- [T-Var]. Trivial, since Γ is self-concordant.
- [T-Del]. We have $P = (v)P'$, and:

$$\frac{\Gamma_{\neq v} \vdash P' \colon g \quad g(v)\ \text{honest}}{\Gamma \vdash (v)P' \colon g\{v \mapsto g(*)\}}\ \text{[T-Del]}$$

  where $f = g\{v \mapsto g(*)\}$. There are the following two subcases:
  - $v \neq u$. By the induction hypothesis, there exist π, g' such that $[\pi]_u = \alpha$, $g'(u) = \mathcal{P}'$, and $g \xrightarrow{\pi}_\sharp g'$. The thesis follows by taking $f' = g'\{v \mapsto g'(*)\}$.
  - $v = u$. Since $f(u) \xrightarrow{\alpha}_\sharp \mathcal{P}'$, then $g(*) \xrightarrow{\alpha}_\sharp \mathcal{P}'$. By the induction hypothesis (applied on $*$), there exist π, g' such that $[\pi]_* = \alpha$, $g'(*) = \mathcal{P}'$, and $g \xrightarrow{\pi}_\sharp g'$. The thesis follows by taking $f' = g'\{v \mapsto g'(*)\}$.